

WSEmail Applications

Kevin Lux
University of Pennsylvania

August 2005

Abstract

WSEmail is a system of web services that aims to show how email and messaging, in general, could be securely redone within a SOAP/XML paradigm. With this fundamental change in message delivery and processing, there exists the opportunity to integrate disparate systems that rely on their own messaging protocols into a common platform. Careful design of the base system allows plug-ins to be written that can bring together different business functions under a unified architecture. The different kinds of plugins available in WSEmail will be explored, along with the sample applications created with them.

Introduction

When one thinks of business applications, a variety of systems come to mind. Typically a business has a messaging system for internal and external communication, a trouble ticket system, change control system, payroll system and ERP or CRM systems, among others. A quick glance at these systems will show that each relies upon messaging of some kind. Most of these systems, however, do not communicate with the others, requiring users to have multiple sets of authentication credentials, usually one for each system. This creates an administrative burden of not only keeping each application up, but also making sure each user has the correct access control. Some of these problems can be averted by the use of middleware specifically designed to tie the systems together, but the fact remains that the systems are generally considered as separate entities, even though they primarily manage dataflow in the enterprise.

This design may, in part, be due to the lack of an extensible messaging infrastructure that can communicate between platforms and support a wide variety of cryptographic features. With the advent of web services and, specifically, the creation of WSEmail, a lot of these concerns can be addressed. WSEmail is able to support a variety of authentication techniques and can be customized for each enterprise. It is capable of being centralized or decentralized. Message content can be as structured or as unstructured as necessary and is capable of being cryptographically secured. Given these

features of WSEmail, the line between application platform and messaging platform becomes extremely blurred.

A platform is generally measured by the availability and programmability of the applications it can run. WSEmail does not place many restrictions on the kinds of applications that can be built and executed within the system and provides a powerful set of plugin interfaces that allows processing to occur at various phases of system operation. The integration with both the client and server software allow very feature rich applications to be built.

WSEmail already has some “applications” (we define an application as a service capable of being provided by the platform, but not typically considered a usual feature of email-like messaging). By exploring the current applications, it should become possible to see how they integrate into WSEmail, both in technical and conceptual terms. A little imagination will show that most large applications could be integrated with the WSEmail platform to create the ultimate business platform.

Implementation Background

Before delving into the applications, it might be helpful to describe a bit about how WSEmail is currently designed and how it is able to adapt to various applications. In the application section, each application will make reference to the kinds of interfaces it uses. These interfaces describe how much access the application has to the mail server or client software, in addition to where the plugin is activated to affect message delivery or reading.

Server-side Plugins

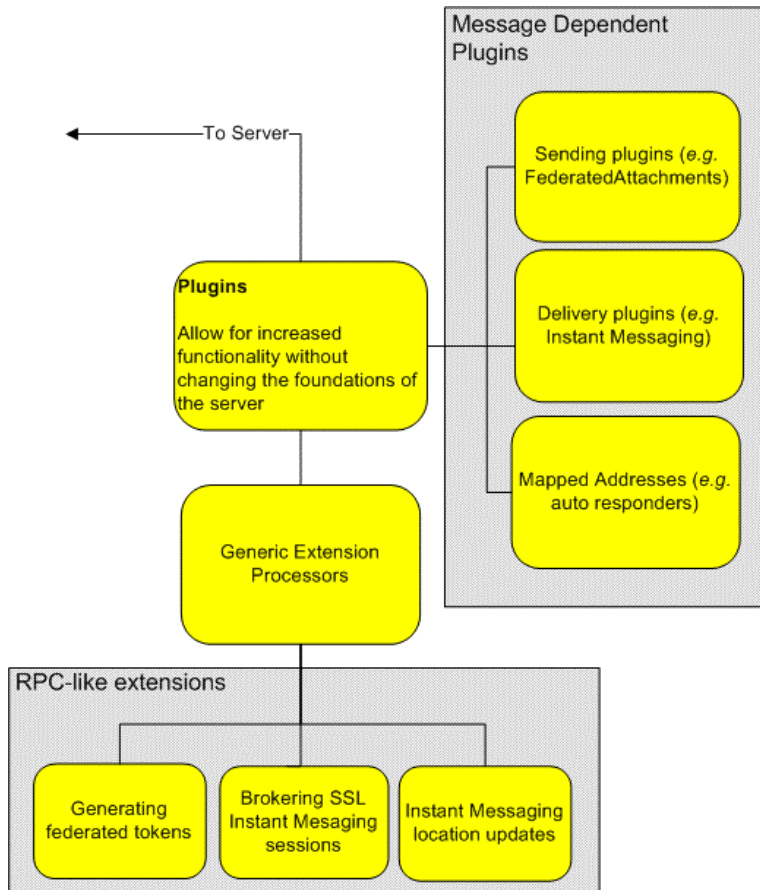
Server plugins are just libraries of code that conform to certain interfaces known to the server. During server initialization, the server goes through the list of plugins it is to load from its configuration file. For each plugin, a specific object class is listed along with the name of a library from which it may be loaded. The server attempts to find the library and instantiate the object. If it successfully loads the plugin, the server will request further configuration data from the plugin and use this information to place the plugin in the appropriate execution queue. The process of loading or unloading plugins is entirely dynamic and can happen at any time. In addition, the execution queues can be reprioritized or disabled while the server is running.

