Mobile Computing Middleware for Context-Aware Applications

Licia Capra
Dept. of Computer Science
University College London
Gower Street, London WC1E 6BT, UK
L.Capra@cs.ucl.ac.uk

1. RESEARCH PROBLEM

Mobile computing devices, such as palmtop computers, mobile phones and personal digital assistants have gained wide-spread popularity. These devices will increasingly be networked, thus enabling the construction of distributed applications that have to adapt to changes in context, such as variations in network bandwidth, exhaustion of battery power or reachability of services on other devices.

Even though devices and networking capabilities are becoming increasingly powerful, the design of mobile applications will continue to be constrained by physical limitations. Mobile devices will continue to be battery driven and users will be reluctant to carry heavy-weight devices. Wide-area networking capabilities will continue to be based on communication with basestations, with fluctuations in bandwidth depending on physical location. Therefore, in order to provide acceptable quality of service to their users, applications have to be *context-aware*, which requires them to adapt to context changes, such as exhaustion of battery power or reachability of services on other devices.

Traditional middleware [3] for fixed distributed systems cannot be used in this scenario, as the principle of transparency that has driven its design runs counter to the new degrees of awareness imposed by mobility. It is largely agreed that different forms of middleware are needed for the mobile setting.

Tuple space-based systems (e.g., Lime, TSpaces, JavaSpace) replace the synchronous communication paradigm supported by many traditional distributed systems with a decoupled and opportunistic style of communication: decoupled in the sense that computation proceeds even in presence of disconnections, opportunistic as it exploits connectivity whenever it becomes available. These forms of decoupling are important in a mobile setting, where the parties involved in communication change dynamically due to their migration or connectivity patterns. However, tuple space-based systems fail in supporting context awareness. Tuples are flat data structures which do not allow complex data organisation and therefore can hardly be exploited to provide a proper context representation to applications.

Context-aware computing is not a new computing paradigm [4]; since it was proposed a decade ago, many researchers have studied and developed systems that collect context information and adapt to changes. However, few contexts other than location have been proposed and used in actual applications. To enable applications to adapt to heterogeneous hosts and networks, as well as variations in the user's environment, location-awareness is not enough and richer context information must be collected and used.

The hypothesis of this thesis is that a new form of middleware can be developed that delivers better quality of service to mobile applications. This middleware should maintain in its internal data structures an updated representation of context information, and make it available to the above running applications, so that they can listen to changes in the context (i.e., *inspection* of the middleware), and influence the behaviour of the middleware accordingly (i.e., *adaptation* of the middleware).

2. THE REFLECTIVE MODEL

Our mobile computing middleware [1] is based on the principle of *reflection* [5] and *metadata.*

Metadata. Through metadata we obtain separation of concerns, that is, we distinguish what the middleware does from how the middleware does it. In particular, each application encodes in an *application profile* (i.e., in the middleware metadata) meta-information regarding *how* the middleware has to behave *when* executing in particular contexts. This meta-information can be split into two parts: *passive information* and *active information.*

Through passive information the application asks the middleware to listen to changes in the execution context and to react accordingly, independently of the task the application is performing at the moment. For example, the application may ask the middleware to disconnect when the bandwidth is fluctuating, or when the battery power is too low. We therefore establish an association between particular context configurations that depend on the value of one or more resources the middleware monitors, and policies that have to be applied.

Through active information the application creates associations between the services that the middleware delivers, the policies that have to be applied to deliver the services and the environmental circumstances that must hold in order for a policy to be applied. For example, different context configurations may require the service 'access data' to be delivered differently: a physical copy of data may be preferred when there is a lot of free space on the device, while a network reference may become necessary when the amount of available memory prevents us from creating a copy.

Application profiles are kept by the middleware. By interacting with the underlying network OS, the middleware maintains an updated representation of the context. Whenever a change in the context is detected, the passive part of the profile is consulted to find out which policy must be applied in accordance with the application needs. The active part of the profile is used instead each time the application directly asks the middleware to deliver a specific service. In both situations, middleware *learns* how to behave according to the information the application has passed down to it.

As both the needs of the user and the context may change quite frequently, we cannot assume that the application fixes its own profile once and for all at the time of installation. We therefore need to provide the middleware with an initial profile, and then grant the application the ability to dynamically access and modify it. Here is where reflection comes into play.

