8 Communication

The Semantic Web would be impossible without the advent of simple and efficient communication networks that allow any user connected to the Web to access any public Semantic Web site without effort, very efficiently and extremely fast (most of the time). The basis for this ease of access is a very simple data format for specifying Semantic Web pages and a very simple communication protocol for their access. Both can be easily implemented on any computing platform. This ease of implementation ensures that everybody can participate independent of their particular computing equipment.

Complementing the Semantic Web, machine-to-machine communication (in contrast to serving up content for human consumption) is addressed by Semantic Web Services. Semantic Web Services are the mechanism for software-to-software communication and coordination (whereas the Semantic Web is for human users). Semantic Web Services are not a disruptive new paradigm, instead, they leverage existing communication knowledge, conventions and technologies and improve on them.

This Section builds the fundamental basis for Semantic Web Services. It discusses the concepts of communication from a principled perspective in Section 8.1. Based on these fundamental concepts, major communication paradigms are listed in Section 8.2. Long-running communication in the context of B2B integration and EAI integration is reviewed in Section 8.3. In Section 8.4 a particular type of communication, Web Services, is emphasized as the focus of the following Chapters. Section 8.6 summarizes this Chapter.

8.1 Communication Concepts

Communication in its basic form allows two or more parties to exchange data that for them has value (at least equivalent to the effort spent on communication). There are basic forms of communication like synchronous or asynchronous communication that provide the fundamental basis. In addition, for senders and receivers to understand each other, data formats have to be agreed upon as well as the possible range of content for those data formats so that the communication partners can understand each other: a date with the value of 01-02-07 can be misunderstood easily if its precise semantics is not captured (one possible interpretation is July 2nd, 2001). Finally, in order for senders and receivers to synchronize the sending and receiving of data, they have to follow specific communication protocols. This Sec-

tion outlines the basics of communication and builds the foundation for the remaining Sections in this Chapter.

8.1.1 Fundamental Types

When parties are communicating they need to establish a communication channel over which the data is communicated between them. All communication channels can be classified into only a few basic classes that define the basic properties of communication. The three basic forms are as follows:

- **Synchronous Connection**. A synchronous communication channel requires all communicating parties to be part of the communication channel concurrently in order to communicate. The parties exchange data between each other. The sending party puts the data on the synchronous channel (or connection) and the receiving party or parties receive the data. In a synchronous connection it is possible that the receiving party starts receiving the initial data while the sending party still sends the remaining data. If one party leaves the synchronous connection, it cannot participate in the communication any more. If the leaving party is one of the last two parties on the connection, the communication finishes (or is disrupted) as the communication requires at least two concurrent communication partners. Elaborate synchronous connections allow parties to send and receive concurrently in both communication directions; less advanced connections can be used only for one direction of data transfer at any given point in time. Examples for synchronous connections are the ancient telephone for humans or the remote procedure call between software systems.
- **Asynchronous Connection**. Asynchronous connections are very different in nature from synchronous connections. An asynchronous connection does not require all communicating parties to be concurrently connected to the connection itself. At any point in time a sending party can put data on the asynchronous connection and at the same or different points in time a receiving party can take data from the connection (as long as data is present and as long as the connection itself is available). The asynchronous connection itself stores the data. In this sense it is stateful and through this mechanism allows the independent presence of sending and receiving parties. If a sending party puts several separate pieces of data on the asynchronous connection, it depends on the particular implementation of the asynchronous channel if the order of the data is preserved or not. If it is not preserved and the receiving party depends on the correct order, the data must contain some information about the order so that the receiving party can reorder the data appropriately independent of the asynchronous communication channel. In the general case data on the asynchronous connection is consumed by the receiving party once it takes the data off the connection. In this sense the reading is removing ("destroying") the data on the channel.
- **Shared Variable**. Communication over shared variables is the third type of connection. A shared variable is accessible by a sending as well as a receiving

party. A sending party can put data into a shared variable any number of times at any point in time. Each time the sending party puts data into the shared variable it overwrites the previous value. A receiving party can take data from a shared variable. When it does so, the data is not consumed; instead, the data remains and other receiving parties can access the data in the shared variable. The receiver can write data to the shared variable, too, of course. In this case, if the receiver does not want any other party to read the shared variable, it can put a "null" on it, i.e., overwriting its value. Putting data into a shared variable and reading a shared variable are asynchronous to each other. It is also not guaranteed that a receiving party sees all values of the shared variable. If the sending party writes very often, it might very well be the case that the receiving party does not read fast enough and misses intermediate values. Since the shared variable allows concurrent access it needs to ensure that the read or the write is atomic to avoid that senders and receivers are interfering while operating on the shared variable.

All specific implementations of communication technology can be reduced to one of the three fundamental types discussed above. For example, communicating through a database is a shared variable communication. Communicating through queues is an asynchronous communication. A remote procedure call is a synchronous connection. Some of the major technologies are introduced later in Section 8.2..

