



TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology
Department of Computer Engineering

Denis Konstantinov 111615 IASM

Embedded service oriented microcontroller architecture.

Extensible client-server communication architecture for small devices

Master thesis

Supervisor: Peeter Elervee
Associate Professor at the Department of Computer Engineering / Ph.D., Dipl.Eng.

Tallinn 2013

Author's Declaration

This work is composed by myself independently. All other authors' works, essential states from literary sources and facts from other origins, which were used during the composition of this work, are referenced.

Signature of candidate:

Date:

Acronyms

- API** Application programming interface. 8
- CRUD** Create, read, update and delete. 16, 31
- FTP** File Transfer Protocol. 10, 11
- HMI** Human Machine Interaction. 6
- HTTP** Hypertext Transfer Protocol. 5, 11, 13, 14
- IPC** Inter-process communication. 18, 22
- JMS** Java Message Service. 13
- JSON** JavaScript Object Notation. 23
- REST** Representational state transfer. 14
- RPC** Remote procedure call. 9, 31
- SMTP** Simple Mail Transfer Protocol. 11, 13
- SOA** Service-oriented architecture. 5, 8, 9, 12, 14, 26
- SOAP** Simple Object Access Protocol. 5, 11–14
- TCP** Transmission Control Protocol. 13
- UDDI** Universal Description, Discovery and Integration. 10–13
- URI** Uniform Resource Identifier. 15
- WSDL** Web Services Description Language. 5, 11–14
- XML** Extensible Markup Language. 5, 11, 12, 14, 23
- YAML** YAML Ain't Markup Language. 23

Annotation

Current work introduces conceptual approaches for implementing an extensible service oriented client-server application on a small microcontroller. This is a general-purpose transport and hardware independent embedded server that uses remote procedure calls as primary communication protocol. This server looks like remote service that could provide defined functions to the client. ...

Annotatsioon

Annotatsioon eesti keeles

Contents

Acronyms	2
1 Introduction	5
1.1 Impact	6
1.2 The goal	6
1.3 Outline	6
2 Background	8
2.1 Service oriented architecture	8
2.2 Web Services architecture	9
2.2.1 Web Services Model	9
2.2.2 Web Services Protocol Stack	10
2.2.3 Service Description and Service Contract	12
2.2.4 Advantages and disadvantages of WS-* standards	13
2.3 REST and RESTful services	14
2.3.1 What is the REST?	14
2.3.2 Key principles of REST	15
2.3.3 Implementation constraints	17
2.3.4 Summary	18
2.4 Remote procedure calls and *-RPC	18
2.4.1 RPC in details	18
2.4.2 RPC summary	22
2.5 Data serialization	22
2.5.1 Serialization technologies	22
2.5.2 XML vs JSON	23
2.5.3 Is there a right serialization format?	25
2.6 SOA and Embedded Systems	26
2.6.1 Devices Profile for Web Services	26
2.6.2 Constrained Application Protocol and Constrained RESTful Environments	27
2.6.3 Performance issues	27
2.6.4 Authentication and Authorization in embedded systems	28
2.7 Final target system requirements	31
3 System architecture	33
3.1 Introduction	33
3.2 Server architecture	33
3.3 Client architecture	33
4 Implementation	33
4.1 System architecture	33
4.1.1 Device connection scheme	33
4.2 Implementation of the embedded server	35
4.3 General purpose service library implementation	36
4.4 Implementation of android client	37
5 Conclusions	38
5.1 Results	38
5.2 Future work	38
A Examples of service contracts	38

1 Introduction

Computers are very essential in our life. Computer is an electronic device that is used in almost every field. It is very accurate, fast and can accomplish many tasks easily. In early days computers were only used by the government and army to solve different high computational tasks. After invention of low-cost microprocessors, computers became available to every person. Nowadays there are billions of personal computers and they are almost at every home.

Present day computers may be divided into two groups: very big and very small systems. In one group are mainly servers and server farms, and in the other are mainly embedded systems. The gap between these groups becomes more wider, because of the availability of new small and low-power devices, which computational power raises constantly. Lot of people prefer now to buy a tiny laptop instead of traditional workstations with a monitor and computer case under the table. There is also a more smaller group of devices, that are implemented for a particular purpose - embedded computers. Every home has several examples of embedded computers. Any appliance that has a digital clock, for instance, has a small embedded microcontroller that performs no other task than to display the clock. Modern cars have embedded computers onboard that control such things as ignition timing and anti-lock brakes using input from a number of different sensors.

Today, there is very little or no communication between embedded devices and large servers in the web. The problem is not only in the communication infrastructure, because the current communication technologies are able to provide different wired and/or wireless connections. The problem is how we design and implement embedded systems. While we try to keep big systems as open as possible (since it is their primary role), we tend to seclude and isolate embedded systems without providing easy ways to add a custom interface to them. Embedded systems are still mainly seen as vendor-specific and task-oriented products, and not as components that can be easily manipulated and reused.

If all classes of devices could speak the same language, they could talk directly to each other in ways natural to the application without artificial technical barriers. This would allow easily creating seamless applications that aggregate the capabilities of all the electronics. The interoperation adds value to all the devices.

This idea comes from conception of Internet of Things (IoT). The Internet of Things is the network of physical objects that can communicate to each other using Internet and embedded technologies. This connections compose an complex system where each member can send information about its state and acquire data about other parties without any intervention of human being. For example sensors at your home could communicate to heater and ventilation system and control temperature and humidity or your alarm can tell all other devices that you are going to wake up soon. This technologies could help to track and count everything and improve the quality of our lives by removing the unnecessary waste and additional cost.

The Internet of Things is quite popular topic of research nowadays. It possibly can change the world like the Internet did. Many companies and universities are trying to find and invent reasonable technologies for implementing this approach.

One of the methods how this communication can be performed is the concept of remote services and service oriented architecture (SOA). World Wide Web Consortium (W3C) defines a "Web service" as:

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 WSDL¹). Other systems interact with the Web service in a manner prescribed by its description using SOAP²-messages, typically conveyed using HTTP³ with an XML⁴ serialization in conjunction with other Web-related standards.

Services are unassociated, loosely coupled units of functionality. Not only large server system are capable of providing this functionality. Services can also be applicable in the resource-constrained embedded devices.

This work would introduce the concepts how SOA⁵ can be in the context of embedded systems. This contains some research of already available technologies for machine-to-machine communications and the implementation of small system prototype, which contains two conneted devices and uses service approach.

¹Web Services Description Language

²Simple Object Access Protocol

³Hypertext Transfer Protocol

⁴Extensible Markup Language

⁵Service-oriented architecture

1.1 Impact

The impact of the research in this thesis has been started during the accomplishment of internship at the university. I was worked for some company and my task was to develop HMI⁶ interface to some embedded system. We were using wireless communication between the control unit and the machine it was controlling. Control unit was a smartphone that was sending commands through Bluetooth protocol. On the other side there was a coffee machine that was receiving and executing that commands.

At the same time i was studying how large enterprise systems communicate to each other. I was reading about web services and related technologies. Then was born an idea that there could also be a "small" device network, where devices communicate to each other.

This was a research project and developed prototype could potentially become a real product. In that case it needs to be connected to existing infrastructure. Coffee machine could provide different remote services: remote coffee product ordering, coffee machine maintenance and acquisition of statistical data, remote payment. This could look like traditional coffee automatic machines at the streets that accept cash, but with a remote wireless control.

I stated to mine the information about different control possibilities. This is how this research became a topic of my master thesis.

1.2 The goal

The purpose of this work is to create a prototype system which has remote service capabilities and make a research of available machine-to-machine communication possibilities. This should be an universal and platform independent architecture, which can be easily ported to any suitable hardware and connected into existing infrastructure.

Already existing hardware are two STM32f103xx family microcontrollers which have 20 and 64 Kbytes of RAM, 128 and 512 Kbytes of flash memory. They are also equipped with UART communication and whole communication need to work using serial line. Remote server with service capabilities should work on that hardware. General requirement for the hardware is low-cost microcontroller with some connectivity, that does not make the already existing system more expensive and in the same time fulfil all required functionality.

Embedded server will be connected to a target device, which is actually a coffee machine, and handle requests from clients by executing various functions on target device. Server should provide a functional interface to the client and know about available functions inside coffee machine. That interface need to be verbose and easy to connect.

Client will be executed on mobile phone and will communicate with remote server using Bluetooth wireless technology. It might be any mobile platform, but the organization decided to try Google Android smartphones first. Android device need to have running program with a graphical user interface, which is able connect to remote service and accomplish functional needs.

Coffee machine application is only the example of such architecture. We need a real world problem and device to show all capabilities of such system. Controlled device and the client application could vary. It might be a remote light control at home or any data acquisition and control system at the production plant. During this work such universal and extensible system will be built.

1.3 Outline

TODO here will be outline. It will be finished when all other section will be ready

First section will introduce available technologies of implementing machine-to-machine communications. The main research is about the Internet technologies and concept of remote web services, this is because the Internet is already an interconnected network with lots of machines are doing distributed computing and interacting with each other. Lots of problems are already solved there and there are available different technologies and tools. Although, all these already implemented features are not limited only with the Internet and related technologies like TCP/IP and HTTP. Some essential features may be extracted from there and ported to resource-constrained devices. This section will also cover some connection and interaction possibilities that embedded systems have.

Implementation section covers embedded system prototype, that uses concepts from different machine-to-machine communication technologies. It contains description of architecture and software and hardware was used. TODO summary here

⁶Human Machine Interaction

Then i will write about how all this technologies could be ported to a small device. Next goes the implementation of a small remote service. There are described implementation details of a server and client library.

2 Background

Internet technology is the environment in which billions of people and trillions of devices are interconnected in various ways. As part of this evolution, Internet becomes the basic carrier for interconnecting electronic devices – used in industrial automation, automotive electronics, telecommunications equipment, building controls, home automation, medical instrumentation, etc. – mostly in the same way as the Internet came to the desktops before. More and more devices getting connected to World Wide Web. Variety of factors have influenced this evolution [1]:

- The availability of affordable, high-performance, low-power electronic components for the consumer devices. Improved technology can assist building advanced functionality into embedded devices and enabling new ways of coupling between them.
- Even low cost embedded devices have some wired or wireless interface to local area networks of the Ethernet type. TCP/IP family protocols are becoming the standard vehicle for exchanging information between networked devices.
- The emergence of platform independent data interchange mechanisms based on Extensible Markup Language (XML) data formatting gives lots of opportunities for developing high-level data interchange and communication standards at the device level.
- The paradigm of Web Services helps to connect various independent applications using lightweight communications. Clients that are connected to the service and the service itself may be written using different programming languages and be executed on different platforms.
- Presence of Internet allows existing of small embedded controllers and large production servers in the same network, with a possibility to change information.

The integration of different classes of devices, which employ different networking technologies, is still an open research area. One of the possible solutions is the use of SOA software architecture design pattern.

2.1 Service oriented architecture

Service-oriented computing is a computing paradigm that uses services as basic building blocks for application development. [2]

The purpose of **SOA** is to allow easy cooperation of a large number of computers that are connected over a network. Every computer can run one or more services, each of them implements one separate action. This may be a simple business task. Clients can make calls and receive required data or post some event messages.

Services are self-describing and open components. There is a service interface, that is based on the exchange of messages and documents using standard formats. Interface internals (operating system, hardware platform, programming language) are hidden from client. Client uses only a service specification scheme, also called contract. Consumers can get required piece of functionality by mapping problem solution steps to a service calls. This scheme provides quick access and easy integration of software components.

