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. 7

CRUD Create, read, update and delete. 15

FTP File Transfer Protocol. 9, 10

HMI Human Machine Interaction. 5

HTTP Hypertext Transfer Protocol. 5, 10, 13

IPC Inter-process communication. 17

JMS Java Message Service. 13

REST Representational state transfer. 13

RPC Remote procedure call. 8

SMTP Simple Mail Transfer Protocol. 10, 13

SOA Service-oriented architecture. 5, 7, 8, 11, 13, 20

SOAP Simple Object Access Protocol. 5, 10–13

TCP Transmission Control Protocol. 13

UDDI Universal Description, Discovery and Integration. 9–12

URI Uniform Resource Identifier. 14

WSDL Web Services Description Language. 5, 10–13

XML Extensible Markup Language. 5, 10, 11, 13

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

1 Introduction

Computers are very essential in our life. Computer is an electronic device 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.

TODO Internet of Things. TODO smooth transition to SOA and Web services.

One of the methods how this communication can be performed is the concept of web services. 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 WSDL[1]). Other systems interact with the Web service in a manner prescribed by its description using SOAP[2]-messages, typically conveyed using HTTP[3] with an XML[4] 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 SOA[5] can be in the context of embedded systems. This contains some research of already available techologies and the implementation of small system prototype, which uses service approach.

1.1 Impact

The impact of the research in this thesis has been started during the accomplishment of internship at the univercity. I was worked for some company and my task was to develop HMI[6] 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 receiveng and executing that commands.

[1] Web Services Description Language
[2] Simple Object Access Protocol
[3] Hypertext Transfer Protocol
[4] Extensible Markup Language
[5] Service-oriented architecture
[6] Human Machine Interaction

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

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 prepearing, coffee machine maintenance and acquisition of statistical data, remote payment. This could look like traditional coffee automatic machines at the streets that accept cash.

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

TODO system requirements Study available tools and technologies. And

Make a prototype Low cost hardware Transport and platform independent Universal.

1.3 Outline

First section will introduce the concept of web services. 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.

Devices Profile for Web Services

2 Preliminaries

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 incluenced this evolutions [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 standart 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 funcionality 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 goverment 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 API[7]s)

There are available many techologies 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.

[7] Application 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 RPC[8]. 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.

