Active XML 을 이용한 코드 이동성의 빠른 확보기법

타로파 에마뉴엘, 이원종, 한탁돈
연세대학교 컴퓨터 과학과
{taropaemanuel[O], hantack}@kurene.yonsei.ac.kr
airtight@ yonsei.ac.kr

Fast Obtaining Weak Code Mobility Using Active XML

Emanuel Taropa[0] Won-Jong Lee, Tack-Don Han
Department of Computer Science, Yonsei University

Abstract

We show how to rapidly build a framework supporting weak code mobility by using Active XML [1]. Weak code mobility is important in mobile applications area by offering functionality at a low bandwidth cost and with little overhead imposed to the user peer architecture. Starting from already known principles involving code mobility, we give a new definition for the functionality transfer process and we materialize it in a working prototype.

1. Introduction

The ever increasing domain of mobile computing applications augments the pressure on device manufacturers and maintainers. So far the focus has been solely on adding functionality to existent applications and on providing better supporting hardware for them. With the current expansion of 3G mobile phones on the market [2], an increasing interest has developed in providing software reconfigurability for these devices. As the hardware platform has evolved enough to run complex operating systems, the next logical step is to provide coherent, inexpensive means of software updates for mobile phones.

In the following sections we chose to limit our analysis to the interaction between the 3G device owner and his carrier, modeled using the classic client-server paradigm, for reasons of clarity of explanations and lack of space. The conclusions we draw and the prototype system we designed, can be used in much more complex interaction patterns, such as ad-hoc networks or collaboration environments.

The previous research in code migration has explored the directions of both compiled and source code migration [3], [4], [5], [6], using sometimes standard technologies such as RMI [7] or defining new mobile scripting languages usually supported by agent frameworks [8]. Initially, the code migration problem was targeted to optimize processor usage in multiprocessor systems, by moving the entire process image and execution context from heavily loaded machines to more idle ones. We say that code migrated this way exhibits *strong mobility*. The interest for low overhead code migration appeared once with the development of mobile networks with limited resources on their nodes (i.e. sensor networks, mobile phones). Instead of moving an entire process image, involving *termination*, *transportation* and *creation* costs, the migration has been limited only to process code. We call this kind of code migration *weak mobility*.

Regardless of code migration's mobility type – *strong* or *weak*, and of it's implementation method, most of the previous work has focused on a "push" interaction paradigm. Typical instances are systems that have one or more code distribution nodes that choose the device onto which the code is to be deployed or systems based on agent frameworks, where the next point of deployment is chose by the agent program itself. In the following sections, we will present a framework for *weak code mobility* supporting both "push" and "poll" based interaction paradigms.

The rest of the paper is organized as follows: section 2 gives a functional specification of weak code mobility from a "push" and "poll" perspective. Section 3 describes a prototype for weak code migration, pointing out architectural details of Active XML. The paper concludes with Section 4 where conclusions are drawn and future work is presented.

2. Functional Specification

For providing the functional description of code migration, we choose a workflow related perspective. Therefore, we will focus our analysis on the following aspects:

1. *advertising and offering functionality* – easily achieved using web-services as interfaces to the code repository
2. *locating remotely available functionality* – expressed in our case by the source / object code of translation units (or programs)
3. *gathering that functionality locally* – bringing remotely available source code on the client's peer, compiling and executing it

We consider the above points to be the minimum requirements that a complete framework supporting weak code mobility needs to address. Further extensions can be made, with direct impact on efficiency of the migration process, on security and on correctness of functional composition. We tried to keep the definition of functionality minimal enough so that it can be extended for other code mobility frameworks as well, yet complete so that we have full expressive power when writing mobile code applications on our proposed framework. We will focus our next analysis on source code migration, observing that object code migration can be achieved in a similar way.

