Web Services: Potential and Future

© Andreas Hart

MMIV/MMV

Universität Koblenz-Landau
info@traha.de

Abstract. Web services are the base technology of service integration. To understand Web services, earlier integration approaches like data flow integration and data integration are reviewed. It will be found out that service integration is very similar to the object integration approach. But will service integration really solve the problems former integration approaches did not? What is the future of Web services? The collection of so many essential, open Web service standards available up to now is very promising. Standard software vendors have already started to adopt the ideas of service integration and are working on system architectures implementing the Service-oriented Architecture (SOA), what German software vendor SAP calls Enterprise Services Architecture (ESA). It will become clear that the Web service technology supports flexible, maintainable integrated systems. This can even reduce the gap between technical and business process integration.

Table of Contents

1 Look Back at Data Flow Integration

1.1 The History of Integration

Since AT&T introduced the first 300-baud modem in 1962, communication between remote computers has become more important than ever before. Advancing communication technology slowly led to loosely coupled distributed systems over the years, interconnected by the Internet and private networks. Three main concepts for the technical integration of distributed systems have yet been identified: data flow integration, data integration and object integration (see also Ferstl et al. 1998).

In the beginning of data communication, temporary point-to-point connections and simple protocols were used to transfer unstructured data (text or binary files) from one machine to another. All these transfers had to be initiated and supervised by human beings. As most data processing was done on a limited number of very expensive central mainframes, there was little need for further integration – at first. When required, the applications running on a mainframe could be tightly coupled. This could be done using techniques the mainframe's operating system itself offered, e.g. via shared memory or shared files. But falling hardware prices made communication between mainframes and the increasing amount of other available computers affordable and therefore marked the start of the era of data flow integration via public telephone networks. The aim was to integrate the up to then existing automation islands spread over different machines.

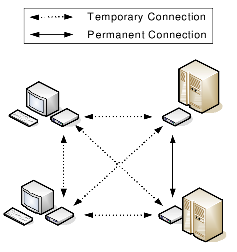

Fig. 1. Data Flow Integration Example

Data flow integration is the simplest form of integration. Only the relevant data is transferred with little protocol overhead, so low bandwidths can be used in a very efficient way. Figure 1 shows an example of a distributed system connected using data flow integration. Two loosely coupled mainframes in a simple local network accept temporary point-to-point connections from remote terminals to exchange structured or unstructured data.

As data flows only contain raw data at application level (above OSI layer no. 7), the recipient must implicitly know how to handle every different type of data. Especially in cases when the recipient is not a human being, in other words data processing at the recipient's side is fully automated, implementation of hard-coded interfaces with static processing instructions for any supported data type is needed. This strongly limits flexibility.

Though some standards for the interchange of structured business data (EDI) were introduced, the biggest disadvantage of the early integration approaches remained. Steadily increasing amounts of data transferred between separated automation islands soon led to a significant rise in data redundancy. Consequences were errors resulting from the parallel processing of decentralized data pools managed by independent applications. With a focus on data integrity, a new integration concept emerged. The main idea of data integration was to centralize all data back to only a few dedicated machines.

Data integration was accompanied by three new technological developments: relational database systems, client/server paradigm and private networks.

The first prototype of a relational database system (System R) was created in the 1970s, the theory going back to the IBM researcher Ted Codd (Committee on Innovations in Computing and Communications 1999). Commercial products based on the Structured Query Language (SQL) became available and accepted till the mid-80s. In a relational database system, data modeling techniques are used to minimize data redundancy. Data integrity is ensured by the database management system (DBMS), changes in the database are conducted solely via transactions. Relational database systems are nowadays highly standardized, for example the query language SQL is now an international ISO standard.

Separation of data and functions directly resulted in a client/server architecture of a distributed system, hence a two- or three-layer architecture. Data storage is done by one or more dedicated machines (two-layer-architecture), and typically user interface (UI) and business logic are separated, too (three-layer architecture).

The success of the client/server paradigm itself depended on an underlying network infrastructure, because servers need to be permanent accessible by a large amount of client computers. Public data networks did not exist at that time, and permanent point-to-point connections via public telephone networks were far too expensive. So private data network providers started to extend new value-added networks (VANs) that could be accessed by customers at lower total costs than the public telephone network. These VANs also offered additional services, e.g. storing of messages. Later on, the continuous demand for networking equipment and falling prices enabled large enterprises to build their own private data networks (‘Intranets’).

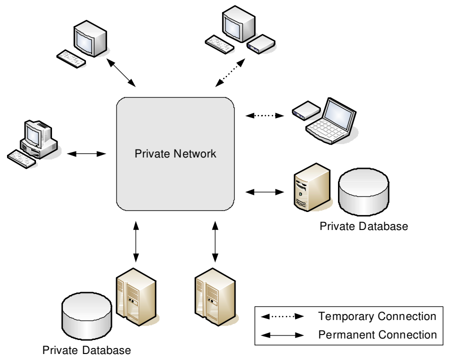

Fig. 2. Example of a data-integrated distributed System

Figure 2 depicts a typical architecture of a data-integrated distributed system from a single-enterprise perspective. The enterprise data is stored on two database servers (one of them a mainframe). A second mainframe is available for executing business logic. All machines are connected via a private network whilst some clients use only temporary dial-up connections. The central servers nevertheless have to be permanently connected to the network.

Object integration is the latest approach in technical integration. It can supplement the data integration concept and is most easily build on top of clean, data-integrated systems. While data integration addresses the integrity and redundancy of data, object integration aims at reducing the functional redundancy and organizing object interactions.

Objects encapsulate functionality (methods) and data (attributes) and communicate with each other using messages (method calls). An object-integrated system consists of specialized objects dedicated to data management on the one side, and specialized objects realizing the system's functionality on the other side. All these objects are glued together by a global communication system. So object integration is the first approach that refers to the structure of a distributed system (data redundancy, functional redundancy) as well as to its behavior (data integrity, object interactions), forming a complete technical system model. Practical solutions often reuse existing relational databases instead of implementing an object-oriented database layer.

The Common Object Request Broker Architecture (CORBA) of the Object Management Group (OMG), Microsoft's Distributed Component Object Model (DCOM) and Sun Java's Remote Method Invocation (RMI) are implementations of the object integration approach, as well as Web services are. Object integration in the context of Web services will be discussed later in chapter 2.3. To illustrate similarities between the earliest (data flow integration) and latest technical integration concept (object integration), a closer look on the EDI standard for data flow integration follows now.

1.2 Electronic Data Interchange

EDI is a generic term for several standards that support automated exchange of structured business data between computers. Besides some industry standards like SWIFT or ODETTE, the North American ANSI X12 and the United Nations EDIFACT are the most important. The EDIFACT standard is the only international standard among them, which is evolving since the early 1980s. EDI messages are structured text documents and therefore platform-independent in the main. This is important for the usage of EDI in distributed systems. In the following, an fragment of an X12-formatted purchase order, transaction set 850, is shown (Ogbuji 1999):

```
ST*850*12345
BEG*00*SA*3429**981201
N1*BY*Internet Retailer Inc.*91*RET8999
N1*ST*Internet Retailer Inc.
N3*123 Via Way
N4*Milwaukee*WI*53202
PER*OC*Obi Anozie
PO1**100*EA*1.23*WE*MG*CO633
SE*9*12345
```

On 1 December 1998, Mr. Anozie Obi ordered 100 units of ‘Fuzzy Dice’ (part no. C0633) from Internet Retailer Inc., 123 Via Way, Milwaukee WI, 53202 at a price of $1.23 per unit (order no. 003429).

