

Enabling Technologies and Design Values for Building the Real World Web

Donald F. McMullen

*University of Kansas
MRB 160, 2030 Becker Drive, Lawrence KS, 66047 USA
e-mail: mcmullend@ku.edu*

(Received: 9 December 2008; published online: 25 March 2009)

Abstract: The notions of ubiquitous computing and networking, and the increasing availability of compact, low power sensing technologies naturally lead to the idea that we can expect to see large numbers of sensors embedded in the fabric of our everyday lives and that these could form a “Real World Web” of real-time data sources about our world. Implications for e-Research include sharing of real-time data from scientific instruments and aggregation of many types of information to answer complex questions as they arise. The World Wide Web was made possible through a combination of the increasing availability of the Internet, simple network protocols, and open content and delivery standards. The Real World Web can only grow and become self-sustaining if we pay attention to these same core design values of simplicity and openness. Recent developments in cloud computing and in Web 2.0 technologies and design stances provide enablers from which to build the Real World Web, but in addition, require a shift in thinking away from a classical Web services and layered standards model. This paper explores these issues and the role of the Real World Web as a paradigm for sharing instruments, sensors and other real-time data sources in e-Research collaborations.

Key words: Web 2.0, Real World Web, remote instrumentation, sensor networks, CIMA, e-Science, Web services

I. INTRODUCTION

The notions of ubiquitous computing and networking, and the increasing availability of compact, low power sensing technologies naturally lead to the idea that we can expect to see large numbers of sensors embedded in the fabric of our everyday lives. These sensors can potentially monitor our health, keep tabs on our children and aging parents, track the quality of our environment and warn us of unsafe conditions, assess the impact of earthquakes and storms on our houses, provide detailed diagnostic information about our cars and appliances and help us find qualified technicians at the lowest possible cost. These are just a few possibilities for a Real World Web that narrows the gap between the Web and the real world, but there are many issues to be considered to achieve this vision. The World Wide Web was made possible through a combination of the increasing availability of the Internet, simple network protocols, and open content and delivery standards. The Real World Web can only grow and become self-sustaining if we pay attention to some core design values of simplicity, openness, and ethics. Recent developments in cloud computing and Web 2.0 technologies and design stances provide enablers from which to

build the Real World Web (hereafter abbreviated RWW) but they also require a shift in thinking away from a classical Web services and layered standards model.

Standards for sensor and actuator networks such as Controller Area Network (CAN-bus), Modbus, and EtherNet/IP are protocols for interacting with automotive, industrial and manufacturing equipment over secure private or in-house networks. Protocol implementations and end-equipment (sequencers, sensors and actuators) are available but these fall into narrow, specialized markets such as automobile engine control systems, manufacturing equipment and industrial process control. Other market spaces such as home automation and security, automotive entertainment electronics and laboratory automation have developed parallel but largely non-overlapping design spaces, protocols and capabilities. On the industrial automation side these standards are generally aimed at supporting low-level register and bit operations, and programming state machines (programmable logic controllers). For consumer electronics the emphasis has been on short-range wireless networks (e.g. IEEE 802.15.1/Bluetooth, IEEE 802.15.4, Zigbee and One-Net) and automatic configuration of a small number of specific device types relevant to each market.

In parallel but largely in isolation from industrial and consumer real-time sensing and control applications, a number of projects have worked to develop open standards for sharing instruments and sensors in e-Science and e-Research. Solutions are now available to develop portable hardware descriptions such as sensorML [1], and for open instrument services and access protocols such as the Common Instrument Middleware Architecture (CIMA) [2], and IEEE 1451 [3]. These systems cover a variety of design stances, from a focus on embedded devices and minimalist protocols to a full service oriented architecture, open protocols and self-describing hardware, all important in an open environment that values collaboration and sharing of hardware and information resources.

One approach to including a broad range sensing and control systems from industrial, consumer and scientific applications in the RWW is to view these as “service clouds” accessed through a gateway that incorporates more broadly used Web protocols and Web 2.0 methodologies on the “Web” side of the Real World Web. This notion of hiding the complexity inside service clouds is central to our architecture for an emergent RWW and will be discussed in more detail below. As appealing as this approach is for managing complexity it presents some challenges in supporting both polled and event driven access to real-time data sources. On the positive side, the ubiquity of streaming media services for on-demand video and audio provides an excellent model for real-time real-world sensor data.

II. THE REAL WORLD WEB AND UBIQUITOUS COMPUTING

Sensors together with their associated transducers and control systems are broadly embedded in our environment and increasingly becoming an indispensable part of our everyday lives. They make our cars run efficiently, keep our indoor climate in perfect balance, monitor our vital signs when we are ill or injured, and continuously monitor the weather. The continuous increase in computing power combined with ever-greater levels of on-chip integration and resulting size reductions inspired work in the 1990s by Mark Weiser [4] at Xerox PARC on Ubiquitous Computing (UbiComp). The core of ubiComp is the idea that computers are so small, cheap and powerful that every artefact in our environment, from toasters to light switches should not be without one. Yet ubiquitous computing promises to bring many orders of magnitude more sensors and other measuring devices into our “data view” of the world. There are a number of technical hurdles to overcome for the vision of

ubiquitous computing and sensors can be realized. Issues such as fabrication, programming and networking notwithstanding, there is another, larger problem of giving meaning to the gigabytes of data all of these sensors produce.

