Impact of Compression and Encryption Methods for Software Architectures in Data-Oriented Mobile Applications

Md. Ashrafur Rahaman and Michael Bauer
Department of Computer Science
The University of Western Ontario
London, Ontario, Canada

Abstract - Mobile applications have evolved to become extensions of enterprise applications and systems that often interact with remote data sources and databases. Two software architectural approaches have emerged as the primary ways to develop mobile applications accessing remote data sources: client-agent-server and client-intercept-server. Both make use of middleware with the main difference being in the way agent components are used. Our previous compared the performance of these architectural approaches under differing sizes of data retrieved and with and without caching at the middleware. In this paper we compare these architectures in similar data scenarios but looking at the impact of data compression and encryption on the performance of the two approaches. Statistical analysis shows that the client-agent-approach generally performs better, though not in all circumstances. The results of this research provide useful guidelines for developing mobile applications needing to connect to remote databases.

Keywords: mobile applications, software architectures, enterprise data sources, performance.

1 Introduction

With the popularization of mobile computing and the emergence of a variety of mobile devices, developers face a number of challenges besides concerns of limited power: the heterogeneity of devices [1], the occasional intermittent communication in a mobile environment, and device limitations, such as: limited working memory, limited storage, limited processing power, and small screen. This means that developing large scale mobile applications which can connect to remote data sources or databases through wireless connections with high computational business logic must take into account these limitations. Support for complex or extensive applications in mobile phones which make use of data from remote sources or need remote computing capabilities still require servers and/or middleware.

Popular mobile relational databases, like IBM's DB2 Everywhere 1.0, Oracle Lite, Sybase's SQL etc., work on hand held devices and can provide local data storage for relational data acquired from enterprise relational databases. The main constraints for such databases relate to the size of the devices’ memory and size of the program, as handheld devices have memory constraints [2]. Moreover, enterprise databases cannot be replaced by these mobile relational databases.

To support extensive applications in mobile phones that require retrieval of data from remote data sources, middleware is needed which has the capacity to deal with mobile agents and remote database servers, and can improve the transmission of data by implementing caching, pre-fetching, data prioritizing and data compression techniques. Similarly, some mobile applications require data encryption, such as medically-related applications involving patient data. Mobile applications access the middleware through an API to make the communication between mobile agent and middleware transparent.

Various software models have been proposed for addressing the development of mobile applications that interact and utilize enterprise data sources; these include the client-server (C/S), client-agent-server (C/A/S), client-intercept-server (C/I/S), peer to peer, and mobile agent models. Spyrou et al. [3] qualitatively and quantitatively analyzed a set of software models built on the client/server model for mobile agents accessing a Web server. They argued that the C/A/S and C/I/S models were most appropriate and they compared the C/A/S and C/I/S models in the context of browsers and web servers for a wired network. Based on their results, the C/A/S model requires considerably less time than any other client/server model. Though the C/I/S model lacks in performance due to the presence of a client-side agent, it supports compatibility, since it can be built on top of existing applications. According to the researchers, the C/I/S model provides more flexibility than C/A/S and that should have been translated to better performance; their results show that the C/I/S model performs better for heavy-weight clients with large computational power.

Their work focuses on browser-based interactions. They do not consider mobile applications which may need to access data sources within one or more enterprises. The submission of queries and retrieval of results introduces different challenges in the movement of data and related processing, e.g. for security purposes. Our previous research [4] examined both the C/A/S and C/I/S models in the context of mobile applications where there is a need to access data sources or databases that are in remote locations. We compared the performance of the C/A/S and C/I/S software architectures for different size data queries and databases and considered the impact of using middleware caching. We
In Section 3, we briefly describe the client-agent-server and architectural approaches for supporting mobile applications. In Section 2, we review some of the existing literature on models and techniques or combination of technologies to use for mobile applications. Our experiments are performed in real life scenarios instead of a laboratory setup. The experimental results provide guidelines to the mobile application developers about the software decomposition or combination of technologies to use for data transmission for mobile applications.

