Electronic Communications of the EASST
Volume 11 (2008)

Proceedings of the First International DisCoTec Workshop on Context-aware Adaptation Mechanisms for Pervasive and Ubiquitous Services (CAMPUS 2008)

Towards Self-evolving Context-aware Services

M. Autili, P. Di benedetto, P. Inverardi and D. A. Tamburri

12 pages

Guest Editors: Romain Rouvoy, Mauro Caporuscio, Michael Wagner
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Towards Self-evolving Context-aware Services

M. Autili[1], P. Di benedetto[2], P. Inverardi[3] and D. A. Tamburri[4]

[1] marco.autili@di.univaq.it, [2] paolo.dibenedetto@di.univaq.it,
[3] inverard@di.univaq.it, [4] damien.tamburri@di.univaq.it
Università dell'Aquila - Dipartimento di Informatica, Italy

Abstract:

The introduction of new communication infrastructures such as Beyond 3^{rd} Generation (B3G) and the widespread usage of small computing devices are rapidly changing the way we use and interact with technology to perform everyday tasks. Ubiquitous networking empowered by B3G networking makes it possible for mobile users to access networked software services across continuously changing heterogeneous infrastructures by resource-constrained devices. Heterogeneity and devices' limitedness, create serious problems for the development and dynamic deployment of mobile applications that are able to run properly on the execution context and consume services matching with the users' expectations. Furthermore, the ever-changing B3G environment calls for applications that self-evolve according to context changes. Out of these problems, self-evolving adaptable applications are increasingly emerging in the software community. In this paper we describe how CHAMELEON, a declarative framework for tailoring adaptable applications, is being used for tackling adaptation and self-evolution within the IST PLASTIC project.

Keywords: Context-awareness, Self-evolution, Adaptation mechanisms

1 Introduction

The widespread usage of small computing devices and the introduction of new communication infrastructures are rapidly changing the ways we use and interact with technology to perform everyday tasks. Today's networking infrastructures are typically made of mobile resource-constrained devices, characterized by their *heterogeneity* and *limitedness*: e.g., ubiquitous networking, empowered by B3G networks, makes it possible for mobile users to access networked software services across heterogeneous infrastructures through resource-constrained devices. Mobility, inducing changes to the execution and network environments (and therefore changes to the availability of resources), demands for self-evolving applications capable of coping with resource scarcity and with the inherently faulty and heterogeneous nature of such environments.

In this paper we describe how CHAMELEON, a declarative framework for tailoring adaptable applications for resource-constrained devices, is being used for tackling a form of self-evolution and adaptation within the IST PLASTIC project whose goal is the agile development/deployment of context-aware/self-adapting services for B3G networks. PLASTIC introduces the notion of *requested Service Level Specification* (SLS) and *offered SLS* to deal with the (extra-functional) preferences that will be used to establish the *Service Level Agreement* (SLA) between the service consumer and the service provider.

For the heterogeneous and ever-changing B3G environment, we propose a CHAMELEON-based PLASTIC solution to address self-evolution of J2ME MIDlet applications by presenting:
• A programming model that allows for easily specifying, in a flexible and declarative way, different adaptation alternatives for Java applications. The model is based on an *agile* and user-friendly extension of the Java language;
• A mechanism that, based on the Over-The-Air (OTA) provisioning technique [OTA], allows for dynamic deployment, un-deployment, and re-deployment of different alternatives of mobile (client) applications able to (*i*) run properly on the execution context and (*ii*) consume services matching with the users' expectations. The mechanism will require the ability to reason, on applications and environments alike, in terms of the resources they need and offer, respectively (i.e., resource supply and resource demand), as well as the ability to suitably adapt the application to the environment that will host it while meeting the requested SLS;
• A mechanism that, providing J2ME MIDlet applications with ad-hoc methods for *saving* and *restoring* the (current) application state, enables applications' evolution (against monitored context changes and according to adaptation policies) by dynamically un-deploying the no longer apt application alternative and subsequently (re-)deploying a new one with the desired aptitude.

CHAMELEON, whose foundations are presented in [IMN04, MI07], is based on a static analysis approach to the inspection of Java programs and their characterization w.r.t. their resource consumption in a given execution environment. Explicitly targeting resource-constrained devices, we implemented and instantiated all the needed machinery on the Java platform due to its widespread availability on today's mobile devices (e.g., smart phones, PDAs, etc.). The approach uses a light extension of the Java language, a resource model, and an SLS model that constitute the basis for a declarative technique that supports the development, deployment, and evolution of adaptable applications. By leveraging this approach we are able to perform a quantitative resource-oriented analysis of Java applications, further accounting for the user preferences specified through SLS model. In the context of such applications, our framework allows for evolution by uniting un-deployment to dynamic re-deployment of appropriate adaptation alternatives.

