Identical Principles, Higher Layers: Modeling Web Services as Protocol Stack

Mario **Jeckle** <mario@jeckle.de>
Erik **Wilde** <xmleurope2004@dret.net>

Abstract

Web Services and their potential applications are currently under heavy discussion in industry, research, and standardization. As a result of evaluation and experience by early adopters, the technology is expected to mature through the advent of new standards and solutions leveraging Web Service's power. In essence, the efforts undertaken to create and complete a stack of Web Service protocols lead to a new communication architecture and extends the stack of classical network protocols. This evolving architecture could serve as a future-proof infrastructure for businesses to rely on.

However the growth of the Web Service stack with respect to the addition of new layers and expansion of the resulting infrastructure has not been studied in comparison with well-established protocol suites like the ISO/OSI stack or the set of protocols constituting the Internet. Strictly speaking, industry's demand for functionality and services enhancing the basic Web Service protocols such as XML-RPC or SOAP, leads to the creation of a full-fledged layered protocol suite on-top of the existing ones. Nevertheless, the various standards, specifications, and ideas have neither been consolidated on a common terminological basis, nor been integrated in a single framework of reference.

This observation also applies to the established trio of Web Service standards composing of SOAP, WSDL, and UDDI. According to the specific usage patterns of these specifications, they are not operating on one layer as the well-known triangular relationship graph suggests, but instead they are connected by means of unidirectional usage dependencies. From this point of view, the message patterns (MP) defined by WSDL 2.0 offer services to layers organized on top of WSDL which rely on the service interfaces exposed by SOAP. More precisely, not the interface definition with WSDL but the accompanying MPs act as the transport layer of the service stack. Based on this and other criteria, SOAP can be categorized as the basic low-level layer of the Web Service infrastructure corresponding to the network-dependent layers of the classical protocol suites. Based on these facts, all of the various efforts relying on the seminal Web Service protocols can be categorized at the various levels layered above the transport layer. This is especially true for specifications dealing with the management of sessions and transactions which are layered directly above the MPs. Also, security standards like XML digital signatures and XML encryption fit well into this by classifying them as part of the presentation layer. Furthermore, within the Web Service environment quite analogous application layer mechanisms (e.g. firewalls for content filtering) emerge are commonly known for classical network operation.

Taking this congruency of established protocol stacks and the Web Service's one step further the analogy may serve as a valuable framework for the comparison of different architectural styles in Web Service deployment. Taking the continuing debate weighing services based on representational state transfer (REST) against those based on RPC-style SOAP as an example, both approaches reveal themselves as heterogeneous protocols. Both ideas are not mutually exclusive nor conflicting at all. Both protocols can be made interoperable by the use of bridges or gateways arbitrating between the two parties.

Our analysis shows that Web Services are a true but yet incomplete protocol suite deploying classical Internet protocols as basic services by the continued addition of supplemental specifications and standards.

Table of Contents

1. Introduction 2
2. Organization Principles 2
    2.1. Separation of Concerns 3
    2.2. Dependency Inversion Principle 3
    2.3. Layering 3
    2.4. Service Provisioning 4
3. The OSI Protocol Stack 4
    3.1. Layer 1 (Physical Layer) 4
    3.2. Layer 2 (Data Link Layer) 5
    3.3. Layer 3 (Network Layer) 5
    3.4. Layer 4 (Transport Layer) 5
    3.5. Layer 5 (Session Layer) 5
    3.6. Layer 6 (Presentation Layer) 6
    3.7. Summary 6
4. The Web Services Protocol Stack 6
    4.1. Layer 1 (Underlying Protocol Layer) 6
    4.2. Layer 2 (URI, XML, and SOAP Layer) 7
    4.3. Layer 3 (Addressing and Routing Layer) 7
    4.4. Layer 4 (Security and Message Pattern Layer) 8
    4.5. Layer 5 (Coordination Layer) 9
    4.6. Layer 6 (Vocabulary Layer) 9
5. Summary 9
6. Other Middleware 11
    6.1. CORBA 11
7. Future Work 11
8. Conclusions 12
Bibliography 12

1. Introduction

Architectures constituting of layered building blocks are prevalent in nowadays business application systems. To a large extend, systems are assembled from components which are custom created as well as ready-made common-of-the-shelf commodities. The ideas underlying this approach and even the benefits expected from its deployment seem new to the design of software architectures, but they have been around for some years now. In essence, a few recurring basic principles guide the creation of layered architectures. Historically, the most clear realization of these principles can be found in network protocol architectures like ISO's OSI reference model or even the stack of Internet protocols. Unfortunately, the term *Service-Oriented Architecture* (SOA) currently coined as an abstraction of classical Web Services so far has not been discussed with respect to its dependence on a well established protocol stack. Furthermore, no approach to put the elements of the ever-growing collection of Web Service related standards in a structure oriented on classical network stacks has been proposed.

The reminder of this paper is structured as follows. First we introduce some basic organization principles present in various layer architectures, especially protocol stacks. Afterwards we describe the ISO/OSI protocol stack and its various layers. We focus on the semantics and specific function of each layer. Thus neither we concentrate on respective protocols proposed by ISO to implement the stack architecture, nor do we cover existing protocols deployed at the Internet to an extent exceeding their mention as example. Finally, we introduce a layered architecture for the protocols underlying Service-oriented architectures. This architecture will be organized similarly to ISO's OSI model.

