From APIs to Affordances: A New Paradigm for Web Services

Mike Amundsen
amundsen.com, inc.
mca@mamund.com

ABSTRACT

The ecosystem of services on the Web continues to grow and evolve while, at the same time, the number and diversity of connected devices increases; challenges lie ahead for both providers and consumers of Web services. This paper is presented as a 'what-if' proposal; an alternate paradigm for dealing with an increasingly heterogeneous network.

Drawing from diverse sources including physical architecture, industrial design, the psychology of perception, and cross-cultural mono-myth, a new implementation paradigm is proposed to help software architects and developers meet these challenges; one that invites participants to shift their mental model from that of programming network devices to programming the network to which those devices are connected.

To accomplish this goal an "affordance-rich message" is proposed; one that is based on shared understanding through network-oriented affordances instead of device-oriented APIs. A working model based on this approach is offered, examples given, and areas of related work identified.

Keywords

HTTP, WWW, hypermedia, networks, SOA, REST, distributed computing, web services, usability, evolvability

1. BACKGROUND

In the last several years, the landscape of the Internet has changed noticeably. There are many more connected devices, more connected applications, and thousands of Web 'APIs' to service them. This represents a new 'ecosystem' for the Web; one dominated by small devices loaded with specialized applications, all talking across the Web using shared application programming interfaces (APIs). While the shift did not happen all at once, probably the date that best marks the start of this new era in the Web would be January 10, 2007; the day the first Apple iPhone was introduced[13].

WS-REST 2012, April 2012; Lyon, France

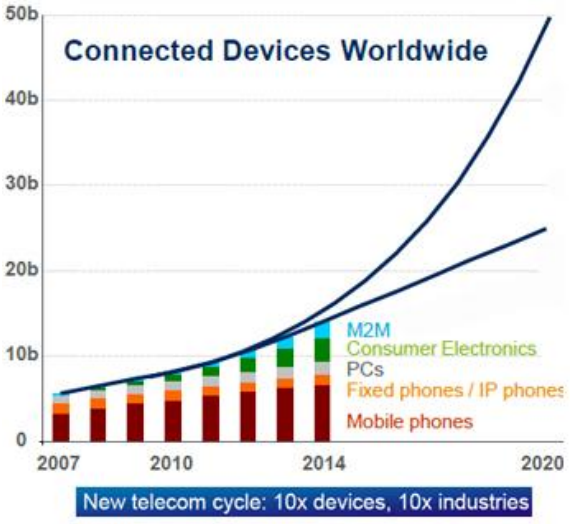

Figure 1: From Ericsson : 50b devices by 2020

The resulting sales boom launched competitors and an industry has grown up around the devices themselves. As an example, even the work force needed to support the creation of applications for hand-held devices is considered worthy of scrutiny.

1.1 More Devices

The common wisdom is that the number of devices connected to the Internet is growing rapidly (See Figure 1). In May of 2009, Intel predicted that the number of 'connected devices' would grow from the then estimated 5 billion to 15 billion by the year 2015 [7]. In an April 2010 press release [18], the CEO of the Swedish telecommunication company, Ericsson, predicted the number of connected devices will balloon to as many as 50 billion by the year 2020.

1.2 The 'App Economy'

Much has also been made of the 'App Economy' - a burgeoning market that exists to feed the many devices we all own and use (See Figure 2). This economy is now viewed by some as a 'job creator' worth of special attention. Researcher Dr. Michael Mandel estimates 466,000 new jobs have been created since the introduction of the iPhone in 2007.[19]

These apps are increasingly written in code that is 'native' to the device; thus requiring custom builds for each

