Proceedings of the International Multiconference on
Computer Science and Information Technology, pp. 47 – 57
ISSN 1896-7094

Utilization of Software Agents and Web Services as Transducers for Legacy Software; Case Study Based on an SMTP Server

Michal Oglodek
Faculty of Mathematics and Information Sciences
Warsaw University of Technology
Pl. Politechniki 1
00-661 Warsaw, Poland
moglodek@o2.pl
Maciej Gawinecki, Marcin Paprzycki
Systems Research Institute
Polish Academy of Science
ul. Newelska 6
01-447 Warsaw, Poland

Abstract. There exists a number of ways in which legacy software can be "wrapped" to become interoperable. Two of currently more popular of them are utilization of Web Services and software agents. The aim of this paper is to experimentally compare efficiency of JADE implemented agents, with Web Services, when used as transducers for an SMTP server.

Keywords: Web Services, Agent-based Computing, Transducers, Legacy Software, Simple Mail Transfer Protocol, Efficiency.

1 Introduction

One of interesting problems in software development is how to deal with legacy software developed using different technologies. Even if it sometimes good to keep and application "isolated," there is also a need for applications to communicate with each other (be *interoperable*). For instance, while companies may opt to develop their own software standards, there are at least two situations where communicating with "other" applications is necessary. First, in the case of company-to-company communication (e. g. when a wholesaler establishes direct link with a retailer). While this case may be relatively easy to solve by generating a few interfaces, the second scenario is more complicated. Nowadays, companies merge on regular basis, resulting in merging their independently created IT support systems, as most likely the IT system for the "new" company will not be created from scratch.

Let us also note that the question is not only how different (sub)systems can "talk to each other," but it is also important to consider effect that the attempted integration will have on exposing directly internal or external access to a given application. It is easy to envision, that security concerns (and resulting practical matters) need to be taken into consideration (e.g. an open port needed to access an application is likely to result in strict firewall policies). To illustrate this, in Figure 1 we present a problem

brought about by an external access to a Java Application. If the company firewall policy is very strict about open ports, the RMI (which is necessary for the external client to communicate with the application) is not easily accessible from the outside.

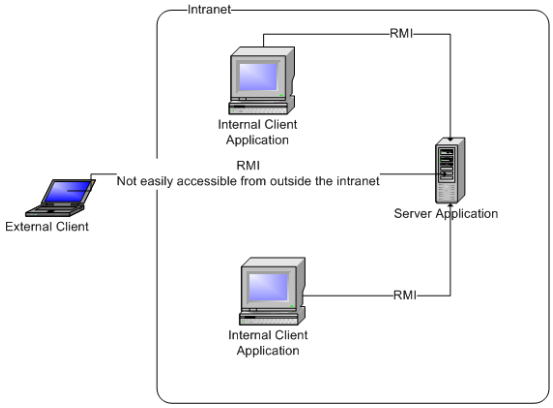


Fig. 1. External access to the application using Java RMI

Here, at least the following concerns regarding direct RMI connection arise:

- firewalls blocking the RMI (company policy, users using public free Internet, or Internet c afés could have an access problem),
- securing data transmission (sending data / man-in-the-middle attacks),
- no language / platform independence (while a Web Service can be used by clients written in many different languages).

Nowadays, *wrappers* and *transducers* are the most popular ways of approaching these problems. Here, a *transducer* is component pattern, which accepts messages from a requesting system, translates them into the program's native communication protocol, and passes to an application [2]. It also accepts responses, translates them back and forwards to the requester. A *wrapper*, on the other hand, can "inject" code into an application to allow it to communicate with the requester. In other words, it can directly invoke API methods, and examine and modify data structures of the program [2]. Both wrappers and transducers can secure (limit, authorize, etc.) access to the remote software; extend its functionality and improve performance [4].

In this paper we explore possibilities brought about by utilization of Web Services and software agents as transducers. Both approaches fit into this role as they use message-oriented communication and are platform independent. Specifically, we show how they can be used to expose access to an SMTP server. Note that transducers are used as we do not have direct access to the server (code injection and method invocation), and we can relay only on messaging utilizing SMTP.