All plugins must implement the `IServerPlugin` interface, which allows the server to understand the purpose of the plugin and add the plugin to the appropriate processing queues. Beyond that, there are two main classes of plugins in the current implementation of WSEmail: message-dependent and RPC-like (or message-independent). Message dependent plugins are those plugins that depend on a message to be present to execute. They can be inserted in various places in the delivery cycle including the initial receipt of a message (`ISendingProcessor`) or the final destination of a message (`IDeliveryProcessor`). To create an analogy to current systems, `ISendingProcessor` is similar to any processing done by `sendmail` (such as verifying relay permissions, stripping oversized attachments, etc). `IDeliveryProcessor` is more like user-space programs such as `procmail` or a vacation messaging scripts.

The RPC-like interface is a generic “catch-all” interface (IExtensionProcessor).

Like the IDeliveryProcessor and ISendingProcessor, it too inherits from the IServerPlugin interface. Upon initializing, the plugin implementing the interface provides the server with an “extension identifier”. This identifier is used by the server route incoming requests to the appropriate plugin. There is no required or defined structure for the actual requests and most plugins tend to view requests as XML documents or fragments. This provides a lot of flexibility in terms of the data an application can process.

All requests that go to any kind of plugin are



first authenticated by the server. In the case of IExtensionProcessor and IDeliveryProcessors plugins, an “environment” object is created and passed to the plugin to allow access to authentication tokens and raw XML streams directly from the server. Other plugins are generally only given the message that triggered their execution.

Plugins can implement more than one interface, which allows increased functionality in one piece of code. It is possible for a plugin to implement both the IExtensionProcessor and the IDeliveryProcessor (or any variant) interfaces. This would allow the plugin to interact with messages as they are being delivered, but also to be configurable using some protocol that interacts with the IExtensionProcessor interface. Implementing multiple interfaces allows plugins to share data that should be accessible though multiple paths. Examples of where composite plugins (a plugin implementing multiple interfaces) are useful will be given in the applications section.

Client-side Plugins

In the current implementation, client-side plugins only affect message reading and are much more dynamic in nature than server-side plugins. Similar to server-side plugins, client-side plugins are libraries of code, but they extend an abstract class (DynamicForms.BaseObject) instead of implementing an interface, mostly for convenience of programming. They also contain more information than their server-side

components including version and network location information. With this additional information, the plugin can create messages that can be received by other clients without the plugin. A common stub can then be executed to download the plugin using its supplied location information. After presenting the user with the chance to authenticate the downloaded code (currently Authenticode is used), it can be dynamically loaded and then executed. The plugin information (version, name and library location) is also saved in a registry that will allow the new recipient to use the same plugin at a later point, assuming the plugin allows them to.