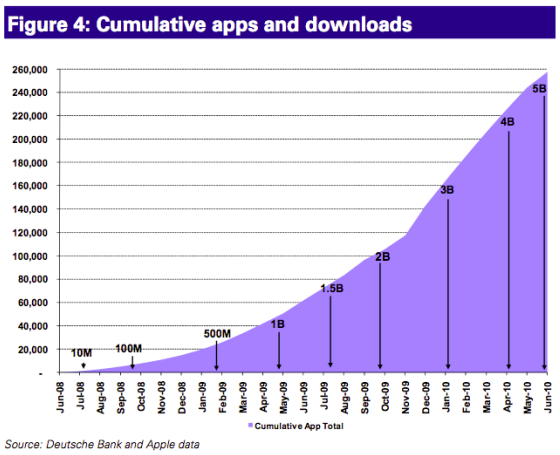

Figure 2: From Smart Insights, October 2010

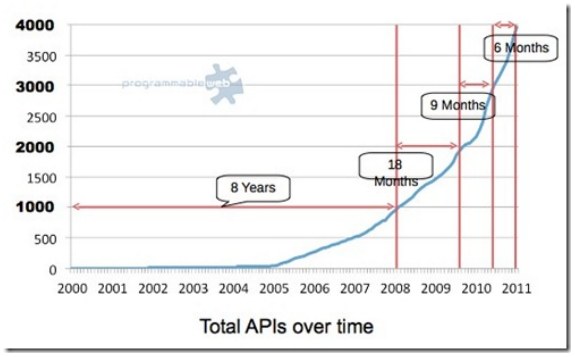

Figure 3: From the Programmable Web

targeted device or family of devices. Not surprisingly, new products have appeared to decrease the effort needed to produce the numerous native code builds (i.e. PhoneGap[23], Appcelerator[3], and others). Of course, there continues to be debate within the mobile community on whether a 'native' or 'browser-based' approach is preferable.

1.3 More Services

To match the growth in devices and applications, a similar growth is occurring in services (or 'APIs') to support the ecosystem (See Figure 3). The Programmable Web, a site which catalogs such things recorded more than 4000 Web APIs as of October 2011 and proudly stated "The whole web as a platform has come a long way and done so very quickly."[24]

1.4 Summary

Each of these areas of growth represent great opportunities. Growth means new markets, expanding use of the Web, and the chance to improve the quality and quantity of data available to all users.

But there are challenges, too. These rapidly expanding markets are, for the present, continuing to rely on dated technologies and mental models. Despite the incredible growth of the Web both in real (device) and virtual (applications & APIs) terms, the most common approaches to service this market are still based on software theories developed at the dawn of personal computing; when devices where most often "stand-alone" machines, sometimes connected to each other over local area networks, and only on occasion (and for brief periods) connected over slow communication lines to other, remote devices.

2. THE PROBLEM

There are a number of problems posed by the current trend in adding many small, dedicated devices to the Internet. This paper focuses on a set of problems at the level of software architecture; problems that related directly to the work of enabling communication between connected devices.

2.1 Technical Difficulties

One type of shortcoming in the common implementation model for Web APIs are technical in nature. Primarily these are due to incomplete or inappropriate implementation of the HTTP protocol. While there are many possible minor difficulties to implementing HTTP (misuse of methods, status codes, headers, etc.) two general problems are identified here:

- Treating HTTP as a Transport
- Lack of Component-Connector Modeling

2.1.1 Treating HTTP as a Transport

Most of the Web API implementations continue to use traditional RPC-style interfaces. The messages sent over the wire are simple data blocks, often meant to faithfully represent a server's internal objects or data graphs. Usually the information in carried via a simple data format such as XML, JSON, CSV, etc. This approach has the effect of treating HTTP as a transport protocol.[20] and weakens one of the three pillars of Fielding's architectural style: the Data element[8]

When this happens, the ability to communicate options to the recipient is lost. Client applications are expected to have a complete understanding of not only the message received, but also any other possible messages that could be received or even requested. This dependence on clients to make all the decisions means that there is almost no "shared understanding" between clients and servers other than an understanding of the data format used to pass messages back and forth and the protocol used to send them.

Using HTTP as a transport can also result in implementation models that limit client server interactions to only those that easily map to the protocol's method set. This is commonly referred to as HTTP-CRUD[28]. Combining this limitation of the client-server interaction with the dependence on simple data formats for messages not only misses key properties of HTTP, it also limits the usability of the Web for anything other than purpose-built client applications that rely on a static, isolated understanding of the problem domain represented by a single server.

2.1.2 Loss of Connector-Component Model

The Connector-Component Model was first described by Taylor, et al in 1995[27] as Chiron-2 or C2. The original model was used to coordinate independent user interface components using a connector to pass messages between them. Later, Fielding used similar terminology to describe network-level communications where independent

components (running on separate machines) use protocol connectors to communicate across the network[8]. For Fielding, Component, Connector, and (as seen above) Data were the three key architectural elements of consideration for his REST style.

This architectural model (one that relies on independent components using connectors as intermediaries) has allowed the Web to continue to scale not just at the runtime level, but also at the implementation level. Since components only need to know how to talk to connectors and connectors only need to know how to talk to other connectors, it is possible to independently build parts of the network with minimal design-time co-ordination between parties.