Neil Gershenfeld, in his book *When Things Start to Think* [5], articulates a vision of sensing, computing and networking embedded in everyday objects. These objects will exchange information among themselves and with humans, leading to a blurring of the line between bits and atoms, and between human intentionality and machine behaviours. Long before we achieve this state of bliss we can see material returns on embedded intelligence. Sensor networks can provide large amounts of useful information with the potential to improve the quality of our lives if these information services are universally accessible and easy to use. Moving materially toward an open and dynamic Real World Web and an economy of real-time real-world information will require advances on several fronts: core technologies for constructing small, power efficient sensing devices that form the basic building blocks of the Real World Web; standards for locating and describing the functions of sensing devices, and communications technologies that allow the placement of sensor packages in any environment.

Emerging enablers of this future are low power, low bitrate wireless and wired networking technologies, increasing density of flash memory, location awareness services like the Global Positioning Satellite system and advances in low cost sensing and transducing technologies. Wireless technologies such as ZigBee combined with ad hoc mesh routing provide a highly resilient local communications environment. Getting beyond local communications is a key issue. In one approach data from many small sensors with wireless interfaces can be aggregated and sent out to the Internet through special gateways. Although this is a functional way to bridge sensing and communication technologies across domains defined by physical access media, the use of gateways limits possibilities for direct access to and fine-grained use of individual sensors and actuators.

IPv6 is an extremely promising high level protocol for sensor networks because of its huge (2^{128}) address space and broad availability. With increasing on-chip memory and processing power IPv6 may also be a candidate for tiny, embedded processors. 6LoWPAN is a specification for IPv6 over ZigBee (802.15.4) at low bit rates (RFC 4919, RFC 4944). A number of routing protocols are under development for 6LoWPAN to implement ad hoc and hierarchical routing through an 802.15.4 PHY and between ZigBee and other link layers for IPv6.

Alternatively there are proposals such as Gershenfeld's "Internet 0" or I0 [6] proposal for encoding IP packets in a wide range of link layers such as low power RF, power lines, and even printed bar codes. With I0 devices can send and receive IP packets using whatever link layer is appropriate, so protocol translations are unnecessary at boundaries between media types. I0 is made possible by embedded system processors on which an IP stack can be efficiently implemented. Unless IPv6 is used, with its very large address space and self-addressing capability, the I0 approach has significant scaling problems.

Location awareness through GPS and cell phone related technologies such as Assisted GPS (using cell towers to triangulate the position of a phone when GPS signal strength is inadequate or unavailable) and Enhanced Observed Time Difference is another key source of "sensor data." This information tells the owner of the device and potential service providers where the device is located, and hence what services might be relevant and available for use. Human movement is a deep indicator of intention and detailed knowledge of location and its derivatives can be used in many ways to improve the quality of our lives. The availability of "location aware" services and the precursor location information services they are built on will radically change the way we live.

The third class of enabling technology, low cost sensors and actuators, forms the all-important interface between the real world and the world of bits. Low cost is critical to ubiquity, but equally important are notions of identity (the location and nature of the measurement are known to a high degree of certainty) and quality (the digital representation from the sensor is a faithful mirror of the world.) Without standards for characterizing the real world-digital world interface by describing identity and quality metrics for a measuring device and means to verify these to build trust, sensors embedded in the environment cannot effectively form the basis for a real-time economy of information about the world. Until such standards exist the scope of sensors and the semantics of the data they produce will necessarily be limited to the individual products they are embedded in. Conversely, having and using standards such as sensorML [7] and the CIMA instrument ontology [8] to describe the location and functionality of sensors will make any sensing device much easier to integrate into a Real World Web of global scope.

Power is a critical design dimension that we will not discuss in detail except to note the apparent tradeoffs between processing power, memory capacity, wireless communications range and total system power consumption in wireless devices. Wired devices perhaps do not have the same set of issues, but require simplified connection

schemes where power and communications can be provided through the same physical wire as a simplifying principle. When considering architectures for embedding devices in structures and vehicles (houses, buildings, cars, etc.) using standard affordances such as power plugs (e.g. X10 home automation) or built in rails that provide power and network connectivity to snap-in sensor and actuator modules. The Media House, a joint project of the Metapolis Group from Barcelona, the MIT Media Lab, and the Fundacio Politecnica de Catalunya [9, 10] is a recent example of using this approach to create a dynamic, highly functional interior space where the structure of the building itself provides the network physical layer through which sensors and actuators in the interior space communicate.

III. BUILDING THE REAL WORLD WEB: SERVICES FOR INSTRUMENTS AND SENSORS

More than sensing hardware and network interfaces are needed to enable the Real World Web. The success of the World Wide Web was due in part to the simple protocols and standards and open architecture used to define access to content. In the same way, the success of the Real World Web will depend on availability of simple but powerful approaches to present real time data sources as services and a service oriented architecture for deploying and accessing nodes.