EDI documents are (with some limitations) readable by human beings, though the syntax is much more restrictive and error-prone than XML syntax. In fact, EDI documents can easily be transformed to XML documents. For this example, the XML document can look like this (Ogbuji 1999):

```
<?XML version="1.0" encoding="UTF-8"?>
<PurchaseOrder Version="4010">
<PurchaseOrderHeader>
  <TransactionSetHeader X12.ID="850">
    <TransactionSetIDCode code="850"/>
    <TransactionSetControlNumber>12345</TransactionSetControlNumber>
  </TransactionSetHeader>
  <BeginningSegment>
    <PurposeTypeCode Code="00 Original"/>
    <OrderTypeCode Code="SA Stand-alone Order"/>
    <PurchaseOrderNumber>RET8999</PurchaseOrderNumber>
    <PurchaseOrderDate>19981201</PurchaseOrderDate>
   </BeginningSegment>
  <AdminCommunicationsContact>
    <ContactFunctionCode Code="OC Order Contact"/>
    <ContactName>Obi Anozie</ContactName>
  </AdminCommunicationsContact>
</PurchaseOrderHeader>
<PurchaseOrderDetail>
  <Name1InformationLOOP>
    <Name>
      <EntityIdentifierCode Code="BY Buying Party"/>
      <EntityName>Internet Retailer Inc.</EntityName>
      <IdentificationCodeQualifier Code="91 Assigned by Seller"/>
      <IdentificationCode>RET8999</IdentificationCode>
    </Name>
    <Name>
      <EntityIdentifierCode Code="ST Ship To"/>
      <EntityName>Internet Retailer Inc.</EntityName>
    </Name>
    <AddressInformation>123 Via Way</AddressInformation>
    <GeographicLocation>
      <CityName>Milwaukee</CityName>
      <StateProvinceCode>WI</StateProvinceCode>
      <PostalCode>53202</PostalCode>
    </GeographicLocation>
  </Name1InformationLOOP>
  <BaselineItemData>
    <QuantityOrdered>100</QuantityOrdered>
    <Unit Code="EA Each"/>
    <UnitPrice>1.23</UnitPrice>
    <PriceBasis Code="WE Wholesale Price per Each"/>
    <ProductIDQualifier Code="MG Manufacturer Part Number"/>
    <ProductID Description="Fuzzy Dice">CO633</ProductID>
  </BaselineItemData>
</PurchaseOrderDetail>
</PurchaseOrder>
```

Despite the focus of this paper, one should not make the error to assume that EDI is already dead. Following wikipedia.org (Wikipedia 2004), 95% of all electronic commerce transactions still rely on EDI (assumingly non-standard EDI implementations are included in this number). There might be situations where it makes sense to transform EDI messages to a XML format, but just a simple transformation would not be a sufficient incentive for companies to invest in the improvement of current productive systems. In simple scenarios, for example the daily upload of turnarounds from the point of sale (POS) to a company's ERP-system, EDI might never be replaced.

So what would be new if we additionally implemented an EDI scenario with Web services? In that case, we would not only send the structured XML data to Internet Retailer Inc., we would also invoke a Web service operation (an object method) in the target system, starting the order management process at the same time. Nothing else is currently done when starting the process with EDI, but in the case of Web services, it could be done in a standardized way along the whole order process, integrating different vendor's applications, ending with shipping and payment (in a flexible, goal-oriented way), as we will see later (see chapter 2.2). But despite the obvious advantages of ongoing standardization, it must no be forgotten that compared to data flow integration, Web services always increase bandwidth usage as well as processing time due to heavy XML wrapping and processing.

2 Basics and Principles of Web Services

2.1 Web Services at a Glance

'Definition: A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.' (W3C Web Services Architecture Working Group 2004-1)

Web services are a set of technology components and standards that allow automated communication between loosely coupled distributed objects over a network. Web services are accepted by most of the big players in the software industry (e.g. IBM, HP, Sun, Microsoft, BEA Systems, SAP, Oracle, Computer Associates), so one can almost be sure that Web services will be omnipresent when it comes to intra- or inter-organizational integration (compare to Hündling et al. 2003). And he/she might be even surer when hearing that no one of the big players in the market ever really criticized the main concepts behind Web services.

Web services are built on a foundation of commonly accepted open standards. The basic standards for technology components in Web services are SOAP, WSDL and UDDI. There are some other, more sophisticated standards like WS-Coordination, WS-Transaction, WS-Resource, WS-Policy and BPEL (Business Process Execution Language), except the latter beyond the scope of this paper.

The Simple Object Access Protocol (SOAP) is a standard for invoking operations within Web services using XML messages. SOAP version 1.1 was submitted for standardization to the World Wide Web Consortium (W3C) in March 1999.

As an introductory example, the ‘Daily Dilbert’ service that can return a path to an image (http://www.esynaps.com/WebServices/DailyDiblert.asmx) is now presented. This is a simple request/response service. It is invoked using the following SOAP-message:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
 xmlns:xsd=http://www.w3.org/2001/XMLSchema
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <DailyDilbertImagePath xmlns="http://tempuri.org/" />
  </soap:Body>
</soap:Envelope>
```

Actually, the remote DailyDilbertImagePath operation is invoked with no parameters. The following response message (DailyDilbertImagePathResponse) has a single parameter DailyDilbertImagePathResult containing the URL of the image and looks like this (* below is a placeholder for the actual URL):

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
 xmlns:xsd=http://www.w3.org/2001/XMLSchema
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <DailyDilbertImagePathResponse xmlns="http://tempuri.org/">
    <DailyDilbertImagePathResult> * </DailyDilbertImagePathResult>
    </DailyDilbertImagePathResponse>
  </soap:Body>
</soap:Envelope>
```

A SOAP message is either a request, response or fault message. It always consists of an Envelope element that contains an optional Header element and a mandatory Body element. The Body element comprises further elements determining the affected operations, plus the appropriate parameters. Very briefly, SOAP is used for the invocation of remote operations (methods), supplying the name of the operation called, together with the serialized parameters. The message data is wrapped into structured XML documents, resulting in human-readable messages.

SOAP is a simple yet powerful protocol supported by W3C and the software industry. As most Web service standards, it is continuously reviewed and might be extended in the future.

2.1.1 Object Orientation

Simula 1 (1962-1965) and Simula 67 (1967) were the first two object-oriented programming languages (Reed 2003). Among other well-known concepts, the basic idea of the object orientation paradigm is that a system consists of independent objects, having an inner and an outer view and interacting using messages. Messages are exchanged so a sender object can use another object’s functionality expressed in the receiver object’s public interface (its supported data objects and methods). This functionality is implemented inside the receiver object, falling back upon internal data objects and internal operations. Internal data objects and operations are not accessible from the outside other than via the public interface of the object (information hiding).

Distributed objects are objects in a distributed system context (see chapter 2.1.4) exchanging messages via a joint network. So what about Web services? Is a Web service really a distributed object?

