Bridging Virtual Communities: The Object-Subscription Protocol

James King and
Jalal Kawash

Department of Computer Science,
University of Calgary,
2500 University Dr. NW,
Calgary, Alberta, Canada, T2N 1N4
E-mail: {kingja, jkawash}@ucalgary.ca

Abstract: Virtual communities play an increasingly important and prevalent role in our daily lives. However, these communities are often *closed* relative each other, forming independent *islands* of communities. Yet, this independence is artificially imposed on such communities by closed support software or policies since the purposes of these communities often overlap. Our objective is bridging these virtual communities, allowing them to seamlessly and securely mesh, in spite of their differing implementations and autonomous policies. Our Object-Subscription Protocol is an attempt in this bridging direction.

Keywords: virtual communities, island bridging, aggregation, syndication, security, standardization

Biographical Notes: James King received his BSc in Mathematics with a focus in cryptology from Florida Atlantic University in 2005. He is currently a MSc student at the University of Calgary. His research interests are in security, privacy, social networks, and the web.

Jalal Kawash received his PhD from the University of Calgary, Canada, in 2000, after which he worked for the IT industry. He then joined the American University of Sharjah from 2002 until 2008. He has also been affiliated with the University of Calgary since 2002. His research interests are in distributed systems and algorithms, mobile virtual communities, and computing education.

1 Introduction

1.1 Background and Motivations

Participating in virtual communities has become an integral part of our daily lives. This is evident in the proliferation of web-based applications that facilitate community formation and support, and being backed by numerous research . Social networking sites, news providers, forums, video sites, photo galleries, blogs, and some source code repositories can serve as examples of these virtual communities. These communities are *closed* virtual organizations: they have their own policies, implementations, and exist in separate virtual spaces. That is, they act like *islands*. Nevertheless, from the collective perspective of a community, it can be advantageous to share content between otherwise separate communities. After all, the interests of these communities and their purposes are not necessarily disjoint. For example, when the Associated Press publishes a news article, the article is then replicated by a variety of other news communities. Additionally, between blogs it is not uncommon to see the replication (or at least quotation) of another blog's content. Additionally, the average participant will be a member of several of these communities simultaneously. As this separation of communities and organizations into islands is only imposed by the fact that these communities function as separate organizations or entities, we are motivated to allow communities to seamlessly mesh in a secure and peer-to-peer manner while still maintaining their independence. This action of island bridging creates a common virtual space between communities that crosses organizational boundaries. The difficulties of this objective stem from the inherent isolation of these communities (especially with their differing and often incompatible policies), and security and privacy issues can only aggravate these difficulties.

Breslin, et. al (2005) have highlighted similar concerns. They point out the fact that "most online communities are islands that are not [inter]linked," and suggest that one method to help bridge these islands is through the use of their Semantically-Interlinked Online Communities (SIOC) ontology. As we argue later, their approach and our approach solve two different problems in the realm of island-bridging, but are not mutually exclusive.

In general, while there seems to be work being done that is related to the task of 'island-bridging' — to the best of our knowledge there does not exist any unifying approach to facilitate the underlying communication that would be required to do this. To this end, we believe that we are proposing a new approach to help solve this problem.

1.2 Our Contributions

This paper presents the Object-Subscription Protocol (OSP) which is a step in the direction of bridging isolated virtual communities. In OSP, a content-provider (or server) is defined to be an entity that publishes content (or data) into various categories (periodical types), from which a client (or subscriber to such periodicals) would then obtain that content. When new content is published, all eligible clients that have 'subscribed' to those categories will then be sent the content as an OSP object represented in XML. The object consists of a header, data area, and an optional signature. The header specifies information regarding the server of origin and may include further (optional) data such as the intended recipient, creation date, and so on. The data area is where the content of an object is placed, and may be optionally encoded, compressed, encrypted, or any combination thereof. Finally, the signature portion of an OSP object is where a message digest or digital signature can be stored - for the content, the header, or both. The protocol consists of specifications for the object format, message format, and the interface through which subscribers may talk to OSP servers. Specifically, the interface allows a client to request an old or missed object (e.g., due to a subscriber's own downtime), alter which categories of content they wish to receive, change their passwords, etc... Interface messages passed within OSP are also be transmitted in the same secure manner as packages delivering content.

The flexibility of OSP allows for a variety of different implementations -a content-provider may disallow subscriptions from the outside or may use some other mechanism to subscribe clients, and older content may or may not be available to be re-sent to a subscriber. For example, a subscriber may have to pay the content-provider, or be a member of a specific group before being given access to certain periodical types. This allows OSP to be used in situations ranging from cross-blog (See **Figure 1**) and cross-site collaboration, to publication of monetary-subscription e-zines, to the distribution of content among a closed group of subscribers, or even just to relay other OSP objects received from other parties. For example, by using OSP, networks of blogs can subscribe to each other based on keywords - letting each blog maintain their own separate databases, yet also sharing common content. Newspapers or journals could provide consumers with content protected by Digital Rights Management (DRM) software, while sharing amongst themselves (or other select groups or individuals) DRM-free versions.