One approach to a standard service model for sensors is the Common Instrument Middleware Architecture (CIMA) [11-13] aimed at developing a Service Oriented Architecture (SOA) using SOAP-based Web services to make instruments and sensors network accessible. CIMA provides a standards-based, uniform way to interact remotely with instruments and the data they produce. CIMA addresses the following issues in remote access:

- standardization of the network protocol for interacting with instruments, sensors and actuators;
- flexibility in the underlying network transport;
- efficient and high throughput data transport;
- availability of computational, storage and networking resources in the instrument or sensor controller;
- co-evolution of instrument and network interface design; and
- reuse of data acquisition and processing codes.

Some requirements for an instrument middleware are explored by Devadithya [11] and McMullen [12]. These include self-describing hardware and control software; functional transparency; resource oriented stateful services; interoperability with other data acquisition and transport

systems; efficient data transfer; lightweight with respect to CPU, memory and network resources; support for intermediaries for signal processing and data aggregation; and support for authorization; multiple modes of interaction (e.g. streaming/event based, polled, callback, etc.)

Web Services with SOAP are not the only way to implement an SOA. Web 2.0 provides alternatives to SOAP and WSDL, using instead parameters encoded in HTTP URLs and HTTP verbs PUT, GET, POST, and DELETE to interact with a service and return data or status as HTML or more generally as plain old XML. The Representational State Transfer (REST) architecture [14, 15] is often used in Web 2.0 as a simple interface to network services. We believe that REST/HTTP is a compact and efficient alternative to RPC style XML-RPC or SOAP and Web services, and fits well with the design ideal of simple, user-oriented service interfaces exemplified in Web 2.0. Services such as Amazon Elastic Cloud and Simple Storage Service, Yahoo Pipes and many other services use REST and HTTP to provide lightweight easy to use interfaces. As we discuss below, Web 2.0, although an uncoordinated activity, does in fact provide a comprehensive distributed computing architecture that challenges both Grids and Web Service architectures. In our usage, Web 2.0 and Cloud Computing are complementary concepts. The computing, data, or instrument “cloud” is internally composed using many tried and true distributed computing techniques. These are hidden from the majority of users. Instead, the user’s view of the cloud is through Web 2.0 services, capabilities, and messages.

With the emergence of Cloud Computing [16, 17] and the development of systems with potential e-Science applications that leverage cloud services such as Google MapReduce [18] and Hadoop (lucene.apache.org/hadoop/) for data intensive computing, Amazon EC2, S3 and SimpleDB services (aws.amazon.com) and Microsoft’s mash-up creation tool Popfly [19] (www.popfly.ms) there are new possibilities for integrating instruments, sensors, and other measuring devices embedded in the real world with services in this new paradigm. Instruments and sensors can and must be available in such a way that they can be composed with other “cloud” services.

There are also significant opportunities for defining protocols and service interfaces that will promote the construction of the Real World Web in a broader sense, making it possible for individuals and groups to add real-world sensing hardware to the web incrementally in a way that promotes universal access to web-based sensor data as well as aggregation across different sensor types and locations, cross validation of readings, multiple re-use of

the data and creation of value added services that aggregate, filter, or monitor primary sensor data.

IV. WEB 2.0 AND WEB SERVICES

Grids and Web Services provide one model for a service oriented architecture for sensors. As a starting point we believe the development of the Real World Web, since it has both physical and computational components, will depend on the actions of a large number of individuals, who are perhaps more interested in the data their contributions to the RWW produce than in the details of the services needed to provide that data. Sensors actually have to be placed somewhere and maintained in order to produce interesting and useful data.

An examination of the engineering principles for building Grids is useful for understanding why the Grid model is perhaps not the best approach for a decentralized and emergent activity like the development of the Real World Web. [20] One critical success factor for the growth of the Web was the low barrier for entry for new content developers due to the simplicity of HTML and its inherent support for incremental development.

The Web Service software engineering model has dominated Grid computing since the Open Grid Service Architecture model appeared in 2001 [21, 22]. In the intervening years the OGSA approach has been modified removing distributed object architectural styles [23] and promoting instead the “WS-I+” approach [24]. Although conceptually valid, it has turned out to be based on a questionable principle: that Web services specifications themselves would be well-designed and well-supported by the software community. The record is mixed. WSDL and SOAP have been relatively stable and well supported, although support for newer versions (WSDL 2.0 and SOAP 1.2) has been limited. On the other hand, a workable, generally acceptable information services model has never emerged: UDDI has had little success and yet no alternative has appeared. Development of Web services security has been slowed by poor performance and the search for solutions to difficult problems such as canonicalization. There are also the issues of too many competing standards (e.g. WS-Reliability and WS-ReliableMessaging) and complex dependencies among existing and proposed specifications that can create difficult global problems when these specifications change [25, 26]. There are more than 60 Web Service (WS-*) specifications and proposals, and although WS-I (www.ws-i.org) is intended to establish interoperability

profiles to remediate the explosion of standards and reduce the confusion progress has been slow.

Grids and Web Services are aimed at supporting many aspects of resource sharing and supporting individual and group work across different organizations. Although a laudable goal, the results have been extremely complex and difficult to implement architectures that are arguably not appropriate to solving the problems of helping users to work with computing resources or with each other through the infrastructure. There is a need for a more user-centric model for distributed services, and, in terms of Cloud Computing, how users interact with cloud services.