8.1.2 Formats and Protocols (FAP)

A communication channel of either type is minimally required in order for parties to communicate with each other. If the communication itself should be successful, meaning, the communicating parties understand each other and have a constructive communication with a defined outcome, more has to be agreed upon then just the communication channel. There are two major aspects of communication that need to be in place for a meaningful communication: formats and protocols.

Formats refers to the data structure and data content that is communicated. The sender as well as the receiver have to agree on the data structure and content in order to understand each other. Structure refers to the particular data elements and their relationship that is communicated whereas content refers to the values in the data elements. Only if sender and receiver agree on structure and content, can they "make sense" out of each other's data and have a meaningful conversation. This agreement that has to be in place and needs to cover all possible values and structures. As the communication has to work under all allowed combinations, the agreement is quite difficult to achieve in general, as it is practically impossible to enumerate all possible combinations to prove that the communicating parties understand each other for each combination.

Protocols refer to the exchange sequence of the data in their particular formats. A communication in the general sense requires that sender and receiver exchange

several distinct sets of data by sending them to each other. This requires a specific order to ensure that both, sender and receiver understand where they are in the communication and what data has to be exchanged next in the sequence of exchanges. The involved parties only make constructive progress during their communicating if they follow the correct exchange sequence or one of several correct sequences (if several are permissible).

No matter which fundamental type of communication channel is used, the formats as well as protocols have to be agreed upon so that all involved parties can participate in the communication in a meaningful way. Furthermore, if formats are transmitted that cannot be understood, or if protocols are violated, then the communication must be able to recognize this error situation and try to get back to a meaningful state. For example, if formats are not understood, then the receiver must be able to send back a "not understood, please send again" response. Otherwise, if a communication error is not detected, no repair is possible and the communication has ended unsuccessfully.

If the protocol is violated, the violation must be detected, the overall communication must stop and synchronize on a state that all parties agree to as a consistent state from which to continue. This might happen if, for example, certain data messages are lost in the communication and the receiver is waiting for a specific exchange that the sender assumes happened already.

Both formats and protocols play an important role in Web Services as well as Semantic Web Services. Formats are described using Semantic Web languages whereas protocols are defined through various elements that the Semantic Web Services efforts provide.

8.1.3 Separation of Interface and Logic

Formats and protocols have to be implemented as software code in order to make communication over communication channels work. The data formats and data values that are sent over communication channels have to be independent of the software of the sender or the software of the receiver to achieve maximum independence. In any communication setup it is impossible to guarantee, ensure or enforce that both, sender and receiver use the same software from the same vendor for sending and receiving the data. Consequently there needs to be a distinction between the implementation of how to produce the data or consume the data (implementation or logic) and the definition of the data itself (including its possible contents). This distinction follows the well-established separation of interface and implementation in computer science. Later on when Web Services and Semantic Web Services are discussed this distinction becomes a very important aspect.

The same applies to the behavior of communication. The order of formats sent and received by the communicating parties must be described in such a way that all communicating parties can agree to it independently of the software used to implement and enforce the behavior. This means that the definition of behavior must be done in such a way that the behavior can be inferred from the language used to

describe the behavior (instead of examining the code of the software that implements the behavior). In consequence, this allows both, the sender and receiver to agree on the behavior while implementing it in their preferred software technology or with technology of their preferred software vendor. Again, when talking about Web Services and Semantic Web Services this aspect becomes important in the formalisms and languages used.

8.1.4 Communicating Parties

Communication cannot take place without communication partners engaging in the communication by sending and receiving data from each other. In a given communication there is always a sending partner (sender) and at least one receiving partner (receiver). The sender sends out data that the receiver obtains by taking part in the communication. Several receivers are possible in a communication and all of them receive the data sent by the sender.

During a communication the role of sending and receiving can change if the communication is conversational. Once a sender has sent out data, and after the receivers have received the data, one of the receivers can assume the sender role and send out data. This is especially the case in the situation where the communicating parties have a dialog in the sense that formats are sent back and forth in order for both parties to accomplish the goal of the communication. In a multiparty communication it is possible that several parties start sending at the same time as they do not know about each other's state and intent. In such a situation it is important to ensure that either the protocol does not allow such a conflict to happen or that a dynamic mechanism is available at run time that enforces only one sender at a time.

Some communication channels allow only one sender at any given point in time. In this case there can only be a single sender for a given communication and no coordination has to be enforced through the protocol. However, some communication channels allow the concurrent sending of data by several senders. In this case several senders can send data, but for the communication to be meaningful, the participating receivers needs to be able to receive data from different senders concurrently. If a channel allows several concurrent senders it is not necessary to enforce the one-sender-at-a-time policy, of course.

