Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 00 (2014) 000–000

Procedia Computer Science

www.elsevier.com/locate/procedia

International Symposium on Emerging Inter-networks, Communication and Mobility (EICM)

Generic Email Connector for Mobile Devices

Hamid Mcheick[a,*], Alexandre Poirrier[a], Jabril Abdelaziz[a], Abass Lakiss[b]

[a]*University of Quebec at Chiucoutimi, 555 Boul De l'Universite, Chicoutimi, G7H-2B1, Canada*
[b]*Lebanese Univerity, Lebanon*

Abstract

With today's technologies focus on mobility and portability, mobile networks suffer more and more from the increasing traffic load. This is due to the fact that millions of email servers are now prompting the network providers to notice their clients when emails are ready to be read using proprietary protocols like MS Exchange. This work improves our existing software connector model by giving the opportunity to use existing protocols such as IMAP, POP3 and so forth. Generic XML-based definition of both the client and the protocol of target system has been introduced. The implementation of a software solution has been done in order to illustrate the work follow of designing a connector based on the model.

Peer-review under responsibility of the Program Chairs of EICM-2014.

Keywords: Connector design; Mobile applications; Middleware

1. Introduction

Distributed Systems take advantage of the computing power in components that exist on several networked devices. Today's electronic device market is no more restricted to desktop computers, it includes many different devices such as tablets, smartphones and even cars. In terms of architecture, connector and component are two key concepts to understand logical organization of software [1]. In point of fact, every device has its own email client (clients) connecting to various email servers through various protocols (SMTP, POP3, IMAP). These protocols are used by Outlook, Gmail and many other email servers. The main goal is to be able to create a software solution which does not involve any extra work from the wireless network operator. However, this also makes the connectors in a key determinant of system's properties such as performance, scalability, reliability, security, and so forth; placing monolithic challenges on the way these connectors are to be designed and deployed. As a result, the predominant focus of researchers and practitioners has been to better design connectors to meet these challenges. Our previous work [2] proposed a Scenario-Based Software Model for designing connectors in distributed systems.

* Corresponding author. Tel.: +1-418-545-5011; fax: +1-418-545-5017.
E-mail address: hamid_mcheick@uqac.ca

Peer-review under responsibility of the Program Chairs of EICM-2014.

The model combined three prominent technologies: Aspect-Oriented Programming (AOP), Design Patterns, and Program Slicing to maintain the independency of components and to reduce the cost of maintenance when adding new connectors.

Following to same vision, in [3] we proposed a layered model to design connectors in distributed MVC architecture. The layered model enables developers to understand all design, implementation and maintenance aspects of connectors. Moreover, developers are guided through a process of high-level connector design, in which each layer is responsible for satisfying its functionalities and corresponding quality attributes.

In this paper, we aim to improve our generic connector model and to implement it as a framework for a "simulated push protocol" connector. Our contribution will, therefore, be a continuation of previous project and consists of adding the support for IMAP protocol and extra device carriers. The aimed enhancement implies finding a way to determine what protocol the connecting email client is using to interact with the email server, since so far the project only supports POP3 protocol. In addition, functionalities to insure interaction with IMAP email servers are implemented. Subsequently, a code adaptation is in order to guarantee a full transparency to the user.

The rest of this paper is organized into sections as follows: Section 2 describes related work. Then, Section 3, presents briefly a generic connector model and the different accomplished steps to improve existing connector model [4]. Study case and results are given in Section 4. Section 5 discusses these results and concludes this paper.

2. Related work

Usually, connectors are designed for target systems. The architecture of such connectors is based on either the APIs or the data type and schema the target system do support and use. This means that the designed connector is tightly integrated with its target system. Indeed, the use of an application-specific connector is the preferred integration method, and this, if one is available for the target system. Out there, many email connectors exist, but they are used for either automatically data parsing and extracting or to merge many different email accounts into one regardless of the used protocols. CommitCRM [5], named after its company, provides CommitCRM Email Connector used by customer services to file and sort emails into tickets for their support system. This acts more as a middleware than as a connector, in the sense that it does not connect different technologies to work together.

F-Response [6] develop and maintain an email connector incorporated with their software solutions. Like the first case, this one is not generic and does not serve any other purpose than to manage the different protocols and retrieve emails. In the market, one can see a lack of generic connectors that can be used for multiple kind of email server.

