Lappeenranta University of Technology
LUT School of Business and Management
Department of Innovation and Software
Master of Computer Science, Software Engineering Major

Roman Arefev

A SOA-BASED FRAMEWORK FOR MULTI-DEVICES AND MULTI-PLATFORM USER INTERFACES

Examiners: Professor Ahmed Seffah
Professor Tatiana Zudilova
Supervisor: Professor Ahmed Seffah
Professor Tatiana Zudilova

ABSTRACT

Lappeenranta University of Technology
School of Industrial Engineering and Management
Degree Program in Computer Science

Roman Arefev

A SOA-based Framework for Multi-Devices and Multi-Platform User Interfaces

Master's Thesis
69 pages, 23 figures, 8 tables
Examiners: Professor Ahmed Seffah
Professor Tatiana Zudilova

Keywords: Service Oriented Architecture (SOA), user interface, multi-platform development

This thesis reports investigations on applying the Service Oriented Architecture (SOA) approach in the engineering of multi-platform and multi-devices user interfaces. This study has three goals: (1) analyze the present frameworks for developing multi-platform and multi-devices applications, (2) extend the principles of SOA for implementing a multi-platform and multi-devices architectural framework (SOA-MDUI), (3) applying and validating the proposed framework in the context of a specific application. One of the problems addressed in this ongoing research is the large amount of combinations for possible implementations of applications on different types of devices. Usually it is necessary to take into account the operating system (OS), user interface (UI) including the appearance, programming language (PL) and architectural style (AS). Our proposed approach extended the principles of SOA using patterns-oriented design and model-driven engineering approaches. Synthesizing the present work done in these domains, this research built and tested an engineering framework linking Model-driven Architecture (MDA) and SOA approaches to developing of UI. This study advances general understanding of engineering, deploying and managing multi-platform and multi-devices user interfaces as a service.

ACKNOWLEDGMENTS

This thesis would not have been possible unless cooperation between Lappeenranta University of Technology and University of Information Technologies, Mechanics and Optics (St. Petersburg). I would like to express the deepest appreciation for the guidance and persistent help to Professor Uolevi Nikula (LUT Department of Innovation and Software), Professor Tatiana Zudilova (Software Development Department ITMO University).

Roman Arefev
Lappeenranta, 19 May, 2015

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

API	Application Programming Interface
CSS	Cascading Style Sheets
ESB	Enterprise Service Bus
IDE	Development Environment
DUI	Distributed user interfaces
HTML	Hyper Text Markup Language
IS	Informational System
MVC	Model–View–Controller
OS	Operational System
PL	Programming language
REST	Representational State Transfer
RWD	Responsive Web Design
SOMA	Service Oriented Modeling and Architecture
SSMN	Service System Modeling Notation
SOA	Service-Oriented Architecture
SOAforMDUI	Service-Oriented Architecture for multi-devices user interface
BPMN	Business Process Model Notation
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SOMA	Service-Oriented Modeling and Architecture
UML	Unified Modeling Language
UDDI	Universal Description, Discovery and Integration
UsiXML	User interface eXtended Markup Language
UIML	User Interface Markup Language
UI	User Interfaces
WSDL	Web Services Description Language
WSRP	Web Services for Remote Portlets

1 INTRODUCTION

This chapter introduces the challenging of multi-platform and multi-devices applications development. It states also the research question, objectives and the research methodology. Finally, the chapter details the content and organization of this thesis.

1.1 Rationale

Developing an interactive system and especially its User Interfaces (UI) is difficult due to the complexity and the diversity of existing environments and to the amount of skills required. UIs account for more than 50% of the total application costs and development time. The difficulties are exacerbated when the same UI is to be developed for multiple contexts of use, such as user preferences, categories of users, computing platforms, and working environments. Various research and development studies highlighted that this diversity of UIs created the following problems that are highly coupled:

1. The complexity of the description (a number of functions, processes, data elements and relationships between them) requires careful modeling and analysis of data and processes;

2. The interrelated components (subsystems) with diverse protocols of communication and data exchange between the functionalities of each sub-system. For instance, traditional applications use transaction processing systems while applications management informational systems such as Web services uses protocols for messaging, for example, Simple Object Access protocol (SOAP) and Representational State Transfer (REST);

3. There are examples of already developed systems, limiting the ability to use any standard design solutions and application systems;

4. The need to integrate on the fly existing and newly developed applications;

5. The operation in heterogeneous environments across multiple hardware platforms and operating systems;

6. The fragmentation and heterogeneity of individual development teams by skill level and traditions of using different tools or program languages;

7. The significant duration of large development project caused the limited abilities of the development team for implementation of Informational System (IS);

8. Daily experience in the industry shows that it is logically difficult, laborious and time-consuming work that requires highly skilled professionals involved in it.

This thesis refers to UI concepts and definitions that are used in this thesis:

- "Multi-device user interfaces (UI) which allow a user to interact with the interactive system using various kinds of computers including traditional office desktop, laptop, palmtop, PDA with or without keyboards, and mobile telephone. Multi-platform user interfaces which UIs that can run on several operating systems including Windows, Linux, and Solaris, if the user interface code is portable."[1] Multi UI (MUI) provides multi-views of the same information to end-users from different computing platforms. A computing platform means a combination of hardware/device, computing networks, an operating system and software development toolkits. The main idea of MUI is to support different types of look and feel for the user. It offers different interaction styles and determines the limitation of each computing platform while maintaining cross-platforms and multi-devices consistency.

- Distributed user interfaces (DUI) has been introduced to deal with the physical repartition of one or many elements from one or many user interfaces in order to support one or many users to carry out one or many tasks on one or many domains in one or many contexts of use, each context of use consisting of users, platforms, and environments [2]. The aim of our research is to making Internet interactive services with a multiple distributed user interfaces secure, yet usable and accessible while incorporating user experiences into the design and engineering loop.

1.2 Research Problem, Objectives

More and more software developers are requested to design, to implement and to validate applications, which are to be run on multi-devices. It could be tablets, smartphones, laptops, PC and etc. It requires operating with different development platforms. Usually, it means that it is necessary to take into account the Operational System (OS), UI including the appearance, programming language. The amount of possible combinations of multi-devices, OS, UI, Programming Language (PL) can be described as Multi-devices = OS * UI * PL. This large amount of combinations of possible implementation requires an army of developers and nowadays it is a problem in software development organizations. The strategic context of this thesis is to find new ways to efficiently design UI that cope with increased complexity and the evolution of the operational use: robustness to organization changes, devices, modalities, operating systems, UI look and feels, etc. This Master's thesis's research addressed the following research question:

> How to simplify the development of the large diversity of multi-devices' and multi-platform UI while ensuring the quality of these interfaces including usability and cross-platform interoperability?

The practical outcomes of this thesis is to face the challenge of lowering "total application costs and development time" is to enhance the interface modelling using an SOA (Service-Oriented Architecture) by adding versatile context driven capabilities that will bring it far beyond the state of the art, up to the achievement of its standardization. There are three practical objectives:

1. Analyze the present approaches to developing of multi-platform and multi-devices applications. Specify architecture models of multi-devices and multi-platform user interfaces framework.

2. Using SOA design principles for re-architecting multi-platform and multi-devices UI in the format a set of interrelated services (SOAforMDUI)

3. Applying and validating SOAforMDUI model using a specific application that can be accessed via different multi-platform devices, for instance, interactive tables, tablets and etc.

1.3 A Brief overview of SOA approach and its benefits

Usually, software systems have a predefined, static, and centralized architecture with firm infrastructure for linking individual operations. On the other side, software applications require increasingly flexible and dynamic means to integrate heterogeneous components [3] such as computational components, context-aware services or multi-device capability.

> "SOA emerged as a response to these challenges. It allows applications to be dynamically composed of individual services and their combinations. Its aim was to meet business demand for IT flexibility by shortening process lifecycle times. "[4]

It is not necessary to design such software system from scratch and reinventing the solutions for each new system. In an SOA environment, all of the IT systems can be presented as services providing particular functions. [5]. SOA includes web-services as units. Web services are web-based applications composed of coarse-grained business functions accessed through the Internet [6]. Web-services become the new step of evolution of object-oriented techniques to e-business.

SOA main advantages are:

- Interoperability by minimizing the requirements for shared understanding
- Enabling just-in-time integration
- Reducing complexity by encapsulation
- Facilitating interoperability of legacy application

1.4 Research methodology

The design research is used as a method to conduct and validate my research objectives. Nunamaker [7] proposed a multi-methodological and iterative approach to information system research. He suggested four stages of research approach:

1. Observation is often used when little is known and it is planned to get common view on research domain. It can help the investigators to generalize the data of experiments or a case study. It will be reviewed the frameworks and approaches for the development of user interfaces for multi-devices applications.

2. Experimentation. Theories are put under testing using various experiments, such as laboratory and field experiments, computer simulations, etc. This stage will include the different blueprints of designs of SOAforMDUI and using of service system modeling notation (SSMN) as a graphical notation.

3. System development consists of 5 stages: concept design, construct the architecture of the system, prototyping, product development, and technology transfer. Product development is a critical step to designing research process as an applied research which aims to produce a working system that helps an organization to solve its problems. Technology transfer means changing new innovations into becoming common and standard. This part will be carried as a case study to illustrate SOAforMDUI.

4. Theory building includes development of new ideas and concepts, and construction of conceptual frameworks, new methods or models. Theories do not have real practical relevance but are the producers of new general knowledge. This point is not described in this work because of the limited volume of gained results and time.

Among limitation of design research Jrad [8] wrote about the lack of rigor and generalisability. The other risk is a potential ambiguity of its demarcation from industrial design effort done by professionals. It often happened that they are interested in a relevant aspect of the solution.

1.5 Structures of the thesis

Chapter 1 introduces the problem of multi-platform and multi-devices user interfaces development. It also states the objectives and the research method. **Chapter 2** takes a look at the common approaches to development multi-platform and multi-devices application. It covers responsive web design, progressive enhancement based on browser-, device-, or feature-detection, markup languages-based approaches, service-oriented approach to UI. It summarizes the advantages and weaknesses of each of these approaches while identifying the best ways/practices to implement the study case. **Chapter 3** details the case study. This chapter describes the methodology and approaches to SOA development. It illustrates how can be designed new system according to the new ideas of SOAforMDUI and shows how services are inter-related. It portrays the usefulness of SOA design pattern for the implementation of SOA application. At the end of Chapter, there is formalization part which shows the number of UI possibilities and allows building a model for assessing the cost of these possibilities. **Chapter 4** explains the implementation of the case study in a sample application using SOA the principles and technology of Enterprise Service Bus (ESB) technology. **Chapter 5** discusses the research methodology, results of the literature review, the implementation of case study and limitations. **Chapter 6** summarizes the research and outlines the future avenues for further research.

In review, the major objectives of my thesis are to investigate the approaches for developing multi-platform and multi-devices applications and to see if such a way is mature and comprehensive to be used in the different context outlined above. My thesis will, in large part, deal with the Service-oriented architecture of software.

2 CURRENT APPROACHES FOR THE DEVELOPMENT OF MULTI- PLATFORM AND MULTI-DEVICES APPLICATION

This Chapter reviews the existing approaches for UI development. The chapter provides a deep understanding of the practices of developing applications for presentation and navigation with a minimum need for resizing, panning, and scrolling to adapt to multiplatform devices constraints and capabilities. The approaches that we surveyed and that are detailed in this chapter are:

Chapter 2.1 describes a Responsive web design (RWD). In this Chapter, it can be found the information about CSS Media Queries and Flexbox layout. **Chapter 2.2** details a progressive enhancement based on browser-, device-, or feature-detection. It is another way to get UI which can change depends on the device. **Chapter 2.3** explains markup languages-based approaches such as User Interface Markup Language and User interface eXtended Markup Language. **Chapter 2.4** is devoted to a service-oriented approach to UI.

2.1 Responsive web design (RWD)

The main idea of RWD is the adaptation of the layout to the screening environment [9]. It uses fluid, proportion-based grids, flexible images, and CSS3 media queries in the following ways [10]. The fluid grid concept calls for page element sizing to be in relative units like percentages, rather than absolute units like pixels or points. Flexible images are also sized in relative units, so as to prevent them from displaying outside their containing element. [9]

Media queries allow the page to use different CSS style rules based on characteristics of the device the site is being displayed on, most commonly the width of the browser. Chapter 2.1.1 details CSS media queries and main media features, which it determines. Chapter 2.1.2 describes another approach to design of layout of UI based on flexible container which lets elements inside to fill available space.

2.1.1 CSS Media Queries

A media query is a part of specification CSS3, which can be found at the official web page of World Wide Web Consortium (W3C). It helps to clarify the scope of the CSS-selector. It determines the unit of output device and media features, such as width, height of the browser window, resolution, orientation in space. It lets the presentation of content be tailored to a specific range of output devices without having to change the content itself.
I had to set different values for a group of blocks marking permits - less than 480px, from 480 to 800, from 800 to 1024, from 1024 to 1280 and more than 1280 [11]. It is obvious that blocks media queries should be placed in order from a lower resolution for more.

As mentioned at [11] the main media types are:

- All - suitable for all devices.
- Print - intended for paged material and for documents viewed on screen in print preview mode.
- Screen - intended primarily for color computer screens.
- Speech - intended for speech synthesizers.

2.1.2 Flexbox

The Flexbox Layout (Flexible Box) module (currently a W3C Candidate Recommendation) is used for providing a more efficient way to lay out, align and distribute space among items in a container, especially when their size is unknown and/or dynamic (thus the word "flex").

> “The main idea behind the flex layout is to give the container the ability to alter its items' width/height (and order) to best fill the available space (mostly to accommodate to all kind of display devices and screen sizes). A flex container expands items to fill available free space, or shrinks them to prevent overflow.
> Most importantly, the flexbox layout is direction-agnostic as opposed to the regular layouts (block which is vertically-based and inline which is horizontally-based). While those work well for pages, they lack flexibility (no pun intended) to support large or complex applications (especially when it comes to orientation changing, resizing, stretching, shrinking, etc.).” [12]

Flexbox layout is most proper to small-scale layouts.

2.2 Progressive enhancement based on browser-, device-, or feature-detection

Another way to get UI which can change due the device, it operates, is using JavaScript facility of device's browser. For basic mobile devices that lack JavaScript, browser ("user agent") detection (also called "browser sniffing"), and mobile device detection are two ways of determination, if certain HTML and CSS features are supported (as a basis for progressive enhancement) - however, these methods are not completely reliable unless used in conjunction with a device capabilities database.

