

Agent Based Computing

*Utilization of an agent and Web Service as
wrappers for existing legacy software*

Michał Oglodek

Warsaw University of Technology

2007-04-02

Introduction

Direct access to application isn't always practical, secure or flexible

Aim of this project is to show how wrapping approach can bring the benefits to existing applications.

Wrapped Application

We will wrap SMTP (Simple Mail Transfer Protocol) service to show its weak parts and to present improvements by using wrappers

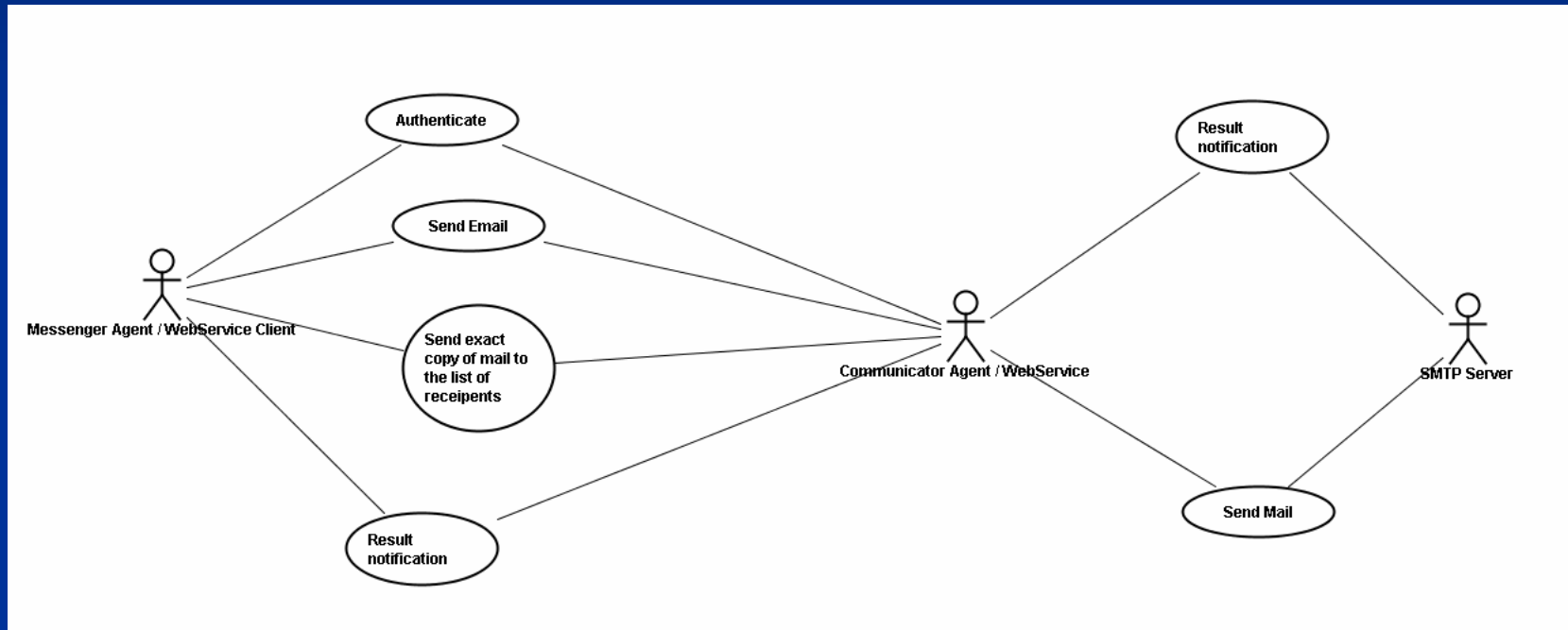
Weak parts of SMTP

- Simplicity
- Not really secure (a lot of exploits in history, transmission of data / passwords not encrypted well)
- Usually exposed via port 25
- Average user does not know how to communicate (syntax, commands, order of commands etc.)