Another problem is that Web services follows rigid software engineering models typified by the OASIS process (www.oasis-open.org). The specification process is slow, and the results are sometimes irrelevant when the process finally emits a standard. The process is further complicated by the proliferation of overlapping, competing standards passing through development and review in parallel. The overall result is a software engineering model that is slow and cumbersome, and which requires the use of complex, sophisticated tools to implement end-user services and applications. The implications for using this approach to develop open, flexible and easy to implement service standards for the Real World Web are not encouraging.

As shown in Table 1 and argued elsewhere [27], Web services architectures possess the general characteristics needed for network programming of distributed services. However, as discussed above, some core processes for producing Web services layered specifications have turned out to have shortcomings suggesting that an alternative methodology is needed. One alternative is the “Web 2.0” approach. Before considering the software development principles of Web 2.0, we first define it and compare it to Enterprise-style Web services in Table 1 to establish that, even though it is an emergent activity, it fulfils the core requirements of a distributed computing model.

The contents of Table 1 are based on an analysis by Pierce, Fox et al. [27] comparing characteristics of network services based on Web Services and Grid techniques to a Web 2.0 approach to SOAs. Web 2.0 presents an attractive model for the emergent construction of the Real World Web largely because it focuses on simple, clear user APIs and widely used applications rather than complex, difficult to use standards with marginal or missing implementations [28].

Web 2.0 follows a development model that accounts for different requirements between internal and user-facing services within a service cloud. Within a given service cloud (the Amazon Elastic Cloud for instance) there is a need for a rich set of capabilities supporting identity

management, messaging, resource allocation, etc. The user, in contrast, is presented only a straightforward and clear task-oriented interface that defines only what is needed to use the service. Furthermore, the details of intra-cloud services and capabilities do not need to be consistent or interoperable with the inner details of any other service cloud. This separation of cloud service functionality into

Table 1. Network programming concepts and their realizations in Web Services and Web 2.0. (After Pierce, Fox et al., 2006 [27])

Concept	Web Services and Grids	Web 2.0
Service interface	Interfaces expressed as WSDL. WS-* layered specifications for security, reliability, addressing, etc.	API is HTTP GET, PUT, POST and DELETE operations on URL resources. Services are based on REST model. Line security through SSL.
Service state	WSRF used for stateful services.	Services are stateless
Network messaging	SOAP carries XML message payloads. SOAP header extensions implement layered specification capabilities such as security and reliability.	XML content such as RSS and Atom, or more compact formats like JSON are exchanged as HTTP payloads. Notifications and events through client polling of external services, e.g. Amazon’s Simple Queue Service.
Service consumers	Consumers use SOAP Web services calls to retrieve data. This is usually done on a server and data aggregated for presentation through a portal.	Consumers use URLs and HTTP verbs to retrieve data. This can be done by browser/client-side applications and mash-ups. Content is aggregated by the client.
Service composition	Composition through workflow systems executing workflow specifications with steps based on individual services.	Mash-ups combine URL-based RESTful services into applications. Mash-ups are implemented as JavaScript interacting with remote services using AJAX.
Identity management	Identity is provided and managed by an organization that owns the services. Individuals must join the organization and be credentialed by it to use a service or resource.	Identity via participation in an emergent social network of individuals with shared interests. Credentials may originate outside the group e.g. Google ID or YahooID. Access to multiple services in a mashup may require multiple credentials.

internal, hidden complexity and external user-facing simplicity is a core design value of Web 2.0. Web 2.0 services clearly distinguish between the potentially complex interiors of service and resource clouds and their simplified boundary interfaces that are used by the larger community of developers. In contrast, software engineering for Web Services treats all services with the same degree of maximal complexity.

From the software design and engineering perspective, we may make the following general observations. First, Web 2.0 development is geared toward the end user, illustrated by the common Web 2.0 concept of the “mash-up”. Mash-ups are composite applications made from several online services, typically consisting of both community and user-provided data (for example, personal photos placed on an online map).

Second, Web 2.0 substitutes asserted programming interfaces in existing scripting languages for e.g. WSDL in Web services. There are no universal API or interface definition languages in Web 2.0, only ad hoc implementations in existing programming languages. Using simple network message formats makes this feasible, eliminating the need for complex message handling.

Third, Web 2.0 introduces the concept of the perpetual beta [29]. That is, online services and capabilities may be continuously upgraded while maintaining backward compatibility with older clients.

Although it is possible to use the term “perpetual beta” pejoratively, we note that this applies to the boundary interfaces (at the user level) rather than the internal component implementations and orchestrations inside the cloud.

Fourth and finally, Web 2.0 services and capabilities are competition-driven. Mash-up competitions thrive as developers try to outdo one another. Feedback helps refine service interfaces and capabilities in a spiral development model. Contrast this to the slow, tedious and failure-prone approach to the development of Web services layered specifications before any services based on them can be imagined and implemented.

Web 2.0/Cloud approaches suggest an asymmetric software design model. The interiors of computing resource clouds can be complex and sophisticated, and traditional design methodologies are appropriate. On the other hand, the user interactions with the cloud must be agile and iterative, demanding different design methodologies.