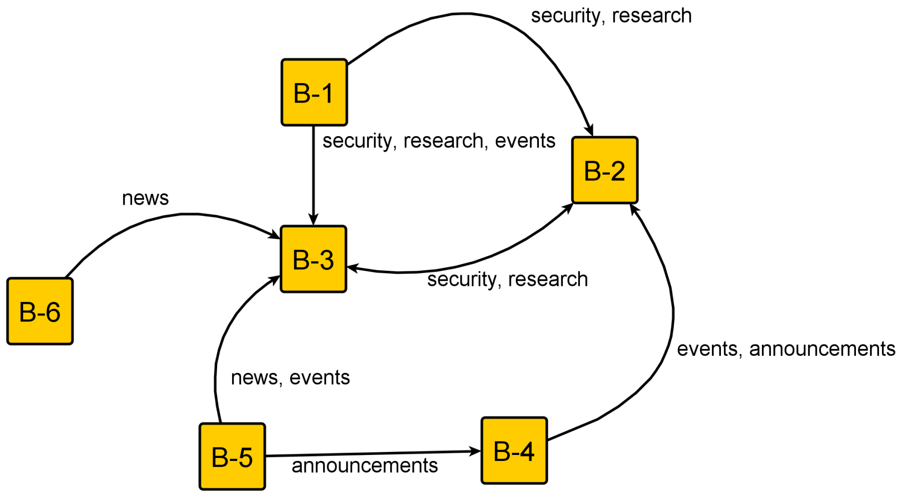


Figure 1 A collection of blogs that are subscribed to different types of content (via keywords) from each other.

Security, though overlooked in other systems like RSS, is something that was of great concern when developing OSP. As a result, OSP was designed from the start to incorporate multiple levels of security to ensure that the seamless incorporation of it would not inherently endanger subscribers or content-providers. In particular, OSP was designed to ensure that subscribers can be confident that received objects truly came from the content-provider they subscribed to, and that eavesdroppers or interceptors would be unable to recover the content from an OSP object.

Our work on OSP brings about several new contributions: (1) a unified mechanism for subscribing to ‘periodicals’ and ‘feeds’, (2) a unified format for encapsulating objects to be sent to subscribers, and (3) a proof-of-concept demonstrating that this mechanism is feasible (in particular for web-based applications).

1.3 Organization of the Rest of this Paper

In **Section 2** we cover OSP and its details, as well as suggested standards. **Section 3** addresses the role of authentication and a public key infrastructure (PKI) within OSP. Various use cases for OSP are given in **Section 4**. **Section 5** describes our prototype implementation of OSP. **Section 6** discusses related work, and **Section 7** concludes the paper.

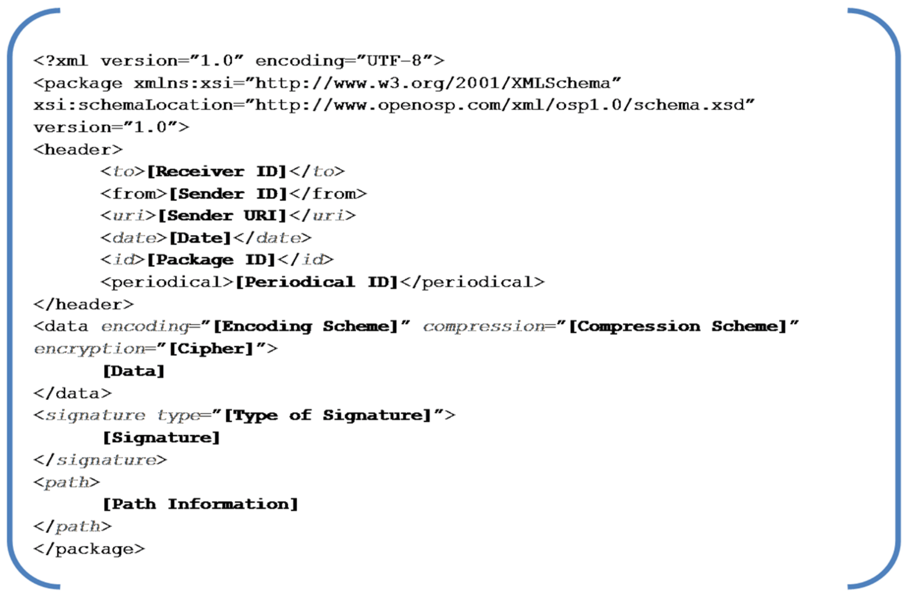
2 The Object-Subscription Protocol

The Object-Subscription Protocol (OSP) is designed to facilitate a simple yet robust and flexible method for securely and seamlessly transmitting and receiving data of arbitrary type between parties, primarily for the role of island-bridging between virtual communities. In this section, we provide a fairly comprehensive overview of the OSP protocol, though we omit certain implementation-specific details from this paper. Those interested can find the full specification online at `http://www.openosp.com/`.

2.1 Packages

OSP is based on the idea of passing messages in specially formed *packages*, between an *OSP server* and its *clients*. These packages are sent in a raw XML form, though support for compressed (e.g., gzipped) or binary XML formats (e.g., EXI from the W3C (2009)) are also possible. The basic structure of an OSP package is illustrated in **Figure 2**.

The package tag’s version attribute designates the version of OSP in use by the package sender. The header tag contains information about to whom the package should be delivered to, when it was sent, and from whom, etc; the data tag contains the package contents; and an optional signature tag would contain a digital signature or checksum information depending on the context in which OSP is being used. Any ID is considered a text field and may consist of a name (e.g., “Bob” or “RFG-3032”) or simply a randomly assigned number or bit-string. Recipient and sender IDs are relative to each pair - for example a client may be known by a different ID to a different server, and it is also possible for the same to occur for servers.



```
<?xml version="1.0" encoding="UTF-8">
<package xmlns:xsi="http://www.w3.org/2001/XMLSchema"
xsi:schemaLocation="http://www.openosp.com/xml/osp1.0/schema.xsd"
version="1.0">
<header>
      <to>[Receiver ID]</to>
      <from>[Sender ID]</from>
      <uri>[Sender URI]</uri>
      <date>[Date]</date>
      <id>[Package ID]</id>
      <periodical>[Periodical ID]</periodical>
</header>
<data encoding="[Encoding Scheme]" compression="[Compression Scheme]"
encryption="[Cipher]">
      [Data]
</data>
<signature type="[Type of Signature]">
      [Signature]
</signature>
<path>
      [Path Information]
</path>
</package>
```

Figure 2 The OSP Object xml format. Optional components are *italicized and in grey*, and variables are in **bold**. Optional variables are treated by both of these conventions.