2. Organization Principles

This section introduces a collection of basic organization principles which can be found in various architectures describing communication protocols or software systems. Some of the principles seem to be overlapping, which is true, since they often cover the same abstract aspect form another point of view.

2.1. Separation of Concerns

Parnas' seminal paper [SoC] introduced *separations of concerns* as the principle of breaking a larger programs into distinct features. These features (termed *concerns*) should ideally provide disjunct functionality in order to minimize overlap among them. Furthermore, each respective feature should expose an explicit (and thus well-defined) interface to other features which hides the internal realization of the functionality offered to other features.

By doing so, each features becomes decoupled from other features using it or used by the respective feature. This leverages the possible reuse of existing features independent from other features connected to it using the well-defined interfaces. Also the separation approach allows to substitute a given component by another which provides the same external functionality (i.e. exposes the same interface) but provides a different internal realization of the functionality. This could be helpful if a given feature does not fulfill changed requirements and needs to be replaced by one which provides a different (e.g. faster or more reliable) implementation offered through the unchanged interface.

Separation of Concerns is a widely adopted paradigm. Basically, the idea can be deployed to arbitrary structures to be defined regardless of the programming language which is actually chosen for implementation. Object-oriented design and its successors and extensions (e.g. aspect-oriented programming) inherently support the decoupling of features (termed *objects* there) by offering explicit interface constructs.

2.2. Dependency Inversion Principle

The *Dependency Inversion Principle* (DIP) popularized by [DIP] states that no higher level module (which is the term the author of [DIP] uses synonymous to Parnas' *feature*) should depend on a low level module. But both should depend upon a common abstraction. Furthermore, these abstractions shared among the modules should not depend upon details specific to the realization of a module.

Compared to the naive application of *Separation of Concerns* in the classical top-down manner, DIP inverses this view. Essentially, the designer is encouraged to create the modules fulfilling the most basic tasks within the architecture first. Afterwards, all parts of the module other modules could depend on are factored out into an explicit dependency module. This module is used by all further modules depending on the originating module and also this module itself.

By depending all modules from the same abstraction creates a less rigid system structure since, regardless if they provide or use a given functionality. As a result the modules dealing with the implementation of functionality are decoupled. This in turn increases system' stability since changes to internal implementations are not exposed to the general public as long as the published interface (i.e., the abstracted module the using module relies on) remains unchanged. Furthermore, possibly fragile changes to the system become more obvious since they will affect the module implementing the functionality and also the abstraction exposing it.

The Dependency Inversion Principle is commonly used in programming languages by providing signatures of functions separated from the code providing the conforming implementation. Classical examples for this include Oberon's Modules and also C's header files. Modern object-oriented languages such as Java offer an explicit construct termed *interface* which essentially forms an implementation of Parnas' abstract features combined with an operationality strictly enforcing conformance to declared interfaces.

2.3. Layering

Based on the operationalization of Parnas' abstraction approach formulated by DIP new architecture patterns emerged. These patterns tend to cluster related functionality horizontally. This in turn separates it from depending functionality implemented by a different block of the architecture. Normally the functionality using other functionality is stacked on-top of the architecture block providing the used functionality. Given the layering pattern, the architecture blocks clustering related functionality are termed *layers*.

Layering in essence combines both approaches discussed before and adds a guideline for organizing the interdependent modules. Ideally, higher layers of an architecture do only depend on functionality exposed by layers organized below it. Taking this further, a higher layer of an architecture should only depend on functionality exposed by layers directly below it. Hence functionality provided by layers not directly facing the layer which

requests this functionality has to be handed over (i.e., tunneled) by an intermediary layer. Adhering to this architectural pattern creates robust building blocks with clearly exposed interfaces connecting them. Furthermore, obeying to the strict principle which allows only direct neighbourts to communicate ensures that every single layer can be exchanged by another implementation. These surrogate implementation has solely to satisfy the interfaces requested by the layers stacked on-top of the replaced layer.

Implementations of layered architectures can be found in various commercial systems and frameworks. Another important field of application are network protocols like ISO's OSI reference model [ISO/OSI] or the stack of Internet protocols [TCP]. In many cases, layering not only is useful for structuring software systems and implementations, but also is reflected in the hardware world, with specialized hardware implementing the functionality of certain layers. For example, well-known networking components (sometimes implemented in hardware, sometimes implemented in software) are *Repeaters*, *Bridges*, *Routers*, and *Gateways*.

2.4. Service Provisioning

Up to now, only pure syntactical aspects of exposed interfaces have been discussed. Often aspects related to the dynamic execution of a given functionality has also to be taken into account. Hence the exposed description of the exposed functionality is supplemented by descriptions of various types. Typically, these additional descriptions include data describing semantic or process-related aspects. As a consequence of this, the modules exposed can be regarded as *services* which provide a given functionality adhering to given semantics and fulfilling a dynamic contract.

Extending the pure syntactic view to modules opens a new dimension of looking at layering, dependencies, and separation of concerns. By taking other descriptive aspects of a feature into account, the importance of the pure syntactic view diminishes for the benefit of the other aspects mentioned. Thus building blocks of an architecture (and even the architecture itself) can be compared to other blocks (or architectures) by comparing a set of aspects instead of limiting the view to the pure syntax.

