ANATTA

WHITE PAPER

Contents

0. Introduction 1

1. The Web 3

2. Web Net Model 5

3. Web System Configuration 6

4. Web Architectural Style 8

5. Anatta Engine(s): `Engine` 11

6. Anatta Engine(s): `Links` 18

7. Anatta Engine(s): `Network` 22

8. Anatta Engine(s): `Prototype` 25

0. Introduction

The inception of the World Wide Web (WWW) occurred about a quarter of a century ago. WWW is based on a loosely-coupled architecture of inter-networked hyperlinks and provides a uniform interface to all users. Web features such as OneWeb and the REST architecture have helped WWW grow.
As a user agent, the browser employs hyperlink navigation to process any document or data, irrespective of its source location. This feature, which requires no control or regulation for documents to coexist, is a major innovation that enables users to create and link documents and have them be freely browsed and processed.

We have adopted an architecture that is in contrast with the traditional systems based on RPC and connection/data-oriented APIs. The proposed architecture is founded on the REST architectural style, where programs can interact with each other asynchronously while processing hyperlink navigations. When these programs are distributed over a network, systems comprising these programs would possess the same relationships as hyperlinks do with web pages.

Therefore, a new programming environment is required that can effectively process hyperlinks. In this new programming environment, distributed programs could acquire the property of loose-coupling, which is one of the primary advantages of the Web. Anatta Engine(s) provides a functional environment for loosely-coupled programs.

Anatta Engine(s) functions in the same manner as browser process documents and all programs are processed on the user agent's system. It provides a "universal message bus" that helps programs interact openly with each other on the Web. By employing Anatta Engine(s), users can customize web-based processing and build systems that combine these processing operations. This epochal innovation of the Web, led by users, may have a significant impact on programming systems and documents.

Anatta Engine(s) applies the property of WWW's decentralization to not only system data and programs, but also to devices. While interacting with events on the Web, all of them could be congregated, despite being dispersed over a network.

WWW could face the crisis of fragmentation caused by the systems associated with user accounts, RDBs, and other traditional systems. This fragmentation could be prevented by employing Anatta Engine(s), which in turn could revive OneWeb services.

1. The Web

At present, we use browsers to view news stories, read blog posts, peruse the latest weather information, purchase appliances and clothing at online stores, and share information regarding our activities and whereabouts. Such activities using browsers are enabled by the World Wide Web (shortened to the WWW or, simply, the Web). The Web is a powerful channel by means of which almost all activities that can be conducted on computers can be made available on the Web.

Although, in general, the Web is often viewed as being synonymous with the Internet, they are not the same. The Internet is a network of computer interconnections that connects millions of machines all over the world, enabling them to communicate with one another. The Internet is the underlying infrastructure that enables the Web to be accessed from around the globe.

In systems (e.g. mail, instant messaging, and VoIP) dependent on computer connections such as the Internet, system elements are usually mapped directly onto connections between individual machines. The layout structure of upstream system elements is determined by the layout structure of the individual machine.

The Web, by contrast, is comprised of a hypertext system that uses hyperlinks embedded in web pages to directly connect to other web pages. Selecting a link on a web page that is displayed in a browser causes the associated web page to be displayed.

An important feature of the Web is that the machine on which a web page is directly connected by hyperlink does not have to be directly connected on the Internet. In other words, although the Web is a system realized on the Internet, it is possible to configure free structures from the connections between machines on the Internet, but also from the individual machines themselves.

The browser is what makes it possible to use the link structure described by a hyperlink while accessing the Internet. The connection on the Internet is a simple structure in which there is only

a connection to each machine on which the browser is running and the machine on which the web page associated with the link destination is present.

One of the strengths of the Web comes from the flexibility that this structure, called a 'hyperlink,' allows. In addition, the fact that hyperlinks can constitute a complex structure in spite of having the simple connection structure described above, is one of the key reasons for the Web's scalability.

Structures based on hyperlink connections on the Web produce 'loose coupling,' which is also one of the strengths of the Web. 'Loosely coupled' refers to a property in which mandatory pre-existing conditions for both ends of each interaction are reduced to achieve the interactions. Hyperlinks on the Web do not have place and time restrictions like those of an Internet connection, and they are unrelated to the structure and format of other web pages. Loose coupling makes it possible for the link source and destination to create and update web pages gradually, independent of each other's content, producing the overall growth and easy updating of information on the Web.

Through hyperlinks, the details of a web page as an element in a machine can be connected to other details, enabling the Web to take on a structure that is sufficiently complex to perform almost all of the activities for which computers are used.

However, it is difficult to take purpose-oriented information systems arranged on individual servers on a computer network that existed before the Web, scale them for the Internet, and then superimpose them there. Even though the systems may be connected to the Internet, they have elements in the machine that cannot be connected with hyperlinks, so they cannot take complex structures with a simple connection structure.

2. The Web Net Model

In addition to web pages, hyperlinks point to images and videos. In some cases, there may be no data at the link destination, giving rise to error messages such as "404 Not Found."

In order to handle these collectively as hyperlink targets, the target pointed to by a hyperlink is referred to as a "Resource" to differentiate it from the data that is actually obtained.

A “Resource,” related by hyperlinks on the Web form a graph structure, which makes it possible for the browser to process the hyperlink relationships as networks of “Resources,” based on the graph structure (as mentioned in the previous chapter, despite the fact that a machine with a Resource may not actually have a connection, a browser may still handle it as though it is connected).

In such a case, in order to associate the resource with a hyperlink, a "representation" for indicating the Resource itself is necessary. The representations are associated with their Resource by Universal Resource Identifiers (URIs) or Universal Resource Locators (URLs).