For more precise deducing of mobile phones and PCs features there are special JavaScript frameworks like Modernizr, ResponseJS, and jQuery Mobile that can directly test browser support for HTML/CSS features (or identify the device or user agent) are popular. These frameworks could be also useful as support for CSS:

> "Polyfills can be used to add support for features - e.g. to support media queries (required for RWD), and enhance HTML5 support, on Internet Explorer. Feature detection also might not be completely reliable: some may report that a feature is available, when it is either missing or so poorly implemented that it is effectively nonfunctional." [13]

Below it is described two of the most prevalent JavaScript frameworks, which were mentioned above:

1. The first JavaScript framework by Faruk Ates and Paul Irish is Modernizr. The main objective is in checking the availability of HTML5 features, which browser can support, for instance, localStorage, WebSQL, and HTML5 drag & drop. A few minor features missing like some HTML5 input types and indexedDB, but all the big ones are there: geolocation, localStorage, HTML5 video and audio. [14]
2. The second JavaScript framework is ResponseJS. In combination with jQuery, it allows to build websites with responsive content. "It can dynamically swap

content based on breakpoints and data attributes. Response's most powerful feature is breakpoint sets. Using the sets, content for more-capable devices and/or larger viewports can be stored in HTML5 data attributes. Designers can create custom sets to make exactly the functionality that they want." [15]

2.3. Markup languages-based approaches

2.3.1 UIML (User Interface Markup Language)

UIML is another approach to multi devices development. It divides UI markup from application code. In the early 90s, HTML gained immense popularity, primarily due to its simplicity. In order to create a small site, it did not require special skills in programming and special tools. Everyone could do it. The progenitor of HTML was Standard Generalized Markup Language. It was developed not only for structuring documents. It was much deeper than mere decoration model for the presentation of the data. The original idea of an ordered structure of distributed data returned with XML and spawned the era of the meta-description of abstract components of web resources. Against this background, the task was separate UI markup from application code. Furthermore, there is the technology of CSS, which paved the way for the creation of a custom design for a specific device interface. These circumstances were a precondition for the creation of UIML. The first specification of UIML was presented by Harmonia in January 1998.

In general, UIML is a concept in which the data path from the application to the physical display device passes through the abstract field of logic, interface and presentation [16]. The interface area includes a description of the structure, style, content and behavior of the elements. The aim of UIML is effectively implementing the interface area.
UIML defines the following [17]:

- Constituent elements of UI
- Modality of UI elements (visual / verbal / tactile)
- Content which is used in the UI (text, images, sounds, etc)
- Reaction of the elements of UI to the user
- Control of events of UI (Java Swing classes or tags HTML)

- External API (Application Programming Interface) for interaction with UI.

2.3.2 UsiXML (User interface eXtended Markup Language)

UsiXML (User Interface eXtensible Markup Language) is a XML-compliant markup language that explains the UI for multiple contexts of apply such as Character User Interfaces (CUIs), Auditory User Interfaces, Graphical User Interfaces (GUIs), and Multimodal User Interfaces.

> "UsiXML consists of a User Interface Description Language (UIDL) that is a declarative language capturing the essence of what a UI is or should be independently of physical characteristics." [18]

UsiXML describes UI elements as widgets, containers, controls, buttons, menus in abstract way, without mentioning of technology, it could be implemented. This way allows the cross-toolkit development. These UsiXML compatible applications are possible to run in all toolkits with the support of UsiXML: compilers and interpreters. Some common features of UsiXML are adapted from [18] in Table 1.

Table 1: General features of UsiXML

UsiXML supports device independence.	UI can be described in a way that remains autonomous with respect to the devices used in the interactions such as mouse, screen, keyboard, voice recognition system. In case of need, a reference to a particular device can be incorporated.
UsiXML supports platform independence.	UI can be described in a way that remains autonomous with respect to the various computing platforms, such as mobile phone, Pocket PC, Tablet PC, laptop, desktop and etc.
UsiXML supports modality independence.	UI can be described in a way that remains independent of any interaction modality such as graphical interaction, vocal interaction, 3D interaction.

2.4 Service Oriented approach to UI (SOA UI)

2.4.1 Main idea of SOA UI

Insightful IT organizations gave a great portion of attention to SOA [19]. The reason is that SOA makes it easier, faster and less expensive than before to deliver essential information to employees, partners and customers. Instead the development of new applications, when a company enters a new line of business or must tackle with new issues, SOA operates with services (such as the UI or the logic to operate business process) available to users across the network.

There is an approach for using SOA not only for providing of data but also as provider of visualizing service for this data, which were obtained from different services. When new business objectives emerge, SOA helps quickly and easily to create new composite application within, for instance, a portal. This application contains only the necessary information.

2.4.2 SOA UI composition Framework

Figure 1 portrays the SOA composition framework which distinguishes the following components [20]:

- Application Template Registry supplies the application specification with UI requirements pending realization. Developers are able to search, modify, upload, and reuse the templates in the registry.
- UI Service Registry includes the service-tized UIs from the UI service provider. It provides UI discovery and matching service to the application developer by publishing the UI profile. It also offers application specification template subscription service to the UI services.
- UI Composition Service plays the important role in the SOAUI framework. It allows combining different services to new one, which satisfies developers' needs.

- Ontology System will provide the relationships and searching for UI-related elements. The ontology should include UI classification information.

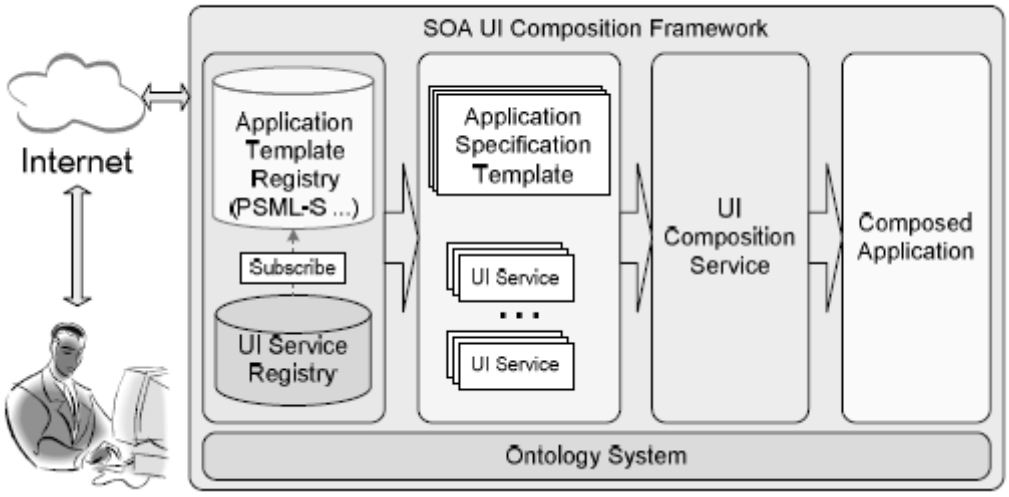

Figure 1: Overall SOA UI Composition Framework (adopted from [20])

2.4.3 WSRP

Web Services for Remote Portlets (WSRP) specification defines a common, well-defined interface to interact with plug-oriented representation of Web-services [21]. These services are engaged in interactions with users and provide code snippets of UI markup language.

These web-services provide transparent access to content and applications, regardless of their location, and during development suggest a simple visual assembly. This is achieved by unifying the specifications of applications, interface and navigation elements. Standards in describing the WSRP interfaces are based on different standards - Web Services Description Language (WSDL), SOAP [21]. Developed on the basis of WSRP 1.0 Web-services can operate on a variety of platforms, including, of course, Java/J2EE and Microsoft .NET.

First of all, WSRP is focused on performance. It means that it provides a UI that allows the end user to interact directly with the application. This is quite different from the traditional Web-services, which focuses on processing a request and generating a response to a software level. Second, the specification describes a common, well-defined interface that

manages the interaction with the portal service and the gathering marking fragments required to represent the page to the end user. This is the common interface that enables applications to use the portal portlets running in remote containers.

WSRP is built upon existing Web-services standards such as SOAP, WSDL and Universal Description, Discovery and Integration (UDDI). Although the most attention is focused on the general usage scenarios of WSRP (generating marking HTML fragments for usage within the portal), nothing prevents WSRP from transportation and other markup languages as it is mentioned below. Figure 2 shows the location of the stack WSRP among Web-services technologies.

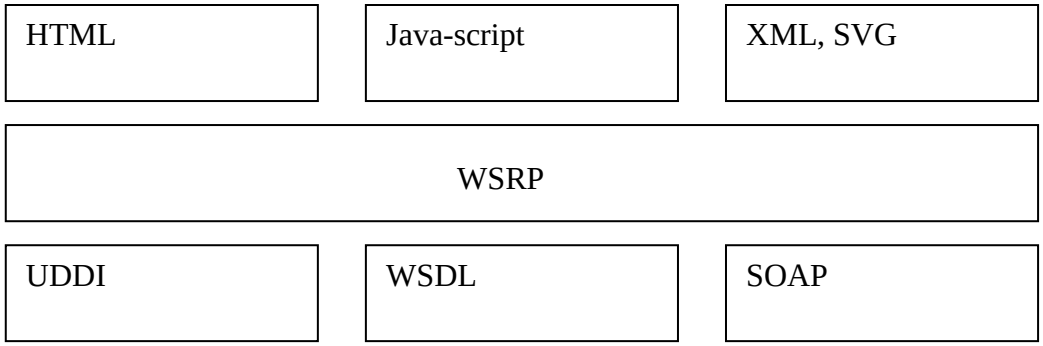

Figure 2: Location of the stack WSRP among Web-services technologies

Why Use WSRP? For web development it is a very attractive to use WSRP for three main reasons [22]:

1. It decouples the deployment and delivery of applications

   You can facade new applications on your portal, independent of release schedule and where and when the code is physically deployed. For example, to get a portlet from one machine to another machine, currently your only method of doing so is to copy the portlet's code, Java Server Pages, and so on, from the first machine to the destination machine. By using WSRP, it is possible to get access and display that portlet on remote machine simply by referencing it through the Producer's Web Service Description Language identifier (WSDL).

2. It delivers both data and logic.

WSRP portlets present both application and presentation logic. It distinguishes WSRP from standard web services, or data-oriented web services, which contain business logic, but lack presentation logic. It requires that every client implements that logic on its own.

3. Its implementation requires no hardcoding.

It is not necessary a lot of programming to make a portlet remote. In a non-WSRP compliant implementation, integrating remote content and application logic into an end-user UI usually takes a significant custom programming effort. Typically, vendors of portals create special adapters for applications and content providers to adopt the variety of different interfaces and protocols those providers use.

Web Services for Remote Portlets (WSRP) is a web services standard. It was developed to "plug-n-play" user-facing web services with portals or other intermediary web applications. The way of implementation is to create a repository of services that users (Consumers) can reference to applications in their portlets as it is shown at Figure 3 or to consume applications from WSRP-compliant Producers, even those removed later [23].

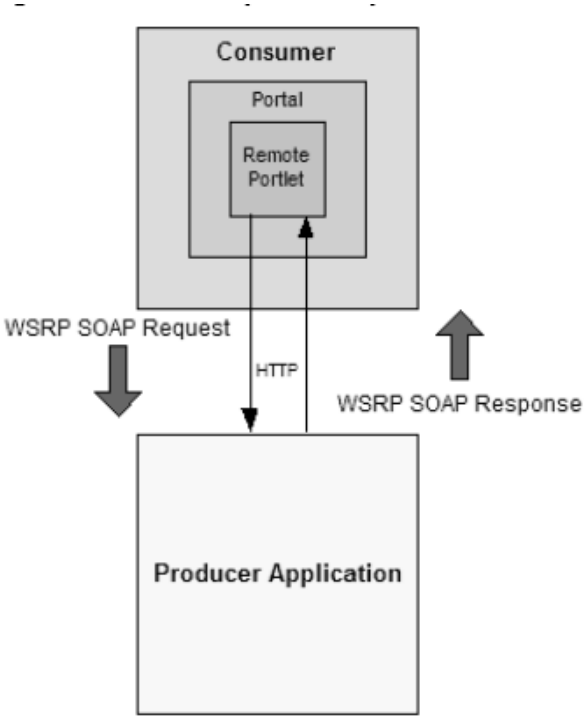

Figure 3: Basic Request/Response flow between Producer and Consumer Application (adapted from [23])

Every Producer of WSRP needs to implement at least two services as you can see from the Table 2.

Table 2: Producer Services (adopted from [23])

Required	**Service**	**Description**
Yes	Service Description	It used to describe the Producer and the portlets that it makes available for Consumers.
Yes	Markup	It manages User interaction with a remote portlet and returns the html markup used to render the portlet.
No	Registration	It allows consumers to register themselves with the Producers. Registration is required for complex Producers.

No	Management	Provided by complex Producers for managing portlets customization and portlet preferences.

These services are described by WSDL format. The Producer provides one URL for all its services. There the Consumer can find the transport-level security information for each service. WSRP messages present themselves over HTTP and use the SOAP message format, but it also requires at a minimum that transport headers to be set in the request message [24]. Consumer must set Actions header, cookie headers, and Content-Type in a header. As the answer, Producer returns a session cookie, application-specific cookies. Consumer must return the session cookie in subsequent request messages.

Portlets

Portlets could be described as pluggable UI software components that are managed and displayed in a web portal.

> "Portlets produce fragments of markup code that are aggregated into a portal page. Typically, following the desktop metaphor, a portal page is displayed as a collection of non-overlapping portlet windows, where each portlet window displays a portlet. Hence a portlet (or collection of portlets) resembles a web-based application that is hosted in a portal. Some examples of portlet applications are email, weather reports, discussion forums, and news." [25]

2.5 Summary

In Table 3 it is inferred about the main approaches surveyed in this chapter. Each of these approaches has advantages and weaknesses. RWD has a small amount of code for support, but browser dependent and for a description of all possible variety of devices it could take a long code. The same way is for progressive enhancement, it also takes to define the devices in additional JavaScript file. The advantage of this approach is the almost full control over OS and browser definition. Markup languages based approach implies the use of middleware and not for all types of devices it could be applied. SOA approach to UI

combines the strengths of markup languages approach with an appliance to SOA principals of loose coupling, autonomy, and composability. It eliminates the weaknesses of RWD and progressive enhancement.

In the proposed framework in the next chapter, we tried to implement the advantages listed here while avoiding the weaknesses we found in existing approaches.