3. The OSI Protocol Stack

In the networking world, layered protocol stacks are the globally accepted way to structure systems. As described before, layering is an effective way to reduce the complexity of networking, resulting in manageable parts. Since the OSI layers 5 and 6 have no counterparts in the Internet protocol world, there are no established metaphors for networking nodes working on these layers. However, there have been developments in these areas, even though they have not been widespread enough to become a generally accepted networking node. An example for layer 5 support are Web application development frameworks providing support for session management. These frameworks provide programmers with the notion of sessions, and programmers are free to build their applications as if client/server-relationships in the Web were stateful. The frameworks then can be configured to implement the session support by using either Cookies or URI rewriting, and they can make this decision even dynamically, depending on whether a particular client provides support for Cookies or not.

3.1. Layer 1 (Physical Layer)

Basically, the main purpose of the physical layer is it to transport a stream of bits from a sender to a receiver. Along the path connecting the sender with the ultimate receiver the raw stream may be transmitted over various physical media. In order to enable bits to travel long distances probably crossing multiple networks, active intermediate components such as repeaters may be deployed. Repeaters, amplify electrical (or, for fiber-based networks, optical) signals, for example for the purpose of using longer network cables than normally possible (because of a cable's inherent attenuation). Repeaters do not attempt to actively interpret a signal in any way, their only purpose is to read it, and then to amplify (or maybe regenerate) the exact same signal for further distribution. Another popular name for a repeater is *hub*.

It should be noted that an implementation of the physical layer can be provided by various transmission media. Most of the physical layers deployed today are provided by wired connections provided by cable-based connections ranging from twisted pair to fiber optics. In the recent years wireless connections gained incresed interest. Thus the physical layer may also be implemented by an air-based connection such as Wireless LAN (WiFi or WLAN) or a mobile telephone system.

3.2. Layer 2 (Data Link Layer)

The main task of the data link layer is it to lay the grounds for safe transmission. This includes dealing with various kinds of transmission errors (e.g., capped transmission lines). Also the data link layer deals with regulating the amount of data transmitted between neighboring networks order to prevent shipping of data too fast thus it cannot handled by the receiver. Additionally, the data link layer provides an service interface to the network layer residing on top of it.

Typical components layered on the network level are bridges. A bridge is smart enough to selectively decide which packets to forward to a particular network segment (often analogous to some interface of the bridge). A good example for a bridge is a WiFi access point, forwarding packets from a LAN cable to the wireless interface and vice versa. The bridge will only forward those packets it receives on its wireless interface that should be forwarded through its cable interface and vice versa. Another popular name for a bridge is *switch*.

Taking the principle of layering further, the data link layer is normally subdivided into the *Logical Link Control* whose duties have been described before, and the *medium access control sub-layer* (MAC). As the bottom part of the layer 2 the latter deals with issues how to physically access the communication channel underlying the data link layer. In case of the Internet, each node able to send and receive data is uniquely identified by a so called MAC address.

3.3. Layer 3 (Network Layer)

While the data link layer solely deals with transmitting data between two adjacentnodes within a network path, the network layer concentrates on the logical connection established between the ultimate endpoints. The main characteristic of the layer is it to retain a specific level of quality of service to the transport layer. Essentially, quality of service means to reduce network congestion and thus improve transfer performance.

Technical components residing on the network layer include routers adapted for the specific needs of the network layer. In order to forward an IP datagram to the right interface which is identified by its respective MAC address (and finally to the right end system), routers need to receive information via routing protocols, which enable them to make the decisions that are required to make global networking possible.

3.4. Layer 4 (Transport Layer)

Especially, the transport layer is crucial for the whole protocol hierarchy. It provides the core transport facilities optimized for cost-efficiency and reliability from sources to destinations regardless of the underlying network layer and physical details constituting the communication channel between processes. In detail the layer provides services such as synchronous and asynchronous data transfer to the protocol layers stacked above the transport layer.

Technically speaking, anything that operates above layer 3, but still is considered to be part of the networking infrastructure, is termed a *gateway*. Gateways can have very different tasks, they can transfer data between different networking architectures, or they can implement application-specific functionality that requires knowledge of particular application-level protocols. A typical example for a gateway is a Wireless Application Protocol (WAP) gateway, that connects the Internet and a mobile phone network (such as a GSM network) and enables mobile phone users to retrieve Web pages with their mobile devices.

3.5. Layer 5 (Session Layer)

The session layer provides the control structure for managing end-to-end communications. Possible examples are establishing, managing, and terminating sessions. The rationale for the session layer is that many applications require communications beyond the simple and stateless transfer of data packets.

In the Internet world, session management is managed by applications. In the case of the popular Hypertext Transfer Protocol (HTTP), the underlying Internet transport protocol does not provide session management. Consequently, session management must be layered on top of the application protocol (popular techniques for HTTP session management are Cookies and URI rewriting).

3.6. Layer 6 (Presentation Layer)

The transport layer provides platform-independent ways of encoding data. It includes mechanisms for communicating peers to negotiate a transfer encoding, and support to produce and consume this transfer encoding. The most important standard on this layer is the *Abstract Syntax Notation One* (ASN.1), which is still in use today in various application areas (X.509 certificates are a popular example).

In the Internet world, XML has taken on the role of a globally accepted transfer syntax. Even though there is no negotiation mechanism (which is not necessary, because there is only one syntax for XML) and no established view of the underlying information model [Wilde-IC], XML certainly has filled a gap that has hampered many interesting developments in global networking of applications.

3.7. Summary

ISO's OSI reference architecture for communication protocols adheres to the basic principles outlined by the previous section. In detail every layer discussed before abstracts the lower level layer covered by it and provides well-defined interfaces to the layer located on top of it.