The most important characteristic of a URI is its universality, defined as meaning that it is applicable to a Resource anywhere on the Web. In addition to having a standard format, each URI on the Web is handled as referring to only one Resource at any time or place.

This universality makes the Web a single graph structure throughout the world. As a result, people all over the world have participated in and shared the one Web, making it the huge network it has become.

3. Web System Configuration

Here, we outline the standard specification of the Web from the viewpoint of realizing universality.

A Resource can be identified from any browser by a URI representing a hyperlink. HTTP is the standard connection protocol used on the URI based Internet to access the data of the Resource indicated by the URI. The data of a Resource has to be able to describe a hyperlink to another Resource, for which the standard format used is HTML.

In terms of the Internet, HTTP is one of its protocols and HTML is one of its data formats. For the Web, however, HTTP and HTML exist primarily for realizing URI based hyperlink connections and are the default for forming a single graph structure that can be universally utilized. Means unrelated to hyperlinks such as ftp, json, xml, jpg, etc, are not even an option.

HTTP enables Resource data to be accessed from the URI. The accessing side is called the "client," and the other side that manages the relationship between the Resource and its data and passes that data via HTTP is called the "web server."

The HTTP client side cannot use the resource graph expressed by hyperlinks merely by receiving data matched a URI through the communication connection to the web server. In order to use hyperlinked Resources as a Web, it is necessary to recognize the group of hyperlinks included in the received data, and to repeatedly connect across to other web servers managing the data of the linked resources (referred to here as "link navigation"). The program that performs this processing is called a Web User Agent ("User Agent").

In other words, only "link navigation" programs that is based on the process of User Agent can process the Web as a network of hyperlinks. An ordinary communication program processes communication and data, but it does not process these as a Web.

A browser is a User Agent that provides a user interface to the Web. As a User Agent outside the Web, in accordance with a user’s input, such as when the user clicks on a link, or with

automatically loading images, style sheets, scripts, etc., the browser create a connection between Resources based on hyperlinks.

Basics of User Agent as processor of Web is a simple mechanism of hyperlink navigation. With this simple mechanism, every system exists as one Web, and it does not require to be controlled to coexist.

4. Web Architectural Style

The Architectural Style of the Web called "REST" established the simple mechanism of "link navigation," which is the basic principle underlying a User Agent.

REST is defined as a set of properties, the most important of which is the “Uniform Interface.” In the Uniform Interface, all objects in the system have the same general-purpose interface specification and all interactions are done via that interface.

The fact that the same interface is applied for everything means that there is no need for preliminary adjustments to individual objects or methods, based on specialized knowledge. In all cases connections can be established is the premise.

On the Web, URIs and HTML, as well as HTTP, are based on this property. Whatever the activity performed on the Web, whatever the information on the Web is, the same method is used for the interaction methods and data formats between browsers and web servers. This Uniform Interface is what makes connections, established from hyperlinks based on the URI, possible.

Conversely, the interaction methods and data formats specialized for types of activities and information do not satisfy the Uniform Interface characteristics, and therefore cannot be applied for "link navigation."

The name REST is the acronym for "Representational State Transfer." In Chapter 2, it was explained that a Resource identified by a URI and data obtained from web servers are differentiated from each other. In the latter case, the HTML and image data obtained from the web server is called a "Representation." What is "transferred" by this "Representation" is the current "State" of the Resource.

What this means is that the Resource does not change as the target pointed by the URI, but the state of the Resource can be changed. In other words, even if the state changes, the fact that the meaning as a resource does not change is a condition for establishing the URI. That is, the URI

must be persistent. This means that when a Resource is linked to another Resource by URI, in addition to there being no change in the meaning of the URI destination, there should be no inconsistency for the link source from changes of its destination.

The universality of this URI makes it possible for the link source and link destination to each autonomously change progressively while the relationship established on the Web remains. If the universality is not guaranteed, it would be difficult to change the link destination and also be difficult to change the link source itself because the link source must, synchronously, cope with the change of the link destination.

To summarize the above, REST is a world in which any kind of object can be connected with the same specification. It is possible to arbitrarily make individual links not subject to control or permission, and it is possible to have individual link objects change autonomously.

There is no doubt that the possibilities inherent in REST have promoted the growth of the "One Web."

On the other hand, the fragmentation of the Web, due to so-called Web Services that diverge from REST, continues to increase.

Web Services provide functions to users in a form that binds their data. Functions are implemented as a program dealing with traditional databases which is contrary to the architecture of REST, and mainly only the result values are provided as a Resource in the form of web page. On the other hand, both the data not provided as web pages and the intermediate forms of the data processing are hidden in the walls of Web Services, and as such cannot be traced with the simple mechanisms described to this point.

This is because it is difficult for a Web Service to provide data and intermediate processing as a Resource. The objects processed by the program are an internal database, and the intermediate processing forms are functions or internal data values. They cannot be treated as universal Resource directly.

These arrays of walled-off Web Services seem like a universe of isolated islands. With this separated structure as the premise, users will try to be satisfied by bringing everything into one of Web Services or choose the several Web Services to use with separating their own data.

A third party that wishes to handle data inside a service wall has to connect by using specialized methods for each individual Web Service, that adheres to an API specification.
Data is acquired by the network connection as a HTTP client from the URI conformed to the API specification, then processed as a data structure specified in the API specification. Even if the URI is included in the data structure, access to that URI must similarly be processed based on the API specification, including the communication methods.