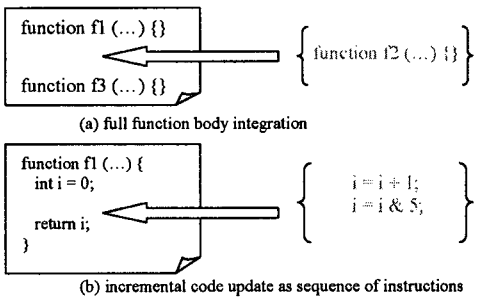

Figure 1: Functionality integration

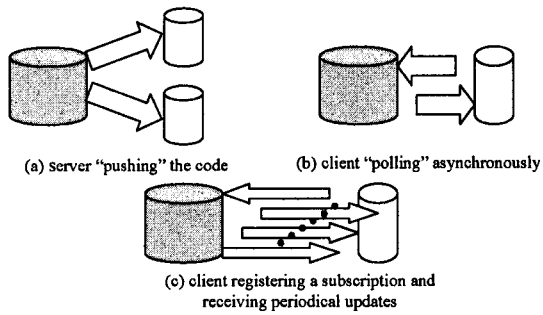

Figure 2: Migration scenarios

2.1 Defining Functionality

We understand by functionality *the smallest code unit that has a specified set of inputs and offers a specified set of outputs.*

This definition can be applied in two different ways, according to the type of code integration performed on the client's peer: *integration of fully functional units (functions in programming languages) and integration of code fragments.* These different types of code integration are illustrated in Figure 1.

2.2 Functionality Migration Scenarios

There are two ways in which code migration can be started: initiated by the server or by the client. Furthermore, the migration can happen only once (i.e. a sporadic update of an application, started by the phone company for all its subscribers), or periodically (i.e. some subscribers pay extra so that their mobile phone software is always kept up to date, by receiving periodic code patches).

Server initiated migration is implemented using a "push" interaction scenario. The server is the one who knows its subscribers and knows how to contact them (i.e. the phone company is sending all its subscribers periodic notices regarding their subscription status).

Client initiated migration requires a "poll" interaction scenario. The client decides to contact the server and requests specific functionality. A straightforward application of client initiated migration arises from the need of secure transmissions. Therefore, a company might offer encryption / decryption software source codes that the clients are able to download and use locally, for initiating a secure exchange of messages over a non-secure transmission media. The difference between classic migration scenarios and the ones supported by our framework is illustrated in Figure 2, where images a) and b) represent classic interaction schemes, also supported by our framework and c) represents a subscription interaction added in our framework.

2.3 Code Discovery

The intermediate layer between the clients and the servers is represented by indexing nodes. The architecture is similar to the UDDI [9] model in respect to the associations maintained on every node between code locations and descriptions. The difference is that these aggregator nodes are active and that they can initiate their own discovery algorithm for maintaining up to date their association tables. Furthermore, each of these nodes can send and receive code patches or full translation units from other nodes in the network.

The difference among source codes is made using added attributes that contain not only the *functional definition* (i.e. function signature) but also attributes related to *security issues* (i.e. allowed type of integration, digital signature). The problem that arises is given by the heterogeneity of possible definitions for the same function, but we feel that by using definition languages such as WSDL [10] a common standard can be obtained.

Discovery of the source code can also be achieved by *crawling the code servers* one by one periodically and indexing their results. This algorithm can be implemented either on the client side or on a modified UDDI proxy. The source code for the crawling algorithm is given in Figure 3

```
1. use initial server URL pool as seeds
2. while Q not empty do
3.     get next URL u from Q
4.     mark u as scanned
5.     fetch code repository for u
6.     for each repository entry e do
7.         if e is local entry for u then
8.             associate e with u
9.         else
10.            identify e's outgoing links
11.            for each outgoing link o do
12.            if o is not in Q then
13.                insert o in Q
14. goto2
```

Figure 3: Crawling Algorithm

2.4 Code Integration Policies

Once the remote code is locally available, we must define a modality for integrating it in our application. The integration process has the following steps: *file placement, compilation* and *scheduling for execution.*

The first step is controlled by the client's application and the security constraints imposed on the code: *public code* can be freely copied into a file and visualized and *private code* can also be compiled and executed in a safe environment, without user access or intervention. The benefits that come from compiling on the user's machine range from better optimization to decreasing the transport overhead (i.e. only increments of source code can be sent and not the entire object file).

3. System Description

Relying on the data integration framework provided by the Active XML language, we were able to build a system supporting the functional specification defined in the above section.

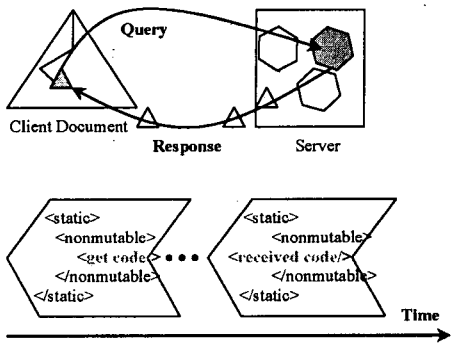

Figure 4: Code integration using Active XML

3.1 Active XML General View

Active XML is a language designed to achieve transparent data integration, in a peer to peer environment, using web-services. The data unit is defined as an Active XML document, which is an XML document with service calls embedded. The service calls are means for requesting data available on remote systems and integrate it into the originating document. Thus we observe that Active XML documents evolve with time, becoming more complete. The system supports both asynchronous and synchronous method invocation scenarios, making possible spontaneous and continuous data integration, suiting most applications needs.