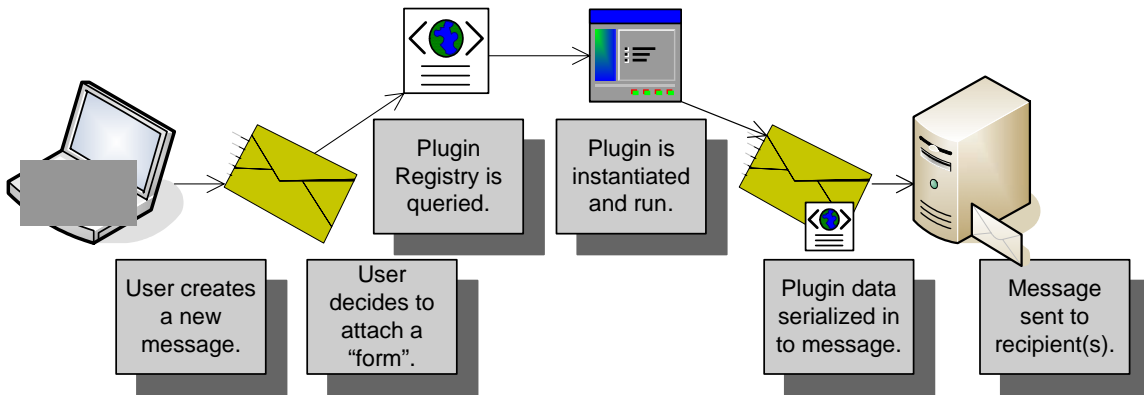


Figure 2 - Client-side plugins from sender's perspective.

The originator of a message creates a new message and adds a "form", picking a specific form from their registry. This form will, usually, present them with a UI that requires information. After filling out the form, the data contained in is then serialized down to an XML document that is "attached" to the message. The original message is notified by the plugin where message should be sent next for the message to be appropriately processed. The plugin is unloaded and the user is then able to send the message.

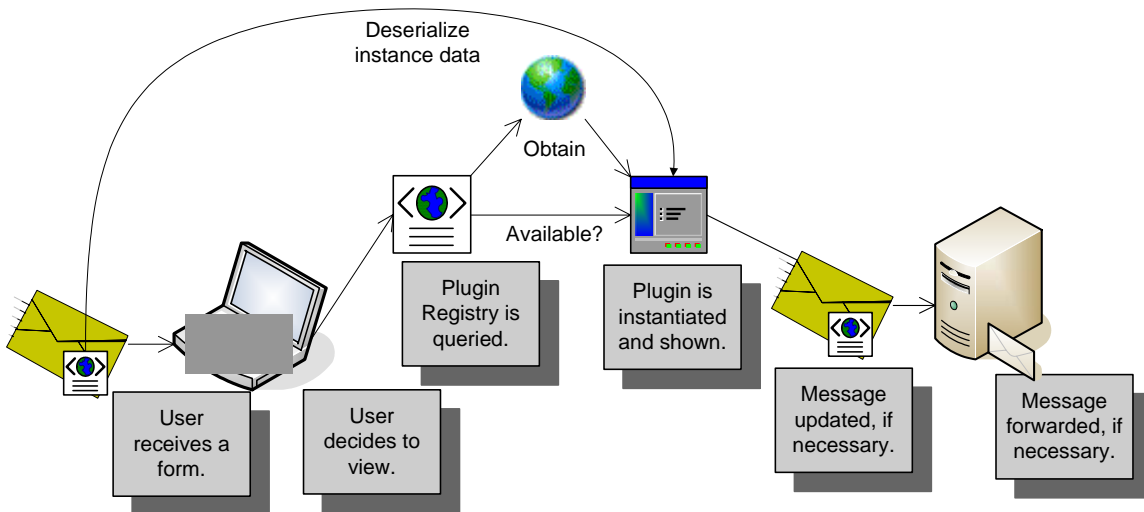


Figure 3 - Client-side plugins from recipient's perspective.

A recipient of the message will see that a form is attached. Upon viewing the form, the mail client will attempt to load the plugin or obtain it if it is not present on the system. After a successful plugin load, the XML document describing the data the plugin processes is pushed to the plugin, which deserializes the data and loads whatever state is necessary. The plugin's logic then takes control.

A typical question is "what do client-side plugin UIs look like to the user". There is not a complete answer to such a question, as the answer is up to the plugin designer. A plugin can have no interface at all, a few message boxes or a full-blown GUI. Since WSEmail is based upon the .Net framework, plugins are able to leverage the rich UI elements that can be in a graphical .Net application, without the need to transfer libraries for rendering graphics, assuming the plugin designer uses built-in .Net object. Plugins also have access to authentication information in the mail client and may petition the user for access to their federated token. This allows plugins to perform secure web service calls to automatically fill in

information or perform other functions.

