Nominalia Web Services

Patrik Wallén

TRITA-NA-E03151
Nominalia Web Services

Patrik Wallén

TRITA-NA-E03151

Master’s Thesis in Computer Science (20 credits)
at the School of Computer Science and Engineering,
Royal Institute of Technology year 2003
Supervisor at Nada was Sten Andersson
Examiner was Stefan Arnborg
Abstract

The project described in this document has been undertaken for the completion of a Master of Science degree in Computer Science at the Royal Institute of Technology in Stockholm, Sweden and at the Technical University of Catalonia in Barcelona, Spain. The objective of the project is the implementation of a new platform based on Web Services, a new set of standards for distributed systems, for the resellers of one of Spain’s biggest Internet domain registrars, Nominalia Internet S.A., to manage their customers’ Internet domains and associated information.

The Web Service application is built on the Microsoft .NET platform. It has been built on top of the legacy systems and exposes the most common operations that Nominalia provide for its resellers. These operations allow the creation, modification and retrieval of information about domains, hosts and contacts. A Windows service to manage non-successful operations and a stand alone application to maintain the internal database have also been implemented.

This new platform is designed to increase the business volume with new resellers, offering them a stable solution that is easy to use and can easily be integrated with their current systems. The goal is to reduce the costs associated with reseller support and avoid the development costs of creating tailor made solutions, thus permitting Nominalia to offer more competitive prices.

Web Services are based on open standards that make them highly interoperable. The technology used to implement them is ubiquitous, which makes them easy to integrate regardless of where they reside or how they are implemented. The thesis also contains an introduction to the technologies upon which Web Services are built: SOAP, WSDL and UDDI.
Sammanfattning


Detta nya gränssnitt är framtaget för att öka handelsvolymen med nya återförsäljare, erbjuder dem en stabil och lättanvänd produkt som lätt kan integreras med deras nuvarande system. Målet är att minska kostnaderna runt återförsäljarstöd och att undvika utvecklingskostnader för skräddarsydda lösningar, vilket skulle göra att Nominalia kan erbjuda konkurrenskraftigare priser.

Rapporten innehåller även en introduktion till de tekniker som Web Services är byggda på: SOAP, WSDL och UDDI. Web Services är baserade på öppna standarder som ger dem mycket hög interoperabilitet. Tekniken som används för att implementera dem är mycket vanlig och erkänd, vilket även gör dem enkla att integrera oavsett var de befinner sig eller hur de är realiserade.
Table of contents

1 INTRODUCTION ......................................................................................... 1
    1.1 DESCRIPTION OF THE PROJECT ............................................................. 1
    1.2 PERSONAL MOTIVATION ....................................................................... 1
    1.3 OBJECTIVES .......................................................................................... 2
    1.4 DOCUMENT STRUCTURE ........................................................................ 3

2 WEB SERVICES TECHNOLOGIES .............................................................. 4
    2.1 INTRODUCTION TO WEB SERVICES ..................................................... 4
    2.2 WEB SERVICE ARCHITECTURE ............................................................ 7
    2.3 SOAP ........................................................................................................ 13
    2.4 WEB SERVICE DESCRIPTION LANGUAGE ....................................... 20
    2.5 UNIVERSAL DESCRIPTION, DISCOVERY AND INTEGRATION ............. 30

3 APPLICATION REQUIREMENT SPECIFICATIONS ................................. 34
    3.2 GLOSSARY .............................................................................................. 35
    3.3 USE CASE DIAGRAM ............................................................................. 36
    3.4 USE CASE SPECIFICATION .................................................................... 37
    3.5 CLASS DIAGRAM .................................................................................. 39
    3.6 CLASS DESCRIPTIONS ........................................................................... 41
    3.7 SEQUENCE DIAGRAMS .......................................................................... 43
    3.8 INTERFACE DESIGN .............................................................................. 45
    3.9 INFORMATION SOURCES ......................................................................... 46
    3.10 CONSTRAINTS ....................................................................................... 46