[8] Remote procedure call

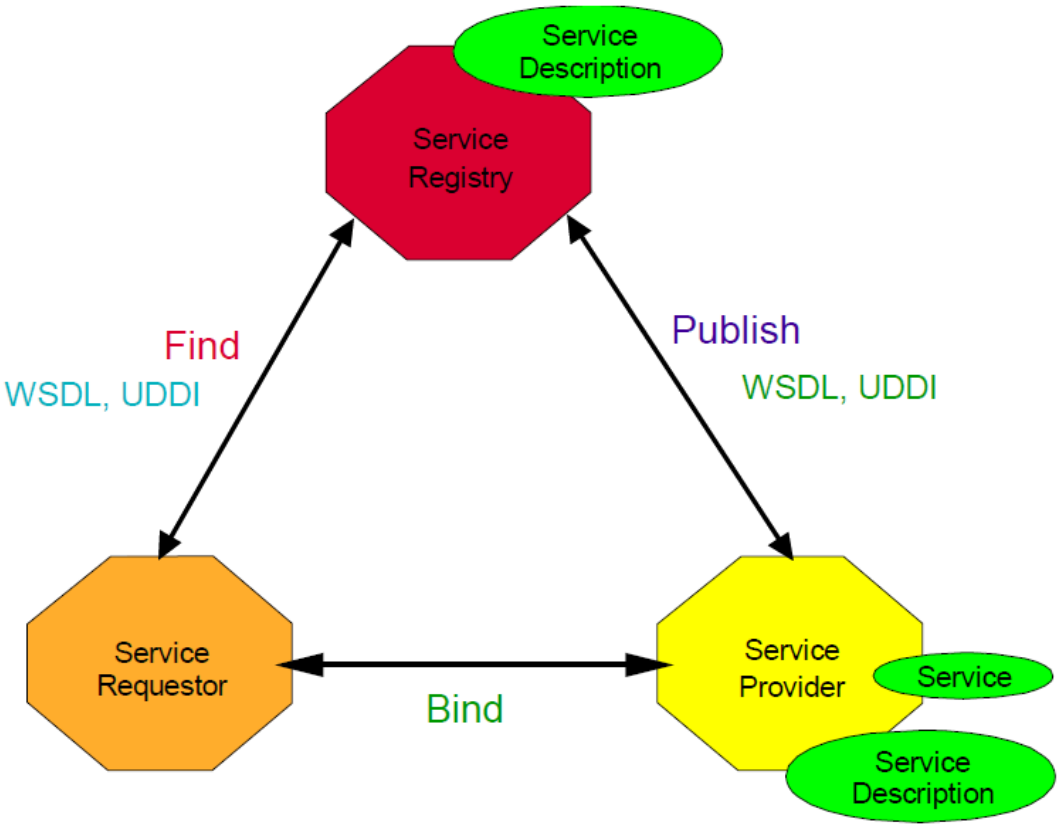

Figure 1: Web Services roles, operations and artifacts [4]

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, FTP[9] 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 UDDI[10] 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.

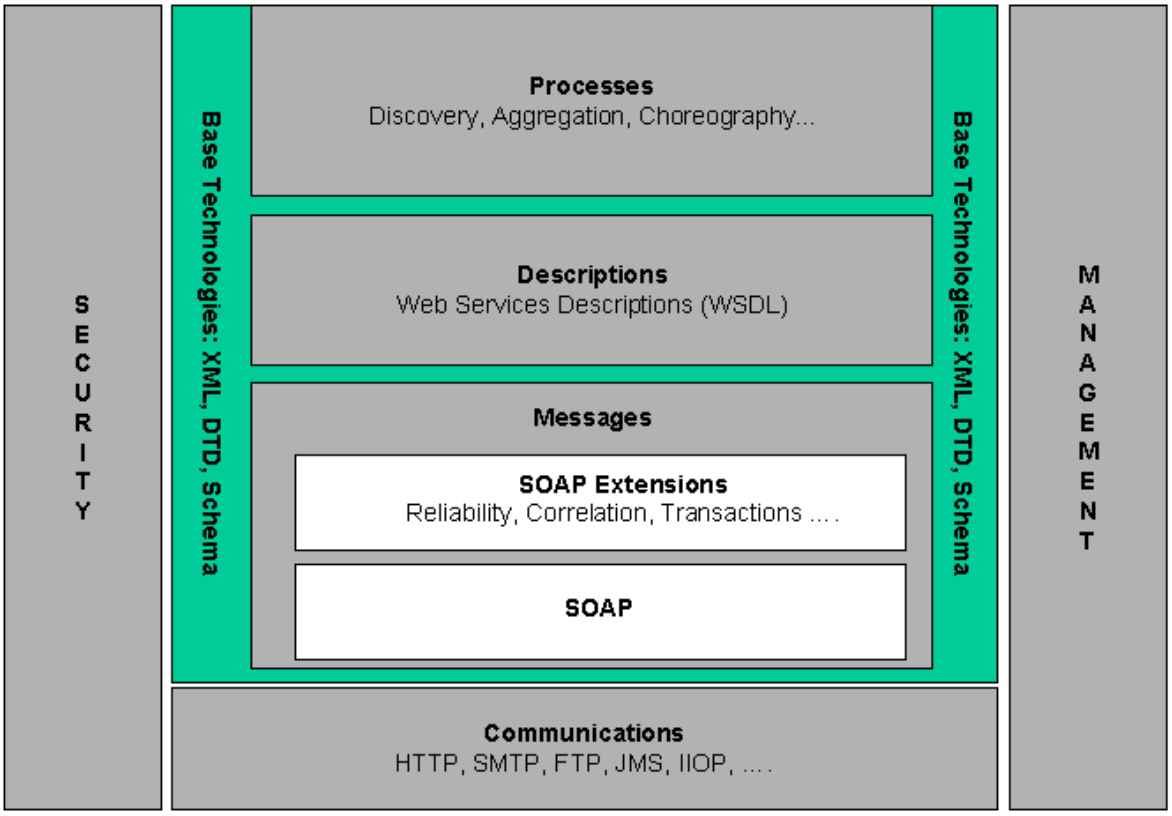

Figure 2: Web Services Architecture Stack [5]

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.