The paper is structured as follows: Section 2 sets the "context" and Section 3 introduces the PLASTIC development process model. Section 4 gives an overview of the CHAMELEON framework. A running example will be presented all throughout the evolution of the paper until Section 5 where a simple usage scenario is described. Related work is briefly discussed in Section 6 and concluding remarks as well as future directions are given in Section 7.

2 Setting the "Context"

Context awareness and *adaptation* have become two key aspects to be considered while developing applications for B3G networks. As pointed out in [BEII06], while providing/consuming services, applications need to be *aware of* and *adaptive to* their context, i.e., the combination of user-centric data (e.g., information of interest for users according to their current circumstance) and resource/computer-centric data (e.g., resource limits and conditions of devices and network).

Strictly concerning our own purposes, *context awareness* identifies the capability of being aware of the user needs and of the resources offered by an execution environment (e.g., processor, memory, display, I/O capabilities, available radio interfaces), in order to decide whether that environment is suited to receive and execute the application in such a way that end-users expectations are satisfied. *Adaptation* identifies the capability of changing the application in order to

comply with the current context conditions. In order to perform an adaptation it is essential to provide an actual way to model the characteristics of the application itself, of the heterogeneous networking infrastructure, and of the execution environment, aiming at a high *end-user degree of satisfaction* (depending on requested and offered SLS). Thus, while delivering services, it is useful to be able to reason about the *resources demanded* by an application (and its possible adaptation alternatives) and the ones *supplied* by the hosting environment. It is worth to note that although a change of context is measured in quantitative terms, i.e., in terms of availability of resources, an application can only be adapted by changing its behavior - i.e., its functional/qualitative specification.

In this setting, three different construction approaches towards adaptable applications might be considered: (*i*) *self-contained* applications that embed the *adaptation logic* as a part of the application itself and, therefore, are a-priori instructed on how to handle dynamic changes in the environment hence reacting to them at runtime; (*ii*) *tailored* applications that are the result of an adaptation process which has been previously applied on a generic version of the application; (*iii*) *middleware-based* adaptable applications in which the middleware embeds the adaptation logic in the form of meta-level information on how the applications can be adapted. Self-contained adaptable applications are inherently dynamic in their nature but suffer the pay-off of the inevitable overhead imposed by the adaptation logic. On the contrary, tailored adapted applications have a lighter code that make them suitable also for limited devices, but are dynamic only with respect to the environment at deployment time, while remaining static with respect to the actual execution, i.e., they cannot self-adapt to runtime changes in the execution environment. Considering the huge variety and limitedness of our target devices, and the complexity of B3G networks would make unfeasible the deployment of a fully self-adaptive application (i.e., a potentially huge-sized *self-contained* application) suitable for any resulting execution environment and possible context in which the user can move.

The framework CHAMELEON is for tailored applications. In this paper we propose how it can be used for tackling self-evolution (against context changes monitored through CHAMELEON as supported by the PLASTIC B3G middleware) through dynamic (re)deployment of adaptation alternatives and finally, in Section 7, we argue how, by means of (physical) mobility patterns [MM07], it can be extended towards a compromise between self-contained and tailored service applications aiming at a more seamless self-evolution.

3 Developing context-aware services

In this section we briefly introduce the PLASTIC development process model, that relies on the PLASTIC Conceptual Model, by only focussing on specific details pertaining the CHAMELEON-based approach to self-evolution we are going to present in following sections. For a complete description (text and diagrams) of the conceptual model and for a detailed instantiation of the PLASTIC development process model we entirely refer to [PLAb] and [ABC+07, PLAc], respectively. The conceptual model will be textually described by emphasizing *words* that identify conceptual model entities.

The conceptual model is a reference model formalizing the needed concepts to realize service-oriented B3G applications. The central entity here is *Service* - abstract archetypical notion of a real world service. In the context of B3G networking, we specifically focus on services concretized by *Adaptive Software Service*s, which are deployed and accessed in the specific network

of interest. Implemented by context adaptable *Software Component*s, *Adaptive Software Service*s allow *Service Consumer* parties to access needed services anywhere over the B3G network, from any device, as provisioned by *Service Provider* parties. In the light of our goal, an adaptive software service might be provided/consumed by CHAMELEON-based adaptable applications that are consequently able to run on resource-constrained *PLASTIC-enabled device*s.

A key feature of services for B3G networks is *Context-aware Adaptation* according to *Consumer-side*, *Network-side* and *Provider-side Context*s of service consumption. These various contexts of service consumption impact upon the QoS of the service, also decomposing into *Consumer-side*, *Network-side* and *Provider-side QoS*s. In particular, services may guarantee QoS properties in a given context, possibly leading to service adaptation in broader contexts.