Another dimension opens up when a single communication channel can "host" several independent communications. In this case it is necessary to distinguish the communication not only by communication channel, but also by identifier within one communication channel. In computer science this case can take place when asynchronous technology like queueing technology is used. In this technology it is possible to send messages across a queue that originate from different senders and are addressed to different receivers. In this case each message must either carry a communication identifier to identify the communication or each message carries a receiver identifier so that the respective receiver knows which messages to read.

If the approach is followed that messages carry the identifier of a communication then the notion of an "instance of communication" is important. This can be further formalized by associating senders and receivers (which are instances, too) to communication instances. Going forward we assume this notion. At any given point in time, when senders and receivers communicate, they do this in context of an instance of a communication. Consequently, it is possible that the same senders and receivers open up a separate instance of communication. In addition, the same senders and receivers can be participants in different instances of communications, either concurrently or sequentially.

This notion of communication is independent of the fundamental types of communication as outlined in Section 8.1.1. They have to agree on a fundamental type (by selecting a given communication technology). Of course, the separation of interface and implementation is important, as stated in Section 8.1.3. The relevance here is that no matter in how many communications a given participant (or party) takes part, it has to maintain the separation.

And, furthermore, the communicating parties have to agree on the FAP, as outlined in Section 8.1.2. Each communication in the general case follows a defined FAP. Different communications can follow different FAPs, as agreed upon by the participants. This ensures that every communication over any channel is meaningful for the participants.

A further generalization is possible, although not really used widely. If a FAP is defined (for example as a standard), then in many cases the definition might be sufficient for the communication requirements of several parties. However, sometimes a given FAP might not be sufficient. This could be if data formats are missing that are required for a particular case. The parties, in order to overcome this problem, can either extend or modify the FAP, or they can change the FAP during a communication and switch over to a different FAP. The switch over then enables the set of participants to use as many FAPs in one communication as required to make the communication work and meaningful. While this is a very interesting generalization, it is not usually done.

If two parties are involved in a communication it is called a binary communication. If more than two parties are involved, it is called a multi-party communication. A multi-party communication enables several parties to take place in a communication. So far it was assumed that there is one communication instance and all parties are related. Furthermore, it was assumed that all parties are aware of each other and all concurrently receive data from a sender. However, this is not necessarily always the case. A multi-party communication can actually be achieved by one party being the "communication coordinator" and all other parties engaging in a binary communication with the coordinator. So only the coordinator is aware of all the parties, but each party is only aware of the one coordinator. This requires that the coordinator is part of the communication for its whole length, can receive all data from all senders, and can relay sent data to all parties that are not sending at a given point in time. This also allows having the coordinator engage with different parties using different FAPs.

8.1.5 Mediation

Formats and protocols are agreed upon between the parties of a communication. Once they have agreed upon it, they will follow it precisely as otherwise the communication will most likely not be meaningful and therefore unsuccessful. The communicating parties have no interest in this situation. Therefore, they will do everything necessary to comply with the FAPs.

In the general case, the FAP is determined by the interface that the communicating party can support (see Section 8.1.3). This interface determines what the communicating party can support, and hence this interface allows the selection of one or more FAPs that comply with this interface.

As in the general case, the interface has to be implemented in order to support it at run time. Therefore, the interface is implemented by software. If the internal data processing environment, however, supports different data structures and data content, then there is a mismatch between the interface and the software used to achieve the implementation. Why would this ever be the case? Why would a party not define the interface in such a way that the interface can be implemented easily?

In the world of communication over world-wide networks across company boundaries there are established practices of FAPs (often referred to as B2B protocols). [263] discusses quite a number of those. In order for a given party to easily participate in a communication it is wise to support the FAPs of a given industry. Therefore, the party is probably inclined to solve the discrepancy between the interface it needs to support and its available implementation technology rather then supporting an interface that does not allow it to easily participate in given FAPs.

Bridging the data structures and data content on the interface and the implementation software is the data mediation problem. This problem is well-studied and many attempts are made to structurally overcome it [263].

The same is true for the protocol aspect. The behavior that a given FAP demands might be the same or might be different from the behavior the underlying software for implementing it exposes. In addition to data mediation as described above the concept of protocol mediation is required. [264] describes the protocol mediation problem in detail. Only if data and protocol mediation are both supported it is possible to map the interface to the implementation within a given party of a communication.

Not all parties can support at their interface all the FAPs that they need to in order to participate properly in the various communications. In this case it is necessary to move the mediation (data and protocol) outside the interface. So instead of mediating the difference between the interface and implementation within a party, the mediation is done outside the interface between parties. This makes the concept of a “middle man” necessary. The middle man is a party to the communication for specifically mediating between communicating parties. The middle man established a communication channel with all participating parties and uses different FAPs for that. The FAPs used are those that the parties can support. The middle

man itself mediates between the FAPs as it passes along the data from the sender to the receivers.

The benefit of this approach is that parties can participate in communications that they could not support directly. Of course, the middle man has to be available and able to mediate appropriately.