IV. SERVICES FOR THE REAL WORLD WEB

We are at a crossroads where the fundamental principles for distributed system design exemplified by Grid

middleware and the larger Web services suite of standards, as well as the process for developing interoperable standards embodied by the OASIS consortium (www.oasis-open.org), need to be re-examined. In the light of the tremendous growth of “cloud” services that value rapid delivery of simple, transparent, composable functionality, and the wholesale adoption of this approach as indicated by the daily appearance of hundreds of value-added mash-ups of these services, it is important to assess the relevance of this paradigm shift to the development of sensor networks in general and the Real World Web in particular. Applying Web 2.0 design values of simplicity and transparency to promote an emergent development of the Real World Web leads to the following design guidelines:

- Sensing and measuring hardware nodes in the Real World Web (RWW) are represented by network-accessible services.
- RWW device services for sensors and instruments are based on simple, widely used protocols such as HTTP.
- RWW sensor and instrument service APIs fully leverage stable features of the protocols that carry them.
- RWW Service APIs simple and clearly documented.
- Service APIs should provide a minimal but expandable set of core functionality.
- APIs should be able to evolve over time to reflect new core capabilities.

V. THE COMMON INSTRUMENT MIDDLEWARE ARCHITECTURE (CIMA)

As an example of the application of these design values we are currently engaged in re-implementing the Common Instrument Middleware Architecture (CIMA) [2] to better support our Web 2.0 approach to e-Science. CIMA is a service oriented architecture that provides instruments and sensors with network interfaces based on Web services. CIMA consists of three main parts: a Service Implementation (SI) that acts as the instrument’s network interface, an XML-based protocol for accessing and controlling instruments, and an extensible vocabulary in OWL-DL for describing instruments and sensors. The Service Implementation is a protocol engine and life cycle management system that is common to all CIMA instruments. It supports plug-ins, which are code templates customized to access specific sensors or actuators associated with an instrument and which are loaded at the time the Service Implementation starts. The Channel Protocol is

used to access plug-ins and control the SI and to transport data from sensors to a consumer and consists of exchanging XML “Parcel” messages that are interpreted by the SI. The Channel Protocol defines a few basic message types defining operations on the “Channel” or connection to a plug-in such as “Register”, “Get” and “Set”. These operations are implemented as tag values in the Parcel tied to functionality in the SI and usage models, and new operations can be added as needed [30, 31]. The SI in the Web services version of CIMA provides a Web services document literal endpoint that accepts a control message as single well-formed XML fragment. If it is essential for an SI to validate messages sent to the SI XML document validation can be used, but this limits interoperability of messages when the protocol is modified to add new tags, and reduces performance. Sensor or status data and associated metadata are sent from the SI as a “data” type Parcel as the body of a SOAP message.

As both the producer (instrument) and consumer of CIMA messages are Web services endpoints the architecture is inherently event driven and once a relationship between a consumer and a CIMA data source is established data can be streamed continuously from producer to consumer or sent at whatever frequency is desirable. This built-in “data flow” capability makes it possible to set up intermediaries that can aggregate, filter and scan data from one or multiple streams from CIMA instruments.

In summary the Web services version of the CIMA architecture provides instrument, sensor and actuator proxies that:

- support fully asynchronous event driven interactions between user applications and instruments;
- leverage available layered (e.g. WS-*) specifications for security, addressing, reliability, etc.; and
- produce and consume XML message payloads that are transparent, easily parsed and can be validated for additional fault tolerance.

Although the CIMA Channel protocol has primarily been implemented using SOAP over HTTP, other protocols can be used for sending and receiving SOAP messages, and CIMA services that use Java Message Service (JMS), BRTT Antelope, and the Kepler workflow system have been implemented.

VI. CIMA, WEB 2.0 AND THE REAL WORLD WEB

As the discussion has developed to this point, we postulate that Web 2.0 approaches will improve collabora-

tion and reduce the effort of developing network services for e-Research by reducing reliance on complex tools and protocols (e.g. Web services and WS-* layered specifications) and instead build services from simple, well characterized components such as HTTP, SSL, and REST models for services. The value of building e-Science services using Web 2.0 techniques has been demonstrated in a number of recent projects [20, 27, 28].

To tie together the strands of Web 2.0, the Real World Web, and instruments and sensors that form the basis for many e-Science collaborations we now consider how to include instruments and sensors in Web 2.0-based systems for e-Research. As a starting point we will discuss how to update the CIMA approach to function appropriately as a Web 2.0 service.

Table 2. Issues in the design of a Web 2.0 CIMA service

CIMA Characteristic	Web services	Web 2.0
1. Interface	Web services/ SOAP/WSDL	REST/URL
2. Communication between instrument proxy and consumer	Asynchronous, event driven, streaming data	Synchronous via server push or asynchronous with polling; event driven if consumer has embedded HTTP server
3. Transport	SOAP/HTTP, document-literal	HTTP
4. Security	SSL and WS-* layered specs	SSL or payload encryption by CIMA service
5. Identity management, authentication and authorization	Internal to the CIMA service, WS-* layered specs, Shibboleth	Internal to the CIMA service, External through SSO service like Shibboleth
6. “Parcel” payload	XML document	XML fragment or MicroFormat in HTML body
7. Instrument description	OWL-DL/RDF embedded in XML document	OWL-DL/RDF in HTML header