The recent trend in building 'native' applications for connected devices can result in a loss of the component-connector paradigm and, in turn, cause problems at scale; at both the implementation and runtime levels. Employing cross-platform build tools only masks the problem.

2.2 Competing Priorities

On a more practical level, creating Web services that are both flexible and easy to use is a challenge. An easy-to-use Web service is essential for attracting customers. A flexible, evolvable service that does not tax developers by obsoleting their work too often, is important for retaining customers over time. In some cases these are competing priorities. For example, flexible evolvable systems can be viewed by developers as difficult to understand.

2.2.1 Immediate Usability

As more Web services appear, they compete for the attention of developers and subscribers. Along with reliability and performance, Web service providers are growing increasing aware that usability is a key factor for increasing adoption.

If developers cannot understand the Web service, can't easily connect and quickly build working solutions for it, they are likely to look elsewhere. In addition, once a developer settles on a provider, that developer is not likely to switch to a new provider since most web service integration today requires special coding for each and every service; even for the same service provided by two different vendors (i.e. peer-to-peer messaging, photo-sharing, etc.).

Thus usability is a key factor in acquiring new customers. Web services need to offer familiar, easy-to-understand examples and documentation in formats developers understand and can quickly convert into working code.

2.2.2 Long-term Evolvability

At the same time, one of the desirable properties of large-scale systems is the ability to support evolution of the service over time; evolvability. Fielding defines this as "Evolvability represents the degree to which a component implementation can be changed without negatively impacting other components."[8]

A common way to support long-term evolvability is to use hypermedia affordances within response representation. Fielding's REST style, which relies on the use of hypermedia was conceived with long-term evolvability in mind. In a 2010 interview Fielding states "Most of REST's constraints are focused on preserving independent evolvability over time, which is only measurable on the scale of years."[9]

2.3 The Time Dimension

Another important aspect of supporting large networks is the element of time; not at the micro, but the macro level. Few architectural models consider the passage of time and how it affects both the participants and the messages they share.

2.3.1 REST Resources Over Time

In Fielding's REST style, a resource (that which can be named) is "a conceptual mapping to a set of entities, not the entity that corresponds to the mapping at any particular point in time." In this way, not only the *representation* of the resource can change over time, the data upon which the resource is based may change as well; including a case that results in an 'empty' resource. Under these conditions, enforcing contracts on *what an identifier will return* at some point in time in the future is likely impossible. Instead, in REST, "The only thing that is required to be static for a resource is the semantics of the mapping, since the semantics is what distinguishes one resource from another." In other words, the conceptual meaning identified when the resource was created is expected to remain constant. If, at the time of creation, the identified resource is meant to represent 'the latest edition of Document A', it can be expected to continue to represent 'the latest edition of Document A' at any future point. However, the *contents* of that document cannot be expected to remain static over time.

2.3.2 Static Contracts

Unlike Fielding's focus on the semantics of the identified resource, the most common attempts to describe how clients and servers interact over the network (i.e. WSDL[6], WADL[16], etc.) rely on static interface contracts that do not change over time. Whereas message-based media types whose processing models describe possible affordances (controls) which could appear within responses, typical Web API contracts contain a set of function calls (including arguments and return types) which the client can use to "compose" their own set of interactions with the server (usually after some level of coordination via additional documentation). Web APIs have no clear means for communicating the semantics of the identified resources.

Static contracts can result in 'frozen' implementations that are unable to easily evolve over time as the problem domain changes. Even more important, focusing on 'native' implementation strategies can result in mixing the connector semantics (i.e. HTTP protocol) with the component semantics (i.e. the problem domain) in ways that make it more difficult to modify the application over time. It is possible that application vendors have little incentive to create long-lived applications since their revenue maybe derived from the constant update/replacement of 'obsolete' apps. However, many app providers may not be focused on gaining revenue through the replacement of what could be perfectly acceptable applications if implemented differently.

2.3.3 Transient Devices, Persistent Networks

Current trends indicate that the hand-held computing devices are not only seen as essential in today's world, they are also treated as disposable. A 2010 report estimated Americans dispose of 130 million cell phones each year[26].

While devices can be viewed as transient, the networks themselves continue to run 24x7 and, now more then ever,

are available via wireless connection to the point where it is possible to remain connected throughout the day, even while traveling. With continued use of CDNs (Content Delivery Networks) and the rise in SaaS (Software as a Service), not only can users stay connected, there is increasing likelihood they can access their personal content at all times.