2.1.1 Header

The header dictates how to handle the package to the entity that is in possession of the package. Its attributes are as follows:

- **To** – this field is reserved for the recipient of the package. If a server is writing to a client, then the client's ID would go here. Otherwise, the server's ID would go here. This field can be omitted in many circumstances (e.g., if the server is communicating directly with their clients), but in a broadcasting environment or one where packages may be forwarded on to the real recipient, this field would be necessary.
- **From** – this field is reserved for the sender of the package. This is mandatory as any client may be subscribed to multiple servers, and this would be the only way to identify them in an efficient manner.
- **Sender URI** – this optional field designates the URI by which a recipient (or anyone the package passes through) may be able to reply back to the sender. In the scenario that a package is being sent by a server, the clients may already know the URI from doing a look-up on the server ID, and including it may be superfluous.
- **Date** – this optional field designates the date and time-zone when this package was created (or "packed") by the sender.
- **ID** – this optional (but recommended) field represents the unique identifier for the package being sent. This allows for a recipient to reply back to the sender in the event of an error, responding directly to that particular package. If an

ID is generated in a predefined manner (e.g., sequential numbers), it may be possible for a recipient to re-request lost packages upon receiving a new one.

- **Periodical** – this mandatory field represents how the package should be treated or what category it belongs to. For example, a server using OSP may have multiple types of data that it pushes - differentiable only by the periodicals they are bound to (e.g., "magazines/popular-science/pdf" or "magazines/popular-science/tex")

2.1.2 Data

The data section of the package contains the actual data or message to be passed from the sender to the receiver. Its attributes are as follows:

- **Encoding** – this field determines what encoding is used on the data (if any). If left unspecified, it defaults to "none", otherwise it may be something like "base64" for Base-64 encoding of binary data.
- **Compression** – this field determines what compression algorithm (if any) is being used on the data. As with encoding, an unspecified value defaults to "none" - otherwise example values could be "gzip", "tar", or "7z" may be used.
- **Encryption** – this field determines the encryption algorithm used on the data. As with the previous two attributes, if this is left blank it defaults to "none". Common values here should be of the form "[Algorithm]-[Key-size and/or block-length]" where applicable - for example "rjindael-256" (AES) or simply "[Algorithm]" if the other attributes are fixed, such as in "gost".

2.1.3 Signature

The signature of the package is optional and allows for the sender to attach a signature to the package, such as a checksum or cryptographic signature (digital signature). It has the following attribute:

- **Type** – This determines what data has been signed. In particular:

    - *all* – The whitespace-free, and path-element-free XML data of *everything.*
    - *all-decrypted* – The same thing as "all", but with the data decrypted (if it is encrypted). This allows for the header and original signature to be passed along to a third party when forwarded, and allows a third party to verify that the package was originally from the initial sender.
    - *header-data* – The whitespace-free XML data of the header and data elements. This should be the practical equivalent of "all".
    - *header-data-decrypted* – The exact same thing as simply "header-data", with the data contents decrypted before signing. Like "all-decrypted", this is to facilitate package forwarding.
    - *data* – The data tag and its contents only.

- *data-decrypted* – The data tag and its contents only, *after* decryption (if encryption is applied). Like the previous two decrypted values, this is to allow the forwarding of a package onto a third party.
- *header* – The header contents only. This is useful, for example, if the data is encrypted as only a correct server ID would yield a correct decryption key for the data. Consequently, modifying the header would render the signature invalid – and a signature corresponding to a different header would result in a failed decryption.

2.1.4 Path

The path tag is optional and can be used by intermediaries (along a send path for a package) to leave messages or signatures for the next intermediary or the end-recipient. In a scenario where the sender and the receiver are only one "hop" away from each other in OSP, this is not necessary - but in situations where a package may be passed from one intermediary to another (e.g., forwarding), the signature can be useful in some applications.

Inside the path element, each intermediary may leave the following (a "*" before the element implies that there may be multiple elements of this type):

```
<received by="[Intermediary ID]" at="[Intermediary URI]" when="[Date]"
encoding="[Encoding]" compression="[Compression]" cipher="[Cipher]">
      <*comments for="[For Whom]" encoding="[Encoding]"
compression="[Compression]" cipher="[Cipher]">
            [Comments]
      </comments>
      <original>
            <header>
                  [Original Header]
            </header>
            <signature type="[Type of Original Signature]">
                  [Original Signature]
            </signature>
      </original>
      <*signature for="[For Whom]" type="[Type of Signature]">
            [Signature]
      </signature>
</received>
```

Figure 3 The structure of the OSP Object Path

A received tag specifies which intermediary receives it (the intermediary's ID should be relative to the final recipient, and not to the sender or anyone else along the path), and when it is received. If there is a need for additional data (e.g. comments, or OSP messages) to be passed along to the next intermediary, the last one, or the final recipient – there is room for this as well. Similar to OSP packages, an intermediary can sign for their receipt of the package, and even possibly encrypt their comments and messages – so as to ensure authenticity of their receipt.

2.1.5 Common Attributes

The following are common attributes for elements within an object's path section:

- **For** – this indicates to whom the comments are directed to. This can be "next" (next intermediary), "last" (last intermediary), or "final" (the client). If encryption is employed, the key used will be between this intermediary and the intermediary (or final recipient) that the "for" refers to. By default, it is assumed that the value is "all", and thus the comments are for everyone.
- **Encoding** – as with the encoding attribute for the data tag in an OSP package, this attribute carries the same meaning and values.
- **Compression** – this attribute borrows from the corresponding attribute for the data tag in an OSP package.
- **Cipher** – this attribute also comes from the corresponding data tag attribute.