SOAP messages are structured messages. SOAP messages are sent from a client to a specific Web service. Operations and data object types a Web service can handle are defined with the means of Web service standards. The UDDI standard (Universal Description, Discovery and Integration) for example includes storage of abstract descriptions of a Web services in a service repository (see chapter 2.1.2). So a Web service has a well-defined interface, hopefully looking the same as described in the repository. The interface is hiding a concrete implementation, either done in an object-oriented programming language like Java or C++, or in any other language, they can even wrap existing legacy applications, so Web services translate the basic and most effective ideas of object orientation like information hiding, message exchange, abstract interfaces and reuse of software components into action.

The concrete implementation being considered a part of a Web service solely determines if Web services are distributed objects or not. According to the definition of the W3C Web Services Architecture Working Group given at the beginning of this chapter, a Web service from their point of view seem to be more a part of the distributed object shown in Figure 3. The standards for Web services do not extend beyond the outer view of a distributed object. Hence, the term Web service currently used by the W3C only refers to a technology that supports 'interoperable machine-to-machine interaction over a network'. It is a lightweight approach (Hündling et al. 2003), separated from implementation aspects. The inside of the distributed objects is not relevant to Web services. Only the outer view needs to be standardized. Therefore, the interfaces of Web services, as well as syntax and maybe semantics of messages have been, are, or will be standardized. But it is important to stress that the W3C's definition focusing on the outside of distributed objects (and the technical framework) does not deny their existence. In fact it depends on the scope of the examination if the distributed objects can be recognized. Generally spoken, a Web service is a distributed object (compare to Birman 2004).

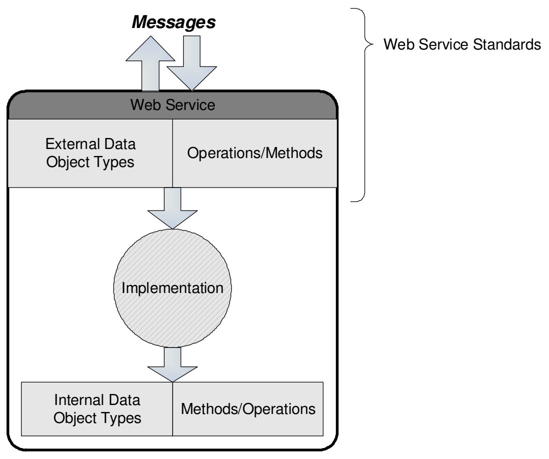

Fig. 3. Model of a Web Service (adapted from Ferstl et al. 1998)

2.1.2 Central Repositories

System architectures built on Web services rely on central repositories wherein the interface descriptions of available services are stored. Clients searching for a specific functionality can browse these Web service repositories to find one or more suitable services offered by a service provider. Communication is done using SOAP messages (so a service repository is a service provider, too). There are two important standards for service repositories: WSDL (Web Services Description Language) and UDDI (Universal Description, Discovery and Integration).

WSDL is a standard for machine-processable interface description. A WSDL document is a XML document that provides an abstract description of a Web service's functionality and data formats (data object types). Besides these abstract descriptions, WSDL documents also provide concrete bindings to communication protocols (e.g. SOAP and HTML) and service implementations, including the location (e.g. URL) where the service can be reached (Hündling et al. 2003). After accessing the service description in a WSDL document, a client can then create the appropriate SOAP message to invoke a particular service (dynamic invocation).

UDDI is a general open standard for functionality, data model and architecture of Web service repositories (Bellwood et al. 2002). Since 2002, the UDDI standard is further improved under the Organization for the Advancement of Structured Information Systems (OASIS). The initial idea behind UDDI was to set up a public and global yellow pages network to connect businesses, regardless of technology and infrastructure. Even with possible security risks and legal restrictions not taken into account, it showed up that not many enterprises or organizations were keen on exposing their private system functionality and data – often a strategic asset – to the world. Therefore, up to now only a handful of public repositories exist.

A more realistic approach currently under development (see chapter 3) is to start within enterprises, building private infrastructures at first. Selected and trusted business partners can then be included in a second wave. With a critical mass of private repositories and growing experience in the field of Web services, an increasing amount of public or semi-public Web services in the consumer area (most probably behind a Web-based front end) may be the future result. Surely, interoperability and security have to be asserted and proved before Web services will really leave the boundaries of private networks.

2.1.3 Internet Technology

Existing public or private network infrastructures can be used for transport of messages between Web services and between Web services and repositories. Nowadays these networks are most often based on open Internet standards, TCP/IP being the most important two of them. TCP and IP are the transport-layer protocols of the Internet, and they are the base of commonly known Internet services like FTP (file transfer), HTTP (transport of Web pages) and SMTP (e-mail dispatch). The complete Internet protocol stack is depicted in figure 4. The Internet protocol stack in essence is a slight simplification of the Open Systems Interconnection (OSI) reference model published in 1984.

SOAP messages are usually transported with HTTP, though FTP or SMTP as well as any other non-Internet protocol can surely be used, too. Web service standards are built on open Internet standards and hence can get benefit from their popularity. With

Internet and private networks converging and Internet technology being a common base, Web services are not limited to the Internet; they can easily be introduced on private networks (‘Intranets’), too.

XML is the other important key to success for Web services. Nearly everything in Web services is encoded in XML, resulting in human-readable, platform-independent and language-independent messages.

Web services should not be confused with Internet services (HTTP, FTP…). Web services are part of an application process (middleware) that can use Internet services to exchange data or messages; hence they are a level higher than Internet services (above OSI layer no. 7). That’s why SOAP messages can be transported via HTTP. EDI messages can be transported via HTTP, too. While EDI messages are virtually raw data, SOAP messages are sort of more sophisticated (structured) data, but still data generated directly by some kind of application. This application uses a network protocol (e.g. an Internet service) to transfer SOAP messages.

Web services should not be confused with the World Wide Web (WWW) either. The Web is a conglomeration of HTML documents, which are managed by Web servers. Like Web servers process HTTP requests and send the requested HTML documents (or not), Web services (‘SOAP servers’) are sending and processing SOAP messages (or not). There can be a Web server and a Web service, both being an application and using HTTP as a transfer protocol (notice also figure 5 on the next page).

OSI Layer Number	Layer Name	Example
5-7	Application	HTTP, FTP, SMTP etc.
4	Transport	TCP, UDP
3	Network	IP
1-2	Network Access	IEEE 802.x etc.

Fig. 4. The Internet Protocol Stack (adapted from Weber 1998)

As well as open Internet standards like TCP/IP and HTTP unified the Internet and private networks, Web services have the potential to unify technical integration inside and in-between enterprises. The position of vendors solely offering proprietary integration solutions will be weakened with lasting effect.

2.1.4 Distributed Systems

A distributed system is the environment of Web services, supplying an often pre-existing infrastructure to be used (with little additional costs). The Internet together with its surrounding private or semi-private networks is one of the biggest distributed systems ever built. But it is only one possible system, separated from other distributed systems like military networks or even simple two-node systems connected via public telephone lines.

A distributed system is defined as a system of two or more computers or processors with at least one runnable distributed application installed. A distributed application is an application running on several computers or processors and exchanging information between them (Weber 1998).