3. Generic Email Connector

Layered software architecture is typically adopted to distribute software into different layers to reduce software design complexity . Since the software architecture is distributed on separated layers, the corresponding functionalities and quality attributes are correspondingly separated into layers. By nature, connectors, in contrast to components, are explicit architectural entities that bind components together and act as mediators between them. This separation from components eases the isolation of interaction concerns (communication, coordination, conversion, facilitation) for the sake of reusability and evolvability of components. In case components are localized at different nodes of a distributed system, the cooperation between the distant components is determined by the semantics of the connectors, which connect them.

According to the properties of such connectors, we provide a description of connectors in distributed system:

"*Connectors link distributed components by using some formatted information according to the dependencies (relationships) between them.*"

In light of this description, and according to the architectural drivers [7], we proposed a layered model called "Connector Stack" which comprises three modules (e.g. layers): the Transport layer, the Dependencies layer, and the Presentation layer. This model easies the design and analysis of connectors (**Erreur ! Source du renvoi introuvable.**).

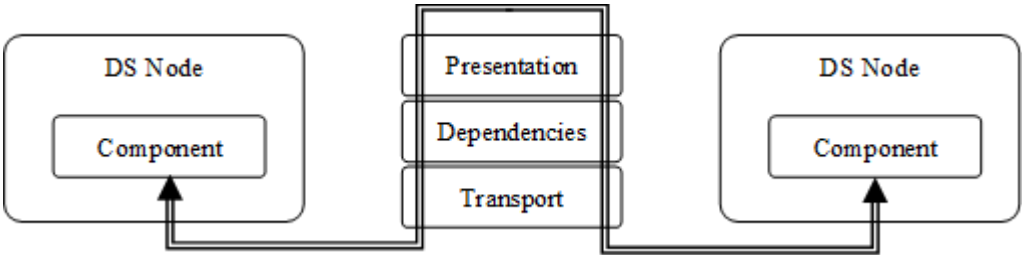

Fig. 1. Connector stack

The Transport layer, situated in the base of the connector design, is responsible for basic delivery of messages in the network. It is also used to support the dependencies or relationships between components. Typically, transport layer is implemented by network protocols (such as TCP/IP, HTTP protocol stacks) or messaging systems (WebSphere MQ/ Open Message Queue etc.). The Dependencies layer describes the dependencies of components in detail. To fulfill this goal, object design patterns or architectural patterns can be used to implement it. Most patterns can provide the loosely coupled dependency mechanism for the system, such as the Publish-Subscribe. The Presentation layer is responsible for building the transferred information depending on the type, format and the amount of information. Connector Stack provides a way to design and analyze connector by dividing the connector into separate layers. Each layer is an important part of connectors. For example, we implemented a connector in distributed system by choosing UDP/IP protocol, Publish-Subscribe architecture pattern and a simple text-formatted messages. There are some phases in the process to achieve the goals introduced for our contribution to this study:

- Design an XML template that will identify the email clients
- Modify the currently hardcoded protocols and scenarios to be generic and take real inputs
- Add support for another emails fetching protocol (IMAP in this case)
- Modify the Android testing device to use the proposed XML template
- Test both supported protocols using the Android test device

These steps have been validated through the following case study. This model has been already validated for a specific email server in a previous work [4]. For this work, this model is validated with different protocols (POP, IMAP). Next section shows the detail of these steps to dynamically configure these protocols and their corresponding email servers.

4. Case Study and Results

4.1. The XML template

The first step to board, to be able to support more than one protocol with the connector, is to be able to identify which protocol is used by the connecting client. It has been decided to use an XML file which the client sends to the connector. The XML file follows this DTD:

```
<!DOCTYPE Info
[
        <!ELEMENT Info (Email, Password, Protocol, ServerAdress, TimeRequested?)>
        <!ELEMENT Email (#PCDATA)>
        <!ELEMENT Password (#PCDATA)>
        <!ELEMEMT Protocol (#PCDATA)>
        <!ELEMENT ServerAdress (#PCDATA)>
        <!ELEMENT TimeRequested (#PCDATA)> <!-- Optionelle -->
   ]>
```

The connector is then built by the server especially for this client using the requested protocol. We are aware that this is not the best way to approach this issue. Since it means that to be able to use this connector, every existing email client would need to integrate this XML file, and therefore the obligation to change the source code. Finding a way to determine the connecting protocol without using an XML file is to be treated in future works.