2.1.6 Path Comments

Comments are an optional text-only field, allowing for an intermediary to put down human-readable messages. This can be useful for debugging or logging.

2.1.7 Original Element

The original element within a received element is where an intermediary can put the original header and signature content of the object that they received. Given an appropriate signature type (e.g., *-decrypted), this would allow the final recipient of a forwarded package to verify the origins of the package. When used in the context of an original sender signing the package, followed by an intermediary signing a forwarded package – the end recipient can verify that the contents (1) have not changed since being sent from the originator, and (2) that the intermediary really *did* receive the package and really was the one to forward it on.

2.1.8 Path Signatures

The signature tag for the received tag has similar properties to the signature tag in an OSP package. In particular, the following values are valid for type:

- "all" – The entire received tag and all its contents (whitespace-free).
- "all-for-you" – The entire received tag and all its contents (whitespace-free) that share the same "for" value.
- "received-only" – Only the initial received tag and its attributes.

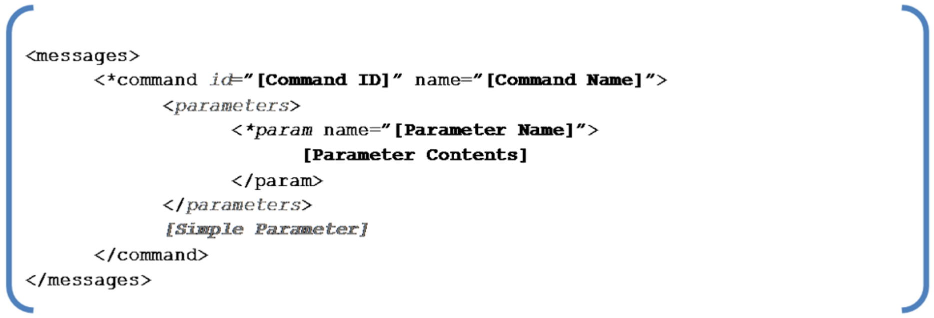
2.2 Client and Server Details

Each client has certain details specific to it (e.g. its URI, name, shared encryption key, etc...), as does an OSP server. These details can be altered and viewed through messages sent between the two, and are stored on the OSP server. In particular, details such as a client's URI allow a server to determine how to send the client objects – or more precisely, *where* to send objects to. Further details are available online[a].

2.3 Interface Messages

An interface (request) messages is an XML element (and its contents and attributes) which are designed to tell a recipient what commands the sender would like the recipient to execute. "Register me to Periodical X" and "Change my password to..." are example interface messages.

Interface (request) messages should be sent on a special periodical type, "!osp" - and take the form shown in **Figure 4**.



```
<messages>
      <*command id="[Command ID]" name="[Command Name]">
            <parameters>
                  <*param name="[Parameter Name]">
                        [Parameter Contents]
                  </param>
            </parameters>
            [Simple Parameter]
      </command>
</messages>
```

Figure 4 The structure of an OSP request message

2.3.1 Commands

The following are valid command types:

- "register" – Registers a new client
- "unregister" – Unregisters the current client
- "subscribe" – Subscribes the current client to a particular periodical type
- "unsubscribe" – Unsubscribes the current client from a particular periodical type
- "get" – Requests a specific OSP object from a specific periodical
- "list" – Requests a list of details pertaining to the current client (e.g., URI, username, etc...), some of which may be marked as alterable or hidden.
- "alter" – Requests to change a particular detail pertaining to the current client (e.g., passkey, URI, etc...).
- "info" – Requests information about the server.

[a]http://www.openosp.com

2.3.2 Response Messages

Responses messages should *always* be sent unless the OSP server has set the server-detail of "respond" to something other than "always". This may be the case if the server expects many command requests and does not want to uselessly reply to every one of them if there are no errors.

```
<messages>
      <*response id="[Response ID]">
            <error type="[Error Type]">[Error Description]</error>
            <detail name="[Detail Name]" alter="[Can Alter?]
hidden="[Hidden?]">[Detail]</detail>
      </response>
</messages>
```

Figure 5 The structure of an OSP response message

2.3.3 Error Types

The following are valid error types:

- "none" – No errors, everything went ok!
- "deny" – Request denied (e.g., insufficient permissions)
- "malformed" – Malformed request. The server did not understand the request (e.g., if a command does not exist, or if the request message itself was corrupted).
- "not-found" – The command issued attempted to get, alter, or [un]subscribe to something that does not exist.
- "fail" – The server failed to process the command for other reasons.

2.3.4 Client/Server Details

If a client or server's *details* are part of a command's request, then they can be provided as well. The optional attributes of "alter" and "hidden" default to "yes" and "no" respectively, and indicate whether or not a particular detail can be edited or seen by a client. For example, a password or encryption key may be hidden, but alterable.

2.4 Standards and Conventions

To conform to OSP standards, we provide a full specification online available at the OSP project's home page. There, we outline what encoding, encryption, and signature support we require, as well as other details.

In particular, while we did not specify any special properties of periodical names previously in this paper, we feel the need to point out that periodical names could be easily styled in a hierarchical manner and it is something we advocate. By this we mean that if there are two periodicals, "test" and "news" – someone who requests to subscribe to the periodical "/*" should be perceived as having made the request to subscribe to *both* "test" *and* "news". Likewise, an unsubscription from "/*" should be interpreted as an unsubscription from *all* periodicals. In doing so, we consider "*" a wildcard operator, and we recommend that "/" be used exclusively for defining hierarchies.