On one side, network-side context identifies the characteristics of the (multiple) network(s) between consumer and provider such as number and type of networks, number of active users, number of available services, security, transmission protocol, access policy, etc. This is of particular relevance, since the network context impacts upon the QoS represented by network-side QoS (e.g., bitrate, transfer delay, packet-loss, network coverage). On the other side, considering the mobile nature of a PLASTIC application deployed over heterogeneous PLASTIC-enabled devices, the provider- and consumer-side contexts model device characteristics in terms of available resources (e.g., screen resolution, CPU frequency, memory size, available radio interfaces).

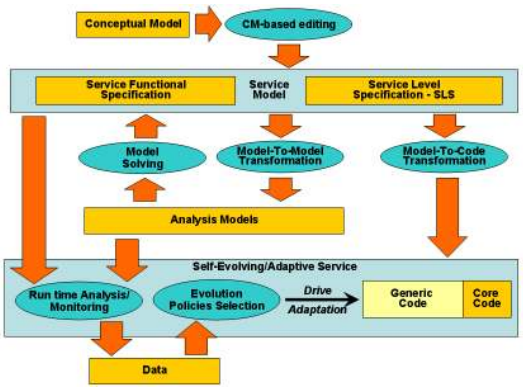

Figure 1: Plastic Development Process

In our setting, overall contextual information is retrieved by CHAMELEON exploiting the *PLASTIC B3G middleware*. The middleware provides runtime support for the deployment and access/consuption of services and is the key to ease the management of B3G networking. Exploitation of this specific middleware then relies upon the use of dedicated software components, i.e., *Consumer-side*, *Bridge-side* and *Provider-side Middleware*s, deployed on the service's consumer, bridge and provider, respectively. The role of bridge is introduced in addition to the canonical roles for SOA, i.e., consumer and provider. More specifically, bridges are those (PLASTIC-enabled) devices which deploy and run the bridge-side middleware component and act as mediators. They are in charge of routing communication among independent networks making up B3G.

Introducing the notion of *requested Service Level Specification* (SLS) and *offered SLS*, PLASTIC considers the (extra-functional) preferences that will be used to establish the *Service Level Agreement* (SLA) between consumer and provider. The SLA is an entity modeling the conditions on the QoS accepted by both consumer and provider. SLA represents a kind of contract that is influenced by the requested SLS and the context where the service has to be provided/consumed. When a new service request is formulated, the PLASTIC platform has to (possibly) negotiate the QoS and upkeep it (possibly) through self-evolution. This contractual procedure may terminate either with or without a QoS agreement between consumer and provider.

With reference to Figure 1, the service and the relative (potential) client models are specified

in terms of their functionalities and SLS (possibly along with demanded/supplied resources). Model-to-model transformation is performed in order to derive models for different kinds of analysis (e.g., for defining/estimating/refining SLSs). Some models (not necessarily different from the previous ones) will be made available at deployment- and run-time to allow the adaptation of the service to the execution context, based on data retrieved by run-time analysis/monitoring and evolution policies (see the box "Self-Evolving/Adaptive Service" shown in Figure 1). The SLS will drive the adaptation strategies through Model-To-Code transformation (right-hand side of Figure 1) that is used to build both the core and the adaptive code of the application. The core code is the frozen portion of the developed self-evolving/adaptive service and represents its invariant semantics. The adaptive one is a "generic code". A generic code embodies a set of (preemptively envisioned) adaptation alternatives that will make the code capable of evolving. This code portion allows for evolution in the sense that, from contextual information and possible changes of the user needs, a currently deployed alternative - that is no longer apt - can be dynamically un-deployed and a new one - with the desired aptitude - subsequently (re-)deployed. In fact, a particular alternative might be suitable for a particular execution context and specified user needs. During service consumption, run-time analysis, monitoring and evolution policies selection are performed and, basing on the results, a new alternative might be selected. The generic code is written by using the extended Java within the *Development Environment* while the alternatives' selection is carried out through the *Customizer*, both part of the CHAMELEON framework we are about to introduce.

4 The CHAMELEON framework

CHAMELEON allows the development and deployment of Java applications that, via the envisioned adaptation alternatives, are generic and can be correctly adapted with respect to a dynamically provided context. Figure 2 shows the components of the framework architecture.