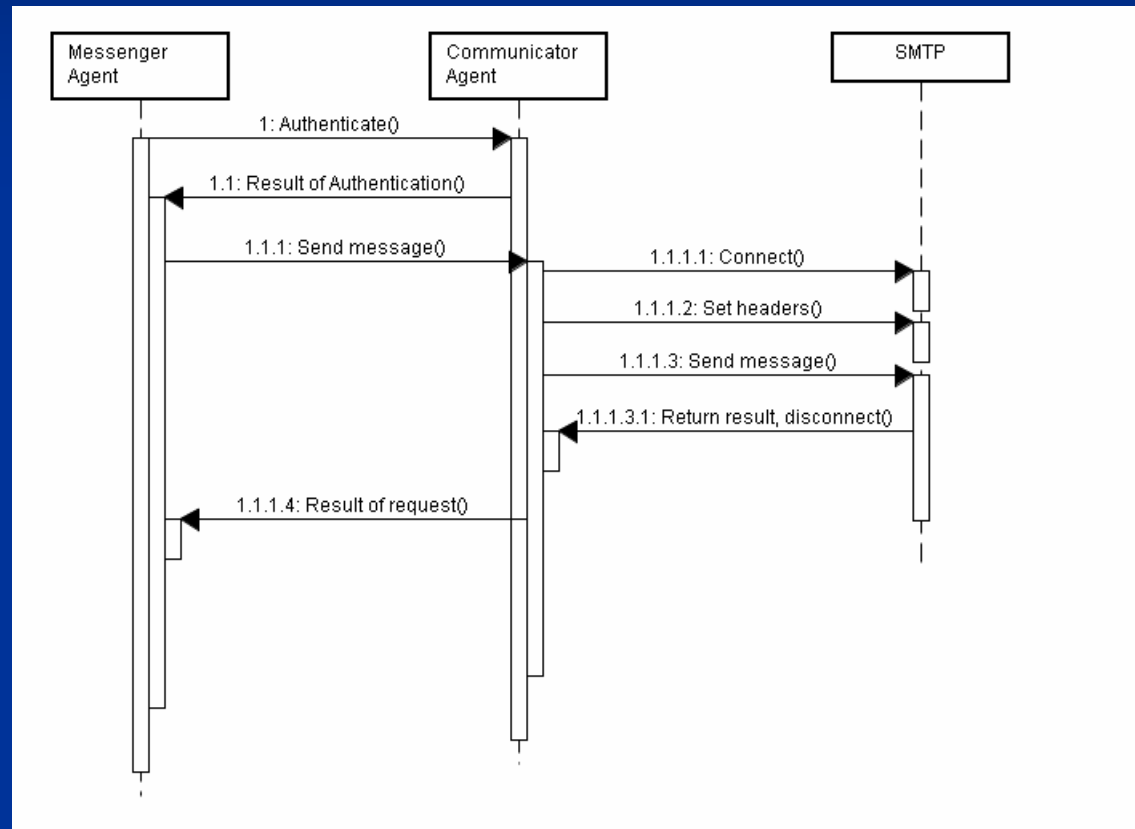
Wrapper improvements

- Port 25 can be blocked
- No Firewall issues
- Security of transmission
 - WebService over HTTPS
 - Agent communication over HTTPS
- Additional Functionality (Email Templates)
- Network Traffic is decreased

Use Case Diagram



Sequence Diagram

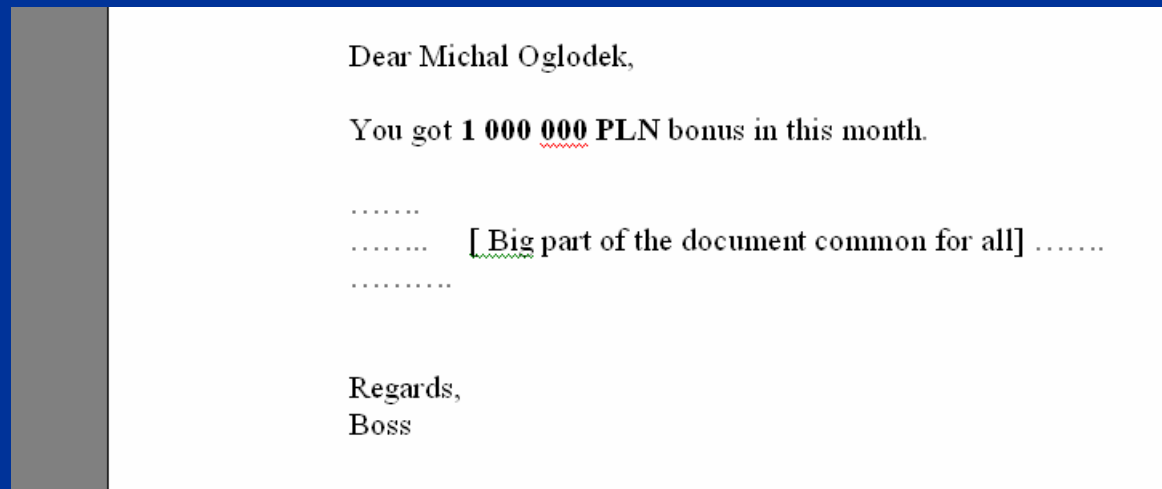


Improvement - security

- Standard Email clients are using simple SMTP authentication – passwords and data are unsafe (easy to decrypt)
- HTTPS = HTTP + SSL (Secure Sockets Layer)
 - ensures reasonable protection from eavesdroppers and man-in-the middle attacks – passwords and data is safe

Improvement – functionality / efficiency

- Example: Accountancy application that needs to send a 50kB email to a large number (1000) of people with custom information for each user (eg. About the bonus the employee got).