4 THE WEB SERVICE APPLICATION .......................................................... 47
    4.1 INTRODUCTION ....................................................................................... 47
    4.2 WEB SERVICE MODULE ...................................................................... 48
    4.3 DATABASE MODULE ............................................................................... 53
    4.4 NOMINALIA CLEANING SERVICE .......................................................... 57
    4.5 ARCHIVING APPLICATION .................................................................... 62

5 CONCLUSIONS .......................................................................................... 64
    5.1 OBJECTIVES REACHED ......................................................................... 64
    5.2 PLANNING AND COSTS .......................................................................... 66
    5.3 EVALUATION .......................................................................................... 68
    5.4 FUTURE ................................................................................................... 68

BIBLIOGRAPHY ............................................................................................... 70

WEB SERVICES TECHNOLOGIES .................................................................. 70
SOFTWARE DEVELOPMENT .......................................................................... 71
BOOKS ........................................................................................................... 71
REFERENCES .................................................................................................. 70
Preface

The project that is specified in this document is a Master’s project for the Master of Science in Computer Science program at the Royal Institute of Technology in Stockholm, Sweden and at the Technical University of Catalonia in Barcelona, Spain. It has been conducted in collaboration with the technical consulting firm Spenta Consulting as part of a project for Spenta’s customer Nominalia Internet.

I would like to thank my supervisors from my two universities: Felix Freitag of UPC and Sten Andersson of KTH for their support.

And a special thanks to Jordi Plana, CEO of Spenta Consulting, for his support and understanding of the fact that I have had to share my time between work and study.
1 Introduction

In spite of its simplicity, XML (extensible mark-up language) [1] is changing the software industry. XML is a simple, text-based language that provides for the definition, transmission, validation and interpretation of structured data.

The Internet revolution changed the way users interacted with their applications. XML is now changing the way that applications communicate with other applications, or in an even wider scope, the way that computers communicate with other computers. Web Services, the new standard for communication in distributed systems, are based on XML and allow applications to exchange data, and more importantly, to invoke functionalities given by other applications. This is possible without having to worry about how applications are built, or on what operating system they are running, or even by what means they are accessible. Web Services is a standard that uses already standardized protocols aimed to help communication in a distributed environment. This thesis describes an implementation of Web Services exposing already existing functionality but in a standardized way to attract more clients. It also contains an introduction to the technologies that Web Services are built on.

1.1 Description of the project

The project has been divided into two parts. The first part involves the research conducted, including studying the Web Services technologies and their area of use.

In the second part, an application is developed for the company Nominalia. This application is a new platform based on Web Services for its resellers to manage their customers’ Internet domains and associated information.

Being based on a real life project, the work has been conducted following a methodology involving various phases.

An initial research phase was conducted to decide the viability of the project and to make an initial system design.

A requirement phase involved reassembling all the information needed for the finalization of the project. Parts of this phase were conducted in collaboration with the professionals of Spenta Consulting and Nominalia.

A design phase included the system design of the application, how to integrate it with the backend system of Nominalia, and the design of the public interfaces.

And finally an implementation phase included implementation of the project in the programming language C# on the Microsoft .NET platform [2], documentation and installation of the final product.

1.2 Personal motivation

I have been working in the Internet business since 1998 initially as a programmer, then as a database designer, and currently as a system architect in a small consulting company. In many of the projects I have been involved in, I have consistently encountered problems concerning integration. This integration treated the linking of
different parts of software applications, sometimes in different languages and sometimes on different technical platforms.

After having spent a significant amount of time solving these problems using everything from proprietary comma separated protocols to platform specific distributed object models, many times with the other end of the communication being done by somebody else and in the worst case at the same time, I have come to the conclusion that there must be an easier way.

So when I first heard of Web Services, on the Microsoft Tech-Ed in Barcelona 2001, I became very interested. Later after spending some time investigating Web Services and their use in a pre-study for a project, the idea to conduct my final project together with the company that I work for was born. As I wanted my final thesis to be truly applicable, I decided to base it on a real project for an actual client.

