# Q-CAD: QoS and Context Aware Discovery Framework for Mobile Systems

Licia Capra, Stefanos Zachariadis and Cecilia Mascolo
Dept. of Computer Science, University College London
Gower Street, London WC1E 6BT, UK
{L.Capra|S.Zachariadis|C.Mascolo}@cs.ucl.ac.uk

## Abstract

*This paper presents Q-CAD, a resource discovery framework that enables pervasive computing applications to discover and select the resource(s) best satisfying the user needs, taking the current execution context and quality-of-service (QoS) requirements into account. The available resources are first screened, so that only those suitable to the current execution context of the application will be considered; the shortlisted resources are then evaluated against the QoS needs of the application, and a binding is established to the best available.*

## 1 Introduction

Technological advances, both in wireless networking and portable device capabilities, have met social popularity, so that we are now witnessing an increase in the number of devices and services we use to accomplish our daily tasks. Interaction with these services and devices is enabled by means of various components, some located on the mobile device, some available for download remotely. We refer to these services, devices and components as *resources*.

Research in the area of resource discovery for pervasive environments has been very intense in recent years. Its main focus has been on the development of efficient algorithms that take the pervasive network topology into account when routing advertisements and queries (e.g., [4, 5]). However, more effort is needed to improve the user experience, so that the resources that the user considers most suited in the current execution context and according to his/her quality-of-service (QoS) needs are actually selected.

In this paper we present Q-CAD, a *context* and *QoS aware* resource discovery and selection framework for pervasive environments. Each application dynamically encodes in an *application profile* the way context should influence the discovery of, and the binding to, resources; Q-CAD uses this information to reduce the resources available to the application in the current context to a subset of 'plausible' ones. Each application also encodes the QoS needs of the user into a *utility function* that Q-CAD applies to select the most suitable resource among the plausible ones. Q-CAD builds on the following assumptions: the existence of a shared ontology to refer to context elements and conditions, resource names and characteristics, and non-functional requirements; the integration with an existing discovery protocol for pervasive networks on which Q-CAD relies to route advertisements and queries.

The paper is structured as follows: Section 2 describes Q-CAD application profiles and utility functions, and details the discovery and selection protocol; Section 3 presents the Q-CAD architecture; finally, Section 4 compares Q-CAD with related work and presents our conclusions. For more detailed and up-to-date information about Q-CAD please refer to [2].

## 2 Q-CAD Model

Q-CAD achieves context and QoS awareness by means of *application profiles* and *utility functions* respectively. In this section, we describe the information they encode and illustrate how the discovery and selection protocol uses them. Before doing so, we define what a resource is in this setting, what *binding* to a resource implies and we introduce the concept of *resource descriptor*.

**Q-CAD Resources, Descriptors and Binding.** Central to our model is the notion of a *resource*. The resources that the Q-CAD model considers are: *services* provided by remote providers, *sensors* from which an application may retrieve data, and *components* located remotely and that can be downloaded and deployed on the local host. We refer to these resources as *remote resources*, to distinguish them from those local to a device (e.g., battery, memory, CPU, etc.). We assume remote resources are uniquely identified by means of an addressable naming scheme that is resolved by the underlying communication framework. We define the *binding* to a resource (i.e., the last step of a resource discovery and selection process) as the association of the selected remote resource to a *component* that is local to the

```
(component, displayVideo)     (size, 70KB)
(code, display800600.jar)     (cost, $10)
(resolution, 800x600)         (memory, 2)
(version, 2.1)                (battery, 4)
(platform, JVM2)
```

**Figure 1. Example of Resource Descriptor.**

device and that is able to interact with it. A remote resource could itself be a component: in this case, binding refers to downloading and deploying the component on the local system. Every remote resource is also associated with a static specification, or *resource descriptor*, that characterises the resource by means of a list of attribute/value pairs. Figure 1 illustrates an example of a remote resource descriptor for a component that displays video at a resolution of $800x600$; besides implementation details, the descriptor contains information that can be used to assess the quality of the resource itself; this includes, for example, estimates of local resources consumption.

**Application Profiles.** Application profiles specify how the user wishes the context to influence the discovery of remote resources. Discovery can be either *proactive* (i.e., the consequence of an explicit request of the user to locate a service) or *reactive* (i.e., the result of context changes). Both types of discovery demand a similar behaviour from the discovery framework: locating and binding to a resource (be it a service provider, a sensor, or a component) that is best suited in the current context (*context-awareness*) and according to the current non-functional requirements of the user (*QoS-awareness*). In the remainder of the paper, we provide examples of proactive discovery only [1].