The most flexible party to a communication is the one that has a declarative way of mediation within its boundaries between its interface and its implementation. If the mediation is declarative, existing mediations can be changed or new mediations can be added. If the change or addition of mediation is fast and flexible, the party can define additional interfaces as required by FAPs as it can build the mediation to its implementation easily. It is therefore assumed for simplicity that this is the approach going forward in the remaining Chapters about processes and the Semantic Web. If a given party cannot implement mediation this way, the way out is the middle man.

8.1.6 Non-functional Aspects

Ideally, communication is secure, reliable, recoverable, fast, and has many other "nice" properties that make it convenient for the communicating parties. Properties like security, reliability, recoverability, performance, and others are called non-functional communication properties (as they are related to the communication system behavior, not the semantics of a communication). Different implementations of communication channels have various support for non-functional communication properties. Depending on the particular needs of the communicating parties they have to select the most suitable mechanism.

A few properties are discussed in the following. The list is not complete and only highlights the most important aspects:

- **Security**. Security has many different facets. The most relevant are that data communicated should not be visible to any party not involved in the communication. This is usually achieved by either encrypting the data packets themselves that are communicated or encrypting the whole communication channel instead of the individual packets. Furthermore, no other attack should be possible like taking data packets off the communication channel, or introducing additional ones in order to cause disruption in the protocol. Another aspect is authentication and authorization of the parties that want to join a communication channel. Not all parties should be easily able to join a communication just like that. The originator of the communication should be able to restrict access as necessary and have parties authenticate themselves in order to allow the proper authorization.
- **Reliability**. Reliability is important in the presence of failures. In case of a failure it must be clear what status the communication is in and which of the last data transmissions succeeded successfully and which did not. This allows after a failure to continue the communication from a consistent state forward.

- **Transactionality**. Transactional behavior of communication is important in the presence of fatal errors. If a server goes down and has to be restarted, if a network fails or a communication software stops working, then it is important that the communication can be recovered to its last consistent state. This is important as it allows the communicating parties to continue the communication without having to execute any recovery strategy itself, let alone compensating actions that would modify already achieved states.
- **Throughput**. If the data sent is of high volume or if many communications are ongoing in parallel then the communication channel might become a bottleneck in the sense that it cannot support all communication as fast as in a low load situation. In this case the communication system degrades in terms of performance. Throughput is important and the ideal situation is that degradation happens only under very high load. Furthermore, it should be gradual, not sudden.
- **Performance**. Performance is related to the speed of data transmission. In many situations speed is of high importance, for example, when communication takes place over synchronous connections that require a fast response. In other situations performance is not as important as the communicating parties do not have to operate within the bounds of specific time lines.
- **Availability**. We are used to the immediate and constant availability of the phone system. Whenever we want to make a call we expect the phone be available and ready. Connections to the Internet are also expected to be "always-on". In this sense every party is expecting to be able to engage in a communication whenever they need to. High availability of the communication channel is important. Of course, this does not mean that all communicating parties are always available; a phone call might not be taken by the intended recipient.

This discussion of non-functional communication properties concludes this Section. The fundamental communication concepts have been introduced that form the conceptual basis of communication as related to Semantic Web Services. In the next Section specific communication paradigms that are based on the fundamental types are introduced.

8.2 Communication Paradigms

Based on the fundamental types of communication, namely shared variables, synchronous and asynchronous communication, different specific communication paradigms were developed over time. Leaving the postal mail approach of storing data on a storage medium like DVDs and sending them by postal mail aside (i.e. "physical communication"), the most important current communication paradigms (like client/server or queueing) that are en vogue are discussed throughout this Section.

For each paradigm, the FAP as well as the number of communicating parties are discussed as well as to which basic type it belongs.

8.2.1 Client/Server (C/S)

The client/server communication paradigm is one of the oldest paradigms and is part of the synchronous connection type. This approach distinguishes a provider of functionality, called server, from the consumer of functionality, called client. Clients and the server can be on the same computer or they can be on different computers. In the latter case, communication is established over a network (be it a local network or a wide-area network).

The FAPs for this paradigm are determined by the server. The server defines and specifies the possible invocations a client can make and their order, it defines the data structures as well as the data content. And it defines the behavior, too. The client has no ability to influence any of these definitions, it can only use whatever the server provides and allows at any given point in time.

A server can serve many clients. The exact number depends the server's capacity and the size of the computation its clients request. The clients do not know about each other, so it is not possible to have a multi-party communication; instead, all communication is binary between a client and the server.

8.2.2 Queueing

The queueing paradigm became popular in recent times with the advent of explicit queueing systems as a separate architecture and technology component or implicit database or application server functionality. Queueing is of the asynchronous connection type as it decouples the communicating parties.

Queueing is in principle a one-way communication mechanism where a sender submits messages to a particular queue using an enqueue operation. The messages will be stored in the queue, generally in the order of receipt. The receiver takes messages from the queue using a dequeue operation. In its simplest form, queues maintain the message order and operate under the first-in-first-out (FIFO) mode.