4.2. Code Cleaning and Adding Support for the IMAP Protocol

Previously, the email server would not take any input to specify which protocol or even which email service would be used. This was specified as a hardcoded String in the server's source code. The user is now prompted to input the protocol used along with the email server's URL on the android application's login page. The user interface is explained in subsection 4.3.

The file *MailServer.java* has been modified to parse the XML String sent by the email client. As result, a *MailReaderBean* object is instantiated using the parsed information:

```
mrb = new MailReaderBean(
Protocol.getText().toLowerCase(),  providerAdress.getText() , mailAdress.getText(),
password.getText(), "inbox", IPAddress, LISTENING_PORT);
```

Ultimately, the connector would compare the part after the <<@>> against a database to find the correct provider's URL. For testing purposes, the URL was selected from a list box in the android application.

The *MailReaderBean* class had an extensive overhaul as well. First, it was restructured and extended to make it easy to add support for more protocols in the future.

```
if(protocol.compareTo("pop3") == 0)
{
        Properties pop3Props = new Properties();
        pop3Props.setProperty("mail.pop3s.port",  "995");

        session = Session.getInstance(pop3Props, null);
        Store store = session.getStore("pop3s");
        store.connect(host, 995, user, password);
        top = store.getFolder(rootName);

        listFolder(top, session, false);
} else {
        if(protocol.compareTo("imap") == 0)
        {
           Store store = session.getStore("imap");
           store.connect(host, user, password);
        }
}
```

4.3. Updating the Android testing application

The Android application as it was given was only used to create the email connector by sending the connection data. We decided to update it providing two ways: i) to be used for a more extensive testing of the connector and ii) to be easily extendable for more protocols and email service providers.

As illustrated in Fig. , there is no need to input the server's IP. In exchange, we added possibility to select the email service provider's URL and the protocol used. The system GUI given acts as a console and gives feedback regarding the application's status. For example, if everything goes well with the login on the server side, it will send

Fig. 2. User interface of Android device

Fig. 3. DialogBox of the message received by Android device

a <<Login approved!>> message that will be displayed in the system (Fig.).

To facilitate the extension of the testing application, the list of supported protocol is implemented as an XML file. Therefore, if one want to add support for any other protocol or email provider, a simple modification of this XML file may be needed to add any new protocol. In this example, POP3 and IMAP are added. Gmail and Yahoo can be added to existing email servers respectively, using <pop.gmail.com> and <pop.mail.yahoo.fr>.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string-array name="protocolArray">
      <item>POP3</item>
      <item>IMAP</item>
   </string-array>
   <string-array name="providerArray">
      <item>pop3.live.com</item>
      <item>imap-mail.outlook.com</item>
   </string-array>
    </resources>
```

5. Discussion, Conclusion and Future Works

This paper proposed a generic email connector for mobile devices. A case study, using Android, IMAP and POP protocols has been used as a proof of the proposed concept. This work came to improve the existing software connector model. The generic connector has been developed using different technologies in order to come up with extensible solution. In the realization of this model, Aspect-Oriented Programming has been used in our implementation. This approach along with design and architectural patterns are combined to resolve the problems of designing generic and extensible connector.

Acknowledgements

This work is supported by the Department of computer science at the University of Quebec at Chicoutimi (QC), Canada.

References

[1] Taylor RN, Medvidovic N, Dashofy E. Software Architecture: Foundations, Theory, and Practice. Wiley; 2009.

[2] Mcheick H, Qi Y, Mili H. Scenario-Based Software Architecture for Designing Connectors Framework in Distributed System. Int J Comput … 2011;8:32–41.

[3] Mcheick H, Qi Y. Dependency of components in MVC distributed architecture. 2011 24th Can. Conf. Electr. Comput. Eng., IEEE; 2011, p. 000691–4.

[4] Mcheick H. Combination of connectors with loosely coupled architecture based on aspect-oriented computing. 2012 Int. Conf. Commun. Inf. Technol., IEEE; 2012, p. 97–101.

[5] CommitCRM. CommitCRM 7.0 2014.

[6] F-Response. F-Response 5.0.3 2014.

[7] Bass L, Clements P, Kazman R. Software Architecture in Practice. Addison-Wesley Professional; 2003.