In this way, in such a Web Service, the Web, which is the only diverse structure realized by hyperlinks able to link directly to individual Resources on web servers, becomes a legacy Internet-style structure in which the relationship is limited between web servers by their API connections. This is how the Web is being fragmented by Web Services.

In Web Services in recent years, the introduction of AJAX has dramatically improved the user experience. AJAX is a mechanism which separates the synchronous Internet style API access of Web Services for the State of Resource and the asynchronous interactions of the user interface on the User Agent side.

However, the AJAX-based development is in the User Interface; the architecture of other parts such as the web servers remains conventional. Because of this, even AJAX has not stopped Web fragmentation caused by Web Services.

5. Anatta Engine(s): `Engine`

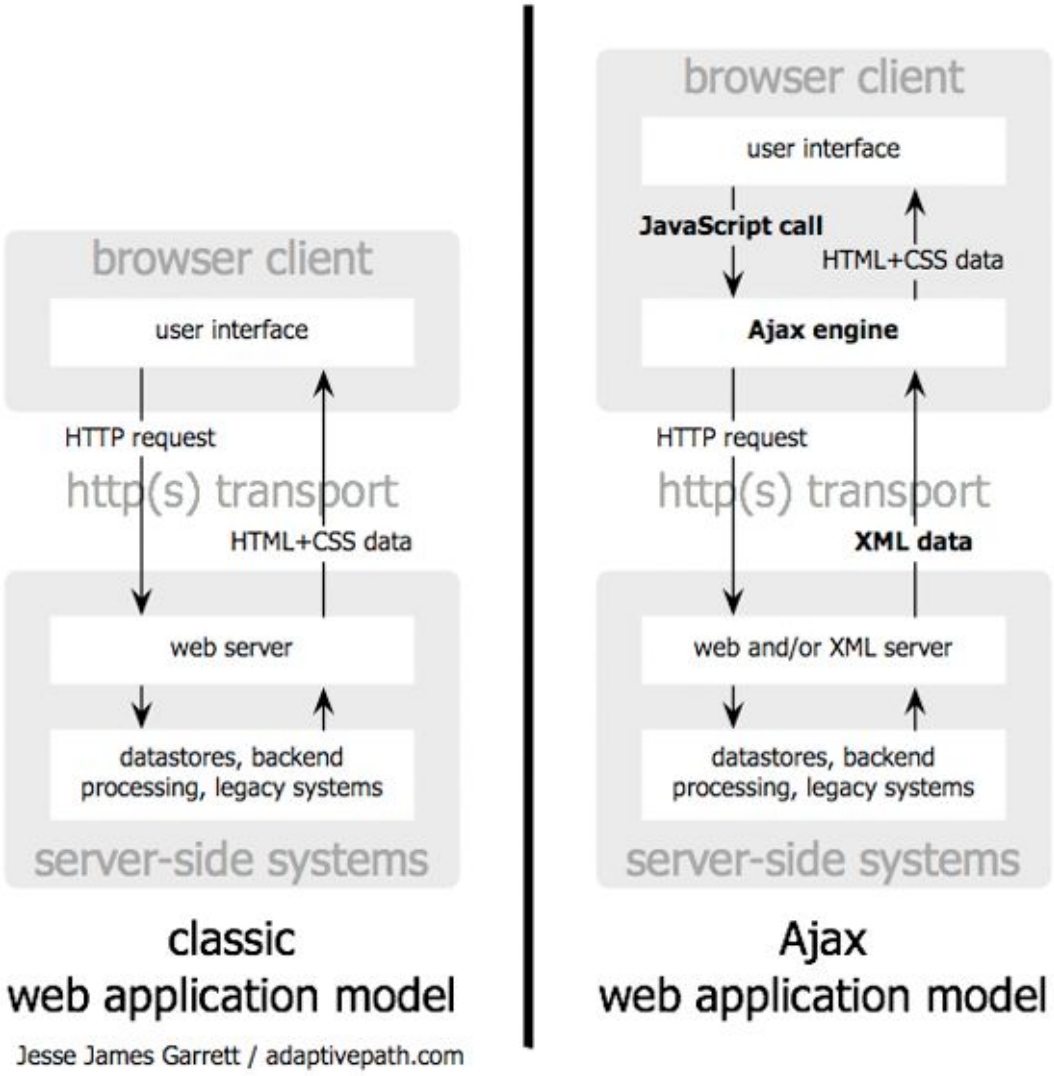
Programs constructed on environments and techniques from prior to the Web, that would handle the hyperlinks in a limited way, are a cause of Web Services that are diverging from the loose coupling of the Web. We provide an entirely new program environment, which is built around the handling of hyperlinks, and describe it accordingly so that we would make the programs themselves benefit from the loosely coupling. The Anatta Engine(s) is an environment for programs with loose-coupling abilities.

The server side of Web Services forms a vertical hierarchy because different APIs are called for each interaction, not only between Web Services (as described in Chapter 4), but also between any part in each Web Services, i.e. web-servers or data stores.

The Anatta Engine(s) changes this to a horizontal relationship on the Web by making each parts as Instance event-driven style.