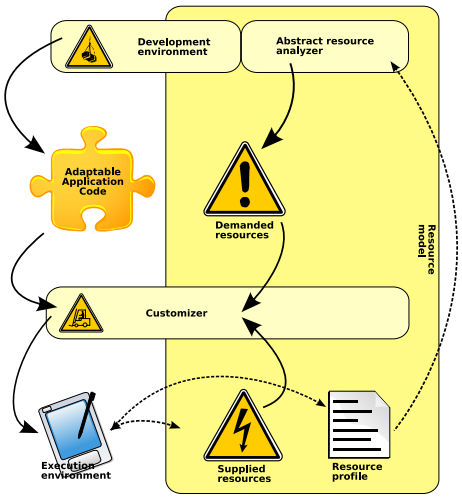

Figure 2: Chameleon Architecture

▷ **Development Environment**. The *Development Environment* is a standard Java development environment that provides developers with a set of ad-hoc extensions to Java for easily specifying, in a flexible and declarative way, the generic code. Methods are the smallest building blocks that can be adapted. Figure 3 shows a snippet of an adaptable MIDlet for consuming an e-learning service that promotes effective student/professor cooperation, providing a set of flexible tools that aid the work of online learning communities.

The *adaptable class* e-learningMidlet contains four *adaptable methods*: connect, getLesson, saveState and restoreState (bearing the keyword adaptable). Adaptable methods do not have a definition in the adaptable class where they are declared but they are defined within *adaptation alternatives* (see the keywords alternative and adapts). It is possible to specify more than one alternative for a given adaptable class provided that for each adaptable method there exists at least one alternative that contains a definition

for it. Such a generic code will be preprocessed and a set of standard Java application alternatives will be derived by suitably combining the adaptable methods implementations provided in the various alternatives. These are then compiled using standard Java compilers.

The CHAMELEON framework is used to adapt the e-learning service according to the execution- and network-context of the student device and to his/her requested SLS. In particular, the adaptable e-learningMidlet has two alternatives, each one providing implementation for all the adaptable methods. The GPRS alternative connects via GPRS and, considering the limited speed of this connection, allows for streaming (via the getLesson method) the lesson slides only. The WiFi alternative connects via WiFi and, exploiting the higher connection speed, allows for streaming the high-quality video lesson with all its multimedia content (slides plus other interactive multimedia objects). Upon student request of the e-learning service, the suitable Midlet alternative for consuming such a service will be delivered and deployed on the device with an On-The-Fly fashion by using our suitable implementation of the Over-The-Air (OTA) provisioning technique [OTA] for automatic delivery and deployment of adapted applications.

Given that CHAMELEON has to allow to dynamically un-deploy a running adaptation alternative and re-deploy a new alternative, it requests that the two methods saveState and restoreState must be implemented, in addition to the natively required MIDlet methods - i.e., pauseApp, startApp and destroyApp. Contextualizing in our example, let us assume that the WiFi alternative is no longer suitable for running due to a change in context: it will have to be un-deployed. Upon un-deployment CHAMELEON will invoke the saveState method to save into the state a number of key informations such as the current frame of the video lesson, current slide number, audio track offset and so on. Let us now suppose that, entering the new context, the GPRS alternative becomes more suitable, and hence has to be (re-)deployed substituting the previously running WiFi alternative. The GPRS alternative will be dynamically deployed and, since only slides can be now viewed, it will be set to start from the slide saved into the state as restored through the restoreState method.

```
adaptable public class e-learningMidlet extends MIDlet
                              implements CommandListener {
    State state;
    public e-learningMidlet(){...}
    /* life-cycle management methods */
    protected void pauseApp(){...}
    protected void startApp(){...}
    protected void destroyApp(){...}
    /* CHAMELEON specific state management methods */
    adaptable protected void saveState();
    adaptable protected void restoreState();
    /* e-learning specific methods */
    adaptable void connect();
    adaptable void getLesson();
    ...
}
//-------------------------------------------------------------
alternative class GPRS adapts e-learningMidlet {
    void connect() {
        Annotation.slsAnnotation("Cost(low),e-lQuality(low)");
        plasticMiddleware.connectViaGPRS();
    }
    void getLesson() { /* streaming of the lesson slides */ }
    protected void saveState() { /* save into the state the
                        number of the current slide */ ...}
    protected void restoreState() {/*restore the saved state*/}
}
//-------------------------------------------------------------
alternative class WiFi adapts e-learningMidlet {
  void connect() {
      Annotation.slsAnnotation("Cost(high),e-lQuality(high)");
      plasticMiddleware.connectViaWiFi();
    }
    void getLesson() { /* streaming of the video lesson
                        with its multimedia content */
        Annotation.resourceAnnotation("Battery(high)");
        ...
    }
    protected void saveState() { /* save into the state the
                    current frame of the video lesson */ ...}
    protected void restoreState() {/*restore the saved state*/}
}
```

Figure 3: An adaptable Midlet

Annotations may also add information about particular code instructions (see the keyword Annotation). They are specified at the generic code level by means of calls to the "do nothing" methods of the dedicated Annotation class. In this way, after compilation, annotations are encoded in the bytecode through well recognizable method calls to allow for easy processing. For instance, in Figure 3, the method call Annotation.slsAnnotation ("Cost(high), e-lQuality(high)"), first line of the connect method in the WiFi alternative, specifies that the WiFi connection, and hence the WiFi alternative, bears a high

cost, but provides a high quality e-learning lesson. On the other hand, the method call Annotation.resourceAnnotation ("Battery(high)"), first line of the getLesson method, demands for a high battery state-of-charge, since the streaming of the video lesson along with its multimedia content calls for a considerable amount of energy to be consumed.