1.3 Objectives

The main objective is to develop a new platform for Nominalia Internet S.A. for their resellers, which will expose their most common operations through Web Service standards. In more detail the objectives can be divided in objectives of the student and objectives of the client.

1.3.1 Objectives of the student

The objectives of this project from the scholarly point of view are to:

- Overcome the software design and system architecture problems and challenges of developing a Web Service project.
- Evaluate the different architectures applicable to the actual project and choose the optimal one.
- Explore the maturity of this technology.
- Research the evolution of the standards on which Web Services are based.
- Obtain a real vision of how Web Services can help a company to improve its processes and reduce its costs.

1.3.2 Objectives of the client

Nominalia is the Spanish market leader in the Internet domain name registering business managing approximately 100,000 domain names. At the current time around 60 % of its sales are generated by its network of resellers.

To register and manage the domains of their clients, the resellers use management tools developed by Nominalia. These application tools are proprietary and, in some cases, not as solid and failsafe as required. In some cases they do not provide all the functionality that is requested by the resellers, forcing Nominalia to implement tailor made solutions, especially for the most important clients. The support given to the resellers implies a high cost for Nominalia, forcing the company to charge above market rate. The result is a significant loss in market competitiveness.
Nominalia has decided to offer some of their services through the Web Services technology. The implementation of a platform for registering and managing domain names based on Web Services will give the following direct benefits to Nominalia:

- Reduce the costs associated with reseller support, permitting Nominalia to offer more competitive prices.
- Increase volume of business through new resellers, offering them a stable solution that is easy to use and easy to integrate with their actual systems.
- Avoid the development costs of tailor made solutions to its resellers.
- Build a new solid technology base for a future migration of the backend systems of Nominalia to an open and scaleable platform based on the Microsoft .NET technology.

Other objectives desired by Nominalia from this project are:

- Media impact – to be one of the first companies to implement this new technology.
- Technology positioning – Nominalia will be one step ahead of its competitors on a technical level and will be able to benefit from this advantage by developing new products and services based on this technology.

1.4 Document structure

The main document is a summary of how the project has been done and in some cases a description why it has been done in this way. The first appendix is the technical part of the pre-study done for Nominalia before starting the project. The second appendix is the user manual of the Web Service application developed. It is given as a separate appendix as it has been delivered to the client as documentation of the Web Service application.

Main document:

- Motivation, description and conclusions of the work that has been done.
- An introduction to the Web Service technologies.
- The functional requirements of the application and a description of how it has been built.
- A description of the different modules of the application with instructions on how to administrate and configure each of the modules.

Appendices:

- The pre-study that was conducted before the implementation started, which was designed to convince Nominalia of the viability of the project.
- A user manual that describes where and how a client accesses the different operations of the application. It was created as a base for Nominalia to develop its own manual for their resellers.
2 Web Services technologies

2.1 Introduction to Web Services

“Web Services represent one of the biggest technical evolutions in the last years. Orientated to facilitate the offering of services through the Internet and improve communications and relations between companies. Helps and reduces the costs of integration in business-to-business environments, thereby creating new business opportunities. Moreover thanks to Web Services the companies can obtain important savings associated to the reduction of time and money necessary for the creation of new developments and the maintenance of their products.” - taken from www.idg.se

This is an example of what can be found reading an article about Web Services. There is no doubt that Web Services are a new growing business trend, however only the future can tell if they will have the importance that many people are trying to give them.

In this chapter a short introduction to Web Services is given: what they are, why one would want to use them, and what they look like from a developer’s point of view.

2.1.1 What is a Web Service?

Web Services are the fundamental building blocks in the move to distributed computing on the Internet. Open standards and the focus on communication and collaboration among people and applications have created an environment where Web Services are becoming the platform for application integration. Applications can be constructed using multiple Web Services from various sources that work together regardless of where they reside or how they are implemented. Web Services are successful for two reasons:

- They are based on open standards that make them interoperable.
- The technology used to implement them is ubiquitous.