Distributed applications can be loosely or tightly coupled. Tightly coupled distributed applications operate on a shared memory (e.g. a single database). Data-integrated applications are tightly coupled, though they not necessarily need to be distributed applications. Loosely coupled distributed applications have their own private memory (internal state/database) and communicate using messages. Web services are a set of technology components and standards that make integration of loosely coupled distributed applications easier.

Figure 5 depicts an example of a loosely coupled distributed application. It is important to realize that this is in effect a client/server implementation: requests or method calls are sent from the left side (client) to the right side (server), from where requested data or processing results are returned. Communication is completely based on open Internet and Web service standards like the Internet protocol stack from figure 4.

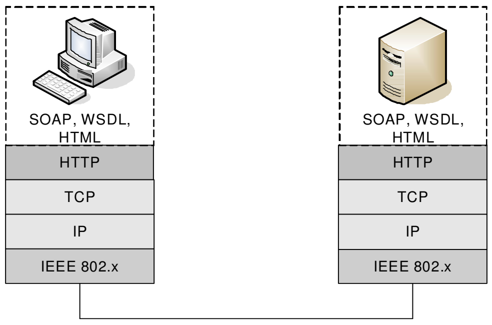

Fig. 5. Example of a simple distributed Application based on Internet Technology

2.2 Composed Web Services

‘There are two types of Web services – simple and complex. Simple Web services provide basic request/response functionality, are typically not transactional in nature, and provide simple HTTP-S/SSL based security. They are developed supporting three primary Internet standards – SOAP, WSDL and UDDI. Complex Web services can be characterized as multi-party, long-running business conversations, that involve sophisticated security, such as non-repudiation and digital signatures, as well as business-to-business collaboration and business process management’ (Oracle 2001)
As a general rule of thumb, simple Web services (as they were described in the previous chapters) should be implemented stateless, i.e. with no access or use of information not contained in the input message (Foster et al. 2004). Following Foster et al., this tends to enhance reliability and scalability, because a stateless service can be restarted following failure without concern for its history and prior interactions. Multiple instances of stateless services can be created for load balancing. Though stateless services themselves have no durable internal state, they can nevertheless interact with stateful resources (e.g. databases), so responsibility for state management can be delegated to these external resources.

The search functionality as part of the Amazon Web Services and in particular of the Amazon E-Commerce Service (http://www.amazon.com/webservices) providing access to Amazon's product inventory listings (by returning a Item object or an Items collection) is an example for a simple stateless Web service. Amazon also offers a remote shopping cart. The remote shopping cart functionality however is a set of operations that can be executed subsequently. The state of the shopping cart is managed by the Amazon database, but a CartID is returned after creation of a shopping cart that can be used for further requests. Subsequent operations on a single shopping cart (e.g. add items, clear cart) are a very simple example of an interactive 'manage shopping cart' process based on Web services technology.

Operations of simple Web services can be composed to reflect business processes and offered as a single service afterwards (Hündling et al. 2003). Composition of operations is the way to create composed Web services, the same what Oracle calls complex services (as mentioned in the quotation above). Functionality of an existing software system for example can be decomposed into rather fine-grained operations, wrapped into several Web services. These rather simple services can then be invoked by services of higher value. An additional, directed control flow must be introduced to manage the time or logical dependencies inside and in-between the composed services. Composition can be done recursively, spanning different levels of granularity. In other words, composed services on the upper levels can invoke simple services as well as other composed services. Composed services aggregate and hide low-level business functionality.

The top-level services, providing higher-value business functionality, can represent activities related to a business process. A business process is defined as a set of business activities being necessary to handle a business transaction. The single business activities can be processed within different organizational units, but are usually connected by time or logical dependencies. Business activities run coordinated, either parallel or successively. They are executed manually or computer-based, and their purpose is to reach a certain goal. Within an organizational unit they serve the purpose of reaching a business goal (Grässle et al. 2004).

Business activities have a well-defined interface, regarding possible inputs and possible outputs. Business activities are connected via transitions. If conditions exist that make clear which transitions become active after completion of an activity, and which do not, then an activity can have more than one outgoing transition. Web service operations can be seen as an implementation of activities in this model.

Process modeling is the domain of Workflow Management Systems (WFMS). For obvious reasons the combination of Web services and existing Workflow Management Systems is hence a promising approach. Web services simplify the deployment of business functionality inside and in-between organizations. Processes involving partners in different organizational units and in particular outside an enterprise can be realized on the base of Web service standards.

Operations offered by Web services are activities in a defined control flow. These operations are described and can be located by browsing service registries. They can be called from the inside and outside of an enterprise using SOAP messages. In a workflow context, SOAP messages are a form of triggers that start, continue or end a business process. When appropriate configured and secured (or when not appropriate secured), SOAP messages can even pass enterprise firewalls.

Many business standards software vendors already have workflow engines integrated into their software solutions. Additionally, the Business Process Execution Language (BPEL), which is part of the still-evolving Web service standards, supports the modeling of composed services and control flows while providing means to store a global process state (Hündling et al. 2003). Hence the integration goal of managing the control flow can be reached with process modeling. This goal pursued by process integration approaches addresses the behavior of a system from a user perspective.

Coming back to the EDI example, it is important to mention that EDI-like SOAP messages transporting business documents like purchase orders, invoices or even CAD drawings can trigger state transition in business processes, hence support the automation of various processes over the Internet or another network.

By composing Web services and defining control flows in a process model, an additional process layer on top of business functionality is provided. This process layer is easier to adapt to dynamic changes, as it is separated from implementation aspects for the most parts. Hence additional flexibility and maintainability are some of the benefits of Web services in the domain of workflow management. Disadvantages however are increasing complexity and decreasing overall performance, especially when service granularity is too low.

Existing process models are usually static and predefined. The possibility of dynamic invocation of Web services, i.e. choosing one out of a set of candidate Web services from a repository, can add dynamic aspects to static process models. With activities being nodes in a process model, one can think about certain standard nodes that can be reused in many processes. Apart from workflow-related nodes (branches, joins or common services like for example mailing or messaging nodes), one of these standard nodes for example could be a generic node taking decision rules as input parameters and returning an appropriate Web service based on the given rules. The rule inference engine itself can be implemented as a Web service. Instead of rule inference (Zeng et al. 2003), other artificial intelligence (AI) techniques can also be used.

In reality, existing software solutions may not be structured to support static or even dynamic composition of Web services yet. There are usually many interdependencies hidden deep inside of program code, even in object-oriented systems. This complexity can vary from slightly complex to extremely complex, depending on the extent of the software solution. So these applications have to be reviewed first, and in some cases existing business functionality has to be reorganized or reprogrammed (e.g. Bayer et al. 2004). In any case, the database layer can usually continue to exist without changes.

To execute composed Web services, an extensive central infrastructure (see chapter 3.2) is needed to manage the distributed functionality. This infrastructure must at least provide an environment for development of the process models (plan), an environment for creation and administration of simple and complex services (develop), as well as a runtime environment for the execution of processes (run). In particular, this runtime environment must provide means to execute long-running distributed transactions. This infrastructure will here be called composition engine.