▷ **Resource Model** The *Resource Model* is a formal model that allows the characterization of the computational resources needed to consume/provide a service. The Resource Model, enables the framework to reason on the set of adaptation alternatives and allows it to decide the "best-fit", depending on execution context information.

Resource Definition

defineRES Battery **as** {low, medium, high}
defineRES WiFi **as** Boolean
defineRES WiFiNet **as** Boolean
defineRES GPRS **as** Boolean
defineRES GPRSNet **as** Boolean

$Demand_{GPRS}$={`GPRS(true),GPRSNet(true)`}
$Demand_{WiFi}$={`WiFi(true),WiFiNet(true),Battery(high)`}
Supply 1={`GPRS(true),WiFi(true),GPRSNet(true),WiFiNet(false),Battery(low)`}
Supply 2={`GPRS(true),WiFi(true),GPRSNet(true),WiFiNet(true),Battery(high)`}

Figure 4: Resource examples

Broadly speaking a resource is any item that is required to accomplish an activity or complete a task. Some resources are subject to consumption (e.g., energy, heap space), while others, if present, are never exhausted (e.g., function libraries, network radio interfaces). Thus, we model a *resource* as a typed identifier that can be associated to *Natural*, *Boolean* or *Enumerated* values. Natural values are used for consumable resources whose availability varies during execution. Boolean values define resources that can be present or not (i.e. non-consumable ones). Enumerated values can define non-consumable resources that provide a restricted set of admissible values (e.g. screen resolution, network type). Left-hand side of Figure 4 shows an example of some resource definitions. A *resource instance* is an association $res(val)$ where a resource *res* is coupled to its value $val \in typeof(res)$ (e.g. `WiFi(true)`). A *resource set* is a set of resource instances with no resource occurring more than once. It is used to specify both the *resource demand* of an alternative and the *resource supply* of an execution environment.

Referring to the adaptable MIDlet in Figure 3, $Demand_{WiFi}$ (as shown in Figure 4) might be a simple resource demand (automatically CHAMELEON-derived through ARA - see below) associated to the WiFi alternative specifying that, to run correctly, the WiFi alternative will require a device equipped with the WiFi radio interface (`WiFi(true)`), connected to a WiFi network (`WiFiNet(true)`) and with a high battery state-of-charge (`Battery(high)`). On the same figure, *Supply 2* might be a resource set specifying the resource supply of a device equipped with both GPRS and WiFi radio interfaces, connected to both GPRS and WiFi networks and with a high battery state-of-charge.

The resource model also defines the notions of *compatibility* between two resource sets. Compatibility is used to decide if an application can run safely on a certain device: a *resource supply* is compatible to a *resource demand* if for every resource demanded by the alternative, a "sufficient amount" is supplied by the execution environment. For instance, in Figure 4, the *Supply 1* is only compatible with $Demand_{GPRS}$ (it neither has `WiFiNet(true)` nor `Battery(high)`); conversely, *Supply 2* is compatible with both demands.

▷ **SLS Model** The *SLS Model* is a formal model that allows the specification of extra-functional preferences that will be used to establish the SLA. Together with the Resource Model, the SLS Model enables the framework to choose the "best" adaptation alternative depending on both ex-

ecution context information and consumer specific preferences. This model bases itself around the same formalisms as the Resource Model. SLS Model is basically identical to the Resource Model and, as shown in Figure 5 is used for specifying both requested and offered SLSs.

SLS Definition:
defineSLS Cost **as** {low, high}
defineSLS e–lQuality **as** {low, high}

***Requested** (or **Offered**) **SLS**$_{(GPRS)}$*={Cost(low), e-lQuality(low)}
***Requested** (or **Offered**) **SLS**$_{(WiFi)}$* = {Cost(high), e-lQuality(high)}

Figure 5: SLS examples

Referring again to the adaptable MIDlet in Figure 3, $Offered\ SLS_{GPRS}$ (as shown in Figure 5) might be a simple offered SLS (automatically CHAMELEON-derived through ARA - see below) associated to the GPRS alternative, offering a low quality lesson at a low connection cost. It must be noted that offered SLSs might also be specified, by the developer/provider, without embedding them within the generic code. In addition to that of compatibility, the resource model and the SLS model together allow to define the notions of *goodness*. Goodness is used for comparing resource demands and offered SLSs associated to different alternatives of the same application, and hence for choosing the best alternative among all its compatible ones. It is based on a notion of priority among resources/SLSs and order between resource/SLSs' instances. For example, let's assume that a student wishes to attend a full quality on-line lesson, even at a high cost (i.e., *Requested SLS*:{`e-lQuality(high),Cost(high)`} in Figure 5). The notion of goodness, combined with the notion of compatibility, permits to state that the WiFi alternative (associated to $Demand_{WiFi}$ of Figure 4 and to $Offered\ SLS_{WiFi}$ of Figure 5) is, yes compatible, but also better than the GPRS alternative (associated to $Demand_{GPRS}$ and to $Offered\ SLS_{GPRS}$). The framework encodes both resource and SLS sets into XML files, not shown for sake of space.

