CENTRAL LIMITATIONS OF THE WEB SERVICES MODEL

Robert F. MacInnis, Ph.D.
June 2017

AetherWorks
501 Fifth Avenue
New York, NY 10017

E: info@activeaether.com

1 RIGID ADDRESSING MODEL

Web Service descriptor documents detail the data types and operations offered by a Web Service. These documents also include a fixed location for the Web Service, described using a URL – a static physical endpoint which, for any number of reasons, maybe temporarily (or permanently) unavailable. The inclusion of a fixed URL in the Web Service descriptor document has far-reaching implications for all participants in a distributed Web Services framework, from developers of Web Services and administrators of Web Service hosting environments, to programmers and users of the applications which utilize them.

Programming against fixed endpoints results in tightly-coupled applications which are fragile with respect to changes in the location and availability of Web Services and are susceptible to the failure of a single host or service. If a Web Service fails or is migrated, users must either re-write code to reflect an updated endpoint, or include failure recovery mechanisms in each implementation of a client which uses a Web Service. The procedures for re-locating migrated Web Services (or locating alternative instances of a failed Web Service) are not specified by a Web Service standard, leading to the development of ad-hoc, application-specific failure-recovery techniques which are themselves susceptible to failure. This approach produces applications which are permeated by exception-handling code and tightly-coupled to both the location and the mechanisms used to locate Web Services. Due to the lack of standardized endpoint location procedures, existing work-arounds or 'advancements' are always proprietary, yielding non-portable, tightly-coupled, and platform-specific client applications which, in one fell swoop, mitigates all of the benefits of the Web Service model.

Providers of Web Services are also burdened by the rigid addressing model. To make Web Services locatable by clients, Web Service providers publish the location of active Web Service endpoints in a directory. Clients use this directory at design time to discover and write programs against Web Services. If service providers aim to provide a reliable and consistently available Web Service, the address of the endpoint must remain fixed, forcing an early decision on the location, amount of bandwidth, and amount of computational capacity required in the hosting infrastructure. In open-world systems, selecting an adequate provisioning level is difficult as it is difficult to predict demand levels before a service is deployed. Further, fixed-resource deployments are not scalable to demand and waste useful resources by statically provisioning them before deployment, either over- or under-shooting the necessary capacity. While dynamic deployment environments provide benefits of balanced resource consumption and high availability they typically present a closed-world view, assuming discrete capacity and highly reliable connections. Further, applications which utilize Web Services hosted in dynamic deployment environments must incorporate run-

time dynamic binding techniques which are specific to the host environment. This solidifies the already tight coupling in client applications by tying them to not only the location of the Web Service, but also the location and implementation of an endpoint directory.

2 SERVICE PROVIDER ROLE

The traditional Web Service model defines a 'Service Provider' actor which encapsulates all of the tasks of developing, deploying, hosting and managing a Web Service. Because of the implicit complexity of handling so many tasks, hosting environments are often built using proprietary 'integrated technology' packages: end-to-end platforms for developing, publishing, deploying, hosting, and managing services in homogeneous, fixed-resource infrastructures. The result of this broad set of responsibilities is the proliferation of proprietary deployment systems and hosting environments composed of services which cannot be deployed or managed by other systems and are only interoperable with other identical environments. The resulting technology lock-in is contrary to the central ethos of platform-independence which is otherwise a hallmark of the Web Services model.

While Web Services themselves are platform-independent, the techniques used to deploy and manage them typically rely on particular platform-specific attributes or tools. In theory, developers of Web Services may use the most suitable platform for a new service, unencumbered by potentially restrictive technologies. In practice, however, this is not the case, as deployment systems are largely proprietary and hosting environments largely homogeneous. Platform- dependence forces developers to develop for specific target environments – a closed-world approach which results in islands of incompatible services and collections of service implementations which are not portable.

Without clearly-defined sub-roles with concrete responsibilities, it is difficult for a Web Service provider to evolve their processes into more mature or automated systems. While tightly integrated systems can provide for efficiencies in design, the customized and often ad-hoc procedures devised in monolithic Service Provider implementations pose significant barriers to system evolution.