As an example for how to use composed Web services, an idea introduced by Geng et al. in 2003 is now discussed. Smart Marketplaces integrate various service providers and thus offer personalized services for consumers or business users over the Internet. Geng et al. mention Expedia.com offering consumers to ‘customize complete leisure trips to meet their individual preferences’ as an example. Consumer portals are outlined where the backend integration of functionality among different enterprises forming an alliance is done with composed Web services, in opposite to situations where each service provider only offers his own (Web) services to his customers. Integration of complementary Web services could really offer an additional value to customers, and it opens possibilities for entrepreneurs to new marketing opportunities and cross-selling their products. It should be mentioned that the main front-end of such marketplaces still has to be a conventional website consumers are already comfortable with. So except for B2B integration partners themselves, Web services are almost invisible (transparent) to end-users.

2.3 Service Integration

In this chapter, an example of an object-integrated system from a single-enterprise perspective is presented. This system depicted in figure 6 is an evolution of the data-integrated system shown in figure 2. Integration based on Web services is an object integration approach (compare to Burbeck 2000). Web services are based on specific, commonly accepted, open standards and implement only a few object-oriented ideas. For example there are no classes (just interfaces), and inheritance is not defined for Web services, though this would surely be possible (interface inheritance). So the Web services approach might be better called service integration or more specifically Web service integration, though it has a strong relationship to object integration. Popularity and broadly accepted open standards may be the most important reasons to distinguish service integration from object integration.

Service integration can be seen as a simplification of the object integration approach, transporting the most successful ideas into an Internet environment and casting them into a lightweight approach based on open standards. So the object-oriented integration approach in a Web services context will here be called service integration.

It is striking that figure 6 is much more extensive than the previous examples. This is mainly because of the Internet coming into visibility. The Internet as a global communication platform evolved since the 1990s till now. When it comes to public networks like the Internet, enterprise data and functionality has to be secured against any unauthorized access. Hence it might be sensible to introduce a semi-private zone or domain between the public domain (Internet) and the private domain of an enterprise. Access restrictions between the Internet domain and the semi-private domain may be lower than restrictions between the semi-private and the private domain.

Direct access to the databases is not necessary under normal circumstances, because private Web services represent a service-oriented interface to access or update business data. Private services should be accessible from the private or semi-private domain, but not directly from the public domain. So access to private databases is only possible from the private domain. Access from the semi-private domain may be subjected to strict rules.

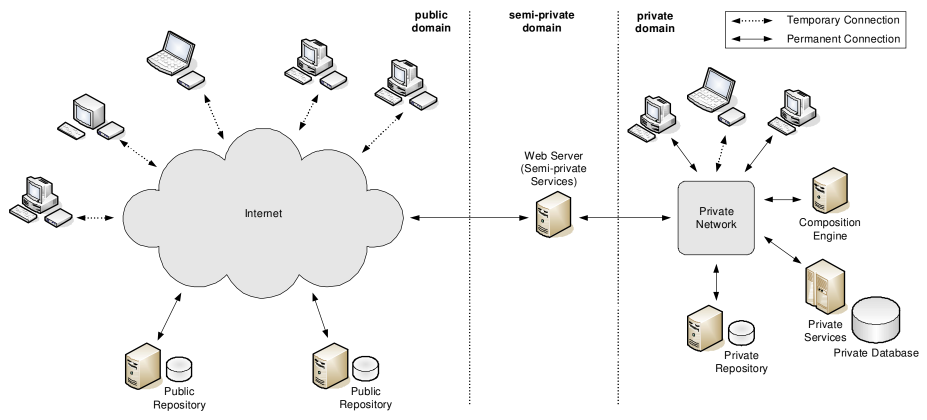

Fig. 6. Example of a service-integrated distributed System

Semi-private services are accessible from the inside or outside of an enterprise. They can either be publicly available (e.g. registered to public repositories), or only for a closed group of users (requiring some means of authentication). Semi-private services should not access private databases directly when receiving messages, but rather invoke private services (low-level or high-level) for further processing. Hence semi-private services can be seen as public accessible entry points, but they only invoke private services that have the exclusive rights to operate on private data. It is remarked that semi-private services provide indirect access to private data while they can be public accessible, that's why they are called semi-private here.

Because Web services in general and semi-private services in particular operate similar to Web servers (see chapter 2.1.3), semi-private services can be hosted on an existing Web server with extended capabilities. In general, Web services are often implemented as an add-on for existing Web servers (e.g. http://ws.apache.org/).

Clients from inside an enterprise can browse private or public registries or invoke public or semi-private services directly. They could also invoke other enterprises' semi-private services (B2B integration). Clients from the Internet can browse only public registries, or they can invoke semi-private services directly (rather in B2B scenarios). It is more likely that access to semi-private services of an enterprise is provided to customers or consumers via a deployed application or a Web application (e.g. a portal or Internet marketplace).

Public repositories are maintained by independent institutions and can be accessed by everyone. Private repositories on the other side should be – like private databases – only accessible from the private domain. They are the 'yellow pages' of business functionality of an enterprise and therefore restricted to internal use.

Finally, the composition engine provides an infrastructure for the execution of composed Web services. These composed services can be made accessible as single services; hence the composition engine is a service provider, too. This means that composed services can be registered to the private registry; and composed services can be invoked just like other (simple) private services. When it comes to the Enterprise Services Architecture (ESA), the composition engine will play a central role in this service-integrated system (see chapter 3.2).

It is noted that the scenario presented is only one of many possible scenarios. This new flexibility facilitated by Web services can also be seen as a thread. There are still many uncertainties how an optimal service-integrated system will look like. The right level of functional decomposition – not too low to degrade performance, but also not too high to prevent reuse – depends on the purpose of a given system; generalities cannot be found easily. On the other side, it is obvious that service integration will initially increase the overall complexity of a system, though later improvements in system functionality are made easier. Hence a clear strategy for Web services rollout is needed before service integration can be translated into action.

3 The Future of Web Services

3.1 Service-oriented Architecture

The Service-oriented Architecture (SOA) is the basic idea behind service integration. The term Service-oriented Architecture was introduced by IBM researcher Steve Burbeck in 2000. Service-oriented systems consist of loosely coupled distributed objects (Web services) that are registered to a central repository and hence can be invoked dynamically. In fact, the combination of central and distributed components is the base of success of the Service-oriented Architecture. This simple concept has made the Internet a reliable platform for information exchange, as well as it is the foundation of data integration. Replication of data, repositories or services can be used to avoid a potential bottleneck.

Web services are a current implementation of the Service-oriented Architecture, though SOA being a generic concept not bound to any specific implementation (Hündling et al. 2003). As much as one knows about Web services now, as much does one know about the Service-oriented Architecture. But it is even simpler: in a system that has a Service-oriented Architecture, each component plays one of only 3 possible roles. Every system component is either a service requestor, a service provider or a service broker (see figure 7). If there is something that is not a service requestor, a service provider or a service broker, it is not part of this system. The system components interact using messages; they are loosely coupled. This is the only form of possible interaction between the system components. Hence the Service-oriented Architecture is a strictly reduced view on a distributed system (the real world). Figure 6 for example contains a Service-oriented Architecture.

Service requestors and service providers are decoupled. A service broker is the intermediary who brings them together. The broker pattern is the essential architectural element of the Service-oriented Architecture, and it is often found in other object integration approaches like OMG CORBA or Microsoft DCOM as well.