[9] File Transfer Protocol
[10] Universal Description, Discovery and Integration
[11] Simple Mail Transfer Protocol

- **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.

  Web services architecture uses WS-Security[12] 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, subsrcibe to events on resources, execute specific management methods.

  WS-Management was created by DMTF(Distributed Management Task Forse, Inc., http://www.dmtf.org/) organization, which is creating stantards for managing the enterprise level systems. Organisation members are largest hardware and sortware corporations like Broadcom, Cisco, Fujitsu, Hewlett-Packard, IBM, Intel, Microsoft, Oracle. DTMF 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 occures 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:

[12]WS-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]

[13]XML 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 *where* 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 *operations*, 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 *pattern* 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]

> *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 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, TCP[14], or JMS[15]), which can be more suitable to your system environment than HTTP.

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 CORBA [16] or custom made protocol messages. If you start reading WS-* standarts 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 REST[17]. 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-* standarts 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 techical 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 intrested 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* standart 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 resource [18]: post data to create and/or update resource, read data (e.g., make queries) to get current state of resource, and delete

[14] Transmission Control Protocol

[15] Java Message Service

[16] The 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]

[17] Representational state transfer

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

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].

2.3.2 Key principles of REST

REST is a set of principles that define how Web standards, such as HTTP and URI[19]s, 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 identificator for the resource. World Wide Web uses URI identificators for that purpuse. 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 identificator examples [19]

URIs identify resources in a global namespace. This means that this identificator 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 identificator.

Link things together

Previous principle introduced an unigue global identificator for the resource. Resource URI gives posibility 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]

[19]Uniform Resource Identifier

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

Use standard methods

There should be a standard inteface 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 `http://example.com/resources`	**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 `http://example.com/resources/item17`	**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 CRUD[21] 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 identificator `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

[20] The 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]

[21] Create, read, update and delete

be also an another representation of same resouce(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.

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 easilly 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 posibility 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 easilly 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 invormation 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 cachable 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 componens more easilly. 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 optiona 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.

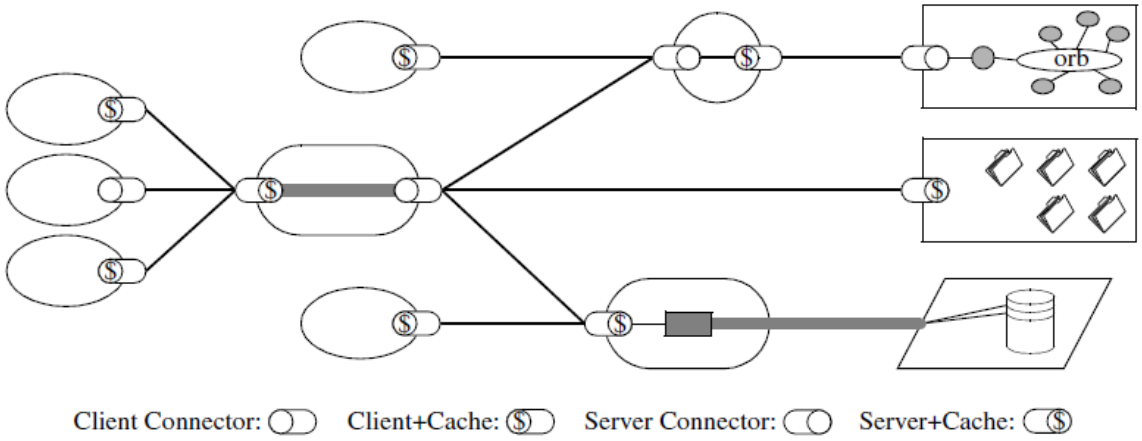

Figure 3: Uniform-Layered-Client-Cache-Stateless-Server [18]

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 techologies (provided above, see subsection 2.1) you can find that many of these techologies 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 techologies are message oriented and use messages for IPC[22]. In RPC messages are sent between client and server to call mathods 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).