In a scenario where there are periodicals named "news" and "news/other", subscribing to "news" would *only* subscribe you to the "news" periodical. If one wanted to subscribe to *both*, then one would need to issue two separate commands to subscribe to them.

In a typical scenario however, an OSP server might automatically add objects to "parent" periodicals, thus eliminating the need to subscribe to "myperiodical/*" — however, we feel that this is something better left to individual OSP server operators at present.

3 Authentication and Public Key Infrastructure

While OSP *supports* authentication through digital signatures, this feature is useless unless there have been pre-shared public-keys of servers between them and clients. Given the nature of the internet and the application of OSP to it, we cannot expect this to be the case in most situations. As such, some form of public key infrastructure (PKI) is required to ensure that this feature of OSP can be used, and can be used in a secure (or trusted) manner. A more detailed overview of PKI is provided by Gutmann (2002).

3.1 Existing Infrastructure

An existing infrastructure for PKI already exists, in particular, for web servers and email (See Mozilla (2009)). This is based on the use of X.509 certificates which are already currently used for Transport Layer Security (TLS)/Secure Sockets Layer (SSL) encryption on the web. While it may be unrealistic to expect that current certificate authorities might issue special certificates for OSP services, it may be more reasonable to consider two other alternatives: (1) one could restrict registration to something done through web forms where data is passed via a SSL connection, or (2) apply a certificate issued for one task to OSP, leaving the client to decide whether or not it will accept the certificate.

Given that in many scenarios, it may not be totally unrealistic to expect a manual registration of clients to OSP servers: it may not be cumbersome to have an end-user copy-paste a pass-key and URI information into their client software, thus completing the transaction of registration, and allowing all future communication between both parties to be encrypted.

Alternatively, if an OSP server resides on a certain website, its use of a web server SSL certificate for its own purposes may not be warrant suspicion either.

4 Use Cases

In theory, any form of communication that can be seen as a “Don’t call us, we’ll call you” style of communication, can be translated into OSP. In particular, news services and blogs are the most obvious examples. Forums, even across different forum software, could also be merged in such a fashion. The following are some example use cases for OSP:

4.0.1 Forums

Suppose website A would like to merge their “General Discussion” forums with website B’s. Website A could subscribe to website B’s periodical for B’s forum posts and updates, allowing website A’s “General Discussion” forums to automatically receive the latest posts and threads from website B. This in turn allows users of website A’s forums to post and comment on the latest happenings over at website B’s forums without ever having to visit website B. If the subscriptions go both ways, then effectively there could be a ‘merging’ of forums, while both sites still retain individual policies and control (e.g., banning users, deleting threads, etc...). This could easily involve several different websites and forums, ultimately resulting in *different but shared* experiences across websites. Additionally, each subscribing website forum could configure whether or not they want to allow edits to be sent — whether they should be approved or not — and whether or not they want to allow thread deletes or various other settings.

4.0.2 News and Blogs

News publishers could utilize OSP to instantly disseminate the latest articles to other associated news websites, similar to how an article from Associated Press can be replicated on other news sites. One blog could subscribe to another blog and request all posts based on keywords. A post with certain keywords (e.g., ‘security’, ‘encryption’, and ‘law’) would then be put into specific periodical types (e.g., ‘keywords/security’, ‘keywords/encryption’, and so on) — and if a user was subscribed to any one of those feeds (or subset thereof) the post would be pushed to their blog as well. Similarly to the forum use-case, comments could be pushed to subscribed blogs too.

4.0.3 Social-Networks

Social networks are another potential area where OSP could be applied. From a participant’s perspective (**Figure 6**), the relevant spaces from different communities (e.g., MySpace, Facebook, etc..) form a personal social network. However, simply aggregating the communities together may be less straightforward than it sounds because of the ‘duplication’ issue. That is, assuming one participant had an advanced enough aggregator that they could use it as the *only* interface for all their social networks – if they were to change their mood or status on it, it would ideally change their mood or status on all social networks that the participant was a member of.

Suppose there are two such people who happen to be friends of each other and happen to be on more than one social network together. This means that one

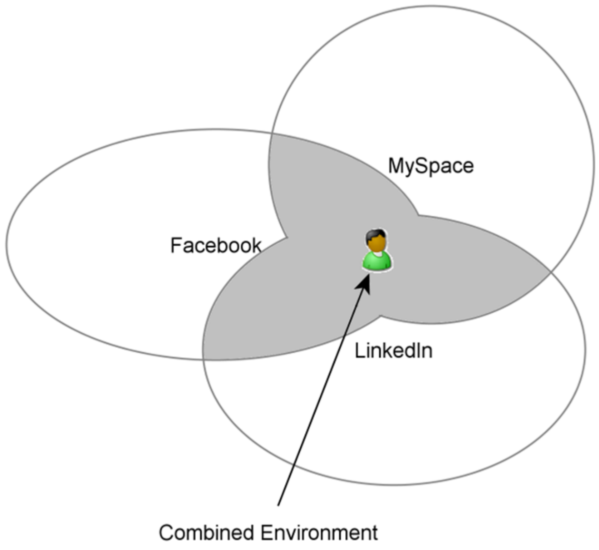


Figure 6 The intersection forms a personal social network.

party will run the risk of content duplication: the fact that their friend's mood on myspace changed will register as a different event than the fact that their friend's status on facebook changed — when in fact, they are the consequences of the same event. Given a unique identification ID for specific actions, which could be shared between social networks or with outside aggregators, this would facilitate links between a profile on MySpace or Facebook *without* the problem of duplicate information. Currently, there exist many social network aggregator services, but it is not clear if or how they resolve this problem, and this remains to be investigated in future work.