▷ **Abstract Resource Analyzer** The *Abstract Resource Analyzer* (ARA) is an interpreter that, abstracting a standard JVM, is able to analyze Java applications and derive their resource consumption also deriving, if specified, the offered SLSs. ARA is parametric with respect to the characteristics of the execution environment as described through a *resource consumptions profile* that expresses the impact that computational elements (i.e., the Java Bytecode Language instructions) have on resources (see also the **Execution Environment** below). Specifically, resource consumption profiles associate resources consumption to particular patterns of Java bytecode instructions specified by means of *regular expressions* [1]. Basically, we can look at ARA as a function that takes as input a Java application's bytecode, a resource consumption profile and gives as output a resource set that represents the resource demand of the application, and (optionally) an SLS set that represents the offered SLS (i.e., the resources needed for that program to be safely executed by the target execution environment with the offered SLS).

1) invokevirtual PlasticMiddleware.connectViaWiFi → {WiFi(true), WiFiNet(true)}
2) invokevirtual PlasticMiddleware.connectViaGPRS → {GPRS(true), GPRSNet(true)}

Figure 6: A resource consumption profile excerpt

Figure 6 shows an excerpt of a possible resource consumption profile defined over the resources of Figure 4 and possibly provided by the student's device in the

[1] Resource consumption profiles can be created on the basis of experimental results based on benchmarking tools, on informations provided by device manufacturers or network instruments.

e-learning scenario. For example, row "1)" states that to execute the bytecode instruction invokevirtual PlasticMiddleware.connectViaWiFi the JVM will require the device to have a WiFi radio interface and a WiFi network within range. Note that this bytecode instruction is part of the WiFi alternative's connect method bytecode as derived from the compilation of the generic code after this has been preprocessed to obtain a standard Java code. Analyzing this bytecode against the provided profile, ARA will realize that for the WiFi alternative to be correctly executed on the student's device, the device itself must have the above mentioned resources. Consequently, the WiFi alternative "asks" for those resources by means of the two entries WiFi(true) and WiFiNet(true) within its resource demand. Finally, the resource demands and offered SLSs of the two MIDlet alternatives of Figure 3 (derived by ARA on the base of the provided resource consumption profile and the resource/SLS annotations) are those shown in Figure 4 and Figure 5, respectively.

▷ **Execution Environment** The *Execution Environment* can be any device that will host the execution of the code. We target execution environments provided by PDAs, smart phones, etc. The execution environment is not strictly part of our framework, however, it has to provide a declarative description of the resources it makes available to consume/provide the service (i.e., the resource supply) and the resource consumption profile. In other words, it has to be a PLASTIC-enabled device running a *Consumer-side Chameleon* component that, via the embedded *Consumer-side Middleware* (see Section 3), is able to communicate with a *Provider-side Chameleon* component.

▷ **Customizer** The *Customizer* takes in the resource supply, the provided profile, explores the possible adaptation alternatives' space and, according to compatibility and goodness (see Resource and SLS Models in Section 4), defines the proper evolution policy and decides the actual best alternative. It will deliver that alternative (i.e., standard bytecode) that can then be eventually un-deployed and substituted by a new dynamically (re-)deployed alternative properly selected according to the defined evolution policy (see Section 3). User decisions will be called up to choose among different alternatives that are "equally-suitable" from the Customizer perspective and negotiation might possibly be performed.

CHAMELEON has been fully implemented in *Java* (although other languages are eligible) and uses XML-based data encoding for the **Resource** and **SLS Model**s. It is worth mentioning that, even though both the ARA and the Customizer could be space and time consuming, they are executed on the provider-side of the framework which does not suffer resource limitations.

5 Case Study

Damien is a student traveling from Italy to Canada. While on the train to the airport, he wants to start an e-learning session through his stand-alone e-learning client, as deployed upon registration to the e-learning service.