[22] Inter-process communication

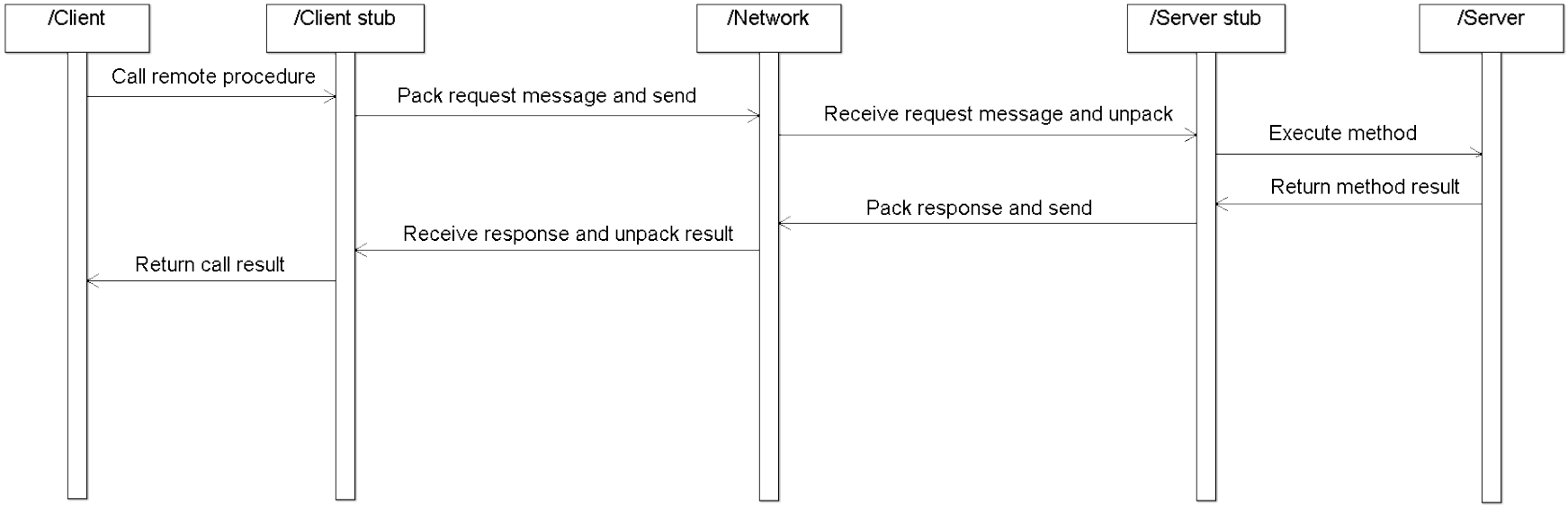

Figure 4: Principle of RPC between a client and server program

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 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 verbose[23] 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:

[23] Python 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.

```
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 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.

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(ASCII[24], EBCDIC[25]), multibyte characters(UTF-8, UTF-8, UTF-32[26],) 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 (one's complement, two's complement) and floating-point numbers.

For example, IBM mainframes use the EBCDIC character code, whereas IBM personal computers use ASCII. As a consequence, it is not possible to pass a character parameter from an IBM PC client to an IBM mainframe server using the simple scheme of Fig. 2-3: the server will interpret the character incorrectly.

DDDDDDDDDDDDDDDDDDDDDDDDDDDD Service - Oriented Architecture can be implemented in different ways. General focus is on whatever architecture gets the job done. SOAP and REST have their strengths and weaknesses and will be highly suitable to some applications and positively terrible for others. The decision of which to use depends entirely on the circumstances of the application.

The RESTful approach enables us to model our domain objects as consistent, hierarchical URLs with predictable semantics mapping (loosely) to CRUD. As well, HTTP comes with standard error codes, MIME types and generally does most of the heavy lifting, so we benefit from not needing to maintain a thick user-developed protocol stack that is subject to frequent modification.

DDDDDDDDDDDDDDDDDDDDDDDDDDDD

2.5 SOA and Embedded Systems

Resent topics had a review of available tools and techologies for implementing a SOA system. Most of them are not directly portable to embedded systems. Most related publications [ ] 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 ration or smaller payload/packet size), use of special constrained protocols, which were designed specially for interaction between small devices. TODO list of suitable protocols.

Main constraints - resources of embedded system. Target STM32 device. Absence of ethernet. Existing implementations require more resources than STM32 or similar has. TODO another constraints of embedded systems.

[24] American Standard Code for Information Interchange
[25] Extended Binary Coded Decimal Interchange Code (from IBM)
[26] Universal Character Set Transformation Format

TODO put is somewhere The benefit of SOA is to allow simultaneous use and easy mutual data exchange between programs of different vendors without additional programming or making changes to the services.

2.5.1 Authentication and Authorization in embedded systems

Need of security TODO some statements from WS-arch about security.

Authorization is the process of granting or denying access to a network resource. Most computer security systems are based on a two-step process. The first stage is authentication, which ensures that a user is who he or she claims to be. The second stage is authorization, which allows the user access to various resources based on the user's identity.

Users are essential part of every system. System should be designed with a requirement, that there will be at least one user. System without any purpose does not make sense. Usually information systems have lots of users with different roles. There should be an system administrator - the most authorized individual in the system, managers and normal users. System should distinct them all somehow.

Another requirement is system and information security. System may contain sensitive data, that should not be available to general users. In case of remote services there are some services that are not open. These services or some of their parts are require some identity to pass through.

Let's take a usual website as example. Common website has at least three different user roles: user or guest, content publisher and system administrator. Last two roles may be joined together, but in general content publishers do not do system maintenance, they just work with content of webpages. There may be more different roles, but these are the main ones. Imagine you open a web page and you see the content. You follow the links and surf the web site. If you want to change something, for example you do not like the design or some words on web page were misspelled, you need to find special place where you can input your **credentials** and get into the system. This will happen only if you have proper **permission** to do that. When you get inside you are still not able to do anything due to lack of privileges. For example you cannot turn of the webserver or disable the your website. There may be lots of different roles and responsibilities in the system and each role has limited access to system resources.

