Design Patterns for Device Capability Discovery and Dynamic Reconfiguration in Adaptive Network Systems

Karen Lawson† and Professor Anthony Finkelstein‡

† University College London/Kodak Limited, ‡ University College London -Dept of Computer Science

Abstract: This paper explores the design of a subsystem for device capability discovery in mobile systems. The work detailed within is motivated by the importance of such functionality in a web services-based adaptive network system. The principle requirements for the proposed subsystem are outlined and the concept of dynamic device reconfiguration is introduced as a key constituent of such a system. Through the use of design patterns, a simplified model of the subsystem is diagrammed. A discussion of the results from the analysis and design effort is also presented.

1 Introduction.

Mobility and multi-device handling are critical to emerging adaptive network systems. Embedded systems and pervasive computing continue to expand the ever-changing landscape of devices that systems application service providers must address. Yet realistic and practical ways for dealing with heterogeneous device proliferation are slow to emerge. In response, most software engineers constrain their deployment platform support to a limited subset of the average consumer's preferred portfolio of devices. A natural extension of this device proliferation is the fundamental need to programmatically determine, quickly and effectively, the capability and context of the device requiring services.. Regardless of the developer's chosen platform, application level services will need to "converse" with the many different device platforms and adapt to their unique characteristics.

In a practical sense, evaluating and coding for all possible capabilities of the potential consumer devices can overwhelm the resources on a systems development effort. In this paper, we will review the requirements for automated discovery of device capability in a consumer-based application. In Section 2, we will outline the key aspects of dynamic device reconfiguration relevant to our work. The requirements of such a system and the recommendation for the appropriate design patterns are covered in Sections 3 and 4. Finally, in Section 5, the relevant work and future direction for the project is detailed.

2. Capability Discovery and Dynamic Reconfiguration

Mobility-centric computing and multiple device usage continue to expand at a significant rate. The concept of one device that can satisfy all requirements in all modalities appears to be unlikely and as such, software developers will need to have some mechanism to deal with the expanding array of network interface devices. Devices like desktops, laptops, PDAs, digital cameras, GPS and even coffee cups can all be networked and may be used to access a given application service [1]. It is both logical and widely published that in order to provide the user with an expected level of service, an application should have some understanding of the level of device current capabilities.

The simple example of sending image files to a device with no appropriate viewer demonstrates the need. The user would be greatly disappointed after waiting for the transmission to complete to find that the file format is not supported on their interface device. Reasonable quality of service guarantees depend on some capability knowledge. The detailed nature of quality of service constructs and architectures are not explicitly covered in this paper as there are many excellent references on the topic elsewhere.

Until recently, it has not been essential to understand the detailed capability of the interface device. The application or its results were pushed to the client in a standard or lowest common denominator format, and it was the responsibility of the user to insure that they had the sufficient platform to support the application.

Components of capability discovery are:

- An understanding of the current device capability
- The ability of the device to reconfigure its internal state without application interruption [2]
- An assessment of the capability enhancement potential or some idea of the ability for a device to accept a certain level of modification

In our work, we have determined these to be the key components necessary for the implementation of device capability discovery in an adaptive network system. Capability discovery is more than just a static list of device characteristics, but includes the assessment of the ability to complete the service request. The current capability

state then is the combination of both relevant static device characteristics coupled with the set of transient device states. This concept of possible states is the subject of future work using dynamic systems theory principles applied to the problem domain. The second key feature in this system is the extent to which the device has the ability to reconfigure itself without interruption of service. This is a complex issue that involves both the deployment of software capability required and the operational details of the device itself. The final feature of interest is the ability of the system to explicitly determine or infer, from other information extracted, the ability of the device to meet the capability needs necessary to complete the request.

Given these elements, we sought to understand and design such a device discovery capability subsystem using software architecture and design patterns instead of a custom application approach. This objective required a thorough review of the diverse concepts in software architecture for mobility, a review and evaluation of the appropriate design patterns in common use and the synthesis of this information into a practical representation, which a developer could utilize directly.

3. Architectural Requirements

A practical device capability subsystem requires decomposition according to the features mentioned in Section 2. We represent the high level requirements for each of these features which may ultimately be segmented into to implementation components.

Requirement 1: An "understanding" of the device capability

- generalized mechanism to reply to a request to report specific parameters of the device itself
- event Handling capability
- listening for capability requests
- interpreting the domain and data requested as in multi-input, single output processing,
- selecting an appropriate response of current capability; mapping output to meaningful assessment

Requirement 2: Extent of reconfiguration support

The ability of a specific device to support dynamic reconfiguration requires real-time validation as reconfiguration support may be added during a session in future systems [3]. Reconfiguration is not just a binary function, meaning it is not necessarily either on or off, but reconfiguration support is dependent on the function deployed and the extent to which it affects the global operating system environment. While this is a subset of assessing the device capability itself, in order for our architecture to evolve in this area, we require that it be handled separately.