We proceed as follows. In the next section we briefly introduce Web Services and software Agents as well as discuss the scenario used in our experiments. We follow with a head-to-head comparison of both approaches.

2 The scenario

The aim of this paper is to compare Web Services and software agents used as transducers to expose an SMTP server. Here we follow the general approach proposed in [14, 15], where we have experimented with performance of software agents in selected scenarios. Our goal is perform an initial assessment which of the two approaches is likely to be better to "glue applications." Obviously, our aim is only to establish some initial guidelines as to pros and cons of both approaches. We start our description from introducing the SMTP protocol and its weaknesses.

2.1 Simple Mail Transfer Protocol – weak points

The Simple Mail Transfer Protocol (SMTP) is a standardized protocol for e-mail transmission [5, 6]. It uses relatively simple, text-based communication. Its simplicity is advantageous but results in lack of *flexibility*, and in *security* and *performance* concerns. Let us look into them in some detail.

Security concerns. Sendmail is one of the first mail transfer agents utilizing the SMTP protocol. It listens for connections using port 25. History of sendmail shows how exposing port 25 can lead to serious vulnerabilities allowing hackers access to the shell of the machine running the sendmail/SMTP Server [8]. Therefore, standard security policies mandate closing port 25. Note that the SMTP-AUTH extension [7], attempts at assuring that only authorized users are able to send messages (reducing the spamming problem). While hosting companies require authentication, typical client applications use simple Base64 algorithm for encrypting authentication data. This does not protect against eavesdroppers and man-in-the middle attacks

```
S: 220 www.example.com ESMTP Postfix
C: HELO mydomain.com
S: 250 Hello mydomain.com
C: MAIL FROM:<sender@mydomain.com>
S: 250 Ok
C: RCPT TO:<friend@example.com>
S: 250 Ok
C: DATA
S: 354 End data with <CR><LF>.<CR><LF>
C: Subject: test message
C: From: sender@mydomain.com
C: To: friend@example.com
C:
C: Hello,
C: This is a test.
C: Goodbye.
C: .
S: 250 Ok: queued as 12345
C: QUIT
S: 221 Bye
```

Fig. 2. Example of SMTP communication: C is a client, S is server.

Lack of flexibility and performance issues. Let us use Figure 2, an example of communication between of a client and the SMTP server. Here, the client wants to send an e-mail from the `sender@mydomain.com` address to the `friend@example.com`. During this session client sends 182 characters to the SMPT server and receives 132 characters back. Real content (sender and recipient addresses, and the message body) contains only 90 characters, which means that the overhead is 204 characters (92 characters of commands, and 132 of response). While this is a relatively small cost in the case of sending a single message to a single recipient, it becomes substantial in the case of 1000+ recipients and large attachment, as in the following scenario:

> *An accountant in a corporation needs to send salary information to all employees. Company uses an external provider for e-mail accounts. The message contains a long legal text as well as a few big attachments – overall, its size is about 500kB. There are 1000 employees and each should receive customized information.*

Usage of the SMTP protocol would require sending 1000 separate e-mails and would transferr approximately 1000*500 kB = 500000 kB (around 500 MB of data).

2.2 Extending SMTP functionality with a transducer

Using a transducer it is possible to extend the SMTP server by allowing for *MailTemplate* messaging. The idea is to allow client to send (to the SMTP server) message body (in our scenario 500kB) in a form of a template (with parameter placeholders) and a set of parameters to be inserted before the mail is actually sent. This allows for e-mail customization as well as limits use of the network (between the mail client and the SMTP server). In this work we compare performance of four possible approaches to communicating with the SMTP server:

- (A1) direct SMTP communication with an external SMTP server,
- (A2) direct SMTP communication with an internal SMTP server,
- (A3) communication using the *MailTemplate* through a transducer realized by Web Service deployed on the remote machine with the SMTP Server,
- (A4) communication using the *MailTemplate* through a transducer realized by a software agent deployed on the remote machine with SMTP Server.