With Web services being an implementation of the Service-oriented Architecture, a repository is a service broker, a client is a service requestor and a server is a service provider. A server publishes (SOAP) its available services to a central repository (UDDI). A client can search these services (SOAP, WSDL) in the repository to find the service that answers its needs. It can then bind the service and invoke the according operations (SOAP).

The biggest potential of the Service-oriented Architecture lies in easily connecting two arbitrary separated systems if they both have been built on open Web service standards. The technical integration of such systems will require less time, it will be more reliable and future changes in the service-integrated system will be easier. In reality, integration solutions from different vendors still have some incompatibilities (Hündling et al. 2003), but hopefully these problems will be overcome in the near future.

In summary, Web services are a simplification of object integration concepts, as well as an implementation of the Service-oriented Architecture, hence an implementation of the broker pattern designed for loosely coupled distributed systems. Integration of service-oriented systems is here called service integration.

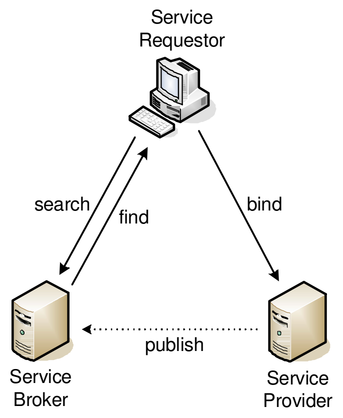

Fig. 7. Service Roles and Interactions (adapted from Burbeck 2000)

3.2 Middleware as a Starting Point

Middleware is a shared, multi-purpose communication infrastructure to connect distributed applications. It provides several communication mechanisms like for example message passing, message queuing, remote procedure calls and object invocation (Siedersleben et al. 2003). Middleware plays an important role in making integration of enterprise applications (EAI) reliable. However, inter-organizational collaboration was always hampered because of the lack of standards among different middleware solutions, which made B2B integration projects expensive and long-running ones. Web service standards will help to overcome these problems.

Software vendors already offer several products (e.g. Microsoft .NET, IBM WebSphere, BEA WebLogic, SAP NetWeaver) that promise vendor-independent interoperability by supporting open Web service standards. These products provide a runtime environment for Web services. Middleware can receive messages, stores them safely and dispatches them to attached Web services, including error handling. The above introduced composition engine is part of an extended middleware solution.

With Web services, existing middleware solutions will be enriched with new functionality to plan, build and run simple as well as composed services. In the future, enterprises can concentrate on the integration of business processes. Software vendors will provide standard solutions that will minimize possible technical problems. So enterprises can concentrate on what to do, not how to do it (compare to Burbeck 2000).

Following a Meta Group survey among 289 medium-sized or large German enterprises, 77% of them plan to implement a Service-oriented Architecture as least inside the organization in 2005. For end of 2006, 75% of them plan to extend this architecture into B2B scenarios (Reiter 2004-1).

The objectives of the first wave are to consolidate and further integrate an existing system of separated applications, either only in occasional organizational units or for the enterprise as a whole. An Enterprise Service Bus (Kossmann et al. 2004), i.e. a whole enterprise application landscape integrated with Web services being the one and only integration technology, may be a long-term, evolutionary strategy.

The second wave will focus on integration with selected external partners, but even more on integrated processes along the supply chain, including suppliers as well as customers. Following the survey, consumers are not part of the strategy yet.

Service integration is an integration technology that is suitable for either intra-organizational enterprise application integration (EAI) as well as for inter-organizational B2B integration. They will replace or supplement proprietary integration solutions and will be the preferred integration technology in the future. Middleware will provide a universal base technology for service integration.

3.3 Enterprise Services Architecture

The Enterprise Services Architecture (ESA) is a strategy of the German enterprise resource planning (ERP) software vendor SAP, though there seem to be some similar approaches like the NATO C3 Technical Architecture (http://nc3ta.nc3a.nato.int). Known SAP products are for example R/2, R/3 and the mySAP business suite. SAP has more than 24000 enterprise customers and over 84000 installations worldwide (http://www.sap.com). SAP is one of the market leaders in ERP.

The Enterprise Services Architecture was first presented by SAP in 2003 (Bayer et al. 2004) and is a recent approach to combine the technical service integration approach with business process integration, i.e. to add more semantics to the Service-oriented Architecture (compare to Burbeck 2000). ESA is built on SAP NetWeaver, SAP's middleware solution supporting Web service standards.

Similar to the above presented idea of Smart Marketplaces, the Enterprise Services Architecture will enable enterprises to built Intranet and Internet portals or other Web applications with backend functionality being provided by Web services or composed Web services. These services will be invoked depending on actions a user takes on the front-end. Messages sent from clients to service providers will be events that trigger business processes. Hence, bringing business processes to the Web, enabling e-business and collaboration without boundaries, as well as further automation of existing business processes is the vision.

The path to an Enterprise Services Architecture can be understood most easily by taking current existing application landscapes inside enterprises into account. SAP R/2, SAP R/3, mySAP business suite applications (e.g. CRM, SCM), as well as most other currently purchasable standard business software systems, are data-integrated systems. So most enterprises are currently operating with a set of (hopefully) data-integrated systems. These systems have been installed and adapted during the past years, usually with high investments involved. So these systems have to be considered when introducing a Service-oriented Architecture.

The combination of the service integration scenario (depicted in figure 6) with these data-integrated applications can lead – after some abstractions – to a basic model of the Enterprise Services Architecture as it is depicted in figure 8.

The client/server paradigm and three-layer architecture are basic principles of modern system architectures, so they are used here to roughly structure the model. A data-integrated enterprise application usually has a three-layer architecture and a single database; a preexisting set of enterprise applications can be recognized on the right side of the figure. Notice that they implement the hidden functionality of Web services as it was already depicted in figure 3 (hatched background).

Web services and the composition engine are located in the business logic or application layer, above the database layer and below the user interface. In NetWeaver, counterparts are called Web Application Server (WAS) and eXchange Infrastructure (XI). The composition engine alias XI will play a central role in ESA (Bayer et al. 2004). Notice that the Service-oriented Architecture from figure 7 can be recovered within figure 8, too (dark background). A service repository is a service broker, composition engine and Web services are service providers, and all of them can be contacted by a client. Every component in ESA can be reached over the network, and components can interact with each other.

Because existing applications will continue to be used as they were used before, the SOA components are located besides these applications, and not above. A single business process layer on top of existing applications might be a long-term goal on the path to an Enterprise Service Bus (ESB), but for now and for at least the next 10-

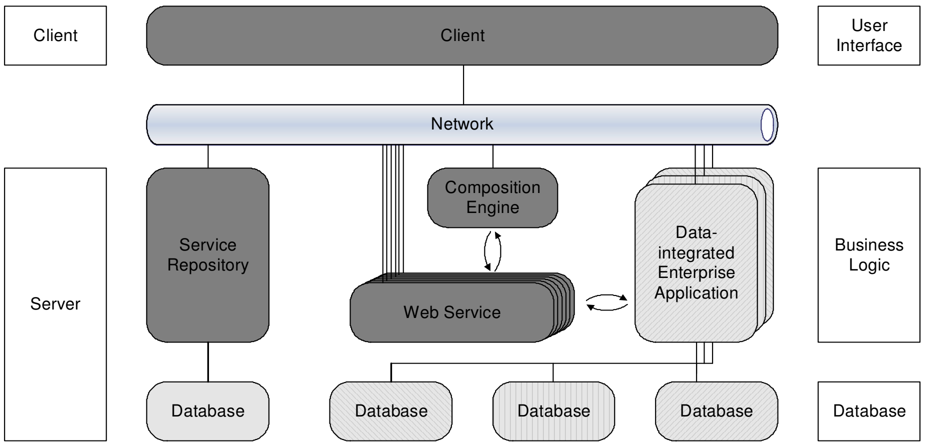

Fig. 8. The Enterprise Services Architecture

20 years, SOA and existing applications will be used parallel, so the Enterprise Services Architecture broadens the existing business logic layer instead of forming the single interface to the business layer; hence it is an evolutionary strategy.