Table 2 lists implications of some key differences between a Web services and Web 2.0 with regard to the design space for the RWW/Web 2.0 version of a CIMA-based

instrument service. The immediate challenge is mapping CIMA as a pair of Web services on the instrument and the consumer application to a RESTful service only on the instrument proxy. In REST terms, the Web 2.0 version of the CIMA service is a resource identified by a URL. Synchronous CIMA Web services calls translate as HTTP GET/PUT operations on a URL that contains the identity of the plug-in and other salient information that would have been contained in the Parcel XML document used in the Web services version of CIMA, or by POSTing a Parcel in the body of the request. In general the REST operations are synchronous and need to return something before the underlying HTTP protocol at the requester times out the request. Asynchronous completion of long operations with notification that are supported easily in the Web services version of CIMA such as commanding a positioning system to move to a specific location can be simulated through client polling. The Web 2.0 CIMA Service Implementation needs to support these kinds of “stateful” operations in a possible violation of the notion that RESTful services are idempotent. The concession is that instruments, sensors and actuators are for the most part stateful.

Table 3 illustrates two possible mappings from CIMA Web services and REST, one based on HTTP GET and PUT where data that were in the XML Parcel in the Web services version are now encoded into a URL, and another which uses HTTP POST to a simple URL where the payload of the request is a CIMA Parcel. The service URL in the GET/PUT implementation consists of the fully qualified domain name or IP address of the service, the CIMA function name, the plug-in and target variable in that plug-in, an identifier for the client’s session with the instrument service, and other data needed specifically for the *Register* (requests the CIMA service to return data periodically to a URL), *Set* (set a variable associated with a plug-in), and *Session* (authenticate a client to the instrument service) commands. The instrument service then either returns data periodically through HTTP PUT calls to the client-provided URL or keeps the request socket open and periodically writes new data Parcels to it.

Figure 1 illustrates example REST URLs for a hypothetical CIMA service with a plug-in named “sensor1”. In this example we are supposing a service that is capable of pushing data to the client on an open connection or returning data to a client asynchronously via calls to an HTTP server provided by the client, analogous to the way the Web services version of CIMA operates. As general problems to be investigated it is noted that the HTTP “callback” method of returning data does not fit the general Web 2.0 model, and the server push solution may not be

stable if there are long delays between data sent from the instrument to the client, or if the connection must be kept open for a long period of time. It is possible that systems and technologies for streaming video and audio content are an appropriate solution to these general problems of streaming data over HTTP.

Table 3. Mapping CIMA Parcel functions into a REST service

HTTP verb \ CIMA Function	GET	PUT	POST
Register (client request for streaming data from a plug-in)	Returns: Multiple Parcels or HTTP error code OR Returns data via HTTP calls to client	X	Input: Parcel XML Returns: Multiple Parcels or HTTP error code
Get (return a plug-in’s variable)	Returns: XML Parcel containing value	X	Input: Parcel XML Returns: Parcel + HTTP status code
Set (set a plug-in’s variable)	X	Returns: HTTP status code	Input: Parcel XML Returns: Parcel or HTTP error code
Describe (the instrument)	Returns XML Parcel w/description	X	Input: Parcel XML Returns: Parcel or HTTP error code
Session (authenticate client)	Returns XML Parcel w/session key	X	Input: Parcel XML Returns: Parcel or HTTP error code

A design feature that CIMA Web services offers is streaming data. This can be mapped into a RESTful Web 2.0 approach as client polling or through server push. Efficient polling requires the client to have a good idea of when new data will be available, so may require some specialized tuning on the client to “sync up” with a data source which

produces data at irregular intervals. Server push and pushlets offers a potential solution but can be subject to instability and protocol timeouts if the instrument service does not send data often enough. Data received by the client must be parsed as it is received to extract individual events (Parcel data from an instrument) which may not be efficient or even possible on some clients.

http://{service-url}/	(base URL for the instrument service)
{CIMA-Function}/	(e.g. Register, Get, Set, Session , etc.)
{plugin-name}/	(for Describe, Register, Get, Set)
{variable-name}/	(for Describe, Get, Set commands)
{session_key}/	
?val=[HTML-escaped-value]	(for Set command)
?url=[callback_spec_URI]	(for Register with callbacks)
(Credentials in HTTP hdr. for Session initiation)	
Examples	
1. Request CIMA REST service to send data from sensor1 to http://my.client.org:9080	
GET http://instrument.your.org/Register/sensor1/b7bf6538-873c-4f9a-84d6-163d40a60632?url=http://my.client.org:9080/	
2. Set variable "filter1" on "CCD" to "Oxygen3"	
PUT http://instrument.your.org/Set/CCD/filter1/b7bf6538-873c-4f9a-84d6-163d40a60632?val="Oxygen3"	

Fig. 1. RESTful URL schema for a CIMA service

A straightforward solution to the problem of asynchronous delivery of data from instrument to client is to duplicate the CIMA Web services model with Web services endpoints on the instrument proxy and in the client software that is using the instrument, but instead of a WS endpoint on the client, provide only the embedded web server. This approach is appealing but it fundamentally breaks with the REST/Web 2.0 approach of URL-based resources and synchronous (idempotent) operations on them, making it difficult or impossible to use these services in mashups. A better solution may be to appeal to external messaging systems such as Amazon's Simple Queue Service or implement data flow using RSS [32]. In this case the instrument service will be expected to provide data through the third-party message service. We have evaluated RSS as a way to deliver streaming data

from instruments and found that it not only works well but has some interesting benefits, such as providing retrospective data to new subscribers, at least to the extent that old data are kept in the RSS feed. Other services such as Yahoo Pipes can be used to filter and aggregate instrument data in much the same way that CIMA Web services intermediaries work.