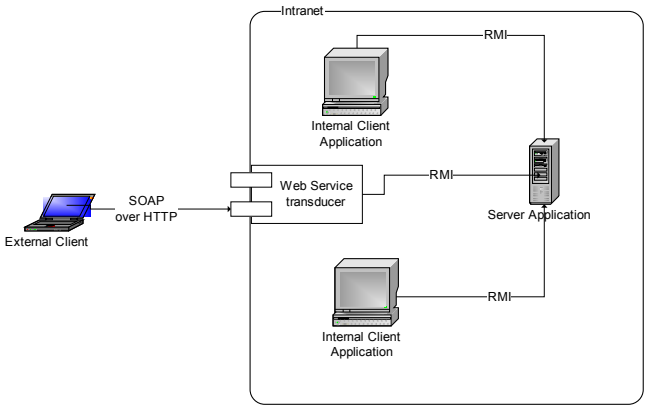


Fig. 3. Secure external access to the application using Web Service over HTTP

The Web Service transducer is deployed on the machine running the SMTP Server, or within the local network where the SMTP Server is located. Web Service utilizes standard functionality of the SMTP Server through direct communication and exposes the extended functionality to the external clients (see Figure 3). Note that the main advantage of Web Services is conformity with most firewall policies and platform independence. At the same time, their basic drawback is overhead introduced by use of SOAP data structures. We will return to these issues in what follows.

The agent-based transducer (*AgentReceiver*), located at the SMTP server, exposes its functionality to other agents. It uses FIPA ACL messaging to communicate with *AgentSender,* which forwards user requests. In our scenario *AgentSender* prepares the template, set of parameters and list of receivers; serializes them and sends it, as an ACL message, to the *AgentReceiver*. *AgentReceiver* deserializes message, acts as a mail-merger that prepares messages (for the SMTP server), authenticates the user and sends messages out. Furthermore, *AgentReceiver* informs *AgentSender* whether sending e-mails was successful. The other direction of communication mirrors this scenario. Here, we assume that both agents reside within different platforms, and thus utilize an agnostic communication protocol (e.g. HTTPS). Finally, note that in our work we use JADE (Java Agent Development Framework) [9] to implement agents.

3 Comparing Web Services and software agents as transducers

When comparing the four possible approaches we consider their following features:

- broadly understood security (including authentication and authorization),
- message overhead (in the “accountant scenario”),
- flexibility (in the “accountant scenario”),
- performance,
- easiness of implementation and deployment.

3.1 Comparing security

As specified in Section 2.1, the approaches (A1) and (A2) are not really secure. More precisely, direct access to the SMTP server results in:

a) lack of protection against the man-in-the-middle attack,
b) lack of protection against retrieving the content of the message by sniffers,
c) lack of protection against retrieving and decrypting user name and password (used for authentication with the SMTP server),
d) lack of conformity with most company security policies,
e) access to the local network of the remote machine.

The latter problem can be solved by closing all ports, but the port for the text-based HTTP protocol (as most company firewall policies do). A transducer can prevent exposing the application and the local network to direct attacks. Other problems can be addressed by use of the secure HTTPS protocol. Note that both JADE agents and Web Services can use HTTP and HTTPS protocols [12, 13]. The potential drawback is that, for JADE agents, message transfer based on HTTPS is about 15% slower than the HTTP MTP [12]. Similarly, the SSL for Web Services will generate overhead [16]. However, one could resign from using SSL to secure the entire transport, and use a solution that provides security on the message level: the WS-Security [13].

3.2 Comparing message overhead

Both the SMTP and the Web Service generate overhead related to headers. As shown in Figure 2, sending a single mail results in about 204 characters of overhead. In order to establish overhead of Web Services, let us consider the following simple SMTP transducer used for mail sending (in Figure 4). The approximate overhead of the SOAP header request is 480 bytes.

Performance and flexibility gains come from utilization of a Web Service transducer that implements the *MailTemplate* solution and is used in the scenario of an accountant sending e-mails to employees. Here, instead of sending 500 MB of data (+ the SMTP overhead) in the standard SMTP approach, only about 500 kB (+parameters +SOAP overhead) of data will be sent.