Dear Michal Oglodek,

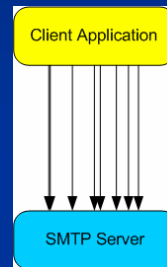
You got **1 000 000 PLN** bonus in this month.

.....
..... [Big part of the document common for all]
.....

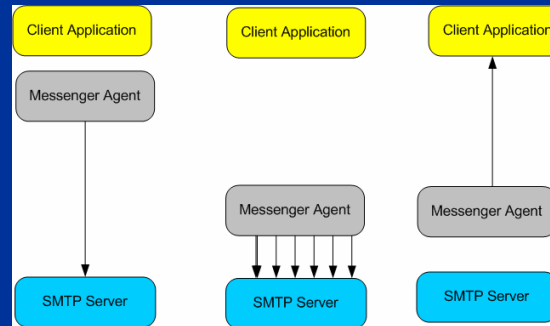
Regards,
Boss

Improvement - comparison

- Standard SMTP approach (50kB * 1000 = 50 MB sent + headers)



- Agents (50kB + list of [email, name, value] for each user + agent size)



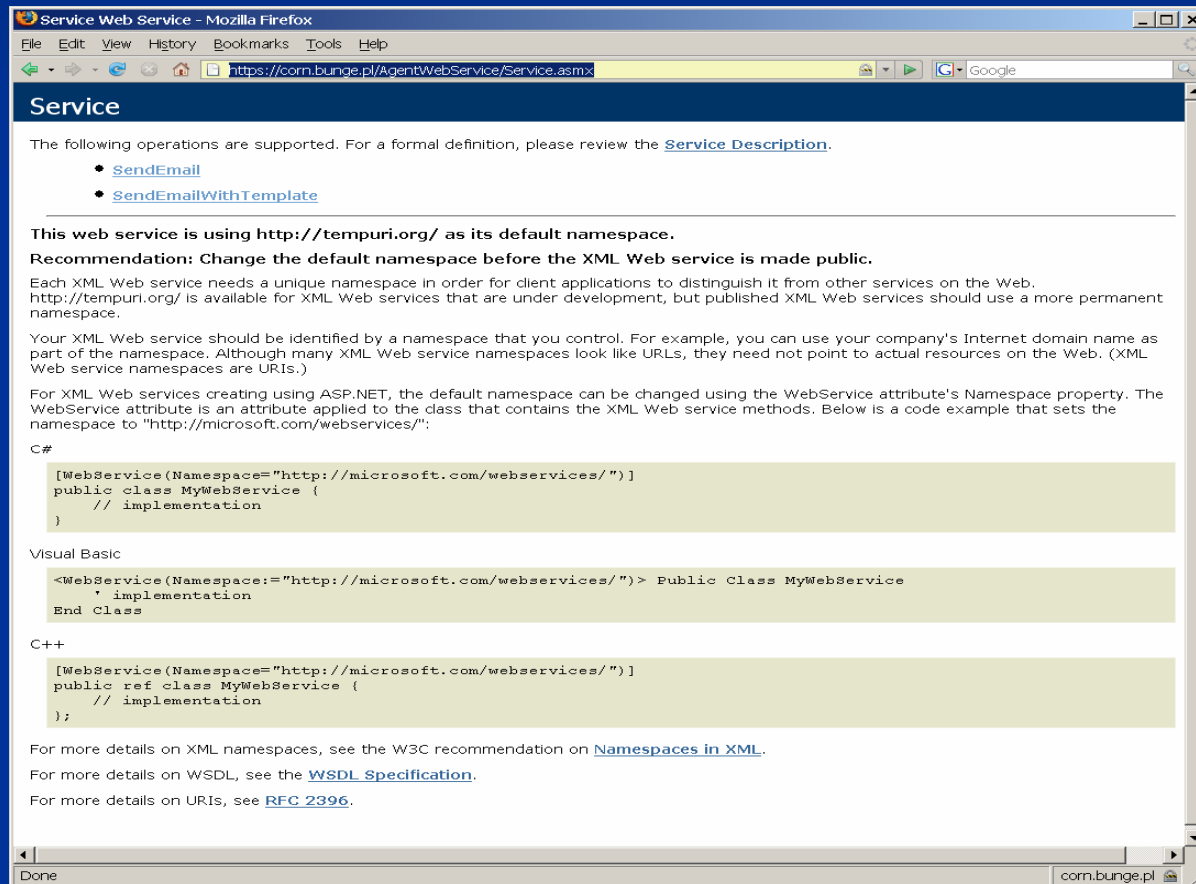
- Improvement also doable with Webservice