Let us imagine a tourist that wishes to print the pictures she has taken with her digital camera. In order to do so, she has to discover and select a photo development service provider, among the many available. Different parameters may influence this choice: for example, location of the provider, cost of the service, quality of the prints, and so on. For each remote resource the application may be willing to bind to, the proactive encoding of its profile contains an association between the resource name (tag `<BIND_RESOURCE>`) and the context conditions that must hold for the binding to be enabled (tag `<REMOTE_CONTEXT>`). For example, the encoding shown in Figure 2 states that only printing service providers that give customers at least 100MB of disk space should be considered. This condition acts as a filter over the possibly high number of providers of the same service. Only one context configuration (tag `<REMOTE_CONTEXT id="1">`), containing a single condition (tag `<CONDITION>`) is specified. More generally, multiple contexts can be associated to the same binding resource, and more conditions may be associated to the same context. The semantics of these encodings are the following: the binding to the remote resource is enabled if and only if *at least* one of the context configurations is enabled (*or* semantics); a context configuration is enabled if and only if *all* the conditions associated to it hold (*and* semantics). If more than one service provider passes the filtering, the actual provider to bind to will be selected using the application's utility function.

[1]A discussion of reactive discovery is available at [2].

```
<PROACTIVE id="1">
  <LOCAL_CONTEXT/>
  <REMOTE_CONTEXT/>
  <BIND>
    <BIND_RESOURCE name="printPicture">
      <REMOTE_CONTEXT id="1">
        <CONDITION name="diskSpace" op="greaterThan" value="100MB"/>
      </REMOTE_CONTEXT>
    </BIND_RESOURCE>
  </BIND>
  <ADAPT>
    <ADAPT_COMPONENT id="1">
      <LOCAL_CONTEXT id="2">
        <CONDITION name="battery" op="greaterThan" value="30%"/>
      </LOCAL_CONTEXT>
      <REMOTE_CONTEXT/>
      <ATTRIBUTES>
        <ATTRIBUTE key="component" op="equals" value="encryptedUpload"/>
      </ATTRIBUTES>
    </ADAPT_COMPONENT>
    <ADAPT_COMPONENT id="2">
      <LOCAL_CONTEXT id="3">
        <CONDITION name="battery" op="lessThan" value="30%"/>
      </LOCAL_CONTEXT>
      <REMOTE_CONTEXT/>
      <ATTRIBUTES>
         <ATTRIBUTE key="component" op="equals" value="plaintextUpload"/>
         <ATTRIBUTE key="location" op="equals" value="local"/>
      </ATTRIBUTES>
    </ADAPT_COMPONENT>
  </ADAPT>
</PROACTIVE>
```

**Figure 2. Example of Proactive Encoding.**

Once a remote service provider has been discovered and selected, the application has to decide how to interact with it (i.e., what component to use), as different behaviours/protocols may be available. The component should be selected out of a list of desirable ones (tag `<ADAPT_COMPONENT>`); the choice depends on the following information, that is attached to each of these components: local context, remote context, and application preferences. For example, the encoding of Figure 2 dictates that pictures should be uploaded to the provider site using a component that supports an encryption protocol when the remaining battery is above $30\%$, while using a plaintext upload otherwise. If multiple components match the criteria given, the utility function will be used to select the one that best satisfies the QoS needs of the user. Note that the chosen component may not be available locally; in this case, discovery, download and deployment of a component implementation is required; this process is almost identical to the one that has been discussed above, as components are treated as yet another type of resource.

**Utility Functions.** Once the pruning operated by application profiles has been completed, utility functions are used to select the best resource out of the context-suitable ones, according to the non-functional requirements of the user. Similarly to profiles, a utility function exists for each application, so that user preferences may vary depending on the particular application. Figure 3 illustrates an example of a utility function encoding. As shown, the encoding is divided into two parts: a `<MAXIMISE>` part, and a `<RETURN>` part. Under the tag `<MAXIMISE>`, the application lists the non-functional parameters it is interested in, together with weights that express their relative importance. The `<MAXIMISE>` part of the utility function is executed on a resource descriptor, as a summation of products (i.e., normalised estimates multiplied by weights, as found in the resource descriptor and utility function, respectively); it returns a single value that can be used to compare the quality of different resources. However, there are cases in which the selection process should not be fully automated. For example, the user may not want to download a component that maximises her non-functional requirements, if it is too expensive. We use the `<RETURN>` part of the utility function specification when intervention on behalf of the application or user is required. Figure 3 dictates that selection can be automated if the cost of the component is less than $10; otherwise, information has to be prompted to the application to obtain the final decision. This information includes, besides the result of the maximisation part, all the attributes listed in the `<FILTER>` part of the utility function.