At the syntax level, all messages of a lower protocol level are encapsulated by the higher level data. Given the use of Internet protocols each layer the data to transmit traverses adds a protocol specific header to the raw data. Figure 1 shows this by color coding a data packet transmitted via HTTP (Session Layer), TCP (Transport Layer), IP (Network Layer), and Ethernet (Link Layer).

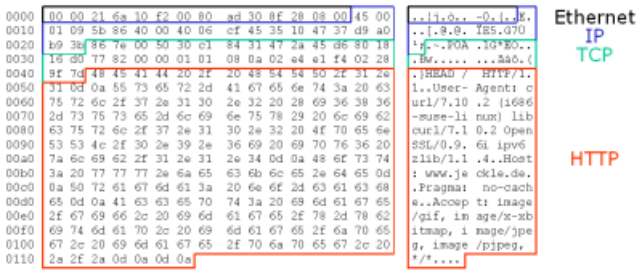

Figure 1.

4. The Web Services Protocol Stack

This section contrasts some elements of the Web Services protocol family to layered architecture as set out by OSI. Normally, two assumptions underly the view of Web Services as a protocol stack. First, the stack of Web Service protocols typically is described to consist of SOAP [SOAP-Part1] [SOAP-Part2] (for transporting messages) layered on the bottom, succeeded by WSDL [WSDL-Part1] [WSDL-Part2] (for describing messages), and UDDI [UDDI] (for storing message descriptions) at the very top. All other protocols are somewhat related to this triumvirate but not put in a definite context. Second, intuitively SOAP is stacked above its underlying transport layer which is in most practical cases HTTP.

Within this section we show that the protocol suite constituting the Web Service stack can be modeled similarly to the OSI reference model discussed in Section 3. By doing so, we are able to provide a sound and well-established framework to putting existing and even upcoming Web Service protocols in context. Corollary, this description approach provides evidence that Web Services tend to establish a complete protocol stack which re-factors some of the classical infrastructure components by putative new specifications.

4.1. Layer 1 (Underlying Protocol Layer)

In the OSI model, layer 1 is the plumbing, so to speak, connecting communication peers. Layer 1 technology is used to make the exchange of data physically possible by agreeing on wiring (serial or parallel, optical or electrical), plugs (shape and arrangement of the pins/holes), and basic transmission parameters (voltage for electrical signaling, wavelength for optical signaling). Accordingly, we can look at the layer 1 for Web Services by providing the plumbing that is used to connect the communication peers.

The plumbing for Web Services can be almost anything, since Web Services can be based on different transport mechanisms. This includes protocols layered on the fifth or sixth ISO layer such as HTTP which is the most widely used plumbing. However, the main reason for this is that HTTP is the most widely used protocol of the Internet, supported on virtually all platforms, and in many cases is the only protocol that, due to its clear-text representation, is allowed through firewalls and other filtering mechanisms. Other sorts of plumbing the Web Services protocol stack are possible and are being used, such as the Simple Mail Transfer Protocol (SMTP). Furthermore, also protocols layered at ISO's transport layer can be deployed for transporting Web Service content. Typical examples in the Internet world include the TCP/IP protocol pair or even the Universal Datagram Protocol (UDP) for small amounts of data where reliability is not required.

In the same way as the independence of the networking software from the actual hardware made networking flexible and powerful (regardless whether a computer is using 10Mbit or 100Mbit Ethernet or one of wireless variants, as long as the appropriate drivers for the networking card are present), the layer 1 for Web Services makes programming with Web Services flexible and powerful, because applications based on Web Services can be moved from HTTP to other underlying protocols fairly easily, without the need to change anything in the upper layers of the Web Services protocol stack.

4.2. Layer 2 (URI, XML, and SOAP Layer)

Layer 2 is responsible for reliably transmitting data between two communication peers that are connected through a layer 1 infrastructure. As described above, the layer is divided into two sub-layers, the lower sub-layer being the Media Access Control (MAC), and the higher sub-layer being the Logical Link Control (LLC).

The MAC sub-layer is primarily responsible for defining an addressing scheme that can be used to address communication peers. Many security mechanisms in today's networks are based on network-level MAC addresses, where firewalls block devices from gaining access to a network based on their MAC address. In the Web Services protocol stack, the equivalent of the MAC address is the Uniform Resource Identifier (URI) [RFC2396]. It is one of the Web's basic principles that resources are addressable through URIs, and since a Web Service simply is a resource, it is addressable through a URI. It should be noted that URIs contain a so called *scheme* part, which identifies the access method through which a certain resource can be accessed. In networking, the MAC address depends on the medium that connects the communication peers (Ethernet MAC addresses are specific to the Ethernet media), and in the same way the *Web Service's MAC addresses* depend on the underlying transport medium (for example HTTP, which would be reflected in a URI with a `http` scheme part). With respect to security provided by allowing only communication with a set of predefined MAC addresses, Web Service specific content filter can be provided by interpreting the URI which serves as identification of a message's ultimate receiver a.k.a. the service provider.