In this situation the determination of the FAP becomes an interesting topic as it depends on the viewpoint of who decides on the FAP. In the majority of cases the queueing paradigm takes the form of an asynchronous client/server model where the receivers determine the FAP and the senders have to comply. However, this is not necessarily the only possible viewpoint. An alternative viewpoint is that the sender is an information source and publishes its message to a queue and is not really interested in which receivers pick up the message content. In this viewpoint fundamentally the receiver is the one that is "interested" in the message and has to comply to the senders FAP accordingly.

Over a queue many senders can communicate with many receivers. In order to establish a two-way communication two queues can be put in place, each queue for one direction of communication. Alternatively one queue can be used and the senders and receivers both put messages and read message from the same queue. If it is important to know in this case what messages are response messages, they have to be marked accordingly either through typing the message or an attribute in the con-

tents of the message. In the case of two queues, one can be marked as the request queue and one as the response queue. Still, in any case one sender and one receiver communicate with each other. The reason is that on dequeue the message gets removed from a queue, meaning, the only one receiver can receive a single message.

However, more advanced queueing systems allow more than one receiver to receive the same message. The queueing system in this case ensures that all receivers receive a copy of the same message. This is accomplished by receivers declaring interest in specific messages and the queueing system notifying all receivers about the advent of those. These systems are called publish/subscribe systems. Receivers (subscribers) then will receive those messages from the senders (publishers) and can proceed with whatever processing they need to do.

8.2.3 Peer-to-Peer (P2P)

The peer-to-peer paradigm is very similar to the client/server paradigm and is of the type synchronous connection. The major difference is that the server in the client/server paradigm is usually stationary and in a central location to which all clients connect to. In the peer-to-peer paradigm this is not the case. Two communicating parties (in this case called peers) communicate directly with each other and establish a connection directly, not going through some central location like a stationary server at all. Each party can become server whenever it wants to and can become client whenever necessary. Fundamentally, every peer is a server or a client at any point in time. In this case any two pairs of parties that know about each other can establish a direct connection at any time, establishing peer-to-peer links.

In the peer-to-peer paradigm, as each party can become a peer at any time, all have to agree on the FAP in order to be able to establish direct connections. More elaborate peer-to-peer protocols allow peers to communication with each other that do not have a direct communication link. In this case other peers act as relay station forwarding the communication. Two peers therefore communicate directly, whereby the actual data transport is over other peers as intermediaries (invisible to the communication channel itself).

It is possible that one peer communicates concurrently with several other peers. In addition, several peers can communicate with each other at the same time. So a true multi-party communication can be established where the peers know about each other.

8.2.4 Blackboard

The blackboard paradigm is an approach to further decouple senders and receivers, even more then a queueing system allows to do. A blackboard architecture provides a space where data can be posted, changed or removed by a sender. There is

no specific guarantee about when data is made available and how long it will reside there. Receivers can read data at any point in time and as often as they want or need to. The basic protocol of how to post data and how to read data is determined by the blackboard. However, the FAP between senders and receivers is not determined by the blackboard, that remains within the control of senders and receivers. Still, the FAP that senders and receivers use between them and the FAP of the blackbaord has to match within the constraints of the blackboard. For example, if the FAP requires versions, but the blackboard does not support versions, the protocols are incompatible.

There can be any number of senders and receivers, and there can be many receivers participate in the same communication as the blackbord does not restrict data to be accessible only to specific senders.

8.2.5 Web Services

Web Services is a new communication paradigm that is centered around the public Internet as communication transport layer. Web Services have three aspects to it. First, the interface of the communicating parties is described using an interface definition language (called the Web Service Definition Language (WSDL)). This formal language allows the definition of the messages a communicating party sends as well as receives. Message sending and receiving is based on operations that have messages as input and output parameters. This approach differs from B2B protocols where messages are sent and received without the notion of operations.

Second, an explicit transmission protocol is defined, called SOAP (initially standing for the Simple Object Access Protocol, however this has since been dropped). This protocol is abstract in the sense that it defines how a message as defined in WSDL is structured when it is sent using a concrete transmission protocol. A separate binding is defined to bind the SOAP protocol to a real transport. Bindings exist (amongst others) for HTTP as well as MIME. This is interesting as HTTP is a synchronous protocol based on the synchronous communication type whereas MIME is an asynchronous protocol based on the asynchronous base type. The interesting aspect is that the interface definition is independent of the actual communication mechanism.

Third, a publication mechanism is defined that allows communicating parties, if they so wish, to publish their interface definitions in public or private directories for others to look up. This supports the detection of communicating parties based on their defined interfaces.

The FAP are in part predefined by the notion of operations with input and output parameters. However, the sequence or order of operations that has to be called is undefined and open for the communicating parties to agree upon. The data structure and content is free for the participating parties to decide, however, the specification language is XML Schema. With XML Schema the communicating parties

can agree on structure and content to the extent XML Schema supports the definition.