The remainder of this paper is organized as follows: In Section 2, we review some of the existing literature on architectural approaches for supporting mobile applications. In Section 3, we briefly describe the client-agent-server and client-intercept-server software architectures. In Section 4 we describe our implementation and Section 5 presents a summary of the experiments and analyses. Section 6 provides a discussion of the results and some future directions.

2 Middleware for Mobile Applications

A number of different research efforts during the last decade that have focused on middleware and data transmission techniques for mobile applications. The work of Spyrou et al. [3] was discussed above. Capra and Mascolo [5,6] identify requirements for middleware that supports mobility. They suggest the use of a reflective middleware [7] in a mobile computing environment to solve the problem of losing network connectivity during the movement of the mobile devices. Campbell et al. [8] propose a middleware, named Mobiware, to support QoS in multimedia applications. The platform uses adaptive algorithms to support QoS controlled mobility. Bellavista et al. [9] present a middleware for mobile users, also based on mobile agents, to keep services running regardless of a user’s mobility. The middleware provides services to support a virtual environment, virtual terminal, and resource manager.

Chan and Chuang [10] propose MobiPADS, a middleware that uses information from the physical environment to perform self-configuration. It allows dynamic adaptation in both the application and the middleware itself. MobiPADS has two parts: a client at a mobile device attached to the Internet through wireless or cellular networks and a server at the wired network. This extended server-client application is designed to support multiple MobiPADS clients and is responsible for most of the optimization computation. MobiPADS collects metrics about the environment such as bandwidth, latency, and processor usage, and notifies the applications that use those data.

Middleware for multi-client and multi-server mobile applications based on the C/A/S model discussed in [11], where the authors addressed the problems of the heterogeneity of devices in such environments. The proposed middleware provides a communication API to develop applications for mobile environments and allows applications to divide their work amongst server and client sides. The middleware enables applications to exchange messages transparently between the application in mobile device and the middleware, and independently of a specific communication protocol. The researchers did not quantitatively compare the performance of the C/A/S and C/I/S models for their applications.

Caching and pre-fetching to improve a mobile application’s data transmission efficiency have been explored by several researchers. Gupta et al. [12] concentrated on the aspects of data management over mobile ad-hoc networks and proposed to estimate the global distribution and then predict and cache the most popular data in the hope of being able to provide it to other devices. Similarly, Yin and Cao [13] propose a cooperative caching scheme for mobile ad hoc networks using caching of popular data so that the availability in mobile ad hoc networks is increased. According to their simulation results, the proposed schemes can significantly improve the performance in terms of query delay and message complexity when compared to other caching schemes. Cheluvanaraj et al. [14] propose anticipatory retrieval of data. Caching is done asynchronously in the background during times of high bandwidth. They propose algorithms to assess the relevance of the data and then prioritize data downloads to cloud storage. The model provides better performance, adapts to varying bandwidth, and pre-fetches data from cloud with better accuracy and relatively little overhead.

Data transmission in the wireless network can be vulnerable to security attacks and, thus, ensuring data security is an important concern in some situations. Huang et al [15] introduced a security model based on message digests, encryption and decryption technology to access remote data securely. Researchers implemented this security model in mobile-agent architecture with large number of remote data processing tasks. This research did not explicitly examine the performance impact of using encryption for delivery of data to mobile applications.

Our research focuses specifically on assessing the performance of the C/A/S model and C/I/S model on different sized query results as well as investigating the impact of compression and encryption. Our main goal is to analyze the performance of the architectures for mobile applications that require access to and retrieval of data from remote enterprise databases.

3 Software Architectures

Mobile applications are normally structured as multi-layered applications consisting of UI, business, and data layers. A developer may choose to develop a thin Web-based
client or a rich client. In the case of a rich client, the business and data services layers are likely to be located on the device itself. On the other hand, the business and data layers will be located on the server for a thin client. Figure 1 illustrates a common rich client mobile application architecture with components grouped by areas of concern [16].

In our research, we focus on the client-agent-server (C/A/S) and client-intercept-server (C/I/S) models because we are retrieving data from remote enterprise databases or data sources and these two models seems to be the best fit for those tasks based on previous research. Moreover, our review of middleware based on mobile browsers says that most of the mobile browser engines were developed using either C/A/S or C/I/S model to retrieve from websites.