In Web Service terms, the LLC sub-layer of layer 2 uses the SOAP protocol for assembling messages to be exchanged between communication peers. SOAP's data model changed dramatically from SOAP 1.1 to SOAP 1.2. While SOAP 1.1 was based on XML 1.0 [XML] as the data model (the receiver was supposed to get the exact same XML 1.0 document that was sent by the sender), SOAP 1.2 switched to the Information Set [Infoset], which is an abstraction of XML. With the Information Set being the new foundation of SOAP, encoding and decoding messages can be done in a more lenient way, because the Information Set ignores some of the syntactic idiosyncrasies of XML. In particular, with the Message Transmission Optimization Mechanism (MTOM) [MTOM] and the XML-binary Optimized Packaging (XOP) [XOP], some additional standards exist which can be used to achieve interoperability on the encoding level of the exchanged Information Set data. MTOM and XOP are also designed to support the efficient interchange of Infosets containing binary data, by fragmenting the encoded data into an XML 1.0 document and additional binary parts, all combined in a multipart MIME message.

4.3. Layer 3 (Addressing and Routing Layer)

Based on the physical connection established by the protocols situated on layer 2 the equivalent to OSI's network layer on the Web Services side also provides logical connections. Two of the main functions provided by the network layer are routing and addressing. Within the Internet protocol stack these are implemented on the basis of the *Internet Protocol* (IP).

Addressing in a way similar to the Internet Protocol is provided by the *WS-Addressing* [WS-Addressing] protocol. Essentially, the so-called protocol is just a set of XML elements with a prescribed semantics. Like the sender's and

receiver's MAC addresses transported over the data link layer and the provided mapping of the MAC address into the IP address, the WS-Addressing protocol operates on top of the SOAP protocol. In detail, WS-Addressing defines elements to store the URIs of both peers, the message's sender and its intended receiver. Additionally, an element identifies the service to route the call to. By doing so WS-Addressing completely re-factors the expressive power of the data directly belonging to the IP layer. As Figure 1 indicates IP stores the sender's IP address within the last-most eight to five bytes (the figure shows the hexademical representation of the sender's IP 53.16.71.55). The hexadecimal address of the receiver is stored in the last four bytes. Note: Since the URI does not identify the physical service but a logical port WS-Addressing is layer at layer 3. In order to provide fail-over handling or load balancing behind a single URI may transparently reside more than one server providing the same service.

Historically, IP also contains a field *Type of Service* which was intended to distinguish between different kinds of services. The example depicted by Figure 1 shows the value `00` (directly succeeding the initial `45` identifying the protocol itself) for the service's type. Today, typically the service 's type is identified by the well-known port the message is directed to. The counterpart on the Web Service side of the basic-idea is the element named `Action` of the WS-Addressing protocol. It identifies the service to invoke by a unique identifier encoded using URI syntax. Figure 2 shows the addressing information, which is encoded by the XML elements `ReplyTo` and `To`. Additionally, the service to invoke is identified by the content of the `Action` element.

What's more, the family of Web Service protocols also re-factors data stored in inside hardware components of network infrastructure such as routers into explicit protocols. *WS-Routing* [WS-Routing] may serve as valuable example for this. It allows the explicit specification of the routing path the message has to travel from the sending client to the ultimate receiver. Interspersed within the path intermediaries which may process and also apply changes to the message may occur. Essentially, WS-Routing forms a stateless-protocol for describing arbitrary routing paths from the message issuing client via a possibly empty set of intermediaries to the ultimate receiver a single message has to obey to. Figure 2 shows the application of the WS-Routing protocol to specify the path of the sample message. The specified path includes processing by an authenticating intermediary first and afterwards by a node offering compression capability.

Furthermore, the network layer should provide a specific quality of service which is required for executing specific tasks over the network. Within the stack of Web Service protocols *WS-Policy* [WS-Policy] addresses this requirement by setting out a framework for expressing arbitrary policies. Examples of such policies include *WS-SecurityPolicy* [WS-SecurityPolicy] (which is also used by the example depicted by Figure 2) for defining requirements a client has to comply to in order to enable it to initially contact to the service in a secured manner.

4.4. Layer 4 (Security and Message Pattern Layer)

Within the Web Service's protocol stack, the transport layer represents the most crucial part of the whole protocol stack. The most notable distinction between the protocols deployed within the classical networking environment and the Web Service's stack is the way message patterns are supported. Typically, the protocols of the ISO/OSI stack and also the Internet protocols bind together transport protocols such as TCP, UDP, or RTP and the pattern messages are sent and received. For example, the *Transport Control Protocol* (TCP) (originally defined by [TCP]) implies a strict request-response-pattern in which every conversation consists of exactly two messages in order in which every incoming message of a node has to be followed by an outgoing message. Other message patterns such as notification style communication requiring the possibility to send a message without assuming a reliable connection, i.e. without requiring the receiver to confirm the message's receipt. Supporting other message patterns implies a change in the communication protocol deployed to transport the actual data. Hence altering TCP's request-response nature to a one-way notification pattern forces the use of the *User Datagram Protocol* (UDP) [UDP] which inherently offers this exchange pattern. Unfortunately, changes to the communication protocol introduces a number of drawbacks. First, changing TCP to UDP requires some changes in code to the communication applications. Second, and more profound, UDP is not solely a notification-only version of TCP but an independent protocol. The fact that UDP solely supports the transfer of small scaled data (since it does not provide flow control, loss detection, duplicate detection, re-sequencing or other mechanisms for dealing with errors) introduces a crippling drawback as a by-product of the intended change in communication style.