Web Services are a bilateral communication mechanism that allows two parties to communicate with each other. A multi-party communication is not supported by the current Web Service standards or technology.

8.2.6 Representational State Transfer (REST)

REST (representational state transfer) [265] is a particular style of enabling communication based on the principle that all data as well as operations on data are enabled using strictly static URLs based on the HTTP protocol. The fundamental approach is to see data and operations as identifiable resources and the identification mechanism is URLs. As such, resources like data or operations are identified by URLs. Accessing a particular car (identified for example as 45671) from a car selling web site (for example, www.sellyourcar.com) could be www.sellyourcar.com/car/45671. The response to issuing the URL would be the data format and data content representing the car identified with 45671. Searching a car could be www.sellyourcar.com/findCarForm. This would provide the client to obtain the data format with the search criteria to be filled for searching a particular car.

REST implements a "classical" client/server model where the client requests action from a server through particular structured URLs. A server provides a response to clients if a well-formed URL is transmitted to it. The communication is synchronous and over the HTTP protocol. The static URLs ensure that the data or functionality can be accessed with the same URL at any point in time.

The form of communication is binary as only two parties can participate in one communication. However, a server can provide responses to several clients, of course. The FAPs are not explicit, but implicit (analogous to the client/server model). In this communication style there is no explicit definition of the data formats, permissible values or the protocols as interface as the interface is not explicitly defined. Instead, all aspects, including the URL structure, are defined by the server in implicit form (as opposed to a WSDL definition). This follows closely "normal" HTML page requests and responses.

8.2.7 Agents

Agents is a concept stemming from the area of Artificial Intelligence (AI). Agents are autonomous entities that form a perception of the world around them and that can communicate with other agents. The communication allows agents to achieve their task by asking other agents to contribute. From a communication viewpoint agents do not only communicate data, but explicitly ask other agents to perform specific tasks. The asked agent can execute the task, delegate the task to another agent or refuse to engage in executing the task (effectively rejecting it). In this sense there is an explicit notion of acceptance as well as refusal of tasks.

An agent can communicate with any number of other agents. However, all agent communication is bilateral in the sense that each agent communicates with one or more other agents directly, never in a multi-interaction way.

The formats are not predefined by an agent protocol. Agents can agree on the formats and data content they want to use. They even can engage in communication without having agreed upfront on the specific formats as an agent can always respond with the "I don't understand" message back to the message originator in order to indicate that the communication will not be possible due to data format or data content misunderstanding. However, in order for every agent to exchange messages with any other agent, a basic protocol for at least exchanging messages needs to be in place. Otherwise any communication is impossible. Also, minimally the "I don't understand" message needs to be agreed upon upfront, too, for agents to be able to tell each other that they could not understand. Otherwise the response message could not be interpreted either, making any communication impossible.

8.2.8 Tuple Spaces

Tuple spaces are like blackboards where communicating parties can add tuples into a space that can be read by other parties. The data structure is predefined as tuples and all communicating parties have to follow this structure. The tuple space itself is a space that exists on its own without communicating parties to be connected to it. Tuple spaces are therefore following the shared variable basic principle. Access to the tuple space is concurrent, however, each individual tuple is accessed atomically for consistency reasons.

The formats are open for the parties to determine or to define. The basic protocol of tuple management is determined by the tuple space. Any protocol beyond that, i.e., the number and order of specific tuples written is solely in the discretion of the communicating parties.

Any number of parties can communicate with each other at the same time using tuple spaces. In this approach bi-lateral as well as multi-party communication is supported. Interestingly enough, from the viewpoint of an individual party, it is not clear at all if there is a bilateral or multi-party communication. It can be the case that one party writes tuples that are never picked up by any other party. In this sense tuple spaces (like blackboards) allow a one-party communication; this is really an oxymoron, unless the storing and reading of tuples by the same party is considered communication with itself.

8.2.9 Co-location

Co-location is a communication paradigm that allows parties to communicate without crossing remote networks for the purpose of the communication. Co-location is based on the principle that the communicating parties share their communication code so that a party communicating with another one really does a local invocation

instead of a remote invocation. This allows the existence of a structured communication without incurring the network overhead for sending data between the parties.

However, one must ask the question how data actually ever gets transferred between the two parties as in the end of the day they are in separate environments? The basic assumption in the co-location paradigm is that the communication code implements database updates as side effects. If this is the case, if one party calls another party's communication code, that updates the database of that party. So in reality the communication on a communication protocol level is local, however, the remote access part is "pushed down" to the database access layer.

This paradigm is relevant especially within organizations where remote database connectivity is possible. In such an environment all communication is limited to a one-hop database invocation without additional remote invocations across a network. This reduces the remote invocations while keeping the database connectivity constant.