A detailed description on web-service invocation and received data integration, as well as possible extensions that can be made to the language is given in [1].

An overview of data integration process is represented in Figure 4. We can observe how a document *requests* data and how the framework is responsible with *providing* the required data and with its integration in the client's document.

3.2 Implementing Migration Scenarios

The previously defined migration scenarios have a rapid implementation using the calling conventions and peer interaction policies supported by Active XML.

3.2.1 Server Initiated Migration

We assume that the server keeps a list of all its *subscribers* for code downloads. Therefore, the implementation is straightforward by sending the same update to all registered subscribers. When a subscription is made, we require the subscriber to provide the server's Active XML running peer with the endpoint information for receiving the updates. As most of this functionality is already implemented in the *service call element* in Active XML, we had just to add code migration attributes, concerning security issues and required software for compiling and executing the code.

3.2.2 Client Initiated Migration

The client can either contact directly a directory index service, containing the associations between source code specification and its location or can start its own discovery mechanism. For mobile systems with limited computing resources we recommend using a predefined proxy for getting the desired source code. Other systems can initiate their own discovery algorithm, having the benefit of obtaining the best possible path to access the remote source code. Our code migration system extends Active XML and provides a dual code discovery scheme.

3.3 Code Integration

Using Active XML's set of integration policies, we can either replace the node containing the service call by the received code, or we can add that code as its sibling, thus supporting incremental code migration [5].

4. Conclusions and Future Work

In this paper we demonstrated a new modality for rapidly achieving weak code mobility. We relied on the data integration framework offered by Active XML [1]. We specified the process of *weakly migrating* and *integrating* the code and we described a prototype system for code migration. Although in an initial stage, our system supports the migration and integration of complete translation units on different peers.

Currently the location of source code servers is assumed known, but in a future step we intend to implement the presented discovery algorithm. We will use these results for supporting alternative code composition scenarios for replicated translation units.

REFERENCES

[1] S. Abiteboul, O. Benjelloun, I. Manolescu, T. Milo, R. Weber *Active XML: A Data-Centric Perspective on Web Services*, Actes des 18emes journees Bases de Donnees Avancees, Hermes 2002

[2] GSM Association - *Growth of the Global Digital Mobile Market*, GSMA Statistics Q4 2004 http://www.gsmworld.com/news/statistics/index.shtml

[3] K. Bharat, L. Cardelli - *Migratory Applications*, Mobile Object Systems: Towards the Programmable Internet, pp. 131-149, Springer Verlag, 1997

[4] A. Tabatabai, G . Langdale, S. Lucco, R. Wahbe - *Efficient and Language-Independent Mobile Programs*, Conference on Programming Language Design and Implementation (PLDI), Proceedings of the ACM SIGPLAN, pp. 127-136, 1996

[5] W. Emmerich, C. Mascolo, A .Finkelstein - *Implementing Incremental Code Migration with XML*, Proceedings of the 22nd International Conference on Software Engineering, pp. 397-406, 2000

[6] Y. Bi, M. Hull, P. Nicholl – *An XML Approach for Legacy Code Reuse*, Jorunal of Systems and Software, Vol. 61, Issue 2, pp. 77-89, Elseiver Science Inc, 2002

[7] JavaSoft Sun Microsystems - *Java Remote Method Invocation Specification*, Revision 1.10, Java 2 SDK, Standard Edition, v 1.5.0, 2003

[8] Y. Aridor and M. Oshima – Infrastructure for Mobile Agents: Requirements and Design Proceeding of 2nd International Workshop on Mobile Agents (MA '98), Lecture Notes in Computer Science, Vol. 1477, pp. 38-49, Springer Verlag, 1998

[9] OASIS – *UDDI Version 2.03, Data Structure Reference* http://uddi.org/pubs/DataStructure-V2.03-Published-20020719.htm

[10] W3C – *Web Services Description Language (WSDL)* Version 1.1 http://www.w3.org/TR/wsdl20/