Within the Web Service protocol stack the actual style of a message-based interaction is not predetermined by the communication protocol, e.g. SOAP. Instead of that, [WSDL-Part2] sets out a collection of message patterns which define the sequence and cardinality of abstract messages. These patterns can be applied to every communication protocol, e.g. SOAP.

Additionally, within the protocol suite deployed within the Internet (especially the Web) the transport layer is widely used to provide basic security. For doing so, typically the *Secure Socket Layer* (SSL) or its IETF standardized version the *Transport Layer Security* (TLS) protocol [TLS]. These protocols provide communications privacy over the Internet by allowing partners to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery. Technically, TLS is layered on top of some reliable transport protocol (e.g., TCP) and thus unsuitable for connectionless transport mechanisms such as UDP.

To circumvent the obstacles discussed before which arise from depending the transport mechanism and communication the style or other communication characteristics such as security, the Web Service stack clearly separates security issues from the underlying protocol. Therefore SOAP-encoded messages can be secured solely on the XML layer by signing and/or encrypting parts of a message or the whole message at all [WS-Security].

This also provides clear evidence that techniques for providing security which are based the Internet protocol stack solely are insufficient when transferred without adaption into the Web Service's protocol stack. This is especially true since neither SSL nor TLS takes into account Web Service protocol's specifics like active intermediaries which need to interpret parts of the message. Hence methods for providing end-to-end security cannot be applied without braking basic protocol principles.

4.5. Layer 5 (Coordination Layer)

Session management essentially forms lightweight version of providing transaction control by obliterating the principles of atomicity and isolation in order to not limiting the degree of possible concurrency. Due to the diverse requirements governing session management no single accepted standard for providing logical correlation of messages sent separated currently exists. Applications operating on-top of the Internet protocol suite typically provide coordination and session management in a proprietary and thus non-interoperable manner.

For Web Services currently no single approach has become widely accepted up to now. Potential candidates include the *Business Process Execution Language for Web Services* [BPEL4WS] and also W3C's *Web Service Choreography* [WS-Choreography] initiative.

4.6. Layer 6 (Vocabulary Layer)

Layer 6 is responsible for mapping the application's data model into a form than can be transmitted between communicating peers. In the Web Services world, this is accomplished by using XML. However, applications need to agree upon a certain schema of XML documents which shall be exchanged, and this is accomplished by XML Schema [XSD1] [XSD2]. Thus, speaking in OSI terms, XML Schema defines the abstract syntax of the XML , while XML are the encoding rules that are being used for encoding data instances. The XML Schema definition is part of the WSDL definition, which (as one part of the WSDL content) contains a part specifying the types being used for the Web Service. Conceptually, WSDL is not limited to XML Schema and can be used to support other schema languages as well, but since every WSDL implementation is required to support XML Schema, it is the most important schema language for Web Services.

However, even though for practical purposes Web Services will always be based on XML Schema types, it may make sense to augment a Web Service definition with more schema information than only XML Schema. For example, the Schematron schema language is a useful and compact schema language, which very often can be used to specify additional rule-based constraints which can not be expressed in XML Schema. More generally, the question of a schema for a Web Service (which basically is a set of constraints separating acceptable from non-acceptable XML payload) could be handled in a modular way, an approach which is taken by the Document Schema Definition Languages (DSDL) initiative of ISO. A DSDL-style approach to Web Services would include complete validation (against possibly multiple schemas) as part of the Web Service infrastructure, and only if the XML payload passed all schema validations would it be passed on to the application.

5. Summary

The hierarchical architecture of Web Service protocols adheres to the basic principles outlined by section 1. In detail every layer discussed before extends the lower level layer covered by it and is integrated on the basis of XML's syntactic principles to the directly succeeding layer.

At the syntax level all messages are expressed using XML's Unicode-based textual representation. Figure 2 shows the protocols discussed within section 2 of this paper applied within a single Web Service massage encoded using the SOAP protocol. The actual data packet is transported using the HTTP protocol serving as the underlying protocol.

Figure 2.

6. Other Middleware

While the focus of this paper is on Web Services, it is interesting to look at other middleware architectures, and try to identify similar concepts.

6.1. CORBA

CORBA takes a much more monolithic approach than Web Services. The only truly open interfaces in CORBA are the interface to the application itself, and the wire format. The wire format depends on the transport protocol, for the most popular IIOP transport the wire format is the Common Data Representation (CDR). Apart from the CDR data being exchanged, CORBA implementations are pretty much black boxes for the application programmer, making it virtually impossible to assemble a CORBA platform from various components in a modular way.

However, since this black box approach of CORBA was identified as a problem in some cases, CORBA introduced the concept of Interceptors, which are isolated interfaces within the *black box* which may be used to access a CORBA implementation at a very limited number of places. The Interceptors do not introduce any concept of structuring or layering in CORBA, they are simply a retrofitted way to loosen up the black box approach. Generally speaking, it would probably be impossible to re-model CORBA using a structured modeling approach, but the Interceptors are evidence that the black box approach may not be the best way to go for a component as big as CORBA.

7. Future Work

The goal of this paper is to provide a starting point for looking at the evolving Web Services architecture in a more structured way. The Web Services landscape is changing rapidly and constantly, and the field would benefit from a more structured way of modeling. In this paper, we argue that the layering approach well-known from networking protocol stacks is also a useful approach for structuring the Web Services architecture. Some of the concepts of networks layering (such as the the underlying infrastructure and the presentation layer) can be applied pretty easily to the Web Services architecture, while for other aspects the layering may be less obvious.