For convenience and to limit bandwidth used, multiple plugins can be contained within one library file. Each plugin will be enumerated and registered with the client application. This allows system administrators to deploy one library with updated plugins instead of worrying about each and every plugin separately.

Applications

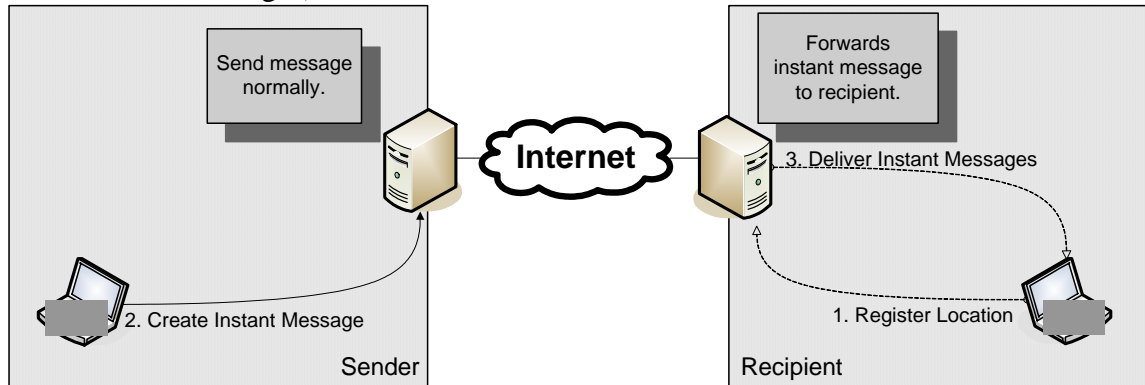
As previously mentioned, WSEmail has created some applications for demonstration purposes. In this section, a few of the applications will be examined for usefulness, integration and "building block" potential. This survey of WSEmail applications should provide the reader with a better idea of how WSEmail is and how modifications are made to the system to support new features.

Instant Messaging

Since its introduction, instant messaging has taken off. The convenience of being able to have a pseudo-realtime conversation with a peer (or a group) is obvious. It eliminates sending short email messages back and forth, instead allowing users to view the exchange as a conversation that has a much smaller lag. Despite these features, there are a few properties of instant messaging that cause quite an irritation among users. Chief among these problems is that there is a distinct handful of chat networks, each with their own protocols and each (generally) incapable of dispatching messages to other networks.

This leads to users being unable to communicate even though they both have chat software, simply because they are not on the same network. It burdens the user because they must create identities for the most popular chat mediums and have a client for each (or a meta client) in order to correspond with most people.

This is extremely unfortunate. Instant messaging is truly just like email, except with slightly different delivery mechanisms and implied higher priority delivery (over non “instant” messages).



The current implementation of WSEmail has a composite plugin that supports instant messaging. It is made up of an `IDeliveryProcessor` and an `IExtensionProcessor` implementation. Upon starting the instant messaging client program, the user automatically registers their location with the server using the `IExtensionProcessor` interface. The plugin records the location information in an internal table. Later, when the server receives a message flagged as instant, the server passes control of the message to the instant messaging plugin using the `IDeliveryProcessor` interface. The plugin simply consults its table of user locations and, if it finds a match, sends the message directly to the client. If a match is not found, the plugin can choose to relinquish control of the message, passing it back to the server. The server will then attempt delivery using the next matching plugin.

Instant messages are currently sent to clients using a Microsoft .Net technology known as Remoting. Similar to RMI, Remoting allows an object to be distributed across a network. Clients remote their instant message queues and have a separate thread watch it. As the server pushes messages into the queue, the client watcher thread will pull them out and coordinate the display of them into the typical conversation like interface.

Business Workflows

Email is the typical venue for all kinds of request processing and tracking. It provides a free-form means of accomplishing work both in and out of the enterprise. The problems with such a system are obvious, including the inability for automated systems to easily process messages due to a lack of structure and, usually, a lack of authentication and tracking. The lack of automation is a major hindrance because it requires humans to decide where the message should go next. If the message is sent to the wrong person or to someone on vacation, business processes can be delayed needlessly.