Besides being a potential tool for aggregators, OSP may also find itself useful as the underlying mechanism for certain types of social networks. For example, Kawash, et. al (2007) proposed a framework for a mobile social network, one based on entering physical regions that were 'hotspots' and being subscribed to certain communities. In this sense, OSP could both be used for updates on hotspot locations and community subscriptions *as well as* subscribing to individual 'hotspots'.

4.0.4 Mirrors and Update Delivery

OSP could also be applied to the realm of website and file mirrors. A mirror could be considered a subscriber to the original site. Should an update occur — or a new file to be uploaded (or an old to be deleted) — the update message and the data could be sent via OSP. Similarly, software developers that utilize multiple libraries could be subscribed to the distribution servers of those libraries. When updates occur, the software developers would automatically obtain the latest libraries in their own source code repository (e.g., a CVS or SVN), ensuring that when they release the next version of *their* code, it incorporates the latest libraries as well.

Likewise, OSP could also be used to deliver patches and updates for software, though it would seem likely[b] that this would be confined to server applications.

4.0.5 Media

While OSP does not seem to be suitable by itself for digital rights management purposes, OSP *could* deliver a stream of signed and authenticated — even encrypted — media, and its mechanism could easily be adopted to a broadcasting scenario. Despite the bulky XML format, the use of a binary representation of XML may mitigate this concern. Already some work towards this has been accomplished by Cuetos, et. al. (2006), and there is a binary XML format called EXI as suggested by the W3C (2009).

4.0.6 E-Commerce

OSP can be used both as a content-delivery system, as well as an update-notification system. Thus, OSP could be used to deliver e-magazines and journals to paid subscribers — as well as allowing users to select categories of products and be notified when an online store adds a new item to those categories.

4.0.7 RSS Delivery

Instead of having many clients querying a server for the latest RSS feed, the latest RSS feed items could be delivered by OSP. While certain measures can be taken (e.g., only downloading the RSS feed if it has been modified since a certain date[c]), there still may be many more queries for updates than there are updates. Furthermore, when there is an update, the entire RSS feed is downloaded, rather than just the newest items. Consequently, OSP delivery of RSS items should alleviate some of the load off a server.

4.0.8 Other Applications

It may be possible to apply OSP in game environments as well. For example, when sending a chat message in a multiplayer game – it may go only to your teammates. Likewise, certain actions (e.g., the amount of money you have, or how it has changed, etc...) may only be visible to members of your intra-game 'organization', and not to your enemies. Thus, it is possible to imagine several different types of periodicals and several groups of subscribers in a gaming environment.

Other possibilities may be using OSP to subscribe to certain types of companies (or specific ones) to receive all relevant changes in their stock prices in a real time manner. Due to the high traffic of stock market systems however, OSP may be more suited for the transmission of aggregated changes rather than individual ones.

[b]Due to the fact that end-users typically operate behind firewalls and changing IP addresses
[c]For example, using the HTTP GET request, with an *if-modified-since* header

5 Prototype Implementation

A prototype for OSP was developed as a proof-of-concept, primarily to demonstrate that there was an effective, painless, and seamless method to securely mesh blogs and blog software together. The prototype and the test consisted of posting between two WordPress blogs.

5.0.9 Overview

OSP can be implemented by a variety of programming environments and platforms. Our prototype PHP implementation of OSP provides a unique and easy interface to link existing PHP software to OSP. The prototype consists of a core "OSP Server" which would manage subscriptions it made as well as subscriptions that other OSP clients would make. The prototype was designed with modularity in mind with regards to object-handlers for specific periodicals — permitting a single OSP Server to manage incoming data from blogs, forums, or other types of software. It also features a PHP include file and a mechanism dubbed the "OSP Hook" by which any existing PHP-based software could be easily and quickly modified to support the Object-Subscription Protocol. The implemented prototype was used to allow cross-blog posting across wordpress blogs.

5.0.10 The OSP Hook

The OSP Hook system consists of using an include file from the OSP Server directory and filling a pre-defined variable with the content that you wish to send and other information, such as the periodical type to send the object to. Once the hook is included, the embedded code will automatically locate the variable in question, add it to a database that stores objects, and call a separate thread (the OSP Sender) to send the object to all subscribers. An illustration of its use is provided in **Figure 7**.

5.1 Architecture

The OSP Server software is partitioned into four sections: (1) the OSP Core, (2) the OSP Hook, (3) the OSP Sender, and (4) the OSP Receiver. The OSP Core provides all the shared functionality between the other parts of our server software (e.g., our OSP Object class), and can be considered an included file by the other three when they run. The OSP Hook functions as a quick include file which can patch existing PHP-based software and get them to utilize OSP. The OSP Sender functions as a program that will, for a given object, send it to all eligible subscribers. The OSP Receiver functions as a web-interface by which OSP Objects (and messages) can be received, after which it determines (based on the periodical) which module should process the data. The architecture of our prototype is illustrated in **Figure 8** and **Figure 9**.

```
function wp_insert_post($postarr = array(), $wp_error = false) {
[…]
      if ( $update)
            do_action('edit_post', $post_ID, $post);
      do_action('save_post', $post_ID, $post);
      do_action('wp_insert_post', $post_ID, $post);
      // NOTE: Now we call OSP Hook!
      if ('publish' == $post_status )
      {
            $this_post_user = get_userdata($data["post_author"]);
            $data = array("subject" => $data["post_title"], "contents" =>
$data["post_content"], "author" => $this_post_user->user_login, "date" =>
time());
            $osp_var = array("data" => $data, "periodical" => "default",
"stateless" => true);
            require("../../../osp/osphook.php");
      }
      return $post_ID;
}
```

Figure 7 OSP Hook code inserted into `/wp-includes/post.php`. The **bold** code indicates inserted code.