The C/A/S architecture is a popular extension of the client-server model, providing a three-tier architecture (Fig. 1). Here, any communication goes through the mobile agent. On one hand, the agent acts as a mobile host. As well, the agent is attached to a remote database or data source and any client’s request and server’s response is communicated through the agent. In this scenario, a mobile host is associated with as many agents as the services it needs to access. Agents split the interaction between mobile clients and fixed servers into two parts: between the client and the agent, and between the agent and the server.

The C/A/S model has several advantages. It alleviates some of the impact of the limited bandwidth and poor reliability of wireless links by constantly maintaining the client’s presence on the network via the agents. The agent splits the interaction between the mobile client and fixed servers into two parts, one between the client and the agent and one between the agent and the server. Data transmission can be optimized in the middleware so the QoS of data transmission improves with lower cost computation in the middleware or agent. A security wrapper in the middleware can provide data security over the wireless network.

Though the client-agent-server model offers number of advantages, it fails to sustain the current computation at the mobile client during periods of disconnection. In addition, the agent can directly optimize only data transmission over the wireless link from the fixed network to the mobile client but not in the opposite direction. In our research, we focus on an enterprise database, which is located at a remote server and connected to the middleware through the Internet.

4 Implementation

To evaluate the different models, we developed: two remote databases, a middleware API, a mobile agent API and mobile applications for the two different domains. The middleware API provides a web service which returns data in XML format, so that any mobile application can retrieve the data and use it. The API has the flexibility to retrieve data from databases through the middleware and make use of caching, compression, encryption or any combination of these technologies. The Mobile Agent API was developed using J2ME and provides functionalities for retrieving cache data, decompressing data, and decrypting data coming from the middleware API.

In the data transmission lifecycle, the mobile application sends a request to the middleware; the middleware retrieves data from the cache or executes the query on the remote database and processes the data; after processing the middleware returns data to the mobile application. Finally, the mobile application processes the data returned from the middleware and displays it. Figure 3 presents a high level overview of the data transmission lifecycle.

The C/I/S model proposes the deployment of an agent that will run at the mobile device along with an agent that will run in the server side or middleware (Fig. 2). This client-side agent intercepts the client’s requests and together with the server-side agent performs optimizations to reduce data transmission over the wireless link, improve data availability and sustain the mobile computation. From the point of view of the client, the client-side agent appears as the local server proxy that is co-resident with the client. Since the pair of agents is virtually inserted in the data path between the client and the server, the model is also called C/I/S instead of C/A/S.

Figure 1: Client-Agent-Server (C/A/S) model with remote database.

Figure 2: Client-Intercept-Server (C/I/S) model with remote database.

Figure 3: High level overview of the system.

In the C/A/S model, the mobile application sends requests to the middleware and the middleware processes the data based on the requests from the mobile application. If the mobile application requests that data be cached, the middleware searches for the data in the cache. If the data is found, the middleware returns it back to the mobile application. If the data is not in the cache, the middleware retrieves the data from the database, caches it and sends it...
back to the mobile client. After getting the data from the middleware, the mobile application processes the XML formatted data from the middleware, and then displays it on the mobile screen.

Figure 4 illustrates the request and response cycle for the C/A/S model. In case of C/I/S, the middleware and the database are same as in the C/A/S model. However, the mobile application does not directly communicate with the middleware or processes returning data from the middleware; rather, it calls the Intercept. Instead of calling web service, the mobile application calls methods of the mobile API which does any local computation and returns data to the application to display it. Figure 5 illustrates the request and response interaction of the C/I/S model.

**Databases:** We experimented with a database containing medical information of around 10,000 patients (with fictitious names and identifying information) The database contains several tables of hospital information, patient information, patient in hospital information, and the diagnosis results of patients. The database contains approximately ten thousand patient results. Using the patient data we can do different experiments.

**Middleware:** The middleware retrieves data from the remote databases and returns it back to the mobile application. The middleware is developed using C# programming language in an ASP.NET platform. We experiment with caching, compression and encryption within the middleware to understand their impact on data transmission in a wireless network. We report only on compression and encryption in this paper. The mobile software developer can use the API to specify whether data should be cached or compressed or encrypted or any combination of these techniques by the middleware, just by calling the web services provided by the API. The API provides a generic interface to a middleware platform and the mobile application developers can customize the use of these techniques to meet the need of their application to retrieve data.