It is not shown here that the user interface functionality on the client side may not only be realized by a proprietary client application or a local Web application, but also by dedicated Web servers, portal servers ('UI servers') or by Composite Applications (SAP 2004), resulting in a so called thin client, i.e. for example a simple Web browser.

A client can access the business functionality provided by a data-integrated application by invoking an appropriate Web service. This Web service hides the complexity of the underlying application. A client can also invoke composed services by contacting the composition engine, which then invokes the appropriate simple Web services. The repository serves as a dictionary of business functionality and is the first instance a client in search for specific business functionality usually contacts.

The most striking innovation within the Enterprise Services Architecture is an additional service-oriented access layer. This technical process layer can be used to automate processes as it provides access to high-level and low-level business functionality. A business process view can be put on top. In fact SAP cooperates with the German consulting company IDS Scheer AG, resulting in the ARIS business modeling toolset to become tightly integrated into SAP NetWeaver.

After SAP has continuously expanded its suite of products in the recent years, SAP today no longer has one single core product (R/3), but several specialized solutions that can be bough by customers one after another. So with the Enterprise Services Architecture, SAP has a powerful toolset to integrate its several existing and future standard products. The most likely way SAP will go is to introduce reference processes that customers can adapt to their needs afterwards. The simplest possible adaptation is to extend these standard processes, e.g. integrate a third-party product into the control flow. However, deep changes to these standard processes will still require a lot of work. So flexibility is raised, but unlimited flexibility cannot be reached without a price. Unfortunately, descriptive programming of business processes was not invented yet.

Web services tend to raise system complexity, but most of this complexity is hidden behind the interfaces, so from a business perspective, it may look like complexity in fact has been lowered. But as soon as one has to look behind these interfaces, he/she will suddenly face the total complexity of the system. So remember that Web services just hide complexity, in any case it does not disappear magically. As standard software vendors continuously improve their products, Web services will hide the implementation details of the 'real' products from the outside world. However, it might become necessary to change the interface of a Web service due to new functionality. So SAP might introduce a versioning concept for Web service interfaces to ensure downward compatibility. Repositories may be used by a client to find the right version of an interface.

In this context, it is very important to realize that Web services have been, are, and will be a technical view on a system. On the other side, business models have been, are, and will be a user view on a system. Somewhere, these different views will clash.

The combination of both views can describe the behavior of a system as a whole (Ferstl et al. 1998), i.e. behavior from a technical perspective (service interactions) as well as behavior from a user perspective (control flow). The business view is a world with only a few, rather simple rules, processes, activities and control flows. The technical world however is much more complex, there are stricter rules, extensive listings of source code, there are potential side effects between different modules, potential side effects of source code changes, there are data types, databases, transactions and so on. So there will always be a gap between these two worlds, and it is important to realize that this gap cannot easily be closed. So if one changes a business process model, this implies in the most cases changes in technical model and implementation, too. If one changes the technical model, i.e. the interfaces, she/he has to adapt the business model. If there is one system and two consistent views on this system, then if either of these views is changed, this results to two systems, one being the desired, new system, and the other being the existing system. Manual changes to the existing system are then necessary to close the newly emerged gap. Both technical as well as business process experts are needed for this task.

Products offered by third-party software vendors can be integrated into ESA, provided that SAP and these vendors cooperate and set up compatible standard processes. In general, the integration of complex systems still requires hard work of experts, technical and business process experts. Web services are just a common ground to start from, not more, and not less. The assumption that anyone can integrate technical systems just with a few mouse clicks only because functionality is wrapped with Web services is certainly an illusion. Development of new functionality and reorganization of existing functionality is often needed after constructing the desired view on a system, and the effort to implement these changes in such a complex environment should not be underestimated.

Technical as well as process standards may help to reduce the complexity of the real world. Network standards, database standards, Web service standards, repository standards and reference business processes are some that are to be mentioned in the context of the Enterprise Services Architecture. High standardization usually gives an extra value to software investments. In the technical domain, another recent shift in SAP's strategy concerns its proprietary programming language ABAP, which will be reinforced by the Web programming language J2EE (http://java.sun.com) and the open source development environment Eclipse (http://www.eclipse.org) in the future. So it seems that even SAP will more and more bet on open (technical) standards.

An Enterprise Services Architecture is mainly addressing larger companies, but it might also be a chance for smaller companies to realize the vision of a completely integrated IT landscape more quickly, because of their usually lower automation level and lower overall complexity of business processes reached by now. Forming Business alliances, outsourcing of processes and even enterprise takeovers (Reiter 2004-2) are made easier if the partners can integrate their IT systems using or enhancing an existing Service-oriented Architecture.

To sum up, the Enterprise Services Architecture is a long-term strategy to build a service integration layer on top of existing data-integrated applications, and a process integration layer on top of the service integration layer. It has to be proven if this approach will speed up system development while ensuring software quality and reliability at the same time. Standard processes, standard system architectures and standardized base technologies, wherever possible, may be a rough guide through complexity and flexibility.

4 Summary

4.1 Web Services as a Base Technology

In this paper, current developments in the area of the Service-oriented Architecture have been discussed. A short overview of the history of integration was presented. The data flow integration approach which came down from the early times of distributed computing was presented. Consistent databases were soon realized as the most valuable asset in an IT landscape, so data integration was the next concept, regarding that data should rather be stored centralized than distributed. Object integration is the latest approach in technical integration. Object integration is the first approach that takes structure and behavior of a technical system into account; hence object integration approaches provide a complete technical view.

Objects encapsulate functionality and data. Objects have a public interface that other objects can invoke by sending messages. Web services are distributed objects. For various reasons, and to distinct the highly-standardized service-oriented integration approach for loosely coupled distributed systems from earlier object integration approaches, the term service integration was here introduced to refer to integration based on Web services and the Service-oriented Architecture.

The technical framework for service interaction, usually implemented within middleware, is also called Web services, which can easily lead to some confusion. Essential Web service standards with a very close connection to the Service-oriented Architecture are SOAP, WSDL and UDDI. Furthermore, the importance of object-oriented concepts, central service brokers and distributed service providers, XML and open Internet standards contributing to the success of Web services was mentioned.

Web services can be composed to provide high-level business functionality. These composed services can be seen as activities in business processes. However, an invocation of a single composed service can result in interactions between an enormous amount of objects until a reply is sent back – functionality in a technical system is very fine-grained. Business processes on the other side are rather coarse-grained. The gap between a technical view and a business process view will remain, though it may be reduced with the help of standard technology and sophisticated development tools. Technical as well as business process experts are still needed.

