Network Working Group
Internet-Draft
Expires: August 22, 2002

M. Nottingham
February 21, 2002

Web Active Resource Monitoring
draft-nottingham-webi-warm-00

Status of this Memo

Copyright Notice

Abstract

WARM is a straw-man proposal for a solution to the RUP requirements of the WEBI WG which reuses the Web Architecture (and HTTP). In particular, it provides a mechanism for distributing cache invalidations from HTTP servers to clients.

Table of Contents

1. Introduction

WEBI's Resource Update Protocol requirements are broad enough that a wide variety of architectural styles could satisfy them. WARM is a straw-man proposal for RUP that concentrates on reusing the Web architecture. This approach has several advantages;

- Generality - The Web is most correctly defined as an information space, rather than the use of any particular protocol or format. A notification protocol that uses the same foundations as the Web (namely, resources identified by URIs and HTTP) will be able to make statements about any resource on the Web, not just a subset of them.

- Extensibility/Evolvability - WARM leverages the Web's properties of extensibility and evolvability, in turn providing them to applications that use it for notifications.

- Simplicity - A HTTP-based system is easier for Web publishers and HTTP implementors to understand.

- Ease of Implementation - Because WARM uses the HTTP, the cost of implementing on HTTP devices it is extremely low. Additionally, WARM will be able to use well-understood HTTP mechanisms like authentication, SSL/TLS, ETag validation, content negotiation, redirection, etc.

This document defines the WARM architecture and a cache invalidation payload for it. Please direct comments to the WEBI WG mailing list, webi@lists.equinix.com.

1.1 Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [2].

An implementation is not compliant if it fails to satisfy one or more of the MUST or REQUIRED level requirements. An implementation that satisfies all the MUST or REQUIRED level and all the SHOULD level requirements is said to be "unconditionally compliant"; one that satisfies all the MUST level requirements but not all the SHOULD level requirements is said to be "conditionally compliant".

1.2 WARM Resources

WARM provides for propogation of events concerning Web resources by defining two new types of resources; Channel Resources and

Subscription Resources.

Note that these classifications are conveniences; they are not fundamentally different from other kinds of resources on the Web.

WARM effects monitoring by transferring representations of the state of Channel Resources and caching it for a short period of time. If subscription-based monitoring is in use, clients expose a Subscription Resource, into which representations of the Channel Resource's state are copied. Clients using Polling-based monitoring will directly fetch a representation of the Channel Resource's state and cache it locally.

Representations of Channel Resources (whether polled or subscribed) have a freshness associated with them, which is functionally similar to HTTP cache freshness. When they become stale, any assertions made by the representations SHOULD be considered invalid.

On their own, these mechanisms allow the transfer of state representations in the channel and subscriptions to it; they do not describe what the representations of that state are. This specification defines one representation format that can be used to maintain coherence in an HTTP cache; other payloads may be defined in the future.

1.2.1 Channel Resources

Channel Resources characterize the state of an arbitrary grouping of Web resources. They are are identified by URIs, are accessed using the HTTP, and can be monitored either by polling or through subscription.

Traditional Web resources may be monitored using the same mechanisms; they behave as Channel Resources that characterize the state of only one Web resource.

For example, the Channel Resource
  http://www.example.com/channel;A

embodies a channel that contains the state of an arbitrary number of resources. Those resources may be under control of the same authority as the Channel Resource, or they may be elsewhere.

Informally, the semantics of common HTTP methods on Channel Resources are:

o GET - retrieve a representation of the state of the channel.

- WATCH - subscribe to the channel.

1.2.2 Subscription Resources

When a Channel Resource is subscribed to using the WATCH method, a reference to a Subscription Resource is provided in the Subscription HTTP request header. This allows the Channel Resource to maintain state regarding who is subscribed, and to locate them to perform operations related to the subscription.

For example, a Subscription Resource
  http://client.example.net/subscription;1

contains the state associated with a particular subscription to the Channel Resource. Subscription resources might be accessed through any number of protocols; this document only defines how they are accessed in HTTP.

Informally, the semantics of common HTTP methods on Subscription Resources are:

- GET - retrieve a representation of the state of the subscription.

- PUT - replace the state of the subscription.

- POST - update the freshness of the subscription.

- DELETE - terminate the subscription.

2. Overview of Operation

WARM's operation is composed of two different modes; discovery and monitoring.

2.1 Discovery

The appropriate Channel Resource for a given Web resource can be discovered either passively or actively. Discovery is OPTIONAL; some deployments may require out-of-band discovery, which is out of scope for this document.

2.1.1 Active Discovery

Active discovery is accomplished by performing a WATCH on the Web resource.

For example:

```
(request)
WATCH http://www.example.org/image.png

(response)
302 Found
Location: http://www.example.org/channel;A
```

Here, the location of the appropriate Channel Resource is found by examining the target of the redirect. The semantics of HTTP status codes in responses to active discovery requests should be honored as they are defined in the HTTP.