**Compression:** Data compression involves encoding information using fewer bits than the original representation. Compression is useful because it helps reduce the consumption of resources such as data space or transmission capacity. As the bandwidth of wireless network is scarce, it may be advantageous to compress data to get the maximum out of the bandwidth. Mobile applications suffer from limited memory where data compression may help save memory and at the same time improve the speed of data transfer.

Lossless compression is mainly used for spreadsheets, text and executable program compression; on the other hand, lossy compression is mainly used for image, video, and audio compression. Lossless data compression is used in many applications such as the ZIP file format and in the UNIX tool gzip. Lossless compression is used in cases where it is important that the original and the decompressed data be identical, or where deviations from the original data could be deleterious. Typical examples are executable programs, text documents and source code.

In the middleware, we used gzip compression; it is based on the DEFLATE algorithm, which is a combination of Lempel-Ziv (LZ77) and Huffman coding. Gzip contains a 10-byte header, containing a magic number, a version number and a time stamp; optional extra headers, such as the original file name; a body, containing a DEFLATE-compressed payload; and an 8-byte footer, containing a CRC-32 checksum and the length of the original uncompressed data. To implement gzip compression in the middleware, we used SharpZipLib [17], an ASP.NET compression library that supports Zip files using both stored and deflate compression methods.

**Encryption:** In the case of enterprise databases, it is often necessary to move data over networks – to other sites or to remote desktop and mobile applications. Data transmission across networks, particularly public networks, creates potential security problems [17]. Given the importance of data security, we have implemented encryption in the middleware so data transmission in the wireless network can become secured.

We encrypt the data to Base64 data format before transmitting it to wireless network. Instead of using a real encryption algorithm for our experiments, we used Base64 encoding to emulate encryption processing; in reality this encoding scheme is much less complex than a real encryption scheme. Base64 is a group of encoding schemes that represent binary data in an ASCII string format by translating it into a radix-64 representation. Base64 encoding schemes are commonly used when there is a need to encode binary data.
that needs be stored and transferred over media that are designed to deal with textual data. This is to ensure that the data remains intact without modification during transport. Base64 is commonly used in a number of applications including email via MIME, and storing complex data in XML.

5 Experiments and Results

Our experimental environment consisted of the following components: A remote database server hosting two databases; A middleware server (providing web service); Two mobile applications (C/A/S application and C/I/S application). Table I summarizes the tools used for the development of experimental systems. The following provides an overview of the experimental environment in more detail.

### Table I. List of development tools for the experimental environment.

<table>
<thead>
<tr>
<th>Component</th>
<th>Development Tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remote database Server 1</td>
<td>MySQL Database</td>
</tr>
<tr>
<td>Remote Database Server 2</td>
<td>MySQL Database</td>
</tr>
<tr>
<td>Middleware API</td>
<td>ASP.NET web service</td>
</tr>
<tr>
<td>Mobile Application C/A/S</td>
<td>J2ME</td>
</tr>
<tr>
<td>Mobile Application C/I/S</td>
<td>J2ME</td>
</tr>
</tbody>
</table>

**Database Server:** The databases are hosted on a remote server (www.godaddy.com), accessible over the Internet through a public IP address. It runs Linux and MySQL server, version 5.0. The middleware communicates with the remote database through the public IP address through the Internet.

**Middleware Server:** The middleware was developed using ASP.NET web service and C#. The middleware server runs Internet Information Services (IIS) 7 server. The configuration of the server is: Windows 7 32-bit operating system, Intel core duo 2.3 GHz processor and 4 GB memory. It is accessible through the Internet through a public IP address.

**Mobile Application:** Our mobile applications were implemented on a Nokia E72 smart phone; the configuration of the mobile phone is: Symbian OS 9.3, 600 MHz CPU, 250 MB Internal memory, and 128 MB RAM. The mobile application connects to the middleware services through wireless (WiFi). Testing was done from a home residence which had a 2Mbps dedicated wireless connection.