▷ **Foreground:** Damien takes up his smartphone and connects, through his e-learning client, to the e-learning service and chooses to get the latest available lesson of the Computer Science course. Since Damien wants to obtain a fully featured lesson, he is willing to invest considerably in the process. When the e-learning client asks for the cost by displaying a multiple-choice form with both high and low cost options, he will opt for a high cost solution delivering the fully featured version. After processing the choice, the e-learning service will inform Damien, via a pop-up message, that the fully featured lesson is not obtainable since a high-speed connection cannot be established and his battery state-of-charge is insufficient to support the energy demands

for the duration of the fully featured solution. As an alternative, the e-learning service proposes a low-cost version of the lessons in which only slides will be provided. The system also informs Damien that as soon the conditions exist, it will automatically switch to the fully featured version. Damien accepts, the *SLA* (see Section 3) is established, and the first slides appear. Damien also recharges his smartphone. Upon reaching the airport Damien, after a slight hiccup, will be able to enjoy video, audio and other multimedia content of the fully featured version.

▷ **Behind the scenes:** Damien's smartphone is indeed a PLASTIC-enabled device. It embeds the *Consumer-side Middleware* and the *Consumer-side Chameleon* components (see Section 3 and 4). Initially, the e-learning client will exploit the former to establish a GPRS connection to the B3G network, and the latter to express its desired cost through the multiple-choice form (i.e., *Requested SLS* ={`Cost(high), e-lQuality(high)`} as shown in Figure 5). After retrieving the *Network-side Context* and the *Consumer-side Context*, (i.e. *device context*), once again through the Middleware and CHAMELEON facilities, the client will additionally send the supplied resources (i.e., `Supply 1 =` {`GPRS(true), WiFi(true), GPRSNet(true), WiFiNet(false), Battery(low)`} as shown in Figure 4) as well as the resource consumption profile (as shown in Figure 6). Processing *Requested SLS*, provider-side CHAMELEON component will deliver, via OTA application provisioning [OTA], the GPRS alternative. While not being the "highest-*goodness*" alternative, it is the only *compatible* one (see **Resource** and **SLS Model**s in Section 4). The established *SLA* states that, as soon as the conditions rise, the fully featured version of the lesson has to be provisioned. To meet such an agreement, the *Consumer-side Chameleon* component monitors the battery state-of-charge and WiFi connection availability, since, via the predefined *Evolution Policy*, it knows that the "higher-*goodness*" WiFi alternative is ready to be re-deployed when these two conditions are met. Upon reaching the airport, the CHAMELEON monitor detects that the battery is fully charged and WiFi networking is in range and, hence, immediately initiates the alternative switching previously envisioned. The WiFi alternative, as selected by the *Customizer* (see Section 4), will be delivered on-the-fly to Damien's device, the GPRS alternative will be un-deployed (after saving the current state), and the newly acquired alternative will be dynamically re-deployed and put into execution, immediately restoring the state.

6 Related work

For sake of space, we cannot address all the recent related work in the wide domain of the PLASTIC project and in the CHAMELEON domain: we provide only some major references.

Current (Web-)service development technologies, e.g., [Ecl, Pro04, YKKP] (just to cite some), address only the functional design of complex services, i.e., they do not take into account the extra-functional aspects (e.g., QoS requirements) and context-awareness. Our process borrows concepts from these well assessed technologies and builds on top of them to make QoS issues clearly emerging in the service development as well as to take into account context-awareness of services for self-adaptiveness purposes.

Our resource model and abstract resource analyzer can be related to other approaches to resource-oriented analysis, such as [AM06, Bar05, AGZ07, SMM07]. All these approaches use a resource model as we do and give an absolute over-estimation of resources' consumption. Differently from us, they do not use these models in the context of adaptation and they do not provide a reasoning mechanism for selecting the proper alternative basing on estimations that

allow for a relative comparison of the alternatives.

In [PP] an interesting approach may be found which considers concerns separation between application logic and adaptation logic. The approach makes use of Java annotations to express metadata needed to enable dynamic adaptation. Stemming from the same idea of concerns separation, in [REF+08], supported by the MUSIC project [2], the authors propose the design of a middleware- and architectural-based approach to support the dynamic adaptation and reconfiguration of the components and service composition structure. They use a planning-based middleware that, basing on metadata included in the available plans, enables the selection of the right alternative architectural plan for the current context. Similarly to us, the idea for the designed approach is based on requested and offered QoS, and supports SLA negotiation. Differently from us, they do not tackle adaptability to the device execution context through resource oriented analysis, parametric w.r.t. its resource consumption profile.

7 Conclusion and Future Work

In this paper we described how a declarative framework for tailoring adaptable services (called CHAMELEON) [IMN04, MI07] is used within IST PLASTIC project [PLAa]. This framework is at the basis of the PLASTIC process model for the development and deployment of adaptable software applications targeted to mobile (resource-constrained) devices in the heterogeneous B3G network. The specific deployed application is customized (i.e., *tailored*) with respect to the context at deployment time but, at run time, it is frozen with respect to evolution. Evolution (against context changes monitored through CHAMELEON as supported by the PLASTIC B3G middleware) is realized by dynamically un-deploying the no longer apt application alternative and subsequently (re-)deploying a new alternative with the desired aptitude.