**Discovery Protocol.** The discovery protocol that Q-CAD realises so to achieve QoS and context awareness consists of three main steps: *matching*, *evaluation* and *selection*. On behalf of the application, Q-CAD sends a discovery message containing details about the wanted resource (e.g., component type, resolution, platform, etc.). This information can be found in the application profile and is used to prune the number of potential matches. The remote resources receiving this message evaluate it locally against their resource descriptors, and only those matching the query will reply (*matching* step). The resources that have survived the pruning now receive a message containing the application's utility function; each remote resource evaluates the function over the relevant resource descriptors and returns an answer to the querying application (*evaluation* step). Note that a resource may refuse to perform this computation, either because it does not have the capabilities to do so, or because it does not want to consume local resources. On the other hand, the application may not be willing, for privacy reasons, to disclose its utility function. In these cases, the resource descriptor may be returned instead, and the application itself will compute the utility function over the descriptor locally. Finally, if no application intervention is required, the resource that maximises the application utility is automatically selected, based on the answers received and/or the local computation performed; if intervention is required instead, the returned values are passed to the application to obtain a final choice (*selection* step).

```
<UTILITY_FUNCTION id="uf1">
  <RETURN>
    <EVALUATE>
      <ATTRIBUTE key="cost" op="greaterThan" value="10$"/>
    </EVALUATE>
    <FILTER>
      <ATTRIBUTE key="cost"/>
    </FILTER>
  </RETURN>
  <MAXIMISE>
    <ATTRIBUTE key="battery" weight="10"/>
    <ATTRIBUTE key="memory" weight="5"/>
  </MAXIMISE>
</UTILITY_FUNCTION>
```

**Figure 3. Example of a Utility Function.**

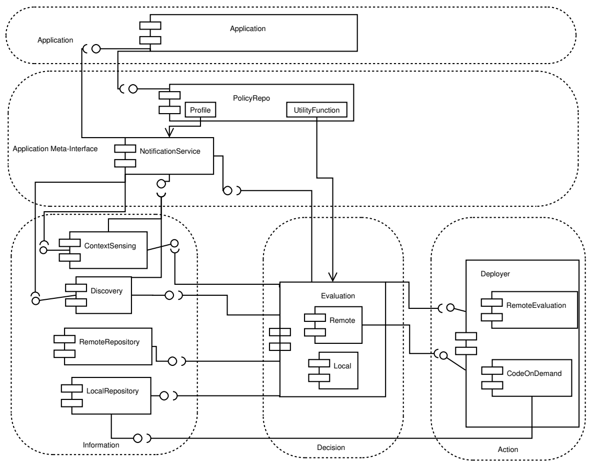

**Figure 4. The Q-CAD Architecture.**

## 3 Q-CAD Architecture

As shown in Figure 4, the Q-CAD architecture is organised into four conceptual layers: the Application Meta-Interface layer, the Information layer, the Decision layer and the Action layer.

**The Application Meta-Interface Layer** encapsulates the interaction of the applications with the Q-CAD architecture. It is composed of the *PolicyRepo* and *NotificationService* components: PolicyRepo represents the *reflective* aspects of the application, as it allows for the dynamic inspection and modification of the application profile and utility function; the NotificationService is responsible for extracting the information from the profile and passing it on to the components in the Information layer, as well as returning the result of a resource discovery to the application.

**The Information Layer** is responsible for the management of local and remote context-related information. In particular, the *ContextSensing* component is responsible for monitoring the state of the local system (e.g., remaining battery power, etc.), while the *Discovery* component is responsible for detecting the remote resources (in particular, services and sensors) currently available to the local host, that the application is interested in. The two repository components are responsible for encapsulating information about components already deployed locally (*LocalRepository*), or available for download and deployment on remote hosts (*RemoteRepository*).

**The Decision Layer** encapsulates the evaluation and selection aspects of the Q-CAD protocol. After the Information layer has performed its pruning, the Decision layer evaluates the utility function against the shortlisted resource descriptors, and selects the one that maximises the application's utility. It comprises both a *Local* and a *Remote* component, for local and remote evaluation of the utility function respectively. The execution of the Evaluation component may generate events that need application input; if that is the case, the NotificationService component in the Application Meta-Interface layer is used to pass the events to the application and get the required input.