Service architecture have been successfully adopted in business environments. Different information systems, that were created inside companies for automation of business processes, are now turned into services which may easily interact with each other. For example, Estonian government uses services to transmit data between information systems of different departments. There are also some free services available. Some Internet search companies like Google, Bing, Yandex provide lots of alternatives how to retrieve data without using regular browser(search , geolocation and maps, spell check API⁷s)

There are available many technologies which can be used to implement **SOA** [3]:

- Web Services
- SOAP Simple Object Access Protocol, is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks.

⁷Application programming interface

- RPC Remote procedure call is an inter-process communication that allows a computer program to cause a subroutine or procedure to execute in another address space (commonly on another computer on a shared network) without the programmer explicitly coding the details for this remote interaction.
- REST Representational state transfer is a style of software architecture for distributed systems such as the World Wide Web. REST has emerged as a predominant web API design model.
- DCOM Distributed Component Object Model is a proprietary Microsoft technology for communication among software components distributed across networked computers.
- CORBA Common Object Request Broker Architecture enables separate pieces of software written in different languages and running on different computers to work with each other like a single application or set of services. Web services
- DDS Data Distribution Service for Real-Time Systems (DDS) is an Object Management Group (OMG) machine-to-machine middleware standard that aims to enable scalable, real-time, dependable, high performance and interoperable data exchanges between publishers and subscribers.
- Java RMI Java Remote Method Invocation is a Java API that performs the object-oriented equivalent of remote procedure calls (RPC), with support for direct transfer of serialized Java objects and distributed garbage collection.
- Jini also called Apache River, is a network architecture for the construction of distributed systems in the form of modular co-operating services.
- WCF The Windows Communication Foundation (or WCF), previously known as "Indigo", is a runtime and a set of APIs (application programming interface) in the .NET Framework for building connected, service-oriented applications.
- Apache Thrift is used as a remote procedure call (RPC) framework and was developed at Facebook for "scalable cross-language services development".
- ...

This list can be continued. Most of these technologies are inspired by idea of RPC⁸. An **RPC** is initiated by the client, which sends a request message to a known remote server to execute a specified procedure with specified parameters. The remote server sends a response to the client, and the application continues its process. This idea is described in details in **subsection 2.4**.

Web Services are the most popular technology for implementing service-oriented software nowadays. Next section will focus on this framework and on the main features that any **SOA** implementation should have.

2.2 Web Services architecture

2.2.1 Web Services Model

The Web Services architecture is based on the interactions between three roles [4]: service provider, service registry and service requestor. This integration has of three operations: publish, find and bind. The service provider has an implementation of service. Provider defines a service description and publishes it to a service requestor or service registry. The service requestor uses a find operation to retrieve the service description locally or from the service registry and uses the service description to bind with the service provider and invoke or interact with the Web service implementation. **Figure 1** illustrates these service roles and their operations.

⁸Remote procedure call

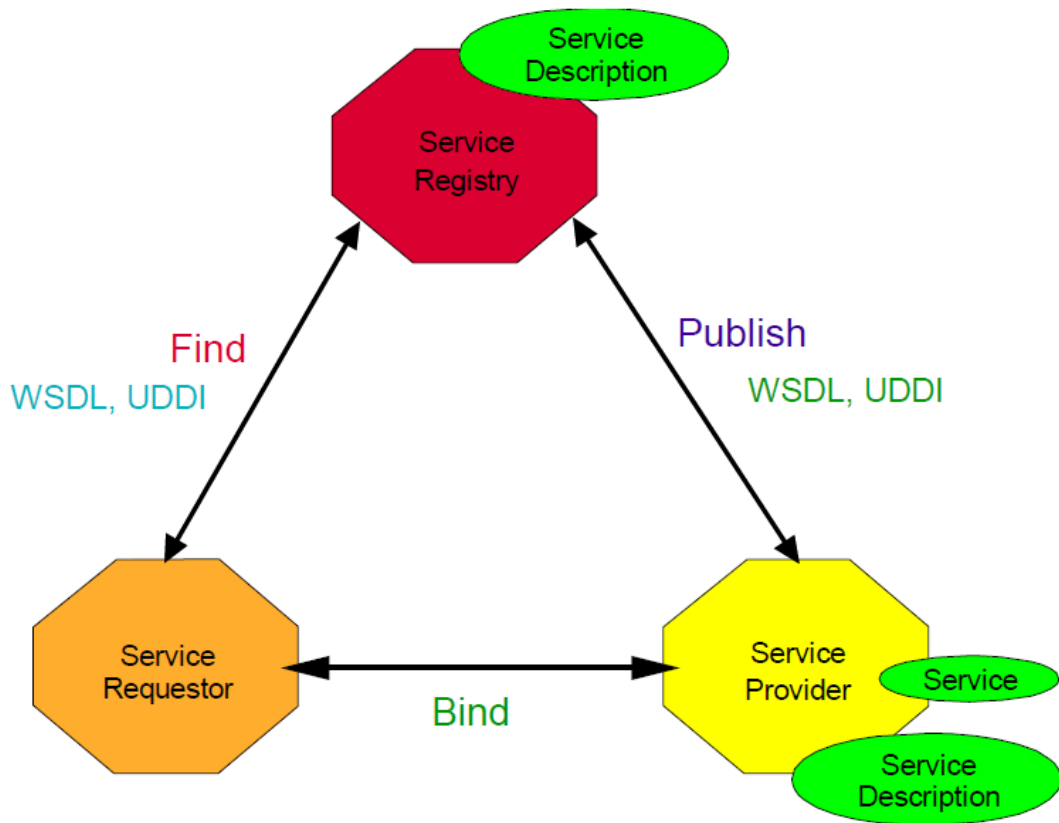


Figure 1: Web Services roles, operations and artifacts [4]

Service registry is a place where service providers can publish descriptions of their services. Service requestors can find service descriptions and get binding information from them. Binding can be static and dynamic. Registry is needed more for dynamic binding where client can get service info at the runtime, extract necessary functional methods and execute them on the server. During static binding service description may be directly delivered to the client at the development phase, for example using usual file, FTP⁹ server, Web site, email or any other file transfer protocol. There are also available special protocols, named Service discovery protocols (SDP), that allow automatic detection of devices and services on a network. One of them is the UDDI¹⁰ protocol, which is also was mentioned on Figure 1. UDDI is shortly described in **Web Services Protocol Stack** section.

Artifacts of a Web Service Web service consists of two parts [4]:

- **Service Description** The service description contains the details of the interface and implementation of the service. This includes its data types, operations, binding information and network location. There could also be a categorization and other metadata about service discovery and utilization. It may contain some Quality of service (QoS) requirements.
- **Service** This is the implementation of a service - a software module deployed on network accessible platforms provided by the service provider. Service may also be a client of other services. Implementation details are encapsulated inside a service, and client does not know the details how server processes his request.

2.2.2 Web Services Protocol Stack

WS architecture uses many layered and interrelated technologies. Figure 2 provides one illustration of some of these technology families.

⁹File Transfer Protocol

¹⁰Universal Description, Discovery and Integration

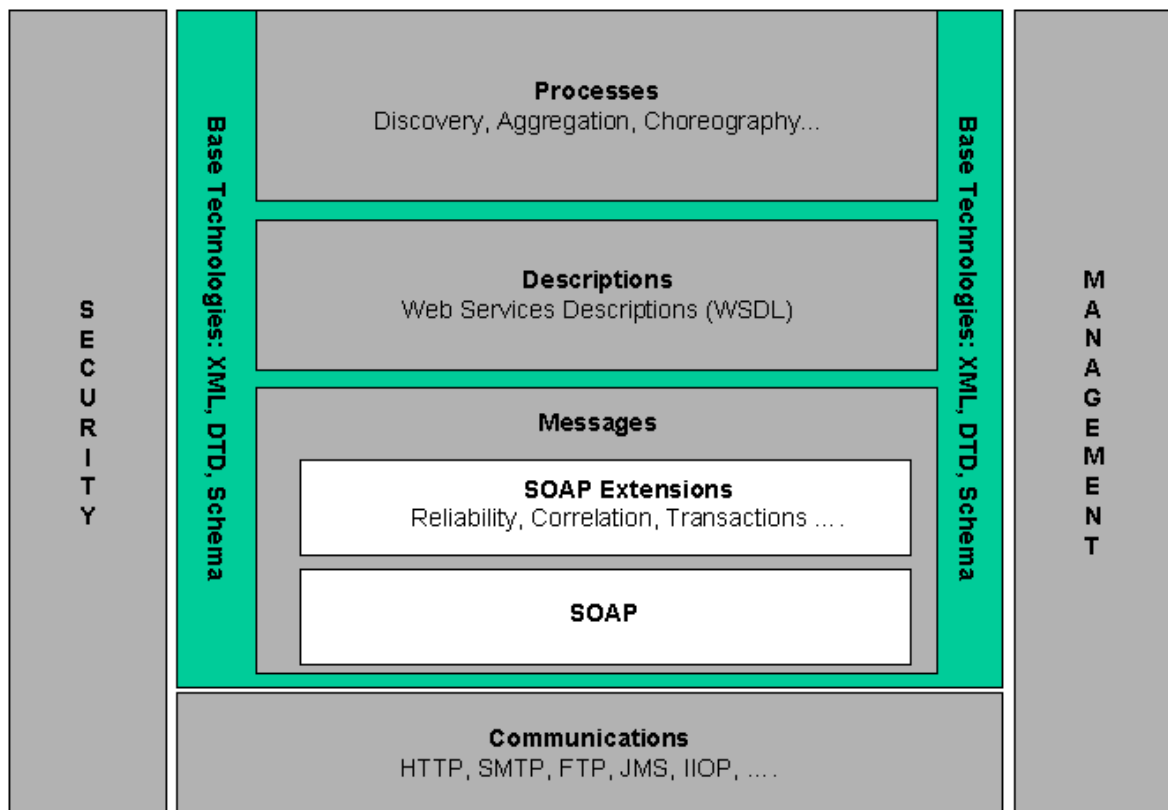


Figure 2: Web Services Architecture Stack [5]

We can describe these different layers as follows:

- **Communications** - This layer represents a transport between communication parties(service provider, client, service registry). This layer can be any network protocol like: **HTTP**, **FTP**, **SMTP**¹¹ or any other suitable transport protocol. If Web service is used in the Internet, the transport protocol in most cases will be **HTTP**. In internal networks there is the opportunity to agree upon the use of alternative network technologies.
- **Messages** - In order to communicate with a service, client should send a message. Messages are **XML** documents with different structure. **SOAP** protocol defines how these messages should be structured. **SOAP** is implementation independent and may be composed using any programming language. Protocol specification and message descriptions can be found in document SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)[6].
- **Descriptions** - This layer contains the definition of service interface (see also section 2.2.1). Web Services use **WSDL** language for describing the functionality offered by a service. **WSDL** file is the contract of service, which contains information about how the service can be called, what parameters it expects, and what data structures it returns. It is similar to method signatures in different programming languages.
- **Processes** - This part contains specifications and protocols about how service could be published and discovered. Web services are meaningful only if potential users may find information sufficient to permit their execution. Service as a software module has its own lifecycle, it needs to be deployed and deleted somehow. Traditional Web Services use **UDDI** mechanism to register and locate web service applications. **UDDI** was originally proposed as a core Web service standard.
- **Security** - Threats to Web services include threats to the host system, the application and the entire network infrastructure. To solve problems like authentication, role-based access control, message level security there is need for a range of XML-based security mechanisms.