In this light, continuing to focus efforts on programming each device natively, encoding all the domain knowledge on these devices, may not be the best way to make use of software developers' (and architects') energies. Instead it may make more sense to leverage the network itself; to actually program the network instead of the connected devices. This idea represents not just an adjustment in focus, but also a change in the way network applications are architected and implemented.

3. OTHER DISCIPLINES

Before moving on to a proposed alternate paradigm for modeling communication along the network it may be informative to highlight similarities in observations about perception and communication from other disciplines. Here five perspectives on the way humans perceive and communication information are offered for the reader's consideration. The purpose here is to identify a similar thread throughout multiple disciplines; a thread that may be applied to modeling communication on widely distributed networks.

3.1 Architecture

In 1979 Christopher Alexander released the first in a series of texts describing his approach to physical architecture: "The Timeless Way of Building"[1]. In it, he asserts that "[P]eople can shape buildings ... using *pattern languages* and that "A pattern language gives each person who uses it the power to create an infinite variety of new and unique buildings, *just as ordinary language gives him the power to create an infinite variety of sentences.*"

In 1987, Beck and Cunningham presented a workshop at OOPSLA-87: "Using Pattern Languages for Object-Oriented Programs"[4]. The abstract contained the following report: "Our initial success using a pattern language for user interface design has left us quite enthusiastic about the possibilities for computer users designing and programming their own applications."

For Alexander, patterns (and languages built up from them) are a transcendent means of communication. Alexander's pattern language relies on the notion that all individuals can recognize abstract patterns, regardless of variance of time (over the centuries) or place (Rome, Africa, China, the Americas, etc.). Beck and Cunningham, and many who followed them, were able to apply this same notion to software designed to support direct human interaction (i.e. graphical interfaces).

3.2 Visual Perception

Around the time that Beck and Cunningham were applying Alexander's pattern model to implementing graphical user interfaces, psychologist James Gibson explored the concept of "affordance" in his book "The Ecological Approach to Visual Perception."[11] For Gibson, affordances were the "action possibilities" of the environment in which the subject resides. These possibilities were perceived by the subject in relation their abilities. For example, an short opening two feet wide might be perceived as a 'doorway' to a small creature, but not to an animal six feet tall.

Gibson claimed that animals continually 'sampled' their surroundings and made decisions based on the affordances available to them at the moment. For Gibson, the world was divisible into ecological niches; each with their own set of affordances and animals within that niche became expert at exploiting the available affordances. Affordances (and therefore options) were everywhere and, like Alexander, Gibson believed these affordances could be described in general terms (doorway, chair, step, etc.) that applied across environments.

3.3 Industrial Design

The notion of affordances was further refined by Donald Norman in his 1988 book "The Design of Everyday Things."[22] Norman applied Gibson's ideas to industrial design and HCI (Human-Computer Interaction) to help launch the field of Usability. Along the way Norman identified the Seven Stages of Action to describe how humans usually interact with their environment in order to accomplish a goal:

1. Set a goal
2. Form an intention to reach that goal
3. Specify and action
4. Execute that action
5. Perceive the state of the world
6. Interpret the state of the world
7. Evaluate the outcome against the goal.

Norman also described the notion that humans approach the environment (and its affordances) with some level of information already "in the head" and use their perception to discover information "in the world". It is this mix of "in the head" and "in the world" that determines the usability of an object (for that individual).

By detailing a series of steps humans use to reach their goals and acknowledging that information resides both within and without the individual, Norman's vision of the world includes not just Alexander's patterns and Gibson's affordances, but also the knowledge and goals of the participant.

3.4 Cross-Cultural Mono-Myth

American mythologist and writer Joseph Campbell described a different kind of shared pattern in his 1949 work "The Hero with a Thousand Faces."[5] For Campbell the 'hero's journey' was a story which not only appeared in multiple cultures across both space and time, but it retained the same general pattern which he summarized as follows:

> "A hero ventures forth from the world of common day into a region of supernatural wonder: fabulous forces are there encountered and a decisive victory is won: the hero comes back from this mysterious adventure with the power to bestow boons on his fellow man."

Campbell's work explored the idea that this shared story was evidence of communication based on archetype and metaphor; something that transcends any single language or culture.

For Campbell, the Mono-Myth was a way to share understanding across the divides of clan, kingdom, and time. However, unlike Alexander who focused on self-standing patterns in the world, Campbell asserts that entire portions of culture and story can be viewed as a single 'shared understanding.'