Table 3: Main approaches for the development of multi-platform and multi-devices application

Approach	Description	Key advantages	Weaknesses
Responsive web design	The main idea of Responsive web design is in adaptation of the layout to the screening environment by using fluid, proportion-based grids, flexible images, and CSS3 media queries	- Light weight	- Browser dependent - in CSS file you should describe all possible variation of devices - Large size of CSS file
Progressive enhancement	UI can change due the device, it operates, because of using JavaScript facility of device's browser	- Large range of features of device can be determined	- Browser dependent - in JavaScript file you should describe all possible variation of devices - Large size of JavaScript file - Confrontation

			different JavaScript libraries
Markup languages-based approaches	Interactive applications with different types of interaction techniques, modalities of use, and computing platforms can be described in a way that preserves the design independently from peculiar characteristics of physical computing platform in XML notation.	- Abstract middleware, can be applied to different devices and interaction technologies	- Not all devises support this technology
Service oriented approach to UI	WSRP define to interact with plug-oriented representation of Web-services. These services are engaged in interactions with users and provide code snippets of UI markup language.	- It is continuation of markup languages approach with appliance to SOA principles - Universal approach which lets us to search and compose UI for different devices	- High complexity of implementation

3 MODELING AN SOA-BASED APPLICATION

This chapter describes a case study, which is used to illustrate the main steps in the design and modeling of an SOA-based application. First, it is proposed to use SSMN to model the case study. The Service Oriented Modeling and Architecture (SOMA) methodology presented and illustrated using our SSMN case study. Patterns are discussed to supplementing SOMA methodology. In the conclusion, a formalization of the approach is proposed.

3.1 Overview of case study

First, at 3.1.1 Ii is introduced the proposed case study which will help to illustrate the main contributions of the different approaches which have been studied in Chapter 2.

In the case study, detailed in Figure 4, it is shown an example of how to structure applications using the SOA principles. The example illustrates also the approach for multi-platform and multi-devices development. This application can use different devices and platforms and forms of user interfaces, but it was chosen 3 common used:

- Tablet device with Android or Apple OS. As a style of interaction, we may use direct touch screen capacity or stylus for Samsung.
- Desktop PC. The UI for this device is keypad, mouse and screen.
- Mobile phone devices, like Samsung Galaxy or Apple iPhone under the different OS and also with different interaction modality as voice or graphic.

3.1.1 Problem statement

Every car owner in northern countries needs to change tires for winter seasons. The wear of studs of winter tires can lead to traffic incident. It is important to know the age and condition for changing tires and to make time reservation for the garage visits. It is also necessary to give used tires for recycling to decrease environmental pollution. Presented Service-oriented system helps to check the time for changing the tires and getting the

notification about time to change and coordinate the schedule of garage and places, where you can leave the tires for recycling.

Economic benefits of presented web-service for customers of service: getting the reminder free with info about place and time to change tires, getting a good offer for recycling tires and purchasing new, discounts for utilization. Economic benefits for garages are they have got the clients for changing tires and sell tires. The profit for recycling company is they get the centralized notification system for collecting used tires. Also, they can plan their production according the information from the system about how many tires should be recycled next year. This Service could be shared for utilization of used car oil and so on.

As one of the outcomes of this work is SSMN proposed as graphic notation. The idea is the separation of services according their functions. It was discovered 3 main groups of services:

1. Source of data
2. Services for manipulating data
3. Services for visualizing data

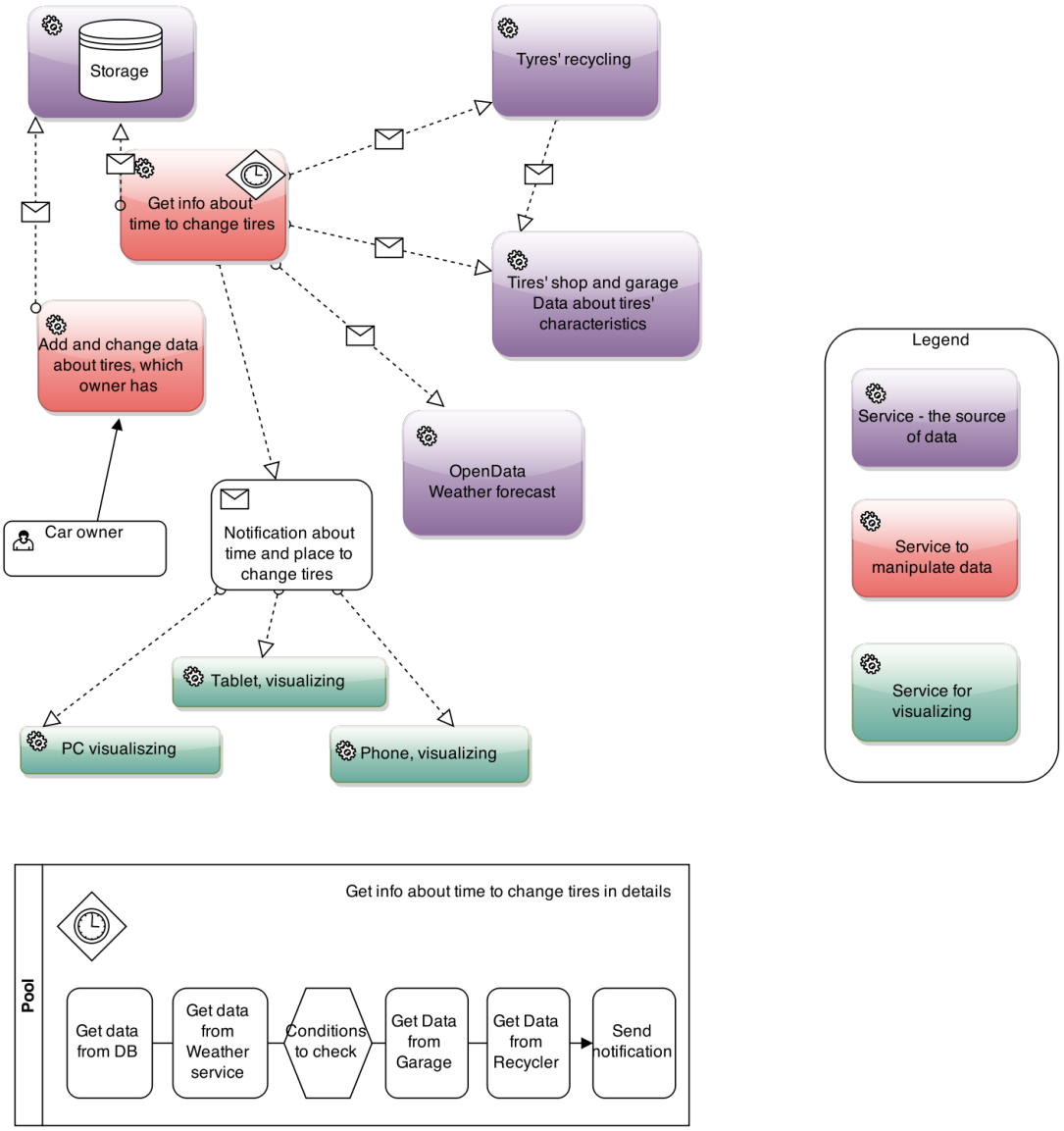

Figure 4: Common architecture of proposed application using SOA principles

3.1.2 Description

The following are the main services that form designed system:

- The Central Service of the distributed system is “Get info about time to change tires in details”. It will collect the info about cars and tires of customers. The servlet will request Open Data Weather forecast source daily about the temperature and check the calendar (1 December and 1 April) to notify the customer about time when it is better to change tires additionally to checking the age of tires.

- As an additional option the Central Service can request the Service of Tires' Shop to check the characteristics of tires and schedule to help customer to make the decision about time to change them.

- The request to Service of Tires recyclers helps to know the place, the time and other necessary conditions to leave the used tires for recycling. The request from Tires recyclers to the Service of Tires' can help to understand the characteristics of tires, the amount of sold tires and their types for balancing marketing and manufacturing.

In building our systems, we have applied the 8 principles of SOA [26]:

1. **Services are reusable.** Described service can be reused by any user who provides necessary information. Additionally any other service which provides mentioned information can reuse the tire service.

2. **Services share a formal contract.** Tire service need exact information to work properly. E-mail to send notifications, current tire type and date of their installation on the car.

3. **Services are loosely coupled.** All services that are used in tire service are loosely coupled because we do not need to know any details about their implementation.

4. **Services abstract underlying logic.** Tire service hides the underlying logic. There is no need to know how it works to get results. User just has to provide necessary information.

5. **Services are composable.** Our service reuses other services. Among them thre are temperature service and service which provides schedule and price.

6. **Services are autonomous.** Tire service has everything that is needed to calculate the best time for changing tires provided that it has received all necessary information according to the contract.

7. **Services are stateless.** Each query to the tire service is separate from each other. There is no need to keep track on state information.

8. **Services are discoverable.** Tire service is going to be included in all related repositories to be found easily if necessary.

3.2 Methodology of developing of SOA application

The efficiency of SOA concerns in providing of business flexibility by integration and reusing individual units of business processes. It is possible through the support of solutions that are based on reusable services in SOA. These services encapsulate functionality which shared with a particular implementation. SOA is also used to provide the management of coupling between the functions.

Modeling connects the business requirements and designed application, based on the use of services. Modeling of SOA development is devoted to a high level of abstraction. It allows designer to focus on business services.

In order to transfer to a SOA, it is important to take into account some additional elements that go beyond service modeling. As described [27] these include:

- Adoption and maturity models. It shows the present situation in enterprise with the adoption of SOA and Web Services. Every different level of adoption has its own uniqueness.
- Assessments. Among questions it concerns how good resulting architecture is or should it keep going in the same direction?
- Strategy and planning activities. It is about the plan to migrate to SOA. It includes steps, tools, methods, technologies, standards, vision.
- Governance. Every service should be created with the intent to bring value to the business in some way.

- Implementation of best-practices by the tried and tested ways of implementing security, ensuring performance, compliance with standards for interoperability.

In addition to identification, specification, and realization, the SOA modeling includes the techniques required for deployment, monitoring, management, and governance required to complete SOA life cycle. An abstract view of SOA can be presented at Figure 5 like a partially layered architecture of composite services which are necessary for business processes.

> "The relationship between services and components is that enterprise-scale components (large-grained enterprise or business line components) realize the services and are responsible for providing their functionality and maintaining their quality of service. Business process flows can be supported by choreography of these exposed services into composite applications. Integration architecture supports the routing, mediation, and translation of these services, components, and flows using an Enterprise Service Bus (ESB). The deployed services must be monitored and managed for quality of service and adherence to non-functional requirements." [27]

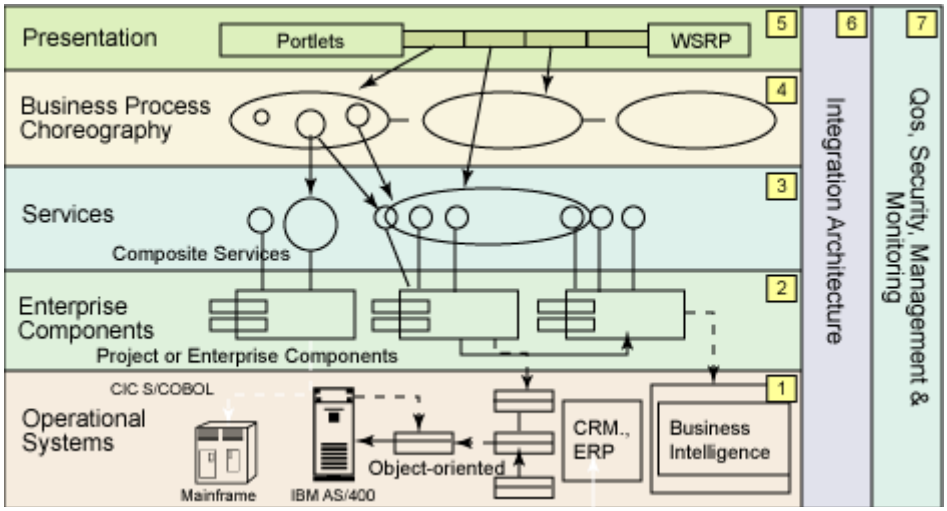

Figure 5: The layers of SOA (adopted from [27])

Every layer needs from designer architectural decisions, usually design documents is divided by sections according layers as wrote [28]:

Layer 1: Operational systems layer. This layer includes legacy system. It means existing Client Relation Management and Enterprise Resource Planning packaged applications, and older object-oriented system implementations, as well as business intelligence applications. The complex layered architecture of an SOA can leverage existing systems and incorporate them using service-oriented integration techniques.

Layer 2: Enterprise components layer. This is the layer of enterprise components that are liable for realizing functionality and upholding the Quality of Service of the exposed services. This layer typically utilizes container-based technologies such as application servers to implement the components, workload management, high-availability, and load balancing.

Layer 3: Services layer. The enterprise components supply service realization at runtime using the functionality provided by their interfaces. The interfaces get exported out as service descriptions in this layer, where they are exposed for use.

Level 4: Business process composition or choreography layer. Compositions and choreographies of services exposed in Layer 3 are defined in this layer. Bundled services act together as a single application.

Layer 5: Access or presentation layer. This layer uses Web Services for Remote Portlets and other technologies, that seek to leverage Web services at the application interface or presentation level. It is very important to note that SOA decouples the user interface from the other components.

Level 6: Integration. The integration of services through the intelligent routing, protocol mediation, and other transformation mechanisms, often described as the Enterprise Service Bus (ESB). WSDL specifies a binding, which implies a location where the service is provided.

Level 7: Quality of Service. This layer provides the capabilities for monitoring, management, and maintenance of security, performance, and availability. This is a

background process through sense-and-respond mechanisms and tools that watch for the health of SOA applications.

According to principles of described model of SOA layers there is software development method Service-Oriented Modeling and Architecture (SOMA).

> "SOMA is a software development life-cycle method invented and initially developed in IBM for designing and building SOA-based solutions. This method defines key techniques and provides prescriptive tasks and detailed normative guidance for analysis, design, implementation, testing, and deployment of services, components, flows, information, and policies needed to successfully design and build a robust and reusable SOA solution in an enterprise." [29]

SOMA is IBM patented method and for the development the SOA architecture they offered to use the following instruments as shown in Table 4.

Table 4: Development process roles, tasks, and tools (adopted from [24])

Role	**Task**	**Tools**
Business Executive	Convey business goals and objectives	IBM® Rational® RequisitePro®
Business Analyst	Analyze business requirements	IBM® WebSphere® Business Modeler
Software Architect	Design the architecture of the solution	IBM Rational Software Architect
Web Services Developer	Implement the solution	IBM® Rational® Application Developer and IBM® WebSphere® Integration Developer

From Table 4 it can be seen that there are several roles in SOMA and each roles has own Software for development. That could say about very detailed and elaborate technique of SOMA.

According to SOMA in design of SOA there are two approaches: top-down, business-driven approach and bottom-up approach, leveraging legacy systems. The top-down approach allows defining large elements in each of SOA layers and determining the critical points at each level. Bottom-up approach concentrates on small-grained services, which can be realized in legacy systems. One of main objectives of such approach is to implement connectors and wrappers.

During the modeling of SOA, it is necessary to decide about architecture of each SOA layer. SOA patterns help to choose the best decision. They are described in Chapter 3.3. For having the balanced SOA architecture it is important to take into account 2 perspectives, one point of view from the service consumer and another one from service provider. At Table 5 it is mentioned the core activities of SOMA process from both views. It is worth to note that provider activities are a superset of consumer activities.