¹¹Simple Mail Transfer Protocol

Web services architecture uses WS-Security¹² protocol to solve security problems. This protocol specifies how SOAP messages may be secured.

- **Management** - Web service management tasks are [5]: monitoring, controlling, reporting of service state information.

Web services architecture uses WS-Management [8] protocol for management of entities such as PCs, servers, devices, Web services and other applications. WS-Management has ability to discover the presence of management resources, control of individual management resources, subscribe to events on resources, execute specific management methods.

WS-Management was created by DMTF(Distributed Management Task Force, Inc., <http://www.dmtf.org/>) organization, which is creating standards for managing the enterprise level systems. Organisation members are largest hardware and software corporations like Broadcom, Cisco, Fujitsu, Hewlett-Packard, IBM, Intel, Microsoft, Oracle. DMTF standards promote multi-vendor interoperability, which is great for the integration between different IT systems.

Most of mentioned protocols are recommended by W3C Consortium and are production standards. Lots of SOA information systems use WS-* protocols for enterprise level services. Mostly these protocols are based on XML and SOAP. One example of such protocol is the WSDL service description.

2.2.3 Service Description and Service Contract

WSDL file is an XML document which has specification of service contract. As it was mentioned earlier(section 2.2.1) contract should be shipped with a component and should tell the client what input does service expect, and what output it will produce if specified input conditions are met. Contract may be a primary specification and it should be enough for a client to start using a service. This is similar to library header file in C language. You have a ready and compiled library shipped with a header file, where are all method declarations and definitions of data structures. If header file is verbose enough, there is no need to use the documentation. You can place this component into your system very easily.

Regular WSDL document contains some necessary elements [9, 10]: Service, Endpoint, Binding, Interface and Message Types. Table 1 describes them in details.

Document may also contain optional element named *documentation*. There may be human readable service documentation, with purpose and use of the service, the meanings of all messages, constraints their use, and the sequence in which operations should be invoked. To be documentation more complete, you may specify an external link to any additional documentation.

You can see the example of WSDL service contract in the appendix Appendix A. This example is from the official WSDL standard [9]. It describes a hotel reservation service, where you can book you a room in a fictional hotel named GreatH. For simplicity it describes only one method - the *opCheckAvailability* operation. This description is quite verbose to understand what it is about. There is input and output object type declaration. It also has an output error response declaration and if some error occurs during client request processing, server should send a message of specified kind. These XML object types are declared in different *namespaces* (see xmlns:* declarations at the start of WSDL document). This gives an ability to group domain types into one separate file and your main description file would not be overcrowded.

WSDL definition of the service does not contain any additional information about service hosting company and its products. At the moment when you get a service contract, you already know what company provides this service and what for this service was made. There is assumption that service provider somehow gave you this service contract. Another possibility to get the service description is to use a special *directory* or catalog, where you can find all information about company you are dealing with. Web Services architecture include UDDI mechanism for that particular purpose.

Official UDDI Version 3.0.2 Technical Specification draft [11] defines UDDI as follows:

The focus of Universal Description Discovery Integration (UDDI) is the definition of a set of services supporting the description and discovery of (1) businesses, organizations, and other Web services providers, (2) the Web services they make available, and (3) the

¹²WS-Security (Web Services Security, short WSS) is an extension to SOAP to apply security to web services. It is a member of the WS-* family of web service specifications and was published by OASIS(Organization for the Advancement of Structured Information Standards, <https://www.oasis-open.org/>). [7]

¹³XML schema is the XML document, that specifies structure of other XML document and describes data types and constraints, that other document might have. You can create a schema for necessary XML data structure and verify if processed message corresponds to schema you have already defined

WSDL 2.0 Term	Description
Service	The service element describes <i>where</i> to access the service. A WSDL 2.0 service specifies a single interface that the service will support, and a list of endpoint locations where that service can be accessed.
Endpoint	Defines the address or connection point to a Web service. It is typically represented by a simple HTTP URL string. Each endpoint must also reference a previously defined binding to indicate what protocols and transmission formats are to be used at that endpoint.
Binding	Specifies concrete message format and transmission protocol details for an interface, and must supply such details for every operation and fault in the interface.
Interface	Defines a Web service, the operations that can be performed, and the messages that are used to perform the operation. Defines the abstract interface of a Web service as a set of abstract <i>operations</i> , each operation representing a simple interaction between the client and the service. Each operation specifies the types of messages that the service can send or receive as part of that operation. Each operation also specifies a message exchange <i>pattern</i> that indicates the sequence in which the associated messages are to be transmitted between the parties.
Message Types	The types element describes the kinds of messages that the service will send and receive. The XML Schema ¹³ language (also known as XSD) is used (inline or referenced) for this purpose.

Table 1: Objects in WSDL 2.0 [9, 10]

technical interfaces which may be used to access those services. Based on a common set of industry standards, including HTTP, XML, XML Schema, and SOAP, UDDI provides an interoperable, foundational infrastructure for a Web services-based software environment for both publicly available services and services only exposed internally within an organization.

UDDI mechanism uses SOAP messages for client-server communication. Service provider publishes the WSDL to UDDI registry and client can find this service by sending messages to the registry (see also Figure 1). UDDI specification defines the communication protocol between UDDI registry and other parties.

UDDI registry is a storage directory for various service contracts, where lots of companies hold their service descriptions. WSDL contracts may also be published on a company website using direct link to the WSDL file, but UDDI contains them all in one place with the ability to search and filter. You have a choice and there is a possibility to find most suitable service from all provided companies and services.

2.2.4 Advantages and disadvantages of WS-* standards

Usage of WS-* technologies gives some benefits [12]:

- Reusability
- Interoperability and Portability
- Standardized Protocols
- Automatic Discovery
- Security

WS-* uses the HTTP protocol as transport medium for exchanging messages between web services. SOAP messages can be transfered using another protocol (SMTP, TCP¹⁴, or JMS¹⁵), which can be more suitable to your system environment than HTTP.

¹⁴Transmission Control Protocol

¹⁵Java Message Service

One another fundamental characteristic of web services is the service description and contract design. Contract specification gives you the ability to reuse service functionality in many different and separate applications. Contract in Web Services is general standardized description of a service in universal data format (XML and WSDL), that is platform and programming language independent. Service description is only the interface and the implementation of that interface may be unknown by the service client. There is possibility to transparently change (totally or partially) implementation details of a service.

The main reason why Web Services standards are bad in context of embedded systems is the performance. Web services impose additional overhead on the server since they require the server to parse the XML data in the request. Web Services use SOAP messages, which are structured XML data, for client-server communication. Some experiments [13, 14] show that performance of SOAP transfer is more than 5 times slower compared to others SOA implementations, like CORBA¹⁶ or custom made protocol messages. If you start reading WS-* standards one by one, you will ensure that all they are interconnected using idea of XML, SOAP and HTTP.

Another statement against Web Services is that these standards are too complicated and their documentation is hard to understand. Document [16] contains a use case survey about two most common implementations of SOA: Web Services and REST¹⁷. Most of people in the survey agreed that WS-* standards is not easy to learn and adapt. WS-* suits better for highly integrated business solutions, but not for simple applications, with atomic functionality.

WS-* standards rely on each other and to implement a small web service with few features you will need to dive into all WS standards. This is at least 783 pages of not just text, but a technical specifications [17]. Surely, Web services have good ideas, but this technology is promoted and developed by large corporations like Microsoft, IBM, Oracle, who are not interested in simple and lightweight solutions, because they need to utilize their thousands of developers and earn money(document [17] says that most of WS-* specifications are hosted by Microsoft and OASIS organisation, which foundational sponsors are IBM and Microsoft). It seems that Web services are trying to solve every business problem and there is a *WS-problem* standard for it.

This topic described Web Service architecture features. There are lots of useful principles like portability, interface description and message exchange patterns, but the WS-* implementation is not suitable for resource-constrained hardware.

The REST has some advantages over WS-*. Next section will shortly describe the main principles of REST approach.

2.3 REST and RESTful services

2.3.1 What is the REST?

Representational state transfer (REST) is a software design model for distributed systems [18]. This term was introduced in 2000 in the doctoral dissertation of Roy Fielding, one of the principal authors of the Hypertext Transfer Protocol (HTTP) specification. REST uses a stateless, client-server, cacheable communications protocol which is almost always the HTTP protocol. Its original feature is to work by using simple HTTP to make calls between machines instead of choosing more complex mechanisms such as CORBA, RPC or SOAP.

REST-style architectures conventionally consist of clients and servers. Clients make requests to servers, servers process requests and return responses. Requests and responses are built around the transfer of *representations* of *resources*. Author defines the resource as the key abstraction of information in REST [18]. It can be any information that can be named and addressed: documents, images, non-virtual physical systems and services. A representation of a resource is typically a document that captures the current or intended state of a resource.

Restful applications use HTTP requests to change a state of resource¹⁸: post data to create and/or update resource, read data (e.g., make queries) to get current state of resource, and delete data to delete existing resource.

REST does not offer security features, encryption, session management, QoS guarantees, etc. But these can be added by building on top of HTTP, for example username/password tokens are often used for encryption, REST can be used on top of HTTPS (secure sockets)[12].

¹⁶The Common Object Request Broker Architecture (CORBA) is a standard defined by the Object Management Group (OMG) that enables software components written in multiple computer languages and running on multiple computers to work together (i.e., it supports multiple platforms).[15]

¹⁷Representational state transfer

¹⁸ all four CRUD(Create/Read/Update/Delete) operations)

2.3.2 Key principles of REST

REST is a set of principles that define how Web standards, such as HTTP and URI¹⁹s, are supposed to be used.

The five key principles of REST are^[19]:

- Give every “thing” an ID
- Link things together
- Use standard methods
- Resources with multiple representations
- Communicate statelessly

Give every “thing” an ID

Every resource need to be reachable and identifiable. You need to access it somehow, therefore you need an identifier for the resource. World Wide Web uses URI identifiers for that purpose. Resource URI could look like:

```
http://example.com/customers/1234
http://example.com/orders/2007/10/776654
http://example.com/products/4554
http://example.com/processes/salary-increase-234
http://example.com/orders/2007/11
http://example.com/products?color=green
```

Listing 1: Resource identifier examples ^[19]

URIs identify resources in a global namespace. This means that this identifier should be unique and there should not be another same URI. This URI may reflect a defined customer, order or product and it might correspond to database entry. ¹ last two examples identify more than one thing. They identify a collection of objects, which is the object itself and require an identifier.

Link things together

Previous principle introduced an unique global identifier for the resource. Resource URI gives possibility to access the resource from different locations and applications. Resources can be also linked to each other. Listing 2 shows such scheme. Representation of an order contains the information about this order and linked product and client resources. This approach gives client an opportunity to change a state of client application by following linked resources. After receiving order information client has two possibilities for choice: to get product information or to fetch customer details.

```
<order self='http://example.com/orders/2007/10/776654' >
  <amount>23</amount>
  <product ref='http://example.com/products/4554' />
  <customer ref='http://example.com/customers/1234' />
</order>
```

Listing 2: Example of linked resources^[19]

The idea of links is a core principle of the Web ²⁰

¹⁹Uniform Resource Identifier

²⁰The World Wide Web (abbreviated as WWW or W3.[3] commonly known as the web), is a system of interlinked hypertext documents accessed via the Internet.^[20]

Use standard methods

There should be a standard interface for accessing the resource object. REST relies on HTTP protocol, which has definitions of some standard request methods: GET, PUT, POST and DELETE. Table 2 describes standard actions on resource.