Reflection. By definition, reflection allows a program to dynamically access, reason about and alter its own interpretation. Mainly adopted in programming languages at the beginning, the principle of reflection has captured the attention of middleware researchers over the last few years. In particular, reflection has been used to add openness and flexibility into middleware platforms. However, the solutions proposed to date (reflective extensions of CORBA such as OpenORB and dynamicTAO) are too heavyweight to run on a portable device. Minimal CORBA specifications exist, as well as light-weight CORBA implementations; however, they do not suit the mobile setting as they support only a synchronous form of communication and do not provide reflective extensions of the basic mechanisms.

In our model, applications use a reflective API provided by the middleware to access their own profile, so that changes in this information immediately reflect into changes in the middleware behaviour. Application profiles are written by application designers and managed by the underlying middleware, that is, there must be an agreement between the two parts about the representation of the profile. We chose to model this information using the eXtensible Markup Language (XML), as it supports a representation of information that is both easily manipulatable by machines and readily understandable by humans. In our scenario, the middleware defines the *grammar*, that is the rules that must be followed to write profiles, in an XML Schema; the application designer then encodes the profile in an XML document that is a valid *instance* of the grammar. Every change done to the profile must comply with the grammar and this check can be easily performed using standard XML parsers.

3. RESEARCH AGENDA

The main contribution of this research is an investigation of the underlying principles of mobile computing middleware, and in particular of reflection as a means to allow applications to dynamically inspect and adapt middleware behaviour according to the current execution context. Our plans for the future are listed below.

Dynamic Conflict Resolution. By changing, through reflection, the meta-information contained in application profiles, application designers can dynamically influence the middleware behaviour. While doing so, however, designers can make mistakes and create profiles that contain ambiguities, contradictions and other logical inconsistencies. We refer to these inconsistencies as *conflicts*. We have designed, and are currently formalising, a microeconomic approach for *conflict resolution* that relies on second-price sealed-bid auctions, sometimes called Vickrey auctions. Our approach treats a mobile distributed system as an *economy*, where a scarce set of *goods* must be allocated to a set of *consumers*. Goods represent resources, such as processing power, memory, battery power, bandwidth, and the like, while consumers are applications seeking to achieve their own goals, by getting the resources they want and maximising their individual utility. Our conflict resolution mechanism is: *simple* (only a low computation and communication overhead is imposed), *customisable* (the application can influence the result of the conflict resolution mechanism), and *stable* (adhering to the mechanism gives each application the highest chance to have the conflict resolved in its favour).

Mobile Code Techniques. Reflection enables adaptability and flexibility only in those conditions that the middleware designers have considered likely to be unstable at design time. We plan to integrate our model with mobile code techniques [2] so that the middleware can dynamically adapt its behaviour to unforeseen situations by downloading newly delivered protocols either from a service provider or from other peers in reach which use the same behaviour. Applications can use reflection to select, for example, from where to download protocols based on application-specific information (e.g., trusted hosts, quality of service, etc.).

Evaluation. A first prototype that implements the reflective mechanism has been completed. In order to prove the usefulness of our model in developing context-aware applications, and to evaluate its performances in a mobile setting, we plan to develop mobile applications on top of our reflective middleware and run tests on a set of mobile devices.

4. REFERENCES

[1] L. Capra, W. Emmerich, and C. Mascolo. Reflective Middleware Solutions for Context-Aware Applications. In *Proc. of REFLECTION 2001. The Third International Conference on Metalevel Architectures and Separation of Crosscutting Concerns*, volume 2192 of *LNCS*, pages 126–133, Kyoto, Japan, Sept. 2001.

[2] L. Capra, C. Mascolo, S. Zachariadis, and W. Emmerich. Towards a Mobile Computing Middleware: a Synergy of Reflection and Mobile Code Techniques. In *In Proc. of the 8th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'2001)*, pages 148–154, Bologna, Italy, Oct. 2001.

[3] W. Emmerich. Software Engineering and Middleware: A Roadmap. In *The Future of Software Engineering - 22^{nd} Int. Conf. on Software Engineering (ICSE2000)*, pages 117–129. ACM Press, May 2000.

[4] B. Schilit, N. Adams, and R. Want. Context-Aware Computing Applications. In *Proc. of the Workshop on Mobile Computing Systems and Applications*, pages 85–90, Santa Cruz, CA, Dec. 1994.

[5] B. Smith. Reflection and Semantics in a Procedural Programming Language. Phd thesis, MIT, Jan. 1982.