Table 5: Activities of service-oriented modeling (adopted from [27])

Role	**Activities in this Role**				
Consumer view	Service identification	Service categorization	Service exposure decisions	Choreography or composition	Quality of service
Provider view	Component identification	Component specification	Service realization	Service management	Standards implementation
	Service allocation to components	Layering the SOA	Technical prototyping	Product selection	Architectural decisions (state, flow, dependencies)

This work uses the SOMA for design the application, which was described in the case study.

Service identification

Service identification includes combination of top-down, middle-up, and bottom-up approaches of domain decomposition, existing asset analysis, and goal service modeling.
The top-down view is business domain decomposition as it is described at Figure 6. The bottom-up view is an analysis of existing systems. In this work it isn't used. The middle-out view at Figure 7 can be used for validating the services not captured by view mentioned above.

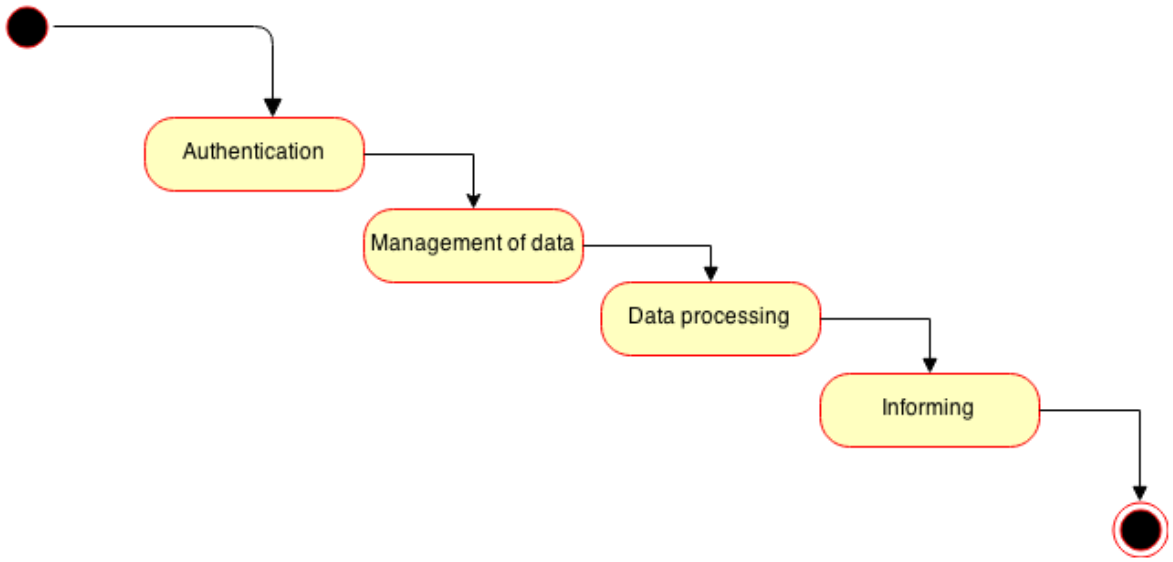

Figure 6: Business Process Model

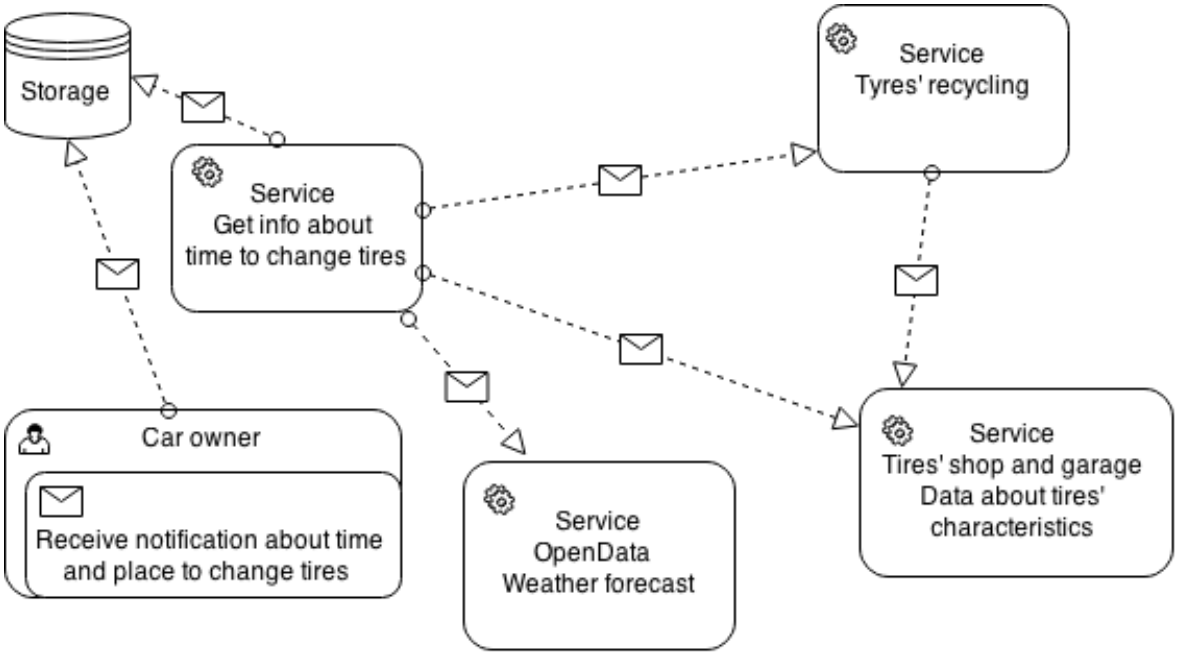

Figure 7: Basic view of system

Service classification and categorization

After identification of service it is time to establish service categorization as at Figure 8. Classification is necessary for composition and layering, as well as it helps building of interdependent services based on the hierarchy.

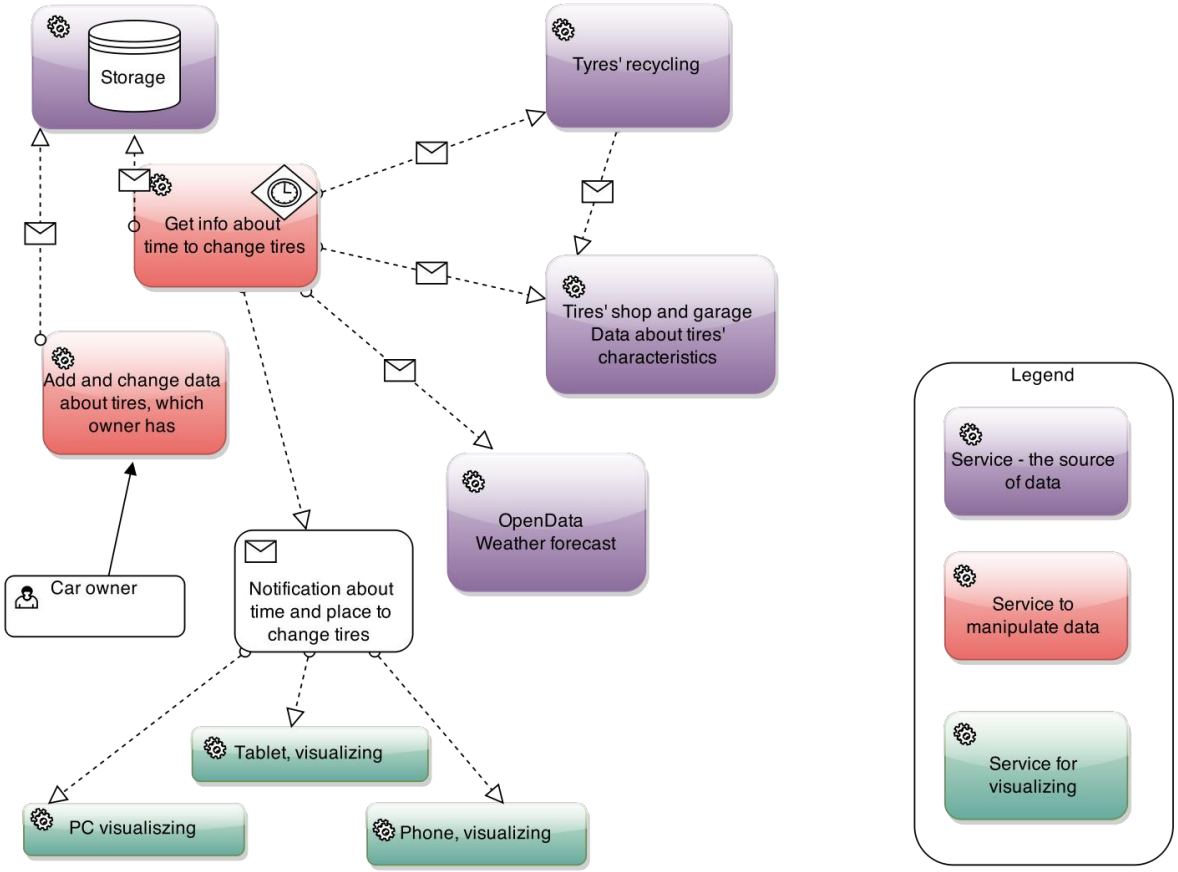

Figure 8: View of system after classification

As it is shown in designed system our services are not allocated at all layers mentioned in theory. It appears because case study system is not as large as enterprise system.

Subsystem analysis

In this stage, it is necessary to define the interdependencies and flow between the subsystems. In the case study, there is not an object model of subsystems which expose services on subsystem interface and realize them.

Component specification

This step assumes that each component, that implements the service, should have the description of data, rules, services, configurable profile, and variations. As far as the case study is small it was decided to describe the main points of description later.

Service allocation

It is assigning services to the subsystems which have enterprise components that implement their published functionalities. Allocation of components and services to layers in the SOA is a key task. It is shown at Figure 9.

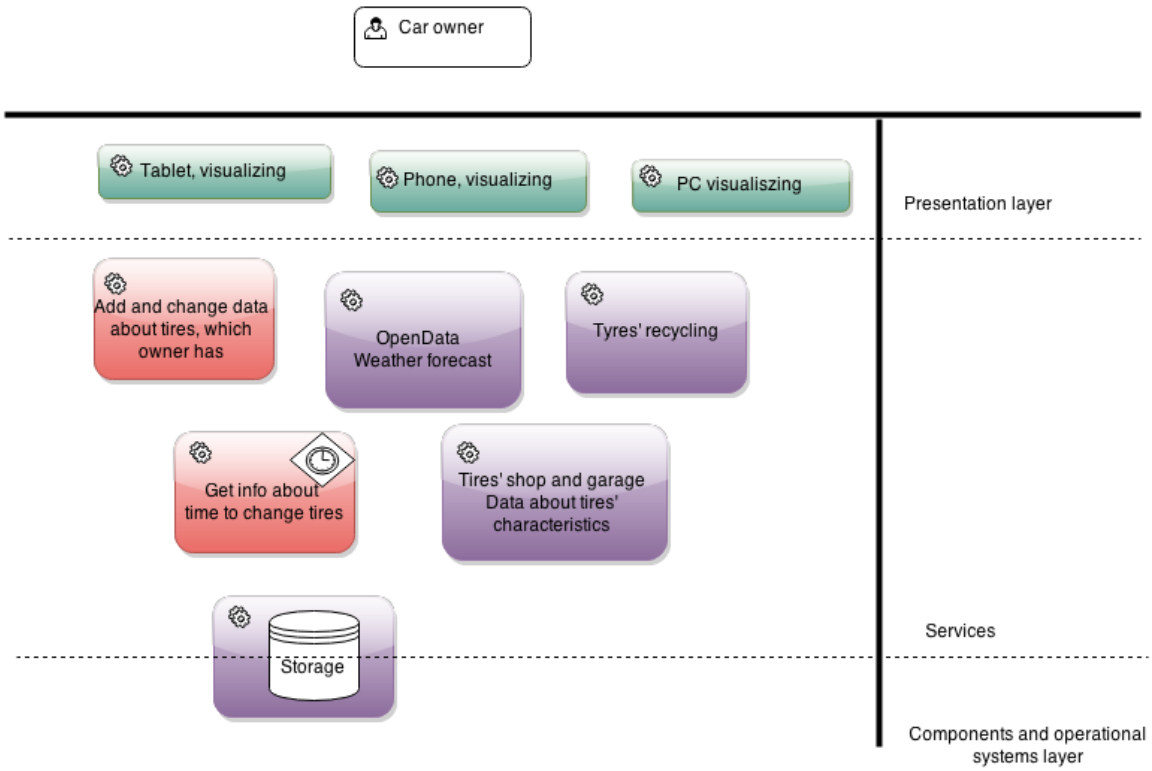

Figure 9: Allocation the components and services to the layers according SOA

Service realization

This step is intended to decide which legacy system will be used to realize a given service and which services will be built. In case of designed system, all services, besides services, which are the source of data (it is planned to use Open Data), are planned to code.

3.3 Using of SOA Patterns. What is it and why?

According to Tomas Erl [30], a design pattern is a proven solution to a common design problem documented using a consistent format.

Erl also stated that:

> "Centralization patterns simply mean limiting the options of something to one. Applying the concept with key parts of a service-oriented architecture establishes consistency and fosters standardization and reuse and, ultimately, native interconnectivity.
>
> Canonical design patterns propose that the best solution for a particular problem is to introduce a design standard. The successful application of this type of pattern results in a canonical convention that guarantees consistent design across different parts of an inventory or solution". [30]

In analysis and design of case study application it was used some patterns of SOA architecture:

a. Capability Composition.

This pattern is applied when the functionality encapsulated by a capability includes logic that can raise other capabilities from other services [31].

The designed Tire recycling service is too large to be implemented within the one single service. In current case, it was separated the functionality of weather request, recycling production and dealer information request from the batch process and part which communicates with user in order to be capable to change easily these services when needed without any major changes in other parts of the entire service ecosystem. Employing this pattern can help achieving the louse-coupling, reusability and composability principles.

b. Service Layers.

This pattern is used in situation when the service models are defined and then form the basis for service layers that establish modeling and design standards [32].

After the separation, it was understood that some of our separate services have functional commonality and can be sorted in layers instead of leaving them in one unordered pile which could decrease overall understandability of the service ecosystem architecture. This way it can increase the reusability of each service because it is easier to find which service fits better for our needs within preliminary dedicated layers and thus there are fewer chances that in case of ecosystem growth any redundancy will appear.

c. Brokered Authentication

An authentication broker at Figure 10 with a centralized identity store assumes the responsibility for authenticating the consumer and issuing a token that the consumer can apply to entrance the service. [33] In this particular case there is no need for user to access all the services separately. The user front end service is dedicated for interaction with users so only one problem remains.