Embedded device as a service may be similar to system example above. Device may have some limited use cases, that are not available to not authorized service clients. This may be internal information retrieving, some device manipulation functions (turn on/off something, delete/remove something from the system, change of system preferences). Some device functionality may be available only for limited people, for example system owner. The real life example of such system is the wireless router. Router clients are other computers, they can send and receive network packets. Router uses wireless security protocols, which permit unauthorized access. Even if you are connected to a secure access point, you are not able to change system settings. You should have admin permission (password) to manage the system. This kind of system, like many embedded systems, is made for one purpose. Router purpose is to provide access for the network. You also can remember lots of similar systems, that use authorized access. Nowadays it is not new to get remotely into some device and to change internals, but embedded system integrations are still not so common. Imagine near future, you are sitting at work and thinking to go home. After a long day you became really hungry. You take your smartphone and connect to your remote wireless fridge service at home. You type your password and get list of all food in your fridge. Now you know what you need to buy and the real candidates to be throwed to the rubbish bin. You adjust power in some fridge area and your beer will be very cold when you get home. Is it just a dream? Is it really hard to realize using present time technologies?

The main problem here is the security. Wired embedded network protocols are mostly not designed with security in mind. These networks were isolated from the Internet in the past. They could only be attacked using direct physical access to the network. Nowadays Internet and wireless networks are popular. Lots of communication between different systems goes through the wireless channel. Radio link is also available to your neighbour behind the wall. Generally, you do not want to broadcast what is in your fridge or to give ability to connect to your air conditioning service. Therefore you need to use some authentication scheme for your service.

Authentication protocols Authentication is any process by which you verify that someone is who they claim they are. (https://httpd.apache.org/docs/2.2/howto/auth.html)

Humanity has already invented a lot of different authentication techniques.

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

- 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 requre to use different security factors together: you say password, provide ID card and show you 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 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 devided into these groups:

- Protocols that transmit the secret over the network. (For example Password authentication protocol). These protocols are not secure.
- Protocols that not send secrets and provide authentification 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 authentification 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 livecycle of average personal computer. Application specific controller may run for decades and it will be still functional. Choosed 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 breaked.

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 sys- tems 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 connnected 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 littel 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 consepts and packet formats.

There are some procol 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 puproses of this work is to develop a prototype of an embedded service. This system uses JSON [SEEE JSON SECTION] object format to encapsulate pieces of information. I will port CHAP packet format to JSON object. It need to be the same CHAP protocol but it should be placed into JSON. See [CHAP IMPLEMENTATION SECTION] for more details.

Conclusion Information security is continuing process. There are lots of scientist 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 sorganisation. These connections need to be secure enough. Resent 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 it the near future.

2.6 Data serialization

2.6.1 JSON

2.6.2 XML

2.6.3 Others

2.7 Final target system requirements

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

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

Here will be implementation report.

4.1 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. It this case all of your devices should support that protocol and you choise 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.

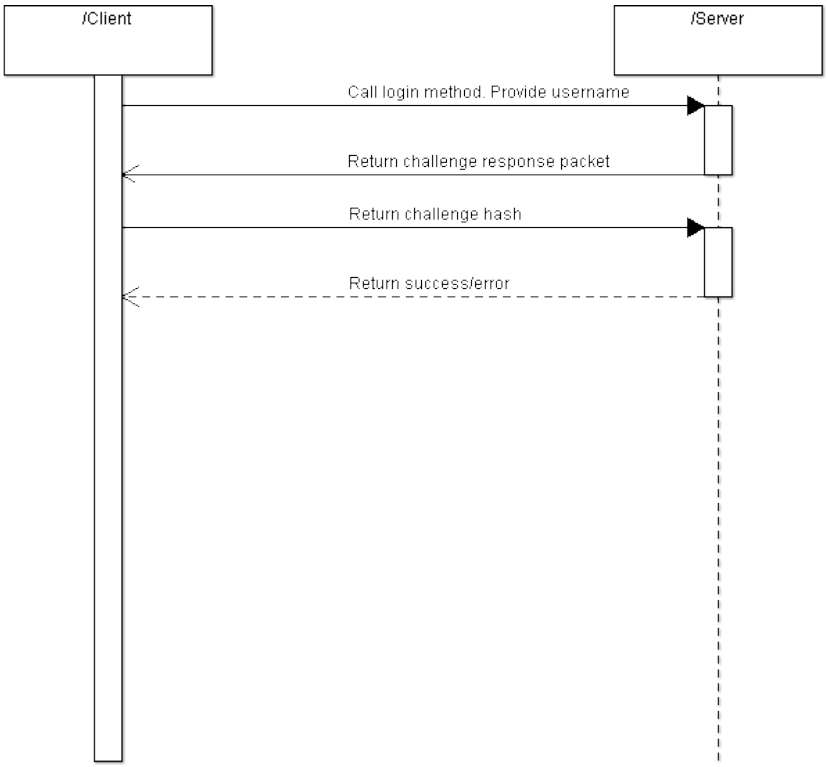

Figure 5: Client authentification process

reversably encrypted form.

4.2 General purpose service library implementation

Here will be general purpose library implementation report.

4.3 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 Future work

A Examples of service contracts

Example of WSDL contract from WSDL documentation [9]:

```
<?xml version="1.0" encoding="utf-8" ?>
<description
    xmlns="http://www.w3.org/ns/wsdl"
    targetNamespace= "http://greath.example.com/2004/wsdl/resSvc"
    xmlns:tns= "http://greath.example.com/2004/wsdl/resSvc"
    xmlns:ghns = "http://greath.example.com/2004/schemas/resSvc"
    xmlns:wsoap= "http://www.w3.org/ns/wsdl/soap"
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:wsdlx= "http://www.w3.org/ns/wsdl-extensions">

  <documentation>
    This document describes the GreatH Web service.  Additional
    application-level requirements for use of this service --
    beyond what WSDL 2.0 is able to describe -- are available
    at http://greath.example.com/2004/reservation-documentation.html
  </documentation>

  <types>
    <xs:schema
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://greath.example.com/2004/schemas/resSvc"
        xmlns="http://greath.example.com/2004/schemas/resSvc">

      <xs:element name="checkAvailability" type="tCheckAvailability"/>
      <xs:complexType name="tCheckAvailability">
        <xs:sequence>
          <xs:element  name="checkInDate" type="xs:date"/>
          <xs:element  name="checkOutDate" type="xs:date"/>
          <xs:element  name="roomType" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>

      <xs:element name="checkAvailabilityResponse" type="xs:double"/>

      <xs:element name="invalidDataError" type="xs:string"/>

    </xs:schema>
  </types>

  <interface  name = "reservationInterface" >

    <fault name = "invalidDataFault"
           element = "ghns:invalidDataError"/>

    <operation name="opCheckAvailability"
           pattern="http://www.w3.org/ns/wsdl/in-out"
           style="http://www.w3.org/ns/wsdl/style/iri"
           wsdlx:safe = "true">
      <input messageLabel="In"
             element="ghns:checkAvailability" />
      <output messageLabel="Out"
             element="ghns:checkAvailabilityResponse" />
      <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

List of Tables

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. 7

[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. 7

[3] Wikipedia, "Service-oriented architecture." http://en.wikipedia.org/wiki/Service-oriented_architecture, 2013. [Online; accessed 17-July-2013]. 7

[4] H. Kreger, "Web Services Conceptual Architecture." WWW, May 2001. 8, 9, 31

[5] W. W. W. Consortium, "Web Services Architecture," tech. rep., 2004. [Online; accessed 19-July-2013]. 10, 11, 31

[6] W. W. W. Consortium, "SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)," tech. rep., 2007. [Online; accessed 19-July-2013]. 10

[7] Wikipedia, "WS-Security." http://en.wikipedia.org/wiki/WS-Security, 2013. [Online; accessed 29-July-2013]. 11

[8] Distributed Management Task Forse, Inc., "DMTF Fact Sheet: WS-Management." http://www.dmtf.org/standards/wsman, 2013. [Online; accessed 29-July-2013]. 11

[9] W. Consortium, "Web Services Description Language (WSDL) Version 2.0 Part 0: Primer," tech. rep., 2007. [Online; accessed 19-July-2013]. 11, 12, 29, 32

[10] Wikipedia, "Web Services Description Language." http://en.wikipedia.org/wiki/WSDL, 2013. [Online; accessed 17-July-2013]. 11, 12, 32

[11] O. U. S. T. Committee, "UDDI Spec Technical Committee Draft, Dated 20041019," tech. rep., 2004. [Online; accessed 29-July-2013]. 11

[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. 12, 14

[13] R. Elfwing, 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. 13

[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. 13

[15] Wikipedia, "Common Object Request Broker Architecture." http://en.wikipedia.org/wiki/Common_Object_Request_Broker_Architecture, 2013. [Online; accessed 30-July-2013]. 13

[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. 13

[17] Bray, Tim, "WS-Pagecount." http://www.tbray.org/ongoing/When/200x/2004/09/21/WS-Research, 2004. [Online; accessed 31-July-2013]. 13

[18] R. Fielding, *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000. 13, 16, 17, 31

[19] Stefan Tilkov, "A Brief Introduction to REST." http://www.infoq.com/articles/rest-introduction, 2007. [Online; accessed 1-August-2013]. 14, 17

[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]. 15, 32

[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. 15

[23] A. Tanenbaum and M. V. Steen, "Distributed systems principles and paradigms," 2007. 17, 19, 20

[24] P. S. Foundation, "xmlrpclib — XML-RPC client access." http://docs.python.org/2/library/xmlrpclib.html, 2013. [Online; accessed 5-August-2013]. 19