The Resource SHOULD be considered associated with the actively discovered Channel Resource until a subsequent WATCH changes the association, or the semantics of the Channel Resource's representation explicitly change the association.

2.1.2 Passive Discovery

Clients can passively discover Channel Resources by looking for the Channel HTTP response header;

```
(request)
GET http://www.example.org/image.png

(response)
200 OK
Channel: http://www.example.org/channel;A
...
```

The Resource SHOULD be considered associated with the passively discovered Channel Resource until subsequent representation has a different or missing Channel response header, or the semantics of a representation of the Channel Resource explicitly change the association.

[[[what about changes to the Channel Resource's state?]]]

2.2 Monitoring

Once the appropriate Channel Resource is discovered, its state can be monitored through the use of one of two techniques; subscription-based monitoring and polling-based monitoring.

Subscription-based monitoring allows notifications to be sent as quickly as network and other resource limitations allow them to be,

in combination with a heartbeat mechanism to assure that the channel remains available.

Polling-based monitoring can be used in situations where the Channel Resource is unwilling to maintain state about subscriptions, or where network conditions (e.g., a firewall) make it impractical to expose a Subscription Resource.

2.2.1 Subscription-Based Monitoring

Clients who wish to use subscription-based monitoring advertise this through use of the Subscription HTTP request header, in combination with the WATCH method. For example;

```
(request)
WATCH http://www.example.com/channel;A
Subscription: http://client.example.net/subscription;1
Accept: text/xml, */*;q=0.0

(response)
200 OK
```

A Channel Resourse SHOULD NOT return a successful status code to the WATCH method until it has initiated the Subscription Resource (with a PUT). If it cannot do so, it SHOULD return an appropriate client or server failure status code. In disconnected deployments, it MAY return 202 Accepted.

In subscription-based monitoring, the Channel Resource must first initialise the state of the Subscription Resource by PUTing a representation of the channel into it. Thereafter, the Channel Resource may update the state of the Subscription Resource with subsequent PUTs, and forceably delete the subscription with DELETE. Integrity of channel connectivity is assured by polling the Subscription Resource with the POST method and an empty entity body.

For example;

```
(initialise/update request)
PUT http://client.example.net/subscription;1
Cache-Control: max-age=600
Content-Type: text/xml
[entity body]

(initialise response)
201 Created

(update response)
200 OK


(heartbeat request)
POST http://client.example.net/subscription;1
Cache-Control: max-age=600
Content-Length: 0

(heartbeat response)
204 No Content


(delete request)
DELETE http://client.example.net/subscription;1

(delete response)
200 OK
```

PUT and POST requests to Subscription Resources SHOULD include a Cache-Control: max-age header. Its value is used to determine when the next heartbeat should arrive by; if a heartbeat is not received by its expiry, the Subscription Resource SHOULD be considered deleted.

[[[is this a good use of Cache-Control, or would it be more correct to define a new header?]]]

The semantics of HTTP status codes in responses MUST be honored. In particular, if any request to a Subscription Resource returns 410 Gone, the Channel Resource SHOULD consider the subscription canceled, and cease monitoring.

2.2.2 Polling-Based Monitoring

Clients using polling-based monitoring make periodic HTTP requests to the Channel Resource; the response represents its current state.

To ensure correctness and efficiency in polling-based monitoring,

Channel Resources MUST support ETag validation. Channel Resources SHOULD use the Cache-Control response header for GETs to declare how long that representation of the channel should be considered fresh (and therefore, how long before the client should poll again).

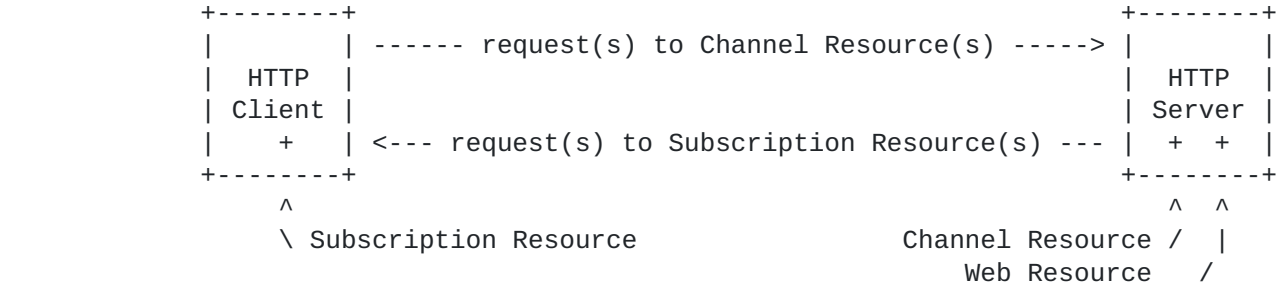
For example;

```
(request)
GET http://www.example.com/channel;A
If-None-Match: "abcde"
Accept: text/xml, */*;q=0.0

(response)
304 Not Modified
Cache-Control: max-age=600
ETag: "abcde"
```