- specific reconfiguration capability support assessment
- maintenance of state – device initiation determines the extent of reconfiguration support for general device – does this device support reconfiguration at all?
- event handling – a request requires reconfiguration support. If support is not available, flag this event for as input to capability enhancement assessment

Requirement 3: Capability Enhancement vs. Proxy Assessment

- Event handling – dequeue request for assessment
- Evaluate extent of capability assessment – extent to which the service request can be accommodated be adding deployable function - this is designed as a separate module the details of which are to be published separately –
- Evaluate the ability to proxy the request – is there a service available on another node or peer that can achieve the request reporting back to the initiator [4]
- Select method to achieve request based on resource cost, return a pointer to either capability enhancement **OR** proxy request address and function

These architectural requirements were then compared against design patterns for architectural, creational, behavioural, and structural objects in an effort to determine the best set of patterns which assist with the implementation of these functions in our subsysytem [5].

4. Software Patterns for Device Capability Discovery

Software patterns are useful for providing a base of information on how to analyze, design and ultimately achieve functions in software development. We found this to be the case in this work. Taking the known functional and non-functional requirements, a review of architectural design languages and software design patterns found that the design patterns were actually more useful in indicating the design implications of this model of device capability subsystem.

The principle difficulty in this pattern analysis step was the wide variety of sources and opinions surrounding the use of design patterns and the low number of references that deal specifically with mobility-related issues in software design.

Applying the most commonly used patterns to our problem domain resulted in the diagram below:

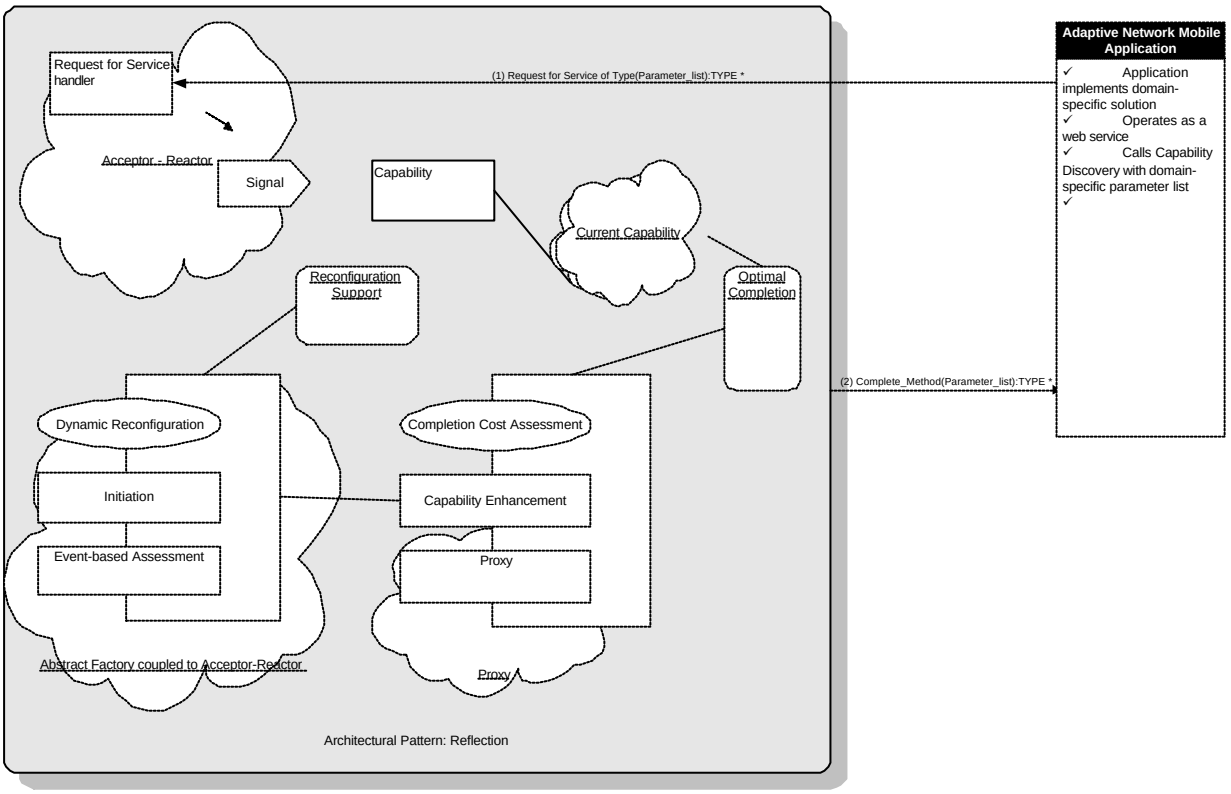

Figure 1: Representation of the Device Capability with Dynamic Reconfiguration Concept

Figure 1 outlines, in a hybrid representation style, the design of the device capability and dynamic reconfiguration subsystem we envision and includes the key design patterns that a developer would use to implement such constructs in code. While far from a detailed technical specification, the diagram was reviewed with several internal professional software engineers who found the design and its representation clear enough to start the process of determining the implementation. Future work in the form of peer code reviews will be done to validate the implementation as this informal validation is not sufficient to assess how the subsystem and the method of its design performed.