Let us now consider an ACL message sent by the *SenderAgent* to the *ReceiverAgent* (see Figure 5). As we can see, the message content is serialized and encoded using the Base64 algorithm (used as default by JADE). The ACL Message header contains 336 characters (non-bold text in Figure 5). Here, (the same as above) 90 characters are contained within a Java object send as a content slot of the ACL message. Serialization of this object generates additional overhead of 415 characters. Next, JADE applies Base64 encoding to the serialized object and as a result output contains 756 characters. Therefore, the total overhead is 1002 characters (more than in the case of a Web Service request).

```
POST /SmtpTransducer.asmx HTTP/1.1
Host: www.example.com
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <SendE-mail xmlns="http://tempuri.org/">
      <from>sender@mydomain.com</from>
      <to>friend@mydomain.com</to>
      <subject>test message</subject>
      <body>Hello,
            This is a test.
            Goodbye.
      </body>
    </SendE-mail>
  </soap12:Body>
</soap12:Envelope>
```

Fig. 4. Example of SOAP Communication for *MailTemplate* request

```
(REQUEST
  :sender  (agent-identifier
    :name SmtpSender@abc123:1099/JADE
    :addresses (sequence http://192.168.1.100:7778/acc )
    :X-JADE-agent-classname oglodek.project.SmtpSenderAgent )
  :receiver  (set ( agent-identifier
     :name SmtpReceiver@corn:1999/JADE
     :addresses (sequence http://corn.bunge.pl:7778/acc )) )
  :X-JADE-Encoding Base64
  :content  " .... Base64 ENCODED VALUE ..... "
```

Fig. 5. Example of ACL Message for *MailTemplate* request sent from the *SenderAgent* to the *ReceiverAgent*

Table below summarizes the overhead of each considered approach:

	Direct access (A1 and A2)	Access via Web Service (A3)	Access via JADE agent (A4)
C haracters sent by the client	182	570	1092
Size of actual content	90	90	90
Overhead	204	480	1002

3.3 Note on (im)possible further flexibility

Note that the *MailTemplate* approach could benefits from an assumption that the *MailTemplate* is not be a data structure, but a mobile executable code, generating a set of messages to be sent. Such mobile code would be carried by the *AgentReceiver* , migrating from a client machine to the SMTP server. This would allow configuring mechanism for generating templates at runtime. However, this solution has a set of problems of its own:

- mobile agent could be dangerous for a target hosting platform (see concept of malicious code, summarized in [10]),
- overhead of sending template and an agent itself higher than in cases A3 and A4,
- required homogeneity of origin and target platforms; as a result (for the time being) *AgentReceiver* would be able to migrate only within JADE platform [11].

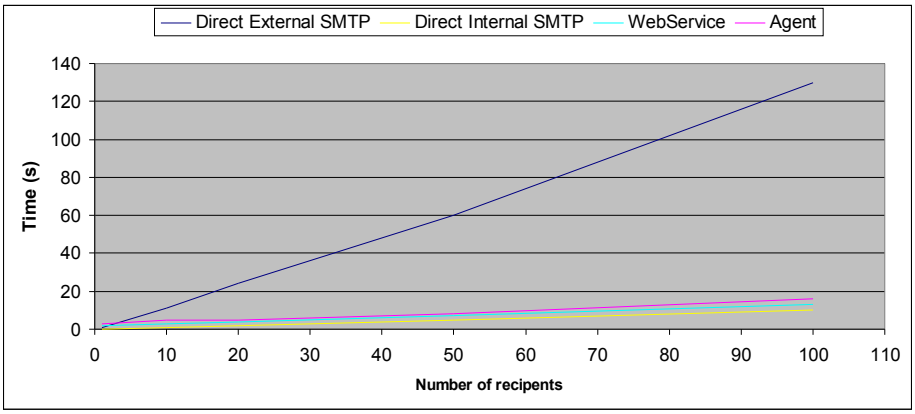
Therefore, we have decided to not to pursue this possible solution further.