3. Relationships to Network Nodes

Because they are located by URIs, Channel and Subscription Resources may be located on any addressable network node. However, it may be helpful to illustrate a typical implementation;



This illustration should not be construed to limit the location of a Channel Resource to the network node on which the resource(s) it characterizes reside, or to prohibit the location of a Subscription Resource on a node other than the HTTP client. Indeed, there are many scenarios where it is beneficial to do so, for purposes of scalability, managability, assertion of administrative control, or for disconnected operation.

4. Using WARM for Cache Invalidation

WARM may be used to maintain coherence of cached representations. In this application, the payload of notifications is a simple XML document identified by the application/xml media type, using a single element, 'cache', in the WARM cache namespace;

```
<?xml version="1.0"?>
<cache xmlns="http://www.intermediaries.org/warm/1.0/cache">
abcdefg
</cache>
```

The content of the element is a nonce generated by the Channel Resource to identify its current revision level; it MUST be guaranteed by the Channel Resource to be unique in its scope. When any Web resource associated with the Channel Resource becomes stale, the Channel Resource state SHOULD change.

This implies that the cache will track the mapping between Web resources and Channel Resources and/or Subscription Resources, so that when notifications are received, the appropriate representations can be marked stale. Web resources that are associated with such WARM Channel Resources SHOULD be considered fresh until such a notification is received, the channel is deleted, or connectivity is lost.

WARM can also be used with other cache-related payloads; their semantics, interactions with cache behaviour, and additional association mechanisms are format-dependent.

For purposes of content negotiation, the media type of this format is "[[[TBD]]]".

5. IANA Considerations

5.1 The WATCH HTTP request method

The WATCH method is used to associate a Subscription Resource with a Channel Resource, or to locate an appropriate Channel Resource. If a Subscription Resource is associated, regular requests containing heartbeat and/or update messages (as described above) will be made to it.

watch = "WATCH"

WATCHing a resource may instigate one or more requests to subscription resources, if subscription-based monitoring is in use (as evidenced by a Subscription request header). If content negotiation is used to determine the representation sent in response to the WATCH, the same representation SHOULD be sent in subsequent PUTs to the Subscription Resource.

Implementations SHOULD interpret HTTP status codes resulting from WATCH as defined in RFC2616. Web resources that are not Channel Resources MAY return 303 See Other to direct clients to the

appropriate Channel Resource, which may then be monitored by polling or subscription.

WATCH requests MAY contain an entity-body; however, this document does not specify a format for them.

5.2 The Subscription HTTP request header

The Subscription request header is used to indicate the URI of the Subscription Resource that the client wishes to associate with a Channel Resource.

subscription = "Subscription" ":" URI

5.3 The Channel HTTP response header

The Channel response header is used to indicate the URI of the Channel Resource associated with the Web resource.

channel = "Channel" ":" URI

6. Security Considerations

WARM uses the same confidentiality, integrity, authorization and authentication as HTTP does. Therefore, the use of SSL/TLS, the Content-MD5 header and HTTP authentication mechanisms are encouraged, and support for them in implementations is RECOMMENDED. Such issues are relevent to both Channel Resources and Subscription Resources.

WARM allows Channel Resources to make statements about Web resources in other administrative domains. Client implementations SHOULD be aware of the impliations of this, and be conservative in their trust of such statements.

Certain modes in WARM imply non-trivial resource use by either the client or the server; implementations SHOULD limit their use through techniques such as increasing the lifetime of representations (through the Cache-Control header), limiting the number of clients accepted, etc.

References

[1] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", 2000, <http://www.ebuilt.com/fielding/pubs/dissertation/top.htm>.

[2] Bradner, S., "Key words for use in RFCs to Indicate Requirement

Levels", RFC 2119, March 1997.

Author's Address

Mark Nottingham

EMail: mnot@pobox.com
URI: http://www.mnot.net/

Appendix A. Acknowledgements

The author would like to thank Paul Prescod for the spark that led to WARM, Roy Fielding for his description of the REST architectural style [1] that it is built upon, Mark Baker for his insight and patience in explaining and applying REST, and Rohit Khare and Adam Rifkin for their overview of Internet-Scale Event Notification Services.

Any error, misconception or bad design in this document is the responsibility of the author, not them.

Appendix B. Issues/TODO

- Discuss WARM Intermediaries, to scale to large deployments.
- Formalize the cache and state models.
- how does a client specify authentication credentials for the Subscrition Resource?

Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.