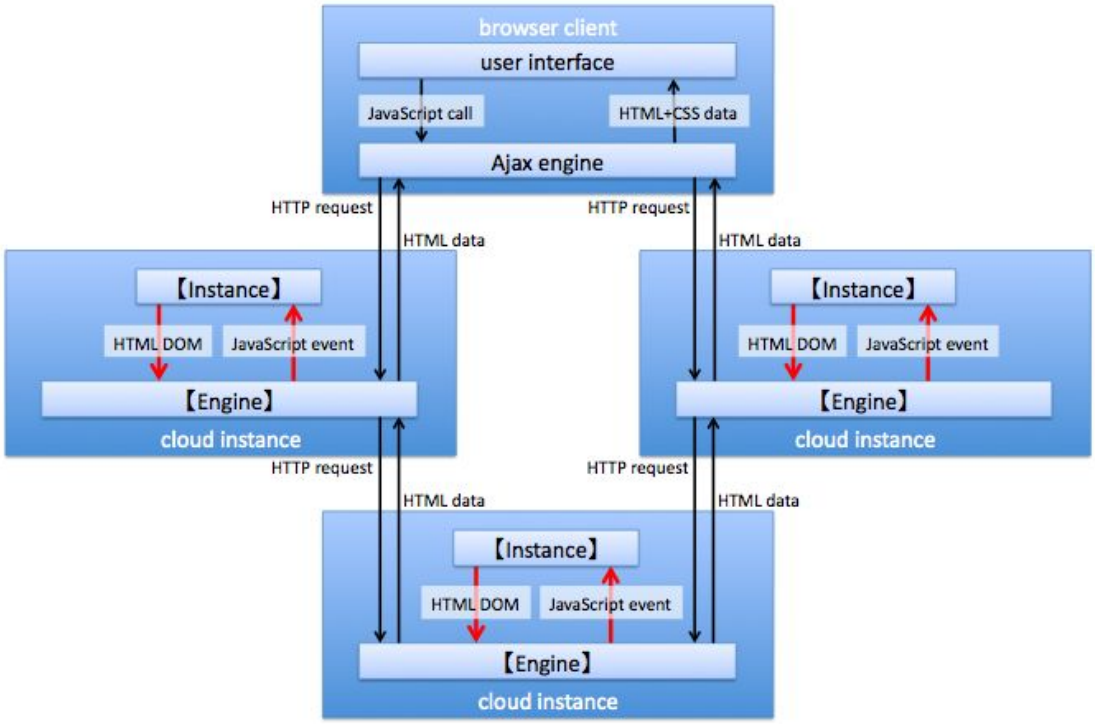
Just as AJAX improves the user experience by changing the architecture of the interface to one that is event driven, the Anatta Engine(s) resolves web fragmentation by changing the server-side structure to event-driven style.

Fig 5.1 Comparison of AJAX model (top) and Anatta Engine(s) model (bottom)



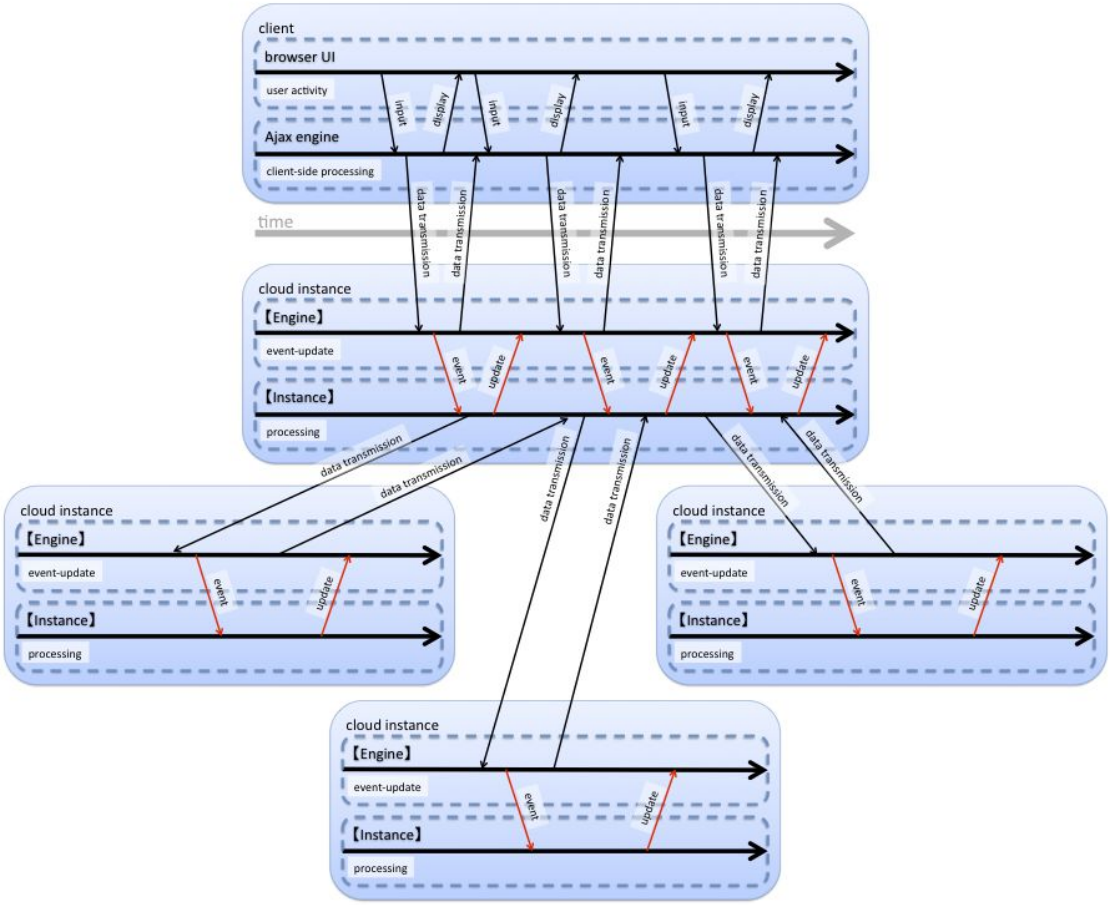
This image is from AJAX: A New Approach to Web Applications

http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications (quoted Feb. 4, 2014)



With the Anatta Engine(s), the `Engine` also introduces asynchronicity and loose coupling in the server-side mechanism. Instead of a synchronous invocation of the middleware service, it is processed as an asynchronous event publishing to an actively living `Instance`.