This paper proposes a complete mapping from the Web Service architecture to the OSI layering system of the Internet. However, as discussed earlier, there is a fundamental difference between the layered encoding of data within network systems (see Section Section 3.7) and the SOAP approach of encoding layer-specific information in the header of a SOAP message. While in strict layering, each layer adds a new envelope to the data received from the layer above, thereby creating a multi-layer envelope structure around the data, the SOAP approach uses XML's Namespace mechanism to add new information to the SOAP message. Thus, the SOAP approach can be seen as being comparable with the general idea of Aspect-Oriented Programming [AOP]. AOP claims that closed components in software systems are not a good idea, because there may be aspects cross-cutting through several components. In the same way as AOP enables code to be written in a way that transcends usual component boundaries, SOAP enables components on top of SOAP to access component-specific information independent from the traditional strict layering enforced by opaque data chunks with layer-specific headers. The question of AOP vs. Layering and possible problems with one of the approaches [Jung] is an interesting one and is a profound question of the most appropriate way to model complex systems.

We do not want to claim that the layering approach presented in this paper is the only possible and true way to structure the Web Services world. It is rather the higher level question of how a framework for the Web Services architecture should look like, and whether by choosing one we could perhaps learn from earlier experience that lead us to the layering approach presented here. AOP and its remarkable similarity to the way SOAP carries additional information in its header is something that we are interested in investigating in more detail. It is possible that the strict layering we describe in the Web Services layers 3-5 could and should be replaced by something more flexible, so that the *layering structure* of communicating Web Service peers must not match structurally, but only semantically (in the spectrum allowed by constraints such as SOAP's `mustUnderstand` attribute). It is possible that alternative models of protocol structuring, such as the protocol graphs used in the context of Dynamic Protocol Configuration [Plagemann], provide a more appropriate way of protocol modeling for the Web Services layers 3-5.

8. Conclusions

As this paper has demonstrated the a priori unordered soup of Web Services acronyms and protocols can be considered as hierarchy of related protocols such as ISO's OSI stack or the hierarchy of Internet protocols.

Figure 2 shows the major difference between the syntactic representation of the logical layering. As Figure 1 shows, within the stack of classical network protocols the binary wire representation of the content is organized on a specific layer is extended by padding the additional data in front of the data resulting from the protocol layered directly below the respective protocol. Due to XML's syntactic nature and the architecture SOAP's extensibility mechanism protocol data resulting from layers organized on-top of layer 2 is physically stored within the SOAP message. Summarizing this, *extending SOAP* which is synonymous to *layering on-top of* means adding new custom headers to the SOAP protocol. Corrolarily, if a header of a given type depends on the presence of another one, the protocol providing it could be regarded as being layered on top of it.

Table 1 illustrates the equivalence of the protocols constituing the Web Service with those being part of the Internet protocol suite.

	Internet	Web Services
Presentation Layer (Layer 6)	various (application dependent)	XML Schema
Session Layer (Layer 5)		various (e.g., BPEL4WS, WS-Choreography)
Transport Layer (Layer 4)	TCP or UDP	WSDL MEP
Network Layer (Layer 3)	IP	WS-Addressing, WS-Routing, ...
Data Link Layer (Layer 2)	Ethernet	SOAP, URI, XML Infoset
Physical Layer (Layer 1)	various Ethernet media	HTTP, TCP, BEEP, ...

Table 1. Equivalence of protocol stacks

Bibliography

[AOP] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, and John Irwin. *Aspect-Oriented Programming*. In Mehmet Aksit and Satoshi Matsuoka, editors. *Proceedings of the 11th European Conference on Object-Oriented Programming*, volume 1241 of Lecture Notes in Computer Science, Jyväskylä, Finland, June 1997. Springer-Verlag, pages 220-242. http://www.cs.ubc.ca/%7Egregor/kiczales-ECOOP1997-AOP.pdf

[BPEL4WS] Tony Andrews, Francisco Curbera, Hitesh Dholakia et al. *Business Process Execution Language for Web Services Version 1.1*. Microsoft, IBM, Siebel Systems et al., May, 2003. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/

[DIP] Robert C. Martin. *The Dependency Inversion Principle*. The C++ Report, Vol. 8, No 6, May, pp. 61-66, 1996.

[ISO/OSI] Hubert Zimmermann. *OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection*. IEEE Transactions on Communication, 28(4):425-432, April, 1980. http://www.comsoc.org/livepubs/50_journals/pdf/RightsManagement_eid=136833.pdf

[Infoset] John Cowan and Richard Tobin. *XML Information Set (Second Edition)*. World Wide Web Consortium, Recommendation REC-xml-infoset-20040204, February 2004. http://www.w3.org/TR/2004/REC-xml-infoset-20040204

[Jung] Matthias Jung and Ernst W. Biersack. *How Layering Protocol Software Violates Separation of Concerns*. In *ECOOP 2000 Workshop on Aspects and Dimensions of Concerns*, Cannes, France, June 2000. http://trese.cs.utwente.nl/Workshops/adc2000/papers/Jung.pdf

[MTOM] Noah Mendelsohn, Mark Nottingham, and Hervé Ruellan. *SOAP Message Transmission Optimization Mechanism*. World Wide Web Consortium, Working Draft WDsoap12-mtom-20040209, February 2004. http://www.w3.org/TR/2004/WD-soap12-mtom-20040209/

[Plagemann] Thomas Plagemann. *A Framework for Dynamic Protocol Configuration*. PhD thesis, Swiss Federal Institute of Technology, Zürich, Switzerland, 1994. Diss. ETH No. 10830.