VII. CONCLUSIONS AND FUTURE WORK

In this paper we describe the Real World Web based on Web 2.0 techniques and design values, and present an argument for developing standards to encourage its development in order to make real-world sensors and instruments more accessible. We then briefly discuss the role of Web 2.0 techniques in building systems for collaborative e-Science. We discuss some particular difficulties of representing instruments and sensors as RESTful services and present a conceptual mapping of the Web services approach used in Common Instrument Middleware Architecture (CIMA) to a REST-based service oriented architecture.

Future work includes an evaluation of GET/PUT vs. POST approaches in terms of performance and integration with other services via mashups, use of MicroFormats instead of XML for CIMA Parcel data structures, how complex preexisting sensor networks can be brought into the Real World Web as service clouds, and models for how authentication and authorization for Real World Web resources should be approached.

Acknowledgements

The author would like to thank his fellow co-investigators, collaborators and students on the CIMA project and to acknowledge their manifold contributions: John C. Huffman, Randall Bramley (Indiana University) and Ken Chiu (SUNY Binghamton); Kia Huffman (Brown University); Marlon Pierce, Geoffrey Fox, Yu Ma, and Gilead Kutnick (Indiana University); Tharaka Devadithya (IBM); Thomas Reichherzer (Enkia Corporation); Peter Turner, Romain Quilici, Doug du Boulay, and Clinton Chee (University of Sydney); Ian Atkinson, Tristan King, Nigel Sim, and Matt Wyatt (James Cook University); Sofia Brenes-Barahona, Nisha Gupta, Carol Deng, and Hunter Davis (Indiana University).

Support for this work from the National Science Foundation is gratefully acknowledged (SCI 0330568, DBI 0446802, IIS 0513768, and IIS 0513687).

