MobJeX: A Declaratively Configurable Java Based Framework for Resource Aware Object Mobility

Caspar Ryan[1], Stephen Perry[1]

[1] RMIT University, School of CS & IT
PO Box 71, Bundoora
Melbourne , Australia, 3083
{caspar,sperry}@cs.rmit.edu.au

Abstract. MobJeX (Mobile Java Objects) is a Java based resource aware adaptive code mobility framework that is currently under development. MobJeX differs from its predecessors such as Sumatra [1] and FarGo [2, 3] by providing support for the transparent distribution of application code to client devices in a heterogenous Next Generation Internet (NGI) environment [4]. This paper briefly discusses the architecture of MobJeX.

1 Declarative Rule Based Application Configuration

MobJeX uses an XML based application deployment descriptor to specify rule-based configuration for object grouping and migration policies. This strategy was chosen to provide greater transparency, and less development effort, than a script-based approach. Although the declarative approach cannot ultimately match the customisability of scripting, it can provide a significantly greater amount of system support and automation. As such, the developer can write applications with a greater level of transparency, with the deployer tuning the system by specifying application specific policy rules for object grouping and migration.

2 Dynamic Object Mobility

MobJeX treats object mobility as a dynamic property, which can vary depending upon the state of the computational environment, as reported by the resource monitor (see section 4). Object mobility rules in the deployment descriptor are specified hierarchically, ranging from abstract application level rules to specialised object level rules. As such, a deployer need only specify application level rules in order to facilitate dynamic object mobility and client/server adaptation in a MobJeX environment. Optimising the application for a specific client is accomplished by specifying per object deployment rules and specific object grouping strategies. Additionally, further optimisation can be achieved by explicitly specifying object groups (by naming the objects comprising a group) and co-location constraints (via declarative resource requirement rules). The use of object mobility rules echoes the hard-coded relocation semantics of FarGo's *complet* approach with less developer effort and without the need to recompile code when adaptation policies change.

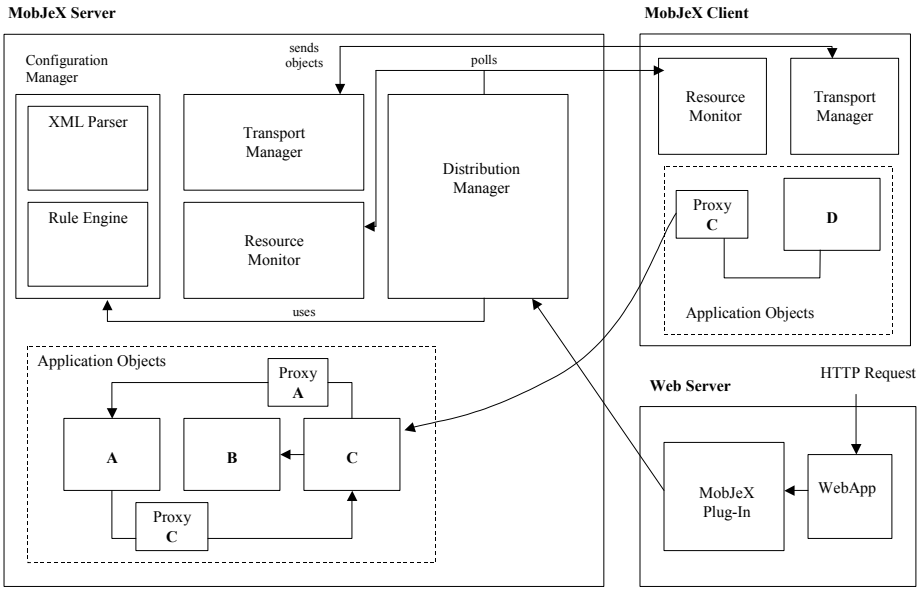

Fig. 1. Overview of MobJeX Architecture

3 Pluggable Resource Monitoring

Like Sumatra, MobJeX provides a resource monitoring API using familiar JavaBean event handling semantics. In order for a device to participate as a client in the MobJeX environment, it must have a resource monitoring implementation that can, at the minimum, report available memory. Although such limited reporting would affect performance, it does enable individual objects or object groups of specified memory requirements to be migrated and executed. Nevertheless, client specific implementations of the MobJeX resource monitoring API should, at a minimum, aim to report basic memory, networking, and CPU information. The authors have developed an initial resource monitoring implementation based on a Java Native Interface (JNI) to the Microsoft Windows performance API.

4 MobJeX Architecture

MobJeX uses a proxy-based object tracking strategy, which is comparable to FarGo, but operates in a simpler client/server fashion. This implementation decision was made primarily to simplify object communication, and more readily facilitate dynamic object grouping and mobility, but has the incidental benefit of being a more secure and trusted model for interaction in an NGI environment. Furthermore, our scheme uses a simpler client-side transport manager that is more suitable for use on limited devices. Fig. 1 depicts the high level architecture of MobJeX, the main components of which are the server based *configuration manager* and *distribution manager,* and client and server *resource monitors* and *transport managers*. Note that distribution decisions, based on information received from the configuration manager, are made on

the server. Nevertheless, clients still require a lightweight transport manager, which facilitates the physical mobility and proxy resolution of migrated objects, as well as a client specific resource monitor.

Application objects that meet the following criteria are accessed via an object proxy that either refers directly to the local object, or in the case of remote objects, knows their location, and provides the means to access them remotely. The circumstances in which proxies are used are: 1) If an object can move, all objects it refers to are proxied, excepting the case where explicit co-locality constraints are present. 2) If a non-moveable object refers to objects that can move, it must also access those objects via a proxy, so that they can be accessed upon migration. Note that the moveability of an object is determined by evaluating the mobility rules of the deployment descriptor via the configuration manager. Furthermore, it is possible to declare at the application level that objects either are, or are not, moveable by default.

5 Example

A typical example of a MobJeX enabled client could involve an initial HTTP request asking for a Java Swing based user interface object. Once delivered to the client and executed, this object would interact with other application objects, either locally or remotely, according to the specified application requirements and adaptation policies. A non-MobJeX enabled client would also initiate its interaction via the web front end, but unlike its MobJeX counterpart, would continue to do so for the lifetime of the session. In this case, all processing would occur on the server, with responses taking the form of a standard mark-up based user interface delivered via standard HTTP responses. In terms of underlying protocols, MobJeX is currently using Java RMI as the underlying transport mechanism, necessitating firewall configuration and the use of Java specific security mechanisms. Currently under consideration is whether an alternative such as SOAP-RPC can provide a more flexible and secure approach.

References

1. Acharya, A., M. Ranganathan, and J. Saltz, *Sumatra: A Language for Resource-aware Mobile Programs*, in *Mobile Object Systems: Towards the Programmable Internet*, C. Tschudin, Editor. 1997, Springer-Verlag: Heidelberg, Germany. p. 111-130.
2. Holder, O., I. Ben-Shaul, and H. Gazit, *System Support for Dynamic Layout of Distributed Applications*. 1998, Techinon - Israel Institute of Technology. p. 163 - 173.
3. Holder, O., I. Ben-Shaul, and H. Gazit. *Dynamic Layout of Distributed Applications in FarGo*. in *21st Int'l Conf. Software Engineering (ICSE'99)*. 1999: ACM Press.
4. Moyer, S. and A. Umar, *The Impact of Network Convergence on Telecommunications Software.* IEEE Communications, 2001. January: p. 78-84.