Figure 5-2 Comparison of asynchronous interactions in AJAX application (top) and asynchronous interactions in Anatta Engine(s) application (bottom)



The asynchronicity of each `Instance` can traverse between distributed `Instances` and function as an asynchronous interaction. The `Engine` also has a mechanism to realize distributed cooperation utilizing the loose coupling inherent in the Web, and by adopting distributed asynchronous cooperation on the Web, `Instances` can be attached and detached like parts. The Anatta Engine(s) adopts the following structure to realize these mechanisms.

Based on the architectural style of REST, the Anatta Engine(s) makes it possible to deal with link navigation asynchronously in a program by setting hyperlinks and Resources as first-class objects.

In the Anatta Engine(s), the `Engine` provides a mechanism to execute any program itself, that utilizes resources on the Web with the link navigations, as an `Instance` that exists as a resource with URI on the Web. The program as an `Instance` is also an existence that could be utilized from other `Instances` within the Web.

For a program `Instance` of the Anatta Engine(s), provides a mechanism that makes HTTP accesses to its web server asynchronous events. It enables it to process the asynchronous link navigations inside the processing from asynchronous events, and overall, in the Anatta Engine(s), all programs are described as being asynchronously processed.

The Anatta Engine(s) provides two basic elements; Links that represents URIs and Resources, and Entities that represents information of a state of a Resource.
The `Entity` functions as a container holding information of URI such as Resources, and hyperlinks. It also has a structure including HTML and Atom as formats that may represent the hyperlinks. The `Link` can be acquired as a group of hyperlinks from the `Entity`, and also the `Entity` can be acquired as a destination of the `Link`.

With a program that uses `Link/Entity` processing, everything is handled asynchronously. Programming handling asynchronous and unified Resource link navigation premised on HTTP and HTML can be available without dealing with the connection management and data parsing that were formerly used.

In an Anatta Engine(s), the most characteristic feature of a program's instance is that itself is also the Resource. As a Resource, accessed from outside with HTTP, the `Instance` executes the process described in the program, to any Resource as its objects.

During this execution of Resources as objects, the instance behaves as a User Agent that performs link navigations with functions of the `Link/Entity`. In the Anatta Engine(s), the `Instance` provides a mechanism that can combine both User Agent and Resource.
As both Resource and User Agent, each program `Instance` holds its current state individually. In the Anatta Engine(s), HTML DOM is used as a structure for managing a state of the `Instance`, and along with this, the HTML format including JavaScript is adopted for the source code of the `Instance` program. On HTML DOM, there are two advantages that

programs as Resource handle HTML as a Representation for describing hyperlinks, and that programs as User Agent could be programmed as a style managing their states with asynchronous events as same as browser-based programs fitted for processing the Web.

In the Anatta Engine(s), also, it is possible to use the `Instance` program to describe processing of arbitrary object Resources. In order to make it possible to process any Resource as an object, the program code describes the information of interest in the Resource to be processed. That is, the code is described as queries to the `Entity` with element names as metadata used on the program side.

From the programs, the Anatta Engine(s) separates processing locations on the structure of Representation formats of the Resource as their objects. This separation eliminates the tight coupling that is bound by the format between the program and the Resource to be processed.
By registering the TermSet in Engine that associates an element as metadata handled by the program with the location of the resource on the data structure, the `Entity` serves information from the element name.
Not only the creator of the program but also a third party such as a user of the program or a data provider can also freely define the `TermSet` and can publish and use the `TermSet`, itself as Resource.

Thus, with respect to when an `Instance` acquires an element from an arbitrary Resource, it can acquire and use a Resource having a Representation defined as a `TermSet` independently from an Anatta Engine(s) or the program. Using `TermSet` splits the processing and its structure for the processing, and when the processing is described as a program, there is no need to assume a specific formats or structures for the Resource, and it is possible to make a program that could apply any object.
Furthermore, by updating the TermSet only, the coverage of the processing Resource can be extended; for example, a Resource, that was not a processing object at the time of the program creation, could be put as the object at the later. The loose coupling based on the `TermSet` between the program and its target Resource not only extends the range of the target Resource but also widens the applicability of the program itself by making it more generalized, in principle giving it the ability to be used beyond the assumptions envisioned by the creator.

In this way, the Anatta Engine(s) utilizes `TermSet` for a Resource as a program, a program processing object, and a program to be processed as an arbitrary Resource, processes an arbitrary Resource, changes the state of the Resource and provides the program `Instance` and the processing.
With the Anatta Engine(s), the system can be realized as combinations of multiple Resources. Programs that perform web processing interact as Resources, executables of the programs are accessed from other arbitrary programs as Resources, and those programs are also accessed as Resources.

The Anatta Engines(s) generate and fill Resources in this way in a world where all targets are connected with the same specification, without being controlled. The Anatta Engine(s) creates a state that generates, coordinates, drives, and updates Resources, not by actions from the web browser or by data obtained by the program, or by fixed updates.

Under this Anatta Engine(s), the server is not for moving a middleware group consisting of calling relationships, but is changed to a structure, which can be called an `Instance` “group” linked on the Web by an event. The cloud will change to something for the `Instance` group, and the `Instance` group structure will accelerate technologies such as IoT (Internet of Things).