3.5 The Map is not the Territory

Another view of human communication was put forward by Alfred Korzybski in his 1933 tome "Science and Sanity."[15] Here, Korzybski outlines his view that human knowledge is limited not only by our ability to perceive the world but also the language we use to describe what we perceive. For him, our perception is always incomplete; always missing details and filtered by our current beliefs. It was Korzybski who coined the phrase: "The map is not the territory."

Korzybski acknowledged that this ability to function using only a general description of the world allowed humans to develop language, create names for things and share understanding (however imperfect). Shared understanding of the general nature of the world is how humans successfully interact with the environment and it is through language that they share knowledge over space and time. Thus, like Campbell, Korzybski saw understanding as rooted in the language we used to describe our surroundings.

3.6 Summary

What all these examples have in common is the notion that there are identifiable entities (patterns, affordances, general concepts, etc.) to which we all can relate; across culture, time and distance. For Alexander, a 'doorway' is a universal concept that can be referenced by all. For Gibson, this 'doorway' is recognizable even when it has only the barest visual resemblance to our common idea of a doorway (i.e. an entrance to a cave). For Norman, doorways can be rendered more (or less) usable through the application of design principles that can be applied across cultures. For Campbell, the possible meanings of a doorway (as a threshold to a magical place, as a metaphor for moving to a new stage of life, etc.) are also shared.

And finally, Korzybski tells us that all communication is essentially approximate. That the concept of a 'doorway' is useful (possibly more so) when it's left vague and general. This generality of shared semantic understanding is the basis for our ability to communicate. We'd be eternally frustrated and lost if we could only successfully communicate when all of us agreed on the exact meaning of every detailed utterance.

So, if the notion of shared understanding through general concepts - ones that are only loosely defined - is a useful paradigm in the fields of architecture, product design, psychology, and story-telling, could it not also be useful in the design and implementation of systems built specifically for sharing information? Do these other disciplines lead us to an alternate way of thinking about information networks themselves?

Instead of working to narrow the scope and meaning - to remove the ambiguity - of network communications; instead of working to create static interfaces that are tied to a specific place and time, maybe there is a way to mimic pattern languages on the network itself; to improve shared understanding by dealing in general concepts communicated through the use of mutually understood affordances in ways that allow for local interpretation and embellishment without the loss of basic meaning.

What would such a system look like? How would it be organized? What are the details of how this kind of communication can be shared over the network?

4. AN AFFORDANCE PARADIGM

If we accept the notion that our view of the world is, by nature always imperfect and that it is shaped not only by our observation but also the language we use to describe it then, it may be possible to use a new description language to help use alter our view of the way devices can communication over the Web.

To this end, this paper identifies two new maxims for the implementation of distributed network applications:

1. Program the network, not the device
2. Rely on affordance-rich messages for communication

The current paradigm for programming network devices is to use the available network as a mere transport over which to ship serialized objects and data graphs based on static programming interfaces. This model is based in the early event-driven, object-oriented paradigm typified by Smalltalk-80.[12] This approach fit well with the (then new) pattern theories of Alexander and Beck Cunningham. It made sense for handling the interaction of small components arranged within a local graphical interface.

But a model for enabling communication between UI components all running in the same computing space is not the best approach for enabling communication between components over a widely distributed heterogeneous network. Fielding outlined this point of view in his 2001 dissertation[8] using an example architectural style he labeled REST. What Fielding did not fully explore, however, were the details of what data messages in his REST style looked like and how components and connectors worked together to enable communication via these messages. One line from the dissertation has been singled out as the only description of what this message style might entail:

> "REST is defined by four interface constraints: identification of resources; manipulation of resources through representations; self-descriptive messages; and, hypermedia as the engine of application state."

4.1 Affordance-Rich Messages (ARMs)

Fielding mentioned hypermedia as the way to modify state on a distributed network. In this he meant "the simultaneous presentation of information and controls such that the information becomes the affordance through which the user obtains choices and selects actions."[10]

These affordances are, essentially, the pattern language described by Alexander. They are the environmental elements of Gibsons world. Fielding's description also tracks very closely to that of Norman. Applications that rely not just on rules and operations within the code ('in the head' as Norman would say) are capable of recognizing and reacting to affordances in the message itself (similar to Norman's 'in the world'). Affordance-rich messages allow for increased shared understanding. They represent more than raw data passed between parties. Additional information about the semantics associated with the data and the options available at that moment in time are also available to the recipient.