2.1.2 Definitions

A Web Service is a software application or a component identified by an internet address that other software applications or components can directly interact with via web-based protocols, where said interactions do not require direct human involvement.

There exist many definitions of Web Services but almost all definitions have these things in common:

- Web Services expose useful functionality to Internet users through a standard Internet protocol. In most cases, the protocol used is SOAP.
- Web Services provide a way to describe their interfaces in enough detail to allow a user to build a client application to talk to them. This description is
usually provided in a XML document called a Web Services Description Language (WSDL) document.

- Web Services are registered so that potential users can find them easily. This is done with Universal Discovery Description and Integration (UDDI) registries. These three technologies SOAP, WSDL and UDDI are explained in more detail later in this chapter. Figure 2-1 shows a graphical representation of a typical Web Service scenario including these three technologies.

![Web Service Workflow Diagram](image)

**Figure 2-1. Web Service workflow**

### 2.1.3 History

Distributed computing is not something new. In the 1980’s CORBA from the Object Management Group and COM from Microsoft where created to accomplish the same goal as Web Services are now, facilitate distributed computing. These protocols are neither open nor standardized. They are binary and proprietary to each vendor. The operations are specified with Interface Definition Language (IDL) and heavily object orientated. The coupling between the server and the clients is very hard. Later, in 1999, XML-RPC introduced remote method invocation with XML messages sent over HTTP. This could be said to be an ancestor of SOAP but the data types were

---

1 Common Object Request Broker Architecture (CORBA) is an architecture and specification for creating, distributing, and managing distributed program objects in a network from the Object Management Group.

2 Component Object Model (COM) is a framework from Microsoft to develop and support program component objects.

3 Interface Definition Language, (IDL) is a generic term for a language that lets a program written in one language communicate with another program written in an unknown language, describing its methods and interfaces.
very limited and it was not at all extendible.

2.1.4 Advantages

One of the primary advantages of the Web Services architecture is that it allows programs written in different languages on different platforms to communicate with each other in a standards-based way. However this is not new and can be done with various technologies already but there are some differences.

The first difference is that SOAP is significantly less complex than earlier approaches, so the barrier to entry for a standards-compliant SOAP implementation is significantly lower. As expected all major software companies have their own commercial SOAP implementations. There are also many implementations that are built and maintained by single developers.

The other significant advantage that Web Services have over previous efforts is that they work with standard Web protocols – XML, HTTP and TCP/IP. A considerable number of companies already have a web infrastructure, people with knowledge and experience in maintaining it, so the entry cost for Web Services is significantly less than previous technologies. Table 2-1 lists some of the problems a solution based on Web Services would solve in comparison with one based on proprietary protocols.

<table>
<thead>
<tr>
<th>Proprietary solution problem</th>
<th>Web Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Significant system changes are difficult.</td>
<td>Loosely coupled, standards-based format and protocol makes changes easier.</td>
</tr>
<tr>
<td>Architectural changes require unrelated components to change.</td>
<td>Coarse-grained, service-oriented request/reply paradigm hides implementation.</td>
</tr>
<tr>
<td>Long development cycle due to poor component reuse.</td>
<td>Loosely coupled, component based services are reusable and dynamically configurable.</td>
</tr>
<tr>
<td>Long time spent on proprietary APIs.</td>
<td>Vendor-neutral standardization of format and protocols.</td>
</tr>
<tr>
<td>High maintenance and integration costs.</td>
<td>Loosely coupled, standards-based format and protocol.</td>
</tr>
</tbody>
</table>

Table 2-1. Web Service advantages.