6. Anatta Engine(s): Links

Multiple Anatta Engine(s) work cooperatively to handle even a program Instance on the Engine as an object that is processed as Resource on the Engine. The Instance of the program extends the Web as a Resource that could be linked, while the program Instance also processes the links.

With the Anatta Engine(s), programs which treat the Instance of a program promoted to Resource on the Engine as a processing object can also run on the Engine. The former programs perform reacting with continuous state updates from the target program. The Anatta Engine(s) uses this chain of process-driven processing to enable programs to interact in a REST structure.

To enable interaction among such programs to form a single network on the Web, the Anatta Engine(s) also provides a mechanism consisting both of an implementation collection Streamer, which realizes interactive processing based on hyperlinks on the Engine, and a message exchanged in cooperation Particle, which is a metadata specification of a Resource that forms a universal event.

The collection Streamer that implements interaction processes on the Engine is comprised of three elements; the first one is called a Channel which is a program Instance on the Engine to focus on mediation processing of interactions, the second one is Port which is a library on the Engine for issuing messages to the mediation Instance, and the last one is another Port which is a library on the Engine for receiving messages from the mediation Instance.

Since the Channel is a sort of Instance on the Engine so that it is also a Resource which has a URI, the Port both of the issuing and receiving sides establishes interactions by specifying the URI of the Channel. The library on the receiving side is designed to handle the Resource linkage processing in asynchronous event processing performed by Instance of the Anatta Engine(s).

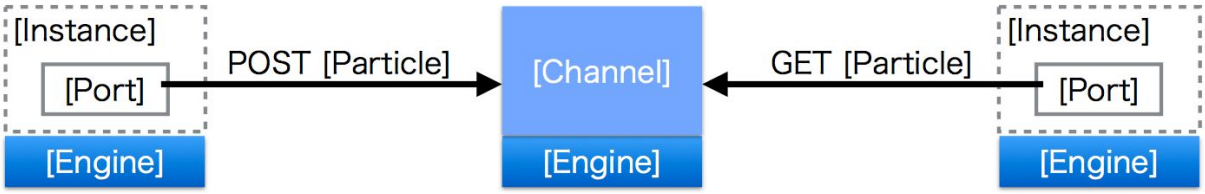
By means of the `Streamer`, individual events and event strings are also represented as Resources using hyperlinks. This makes it possible to express events and event lists on standard specifications such as HTML and Atom, and it can be handled as a normal HTML page or Atom feed regardless of the specification other than that of the programs' "GET" the URI of the `Channels` or the events.
Since the connection processing is implemented as a metadata process of the Anatta Engine(s), it is also possible to interpret normal HTML pages or Atom feeds as `Channels` on the receiving side.

The `Particle` would be prepared as a definition of universal general-purpose event information so that any connection library can be connected to the stream data flow through the `Channel` for processing event interactions. The `Particle` is a protocol to describe how the event occurred, such as the issuer of the event, the event object, the name of the event, and the date and time, using the metadata mechanism of the Anatta Engine(s).

The protocol does not contain any information related to "processing" of the application program. It specifies the information just related to the "propagation" of events. Both metadata prescribed in the protocol and metadata used in the processing of individual application programs are embedded as a Resource into one event.

Information in the `Particle` represents the interaction network itself that is constituted by the Anatta Engine(s), and is used by the programs that process the network itself such as the `Channel` itself or a program which propagate events from one `Channel` to another.

Figure 6-1 The relationship of `Instance`, `Particle`, `Channel` and `Port` comprising a Streamer



In this way, the Anatta Engine(s) realizes a "general message bus" on the Web that allows programs to link in an open and chained manner using `Streamers` and `Particles.`

> Note: General message bus:
> In the Web Services industry there is a system that employs a structure using event-driven via a message bus such as an Enterprise Service Bus (ESB). However, that's a middleware product used as a backend bus; it is not designed to perform open interaction based on the Web.

Interaction method performed on an Anatta Engine(s) network is designed to realize that it can produce open connectivity throughout the Web, like the hyperlink relationships. As a Resource, the "general message bus" of Anatta Engine(s) is a mechanism on the current Web using URI, HTML, and Atom. The mechanism of event linkage works by applying the function of the Anatta Engine(s) that realizes loose coupling, such as metadata processing, to Resources of Web standard ones.
As such, it does not vertically connect, like the backend layers of a middleware product (an OS and application, for example), instead the Anatta Engine(s) horizontally links the programs as Resource.

With the hyperlink and the general message bus implemented by the Anatta Engine(s), the Web becomes a single network of programs that are interacting, while each continues with its independent processing. The world of the Web, where applications-bound data and users form

individual networks is changing to one in which data and users are autonomously and freely linked as Resources expanding day by day and giving rise to new data and applications.

7. Anatta Engine(s) : Network

The Anatta Engine(s) processes arbitrary programs with the User Agent mechanism. The first step is to run on the Web as an Anatta Engine(s) program what the user wants to process automatically on the Web while using the browser. Here, the Anatta Engine(s) assumes a situation in which the user arbitrarily prepares and executes a program from each viewpoint, in the way that a user performs an add-on of a browser or the like. The user customizes it as a program for processing the Web for themselves and runs the program on the Engine.

Those small programs running on the Web in a distributed manner will interact in an event using the following Resources. That is, by means of a generic Channel, that has a URI and uses events to connect Engines together, and a Resource having a specification that enables open event connection, programs on Engines dispersed on the Web are capable of making open event interactions, then they have the ability to connect even if they do not know each other. Programs are linked via the Channel, and a Channel can be linked with another Channel via a program.