And, as outlined by Campbell and Korzybski, sharing the exact meaning of each and every affordance and semantic detail is not necessary in order to share general understanding. Knowledge that an in-message control "affords sending data" or "affords filtering results" is sufficient to support a wide range of operations. Rather than designing systems that fail unless all the exact details in are in place, a better approach is to design what Norman refers to as 'explorable' systems; ones that allow users to safely make attempts at reaching their goals.

> "Think of each action by the user as an attempt to step in the right direction; an error is simply an action that is incompletely or improperly specified. ... Try to support, not fight, the user's responses. ... Design explorable systems."[22]

Affordance-Rich Messages (ARMs) can make it possible to design "explorable" systems.

4.2 Programming the Network

Programming the network means focusing on the messages that are passed along the network instead of the devices sending and receiving those messages. There are a number of reasons this shift from device-orientation to network-work orientation can benefit software architects and developers.

First, in widely distributed networks where components have little to no view into the workings of other components on the network, the message passed between them is the only means "in the world" by which communication is possible. In this light, it make sense to consider a way in which the network itself can be 'programmed.' It is through messages that both sender and recipient share understanding and the network is where this understanding lives.

Second, programming the network is not only possible, it is essential in order to support networks as they grow in both the space and time dimension. Think of the possibility of communicating with devices at great distances (i.e. in outer space). It is reasonable to include affordances and additional options within messages that might not reach their recipient for minutes, hours, or even days in order to allow the recipient to engage in additional evaluation of availble options before making a (local) decision. As the reach of the network grows, the messages carried by that network need to be more rich and informative.

Finally, by adopting a paradigm where the messages contain 'programming code', the network of machines that touch the message as it moves along to its final destination can participate in the communication. The HTTP protocol today is designed to support one level of communication at the network level (the HTTP Header space) and one at the sender-recipient level (the body). Manipulating the message metadata sends signals to intermediaries along the network path; signals devices may understand and act upon when appropriate. This, too, is programming the network.

The network is 'programmable' today using ARMs and message metadata over HTTP.

4.3 A Working Model

What would a working model of a programmable network look like? We already have all the technical tools needed to make this a reality. What is needed is to delineate the necessary parts of a working system and show how they can be used to support the proposed paradigm.

The following elements of a working model for programmable networks are described:

- ARM-Aware Network
- ARM-Capable Devices
- ARM Design Model
- ARM Evaluation Model

NOTE: While this section of the paper describes the working model using HTTP as the protocol, ARM-style communication is protocol-agnostic as it sits atop the transfer protocol. As HTTP changes and/or new transfer protocols become available, the ARM paradigm can still be a viable model for programming the network.

4.3.1 An ARM Network

ARMs are of benefit when network communication may span notable distances. When these distances (either in space or time) are long enough to be noticeable by either the recipient or sender (i.e. messages take more than a few seconds to travel between parties), enriching the message with affordances that explain what the recipient can do adds benefit. This includes allowing recipients to store (and possibly forward) messages for later use.

HTTP request/response today has all the properties needed to support this aspect of programmable networks. The HTTP header space contains enough information to know whether the message is fresh or stale, the origin of the message, etc. HTTP responses may also include a body (the part that holds the ARM) which recipients can parse, process, and act upon independent of the sender who originated the message.

A network of machines that understands HTTP can be an ARM-Aware network.

4.3.2 ARM-Capable Devices

Currently most connected devices utilized either a generic HTTP browser or a 'native' application with a built-in HTTP library in order to communicate along the network. ARM-Capable devices would be able to leverage available support for a protocol (i.e. HTTP) plus one or more ARM processors. This is close to the way Web browsers work; they have strong support for the HTTP protocol and a limited number of affordance-rich media types (i.e. HTML). However current browsers do not easily support adding new media-type processors. ARM-capable devices would be able to treat ARM processors as 'plug-ins' and make it easy to 'upgrade' a device by adding new ARM processors.

ARM-capable devices are ones that not only have strong support for one ore more network protocols, they also have support for multiple message models and/or can add new message models as they become available.