2.1.5 What can Web Services do?

A Web Service has been defined as a software service exposed on the web through SOAP, described with a WSDL file and registered in a UDDI register. The next logical question is: “What can be done with Web Services?” The first and easiest Web Services were often information sources that could easily be incorporated into applications, like stock quotes, weather forecasts, sports scores etc. Most of this information is already available on the Internet, but Web Services make programmatic access to it easier and more reliable. Exposing existing applications as Web Services will allow users to build new, more powerful applications that use Web Services as building blocks. For example, a user might develop a purchasing application to automatically obtain price information from a variety of vendors, allow the user to select a vendor, submit the order and then track the shipment until it is received. The vendor application, in addition to exposing its services on the Internet,
might in turn use Web Services to check the customer's credit, charge the customer's account and set up the shipment with a shipping company.

2.2 Web Service Architecture

2.2.1 Technical Platform Requirements

There is a big similarity between the Web Service architecture and a normal web application as can be seen in Figure 2-3. Thus in principle Web Services can be developed with the same technologies and tools as normal Web applications. However, there are some notable differences both on the server hosting the Web Server and the client consuming it.

- **Server side**: In particular, in order to implement a Web Service listener, the application needs to understand SOAP messages and generate SOAP responses. And optionally also provide a WSDL document for the service and eventually also advertise it in a UDDI registry.

- **Client side**: When consuming Web Services, you might need to use UDDI to locate services and service contracts, interpret a WSDL contract for the service, and generate appropriate SOAP messages and interpret SOAP responses.

The abilities

In addition, because applications rely on Web Services, it is critical that these services are completely dependable. A Web Service should always be available. It must not make mistakes, loose requests or fail in the event of invalid requests. It should always be able to meet client demand with acceptable performance. In the rare event that a fault occurs, the Web Service should continue processing requests as best it can. In other words, a Web Service needs, even more than normal applications:

- Scalability
- Reliability
- Availability

In short, if a Web Service is not dependable, application developers will not use it.

Middleware

The requirements of Web Services consumers and system administrators impose a considerable burden on Web Service developers that is not always related to the functionality provided by the service. There exists a need for infrastructure and tools that make it easier to implement secure, reliable, scalable and highly available Web Services. Developers will also want infrastructure and tools to help them debug and trace execution of their code and the infrastructure services they are using.

Ideally the learning of a new programming language will not be needed in order to use this infrastructure and these tools. Much of the difficulty in implementing scalable, highly available code is connected with properly managing resources such as processes, threads and shared state. For example when multiple concurrent requests are received by a service, preferably the Middleware should handle the concurrency.
Middleware that manages these resources and lets the developer write service logic as if a single client is accessing it can greatly improve the reliability, scalability and availability of Web Services, as well as greatly simplify their development.

In practical terms, the middleware provides a standard hosting environment for the Web Service code. This hosting environment is responsible for:

- Listening for incoming HTTP requests.
- Performing security authentication and authorization checks.
- Dispatching authorized requests to the correct service.
- Ensuring that services are isolated from each other and the hosting environment.
- Automatically recovering from service, hosting environment, and system failures.
- Providing administrative facilities for deploying, monitoring, and controlling services.
- Managing resources such as processes, threads, and shared state on behalf of each service.

A middleware will improve productivity as well as overall reliability by eliminating the need of writing, parsing and formatting logic for the above tasks. The Application Programming Interfaces (API’s) given by the middleware or framework will also reduce the need of learning every detail of the Web Service specifications. When implementing Web Services using component technologies, additional productivity gains can be achieved if the system provides services to activate objects on demand and map messages to object method calls - essentially implementing the Web Service listener. Similarly, when consuming Web Services, tools that construct proxy objects hiding the details of formatting and sending a service request and then interpreting the response are wanted.

### 2.2.2 Generic architecture for Web Services

Figure 2-2 shows the generic architecture of a Web Service. The architecture is divided into five logical layers.

- Furthest from the client is the **data layer**, which stores information required by the Web Service.

- Above the data layer is the **data access layer**, which presents a logical view of the physical data to the business layer. The data access layer isolates business logic from changes to the underlying data stores and ensures the integrity of the data.