### 5.1 Basic Experiments with a Medical Database

We developed a Medical Information Application for the mobile device. Using this application, a user (e.g., doctor, nurse, etc.) can log in and search a patient database by last name or first name or any part of the name. The search result returns a list of patients along with their unique identity number, name, age, and sex. Selecting a patient from the list returned, will cause the application to search the database and return the medical history and previous diagnosis results of the patient. Using the Medical Application, we carried out three experiments:

- **Experiment 1:** Basic experiment: compare the two architectures for two different “data” scenarios.
- **Experiment 2:** Same scenarios as basic experiment using compression.
- **Experiment 3:** Same scenarios as basic experiment using encryption.

For each experiment, the same retrievals and same steps were used:

- **Step 1:** User logs in using their user name and password; saved in the database along with permissions to check a patient’s medical information.
- **Step 2:** After successful login, the user can search for a patient by first name, last name, or any part of the name, for example: ‘Henry’, ‘John’, or ‘Jo’.
- **Step 3:** Based on the search key, data is retrieved from the remote database; basic information is displayed: Patient Identification Number, Name, Age and sex. The total time taken in milliseconds to retrieve data from database is recorded along with total number of records found and the size of the returned data in bytes.

In the basic experiment, the mobile application requests, through the middleware, data from the remote database; the middleware retrieves the data and sends it to the mobile application through the web service in XML format. The mobile application extracts the retrieved data from the XML formatted data and display it on the mobile screen. We execute the experiment for three different scenarios involving different size data results. The scenarios with the data sizes are presented in Table II.

### Table II. Scenarios and Data Sizes for Experiments.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Search key for patient search is ‘Henry’</th>
<th>Total patients found: 17, Data Size: ~2KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario</td>
<td>Search key for patient search is ‘Li’</td>
<td>Total patients found: 397, Data Size: ~45KB</td>
</tr>
<tr>
<td>Scenario</td>
<td>Search key for patient search is ‘T’</td>
<td>Total patients found: 1850, Data Size: ~230KB</td>
</tr>
</tbody>
</table>

### Table III. Experiment 1 Results.

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>C/A/S (SD)</th>
<th>C/I/S (SD)</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario 1</td>
<td>525.67 ms</td>
<td>577.67 ms</td>
<td>C/A/S little faster than C/I/S but negligible difference.</td>
</tr>
<tr>
<td>Scenario 2</td>
<td>856.00 ms</td>
<td>1073.00 ms</td>
<td>C/A/S faster than C/I/S.</td>
</tr>
<tr>
<td>Scenario 3</td>
<td>-----</td>
<td>-----</td>
<td>Failed, unable to complete.</td>
</tr>
</tbody>
</table>

We replicate each experiment three times. The experimental results and analyses are presented in Table III.

1 SD = Standard Deviation
The application on our test mobile phone could not handle the data of the Scenario 3 (returns around 230KB data from middleware) to display. We compared the performance of the two architectures for each of the scenarios using a t-test. For both Scenario 1 and Scenario 2, there is a significant difference between C/A/S and C/I/S at a 95% confidence level, that is, C/A/S performs better.

<table>
<thead>
<tr>
<th>Software Architecture (SA)</th>
<th>Scenarios (S)</th>
<th>SA+S %Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.94%</td>
<td>84.18%</td>
<td>3.36%</td>
</tr>
<tr>
<td>3.52%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Further analysis using an analysis of variance (ANOVA) was also performed and the percentage of variation explained by the factors is presented in Table IV. The Software Architectures (SA) explained 9% of the variation, the Scenarios (S) 84% and the interaction of these (SA+S) was 3%. The error also explained about 3% of the variation. Clearly, the different scenarios had the greatest impact on the variation in results while the variation attributable to the different architecture was minor.

**Summary of Experiment 1**: Data size has the greatest impact on transmission time, though C/A/S performs better than C/I/S.

### 5.2 Compression Experiments

In Experiment 2, the scenarios are the same as Experiment 1, but here, we compress data in the middleware. For this experiment, the results are mixed: the C/A/S model performs better than the C/I/S model for Scenario 1, but the C/I/S model performs better for Scenarios 2 and 3. Note that in this case the use of compression enabled Scenario 3 to work successfully, unlike the case without compression. The results for the compression experiment are shown in Table V.