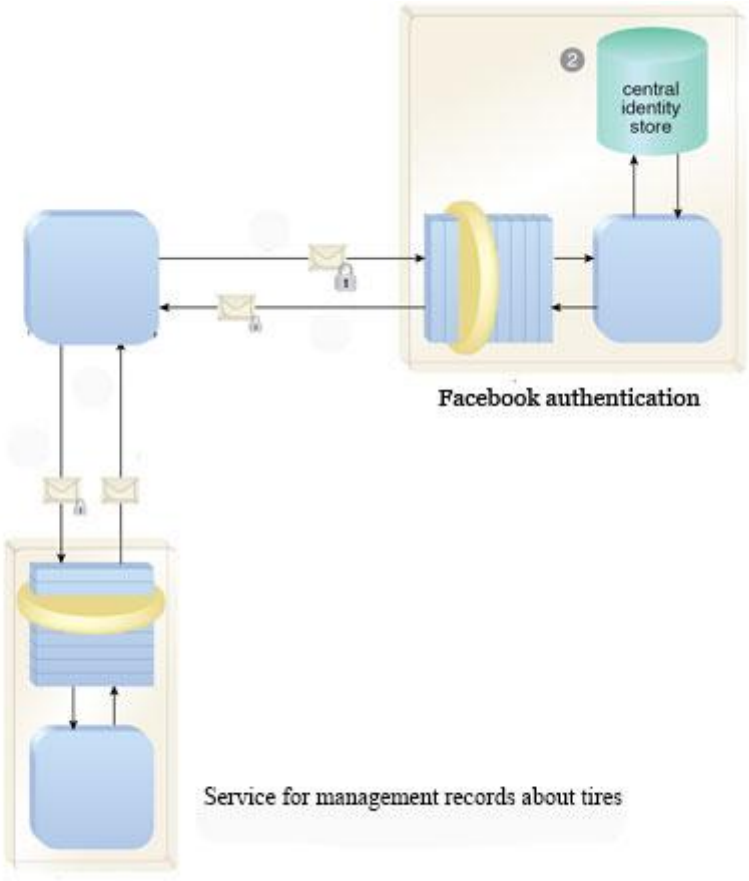

Figure 10: Broker Authentication Pattern applying to case study application

This problem is how we can make sure that any changes that are made to an account are made by its user. The solution to this problem underlies in adding to our service ecosystem one additional component - an authentication broker. For this role it was decided to chose Facebook authentication service which can help any new user get benefits of our service without any need to register them self again if he or she has already a Facebook account. In this case our service remains stateless because we do not have to store user's credentials in it. For this purpose at Figure 10 it was encapsulated them in the authentication broker.

3.4 Formalization of Multi-devices and Multi-platform UI development

The problem addressed in this ongoing research is the large amount of combinations for possible implementations of applications on different types of devices Di. They can be:

- Computing gears from mobile to desktop systems,
- Furniture with computer capacity,
- Sensor and infrastructure devices.

Formula (1) defines D_i as follows:

$$D_i \quad i = \overline{1..n} \qquad (1)$$

In formula (2) DS_j is defined as different operating systems, for example, Windows, Linux, Android, iOS. It can vary from 1 to m.

$$DS_j \quad j = \overline{1..m} \qquad (2)$$

Diversity of humans Du_b is one of the main variables which show the different types of users in formula (3). For instance, they can be persons with high education or children from the primary school.

$$Du_b \quad b = \overline{1..e} \qquad (3)$$

The next variable Dc_a in formula (4) which reflects on diversity of UI, is context variable. It means the place or environment where the UI is usually used. For example, it can be the control room of some factory or control system of container ship or helicopter.

$$Dc_a \quad a = \overline{1..d} \qquad (4)$$

Also it should mention in formula 5 about the different types of multimedia Dm_f – text, image, audio and video content, which we should include into our UI.

$$Dm_f \quad f = \overline{1..4} \qquad (5)$$

One of most important accents is modality and multimodal adaptation of UI at Figure 11.

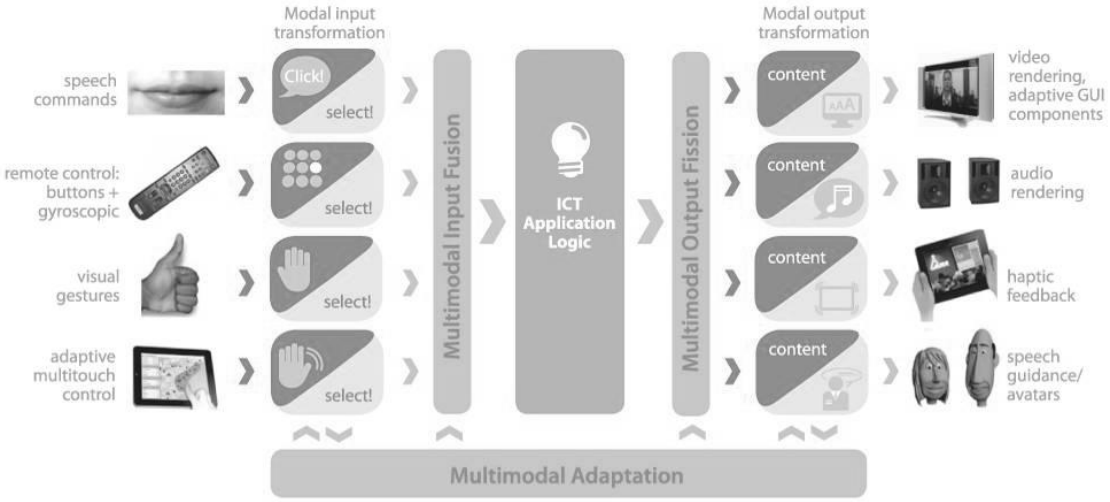

Figure 11: Examples of possible modality for input and output (adapted from [34])

At Figure 11 you can see different approaches to interacting with information and communication devices. For example, for selection we can use speech, fingers, gestures or multitouch control and get output also in different style. It is reflected in formula (6).

$$Dmod_g \quad g = \overline{1..t} \qquad (6)$$

During the development of application and UI it is necessary to take into account the structure or company where this UI will be used. Do_c defines it in formula (7).

$$Do_c \quad c = \overline{1..u} \qquad (7)$$

$UIToolkit_l$ in formula (8) defines different style of UI, it can vary from 1 to k.

$$UIToolkit_l \quad l = \overline{1..k} \qquad (8)$$

Thus for different devices with different OS we can define multiplicity in formula (9).

$$Each \ _{D_i \underset{1..n}{\rightarrow} DS_j} = [D_i \text{ x } DS_j] \qquad (9)$$

Summarizing above it can be deduced the following formula (10) of multiplicity of variants for possible implementation of UI. It can be concluded from this formula, that the number of UI possibilities to create is highly important. It should require an armada of developers to implement and validate all these possibilities. The presented formalization is the first stage to build a model for assessing the cost of these possibilities.

$$Each \ _{\left(D_i, DS_j, Du_b, Do_c, Dm_f, Dmod_g, Dc_a\right) \underset{1..n,m,d,e,u,4,t}{\rightarrow} UIToolkit_l} =$$
$$\sum [(D_i \text{ x } DS_j \text{ x } Du_b \text{ x } Do_c \text{ x } Dm_f \text{ x } Dmod_g \text{ x } Dc_a)] \text{ x } UIToolkit_l \qquad (10)$$

4 IMPLEMENTATION OF AN APPLICATION USING SOA-MDUI FRAMEWORK

This chapter illustrates how the case study has been implemented using the proposed SOA-MDUI framework. It describes the development environment and tools. It also details the implementations of the services. It also presents also various scenarios that illustrate how the proposed application has been used.

4.1 Development environment and tools

In the implementation, it is emphasized on UI using different devices and platforms and forms of UI. It was defined 3 types of UI for the case study:

- Tablet devices with Android or Apple OS. For interaction, we may use direct touch screen capacity or stylus for Samsung.
- Desktop PC. The UI for this device is keypad, mouse and screen.
- Mobile phone devices, like Samsung Galaxy or Apple iPhone under the control of different OS and also with different interaction modality as voice or graphic.

According to the description in Chapter 3.2, it is accented on 2 main areas of the case study system: Service for management of the records about tires and back-end Service of notifications.

Tools for back-end Service, which gets info about time to change tires

The first description is about back-end Service. This Service is not visible to customers and it has no UI. It produces HTML and sends it via Simple Mail Transfer Protocol (SMTP) server. This e-mail can be seen by subscribers on their devices using multi-platform environments and UI.

It is implemented with appliance of ESB technology using the Integrated Development Environment (IDE) Anypoint Studio of MuleSoft. It is very wide-known and rapid ly

developing technology. It allows using the power of Java Enterprise Edition and Spring framework. It is also very easy to use embedded Application server.

Tools for management Service of the records about tires

The second one is a service for management of customers' data about the tires to Storage service. It was used the Node.js technology for server side for API to save data according SOA patterns. As a front-end for entering data I took HTML page developed according Model–view–controller (MVC) model with Java Script Ajax asynchronous technology with appliance of JQuery library.

The Node.js is chosen because it is a fast and modern technology that helps prototyping and has a wide community with a large variety of add-ons. This framework is perfect for the development of scalable and fast applications. One of the additional libraries for brokered authentication via OAuth2, which was added to case study system, uses the authentication service of Facebook.

> "Node.js is a platform that is built on v8, the JavaScript runtime that powers the Chrome browser designed by Google. Node.js is designed to be great for intensive I/O applications utilizing the nonblocking event-driven architecture. While Node.js can serve functions in a synchronous way, it most commonly performs operations asynchronously. This means that as you develop an application, you call events with a callback registered for handling the return of the function. While awaiting the return, the next event or function in your application can be queued for execution." [35]

Service of data storage

For storing data, we use MySQL, because it is open source database with an easy installation and web-interface for administration.

4.2 Detailed description of services

4.2.1 Service for management of records about tires

This Chapter gives the detail view of implementation of services in the case study which was shown in Chapter 3.2.

a. Description of API for storage

Table 6: API for back-end service for storage of records

Description of action	**Route**	**Example of message**

Sending the data about new record	'api/user1' by POST method	{
		"tiresmodel": "Nokian",
		"seasons": 0,
		"type": "winter",
		"login": "roman@mail.com",
		}

Request of 1 record	'api/user1/:item_id' by GET method	

Edition of record	'api/user1/:item_id' by PUT method	{
		"tiresmodel": "Nokian",
		"seasons": 0,
		"type": "winter"
		}

Deleting the record	'api/user1/:item_id' by DELETE method	

b. Implementation of the UI for this service, MVC in JavaScript, JQuery

The MVC pattern aims at separating the application code into three distinct components. This separation means that the application code is easy understand and maintain, the application is also easy to test. MVC allows multiple developers to working on one project. The following are three major components of the MVC pattern [36]:

1. Model, presents model domains in code, including the storage and retrieval of data. The observation pattern can supplement the MVC pattern to trigger an event passing across the updated data after Model changed;

2. View, clusters together code relating to the presentation of the data stored in the Model on screen (it concerns JavaScript approach) - essentially dealing with Domain Object Model of elements. The View uses also observer pattern to look for changes to Model data and update the UI with this new data;

3. Controller contains business logic. It updates the Model and/or View when necessary. Model and View don't need to directly talk to each other, making them loosely coupled from each other.

A view can only read data directly from a model. It cannot set data which is the role of the Controller. It is worth to notice here that in JavaScript applications the Model is most often connected via Ajax to back-end service which acts as a database for stored data represented there.

Ajax is popular choice for developing powerful and interactive websites. This technology bases on asynchronous JavaScript and XML. jQuery is a tremendously popular JavaScript framework created by John Resig [37]. The main objective of jQuery is to support developers to leverage JavaScript across a variety of browsers without low level hardcoding in JavaScript that operates similar across those browsers. The strength of framework includes in its syntax and shortness coupled with its highly extensible nature.

Jesse James Garrett wrote a well-known article called “Ajax: A New Approach to Web Applications” [38] that described the concepts which were used in the presented case study. He reported about an effective way to leverage various browser and server technologies for the development of web pages that are capable of operating and updating in an asynchronous manner. The main idea is not new, but the manner in which Ajax were combined opened a new level of capability (as well as complexity) to the web arena.

Figure 12 presents the sequence of steps for use of the Service for the management of records about tires.

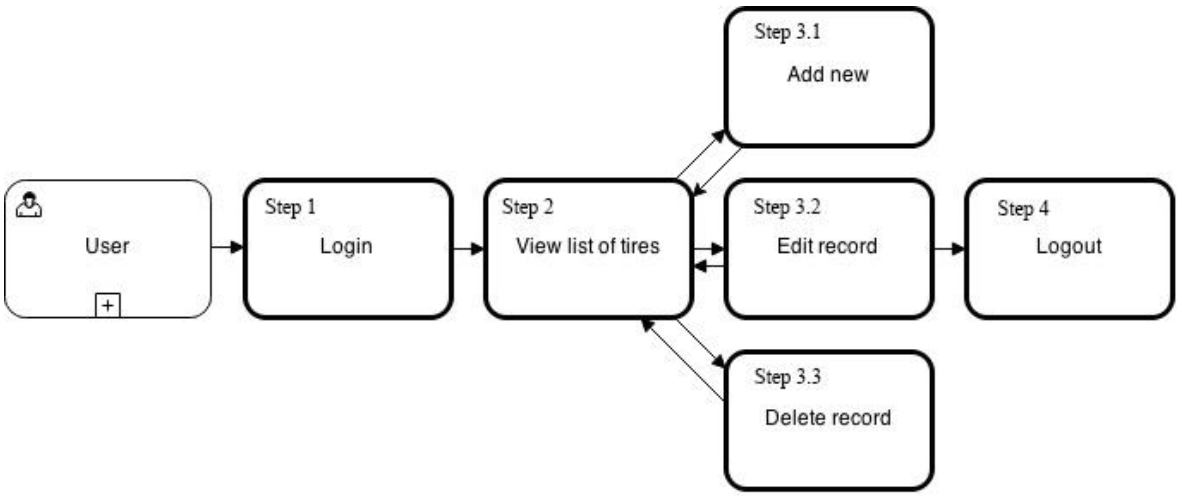

Figure 12: Sequence of steps with the Service for the management of records about tires.

Each of presented steps implies action with data. Below these steps are illustrated on screenshots to show the work of UI. Figure 13 portrays the first step “Login” in the initial page for Login. Figure 14 shows the widget of the Facebook Authentication service from where designed service gets emails of the user. In Chapter 4.2.2 there is a detailed description of implementation.