**The Action Layer** encapsulates the logical mobility techniques [3] required by the Decision layer (i.e., code-on-demand and remote evaluation). It consists of the *Deployer* component, which comprises: the *RemoteEvaluation* component, used by the Remote component in the Decision layer to deploy the utility function on a remote host, and the *CodeOnDemand* component, that is responsible for downloading any remote component locally needed to perform adaptation. The downloaded components are registered with the LocalRepository, so that the Information layer maintains an up to date status of the system.

## 4 Discussion and Conclusions

In recent years, research has been very active in the area of service discovery for pervasive systems. Most of the work has concentrated on designing protocols and architectures that could fit the mobile network topology. Examples include: directory-based approaches, such as Jini, UPnP, the Service Location Protocol, the Salutation Architecture, and the Bluetooth Service Discovery Protocol; totally decentralised approaches based on flooding algorithms, such as SSDP; approaches for single-hop ad-hoc networks (e.g., IBM DEAPspace [8]), multi-hop (e.g., Lanes [5]), and P2P (e.g., JXTA-Search [9]). A common limitation of these approaches is that they concentrate on providing a communication infrastructure, while supporting only primitive service matching mechanisms based on the exact match of simple pre-defined attributes.

Approaches that move a step closer to our goal include: the Intentional Naming Scheme [1], an overlay that allows each node to intelligently choose the nodes to which to forward queries based on the semantics of the request; MAGNET [6], a trading framework that has been proposed to allow user-customised service matching, based on service types, rather than on service names, and [7], a QoS-aware service selection framework that takes both the user perspective and resource consumption into account. However, the semantics of service queries and matching is still not rich enough.

In this paper we have described Q-CAD, a resource discovery and selection framework for pervasive environments that supports semantically rich descriptions of both the current context and QoS needs. The Q-CAD architecture has been implemented using Java 2 Micro Edition and the SATIN [10] component model and middleware system for adaptive mobile systems. In total, the Q-CAD implementation occupies 127KB (compressed), making it suitable for mobile devices. We have implemented a benchmark application to evaluate Q-CAD performance in terms of: overhead imposed by the evaluation of context information (as encoded in application profiles), and overhead imposed by the evaluation of QoS information (as encoded in utility functions). Experimental results (available at [2]) demonstrate that Q-CAD supports rich queries and matching, while imposing a low overhead on the device.

## References

[1] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The Design and Implementation of an Intentional Naming System. In *Proc. of the 17$^{th}$ ACM Symposium on Operating Systems Principles*, pages 186–201. ACM Press, 1999.

[2] L. Capra, S. Zachariadis, and C. Mascolo. Q-CAD: QoS and Context Aware Discovery Framework for Mobile Systems. http://www.cs.ucl.ac.uk/staff/l.capra/projects/qcad, 2005.

[3] A. Fuggetta, G. Picco, and G. Vigna. Understanding Code Mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361, May 1998.

[4] R. Harbird, S. Hailes, and C. Mascolo. Adaptive Resource Discovery for Ubiquitous Computing. In *Proc. of the* 2$^{nd}$ *Int. Workshop on Middleware for Pervasive and Ad-Hoc Computing*, pages 155–160. ACM Press, Oct. 2004.

[5] M. Klein, B. Konig-Ries, and P. Obreiter. Lanes - A Lightweight Overlay for Service Discovery in Mobile Ad Hoc Networks. In *Proc. of the 3$^{rd}$ IEEE Workshop on Applications and Services in Wireless Networks (ASWN2003)*, Berne, Switzerland, July 2003.

[6] P. Kostkova and J. McCann. *Adaptive evolutionary information systems*, chapter Support for dynamic trading and runtime adaptability in mobile environments, pages 229–260. Idea Group Publishing, 2003.

[7] J. Liu and V. Issarny. QoS-aware Service Location in Mobile Ad Hoc Networks. In *Proc. of the 5$^{th}$ IEEE Int. Conf. on Mobile Data Management*, Berkeley, USA, Jan. 2004.

[8] M. .Nidd. Service Discovery in DEAPspace. *IEEE Presonal Communications*, pages 39–45, Aug. 2001.

[9] S. Waterhouse. JXTA Search: Distributed Search for Distributed Networks. http://search.jxta.org/.

[10] S. Zachariadis, C. Mascolo, and W. Emmerich. SATIN: A component model for mobile self-organisation. In *International Symposium on Distributed Objects and Applications (DOA)*, volume 3291, pages 1303–1321, Agia Napa, Cyprus, October 2004. LNCS, Springer.