3.4 Comparing performance

We examined performance of all four approaches by comparing time of sending e-mail messages of size *s* to *n* recipients (one e-mail per recipient). To be precise, by the *size of a message* we mean:

a) in approaches with direct access to the SMTP server (A1 and A2): the average size of an e-mail message body (without header)
b) in approaches with the *MailTemplate* (A3 and A4): size of the message body (with parameter placeholders), plus vector of parameters, without header.

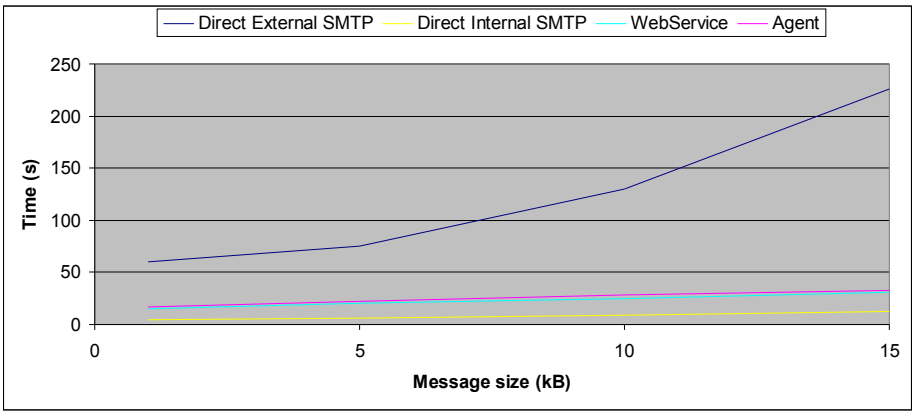
In approaches A1 and A2, *n* messages were send to the SMTP server, whereas in approaches A3 and A4, a single *MessageTemplate* was send to the transducer, which translated it into *n* requests to the SMTP server. Under these assumptions we have performed two tests. First, we studied the relation between completion time and number of recipients (n), assuming constant message size (s ~10 kB). Second, the number of recipients was fixed ($n = 1000$), while the relation between sending time and message size (s) was evaluated. Both tests were performed on the system consisting of two PC's: (1) PC with 2 GHz Intel Core Duo; 2.0 GB RAM; running Microsoft Windows XP; it hosted client applications (in approaches A1-A3) or the *AgentSender* ; (2) 4200+ AMD Athlon 64 X2 Dual; 4.0 GB RAM; running Microsoft Windows Server 2003 R2 64 bit; it hosted the ESMPT Mail Service (version 6.0.3790.3959) and transducers (in A3 and A4): the *AgentReceiver* or the WebService. The WebService was implemented in C# .NET under Microsoft .NET Framework 2.0 and was hosted by the IIS 7 web server. Agents have been implemented in JDK 1.6, and run within JADE 3.5 framework. Finally, both machines were located in a LAN with 1 Mbit/s download and 512 kbit/s upload bandwidths. The results of the tests are presented on Figures 6 and 7, respectively.



Number of recipients	Direct External SMTP	Direct Internal SMTP	Web Service	Agent
1	1.04	0.14	2.13	3.02
10	11.21	1.18	3.21	4.35
20	24.38	2.23	4.42	5.53
50	60.92	5.56	7.58	8.77
100	128.89	10.98	13.64	15.21

Fig. 6. Relation between time (seconds) and number of recipients

We can observe a very large difference between results obtained with the external SMTP server (A1) and the remaining approaches. This was to be expected as the external SMTP server requires sending *n* messages across the network. The situation changes with the internal SMTP server, where messages are generated "internally" and no extra time is used sending them (over the network) to the SMTP server. As a result this solution is the fastest. It is however similar to utilization of transducers, where the message template is send once to the remote server and the transducer communicates with the SMTP server locally. The only difference is time used by SOAP / ACL messaging, and overhead of transducer processing messages. We can also notice that the Web Service (A3) is slightly more efficient than using software agents (A4). This can be explained by the bigger overhead of an ACL message request comparing to a SOAP request. Technology used for implementation of Web Service (.NET on IIS 7.0) versus Agents and HTTP MTP (Java with internal engine acting as the HTTP server) can also have some impact on performance.