Resource	GET	PUT	POST	DELETE
Collection URI, such as <code>http://example.com/resources</code>	List the URIs and perhaps other details of the collection's members.	Replace the entire collection with another collection.	Create a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation.	Delete the entire collection.
Element URI, such as <code>http://example.com/resources/item17</code>	Retrieve a representation of the addressed member of the collection, expressed in an appropriate Internet media type.	Replace the addressed member of the collection, or if it doesn't exist, create it.	Not generally used. Treat the addressed member as a collection in its own right and create a new entry in it.	Delete the addressed member of the collection.

Table 2: RESTful web API HTTP methods [21]

All four CRUD²¹ operations may be done with these HTTP methods. It is possible to manage a whole lifecycle using only these methods. Client of the service should only implement the default application protocol (HTTP) in correct way.

Resources with multiple representations

When client gets representation of a resource using GET method he does not know which data format will server return. REST as architectural method does not provide any special standard for resource representation. How can client and server could make an agreement about data format they will use?

HTTP protocol help to solve these problems again. It has a special field that defines the operating parameters of an HTTP transaction. Field Accept in the header of HTTP request message specifies content type that is acceptable for the response [22].

Such request header could look like this:

```
GET /images/567 HTTP/1.1
Host: example.com
Accept: image/jpeg
```

Listing 3: Request for a representation of resource in a particular format

This means that client expects that representation of a resource having identifier `http://www.example.com/images/567` should be in *image/jpeg* format. Both client and server should be aware of such format, whole system may be designed around any special format. There could be also another representation of same resource (the same image), for example *image/png* or *image/bmp* formats, that server could send according to received request.

Not only outgoing data format could be specified. Server can also consume data in specific formats (there are different specific header fields for this, for example *Content-Type*).

Using multiple representations of resources helps to connect more possible clients to the system.

²¹Create, read, update and delete

Communicate statelessly

Stateless communication helps to design more scalable systems. RESTful server does not keep any communication context. Each incoming request is new for the server and there should be enough information to necessary to understand the request[18].

Server could contain all the information about each connected client, but it requires a meaningful amount of resources. Server need to keep and control the current state of application for every client, which locks the server resources while client is not active (opened connections, memory, data integrity locks).

There is no need for keeping using all these resources if the application state is on the client side. Client controls the flow of the application and changes its state by making requests to server. Server does not make any work while clients are not sending requests, it starts to work only on demand.

You can easily switch between different servers if there is no any client context on the server side. Imagine a system with some amount of application servers and load balancer server in it. All application servers run the same application. While some servers are making hard work, another servers may be idle. Load balancer have a possibility to route new incoming requests to a server, which has a smallest workload, because of the absence of application context between client and server. Even parallel request of the same client could be handled by different servers. Such system become more scalable and new server nodes can be easily added or removed.

The main disadvantage of such approach is the decrease of network performance. Client needs to repeat sending the same session data on every new request, because that context data cannot be stored on the server.

2.3.3 Implementation constraints

The REST architectural style can be described by the following six constraints applied to the elements in this architecture[18]:

1. **Client-Server** Separation of concerns is the principle behind the client-server constraints. User interface is separated from data storage and has improved portability. Server side does not aware of client application logic and server tasks may be more optimized and independent.
2. **Stateless** This constraint reflects a design trade-off of keeping session information about each client on the server. Stateless method does not keep any application context on the server and allows to build more scalable server components. Each new request contains enough information to process it, but such technique increases network traffic by sending repeating session information over the network again and again.
3. **Cache** This constraint improves network efficiency. The data in the server response may be marked as cacheable or non-cacheable. Client is able to store cacheable data on its side and reuse it, if it needs to send the same request again. Frequently changed data should not be marked as cacheable in order to provide data integrity.
4. **Uniform Interface** System becomes more universal if the interface of all components is the same. You can add new and replace existing components more easily. Component implementations are decoupled from the services they provide, system components are more independent.
5. **Layered System** style allows an architecture to be composed of hierarchical layers, that separate knowledge between components from different layers. Components in each layer do not know the structure of a whole system, but can only communicate to each other and with neighbour layers through a specified interface. Layers can be used to encapsulate legacy services and to protect new services from legacy clients. Structures inside a layer may be transparently changed. **Figure 3** shows such a complex layered system.

The primary drawback of layered systems is that they add overhead and latency to the processing of data. Every additional layer requires new amount of resources (which could be shared using common access, but layers are separated and they don't) and reduce communication performance by introducing new bottleneck at the layer boundaries.

6. **Code-On-Demand** This is an optional constraint. REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts. Not all features on

a client side may be implemented. Your system could download execution instructions from the server. This is how JavaScript and Java applets work in the Web browser.

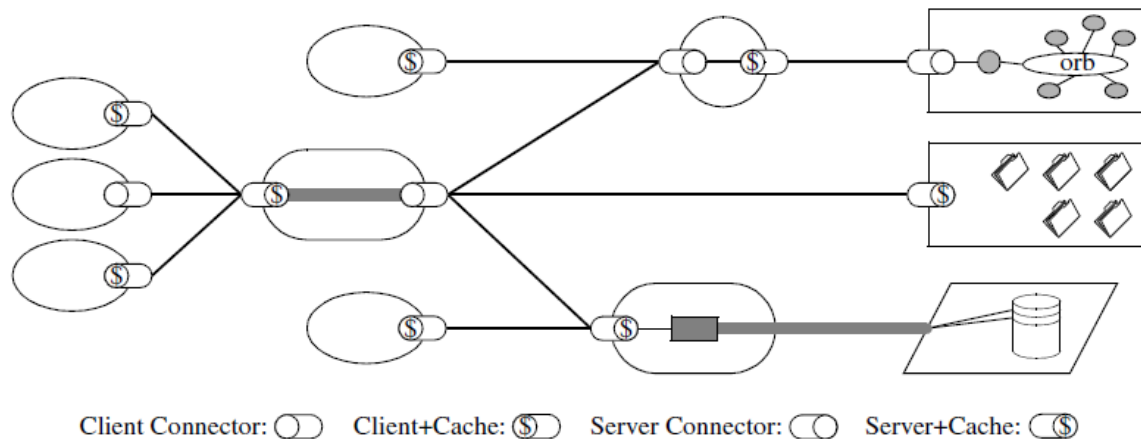


Figure 3: Uniform-Layered-Client-Cache-Stateless-Server [18]

2.3.4 Summary

Notion of independence was mentioned above in this section quite many times. This is because REST itself is a high-level style that could be implemented using any different technology and your favourite programming language. It was initially described in the context of HTTP, but it is not limited to that protocol. You can use all your creativity in a system design and you are only limited with a small amount of abstract constraints. RESTful applications maximize the use of the existing, well-defined interface and other built-in capabilities provided by the chosen network protocol, and minimize the addition of new application-specific features on top of it[19]. Therefore RESTful Web services seamlessly integrate with the Web and are straightforward and simple way of achieving a global network of smart things.

2.4 Remote procedure calls and *-RPC

Many distributed systems are based on explicit message exchange between processes. If you see a list of SOA technologies (provided above, see [subsection 2.1](#)) you can find that many of these technologies use RPC within them. Some of them don't, for example REST described in previous section, it uses different resource oriented approach(resources which the client can consume). Other majority of technologies are message oriented and use messages for IPC²². In RPC messages are sent between client and server to call methods and receive results.

2.4.1 RPC in details

Remote procedure calls have become a de facto standard for communication in distributed systems[23]. The popularity of the model is due to its apparent simplicity. This section gives a brief introduction to RPC and the problems in there.

Only one figure is enough to describe RPC(see [Figure 4](#)).

When a process on client machine calls a procedure on server machine, the calling process on client is suspended, and execution of the called procedure takes place on server. Information can be transported from the caller to the callee in the parameters and can come back in the procedure result. No message passing at all is visible to the programmer. Programmer operates only with method calls.

The idea behind RPC is to make a remote procedure call look as much as possible like a local one. In other words, we want RPC to be transparent—the calling procedure should not be aware that the called procedure is executing on a different machine or vice versa[23]. To achieve this

²²Inter-process communication

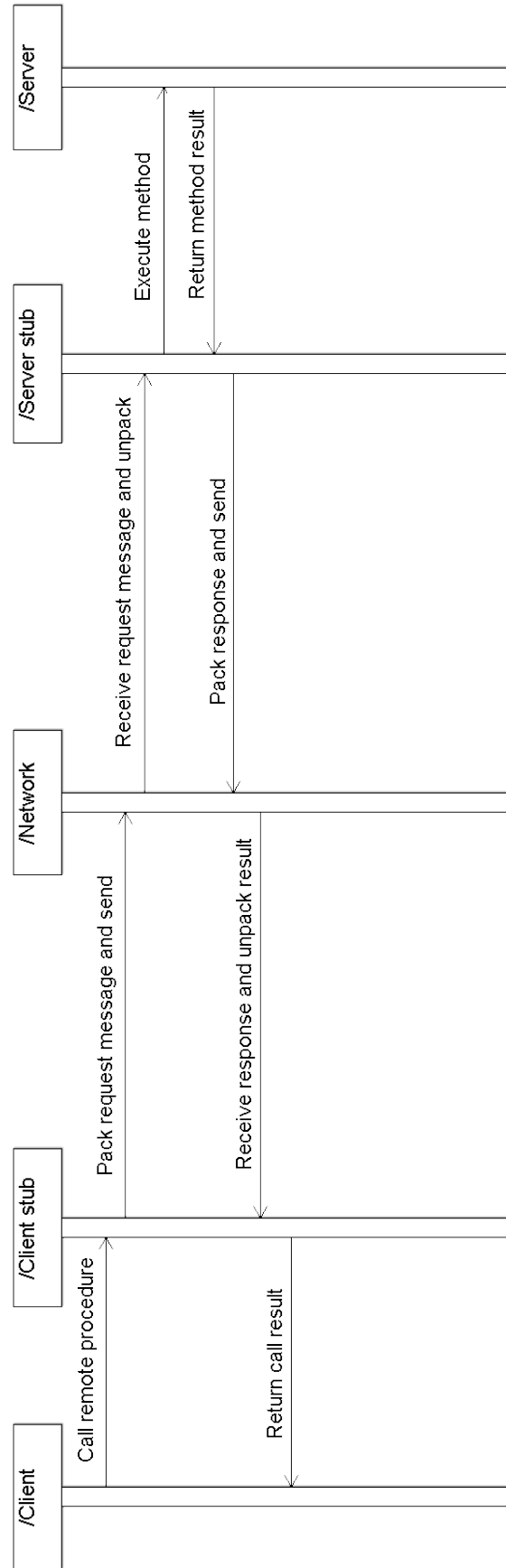


Figure 4: Principle of RPC between a client and server program

transparency special software modules are used. They are called **stubs**. The main purpose of a stub is to handle network messages between client and server.

Whole RPC call process could be described using words like this:

1. The client procedure calls the client stub on the client machine.
2. The client stub builds a message and sends it over the network to remote machine using local operating system(OS).
3. The remote OS receives the message from the network and gives this message to the server stub. Server stub unpacks the parameters and calls the server.
4. The server does the work and returns the result to the server stub.
5. The server stub packs result in a message and sends it to client using network and underlying OS.
6. The client's OS gives the message to the client stub. The stub unpacks the result and returns to the client.
7. Client receives procedure result and continues his processing.