4.3.3 ARM Design Model

In an environment where networks support ARM-style communication and devices can support new ARM processors as they become available, having a clear design model for affordance-rich messages is critical. Luckily, one already exists: Hypetext media types or Hypermedia Types.[29]

```
<root>
  <customer id="123">
    <name>Smith, Inc.</name>
    <region>South</region>
    <balance>1000</balance>
  </customer>
</root>
```

Example 1: Non-ARM Response

Media types with native hypermedia controls provide the affordances needed to support ARM-style designs (compare Examples 1 & 2). The author has previously identified a candidate set of these affordances as H-Factors in the book "REST: From Research to Practice."[29] Additional material on a design methodology for Hypermedia Types was detailed in "Building Hypermedia APIs with HTML5 and Node."[2]

```
<root>
  <customer id="123" rel="item" href="...">
    <name>Smith, Inc.</name>
    <region>South</region>
    <balance>1000</balance>
    <link rel="edit" href="..." />
    <link rel="collection" href="..." />
    <link rel="search" href="..." />
  </customer>
</root>
```

Example 2: ARM-style Response

Designing messages that carry affordances is, essentially, designing a language through which clients and servers can share understanding. Designs can be targeted (See Example 2) or very general (See Example 3). The design style chosen should fit the needs of the network participants and the nature of the problem domain.

```
<root>
  <item id="123" rel="customer" href="...">
    <data name="name">Smith, Inc.</data>
    <data name="region">South</data>
    <data name="balance">1000</data>
    <link rel="edit" href="..." />
    <link rel="collection" href="..." />
    <link rel="search" href="..." />
  </item>
</root>
```

Example 3: General ARM-style Response

4.3.4 ARM Evaluation Model

Producing affordance-rich messages is only helpful if network participants can "understand" and use them when they arrive. A consistent evaluation model for ARMs is needed; one which clients and servers can count on when attempting to use ARM-style responses.

Donald Norman's seven stages of action (see above) provides an likely candidate for evaluating (and acting upon) ARM-style responses. Network participants can be coded to perform the same general steps as humans when interacting with the environment:

1. Identify a goal
2. Establish a set of tasks to reach that goal
3. Execute the identified task
4. Capture the results of that action
5. Evaluate the captured results
6. Compare the results to the identified goal

These steps can be applied to response messages on the network by creating ARM processors that can identify goals, establish, execute, and evaluate the results of tasks performed to reach that goal. This is the heart of any goal-seeking state machine. There are quite a number of possible ways to implement ARM processors. It could be done using a human to perform the stages directly or via automation by relying on 'crowd-sourcing' logic for the establishment of tasks and the evaluation of the results.

By treating each ARM design separately and providing processors that understand the ARM 'language', connected devices can become active participants in the programmable network.

5. RELATED WORK

While much of the ideas in this paper have been identified previously, there are no tangible ARM-style implementations extant on the Web today. There are however, some encouraging examples. The items mentioned here fall short of the paradigm of 'programming the network through affordance-rich messages' but they do represent attempts to solve the same problem or mitigate similar perceived shortcomings in the current implementation models.

5.1 Web Intents

Paul Kinlan's Web Intents[14] "is a discovery mechanism and extremely light-weight RPC system between web apps, modeled after the similarly-named system in Android." While a decidedly RPC-style approach (as opposed to the ARM-style described in this paper), the general aim of Web Intents is similar. Clients are able to discover and register with providers to handle actions using a declarative model from within the common browser. Like the ARM-style paradigm, client applications can 'augment' their ability to handle affordances as they appear within responses.

5.2 ql.io

Ebay's `ql.io`[25] project is a "Declarative data-retrieval and aggregation gateway for quickly consuming HTTP APIs." Even though this project does not use affordances as means to interact along the network, it *does* provide a similar service to client applications; a gateway to normalize Web API interactions. This service, like ARM-style designs makes it possible for client applications to better utilize remote services on the network.

5.3 Hypertext Application Language (HAL)

The HAL media type "is a lean, domain-agnostic hypermedia type in both JSON and XML, and is designed specifically for exposing RESTful hypermedia APIs" and it could be used to craft ARM-style responses. HAL defines a small set of hypermedia affordances (Resources and Links) and allows designers to use Link Relations[17] to add semantic

meaning to the representations. In this way, HAL represents a tangible example of a message design aimed at increasing the reliance on affordances used in network communication.

6. CONCLUSION

Moving from an object-oriented API paradigm to a network-oriented affordance paradigm allows software architects and developers to begin programming the network using affordance-rich messages (ARMs) instead of using traditional functional APIs to program the devices connected to the network. This requires a working model where 1) the network is able to support ARM-style responses (which HTTP does today), 2) connected devices understand not just the transfer protocols in use (HTTP, FTP, IRC, etc.) but also the ARMs being transferred, 3) message designs that allow developers to successfully map actions to affordances, and 4) a message evaluation model that follows Norman's seven stages of action.