Through the program "links" in other words linking these processes with each other by events,, the Anatta Engine(s) programs form a dispersed system on the Web. Events on the system can be extended or another distributed system constructed by connecting other Anatta Engine(s) programs. In this way, links through the interaction of Anatta Engine(s) form one big network, among which dispersed programs on the Web become components that span multiple dispersed systems, a network in which dispersed systems coexist.

The program on an Engine changes its own state from an event by its processing, and the state change is also accessed as a Resource, so it has dynamics that cause an event acting on another Engine. Also, as indicated by the URI from the Engine, the Channel forms a medium to propagate the dynamics and, to expand on the explanation of Chapter 6, becomes a Resource that indicates the links of an indefinite number of Engines. The Channel forms a Resource representing a "flow" of events caused by a large number of dispersed Engines. The interaction between Engines and Channel brought about by the dispersive properties constitutes a network in which the spread of "dynamics" is manifested as the "flow."

The existence of a network with the "flow" changes the way itself of handling the network. Since a network, in which there is already a flow can be openly available, in anticipation of the effect of the flow, services link programs for executing application processing of data coming from the flow, and conversely, in anticipation of the processing at the flow destination, the user links a program that puts data on the event in order to flow the data. The increase in individual links to the network will cause it to expand, leading to further flows and increased links. Conventionally, a network has been attached to a specific purpose application service like a database, and individual applications have provided separate networks, but in this "flow" network, application processing providers and users are linked to one network, and, in effect, are handling the same flow.

Here, it is rational for both provider and user to take action premised on the flow network. Rather than prepare a system from scratch by oneself, utilize the existing network, prepare the necessary parts and link to that network to achieve the purpose. If what is on the network is unsatisfactory or defective, you will prepare and link to something better that is updated or changed. The existence of such variations will also encourage original updates. In this way, the changes of individual dynamics accumulate and become the dynamics of the network itself.

A program on the Web should accept and process hyperlink destinations that include a variety of events brought in via the flow not limited to a specific event source, thereby becoming an open program, and its value varies in accordance with changes from the flow.

The Anatta Engine(s) has a function to process a Resource by using "the information structure metadata." In the program, Resources such as events and their hyperlink destinations are processed using metadata based on the `TermSet` of Chapter 5, not data structures or contents in a specific format, thereby making it possible to make use of the diversity brought about by the flow.

Conversely, the program can also implement control to adapt the flow itself to its purpose using information coming from the metadata of various events. Acquiring the Channel from the event metadata, the program dynamically switching destinations of autonomous links or metadata

restrictions. With respect to the same flow, when multiple programs handle different events, they become diverse network activities, further increasing the dynamics of the network.

In this way, the Anatta Engine(s) attempts to apply the architecture that systems comprised of programs link each other with hyperlinks like web pages by having dispersed systems mix on the network.

The dispersive property of the Web is also useful in system data and programs. For example, while in Web Services, personal data is managed in one place as account data and exists independently for each service, it is more natural to gather the data together on an individual basis rather than on a service by service basis and interact in a distributed manner. In ubiquitous computing, for example, instead of installing a specialized device for a specific application, devices such as sensors and actuators are personally equipped, then linked the devices together or linked between the devices and programs on Anatta Engine(s), it applies the way to expand their applications.

With Anatta Engine(s), the programs that make up a system, the persons that make up a social network, and the devices that make up ubiquitous computing could all be dispersed and placed on the same network and mixed through event linkage on the Web. This is the "new Web."

8. Anatta Engine(s) : `Prototype`

We have prepared a prototype implementation of the Anatta Engine(s) running on node.js[ii]. The `Engine` can be executed in an environment, in which node.js runs.

In this implementation, we use promise[iii] for the `Engine` itself and for user program the `Engine` runs, as asynchronous programming, for which we adopt nodejs and q[iv], which is a promise implementation in Browser JavaScript. We also used jsdom[v], which is highly compatible with HTML DOM on the browser, as an implementation of HTML DOM, which is the state of the `Instance` on the `Engine`.

Here is an example of the program shown in Chapter 5 in which the `Instance` of the `Engine` processes the Web.

```
"use strict";
window.addEventListener("agent-load", function (ev) {
    var url = document.getElementById("url").href;
    var link = anatta.engine.link({href: url});
    var contents = document.createElement("div");
    var template = document.querySelector(".newsitem");

    var content = function (entry) {
        var item = template.cloneNode(true);
        item.querySelector(".href").href = entry.attr("href");
        item.querySelector(".title").textContent = entry.attr("title");
        item.querySelector(".date").textContent = entry.attr("date");
        item.querySelector(".desc").textContent = entry.attr("desc");
        return item;
    };

    window.addEventListener("agent-access", function (ev) {
```

```
        ev.detail.accept();
        link.get().then(function (entity) {
            entity.all().map(content).forEach(contents.appendChild,
contents);
            ev.detail.respond(200, {
                "content-type": "text/html;charset=utf-8"
            }, contents.innerHTML);
        });
    }, false);
}, false);
```

This program code shows the asynchronous process that extracts the `href`, `title`, `desc`, and `date` of the enumeration entries, with the metadata functionality from the web page acquired by `GET`-ting the link destination stored in the HTML DOM as the `Instance` state, and returns self-states for each HTTP `GET` arrived.

As shown in Chapter 5, the metadata handled by the program is independent of the format and the document structure. This code would work when mappings that associates externally provided metadata with the document structures is registered to the `Engine`.