Modern software tools help to make this process very easy. Here is the real world example of the small RPC system(written using Python programming language) that proves this^[24]:

```
import xmlrpclib
from SimpleXMLRPCServer import SimpleXMLRPCServer

def is_even(n):
    return n%2 == 0

server = SimpleXMLRPCServer(("example.com", 8000))
print "Listening on port 8000..."
server.register_function(is_even, "is_even")
server.serve_forever()
```

Listing 4: RPC server example (Python and xmlrpclib)

Code is quite verbose²³ and people, who are not familiar with Python, could understand it. You create a server using the special RPC server implementation from *xmlrpclib* library. You specify a remote host and a port number in a object constructor. When server object is created you need to specify remote methods, which may be executed. Example above uses small even check method. Server registers the methods and starts to wait for incoming calls.

Client implementation for the corresponding server looks like:

```
import xmlrpclib

proxy = xmlrpclib.ServerProxy("http://example.com:8000/")
print "3 is even: %s" % str(proxy.is_even(3))
print "100 is even: %s" % str(proxy.is_even(100))
```

Listing 5: RPC client example (Python and xmlrpclib)

Actually, it is more shorter and simpler than server code. You just specify a remote server object and receive a proxy object. Then you can use this proxy like a usual local object. This creates

²³Python programming language has its own philosophy, called The Zen of Python. It has some statements how software should be designed. Two statements in the Zen of Python that are related to this example are:

Simple is better than complex.

and

Readability counts.

an illusion that you are not going anywhere for the result and working with a usual objects in the code. Proxy object may be passed as a parameter to a function and be used in that function without knowledge where it was came from.

This idea is simple and elegant, but there are exist some problems. First of all, calling and called procedures run on different machines and are executed in different address spaces, which introduce additional complexity in passing parameters and results between client and server. Finally, both machines can independently crash, therefore special error handling mechanism is required. System developers must deal with such failures without knowing about remote procedure was actually invoked or not.

As long as the client and server machines are identical and all the parameters and results are scalar types, such as integers, characters, and Booleans, this model works fine. However, in a large distributed system, it is common that multiple machine types are present. Each machine often has its own representation for numbers, characters, and other data items[23].

There are several representations of character data used in computer systems: one byte characters(ASCII²⁴, EBCDIC²⁵), multibyte characters(UTF-8, UTF-8, UTF-32²⁶,) and characters in different encodings(all tree character encodings are used for cyrillic symbols: Windows-1251, Code page 855, ISO/IEC 8859-5). Each RPC client and server should agree about charater encoding they will use.

In addition to that, problems can occur with the representation of integers (sign-and-magnitude method, one's complement, two's complement) and real data types(floating-point, fixed-point, binary-coded decimal and single precision, half precision, double precision). Some machines have different endianness²⁷. If two different machines, one little-endian and other big-endian, are communicating to each other, they should use common endianness, otherwise they will accept bytes in wrong order and data will be invalid.

There was description of primitive data types in RPC so far, but client and server not always send primitive data types to each other. There could also be a complex data structures, that contains several primitives like characters, numbers or just raw bytes. Client and server should be aware of structure of messages they send and receive. Usually these stuctures are specified by Interface Definition Language(IDL). IDL is a is a specification language used to describe a software component's interface. IDLs describe an interface in a language-independent way, enabling communication between software components that do not share same language and platform. An interface is firstly specified in an IDL and then compiled into a client stub and a server stub. RPC-based middleware systems offer an IDL to support application development[23].

In most cases communication scheme is well known and some standard message protocol is used. Previous example of RPC system, that was written using Python programming language, used XML-RPC protocol. XML-RPC defines XML data types that are used in RPC messages. There are several alternative common used schemes, that provide similar functionality. They all could be divided into two groups: platform and programming language dependent and systems that can be used in multiplatform and multilanguage environment. XML-RPC belongs to second group, the similar technologies are JSON-RPC, SOAP and CORBA. Actually, most RPC implementations in the first group do not require any special harware. They are programming language dependent and may be used with the assumption, that components in the system are written using that specific language. Some examples are provided below:

- Java Remote Method Invocation
- Pyro object-oriented form of RPC for Python.
- Windows Communication Foundation (.NET framework)
- ...

Most of these programming languages have multiplatform implementations and RPC system may be built using various hardware.

²⁴ American Standard Code for Information Interchange

²⁵Extended Binary Coded Decimal Interchange Code (from IBM)

²⁶Universal Character Set Transformation Format

²⁷The terms little-endian and big-endian refer to the way in which a word of data is stored into sequential bytes of memory. The first byte of a sequence may store either the least significant byte of a word (little-endian) or the most significant byte of a word (big-endian). Endianness refers to how bytes and 16-bit half-words map to 32-bit words in the system memory. [25]

2.4.2 RPC summary

This section described one another possible way of **IPC**. RPC is used in many distributed systems in obvious or implicit way. It provides mechanism for calling subroutines or procedures on another computer or system. Communication in RPC is based on message passing. The Client sends to the server a message containing request for method call. The Server sends to the client a message with procedure results. Modern software tools provide ready RPC libraries and application programmer has no need to explicitly reinvent whole RPC system from scratch. You can build various distributed systems using RPC.

2.5 Data serialization

Serialization is a process for converting a data structure or object into a format that can be transmitted through a wire, or stored somewhere for later use [26].

Previous sections described some possible implementations of Service Oriented Architecture. These technologies use client-server communication and send information between client and server, that need to be understood at both destinations. No matter how this information is sent, using resource/object representation in case of REST or request/response message in case of RPC, it needs to be converted to format that can be decoded with the user of that information. Common transmission scenario can look like:

1. Client wants to send some information to a server. It has some data in memory and that data is in application specific format(object structure, text, image, movie file).
2. Client **packs** his information into a message and sends it to server using any possible transport channel(email, paper mail, homing pigeon, tcp socket, etc).
3. Server receives this message, **unpacks** the message and gets a piece of information that client wanted to send.
4. Server reads the information and decides what to do with it.

Process of packing information is called **serialization** (also deflating or marshalling) and process of unpacking is called **deserialization** (inflating or unmarshalling).

There are lots of different ways and formats that can be used. Which method and format to choose depends on the requirements set up on the object or data, and the use for the serialization (sending or storing). The choice may also affect the size of the serialized data as well as serialization/deserialization performance in terms of processing time and memory usage[26]. Next section describes possible serialization solutions.

2.5.1 Serialization technologies

Serialization is supported by many programming languages, which provide tools and libraries for data serialization to different formats. Article [27] provides a summary of well known data serialization formats. Most of them could be divided into two groups: human-readable text based formats and binary formats. Both groups have their own advantages and disadvantages. Table 3 shortly describes them.

Property	Binary formats	Human-readable formats
Format examples	Protocol Buffers from Google Apache Thrift(TBinaryProtocol) BSON used in MongoDB database MessagePack(http://msgpack.org) and most of native serialization mechanisms in various programming languages(Java, Python, .NET framework, C++ BOOST serialization)	XML JSON YAML
Advantages	The two main reasons why binary formats are usually proposed are for size and speed . Typically use fewer CPU cycles and require less memory. Binary data is transformed as is, no need to encode data bytes(image, video, etc) Better for larger datasets Random data access.	Do not have to write extra tools to debug the input and output; you can open the serialized output with a text editor to see if it looks right. Self-descriptive and easily recoverable. No need to use programming issues like sizeof and little-endian vs. big-endian. Platform and programming language independent. Broad support by tools and libraries
Disadvantages	Not verbose. Hard to debug. Fixed data structures. Hard to extend. Not self-descriptive(it is hard to understand for human where actual data starts in the array of bytes,), has no data description(metadata, layout of the data) Require special software and highly customized data access algorithms. Hard to recover data after software version change (remember different MS office formats)	Binary data needs to be converted to text form(Base64). Additional processing overhead (CPU and memory consumption). Lot of redundancy. Representing your data as text is not always easy/possible or reasonable(video/audio streams, large matrices with numbers)

Table 3: Comparison of binary and human-readable serialization formats

Text-based nature makes human-readable format a suitable choice for applications where humans are expected to see the data, such as in document editing or where debugging information is needed. Binary formats are better for high speed and low latency applications.

2.5.2 XML vs JSON

Choosing the right serialization format mostly depends on your data and application. But if there is no any constraint what protocol to use, text protocols are more preferable. They give you advantages like: verbosity, extensibility, portability.

Most popular human and machine readable serialization formats are XML and JSON.

XML

XML is hugely important. Dr Charles Goldfarb, who was personally involved in its invention, claims it to be “the holy grail of computing, solving the problem of universal data interchange between dissimilar systems.” It is also a handy format for everything from configuration files to data and documents of almost any type. [28]

The fundamental design considerations of XML include simplicity and human readability[29]. W3C²⁸ specifies the design goals for XML like [30]:

1. XML shall be straightforwardly usable over the Internet.
2. XML shall support a wide variety of applications.
3. XML shall be compatible with SGML²⁹.
4. It shall be easy to write programs which process XML documents.
5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
6. XML documents should be human-legible and reasonably clear.
7. The XML design should be prepared quickly.
8. The design of XML shall be formal and concise.
9. XML documents shall be easy to create.
10. Terseness in XML markup is of minimal importance.

The primary uses for XML are Remote Procedure Calls and object serialization for transfer of data between applications.

Simple data structure in XML looks like:

```
<person>
  <firstname>John</firstname>
  <surname>Smith</surname>
  <email>john.smith@example.com</email>
  <mobile>1234567890</mobile>
</person>
```

Listing 6: XML structure describing abstract person

JSON

JSON (JavaScript Object Notation) is a lightweight data-interchange format[32]. It is easy for humans to read and write and also is easy for machines to parse and generate. JSON is based on JavaScript Programming Language and is directly supported inside JavaScript. It has library bindings for popular programming languages.

JSON is built on two structures[32]:

- A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

²⁸World Wide Web Consortium

²⁹Standard Generalized Markup Language. SGML is a system for defining markup languages. Authors mark up their documents by representing structural, presentational, and semantic information alongside content. [31]

The same structure from XML section looks in JSON encoding like:

```
{
    "person" : {
        "firstname" : "John",
        "surname" : "Smith",
        "email" : "john.smith@example.com",
        "mobile" : 1234567890
    }
}
```

Listing 7: JSON structure describing abstract person

JSON value can be a string in double quotes, or a number, or true or false or null, or an object or an array. These structures can be nested.

JSON specification is quite easy and it is described using only one page [32]. This page provides also a list of programming languages and libraries that support JSON.

Research in different documents [33, 29] showed that JSON is faster and uses fewer resources than its XML counterpart. Transferring data in the form of JSON instead of XML can speed up the server data transfer efficiency.

This is because XML is characterised by its rich verbosity, meaning that it requires a separate start-tag and end-tag for describing the content. As JSON does not use end-tags at all for the description of the content, the resulting number of bytes is smaller. Paper [33] contains performance analysis of mobile device (Apple iPhone), which executes different tests working with XML, JSON and SOAP. Research results prove that SOAP and XML have overhead over JSON.

To make messages more smaller binary formats or compression (for example gzip) should be used. There is also available newly emerged Efficient XML Interchange (EXI)³⁰ standard for binary XML encoding. It is expected to become an alternative to XML for exchanging data between embedded systems [34].

Parsers

All JSON and XML parsers can be divided into two groups: stream-based and tree-based. Stream based parsers (also known as Simple API for XML (SAX)) are event-based. They read input message sequentially and signals the application when a new component has been read. They raise notifications on reaching different document parts and programmer needs to decide what to do with this part of document (store or skip). Such parsers require less resources than tree-based, because they do not need to keep whole document in memory.

Tree-based parsers load whole message to memory and then parse it. They create a tree of this document (also known as Document Object Model (DOM)) and return it to application developer. Programmer receives whole document tree and is able to extract needed parts from that.