References

- [1] G. Aloisio, D. Conte, C. Elefante, I. Epicoco, G. P. Marra, G. Mastrantonio and G. Quarta, *SensorML for Grid Sensor Networks*. Proceedings of the 2006 International Conference on Grid Computing and Applications, GCA 2006, Las Vegas, Nevada, USA, June 26-29, 2006. pp. 147-152. CSREA Press.
- [2] D. F. McMullen, R. Bramley, K. Chiu, H. Davis, T. Devadithya, J. C. Huffman, K. Huffman and T. Reichherzer, *The Common Instrument Middleware Architecture: Experiences and Future Directions* in: *Grid Enabled Remote Instrumentation*. F. Davoli, N. Meyer, R. Pugliese and S. Zappatore, (Eds.) Springer, 2009.
- [3] K. B. Lee and R. D. Schneeman, *Distributed measurement and control based on the IEEE 1451 smart transducer interface standards*. IEEE Transactions on Instrumentation and Measurement 49 (3), 621-627 (2000).
- [4] M. Weiser, *The Computer for the 21st Century*. Scientific American 265 (3), 94-104 (1991).
- [5] N. Gershenfeld, *When Things Start to Think*, Henry Holt and Co., First edition: January 12, 1999.
- [6] N. Gershenfeld, R. Krikorian, D. Cohen, *The Internet of Things*. Scientific American Magazine 291 (4) 76 (2004).
- [7] M. Botts, Sensor Modeling Language (SensorML) Status. 2006. Retrieved from <http://stromboli.nsstc.uah.edu/SesorML/status.html>
- [8] D. F. McMullen and T. Reichherzer, *The Common Instrument Middleware Architecture (CIMA) Instrument Ontology & Applications*. Proceedings, Second Workshop on Formal Ontologies Meet Industry. Trento, Italy, December 15, 2006.
- [9] L. Bullivant, *Media House Project: the House is the Computer, the Structure is the Network*. Special Issue: 4dspace: Interactive Architecture, Architectural Design 75 (1), 51-53.
- [10] L. Cantarella and V. Guallart (Eds.), *Media House Project: The House is the Computer, the Structure the Network*. Actar (March 1, 2005).
- [11] T. Devadithya, K. Chiu, K. Huffman and D. F. McMullen, *The Common Instrument Middleware Architecture: Overview of Goals and Implementation*. Proceedings of the First IEEE International Conference on e-Science and Grid Computing (e-Science 2005), Melbourne, Australia, Dec. 5-8, 2005.
- [12] D. F. McMullen, I. M. Atkinson, K. Chiu, P. Turner, K. Huffman, R. Quilici and M. Wyatt, *Toward Standards for Integration of Instruments into Grid Computing Environments*. Proceedings of IEEE International Conference on e-Science and Grid Computing (e-Science 2006). December 2006. Amsterdam, The Netherlands.
- [13] I. M. Atkinson, D. du Boulay, C. Chee, K. Chiu, T. King, D. F. McMullen, R. Quilici, N. G. D. Sim, P. Turner and M. Wyatt. *CIMA Based Remote Instrument and Data Access: An Extension into the Australian e-Science Environment*. Proceedings of IEEE International Conference on e-Science and Grid Computing (e-Science 2006). December 2006. Amsterdam, The Netherlands.
- [14] R. Fielding, *Architectural Styles and the Design of Network-based Software Architectures*. Ph.D. Dissertation, University of California, Irvine. 2000.
- [15] R. T. Fielding and R. N. Taylor, *Principled Design of the Modern Web Architecture*. ACM Transactions on Internet Technology (TOIT) (New York: Association for Computing Machinery) 2 (2), 115-150 (2005), <http://www.ics.uci.edu/~taylor/documents/2002-RESTTOIT.pdf>
- [16] L. Siegele, The Beast of Complexity, Special Section: The Age of The Cloud. The Economist, April 12, 2001.
- [17] E. Hand, *Head in the clouds*. Nature 449, 963.
- [18] J. Dean and S. Ghemawat, *MapReduce: Simplified Data Processing on Large Clusters*. Proceedings of OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004.
- [19] M. Foley, Microsoft Popfly: Yahoo Pipes for the rest of us. ZDNet, May 18th, 2007.
- [20] G. C. Fox, R. Guha, D. F. McMullen, A. F. Mustacoglu, M. E. Pierce, A. E. Topcu and D. J. Wild, *Web 2.0 for Grids and e-Science*. Proceedings of INGRID 2007 – Instrumenting the Grid 2nd International Workshop on Distributed Cooperative Laboratories, S. Margherita Ligure Portofino, ITALY. April 18 2007.
- [21] I. Foster, C. Kesselman, J. Nick and S. Tuecke, *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*. Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002.
- [22] I. Foster, C. Kesselman, J. M. Nick and S. Tuecke, *Grid Services for Distributed System Integration*. IEEE Computer 35 (6), 37-46 (2002).
- [23] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell and J. Von Reich, *The Open Grid Services Architecture, Version 1.0*. Informational Document, Global Grid Forum (GGF), January 29, 2005.
- [24] M. P. Atkinson, D. De Roue, A. N. Dunlop, G. Fox, P. Henderson, A. J. G. Hey, N. W. Paton, S. Newhouse, S. Parastatidis, A. E. Trefethen, P. Watson and J. Webber, *Web Service Grids: an evolutionary approach*. Concurrency – Practice and Experience 17 (2-4), 377-389 (2005).
- [25] [25] S. Pallickara, G. Fox and S. Lee Pallickara, *An Analysis of Reliable Delivery Specifications for Web Services*. ITCC (1) 360-365 (2005).
- [26] K. Birman, *Can Web Services Scale Up?* IEEE Computer. 38 (10), 107-110 (2005).
- [27] M. E. Pierce, G. Fox, H. Yuan and Y. Deng, *Cyberinfrastructure and Web 2.0*. In: *High Performance Computing and Grids in Action* (L. Grandinetti Editor) published by IOS Press, Amsterdam, as the volume no 16 in the series Advances in Parallel Computing Proceedings of HPC2006 July 4 2006 Cetraro Italy.
- [28] M. E. Pierce, G. C. Fox, J. Y. Choi, Z. Guo, X. Gao and Y. Ma, *Using Web 2.0 for Scientific Applications and Scientific Communities* To appear in Concurrency and Computation: Practice and Experience Special Issue for 3rd International Conference on Semantics, Knowledge and Grid SKG2007 Xian China October 28-30 2007.
- [29] T. O'Reilly, *What is Web 2.0: Design Patterns and Business Models for the Next Generation of Software*. Available from <http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html>.
- [30] I. Atkinson, D. du Boulay, C. Chee, K. Chiu, P. Coddington, A. Gerson, T. King, D. McMullen, R. Quilici, P. Turner, A. Wendelborn, M. Wyatt and D. Zhang, *Developing CIMA based Remote Access for Collaborative e-Research*. Proceedings of the 5th Australasian Symposium on Grid Computing and e-Research (AusGrid 2007). Ballarat. January, 2007.
- [31] [31] D. F. McMullen and K. Huffman, (2005b) *Connecting Users to Instruments and Sensors: Portals as Multi-user*

GUIs for Instrument and Sensor Facilities. Proceedings of GCE 2005: Workshop on Grid Computing Portals held with SC05. Seattle, WA, November 18, 2005.

[32] See *RSS Specifications and RSS Feeds*. <http://www.rss-specifications.com/>.



DR. DONALD F. (RICK) McMULLEN is the Director of Research Computing and Senior Scientist at the University of Kansas. Previously he was Director and Principal Scientist of the Knowledge Acquisition and Projection Lab in the Pervasive Technology Laboratories at Indiana University. He received a Ph.D. in Chemistry in 1982 and served in a variety of research, engineering and management roles in the chemical and electronics industries before joining the Supercomputer Computations Research Institute as a Research Scientist in visualization and high performance computing. Dr. McMullen's primary research interests are in ubiquitous computing, knowledge-based support systems, international research and education networks, visualization, and virtual reality. Recent projects include software systems for remote access to instruments and sensors in support of e-Research collaborations, work with the U.S. Navy to develop knowledge-based remote maintenance and repair (tele-maintenance) systems, and development of distributed data acquisition with the Department of Energy Next Generation Internet (NGI) program.