The selection of the design patterns was done based as follows [6]:

- Acceptor-Reactor – these two patterns form the principle infrastructure for subsystem. The Acceptor is a creational pattern used to establish our session, it provides the ability to modify the service at an application level without disturbing the connection mechanism
- Abstract Factory – this pattern is used to provide some interface to families of object without the need to specific concrete classes. The Abstract Factory will allow the subsystem to create complex objects which are exchangeable during session which will be important in the dynamic reconfiguration

- Proxy – this pattern performs as its name implies, it provide an alternative means to access an object, easily extending to remote invocation
- Reflection – this architectural pattern is utilized to design and implement the dynamic flexibility required. The Reflection pattern represents a system as foundation information and "metalevel" information allowing the developer to reflect changes to the foundation from the metalevel, a key need for reconfiguration.

There are clearly further patterns of use in the design and implementation of the subsystem. The State pattern for example, would be useful in the handling of the changes of capability state required to make the system work. For brevity however, we have left those patterns that are obvious out and focussed on those that add to the novelty of subsystem function. Also we do not discuss the details of these patterns, as there are many resources available for basic information on the pattern details. Specifically [7] was very useful in providing background for the analysis.

7. Conclusions.

This architectural design experiment revealed several key results that were unexpected. From a practical perspective, we determined that future research around inference of future device capability would indeed be possible perhaps even using the multi-input, single output (MISO) process commonly utilized in process control industries [8]. Additionally, we determined that understanding of current and even inferred capability state is not sufficient to handle the needs of network adaptive systems. It become clear that dynamic reconfiguration and more importantly the extent to which dynamic reconfiguration is supported is critical to deploying such systems. This handling of variable dynamic reconfiguration of devices is necessitated by the quality of service expectations inherent in adaptive network systems. Simply put, if quality of service is determined by the extent to which a user's request is satisfied in a given time with a given quality, then deploying functions to satisfy such a request cannot incur the destruction and re-initiation of the session.

Finally, we found that the use of the Proxy design pattern added an interesting element to our model of device capability discovery by permitting a resource cost evaluation between function deployment and proxying of the request to some other appropriate networked node or available service. While these structural objects are well known in the literature, the separation of this component from the other modules in the subsystem allows us to implement resource cost-awareness in the design. This resource awareness is an important theme in mobility computing as well as fixed peer-to-peer and GRID-style systems.

References.

[1] Hans-W.Gellersen, Albrecht Schmidt, and Michael Beigl. Adding Some Smartness to Devices and Everyday Things. December 2000. Monterey.

[2] Fabio Kon, Tomonori Yamane, Christoper Hess, Roy H.Campbell, and M.Dennis Mickunas, "Dynamic Resource Management and Automatic Configuration of Distributed Component Systems," *Proceedings of the 6th USENIX COOTS*, Jan.2001.

[3] Peyman Oreizy and Richard N.Taylor. On the Role of Software Architecture in Runtime System Reconfiguration. 6-5-1998. Annapolis, Maryland. Proceedings of the International Conference on Configurable Distributed Systems (ICCDS 4). May 1998.

[4] Davide Mandato, Andres Kasssler, Tomas Valladares, and Georg Neureiter. Concepts of Service Adaptation, Scalability and QoS handling on mobility enabled networks. 1st Global Summit . September 2001. Barcelona, Spain.

[5] Eyoun Eli Jacobsen, Bent Bruun Kristensen, and Palle Nowack. Characterising Patterns in Framework Development. http://www.mip.ou.dk/~jacobsen/publications.html . 1997.

[6] Douglas Schmidt and Paul Stephenson. Achieving Reuse Through Design Patterns. November 1994. Austin Texas. 3rd SIGS C++ World Conference.

[7] Allan Shalloway and James R.Trott, *Design Patterns Explained* Boston: Addison-Wesley, 2002.

[8] Gancho Vachkov and Toshio Fukuda, "Simplified Fuzzy Model Based Identification of Dynamical Systems," *International Journal of Fuzzy Systems*, vol. 2, no. 4, pp. 229-235, Dec.2002.