Assuming that, upon service request, the user knows (at least a stochastic distribution of) the mobility pattern [MM07] he will follow during service usage, an amount of determinism is introduced permitting to identify the successive finite contexts the user can move along during service usage. Following this approach, an enhanced version of CHAMELEON would be able to generate code that is a compromise between self-contained and tailored adaptable code. The code would embed all the adaptation alternatives that, associated to the specified mobility pattern, are necessary to preserve the *offered SLS*. That is, the code would also embed some dynamic *adaptation logic* which is able to recognize context changes, seamlessly "switching" among the embedded adaptation alternatives. In this way, a seamless evolution is performed among the embedded alternatives associated to the mobility pattern, while the un-/re-deployment evolution we presented is performed when moving out of the mobility pattern's context.

Ontology based specifications might be used to establish a common vocabulary and relationships among resource/SLS types. This would allow to relate resource/SLS types and to predicate about a common set of related types. For instance, a demand of {`Energy(high)`} could be related to a supply of {`Battery(high)`}.

Moreover, the resource model and the SLS model might be described as Java types. These types can then be used within *Java5* annotations and our annotation mechanism might be improved, accordingly. This would provide the developer with a solid means for describing fully typed resource/SLS needs and offers, and static analysis tools (such as Spoon [Spo]) are then able

[2] http://www.ist-music.eu/

to derive an XML-based representation of the models that are used to match service alternatives to resource consumption profiles and requested SLS.

Acknowledgements: This work has been partially supported by the IST project PLASTIC. We would like to acknowledge the anonymous reviewers for the valuable contributions in providing high-quality and constructive comments.

Bibliography

[ABC+07] M. Autili, L. Berardinelli, V. Cortellessa, A. D. Marco, D. D. Ruscio, P. Inverardi, M. Tivoli. A Development Process for Self-Adapting Service Oriented Applications. In *ICSOC*. 2007.

[AGZ07] E. Albert, S. Genaim, M. Zamalloa. Heap Space Analysis of Java Bytecode. In *ISMM'07*. ACM Press, Oct. 2007.

[AM06] D. Aspinall, K. MacKenzie. Mobile Resource Guarantees and Policies. In *Construction and Analysis of Safe, Secure, and Interoperable Smart Devices*. 2006.

[Bar05] G. Barthe. MOBIUS, Securing the Next Generation of Java-Based Global Computers. ERCIM News, 2005.

[BEII06] A. Bertolino, W. Emmerich, P. Inverardi, V. Issarny. Softure: Adaptable, Reliable and Performing Software for the Future. *FRCSS*, 2006.

[Ecl] Eclipse.org. Eclipse Web Standard Tools. http://www.eclipse.org/webtools.

[IMN04] P. Inverardi, F. Mancinelli, M. Nesi. A declarative framework for adaptable applications in heterogeneous environments. In *SAC*. 2004.

[MI07] F. Mancinelli, P. Inverardi. Quantitative resource-oriented analysis of Java (Adaptable) applications. In *WOSP*. 2007.

[MM07] A. D. Marco, C. Mascolo. Performance analysis and prediction of physically mobile systems. In *WOSP*. 2007.

[OTA] Over-The-Air (OTA). http://developers.sun.com/mobility/midp/articles/ota/.

[PLAa] PLASTIC IST STREP Project. http://www.ist-plastic.org.

[PLAb] PLASTIC IST STREP Project. Deliverable D1.2: Formal description of the PLASTIC conceptual model and of its relationship with the PLASTIC platform toolset.

[PLAc] PLASTIC IST STREP Project. Deliverable D2.2: Graphical design language and tools for resource-aware adaptable components and services.

[PP] N. Paspallis, G. A. Papadopoulos. An Approach for Developing Adaptive, Mobile Applications with Separation of Concerns. In *COMPSAC. 2006*.

[Pro04] A.-M. Project. Methodological Framework for Freeband Services Development. 2004. https://doc.telin.nl/dscgi/ds.py/Get/File-47390/.

[REF+08] R. Rouvoy, F. Eliassen, J. Floch, S. O. Hallsteinsen, E. Stav. Composing Components and Services Using a Planning-Based Adaptation Middleware. In *SC*. 2008.

[SMM07] C. Seo, S. Malek, N. Medvidovic. An energy consumption framework for distributed java-based systems. In *ASE*. 2007.

[Spo] Spoon project. http://spoon.gforge.inria.fr.

[YKKP] H. Yun, Y. Kim, E. Kim, J. Park. Web Services Development Process. In *PDCS'05*.