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>C/A/S (SD)</th>
<th>C/I/S (SD)</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario 1</td>
<td>478.67 ms (21.08)</td>
<td>545.00 ms (45.92)</td>
<td>C/A/S little faster than C/I/S but negligible difference.</td>
</tr>
<tr>
<td>Scenario 2</td>
<td>875.67 ms (41.02)</td>
<td>817.67 ms (39.72)</td>
<td>C/I/S faster than C/A/S; difference is small.</td>
</tr>
<tr>
<td>Scenario 3</td>
<td>2365.00 ms (39.92)</td>
<td>2190.67 ms (63.36)</td>
<td>C/A/S faster than C/I/S; difference is small.</td>
</tr>
</tbody>
</table>

As with Experiment 1, a t-test at a 95% confidence level shows a significance difference between the two architectures for Scenario 1. However, for Scenarios 2 and 3, at a 95% level there was no significant difference between the two architectures.

### 5.3 Encryption Experiments

In Experiment 3, the scenarios are the same as before, but here, we encrypt data in the middleware. As with Experiment 1, at a 95% level the C/A/S model performs better than the C/I/S model for Scenarios 1 and 2; again, the data transfer in Scenario 3 failed. Encryption tends to increase data size, even for the simplistic method used in our experiments. The data in Scenarios 1 and 2 changed, respectively, from 2Kb to 3Kb and from 45Kb to 60Kb. The results for the encryption experiment are shown in Table VII.

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>C/A/S (SD)</th>
<th>C/I/S (SD)</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario 1</td>
<td>669.33 ms (15.95)</td>
<td>806.00 ms (81.07)</td>
<td>C/A/S better but negligible difference.</td>
</tr>
<tr>
<td>Scenario 2</td>
<td>1134.00 ms (46.81)</td>
<td>1236.00 ms (15.72)</td>
<td>C/A/S better but negligible difference.</td>
</tr>
<tr>
<td>Scenario 3</td>
<td>-----</td>
<td>-----</td>
<td>Failed; unable to complete.</td>
</tr>
</tbody>
</table>

As with Experiment 1, a t-test at a 95% confidence level shows a significance difference between the two architectures for Scenario 1. However, for Scenarios 2 and 3, at a 95% level there was no significant difference between the two architectures.

### 6 Discussions and Directions

We examined the performance of the C/A/S and C/I/S models because they had been identified as the most plausible models for data-oriented mobile applications by previous researchers though primarily in the context of browsers. One
objective of our research was to investigate whether one performed better than the other and for which scenarios and techniques. Based on our research results, we conclude that the C/A/S model generally performs better than the C/I/S model, though not in all situations. This is consistent with previous findings of Spyrou et. al [3] and others. However, an analysis of variance shows that the differences in execution times stems from the different scenarios (size of data retrieved by the application) and caching (when used).

Though the C/A/S application performs better than the C/I/S application in our experiments, the magnitude is still relatively small and based on our experiences in using both architectures for implementation of the application, there are other reasons to consider the C/I/S model:

- In the C/I/S model, the communication is through the Intercept API. In our experience, using the API makes mobile programming simpler and may be an advantage when developing application for heterogeneous mobile operating systems.
- Using the C/I/S model provides reusable code, which can help improve the quality of the product. So even if in some circumstances the C/I/S model quantitatively requires a little extra transaction time, it may qualitatively improve the system.

A number of future directions exist based on this work:

- Future work could look at extending the C/I/S model for client to utilize background threading in the intercept when retrieving or processing data from the middleware.
- Our experiments used a Nokia E72 smart phone which runs the Symbian Operating System. It would be useful to evaluate the APIs by porting them to other mobile operating systems and developing mobile applications. There are many new and different mobile devices, and testing the APIs with other types of smart phones tablets, etc would be useful.
- With other devices, it would be useful repeat some of these experiments to see if similar results can be obtained. Newer devices have more capabilities, faster processors, more memory so the absolute times may be faster or data sets larger. It would be interesting to see if the impact in performance of these techniques follows a similar pattern.

References


