Coping with Un-Cool URIs in the Web of Linked Data

John Arwe
IBM Corporation

Contents

Problem Statement

1. Linked data's assumptions are based on HTTP URLs being immutable.
2. Despite everyone's best-laid plans for Cool URIs, production URLs sometimes do change.
3. On the Web, broken links are often "good enough"; humans know how to compensate for broken links. In the Web of Linked Data, broken links violate the principle of providing useful information when a URI is "looked up".
4. We are dealing with a distributed ecosystem of components. Those components have different abilities to respond to URL changes, and different incentives to invest in responding.
5. There are trade-offs amongst performance, availability, responsiveness, etc. based on the usage scenario.

Immutable URLs

1. URIs are resource identifiers (uniquely identify the resource) – this is what confers the requirement for immutability.
2. Linked Data wants URIs to be HTTP URIs that people can "look up" using standards; i.e., it wants them to be URLs that support at least HTTP GET. So the URI used to identify a resource also gives its location, making it a URL. In particular, an immutable URL.
3. URLs that both do not change and are dereferencable require generally some level of effort (virtualization) to achieve that illusion for the HTTP clients. Examples include, but are not limited to:
    a. Use of hostnames instead of IP addresses: by using a mapping service (DNS) to translate from hostname to IP address, the physical location expressed by the IP address can actually change without affecting HTTP clients.
    b. Virtual IP addresses (VIPA and DVIPA): by using redirection within the IP layer (a mapping similar in spirit to DNS), multiple back-end IP addresses can respond to what HTTP clients think is a single IP address.

c. DNS aliasing.

URLs Do Change

1. Needs evolve: successful proof-of-concept systems are pressed into wider use; department-level systems grow into corporate systems with different quality of service needs that require different deployments.
2. Domain name ownership changes. Acquired organizations' names are retired, and there is a (small but non-zero) economic incentive to release unneeded domain names. Sometimes there is a legal requirement to do so.
3. Some URLs are poorly constructed to begin with, in that they include components that lead people to *want* to change them over time. They include brand names, or version numbers, that are really mutable properties of the identified resource. Some organizations are simply used to being able to change URLs because their current consumers are human-attended user agents rather than fully automated/autonomous processes.
4. Organizations out-source control of their network environments; exerting control of the sort required to control DNS entries or issue 301 redirects essentially becomes a legal process, ill-suited to solving technical problems.

Distributed System Components

1. Humans: the least controlled components. They can write down URLs on napkins, type them in by hand, email them.
2. Applications: implementations that play one or more of the following roles.
3. HTTP clients: wide variance, varying degrees of control, each "does whatever it does" and in the short term URL authorities have to adapt to the behavior of their installed base. Clients use HTTP resources exposed by HTTP origin servers; they include browsers, the most common software component that humans use to consume URLs, which all store bookmarks (URLs). An application
4. HTTP intermediaries: proxies, caches, reverse proxies; some transparent, some not. Typically there are fewer implementations of intermediaries, and therefore a bit more control over their behavior but only in the long term. In the short term, switching costs and the business risk of change limits the flexibility of organizations to change implementations.
5. HTTP origin servers: expose HTTP resources for use by others. Similar issues as intermediaries.

Classes of URL-Handling Behavior

1. Client: Redirect-aware
    a. Processes 301 Moved Permanently redirects.
2. Client: Redirect-caching/persisting
    a. Updates any local "bookmark"/cache/persisted URL from any 301 redirects.
3. Client: Consultative

a. Aware of the existence of some external link authority (a link oracle) that has knowledge of URL changes that (usually for legal or organizational rather than technical reasons) are not responded to with 301s, and is capable of essentially "repairing" links by returning the equivalent current URL given a broken link.

b. Link oracles are likely to be implementation-specific, so it affects the client programming model; humans use search engines; peering and quorum-based systems are also possible. Not all organizations/deployments will have one.

4. Client: Link-mappable

    a. Translate URLs in its incoming *and* outgoing messages.

    b. Capable of consuming mappings describing relocated resources' URL patterns. Mapping URLs raises a number of questions

        i. Atomicity of origin server resource movement (all at once atomically, or "trickle" of resources from old to new location) as observed by its clients, and how that affects the contents of mappings.

        ii. Duration of the mappings, which may need to vary depending upon whether the old location is re-purposed after some period of time (after which it might be incorrect to blindly map to the new URLs) or not.

5. Origin server: Cool

    a. Origin server allows configuration of a stable URI prefix that is used in all responses, regardless of the URI prefix any client uses in its request.

6. Origin server: Movable/Relocatable

    a. Origin servers may need to consider questions of persistence for the set of resources each owns – how are links within their scope of authority persisted – depending on their quality of service requirements.

    b. Origin servers may need to consider questions of message processing and translation in order to be movable.

        i. Input representations, e.g. on PUT.

        ii. Output representations, e.g. on GET.

        iii. Input parameters, e.g. on queries over a collection where the query parameters may include URLs.

    c. Origin servers whose resources have strict audit/compliance requirements likely require special handling.

7. Origin server: Cloneable

    a. Cloud deployments are often based on the assumption that snapshots can be taken of running software and deployed elsewhere multiple times.

What is the set of tools available to draw from?

These are composable, not mutually exclusive. They assume that in the near term, client applications cannot be changed. Since even with client application change added as a degree of freedom we still need to tolerate legacy client applications, we end up in the same place effectively.

1. Organization: Plan for durable URLs on all software tool deployments up front
    a. Simply put, avoid the problem
    b. Use DNS, (D)VIPA, etc.
    c. Configure origin servers with Cool URLs
2. Organization: Deploy redirect server at old location responding with 301 Moved Permanently and new URL.
    a. Web standard
    b. Works for “any” web client
    c. Organizations not always able to do so, e.g. when ceding control of domain name.
3. Organization: Provide knowledge of old-new mapping to each link-mappable component
    a. Would benefit from standardization of the mapping format.
4. Origin server: support relocation and/or cloning
5. Origin server: Advertise its old location(s)
    a. List of “formerly known as” values + current comprises a set of URL mappings consumable by link-mappable components.
    b. Would likely need standardization of the mapping format.
    c. Removes need for humans to fiddle with files/UI in common case of “moved server”.
    d. Does not cover case where old hostname is re-used for new component that is part of ecosystem unless human consent is part of the process. The moved origin server needs to ask: should I share this URL mapping from “old” to “new”?
6. Applications: Change the client programming model
    a. If your link is broken [and current standard Web methods like 301 handling are insufficient], consult a link repair authority.
    b. This would require change across both application-consuming organizations (to deploy the link repair authority components) and application writers, so it is likely to occur relatively slowly in general. There may be specific pockets however which adopt more aggressively, e.g. those where higher qualities of service are required and Linked Data is used in the implementation.
7. Standards-defining organizations: add informative statements in specifications
    a. As is current best practice in areas like security, listing the issues is a fairly light-handed way to get implementers thinking about the problems.
8. Standards-defining organizations: add normative statements in specifications

a. Saying for example that origin servers for a given set of resources SHOULD be movable.

b. This may become more acceptable later, but it is probably too early at this point in time. Assuming the adoption of Linked Data continues to grow, the issues of non-durable URLs will become more widely known so the need for a more organized approach to a solution will become more urgent to a wider audience.