Although stream-based parsers often have better performance than tree-based parsers, they make application code more complex due to their event-driven nature.

Document serializers work using similar scheme.

2.5.3 Is there a right serialization format?

There is no direct answer for that question. The format and the parser used has to be chosen according to your application constraints and requirements. Binary formats give you a small message size and reduce required amount of CPU cycles for data processing, but there is a lack of verbosity and it is hard to extract data without using additional tools. Text based formats reduce development and debugging time, but they have overhead because of their verbosity. There is a hardware resources/development time tradeoff in this problem. Some calculations should be made before making a decision.

Whatever format you choose, binary or human-readable, this should be a standard and open format. It means that this format should be supported by different software tools and programming languages, you can read its specification and understand its features, your people can study it more

³⁰It was adopted as a Candidate Recommendation by the W3C, for more information see <http://www.w3.org/TR/2013/CR-exi-profile-20130416/>

quickly, system integration becomes more easy, because other system understand it too. If it is a production standart, there is a chance that it will not essentially change in near future.

Text formats like XML and JSON are user all over the world and are good candidates to be in your system.

2.6 SOA and Embedded Systems

Resent topics had a review of available tools and technologies for implementing a SOA system. Most of them are not directly portable to embedded systems. Related publications [14, 16] claim that SOA tools need to be adapted to a constrained hardware by using more lightweight approaches. Common possibilities are: use of more resource friendly protocols, use of existing service protocols in a contstrained manner(low request per second ratio or smaller payload/packet size), use of special constrained protocols, which are designed specially for interaction between small devices.

This section will describe two possibilities of implementing SOA on an embedded device, that are based on different research papers[35, 1]. **Devices Profile for Web Services** section will introduce a Web Services based device mmcounication framework. **Constrained Application Protocol and Constrained RESTful Environments** section will cover RESTful device interactions.

2.6.1 Devices Profile for Web Services

The Devices Profile for Web Services (DPWS) was developed to enable secure Web service capabilities on resource-constrained devices[36]. DPWS was mainly developed by Microsoft and some printer device manufacturers. DPWS allows sending secure messages to and from Web services, discovering a Web service dynamically, describing a Web service, subscribing to, and receiving events from a Web service.

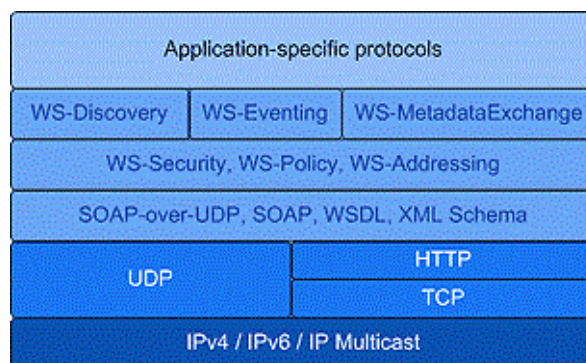


Figure 5: DPWS protocol stack [36]

Figure 5 shows DPWS protocol stack. It is based on several Web Services specifications[36]:

- WS-Addressing for advanced endpoint and message addressing
- WS-Policy for policy exchange
- WS-Security for managing security
- WS-Discovery and SOAP-over-UDP for device discovery
- WS-Transfer / WS- Metadataexchange for device and service description
- WS-Eventing for managing subscriptions for event channels

Like Web Services, DPWS uses SOAP, WSDL, XML-Schema.

DPWS has been ported to several target software platforms (Linux, Microsoft's Windows and Windows CE, ExpressLogic's ThreadX and Quadros Systems' Quadros). It was tested on a processor board comprising a 44-MHz ARM7 TDMI and associated memory (but no cache memory), running ThreadX. The static memory footprint of the device software including the OS, the TCP/IP protocol stack and the DPWS software is less than 500 KB, while the dynamic memory requirements are below 100 KB.[1]. Another research [14] reports that system disk space requirements are between 61 and 478 Kbytes.

The ways in which someone may be authenticated fall into three categories: (<http://en.wikipedia.org/wiki/Auth>)

- the ownership factors: Something the user has (e.g., wrist band, ID card, security token, software token, phone, or cell phone)
- the knowledge factors: Something the user knows (e.g., a password, pass phrase, or personal identification number (PIN), challenge response (the user must answer a question), pattern)
- the inherence factors: Something the user is or does (e.g., fingerprint, retinal pattern, DNA sequence (there are assorted definitions of what is sufficient), signature, face, voice, unique bio-electric signals, or other biometric identifier).

Authentication may be one way (only client is checked for validity) and two way (both client and server check each other). Some systems may require to use different security factors together: you say password, provide ID card and show your fingerprint. There are also available many standard authentication protocols. If you start searching you will probably find similar list:

- Transport Layer Security (TLS)
- Extensible Authentication Protocol (EAP)
- Password authentication protocol (PAP)
- Challenge-Handshake Authentication Protocol (CHAP)
- Password-authenticated key agreement
- Remote Authentication Dial In User Service (RADIUS)
- Kerberos
- Lightweight Extensible Authentication Protocol (LEAP)

Choosing suitable protocol is not a trivial problem. There is no any case general protocol. Most of them are designed to interconnect big computers inside a network. Mostly they operate on transport and application level and use TCP/IP protocol stack.

All these protocols could be divided into these groups:

- Protocols that transmit the secret over the network. (For example Password authentication protocol). These protocols are not secure.
- Protocols that do not send secrets and provide authentication through sending messages. (CHAP and Password-authenticated key agreement).
- Protocols that require a trusted third party.

Protocols of first type have been deprecated because of security reasons. They send sensitive data over the network and everyone else between two nodes can catch this data.

Second group of protocols was invented because the first group was unable to provide proper level of security. Link between client and server (two parties) does not contain pure information about the secret. Parties use cryptography and send encrypted messages to each other. Finally they authenticate each other when there is enough information gathered to validate the authority.

Last group uses trusted third party authority to check each other. There is assumption that all three parties should have connection between each other. Embedded device during client authentication needs to connect some server and ask for a secret. Third party should always have a high authority, two other parties should trust him. This scheme should be used in case of high security requirements.

Choosing of right authentication protocol in general should depend on application. Sometimes, there will be enough just to send plain text passwords over the network. Engineer should analyze all hazards during system design process.

Lifecycle of an embedded system is more longer than lifecycle of average personal computer. Application specific controller may run for decades and it will be still functional. Chosen security algorithm may be not secure enough after some years. Some vulnerabilities can be discovered during that period. Computational power of modern processors raises every year and secure encryption may be cracked during some seconds in the future. There is no 100% secure system, everything can be broken.

Your system security should have such encryption, that provides proper security level to your application data and can not be cracked quickly. How quick it is depends on your data and security requirements of your data.

Another aspect is the complexity of cryptography algorithms. Embedded devices are usually small low power devices with limited computation abilities. Not every algorithm suits well. It should be quick and resource friendly, and in same time it should be secure.

Nowadays, the last versions of the Wifi and Wimax standards include the use of Extensible Authentication Protocol (EAP) declined in different versions (LEAP - EAP using a Radius Server -, EAP-TTLS, etc...). In practice, EAP is interesting for workstations or desktop computers but does not fit the needed security of particular systems such as handheld devices, short-range communication systems or even domestic Wireless LAN devices. The reason being that many versions of EAP use certificates, public key encryption or exhaustive exchanges of information, that are not viable for lightweight wireless devices.[A new generic 3-step protocol for authentication and key agreement in embedded systems]

Protocol should small in code size. You should not to place a separate controller, which deals with communication, into your system. Everything is needed to be inside one small and cheap device. Business requires as low price as possible, because only that it could give you any money.

Embedded networking has constraints that developer should keep in mind while developing a system.

Embedded Network Constraints [A Flexible Approach to Embedded Network Multicast Authentication] Embedded networks usually consist of a number of Electronic Control Units (ECUs). Each ECU performs a set of functions in the system. These ECUs are connected to a network, and communicate using a protocol such as CAN, FlexRay, or Time-Triggered Protocol (TTP). These protocols are among the most capable of those currently in use in wired embedded system networks. Many other protocols are even less capable, but have generally similar requirements and constraints:

- **Multicast Communications** - All messages sent on a distributed embedded network are inherently multicast, because all nodes within the embedded system need to coordinate their actions. Once a sender has transmitted a packet, all other nodes connected to the network receive the message. (In CAN, hardware performs message filtering at the receiver based on content.) Each packet includes the sender's identity, but does not include explicit destination information. The configuration of the network is usually fixed at design time, and changes a little or does not change at all.
- **Resource Limited Nodes** - Processing and storage capabilities of nodes are often limited due to cost considerations at design time. Controllers, that are used usually have no more than 32 kilobytes of RAM and 512 kilobytes of Flash memory. Their operating frequency is no more than 100 MHz. Authentication mechanisms which require large amounts of processing power or storage in RAM may not be feasible.
- **Small Packet Sizes** - Packet sizes are very small in embedded network protocols when compared to those in enterprise networks. The bandwidth is very limited. Network synchronization and packet integrity checks should be added to this. For example data rates are limited to 1 Mbit/sec for CAN and 10 Mbit/sec for TTP and FlexRay. Devices cannot store large packets in memory during processing, as it was mentioned in previous requirement. Authentication should have minimal bandwidth overhead.
- **Tolerance to Packet Loss** - Embedded systems often work in a very noisy environment. Data may corrupt during transmission. Authentication schemes must be tolerant to packet loss.
- **Real-Time Deadlines** - In real-time safety-critical systems, delays are not tolerated. Processes should be completed within specified deadlines. Authentication of nodes must occur within a known period of time. There should not be unspecified delays.

Challenge-Handshake Authentication In this work i decided to use Challenge-Handshake Authentication Protocol [<http://tools.ietf.org/html/rfc1994>].

CHAP is an authentication scheme used by Point to Point Protocol (PPP) servers to validate the identity of remote clients. CHAP periodically verifies the identity of the client by using a three-way handshake. This happens at the time of establishing the initial link (Link control protocol), and may happen again at any time afterwards. The verification is based on a shared secret (such as the client user's password).

1. After the completion of the link establishment phase, the authenticator sends a "challenge" message to the peer.
2. The peer responds with a value calculated using a one-way hash function on the challenge and the secret combined.
3. The authenticator checks the response against its own calculation of the expected hash value. If the values match, the authenticator acknowledges the authentication; otherwise it should terminate the connection.
4. At random intervals the authenticator sends a new challenge to the peer and repeats steps 1 through 3.

The secret is not sent over the link. Although the authentication is only one-way, you can negotiate CHAP in both directions, with the help of the same secret set for mutual authentication.

This protocol is described in the document [<http://tools.ietf.org/html/rfc1994>]. Document specifies main protocol concepts and packet formats.

There are some protocol extensions like MS-CHAP and CHAP is used as a part of other protocols like EAP(EAP MD5-Challenge) and RADIUS (uses CHAP packets). They all use CHAP concepts somehow.

One of the main purposes of this work is to develop a prototype of an embedded service. This system uses JSON [SEE JSON SECTION] object format to encapsulate pieces of information. I will port CHAP packet format to JSON object. It needs to be the same CHAP protocol but it should be placed into JSON. See [CHAP IMPLEMENTATION SECTION] for more details.

Conclusion Information security is a continuing process. There are lots of scientists all over the world, that are trying to invent new approaches how to protect data.

In the Internet-connected future, designers will have to port existing security approaches to embedded control systems. This requires the use of lightweight security protocols.