[RFC2396] Tim Berners-Lee, Roy T. Fielding, and Larry Masinter. *Uniform Resource Identifiers (URI): Generic Syntax*. Internet draft standard RFC 2396, August 1998. http://www.ietf.org/rfc/rfc2396.txt

[SOAP-Part1] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen. *SOAP Version 1.2 Part 1: Messaging Framework*. W3C Recommendation, June, 2003. http://www.w3.org/TR/2003/REC-soap12-part1-20030624/

[SOAP-Part2] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen. *SOAP Version 1.2 Part 2: Adjuncts*. W3C Recommendation, June, 2003. http://www.w3.org/TR/2003/REC-soap12-part2-20030624/

[SoC] David L. Parnas. *On the Criteria To Be Used in Decomposing Systems into Modules*. Communications of the ACM, Vol. 15, No. 12, December, pp. 1053-1058, 1972. http://www.acm.org/classics/may96/

[TCP] *Transmission Control Protocol*. DARPA Internet Program Protocol Specification, September, 1981. http://www.ietf.org/rfc/rfc793.txt

[TLS] T. Dierks, C. Allen. *The TLS Protocol Version 1.0*. Internet Engineering Task Force RFC 2246, January, 1999. http://www.ietf.org/rfc/rfc2246.txt

[UDDI] Oasis UDDI TC (ed.). *UDDI Version 3 Specifications*. Oasis Open, 2003. http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv3

[UDP] John Postel. *User Datagram Protocol*. Internet Draft Standard RFC 768, August, 1980. http://www.ietf.org/rfc/rfc768.txt

[WS-Addressing] Adam Bosworth, Don Box, Erik Christensen et al. *Web Services Addressing (WS-Addressing)*. Microsoft, IBM, BEA, 2003. http://msdn.microsoft.com/ws/2003/03/ws-addressing/

[WS-Choreography] Daniel Austin, Abbie Barbir, Ed Peters et al. (eds.). *Web Services Choreography Requirements*. W3C Working Draft, March, 2004. http://www.w3.org/TR/2004/WD-ws-chor-reqs-20040311/

[WSDL-Part1] Roberto Chinnici, Martin Gudgin, Jean-Jacques Moreau, Jeffrey Schlimmer, Sanjiva Weerawarana. *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*. W3C Working Draft, 2003. http://www.w3.org/TR/2003/WD-wsdl20-20031110

[WSDL-Part2] Martin Gudgin, Amy Lewis, Jeffrey Schlimmer. *Web Services Description Language (WSDL) Version 2.0 Part 2: Message Patterns*. W3C Working Draft, 2003. http://www.w3.org/TR/2003/WD-wsdl20-patterns-20031110

[WS-Policy] Don Box, Francisco Curbera, Maryann Hondo et al. *Web Services Policy Framework (WS-Policy)*. IBM, Microsoft, SAP, 2003. http://www-106.ibm.com/developerworks/library/ws-polfram/

[WS-Routing] Henrik Frystyk Nielsen, Satish Thatte. *Web Services Routing Protocol (WS-Routing)*. Microsoft, 2003. http://msdn.microsoft.com/ws/2001/10/Routing/

[WS-SecurityPolicy] Giovanni Della-Libera, Phillip Hallam-Baker, Maryann Hond et al. *Web Services Security Policy (WS-SecurityPolicy)*. Microsoft, IBM, Verisign et al. 2002. http://www.ibm.com/developer-works/webservices/library/ws-secpol/index.html

[WS-Security] Anthony Nadalin, Chris Kaler, Phillip Hallam-Baker et al. (eds.). *Web Service Security: SOAP Message Security 1.0*. OASIS Open, February, 2004. http://www.oasis-open.org/committees/download.php/5531/oasis-200401-wss-soap-message-security-1.0.pdf

[Wilde-IC] Erik Wilde. *XML Technologies Dissected*. IEEE Internet Computing, 7(5):74–78, September 2003.

[XML] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and François Yergeau. *Extensible Markup Language (XML) 1.0 (Third Edition)*. World Wide Web Consortium, Recommendation REC-xml-20040204, February 2004. http://www.w3.org/TR/2004/REC-xml11-20040204/

[XOP] Noah Mendelsohn, Mark Nottingham, and Hervé Ruellan. *XML-binary Optimized Packaging*. World Wide Web Consortium, Working Draft WD-xop10-20040209, February 2004. http://www.w3.org/TR/2004/WD-xop10-20040209/

[XSD1] Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. *XML Schema Part 1: Structures*. World Wide Web Consortium, Recommendation RECxmlschema-1-20010502, May 2001. http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/

[XSD2] Paul V. Biron and Ashok Malhotra. *XML Schema Part 2: Datatypes*. World Wide Web Consortium, Recommendation REC-xmlschema-2-20010502, May 2001. http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/

Biography

Mario **Jeckle**
University of Applied Sciences Furtwangen (FHF)
Furtwangen
Germany
mario@jeckle.de

Mario is a professor for software engineering at FHF and member of the W3C's Technical Architecture Group.

Erik **Wilde**
Swiss Federal Institute of Technology (ETH) Zürich
Switzerland
xmleurope2004@dret.net

Erik is lecturer at ETH and the University of Applied Sciences Aargau (FHA).