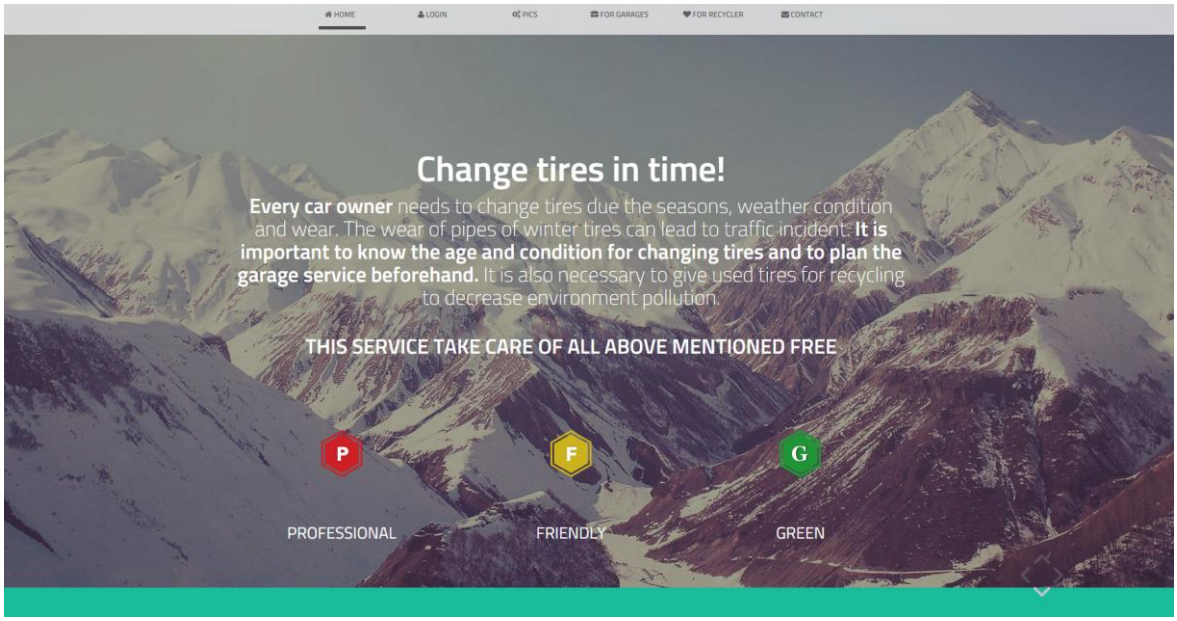

Figure 13: The initial page

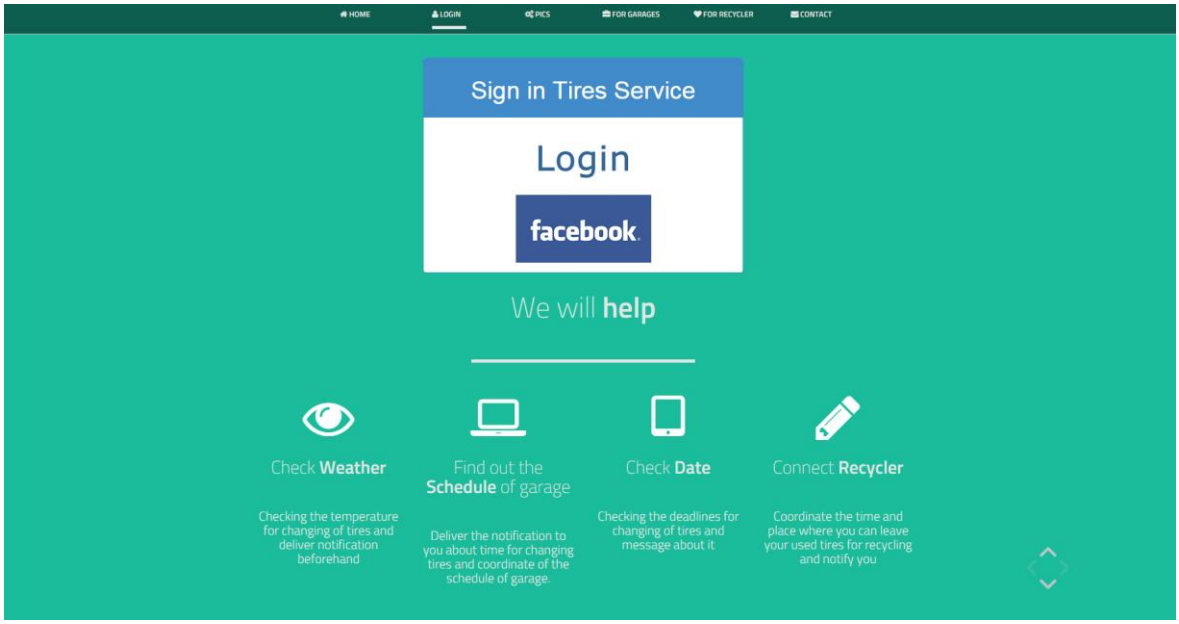

Figure 14: Step 1 – "Login"

The second step "View list of tires" with the UI for managing the record about tires is displayed at Figure 15. At the left part of the UI, the table displays the description of all records that the system monitors. There are the action buttons which can be used for transition to steps 3.2 "Edit record" and 3.3 "Delete record". At the right side of the table, there is a form for step 3.1 "Add new".

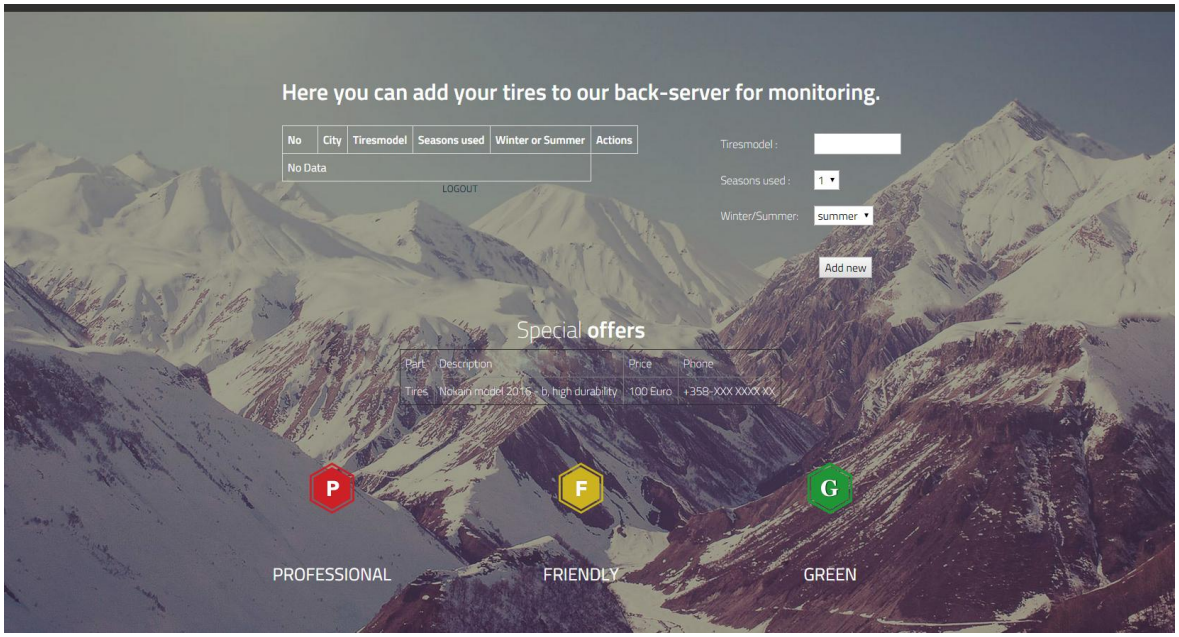

Figure 15: Step 2 – "View list of tires"

Figure 16 shows the results after step 3.1.

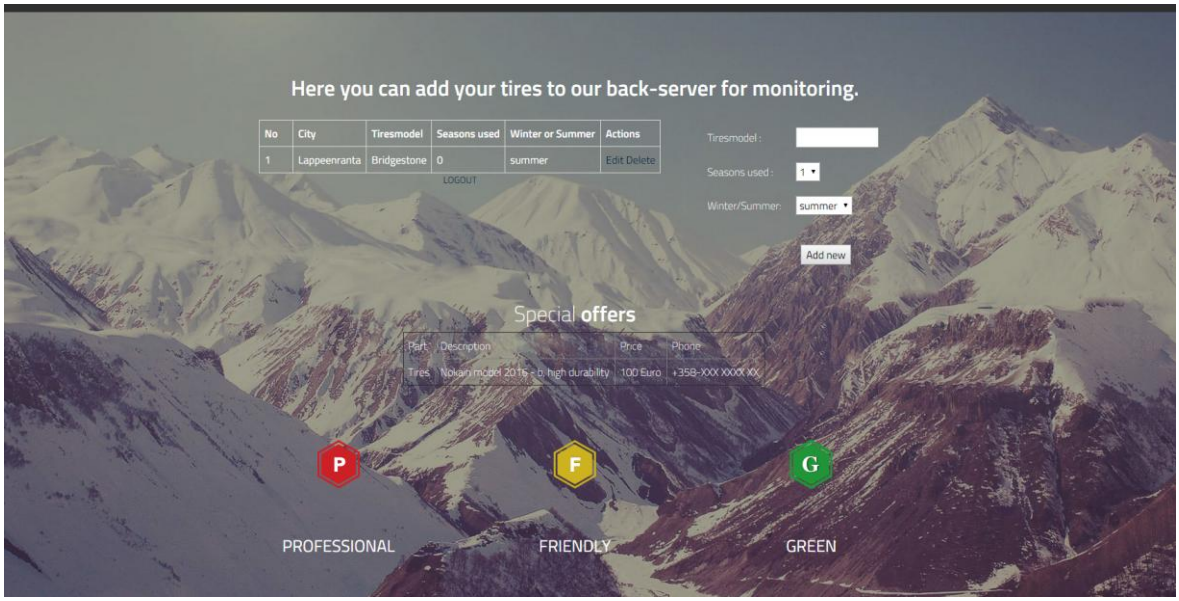

Figure 16: Result of Step 3.1 – "Add new"

Figure 17 displays step 3.2 "Edit record". The standard UI of step 2 appears after edition and deleting (Figure 18).

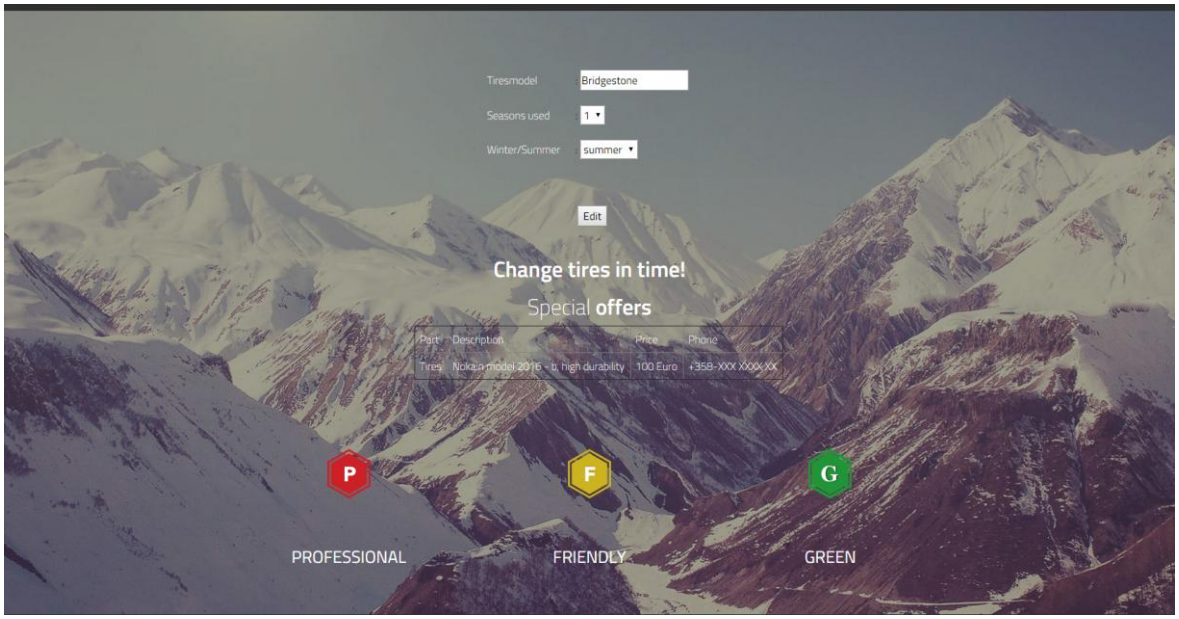

Figure 17: Step 3.2 – “Edit record”

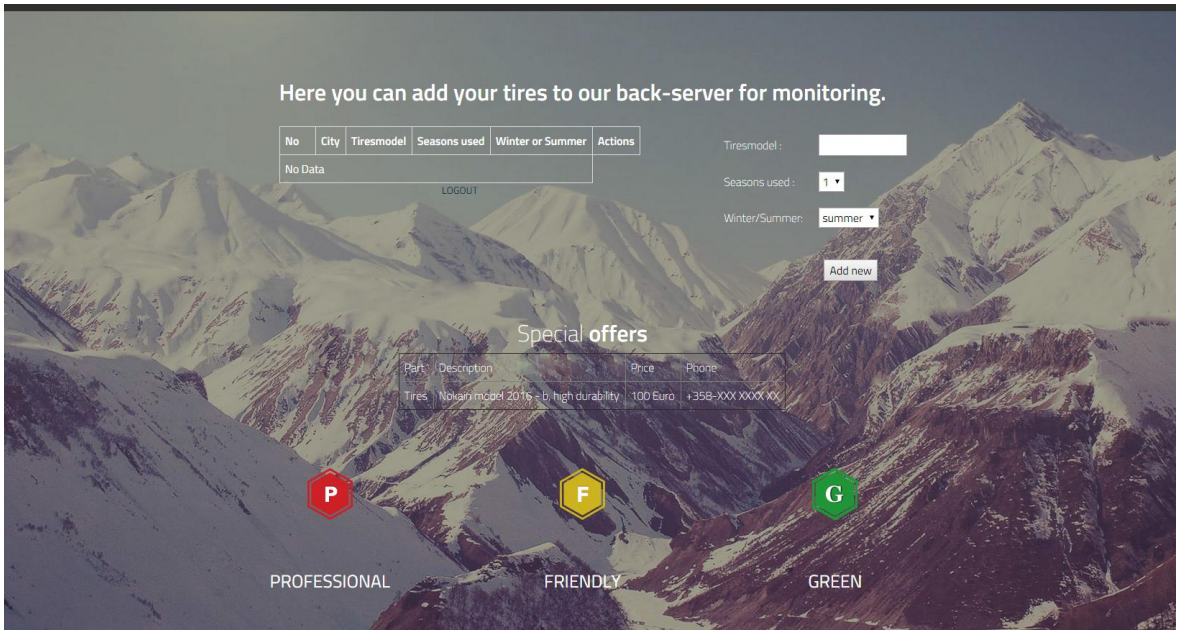

Figure 18: Result of Step 3.3 – “Delete record”

4.2.2 Description of Passport Facebook JavaScript library for authentication via OAuth

Passport is middleware for Node.js. It aims at authenticating request. A technology for authentication has been defined as follows:

> “When writing modules, encapsulation is a virtue, so Passport delegates all other functionality to the application. This separation of concerns keeps code clean and maintainable, and makes Passport extremely easy to integrate into an application. In modern web applications, authentication can take a variety of forms. Traditionally, users log in by providing a username and password.” [39]

OAuth authentication via social pages’ credentials (Facebook, Twitter and etc.) becomes more and more popular because of simplicity and promptness for users. Services that expose an API often require token to protect access. Passport API distinguishes that each application has unique authentication requirements. Authentication mechanisms, known as strategies, are packaged as individual modules. These modules are available for developed application without creating unnecessary dependencies.