While the current Web can support this paradigm, there are only faint examples of this model emerging at this time (i.e. Web Intents, `ql.io`, HAL, etc.). What is needed at this time is an increased focus on coding clients that can support ARM evaluation and a parallel increase of new ARM-style implementations and message designs.

7. REFERENCES

[1] Alexander, Christopher. In *A Timeless Way of Building*, Oxford University Press, New York, NY USA, 1979

[2] Amundsen, Mike *Building Hypermedia APIs with HTML5 and Node* O'Reilly Media, CA USA 2011

[3] Appcelerator *http://www.appcelerator.com/*

[4] Beck, Kent and Cunningham, Ward. *Using Pattern Languages for Object-Oriented Programs* Presented at OOPSLA-87, September 1987 *http://c2.com/doc/oopsla87.html*

[5] Campbell, Joseph. *The hero with a Thousand Faces* Pantheon Books, USA 1949

[6] Christensen, Erik, et al. *Web Services Description Language (WSDL) 1.1* W3C Note, March 2001 *http://www.w3.org/TR/wsdl*

[7] Davis, Doug, VP of the Digital Enterprise Group and general manager of the Embedded and Communications Group at Intel. In *Intel Inside Becomes Intel Everywhere*, Mar 2, 2009 *http://g.mamund.com/nxmnj*

[8] Fielding, Roy T. *Architectural Styles and the Design of Network-based Software Architectures* Ph.d dissertation, University of California, Irvine, 2000. *http://roy.gbiv.com/pubs/dissertation/top.htm*

[9] Fielding, Roy talking to Richard Morris. In *Roy Fielding: Geek of the Week*, August 2010 *http://g.mamund.com/dozcf*

[10] Fielding, Roy T. In the presentation *A Little REST and Relaxation* for ApacheCon 2008, November 2008. *http://g.mamund.com/hfgdl*

[11] Gibson, James J. *The Ecological Approach to Visual Perception* Psychology Press, New Yorik NY USA, 1986

[12] Golberg and Reuben. *Smalltalk-80: the language and its implementation* Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA 1983

[13] Grossman, Lev, Time Magazine. In *Apple's New Calling: The iPhone*, January 10, 2007. *http://g.mamund.com/qgkxr*

[14] Kinlan, Paul On Github *http://g.mamund.com/hanql*

[15] Korzybski, Alfred. *Science and Sanity* Colonial Press Inc., Clinton, MA USA 1933

[16] Hadley, Mark. *Web Application Description Language* W3C Member Submission, August 2009 *http://www.w3.org/Submission/wadl/*

[17] IANA Link Relations *http://g.mamund.com/jewho*

[18]

[19] Mandel, Dr. Michael, South Mountain Economics, LLC. In *Where are the Jobs: The App Economy* Whitepaper, February 7, 2012

[20] Microsoft Software Developer Network *Using HTTP as an RPC Transport*, September 2011 *http://g.mamund.com/vxsoc*

[21] Murray, Bill as Dr. Peter Venkman In *Ghostbusters.* Dir. Ivan Reitman. Columbia Pictures Corporation, 1984.

[22] Norman, Donald. *The Design of Everyday Things* Basic Books, September 2002

[23] PhoneGap *http://phonegap.com/*

[24] Programmable Web, The In*4,000 Web APIs: What's Hot and What's Next?*, October 3, 2011. *http://g.mamund.com/qspml*

[25] QL.IO *A Declarative data-retrieval and aggregation gateway for quickly consuming HTTP APIs http://ql.io*

[26] Shogren, Elizabeth for NPR. *Don't Trash Or Stash Old Cell Phones; Recycle Them*, April 2010. *http://g.mamund.com/yqeuz*

[27] R. N. Taylor, N. Medvidovic, K. M. Anderson, E. J. Whitehead, Jr., J. E. Robbins, K. A. Nies, P. Oreizy, and D. L. Dubrow. *A Component- and Message-Based Architectural Style for GUI Software* IN *Transactions on Software Engineering*, pages 390- 406, June 1996.

[28] Tyagi, Sameer for Oracle Technology Network. *RESTful Web Services* August 2006 *http://g.mamund.com/cxkow*
Vestberg, Hans President and CEO of Ericsson. In *CEO to shareholders: 50 billion connections 2020* Press Release, April 13, 2010 *http://g.mamund.com/zkjww*

[29] Wilde, Erik Pautasso, Cesare, Eds. Chapter 4: Hypermedia Types in *REST: From Research to Practice* Springer; 1st Edition. edition August 2011