Message size (kB)	Direct External SMTP	Direct Internal SMTP	Web Service	Agent
1	59.42	4.67	15.12	17.43
5	76.38	6.53	20.41	22.21
10	127.65	9.28	25.32	27.15
15	226.21	12.43	31.26	33.13

Fig. 7. Relation between time (seconds) and message size.

3.5 Implementation and deployment issues

Most of modern development tools allow easy use of Web Services. For example Visual Studio can generate proxy class for a given URL of a Web Service so that one does not need to deal with SOAP message creation. Similarly, all serialization / deserialization of objects in a SOAP message will be done automatically by the proxy class. Hence the developer that has no knowledge about SOAP headers or SMTP commands can start working with them without any additional learning required. In the case of the SMTP server one needs to know only the specification of the SMTP protocol – commands, syntax, order of commands, etc. At the same time implementing software agents with JADE, especially behavioral programming, implementing interaction protocols and developing communication ontologies, can be somewhat more difficult for users that are working with agents for the first time.

4. Conclusions

In this paper we have compared two transducer-based approaches to wrapping legacy software. One utilized Web Services, while the other was based on software agents.

We have fund that (a) utilization of either of the transducers can have clear positive effects on performance, flexibility and security of legacy software; and (b) Web Services seem to be able to handle messaging somewhat more efficiently, by introducing less overhead. The next step that should be undertaken is extending the breadth of scenarios under which the performance comparison is undertaken.

References

1. M. Gawinecki, M. Kruszyk, M.Paprzycki, M.Ganzha (2007) *Pitfalls of agent system development o the basis of a Travel Support System,* In Proceedings of the BIS 2007 Conference (to appear), http://agentlab.swps.edu.pl/agent_papers/BIS_2007.pdf
2. Micheal R. Genesereth, Steven P. Ketchpel (1994), *Software Agents,* In Communication of the ACM, Vol. 37, No. 7 July. http://citeseer.ist.psu.edu/genesereth94software.html
3. S.Srivatsa Sivan and R. Venkatavaradan (2005), *Design Guidelines: Building Web Service Wrappers for an XML-based System,* http://www.devx.com/enterprise/Article/27882
4. Design Patterns, *Elements of Reusable Object-Oriented Software,* Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides , Addison Wesley.
5. RFC 821, http://www.faqs.org/ftp/rfc/pdf/rfc821.txt.pdf
6. RFC 1123, http://www.faqs.org/ftp/rfc/pdf/rfc1123.txt.pdf
7. RFC 2554, http://www.faqs.org/ftp/rfc/pdf/rfc2554.txt.pdf
8. *Hacking: The Art of Exploitation,* 2nd Edition, Jon Erickson.
9. JADE (Java Agent DEvelopment Framework), http://jade.tilab.com/
10. Łukasz Nitschke, Marcin Paprzycki, Michał Ren, *Mobile Agent Security,* http://agentlab.swps.edu.pl/agent_papers/NATO_2006.pdf
11. Inter-Platform Mobility Project, https://tao.uab.cat/ipmp/
12. Jose A. Exposito, Joan Ametller, Sergi Robles , *How to use the new HTTP MTP with JADE,* http://jade.tilab.com/doc/tutorials/HTTP_UAB.html
13. Mike Lehman Securing Web Services, http://www.oracle.com/technology/oramag/oracle/05-jan/o15web.html
14. K. Chmiel, M. Gawinecki, P. Kaczmarek, M. Szymczak, M. Paprzycki: *Efficiency of JADE Agent Platform,* Scientific Programming, vol. 13, no. 2, 2005, 159-172.
15. K. Chmiel, D. Tomiak, M. Gawinecki, P. Kaczmarek, M. Szymczak, M. Paprzycki: *Testing the Efficiency of JADE Agent Platform,* In: *Proceedings of the ISPDC 2004 Conference,* IEEE Computer Society Press, Los Alamitos, CA, 49-57.
16. Use of SSL Creates Performance Overhead for Browsers, http://support.microsoft.com/kb/15003