Below is the code that shows how passport has been used in the main routes for authentication function. The function implemented also the checking if the User is authenticated on the route ‘api/user1’.

```
app.get('/auth/facebook', passport.authenticate('facebook', { scope: [ 'email' ] }));
app.get('/auth/facebook/callback',
  passport.authenticate('facebook', {
      successRedirect : '/api/user1',
      failureRedirect: '/login'
  })
```

4.2.3 Back-end Service, which gets info about time to change tires

Possibly one of the first widely accepted open source integration frameworks, Mule ESB was the result of Ross Mason’s system integration consulting work looking for a improving of efficiency and maintainability of custom integration activities. It has been defined as follows:

> “Philosophy and Approach Mule’s approach was to create a lightweight container that can run standalone, without requiring the support of a J2EE application server

where the integration component can be deployed. Although Mule can also be deployed within an application server, the standalone mode is the recommended approach, eliminating the overhead of a heavyweight container." [40]

Mule ESB includes out-of-the-box service components, which are responsible for message routing and data transformation. The format of Mule's messages may be in many formats from SOAP to binary. Adapters (or transports, as they're called in Mule), supports everything from Java Database Connector to Short message peer-to-peer protocol.

Figure 19 shows the main flow which gets the records from Storage Service and additional data from data services.

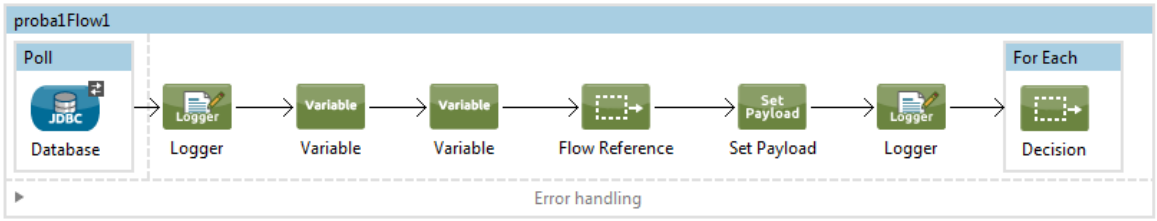

Figure 19: The main flow which gets the records from Storage Service

It is necessary to get the list of data from Storage or Database. This process is launched once every day. After that, it adds some variables to payload of messages for technical proposes. Figure 20 shows the supplementary flow which marked on the main flow as Flow reference. GetTemp is necessary to get the data via REST request.

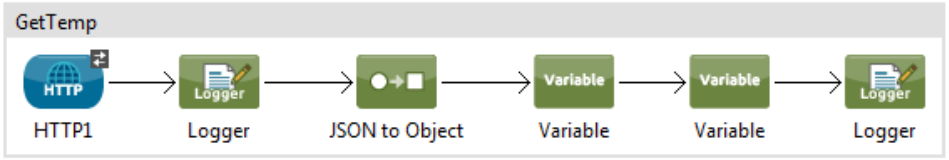

Figure 20: The supplementary flow GetTemp

REST technology is typically referred to as an architectural style [41]. It can be described as Stateless, Client-server, and cacheable communications protocol over HTTP. RESTful application use HTTP requests to handle the usual CRUD-operations (Create, Read, Update, Delete). REST uses URL routes to communicate with the server.

As it is shown at Figure 21, after the data is acquired from the flow returned to the main branch and each object or record in payload of message is transmitted to Supplementary flow Decision.

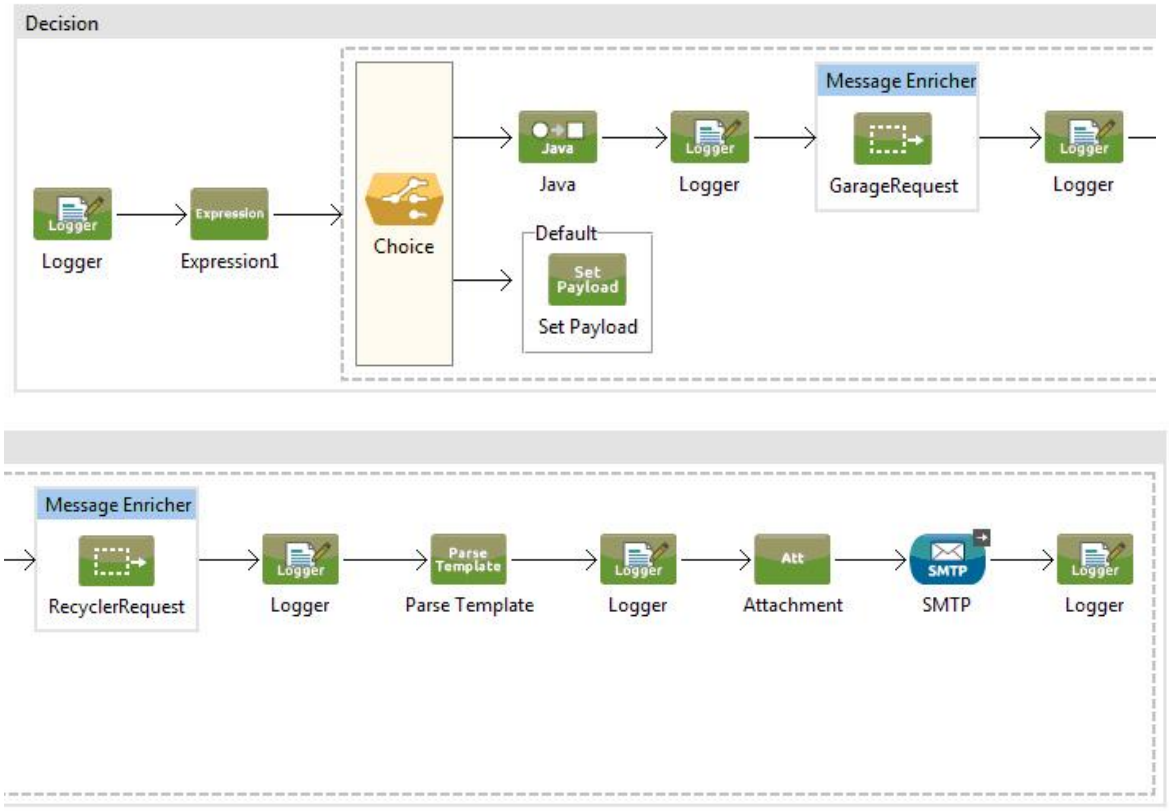

Figure 21: The flow of decision (the left side of bottom picture follows the right side of the picture above)

There are 4 conditions and each record should be checked on them.

1. If Type field of tires is “summer” and the current date is more than the dates - the 1st of December.
2. If Type field of tires is “winter” and the current date is more than the dates - the 1st of April.
3. If the next 3 days the temperature will be less than 4 degrees bellow zero.
4. If Season field is more than 4 and Type field is “winter”.

Even if, one of the states above is true, the message goes alternatively through two flows as it is pictured at Figure 22. There you can see the request of data from SOAP services about time at Garage and from Recycler production.

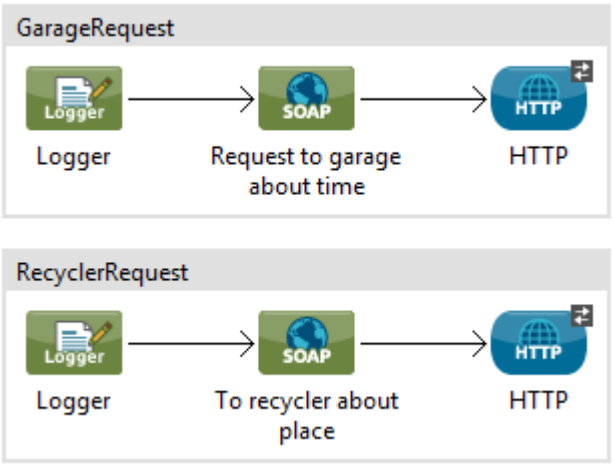

Figure 22: The flows for request to outward services

The definition was identified and maintained by the XML Protocol Working Group of the World Wide Web Consortium. It is defined as follows [42]:

> "SOAP is an application-level protocol standard used to transport messages in distributed systems."

SOAP was originally intended as the "Simple Object Access Protocol" and proposed to be used with traditional distributed object technologies such as remote method invocation [43]. Later SOAP became the most often used for Web services for which SOAP messages are encoded using XML and intended to carry XML encoded application data. SOAP messages contain both a header, for infrastructure data, and a body for application data. OAP layers on top of HyperText Transfer Protocol. It makes easy to deploy SOAP services behind network firewalls. After all data are gathered from Web Services, they are entered to html template and as attached file it is sent via SMTP server to the subscriber or user of server.

4.2.4 Storage service

The storage service is represented by MySQL Relational Database, as schema of project is shown in Figure 23.

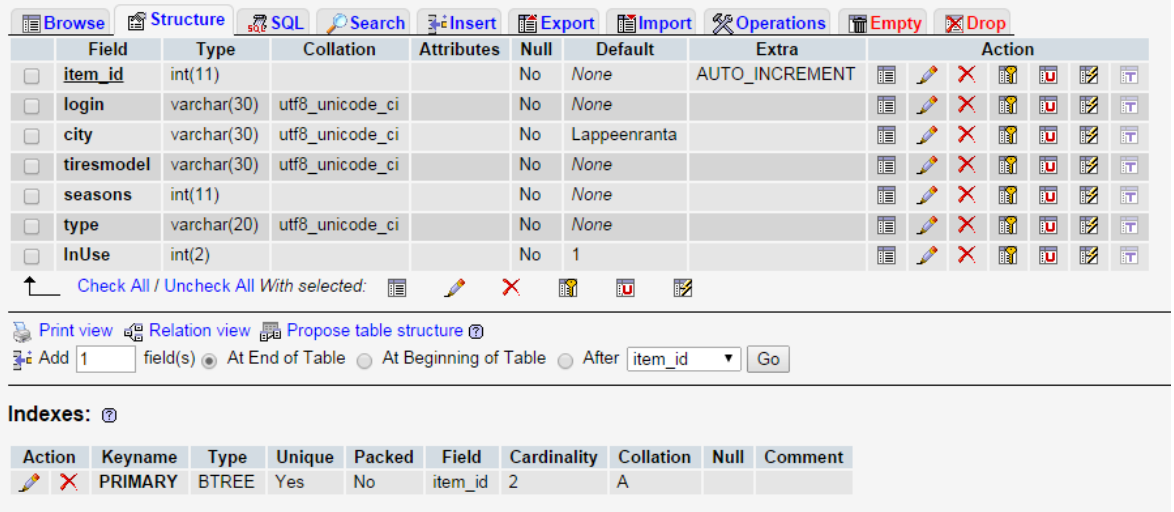

Figure 23: MySQL database schema

4.3 Main scenarios of using the example application

Scenarios can be a good way in description of application. A scenario describes the use of a system from the perspective of users. In the scenario-based design approach, scenarios are used as a medium for involving users as well as for specifying user needs and driving the entire user-centric design process. [44]

In Tire Recycling Service, scenarios can be used to describe the main functions of service. It was classified scenarios in Table 7. It can be a story (elaborated stories) that captures the goals, intentions, and reactions of the user and the context in which the system will be used. Scenarios can also describe the user's work or goal (goal-driven scenario) or can describe the details of a specific task including the interactions with the system, actions of users and reactions of the system [45].

Table 7: Functions and used scenarios

Functions	**Definition**	**Example**
View tires added to the system	User uses the service for managing records about tires and after authentication, he or she	*Mr. Paterson lives in Lappeenranta. He has family. The family owns 2 cars. Totally with summer and winter tires they have 8 tires, but they have also*

	can see all tires.	*old tires. Now Mr. Paterson decided to count the tires to understand what tires are necessary to change or send for recycling.*
Add new tires	User uses service for managing of records about tires and after authentication, he or she can see all tires, which system is monitoring these tires, and add new tires with the amount of seasons tires can be used, name of tires, and type (winter or summer).	*Last year you bought a new car and recently you bought the winter tires for it. You have used already the Tire Recycling service and want to add new tires.*
Edit options of the tires	User uses the service for managing the records about tires and after authentication, he or she can see all tires, which system is monitoring, and change the amount of seasons, name of tires, and type (winter or summer) of already entered tires.	*Mr. Paterson lives in Lappeenranta. He has family. The family owns 2 cars. For summer and winter, the family has 8 tires, but they have also old tires. Now Mr. Paterson decided to change the property season of records according the current situation.*
Get notification about the date to change tires	According to the expression in back-end service, which starts every day, the system decide to send notification in HTML with embedded data via	*Jessica has one car and lives in Helsinki. She bought a car in summer and she doesn't get used of the car. Jessica's friend Tony advised her to use Tires recycling, because he has car for several years and he is aware that*

	SMTP to user	*used tires should be recycled and it would decrease the pollution of environment. Before the 1st of December, Jessica gets notification by e-mail via her Tablet. This notification includes information about the date for changing the summer tires to winter ones. This notification will also includes offers of new winter tires from garage and other best offers.*
Get notification about the temperature to change tires	According the expression in back-end service, which starts every day, the system decides to send notification in HTML with embedded data via SMTP to user.	*Jessica has one car and lives in Helsinki. She bought car in summer and doesn't get used with cars and its service. Jessica's friend Tony advised her to use Tires recycling, because he has car for several years He is aware that used tires should be recycled and it would decrease the pollution of environment. Before the 16th of November, when the weather forecast shows snowstorm and -5 C degrees, Jessica gets notification by e-mail and opens it at to her Tablet. This notification includes information about the date for changing of summer tires to winter tires to decrease the probability of traffic incidents. This notification will also includes offers of new winter tires from garage and other best offers.*
Get	According to the	*Veronica bought car 5 years ago.*

notification about the wear of tires	expression in back-end service, which starts every day, the system decide to send notification in HTML with embedded data via SMTP to user.	*Veronica's friend Roman advised her to use Tires recycling, because he has car for several years and valued this service and he is aware that used tires should be recycled and it would decrease the pollution of environment. Beforehand the 1st of December Jessica gets notification by e-mail and opens it at to her Tablet. This notification includes information about the date for changing of summer tires to winter tires according necessary rules. This notification also includes information that winter tires were used more than 4 seasons and pipes is wearied already and offers to buy new winter tires from garage or to chose the other from the best offers.*

In Table 8 it can be found planned but not implemented functionalities of system because of limitations described in Chapter 4.4.