Other improvements

- Other improvements:
 - Authentication process irrespective of SMTP service
 - Authorization (for example Communicator Agent could reject the request coming from Messenger Agent, if the request did not meet the authorization policy – eg. Given user cannot send emails for more than 10 users at once or cannot send email to boss etc...)

Project modules

- <https://corn.bunge.pl/AgentWebService/Service.asmx> (WebService)



The screenshot shows a Mozilla Firefox browser window with the address bar containing <https://corn.bunge.pl/AgentWebService/Service.asmx>. The page title is "Service Web Service - Mozilla Firefox". The main content area is titled "Service" and contains the following text:

The following operations are supported. For a formal definition, please review the [Service Description](#).

- [SendEmail](#)
- [SendEmailWithTemplate](#)

This web service is using <http://tempuri.org/> as its default namespace.

Recommendation: Change the default namespace before the XML Web service is made public.

Each XML Web service needs a unique namespace in order for client applications to distinguish it from other services on the Web. <http://tempuri.org/> is available for XML Web services that are under development, but published XML Web services should use a more permanent namespace.

Your XML Web service should be identified by a namespace that you control. For example, you can use your company's Internet domain name as part of the namespace. Although many XML Web service namespaces look like URLs, they need not point to actual resources on the Web. (XML Web service namespaces are URIs.)

For XML Web services creating using ASP.NET, the default namespace can be changed using the WebService attribute's Namespace property. The WebService attribute is an attribute applied to the class that contains the XML Web service methods. Below is a code example that sets the namespace to "http://microsoft.com/webservices/":

C#

```
[WebService(Namespace="http://microsoft.com/webservices/")]
public class MyWebService {
    // implementation
}
```

Visual Basic

```
<WebService(Namespace="http://microsoft.com/webservices/")> Public Class MyWebService
    ' implementation
End Class
```

C++

```
[WebService(Namespace="http://microsoft.com/webservices/")]
public ref class MyWebService {
    // implementation
};
```

For more details on XML namespaces, see the W3C recommendation on [Namespaces in XML](#).

For more details on WSDL, see the [WSDL Specification](#).

For more details on URIs, see [RFC 2396](#).

The browser status bar at the bottom shows "Done" and the address "corn.bunge.pl".

Project Modules

- Agent Wrapper (SmtpReceiver) and Agent Sender (SmtpSender) – deployed on the remote server (<http://corn.bunge.pl:7778/acc>)

Project Modules

- User Interface for testing SMTP Direct communication and WebService

The screenshot shows a Windows application window titled "SMTP Direct Test". The interface is divided into several sections:

- Company sending customized emails to employees:** This section contains three columns of data:
 - Emails:** moglodek@o2.pl, michal@oglodek.pl, michalo@ammeeting.com
 - Names:** Michal Oglodek, Mr. ABC, Mr. XYZ
 - Bonuses:** \$100, \$200, \$100000
- Repetition:** A text box containing the value "100".
- Time elapsed:** A label with a corresponding empty text box.
- Local:** A checkbox that is currently unchecked.
- Buttons:** "Send using Direct SMTP" and "Send using WebService".
- Base64 Decoder:** A separate section on the right with a "Password:" label, an empty text box, and a "Decode" button.

Testing

- Following test were performed to show the improvement with new functionality that wrappers brought – MailTemplates
- Template File – size: 5 820 bytes
- 2 parameters to be passed in template (name, bonus)
- 3 different mail account with repetition = 100 (simulation of sending emails to 300 users)

Testing - Synchronous

Mode	Time
Direct SMTP Communication	3 min 46 sec
Local SMTP Server	1 min 11 sec
WebService	31.265
Agent	32.178

Testing - Asynchronous

Mode	Time
Direct SMTP Communication	45 sec
Local SMTP Server	0.81 sec
WebService	1.32 sec
Agent	1.59 sec