Two examples shown were Smart Marketplaces and the Enterprise Services Architecture, ESA being SAP's strategy to add more semantics to SOA. Finally it has been shown that even now, enterprises are taking the first steps towards a Service-oriented Architecture. With Web services as a base technology, they can concentrate on what to do, not how to do it.

4.2 Conclusion

Service integration combined with data integration is the state-of-the-art in technical integration. Web services are the technological counterpart to process integration in loosely coupled distributed systems. Service integration is accompanied by three new technological developments: XML, Web service standards and public networks.

Service integration has the potential to absorb data flow integration (e.g. EDI), especially in scenarios were maintainability and flexibility criteria outweigh performance criteria. This is in particular true for integration scenarios in loosely coupled distributed systems, and it should be true for many more business-related integration scenarios inside or in-between enterprises. In short, existing EDI standards can be reused to give semantics to service integration.

Technically seen, Web services are not much more than highly standardized 'SOAP servers' reinforced by service repositories. Maintainability and Flexibility are increased; flexibility on the other side might be the biggest thread. Web services need semantics to be useful. Fine-grained (low-level) Web services decrease performance and raise complexity, but raise flexibility of the system. Coarse-grained (high-level) services raise performance and lower complexity, but lower flexibility, too. The optimal granularity cannot easily be determined.

Technical integration based on Web services has to be separated from business process integration. These are two different views on a technical system. There is a gap between both views. A service-oriented view on system functionality might however reduce the gap between both worlds. In particular, tools can be introduced that support the development of new functionality based on existing standard processes. Hence, Web services as part of a long-term strategy can support faster and cheaper development of new functionality while preserving the possibility of reusing already existing functionality.

So that might be the potential and the future of Web services: First a step back, encapsulating proprietary systems and EDI functionality, building a common ground, then a step forward, enabling intra- and inter-organizational collaboration based on process models that are easier to change and maintain.

A question often asked is what can be done with Web services. This is certainly an approach starting from the wrong side. There must be an initial business goal; and only the next question is how this goal can be reached by using a technical system. Web services are only one of many possible technologies that can be used, though they may become the single common integration technology.

References

Bayer, M. and Niemann, F. (2004) 'SAPs langer Weg zur ESA', COMPUTERWOCHE, online at: http://www.computerwoche.de/index.cfm?pageid=254&artid=57375 [accessed 20 November 2004]

Bellwood, T., Clément, L., Ehnebuske, D., Hately, A., Hondo, M., Husband, Y. L., Januszewski, K., Lee, S., McKee, B., Munter, J. and von Riegen, C. (2002) 'UDDI Version 3.0, Published Specification', online at: http://uddi.org/uddi-v3.00-published-20020719.htm [accessed 20 December 2004]

Birman, K. P. (2004) 'Like It or Not, Web Services Are Distributed Objects', Communications of the ACM, Vol. 47 no. 12, 60-62

Burbeck, S. (2000) 'The Tao of e-business services', IBM Corporation, online at: http://www-106.ibm.com/developerworks/webservices/library/ws-tao/ [accessed 23 December 2004]

Chandrasekaran, S., Miller, J. A., Silver, G. S., Arpinar, B. and Sheth, A. P. (2003) 'Performance Analysis and Simulation of Composite Web Services', Electronic Markets, Vol. 13 (2), 120-132

Committee on Innovations in Computing and Communications: Lessons from History, National Research Council (1999) 'Funding a Revolution: Government Support for Computing Research', National Academy Press Washington, online at: http://www.nap.edu/readingroom/books/far/contents.html [accessed 20 December 2004]

Ferstl, O. K. and Sinz, E. J. (1998) 'Grundlagen der Wirtschaftsinformatik', 3rd edition, Oldenbourg Verlag München

Foster, I., Frey, J., Graham, S., Tuecke, S., Czajkowski, K., Ferguson, D., Leymann, F., Nally, M., Storey, T. and Weerawaranna, S. (2004) 'Modeling Stateful Resources with Web Services', Globus Alliance, online at: http://www-106.ibm.com/developerworks/library/ws-resource/ws-modelingresources.html [accessed 25 December 2004]

Geng, X., Huang, Y., Whinston, A. B. (2003) 'Smart marketplaces: a step beyond Web services', Information Systems and e-Business Management, Vol. 1 (1), 15-34

Grässle, P., Baumann, H. and Baumann, P. (2004) 'UML 2.0 projektorientiert', Galileo Press Bonn

Hündling, J. and Weske, M. (2003) 'Web Services: Foundation and Composition', Electronic Markets, Vol. 13 (2), 108-119

Kossmann, D. and Leymann, F. (2004) 'Web Services', Informatik Spektrum, Band 27 Heft 2 (April 2004), 117-128

Ogbuji, U. (1999) 'XML: The future of EDI?', online at: http://uche.ogbuji.net/tech/pubs/xmledi.html [accessed 21 December 2004]

Ogbuji, U. (2002) 'The Past, Present and Future of Web Services, part 1', online at: http://www.mywebservices.org/index.php/article/articleview/663/1/24/ [accessed 21 December 2004]

Ogbuji, U. (2002) 'The Past, Present and Future of Web Services, part 2', online at: http://www.mywebservices.org/index.php/article/articleview/679/1/24/ [accessed 21 December 2004]

Oracle Corporation (2001), 'Oracle9i Application Server Web Services: Technical White Paper', online at: http://www.oracle.com/technology/tech/webservices/webservices_twp.pdf [accessed 2 January 2005]

Reed, A. (2003) 'Object-Oriented Programming and Objectivist Epistemology: Parallels and Implications', The Journal of Ayn Rand Studies 4, no. 2 (Spring 2003), 251-284, online at: http://www.objectivistcenter.org/events/advsem03/ReedOOP.pdf [accessed 27 December 2004]

Reiter, M. (2004-1) 'SOA verbindet zukünftig Firmen', Computer Zeitung 46/2004/hd

Reiter, M. (2004-2) 'Serviceorientierung macht Wartung komplex', Computer Zeitung 46/2004

SAP AG (2004) 'Enterprise Services Architecture, Composite Applications, SAP xApps und mySAP Business Suite Überblick', online at: http://www.sap.com/germany/media/mc_239/50068052.pdf [accessed 9 January 2005]

Siedersleben, J. et al. (2003) 'Softwaretechnik: Praxiswissen für Softwareingenieure', 2nd edition, Hanser Verlag München

W3C Web Services Architecture Working Group (2004-1) 'Web Services Architecture', online at: http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/ [accessed 25 December 2004]

W3C Web Services Architecture Working Group (2004-2) 'Web Services Architecture Usage Scenarios', online at: http://www.w3.org/TR/2004/NOTE-ws-arch-scenarios-20040211/ [accessed 3 January 2005]

Weber, M. (1998) 'Verteilte Systeme', Spektrum Verlag Heidelberg

Wikipedia entry (2004) 'Electronic Data Interchange', online at: http://en.wikipedia.org/wiki/EDI [accessed 21 December 2004]

Zeng, L., Benatallah, B., Lei, H., Ngu, A., Flaxer, D. and Chang, H. (2003) 'Flexible Composition of Enterprise Web Services', Electronic Markets, Vol. 13 (2), 141-152