Table 8: Planned but not implemented functionalities

Get notification about the schedule and personal contact in garage for	According to the expression in back-end service, which starts every day, the system decide to send notification in HTML with embedded data via SMTP to user. During the	*Mr. Johnson lives nearby the Rovaniemi, he has contracted one company. He is newcomer and not familiar with Finnish language. He drove to Rovaniemi using his own car and his coworker advised him to use Tires recycling Service. With*

changing tires	request the Tire's shop and garage service should return the name and phone of one of the managers and the schedule for garage visits.	*notification to change tires Mr. Johnson will get the contact of the personal speaking English with whom he can discuss all unclear points.*
Get notification about the price for used tires from recycler	According the expression in back-end service, which starts every day, the system may send notification in HTML with embedded data via SMTP to user. During the request, the Tire's recycling service should return the amount of money which it calculated according to the parameters of tires.	*Veronica bought a car 5 years ago. Veronica's friend Roman advised her to use the Tires recycling, because he has a car for several years and valued this service and he supports the idea that used tires should be recycled and it would decrease the pollution of environment. Now Veronica has 2 set of old tires used more than 4 seasons. She is interested to get an offer for them.*

5 DISCUSSION

In this Chapter, it is discussed 4 views related with chosen research methodology, results of the literature review, the implementation of case study and limitations. In design research there is problem of generalizability, because other researchers have their own approaches and can come to different results. It often happened that they are interested only in some special aspect of the solution. In this work in Chapter 2 it was tried to get the widest view on existed approaches to multi-devices and multi-platform development.

5.1 Discussion of literature review

In Chapter 2.5 it was inferred about advantages and weaknesses of four different approaches to the development of multi-devices web-applications. RWD and progressive enhancement are browser dependent and for a description of all possible variety of devices it could take a long code [11, 14]. It is well-known and wide-used way in the development, but this approach depends mostly on the side of clients and decision is not reusable for other applications. Markup languages based approach is universal, because it separates the realization from the model, but this approach is not approved widely that's why not all devices support it [17, 18].

SOA approach to UI has several methods of implementation, but WSRP is the most practical used. It requires the special Consumer–Producer architecture and infrastructure [23]. That's why now the trend in studies of SOA UI is in a composition framework and its realization. These frameworks took the best features of RWD and progressive enhancement for the front-end and the power of SOA for back-end.

5.2 Discussion of implementation of case study

The presented approach for implementing the proposed in Chapter 3 architecture uses the modern tools and AnyPoint Studio IDE with a support of ESB realization. It results in powerful capabilities such as exposing and hosting reusable services, using AnyPoint Studio decision as a lightweight service container. Also, such decision provides Service mediation. It separates services from message formats and protocols, divides business logic

from messaging, and enables location-independent service calls. Another good finding of realization with the help of AnyPoint Studio is message routing, which means routing, filtering, aggregating, and re-sequencing of messages based on content and rules and data transformation - exchanging of data across varying formats and transport protocols.

It should be mentioned the limitations of implementation such as in using Relation Database MySQL as the Storage service with Node.js. It was used because of the speed of realization and it was well-known for developers. But the Storage service should be universal and reachable from the different platforms. Of course, in future one of the main goals is transition to NoSQL databases, for instance, MongoDB, because in the scheme of the database there is only 1 table with data and stack MEAN (Mango, Express, Angular, Node) designed to operate with NoSQL databases.

NoSQL (Not Only SQL) has been proposed to encompass the general trend of a new generation of database servers. Such servers have been introduced to solve the challenges that were not being met by traditional SQL (Structured Query Language). The following is main motivation behind NoSQL:

> "Scalability is a key motivation common to all of these. For most purposes, a document database provides the largest feature set with an acceptable/scalable performance. For simple cases where you do not want complicated query requirements, key-value databases provide the best performance." [46]

Some functionality has not been implemented such as data services due the lack of Open Data services via Internet regarding tires technical specifications and tires manufacturers.

5.3 Limitations

This thesis tried to answer a very complex and difficult problem to solve both the academic and industry point of views, how to develop an application for all type of platforms or devices. And that's why in addition to mentioned above limitations of implementation there is a limitation of the decision. The SOAforMDUI presented in this work is not fully validated, but it offers a good starting that shows how it could be implemented.

It also should noted that though SOMA was the first publicity announced SOA-related methodology, there are also and others, for instance, Service-oriented modeling framework. SOMA was chosen like the well-known.

6 CONCLUSIONS

Every day the number of computer devices is drastically increasing. As a result, it is more and more becoming crucial that software developers are empowered with new tools and methods to design and develop different configurations of the same application for this wide diversity of devices. The standardization of the development process, as well as the normalization of a UI model, is required as never before.

This Master's Thesis work is a contribution to this efforts and objectives. The literature review and the case study we conducted suggested a service-oriented architecture of a multi-platform and multi-devices user interfaces.

It was described four different approaches to the development: responsive web design, progressive enhancement based on browser-, device-, or feature-detection, markup languages-based approaches, and service-oriented approach to UI.

It was investigated these approaches while summarizing their advantages and weaknesses. Then it was applied SOA principles to propose an innovative SOAforMDUI. It was also shown how the process of SOAforMDUI design can be aligned with the SOMA methodology proposed IBM as well as how SOA patterns can be applied. It was also proposed formalization to understanding the complexity of the huge numbers of variants for possible implementations of a UI. The practical part of this work includes the implementation of the proposed SOAforMDUI.

One innovative outcome of this research is that there is no recognized notation to describe SOA applications at a high level of abstraction. It was proposed a first version of SSMN in Chapter 3.1.1. Similar to Business Process Model Notation (BPMN) and Unified Modeling Language (UML), SSMN aims to be a graphical notation to describe services and their relationships and interactions.

A possible extension of this work is the development of MDA framework, which will help to instantiate the SOAforMDUI more easily using models of UI and their transformations into a concrete UI. According to [47] Model-driven architecture arouses interest by its

ability to transform one Platform-independent base model into several Platform-specific models, one for each platform or technology where application will be operated, and the automatic code generation that implements the application for those platforms. SOA are also suitable for business agility, especially when combined with a model-driven approach. Both these two technology complement each other.

The study connected with a theoretical part of SOA UI appeared 10 years ago and the practical approaches to these models became mature no further than 2-4 years ago. In this way SOA-based Framework for UI composition is a new approach and is an important add-on to existing SOA techniques. The presented work paves the direction for modeling UI and user interaction as a part of integrated application development. The following research is able to lead to automatic transformations of UI models into the concrete code.

REFERENCES

1. Seffah, A., Javahery, H. 2004. Multiple user interfaces: cross-platform applications and context-aware interfaces, John Wiley & Sons Ltd., pp. 4

2. Melchior, J., Vanderdonckt, J., Van Roy, P. 2011. A model-based approach for distributed user interfaces. *Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems*. ACM. pp. 11-20

3. Baresi, L., Di Nitto, E., Ghezzi, C., Guinea S. 2007. A framework for the deployment of adaptable web service compositions. *Service Oriented Computing and Applications* 1(1). pp. 75

4. Dr. Trkman, P., Dr. Kovačič, A., Dr. Popovič, A., SOA Adoption Phases, *Business & Information Systems Engineering*. August 2011.Volume 3, Issue 4. pp 211-220

5. Pasley, J. 2005. How BPEL and SOA are changing web services development. *IEEE Internet Computing*. pp. 60–67

6. Chung, J., Lin, K., Mathieu, R. 2003. Web Services Computing: Advancing Software Interoperability. *IEEE Computer 36*. October 2003. pp. 35–37

7. Nunamaker, F., Chen, M., Titus, D. M. P. 1990. Systems Development in Information Systems Research, *Twenty-Third Annual Hawaii International Conference System Science*. Volume 3. pp. 89-106

8. Jrad, R.B.N., Ahmed, M.D., Sundaram, D. 2014. Insider Action Design Research a multi-methodological Information Systems research approach. *2014 IEEE Eighth International Conference Research Challenges in Information Science (RCIS)*. pp 5

9. Marcotte, E. 2015. Responsive Web Design [Online]. Available at: https://alistapart.com/article/responsive-web-design [Accessed: 13 March 2015]

10. De Graeve, K. 2011. HTML5 - Responsive Web Design [Online]. Available at: https://msdn.microsoft.com/en-us/magazine/hh653584.aspx [Accessed: 13 March, 2015]

11. Mozilla Developer Network and individual contributors 2015. HTML5 - Responsive Web Design [Online]. Available at: https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Media_queries [Accessed: 13 March 2015]

12. Coyier, Ch. 2015. A Complete Guide to Flexbox [Online]. Available at: https://css-tricks.com/snippets/css/a-guide-to-flexbox/ [Accessed: 13 March 2015]

13. Mullany, M., 2015. A Complete Guide to Flexbox. CSS-Tricks [Online]. Available at: http://www.sencha.com/blog/blackberry-torch-the-html5-developer-scorecard [Accessed: 13 March, 2015]

14. Ateş, F., Irish, P., Sexton, A. 2015. Modernizr Documentation [Online]. Available at: http://modernizr.com/docs/#installing [Accessed: 13 March, 2015]

15. Ryanve, A., Response JS: mobile-first responsive design in HTML5 [Online]. Available at: http://responsejs.com/#create [Accessed: 13 March, 2015]

16. Ali, M.F., Pérez-Quiñones, M.A., Shell, E., Abrams, M. 2002. Building Multi-Platform User Interfaces with UIML. Springer Netherlands

17. Abrams, M., Helms, J. 2004. User interface markup language (UIML) specification [Online]. Available at: https://www.oasis-open.org/committees/download.php/5937/uiml-core-3.1-dra%20ft-01-20040311.pdf [Accessed: 08 May 2015]

18. Lepreux, S. 2015. What is USIXML? UsiXML - USer Interface eXtended Markup Language [Online]. Available at: http://www.usixml.org/en/what-is-usixml.html?IDC=236 [Accessed: 13 March, 2015]

19. Chung, J. Y., Chao, K. M. 2007. A view on service-oriented architecture. *Service Oriented Computing and Applications* 1(2), pp. 93-95

20. Tsai, W.-T., Huang, Q., Elston, J., Chen, Y. 2008. Service-Oriented User Interface Modeling and Composition. Research Challenges in Information Science (RCIS). *2008 IEEE International Conference on e-Business Engineering*. pp 21

21. Castle B. Introduction to web services for remote portlets [Online]. Available at: http://www.ibm.com/developerworks/ru/library/ws-wsrp/index.html [Accessed: 08 May 2015]

22. BEA Systems Incorporated 2006. Introduction to WSRP [Online]. Available at: https://docs.oracle.com/cd/E13218_01/wlp/docs81/wsrp/intro.html [Accessed: 02 April, 2015]

23. BEA Systems Incorporated 2006. Using WSRP with WebLogic Portal [Online]. Available at: https://docs.oracle.com/cd/E13218_01/wlp/docs81/pdf/wsrp.pdf [Accessed: 02 April, 2015]

24. Graham, S., Daniels, G., Davis, D., Nakamura, Y., Simeonov, S., Boubez, T., Koenig, D. 2004. Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI. SAMS publishing

25. Roebuck, K. 2012. Portlets: High-impact Strategies - What You Need to Know: Definitions, Adoptions, Impact, Benefits, Maturity, Vendors, Emereo Publishing

26. Erl, Th. 2005. Service-Oriented Architecture (SOA) Concepts, Technology and Design, Prentice Hall PTR

27. Arsanjani, A. 2004. Service-oriented modeling and architecture, IBM Corporation

28. Arsanjani, A., Zhang, L. J., Ellis, M., Allam, A., & Channabasavaiah, K. 2007. S3: A service-oriented reference architecture. *IT professional 9*(3), pp. 10-17

29. Arsanjani, A., Ghosh, S., Allam, A., Abdollah, T., Ganapathy, S., Holley, K. 2004. SOMA: A method for developing service-oriented solutions, *IBM Systems Journal*, 47 (3), pp. 377-395

30. Erl, Th. 2008. SOA design patterns, Pearson Education

31. Erl, Th. 2008. Modeling SOA: Part 3. Capability Composition [Online]. Available at: http://soapatterns.org/design_patterns/capability_composition [Accessed: 06 April, 2015]

32. Erl, Th. 2008. SOA Patterns. Design Patterns. Service Layer [Online]. Available at: http://soapatterns.org/design_patterns/service_layers [Accessed: 06 April, 2015]

33. Hogg, S., Smith, J., Chong, H., Hollander, P. 2008. SOA Patterns. Design Patterns. Brokered Authentication [Online]. Available at: http://http://soapatterns.org/design_patterns/brokered_authentication [Accessed: 06 April, 2015]

34. Dumas, B., Lalanne, D., & Oviatt, S. 2009. Multimodal interfaces: A survey of principles, models and frameworks. *Human Machine Interaction*. Springer Berlin Heidelberg. pp. 3-26

35. Gackenheimer, C. 2013. Node.js Recipes, Apress

36. Odell, D. 2014. Design Patterns: Architectural. Apress. pp 199-221

37. Harris, A. 2010. jQuery and Ajax in the Presentation Tier. Apress. pp 135-164

38. Garrett, J. J. 2005. Ajax: A New Approach to Web Applications [Online]. Available at: https://courses.cs.washington.edu/courses/cse490h/07sp/readings/ajax_adaptive_path.pdf [Accessed: 08 May 2015]

39. Hanson, J. 2013. Passport. Overview [Online]. Available at: http://passportjs.org/guide/5 [Accessed: 28 March , 2015]

40. Dr. Lui M., Gray M., Chan A., Long J. 201. Pro Spring Integration. Apress. pp. 16-17

41. Fielding, R. T. 2000. Architectural styles and the design of network-based software architectures. Ph.D. Dissertation. University of California

42. Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J-J., Nielsen, H.F., Karmarkar, A., Lafon, Y. 2007. W3CSOA. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition) [Online]. Available at: http://http://www.w3.org/TR/soap12-part1 [Accessed: 01 April, 2015]

43. Liu, L., Ozsu, T.M. 2009. Encyclopedia of Database Systems. Springer Science+Business Media

44. Carroll, J.M. 2000. Making Use: Scenario-Based Design of Human-Computer Interactions. MIT Press

45. Seffah, A., Engelberg D., Karaseva, V. 2015. UXModeler: A Social Network Infrastructure for Engaging Users and Modeling their Experiences in the Design Loop

46. Syed, B. 2014. Beginning Node.js. Apress. pp. 181-182

47. Kim H.-K., Kim T.-H. 2014 SOA Modeling Based on MDA, Distributed Computing and Artificial Intelligence, *11th International Conference Advances in Intelligent Systems and Computing Volume 290*. pp 181-194