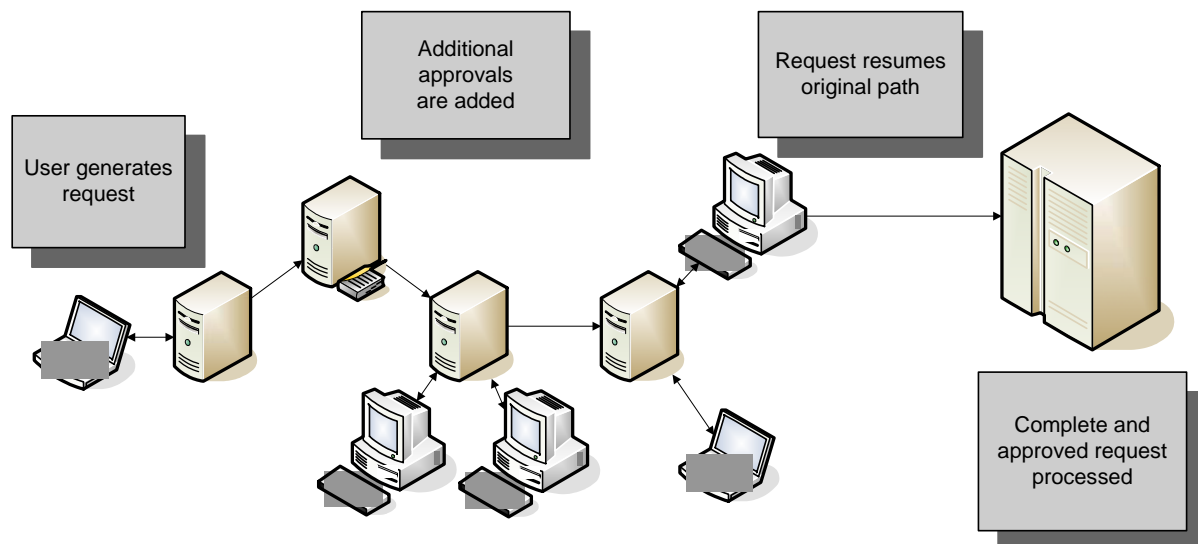
WSEmail has the idea of routed forms to introduce workflow-type scenarios to the platform. A routed form will typically use the client-side plugins mentioned

previously to create a rich UI. These forms generally look very similar to their paper counterparts such as time sheets or requisition forms.

The sender follows the flow of activities described in client-side plugin section. In our prototype workflow plugins, the forms are much smarter than their paper counterparts. They can, for example, provide basic spreadsheet-type function or automatically populate data using the user's federated token and a secure web service query to an HR database. To address the security of the workflow, we assume each user has a unique X.509 certification such that the certificate's CN matches the user's email address. As approvals are acquired, the XML data contained in the form is signed by each approver. This creates an approval list that can be verified by a third party or program, which demonstrates that the data has not been tampered with and can authenticate each member in the workflow.

A user is able to delegate their responsibility in a workflow. They provide the name or names of people who will approve the request in their stead. This feature provides a powerful automation feature. Using server-side plugins, a workflow can be received by a program, which can make decisions about delegation given the current approval list and data contained within the form. A common business process that might require such a feature would be a requisition form. A department might have the ability to buy items under a certain price. If the total requisition is greater than a certain number an additional approval by a member of the purchasing department might be required. A program could easily detect this condition and expedite the entire process.

If so desired, a workflow in our system could end in another program. This program could validate the entire workflow and, if approved, actually order items, perform database manipulations, etc. With an increasing number of online retailers exposing their order process as a web service, it becomes possible to automate a larger number of business functions end-to-end.



Because WSEmail is also functional as a decentralized system, a workflow is capable of extending across multiple enterprises. Although all the enterprises would need to have an agreement to trust a common CA or each others, that is the only additional configuration that is needed. Assuming the enterprises were capable of communicating

before, they can now use workflows on WSEmail to create multi-enterprise processes. There many applications that could use such a setup such a negotiating prices with a supplier or gathering approvals for press releases from interdependent corporations.