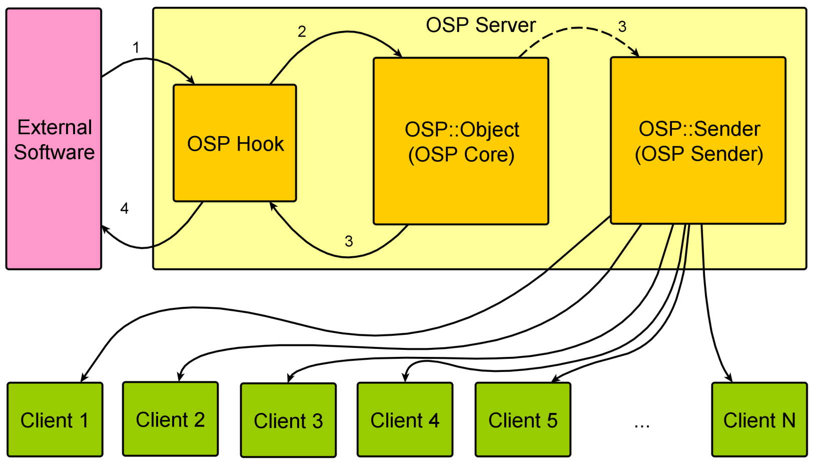


Figure 8 The architecture of the OSP Server from the sending perspective

6 Related Work

6.1 Syndication

The approach of Breslin, et. al (2005) to solving the problem of linking islands and syndication was to put forth a unifying structure for describing content across multiple media (email, forums, blogs, etc...), thus allowing one website (or 'island') to search across multiple other islands with related content. This would, in effect, bridge these islands together. Their SIOC ontology is described using the Resource Description Framework Schema (RDF Schema/RDFS). The authors have developed

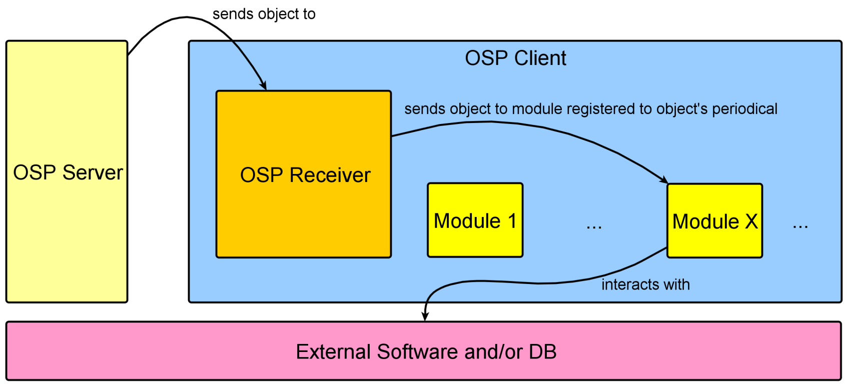

sends object to
OSP Client
sends object to module registered to object's periodical
OSP Server
OSP Receiver
Module 1
...
Module X
...
interacts with
External Software and/or DB

Figure 9 The architecture of the OSP Server from the receiving perspective

various tools to integrate their framework within multiple popular web services, including blogs like WordPress, forums such as vBulletin and phpBB, and even Drupal (See SIOC-Project (2009) and Breslin (2009)).

Work done by Halaschek-Wiener and Hendler (2007) and Golbeck and Halaschek-Wiener (2008) also describe syndication within the framework of a publisher/subscriber scenario, in particular with regards to determining what to send to a subscriber based on an expressive query for RDF documents. OSP could be considered to have a primitive approach to determine what to send to a subscriber in the sense that OSP could easily be seen to support a hierarchy of keywords to which a client may subscribe to. While it may also be possible to go a step further and support an attribute based approach (e.g., subscribing to "keywords=technology+web&minviews=20"), this is not the direction we are interested in taking OSP.

In particular, we do not aim to propose a common language to describe content as our method for linking websites, though we recognize that this is a valuable and important step in the process. Instead, we aim to propose a mechanism by which mutually-understood content could be *delivered*. While our approaches are different, they are not exclusive: SIOC and other systems could easily be used as the primary format for the exchange of data between blogs, forums, and so-on using OSP. More specifically, based on the syndication frameworks described by the previously mentioned works, OSP could function as a transportation mechanism between publishers, brokers, and subscribers – but for their purposes, would not be the replacement for deciding *what* content to send.

Superficially, OSP can be compared to a "reverse" RSS system: instead of clients polling a server for updates, the server goes to the clients and delivers the latest information. However, unlike RSS (or similar methods), OSP content is not limited to a specific format or structure. Content-handling is done by subscriber-end software that has been registered to a particular periodical, and an ideal implementation of OSP would manage this seamlessly.

As previously mentioned, web syndication methods (such as RSS and Atom) are the closest existing systems to OSP, however they are also fundamentally different. Syndication methods are concerned with *content* and less so with the method of *delivery*. In particular, RSS and Atom are designed primarily as web syndication mechanisms and may not be the best option for all types of objects that can have subscriptions. While Atom can theoretically support any diverse payload for content – and while it does support encryption[d] unlike RSS – both are used to broadcast a list of *all* entries, rather than simply putting out the "latest". Also, these web services operate on a "pull" mechanism rather than OSP's "push" mechanism. In a setting where there are both many more subscribers than there are content-providers and many more requests for new content than there are new content to "pull", a "push" based model can have its advantages. This was noted as early as Martin-Flatin (1998). Even though their work was only tangentially related to ours, Petrovic, et. al. (2005) recently highlighted some of the advantages of a "push" based model with regards to RSS. In particular, they write that "Existing RSS systems do not scale well. Anecdotal evidence suggests that websites hosting popular RSS feeds can be significantly overloaded with useless network traffic due to the architecture of the current RSS delivery systems [due to continuous polling by subscribers]."