Embedded control and acquisition devices may be integrated to the main infrastructure of the several organisation. These connections need to be secure enough. Recent decades ago Internet was also a research project, and top computers were like nowadays microcontrollers are. But now it is used in whole world as one of the main communication methods. Even banks are using it for transactions. There are lots of security schemes with different level of protection. I believe that even small 8-bit microcontroller can be securely connected to World Wide Web in the near future.

2.7 Final target system requirements

TODO list of final requirements here. This list is needed to be proved by the implementation

All previous sections introduced ideas how various devices can communicate to each other. Each described technology has its own pros and cons. In general, we need to choose a technology without any drawbacks or a technology, which is the smallest evil chosen from list of suitable ones.

Web Services have these advantages:

- Service description
- Service discovery
- Portability and platform independence(XML)
- Standardized protocols and message structures.
- Ability to transfer messages using other transport protocol than HTTP

The RESTful approach enables to model our domain objects as consistent, hierarchical URLs with predictable operations for **CRUD**(GET, POST, PUT, DELETE). It is also based on HTTP that comes with standard error codes, message types(see also **Resources with multiple representations**) and generally does most of the transport hard work, so we benefit from not needing to maintain any user-developed protocol and using ready and well defined technologies. One of the main concepts of REST is that RESTful services are stateless and do not store session information. This reduces resource consumption and makes client-server applications more loosely coupled and scalable.

REST and Web services have different ideas. REST is based on resources and their representation, while Web Services use messages to send data and call remote methods between server and client. This technology is based on very simple idea of **RPC**. RPC has some essential benefits

over REST and WS-* technologies: it does not require traditional transport like HTTP, TCP, Ethernet or Wi-fi. It can be implemented using any radio link, serial line or any other suitable transport, that is able to deliver response and request messages. Web Services in theory could also be transport independent, but most WS-* tools assume the HTTP (and underlying technologies) as de facto standard³¹. Support of other underlying protocols for SOAP (the main messaging protocol in WS-*) should be implemented separately and most standard tools do not have this, they just use standard HTTP. WS-* has a huge overhead in doing simple things like implementing small light controlling service in your room. This is not right tool for that. Simple RPC would be enough.

Table xxx defines system requirements and features for service prototype in this work. These are ideas that were kept in mind while developing of embedded service system was in progress.

Feature	Description
Transport independent solution	Company did not defined final communication yet. This may be one of these: UART, Bluetooth, RF, HTTP. In this work i implement only first prototype of such system and requirements are about to be changed in future. Therefore i need to implement a portable solution that could be quickly ported to another environment.
Service description and service contract	Gives overview of all service capabilities. Provides an interface definition language. Is a specification(sometimes may be only available one) to the server and server interface.
RPC based communication	Our prototype needs to control coffee machine and execute some requested actions. If we had an Ethernet, the design decision would be REST (with all its benefits and philosophy). Otherwise RPC is the most suitable solution here. This system is message oriented. Some standard RPC solution should be used or ported to the embedded device.
Lightweight and verbose messages	Microcontroller has limited resources. We cannot use huge messages for really small request data (TODO link to SOAP XML-RPC and JSON-RPC AND binary comparison). Some text based protocol need to be used. Binary protocols are evil, it is real hard to understand and debug them.
Simple design	Simple is better than complex. Not like WS-*
TODO	Some more features

Table 4: Ideas about embedded service

TODO In comparison to REST, where resource representations (documents) are transferred, XML-RPC is designed to call methods.

³¹ SOAP protocol transport independence started from SOAP 1.2 [6]

3 System architecture

This section will introduce you a main architecture of the system.

3.1 Introduction

Here will be about coffee machine example in general

3.2 Server architecture

3.3 Client architecture

4 Implementation

4.1 System architecture

4.1.1 Device connection scheme

Devices in a system may be interconnected in various ways. Some embedded systems **do not have** any connections at all. Such systems only sense or control the environment and there is no need to send data somewhere. These are usually highly embedded devices with limited amount of functions (alarm display controller, microwave or washing machine controller).

Another group of devices are systems that are able to interchange information using **proprietary** communication methods at **physical and logical** layers. These systems can send information to another systems, but they are using non standart protocols for data transmission.

Third group uses standard (serial line, ethernet with ip protocol) communication techniques at the physical and transport level and some **proprietary logical** protocol above that.

Last group can integrate with all other systems and has **full communication** possibilities. These systems use standard application protocols and transfer data over well known channels.

Research paper [2] introduces three architectures , that cover three possible ways of connectivity between devices in previously mentioned groups: proxy, translator and full architecture.

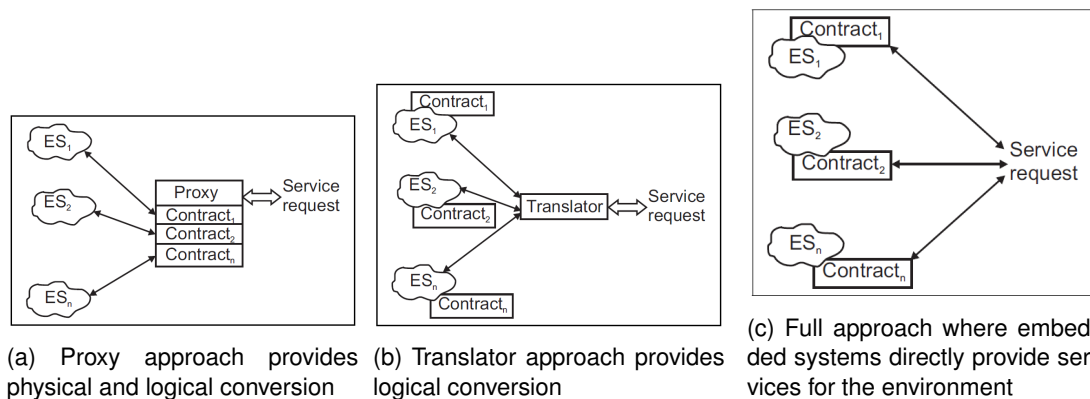


Figure 6: Possible connection architectures for the embedded services

Proxy is a device which is between the client and the embedded service. It provides services to the client and in the same time can communicate with embedded system using closed protocols. Proxy device stores service contracts onboard and know all specifications of connected embedded systems.

Translator approach is similar to the Proxy, but it covers devices with proprietary logical communication. The underlying physical transport is common to client and service provider. The main purpose of Translator is to convert messages that come from clients into into a logical format that the embedded system can understand. Service contracts are stored inside services on the other embedded systems, not on Translator. Translator may not be a separate device and it can be only a software module.

In the Full architecture client and service provider can directly connect to each other without need of any device in the middle.

Our coffee machine system use closed proprietary protocol inside, but all communication messages are transfered over standart serial line. This is more similar to Translator approach, but there is one problem in implementing such architecture. We cannot directly store service contract inside coffee machine system. Coffee machine internal architecture and implementation does not allow us to store any additional code for implementing service functionality. Internal processor is utilized enough and there are no resources for anything else except controlling coffee machine. In addition, the company did not provided the specification for internal mechanism of communication to me. There are several microcontrollers inside that are controlling different machine parts. They provided only the external interface to me, therefore my implementation is more similar to Proxy approach, where contracts are stored inside Proxy machine and Proxy is a separate physical device.

TODO continue with the architecture

Figure 7 shows general system architecture.

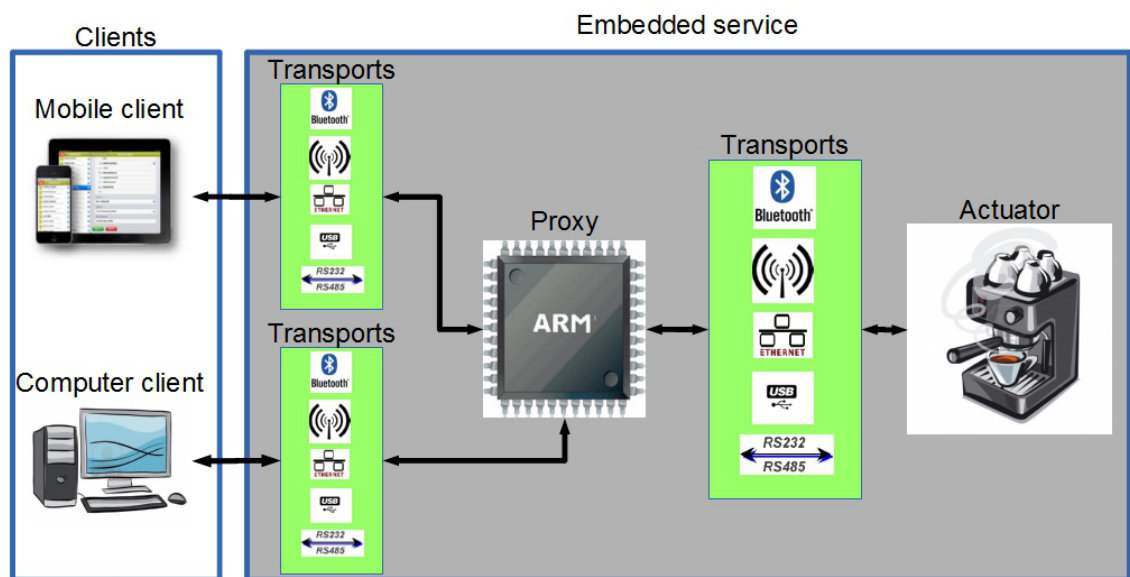


Figure 7: General system architecture

4.2 Implementation of the embedded server

Here will be STM32 server implementation.

One solution is to use closed encrypted proprietary protocol and be calm, but as it was mentioned earlier, it limits the possibility of integration between other embedded systems. In this case all of your devices should support that protocol and your choice of different hardware is limited. Proprietary protocols are often vendor-specific, code is closed, documentation is not free and all it works only with the proprietary devices from the manufacturer.

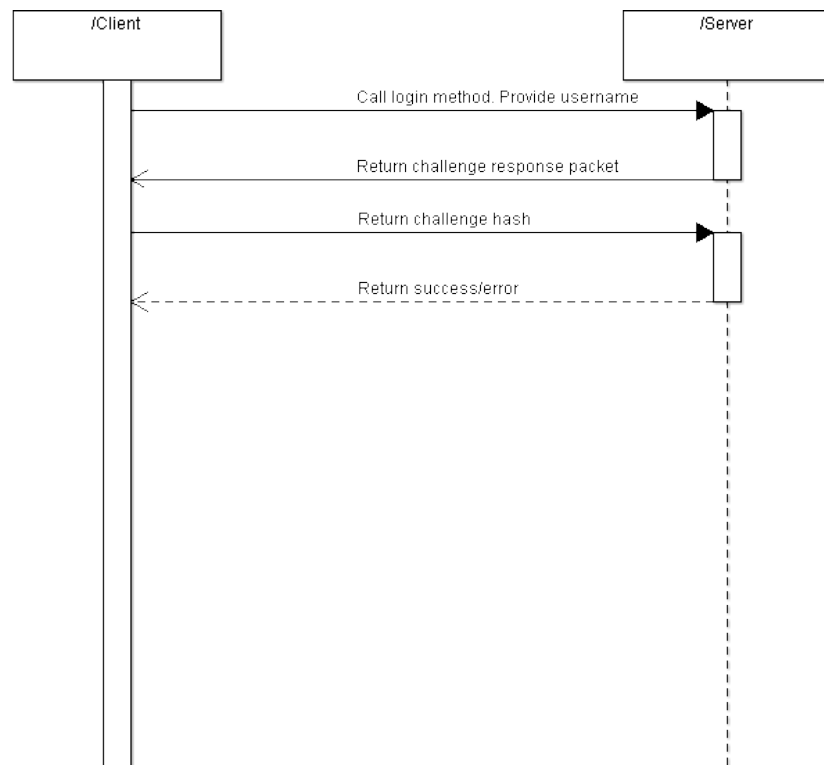


Figure 8: Client authentication process

reversably encrypted form.