For example, the mapping of metadata to correspond to the structure of the Atom feed put out by the site "W3C News" is as follows.

```
{
    "name": "w3cnews-feed",
    "content-type": "application/atom+xml",
    "link": {
        "href": {"selector": "link[rel='alternate']", "value": "href"},
        "title": {"selector": "entry > title", "value": "textContent"},
        "date": {"selector": "updated", "value": "textContent"},
        "desc": {"selector": "entry > content > p", "value": "textContent"}
```

```
    }
}
```

As shown in Chapter 5, the state of the `Instance` is HTML DOM, and in order to create an `Instance`, it is necessary to prepare HTML representing its initial state. The HTML for processing "W3C News" using the above program (referred to as script.js) is as follows.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>agent</title>
    <script src="script.js"></script>
  </head>
  <body>
    <a id="url" href="http://www.w3.org/News/atom.xml">W3C News</a>
    <article class="newsitem">
      <h1 class="title"></h1>
      <a class="href"><div class="date"></div></a>
      <p class="desc"></p>
    </article>
  </body>
</html>
```

This is common HTML for which library codes for the browser code can be used. Alternatively, library codes written for the `Instance` can be executed on the browser.

The libraries bundled for an `Instance` in the prototype can be made available to use in the JavaScript code for the browser as it is. These bundled libraries include a simple template

system based on DOM, a testing framework, and a `Streamer` for “dispersed event” interactions.

We plan to integrate the `Streamer` functionality into the `Engine`, which will make it possible to handle from the program code in the same way as the JavaScript code, described above.

Furthermore, as an example of the system constructed by dispersed interactions,, we have implemented an online bookmark sharing system (codename "potluck"). In this system, the following six Anatta Engine(s) programs run as independent `Instance`s and adopt the event-driven architecture via `Streamer`s.

"activities": A program `Instance` that makes bookmark activity events as Resource. They are responsible for the functions corresponding to `Channel` described in Chapter 6.

"post": A program `Instance` that converts form data POSTed from the browser UI into HTML as a bookmark activity event and issues it to "activities.”

"index": A program `Instance` that receives events from "activities," then builds a Resource as a view of the new page arrivals list.

"link": A program `Instance` that receives events from "activities" then builds a Resource as a view of comments listed for each page.

"author": A program `Instance` that receives events from "activities," then builds a Resource as a view of pages listed for each actor.

"tag": A program `Instance` that receives events from "activities," then builds a Resource as a view of comments listed for each tag.

This system applies a simple architecture consisting of an `Instance` which only issues or only receives events. Individual Resources are linked to each other by hyperlink. It does not carry out any database-like relations or network connections.

In this architecture, "index" and "link" keep updating their respective specialized data from the received event, and manage it locally. Since each `Instance` is independent, when the system is extended, such as by implementing a "popular comment list," for example, it can be done by creating a program that constructs data representing that function from the received event and adding a new `Instance` of this program. That can be done without affecting other active `Instance`s. After the addition, the `Instance` the update of the system completes by each Instance adding a hyperlink to it at any time.

On the event-issuing side, the system can be enriched in other ways. In addition to the ablity to "post," a bot-type program `Instance` can be operated which watches over twitter and converts tweets into events that are continuously issued to "activities." In this case, in each `Instance` an event that includes the link is interpreted as a bookmark activity event and processed in the same way as the event issued by "post." In this way, "activities" will change beyond the `Channel` for bookmarking only, resulting in an evolution to a mixed network of distributed systems as described in Chapter 7.

The architecture shown in this example of a system is not special and not one having a narrow range of application. The use of the Command Query Responsibility Segregation (CQRS) architecture also makes it possible to realize a general application system with the same style on the Anatta Engine(s). If the CQRS architecture were adopted by usual systems, it would be divided into possible subsystems for each structure which can be centrally managed inside the databases. But with the Anatta Engine(s), it becomes possible to disperse for each Resource, so that, as shown in this example, it is possible to adopt a CQRS structure with finer elements.

For this reason, even when performing a function extension that connects a single instance, or when constructing a larger system by linking multiple systems, it is possible for it to evolve as a CQRS based system using hyperlinks and event linkage.

This implementation of prototype “nodejs” allows one of the environments which runs the user program of the Anatta Engine(s) that performs Web processing. The implementation can be executed directly using PaaS or IaaS or a PC or small ARM[vi] machine.

The prerequisites of the implementation of Anatta Engine(s) are HTML DOM and an ECMAScript 5[vii] execution environment. Therefore, the Anatta Engine(s) can be implemented in a WebKit[viii]-based JavaScript execution environment including modern Web browser addons, a WebKit library prepared in various environments and phantomjs[ix]. The Anatta Engine environment can also be implemented by forking open-source Chromium[x].

---

i Anatta: "Anatta" is a Buddhist concept and means "non-self." The ancient Indian philosopher Nagarjuna equated "emptiness" with "anatta" that refers to the doctrine of "non-self" in “Dependent Origination”, saying: "All things and phenomena that exist in this world depend on each other and their essence is not forever immutable; if one element or condition changes, another will change and the whole world will already be different from the world of yesterday." Leading on from that to the "One Web, " we propose "Anatta" as a technology that enables a world different from yesterday’s.

ii node.js http://nodejs.org/

iii promise http://wiki.commonjs.org/wiki/Promises

iv q https://github.com/kriskowal/q

v jsdom https://github.com/tmpvar/jsdom

vi ARM http://en.wikipedia.org/wiki/ARM_architecture

vii ECMAScript http://www.ecma-international.org/publications/standards/Ecma-262.htm

viii WebKit http://www.webkit.org/

ix PhantomJS http://phantomjs.org/

x chromium http://www.chromium.org/

ANATTA Overview

December 2015

KANATA Limited