6.2 Linking Islands

Pingbacks and trackbacks share some similarities to OSP but are substantially different. They are "push" mechanisms (either through plain HTTP, or XML-RPC) designed to notify content-providers that they have been linked to, rather than being used to push from content-sites to subscribers. Additionally, anyone can "push" to anyone, with the stipulation that the content-provider is astute enough to ensure that a real link has been made.

6.3 Aggregation

Social network aggregation services, such as PageFlakes, SocialURL, and others also share some similarities with OSP in that they attempt to combine several different views of different virtual communities into one combined and personalized interface. However, they are specific for the social networks they bridge and function purely as a centralized tool for end users - rather than a distributed system that could function as a tool for both end users as well as the applications supporting the virtual communities themselves. Finally, mash-ups also share some similarities to what OSP aims to do: they bridge specific web applications (and sometimes, virtual communities) together and provide an interface for this bridge to end users.

7 Conclusion

We have presented a unifying mechanism by which categories of arbitrary 'objects' produced by content-providers could be subscribed to by end-users. More specifically, we have proposed the Object-Subscription Protocol and have implemented a prototype of it that allows for cross-blog posting of content. We have argued that while virtual communities on the internet are like islands, there are

[d]Atom supports the XML Encryption standard at `http://www.w3.org/TR/xmlenc-core/`, which we believe to be too bulky and unnecessary for our purposes

many motivations for bridging these islands, and that OSP would be well suited to this task.

With future development of OSP and its potential adoption by individuals and organizations, we may see a rise in cross-site collaboration that has otherwise been elusive or otherwise uncommon and site-specific. Combined with the standardization of data formats, OSP may quickly become the underlying mechanism for island-bridging between blogs, news sites, and forums.

7.1 Future Work

With the current and future direction and development of both the protocol and server software, OSP has the potential to be the underlying bridge between varieties of island-types. While it is important that OSP is designed to fulfill this role, it is equally important that implementing OSP be made as easy as possible - something that the prototype takes into account.

We would like to continue the development of the OSP Server prototype and various applications that utilize it. In particular, further development of both blog and forum tools, as well as examining the role of OSP in a social-networking context. Ideally, we aim to turn the OSP Server implementation from a prototype to an enterprise-ready service that could be deployed on a variety of web-based environments, to help facilitate the use of OSP. In doing so, we also will likely examine the role of standards with regards to data representation – for example, utilizing SOIC for the representation of blog and forum posts as proposed by Breslin, et. al (2005).

While OSP has several strengths, it also has several limitations. The bulky representation of content due to XML verboseness would likely render OSP unattractive for broadcasting streaming media, and as such we would also like to examine the potential for switching the standards for future OSP versions to a another format. Conner and Mendelsohn (2003) showed in their presentation that compression of XML documents could yield enormous savings in space, and a format such as EXI which employs compression may be a viable option. Standardized support for a format such as EXI would however likely only occur after further ratification of the existing EXI draft and standards by the W3C.

References

Jean-Philippe Martin-Flatin (1998) ‘Push vs. Pull in Web-Based Network Management’, *Technical Report from the Communication Systems Division of the Swiss Federal Institute of Technology Lausanne.*

Peter Gutmann (2002) ‘Everything you never wanted to know about PKI but have been forced to find out’, *Peter Gutmann's Home Page, Department of Computer Science at the University of Auckland,* `http://www.cs.auckland.ac.nz/ pgut001/pubs/pkitutorial.pdf`.

Michael Conner and Noah Mendelsohn (2003) ‘Issues Relating to the Creation of a Binary Interchange Standard for XML’, *W3C Workshop on Binary Interchange of XML Information Item Sets.*

John G. Breslin, Andreas Harth, Uldis Bojars, and Stefan Decker (2005) 'Towards Semantically-Interlinked Online Communities', *Proceedings of the Second European Semantic Web Conference*, pp.500–514.

Milenko Petrovic, Haifeng Liu, Hans-Arno Jacobsen (2005) 'CMS-ToPSS: Efficient Dissemination of RSS Documents', *Proceedings of the 31st international conference on very large data bases*, pp.1279–1282.

Philippe de Cuetos, Claude Seyrat, and Cedric Thienot (2006) 'BiM White Paper', *International Organization for Standardization - ISO/IEC JTC 1/SC 29/WG 11 - Coding of Moving Pictures and Audio*, `http://www.chiariglione.org/mpeg/technologies/mpb-bim/index.htm`.

Jalal Kawash, Christo El Morr, and Mazen Itani (2007) 'A novel collaboration model for mobile virtual communities', *International Journal of Web Based Communities*, Vol. 3, No.4 pp. 427–447

Christian Halaschek-Wiener and James Hendler (2007) 'Toward Expressive Syndication on the Web', *Proceedings of the 16th international conference on World Wide Web*, pp. 727–736.

Jennifer Golbeck and Christian Halaschek-Wiener (2008) 'Trust-based Revision for Expressive Web Syndication', *Journal of Logic and Computation*, Oxford Journals.

John Breslin (2009) 'Social Semantic Information Spaces', *Semantic Digital Libraries*, Springer Berlin Heidelberg, pp. 55–68 ISBN: 978-3-540-85434-0

SIOC-Project (2009) 'SIOC Applications', *Semantically-Interlinked Online Communities*, `http://sioc-project.org/applications`.

W3C (2009) 'Efficient XML Interchange Evaluation', *World Wide Web Consortium*, `http://www.w3.org/TR/2009/WD-exi-evaluation-20090407/`.

Mozilla (2009) 'Included Certificate List', *Mozilla Security Projects: X.509 v3 Certificate Store*, `http://www.mozilla.org/projects/security/certs/included/`.