4.3 General purpose service library implementation

Here will be general purpose library implementation report.

4.4 Implementation of android client

Here will be android java client implementation report.

Android development Some words about development under Android platform

5 Conclusions

5.1 Results

5.2 Future work

A Examples of service contracts

Example of WSDL contract from WSDL documentation [9]:

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <description
3      xmlns="http://www.w3.org/ns/wsdl"
4      targetNamespace= "http://greath.example.com/2004/wsdl/resSvc"
5      xmlns:tns= "http://greath.example.com/2004/wsdl/resSvc"
6      xmlns:ghns = "http://greath.example.com/2004/schemas/resSvc"
7      xmlns:wsoap= "http://www.w3.org/ns/wsdl/soap"
8      xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
9      xmlns:wsdlix= "http://www.w3.org/ns/wsdl-extensions">
10
11  <documentation>
12      This document describes the GreatH Web service.  Additional
13      application-level requirements for use of this service --
14      beyond what WSDL 2.0 is able to describe -- are available
15      at http://greath.example.com/2004/reservation-documentation.html
16  </documentation>
17
18  <types>
19      <xs:schema
20          xmlns:xs="http://www.w3.org/2001/XMLSchema"
21          targetNamespace="http://greath.example.com/2004/schemas/resSvc"
22          xmlns="http://greath.example.com/2004/schemas/resSvc">
23
24          <xs:element name="checkAvailability" type="tCheckAvailability"/>
25          <xs:complexType name="tCheckAvailability">
26              <xs:sequence>
27                  <xs:element name="checkInDate" type="xs:date"/>
28                  <xs:element name="checkOutDate" type="xs:date"/>
29                  <xs:element name="roomType" type="xs:string"/>
30              </xs:sequence>
31          </xs:complexType>
32
33          <xs:element name="checkAvailabilityResponse" type="xs:double"/>
34
35          <xs:element name="invalidDataError" type="xs:string"/>
36
37      </xs:schema>
38  </types>
39
40  <interface name = "reservationInterface" >
41
42      <fault name = "invalidDataFault"
43          element = "ghns:invalidDataError"/>
44
45      <operation name="opCheckAvailability"
46          pattern="http://www.w3.org/ns/wsdl/in-out"
47          style="http://www.w3.org/ns/wsdl/style/iri"
48          wsdlx:safe = "true">
49          <input messageLabel="In"
50              element="ghns:checkAvailability" />
51          <output messageLabel="Out"
```

```

52         element="ghns:checkAvailabilityResponse" />
53     <outfault ref="tns:invalidDataFault" messageLabel="Out"/>
54 </operation>
55
56 </interface>
57
58 <binding name="reservationSOAPBinding"
59     interface="tns:reservationInterface"
60     type="http://www.w3.org/ns/wsdl/soap"
61     wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/">
62
63     <fault ref="tns:invalidDataFault"
64         wsoap:code="soap:Sender"/>
65
66     <operation ref="tns:opCheckAvailability"
67         wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response"/>
68
69 </binding>
70
71 <service name="reservationService"
72     interface="tns:reservationInterface">
73
74     <endpoint name="reservationEndpoint"
75         binding="tns:reservationSOAPBinding"
76         address ="http://greath.example.com/2004/reservation"/>
77
78 </service>
79
80 </description>

```


List of Figures

1	Web Services roles, operations and artifacts [4]	10
2	Web Services Architecture Stack [5]	11
3	Uniform-Layered-Client-Cache-Stateless-Server [18]	18
4	Principle of RPC between a client and server program	19
5	DPWS protocol stack [36]	26
6	Possible connection architectures for the embedded services	33
7	General system architecture	34
8	Client authentication process	35

List of Tables

1	Objects in WSDL 2.0 [9, 10]	13
2	RESTful web API HTTP methods [21]	16
3	Comparison of binary and human-readable serialization formats	23
4	Ideas about embedded service	32

References

- [1] F. Jammes, A. Mensch, and H. Smit, "Service-oriented device communications using the devices profile for web services," in *Advanced Information Networking and Applications Workshops, 2007, AINAW '07. 21st International Conference on*, vol. 1, pp. 947–955, 2007. 8, 26
- [2] N. Milanovic, J. Richling, and M. Malek, "Lightweight services for embedded systems," in *Software Technologies for Future Embedded and Ubiquitous Systems, 2004. Proceedings. Second IEEE Workshop on*, pp. 40–44, 2004. 8, 33
- [3] Wikipedia, "Service-oriented architecture." http://en.wikipedia.org/wiki/Service-oriented_architecture, 2013. [Online; accessed 17-July-2013]. 8
- [4] H. Kreger, "Web Services Conceptual Architecture." WWW, May 2001. 9, 10, 40
- [5] World Wide Web Consortium, "Web Services Architecture," tech. rep., 2004. [Online; accessed 19-July-2013]. 11, 12, 40
- [6] World Wide Web Consortium, "SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)," tech. rep., 2007. [Online; accessed 19-July-2013]. 11, 32
- [7] Wikipedia, "WS-Security." <http://en.wikipedia.org/wiki/WS-Security>, 2013. [Online; accessed 29-July-2013]. 12
- [8] Distributed Management Task Force, Inc., "DMTF Fact Sheet: WS-Management." <http://www.dmtf.org/standards/wsman>, 2013. [Online; accessed 29-July-2013]. 12
- [9] World Wide Web Consortium, "Web Services Description Language (WSDL) Version 2.0 Part 0: Primer," tech. rep., 2007. [Online; accessed 19-July-2013]. 12, 13, 38, 41
- [10] Wikipedia, "Web Services Description Language." <http://en.wikipedia.org/wiki/WSDL>, 2013. [Online; accessed 17-July-2013]. 12, 13, 41
- [11] OASIS UDDI Specification Technical Committee, "UDDI Spec Technical Committee Draft, Dated 20041019," tech. rep., 2004. [Online; accessed 29-July-2013]. 12
- [12] P. F. A. Albreshne and J. Pasquier-Rocha, "Web Services Technologies: State of the Art," Internal Working Paper 09-04, Department of Informatics, University of Fribourg, Switzerland, September 2009. 13, 14
- [13] R. Elfving, U. Paulsson, and L. Lundberg, "Performance of soap in web service environment compared to corba," in *Software Engineering Conference, 2002. Ninth Asia-Pacific*, pp. 84–93, 2002. 14
- [14] C. Groba and S. Clarke, "Web services on embedded systems - a performance study," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on*, pp. 726–731, 2010. 14, 26, 27
- [15] Wikipedia, "Common Object Request Broker Architecture." http://en.wikipedia.org/wiki/Common_Object_Request_Broker_Architecture, 2013. [Online; accessed 30-July-2013]. 14
- [16] D. Guinard, I. Ion, and S. Mayer, "In search of an internet of things service architecture: Rest or ws-*? a developers' perspective," in *Proceedings of Mobiquitous 2011 (8th International ICST Conference on Mobile and Ubiquitous Systems)*, (Copenhagen, Denmark), pp. 326–337, Dec. 2011. 14, 26
- [17] Bray, Tim, "WS-Pagecount." <http://www.tbray.org/ongoing/When/200x/2004/09/21/WS-Research>, 2004. [Online; accessed 31-July-2013]. 14

- [18] R. Fielding, *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000. 14, 17, 18, 40
- [19] Stefan Tilkov, "A Brief Introduction to REST." <http://www.infoq.com/articles/rest-introduction>, 2007. [Online; accessed 1-August-2013]. 15, 18
- [20] Wikipedia, "World Wide Web." http://en.wikipedia.org/wiki/World_Wide_Web, 2013. [Online; accessed 1-August-2013]. 15
- [21] Wikipedia, "Representational state transfer." <http://en.wikipedia.org/wiki/REST>, 2013. [Online; accessed 1-August-2013]. 16, 41
- [22] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Rfc 2616, hypertext transfer protocol – http/1.1," 1999. 16
- [23] A. Tanenbaum and M. V. Steen, "Distributed systems principles and paradigms," 2007. 18, 21
- [24] Python Software Foundation, "xmlrpclib — XML-RPC client access." <http://docs.python.org/2/library/xmlrpclib.html>, 2013. [Online; accessed 5-August-2013]. 20
- [25] American Arium, "Endianness and ARM® Processors. Application note," 2003. 21
- [26] Malin Eriksson, Victor Hallberg, "Comparison between json and yaml for data serialization," Master's thesis, Royal Institute of Technology, Sweden, 2011. 22
- [27] Wikipedia, "Comparison of data serialization formats." http://en.wikipedia.org/wiki/Comparison_of_data_serialization_formats, 2013. [Online; accessed 6-August-2013]. 22
- [28] Tim Anderson, "Introducing XML." <http://www.itwriting.com/xmlintro.php>, 2004. [Online; accessed 6-August-2013]. 24
- [29] Nurzhan Nurseitov and Michael Paulson and Randall Reynolds and Clemente Izurieta, "Comparison of JSON and XML Data Interchange Formats: A Case Study," in *Proceedings of the ISCA 22nd International Conference on Computer Applications in Industry and Engineering, CAINE 2009, November 4-6, 2009, Hilton San Francisco Fisherman s Wharf, San Francisco, California, USA* (D. Che, ed.), pp. 157–162, ISCA, 2009. 24, 25
- [30] World Wide Web Consortium, "Extensible Markup Language (XML) 1.0 (Fifth Edition)," tech. rep., 2008. [Online; accessed 6-August-2013]. 24
- [31] World Wide Web Consortium, "HTML 4.01 Specification." <http://www.w3.org/TR/1999/REC-html401-19991224/>, 1999. [Online; accessed 6-August-2013]. 24
- [32] Douglas Crockford, "Introducing JSON." <http://www.json.org/>, 2013. [Online; accessed 6-August-2013]. 24, 25
- [33] G. Wang, "Improving data transmission in web applications via the translation between xml and json," in *Communications and Mobile Computing (CMC), 2011 Third International Conference on*, pp. 182–185, 2011. 25
- [34] R. Kyusakov, H. Makitaavola, J. Delsing, and J. Eliasson, "Efficient xml interchange in factory automation systems," in *IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society*, pp. 4478–4483, 2011. 25
- [35] B. Villaverde, D. Pesch, R. De Paz Alberola, S. Fedor, and M. Boubekur, "Constrained application protocol for low power embedded networks: A survey," in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on*, pp. 702–707, 2012. 26
- [36] Web Services for Devices (WS4D), an initiative bringing., "Devices Profile for Web Services." <http://ws4d.e-technik.uni-rostock.de/technology/dpws/>, 2013. [Online; accessed 7-August-2013]. 26, 40
- [37] Internet Engineering Task Force (IETF), "Constrained Application Protocol (CoAP). draft-ietf-core-coap-18." <http://datatracker.ietf.org/doc/draft-ietf-core-coap/>, 2013. [Online; accessed 7-August-2013]. 27

- [38] Wikipedia, "Constrained Application Protocol." http://en.wikipedia.org/wiki/Constrained_Application_Protocol, 2013. [Online; accessed 8-August-2013].
27
- [39] M. Kovatsch, S. Duquennoy, and A. Dunkels, "A low-power coap for contiki," in *Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on*, pp. 855–860, 2011.
27