Any number of parties can participate in such a co-location as each party uses the communication code of every other party it communicates with. In addition, the formats and protocols can be freely agreed upon as the shared communication code can be invoked as needed by the sender.

8.2.10 Summary

Many communication paradigms exist, each having its very own properties. In a given communication situation, some might be more appropriate then others. However, at the end of the day, all allow the transmission of data from a sender to a receiver.

8.3 Long-Running Communication

Communication between two parties is not restricted to only a single individual exchange of data. In many cases several exchanges take place, from sender to receiver and back. These exchanges usually take place one after another. If this communication is following a protocol for the whole duration of the communication and is about the same business process or about the same business objective (like for example clarifying the insurance coverage of a patient) then it is considered a long-running communication.

The term long-running comes from the fact that the individual communications are related to each other and not arbitrary. Furthermore, if one individual communication fails, then only the failed one needs to be repeated or corrected, not the whole communication from the beginning up to this point. In a long-running communication each individual successful communication is regarded as a consistent state. So if the last individual communication fails, and as the last consistent state

is persisted, it can be retrieved and taken as the restart point for continuing the long-running communication.

Long-running communication is widely used, especially in context of inter-organization communication in form of B2B protocols as well as in intra-organization communication in form of Enterprise Application Integration (EAI), also called Application-to-Application (A2A) integration. In the following each is discussed separately in turn.

8.3.1 Business-to-Business (B2B) Protocols

B2B communication takes place when company boundaries are crossed while data is passed back and forth between the communicating parties. A typical situation in the supply chain industry is when a buyer sends a purchase order to a seller and the seller responds with an acknowledgment that the order will be fulfilled. Later on the seller would send an invoice for the goods shipped, expecting a payment from the buyer. Another situation in the healthcare domain would be the communication about clinical tests between different healthcare providers.

B2B communication is about sending and receiving meaningful business data that allow businesses to act upon or to react to. The formats have to be agreed upon so that the communicating parties can understand each other. The same is true for protocols as the communicating parties have to comply to the protocols in order to send or to wait for a message at the precisely correct time.

As outlined in [263] there are several standards organizations maintaining and further extending standards that define the formats and protocols. Examples are RosettaNet, HIPPAA, EDI, just to name a few.

The challenges in setting up proper B2B communication are manifold. The main challenge is the semantically correct interpretation of the various data elements in the messages that are exchanged. As the data sent across is in general coming from various back end application systems, it reflects many different data models. As the formats themselves represent a data model the correct interpretation depends not only on the structure of the messages but also their contents.

Another challenge is to design the long-running B2B interactions and to ensure run-time compliance to the agreed long-running protocol. Depending on the complexity of the long-running process it is possible that many correct executions of the processes exist. Plus, many different error situations can happen that require compensation in order to get back on track to a correct execution.

Since important business data are communicated other aspects are of importence. Security is a big and important item, and so is reliability. Reliability refers to the guarantee that a message was not only passed, but also received. Nirvana in this case would be an exactly-once transmission so that both, sender and receiver are guaranteed that each message was transmitted exactly once. This ensures that every message is accounted for and not lost. In addition, neither sender nor receiver have to worry about error detection, handling and recovery on a message passing level.

In addition to the reliability of a single data exchange, the consistency of the overall long-running process is very essential as both communicating parties rely on the consistency in order to do successful business with each other. If one transmission in a process fails (and keeps failing) then it might be the time when the overall process needs to be abandoned. As it is a long-running process many states were committed along the way clearly requiring compensation to undo already achieved work.

At run time, speed and throughput is of concern as well as security. However, these are non-functional properties not relevant to the discussion in context of semantics.

8.3.2 Application-to-Application (A2A) Protocols

Application-to-Application integration is also referred to as Enterprise Application Integration (EAI). A2A protocols are very similar to B2B protocols in that data is passed back and forth. In contrast to B2B protocols, the endpoints are applications within an enterprise, not systems across a network between companies.

The issues and problems are the very same as in B2B protocols, including security. This might be a surprising statement as in general applications are invoked over their interfaces. However, applications may still provide purchase orders or wait for acknowledgements at their interfaces. In this sense there may be a long-running process implemented inside that requires compliance. Transformation is necessary as the data model inside the application might be different from that of the B2B protocol or other applications that are integrated. As enterprises, especially larger ones have different physical locations communication between applications is possibly going over the Internet, so security becomes an important aspect. In this sense, A2A and B2B integration are very similar.

Some of the problems can be solved a lot easier due to the fact that the integration is within an enterprise. For example, the exactly-once semantics in data transmission can be achieved using transactional communication systems like transactional RPC or transactional queueing systems. Also, the endpoint of the integration are applications within the same enterprise, so supposedly the communication between the governing groups should be a lot easier then across company boundaries.

From a semantics viewpoint there is no difference between B2B and A2A protocols at all. This is the reason why not distinction is made in the remainder of the book.