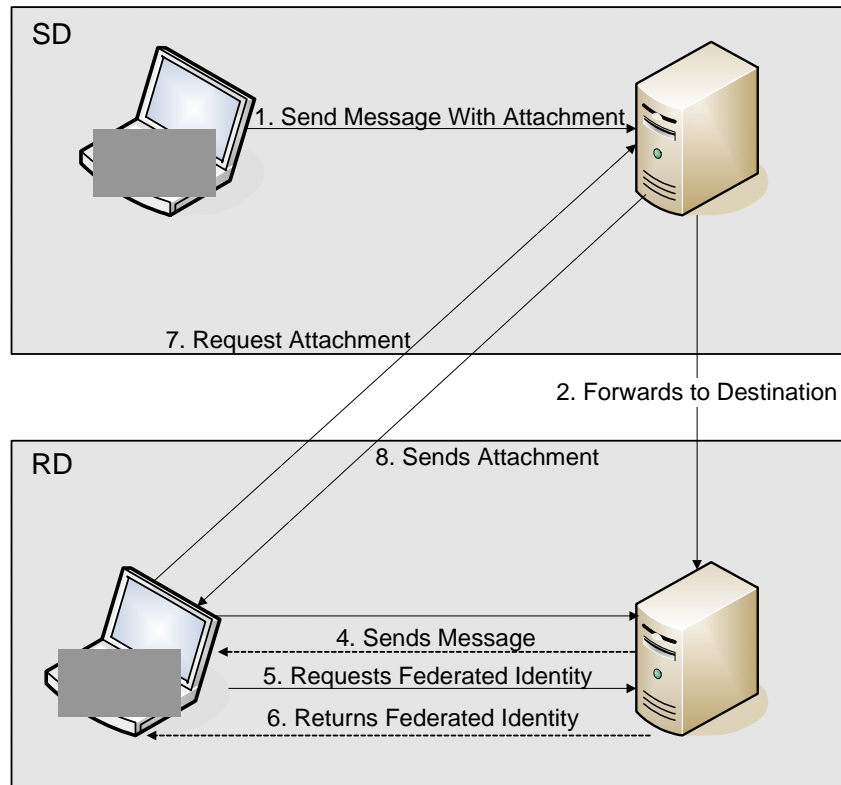
On-Demand Attachments

Attaching files to email has long been a poor man’s way of distributing files to a group. It’s convenient enough and accessible to anyone with an email client. Unfortunately, there are a variety of problems with current email attachments that WSEmail can improve on. Except for certain proprietary email systems, there is no version control system for email attachments, which results in an irritating deluge of costly message resends. The way attachments are currently bundled also requires users to download the entire message and attachment, even if the user only wants to read the message. A solution is to post the attachment on a secured website and just send a link in the message. Unfortunately, this creates an administrative headache, as each message would likely have different access control requirements.

WSEmail solves the problem by introducing the concept of “on-demand” attachments. Simply put, message attachments are handled as a plugin to the WSEmail base protocols. The plugin implements both `IExtensionProcessor` and `ISendingProcessor`.

The client creates a message that contains information about the attachment such as its size and hash, in addition to the normal fields a message contains. The request to the server contains

the message as normal, along with the attachments in DIME format. As the message is received server, the request is intercepted by the plugin using the `ISendingProcessor` interface. Through the environment objects supplied to the plugin, access is gained to all DIME attachments in the request. The files are saved to a database, along with a list of all the recipients. Identifiers are added to the original message that references the files in the database. Delivery of the message now continues as it would normally.



A user, who receives a copy of the message, will see that a file is attached but will not actually have a copy of the file. The user simply has an identifier for the file and the location from which it can be obtained. If the user decides to obtain the attachment, they first acquire a federated identity token. The token is presented using the IExtensionProcessor interface to the server that originally stripped the attachment, along with the unique identifier. The server verifies the authenticity of the token and that the supplied identifier is permitted access. If there is a successful match, the server sends the attachment back to the requestor as a DIME attachment.

Conclusion

These applications just scratch the surface of what WSEmail is capable of as a messaging and application platform. The server can be extended to allow plugins to execute in more places and the same holds true for the client software. Perhaps the biggest hurdle is changing the general perception of email as a messaging tool into that of an integrated application and messaging platform that is still capable of being secure.