8.4 Web Services

Web Services are the current silver bullet for (remote) communication in context of the Web as well as within enterprises or governmental organizations. Aside from the fact that Web Services is a relatively new development (only a few years old)

the initial charm lay within its perceived simplicity. To keep things simple, there was the notion of an interface, a transport protocol and a mechanism to register interfaces. And that was all there was to it initially. Everybody liked this simplicity despite the fact that everybody knew from the very beginning that additional features and functionality are needed like security, transactions, policy, processes, and so on.

The initial run-time model was that of a client/server model. A server defines services by means of interfaces in a XML-based language call WSDL (Web Service Description Language). These interfaces are made available in a registry that clients can lookup. The initial effort was UDDI (Universal Description, Discovery and Integration). Clients wanting to invoke services, could lookup UDDI repositories, retrieve interfaces and (having this knowledge) invoke those interfaces dynamically. Of course, there was no security, no reliability, no policy or any additional functionality.

Over time, the realization set in that these features were necessary leading to the development of WS-* (spoken "Web Service Star" or half-jokingly "Web Service Death Star") which refers to a (relatively large) set of standards and proposals that together lead to an acceptable set of technologies for communication. This set of standards is in the meanwhile so complex that efforts are underway to simplify this (leading to a split and proliferation of multiple activities).

All these efforts have as common denominator a few technical concepts and principles that will be discussed in the next Chapter of the book in more detail. The next Chapter discusses also the most important standards from this set in more detail to outline the overall complexity that is to be mastered in order to implement useful Web Services.

One of the efforts that wants to enhance the state-of-the-art are Semantic Web Services. This is a very important activity as it strives to incorporate semantics at the level of description models, mechanisms and languages. A separate Chapter is devoted to those to introduce current efforts in this space and yet another Chapter of the book looks at standards activities for Semantic Web Services.

8.5 Clinical Use Case

The clinical use case (see Chapter 2 and Chapter 13) is fundamentally a distributed application system with many concurrent activities. Doctors retrieve and enter data, clerks schedule appointments, test results are forwarded or inferencing takes place to derive information.

All the various subsystems of the clinical use case implementation are independent in their data management and they communicate with each other in order to exchange data. Chapter 2 outlines the basic use case as well as its requirements. Chapter 13 outlines the complete use case in detail and shows how semantic technologies are used to define the functionality.

Here we discuss how communication technologies are used to support the clinical use case implementation. First of all, as the use case takes place in a clinical environment, security and reliability play very big roles. Security ensures that the patient specific data is only accessible to those authorized to see it. For the communication technology this means that only those communication technologies and their implementations can be used that support security.

The second big requirement in clinical environments is reliability. It is essential that any data that is collected manually is transmitted and stored in such a way that it cannot get lost. Transactional communication technologies ensure this requirement. Also, any derived information as well as information gathered through test results have to be reliably communicated so that it is ensured that data is not lost during communication, be it a one-step communication between two systems or a multi-step (long-running) communication between several systems.

For the latter case, when several systems have to cooperate, long-running communication is the best approach to ensure that the cooperation finishes and does not stop undiscovered at a partial state. Long-running communication maintains the state during the cooperation and can restart or continue after a failure from the last consistent state.

The next few chapters will introduce Semantic Web Service technology. Services will be discovered, put together into an orchestration as well as invoked. This technology is in principle independent of the underlying communication technologies as they can use any available one. For the clinical use case this means that all remote invocations of services is based on secure and reliable communication. Web Service orchestration will be executed on long-running communication so that state is not lost.

Some of the systems, like for example the system that discovers services, does not have to be reliable. In these special cases, when a discovery fails, it can be re-run in order to obtain a good result. In more general terms this means that idempotent functionality does not have to be reliable as it can be re-executed without loss of information.

When it comes to the selection of a sepecific technology implementation no general advice can be given as in every real life implementations the choice is determined mostly by the already existing infrastructure. For example, if a hospital has already a transactional queueing system from a specific vendor, than this is the one that needs to be deployed in Semantic Web Service implementations, too, for pragmatic and financial reasons. Also, if some of the remote connections are based on B2B protocols using healthcare document or message standards than this is the one to be chosen. However, independent of the specific technology that is available, the aspects of security and reliability are essential and must be achieved.

The use case in Chapter 13 focusses on the functional definition of the clinical use case and does not further go into the specific communication technologies any more.

8.6 Summary

In summary, communication is a highly utilized and very well researched area of computer science. Communication between computers is at an all time high as the networks around the globe get more tightly integrated every day. The latest development, Web Services, made communication a lot easier (initially) leading to a lot of development of remote services.

Various communication styles and mechanisms exist that address different functionality. However, common to all is the lack of semantic description languages and concepts, as pointed out in this Chapter. The following Chapters focus first on Web Services and later on Semantic Web Services that in the end strive to overcome the semantic description problem of dynamic behavior expressed as long-running processes.