- The business layer implements the business logic of the Web Service. As in Figure 2.2, it is often subdivided into two parts: the **business facade** and the **business logic**.
  
  - The business facade provides a simple interface that maps directly to operations exposed by the Web Service.
• The business facade uses services provided by the business logic layer. In a simple Web Service, all the business logic might be implemented by the business facade, which would interact directly with the data access layer.

• Client applications interact with the Web Service listener. The listener is responsible for receiving incoming messages containing requests for service, parsing the messages, and dispatching the request to the appropriate method on the business facade. If the service returns a response, the listener is also responsible for packaging the response from the business facade into a message and sending that back to the client. The listener also handles requests for contracts and other documents about the Web Service.

Finally the only part of the Web Service that is aware of that it is part of a Web Service is the listener.

![Diagram of Generic Web Service architecture](image)

This architecture is very similar to the normal n-tier application architecture. As shown in Figure 2-2, the Web Service listener is equivalent to the presentation layer of a normal Web application. A common development scenario is likely to include exposing functionality of an existing Web application for programmatic access, that is, adding a Web Service to an existing application. As seen in Figure 2-2, this could be as simple as implementing a Web Service listener that accesses the existing business facade.
Web Services must be agnostic regarding the choice of operating system, object model and programming language to obtain the interoperability wanted and succeed in the heterogeneity of the Web. For Web Services to enjoy the same widespread adoption as other Web based technologies, they must be:

Loosely coupled: Two systems are considered loosely coupled if the only mandate imposed on both systems is to understand the agreed messages between them. Web Services achieve this capability using XML.

Ubiquitous communication: Nowadays it hardly exists an operating system that does not incorporate the ability to connect to the Internet and therefore providing a ubiquitous communication channel. As such, the ability to connect almost any system or device to the Internet will ensure that such systems and devices are universally available to any other system or device connected to the Internet.

Universal data format: By adopting existing and open standards any system supporting the same open standards is capable of understanding and using Web Services. Utilizing self-describing, text-based messages that Web Services and their clients can share without the need to know what constitutes each underlying systems, enables communication between autonomous and disparate systems. This is achieved using XML and XML Schemas.

Web Services employ an infrastructure that provides the following:
- A service description for defining how to use the services (WSDL)
- A standard wire formats with which to communicate (SOAP)
- A discovery mechanism to locate Web Services (UDDI)

These three acronyms form the base upon which Web Service are built and are explained in more detail in the corresponding chapters.
2.2.3 Different Platforms

There are many platforms competing on the market to provide means to fulfill the requirements mentioned above. All major platform and software providers in the Internet business, for example Microsoft, IBM and Sun, have their own Web Service platform implementations. There also exist many small proprietary and open-source implementations, which can be entire solutions or just some parts that need to be adapted to special needs.

Web Service platform providers:
- Microsoft .NET [2]
- J2EE [3] (Sun One, IBM WebSphere, etc.)
- Open-source implementations

The most common is to use existing commercial platforms like the Microsoft Visual Studio .NET, that offer services that make the development much easier and foremost much faster.

The Web Services of any of the different platforms should be interoperable with any of the others. As the formats automatically generated by the different platforms might not be 100% interoperable, special care must be taken when specifying the SOAP messages and the WSDL document of the Web Service. There exists a branch of the W3C that is specialized only on interoperability issues: the Web Services interoperability organization (WS-I Org.) [4].

In the Pre-study, appendix A, and the User manual, appendix B, of the Nominalia Web Services the main focus is on the Microsoft .NET platform as the application is implemented on that platform. Therefore in this chapter an example of an open source J2EE platform, the Apache Axis, will be presented briefly.

2.2.4 Apache Axis

The Apache Axis [5] is developed by the Apache foundation and is therefore totally Open-source. It is written in the Java programming language and can thus be used on any platform supporting Java. It can be integrated with any J2EE application server and supports the latest evolutions of the SOAP specifications like synchronous and asynchronous messages. Following is a listing of some of the other features offered:

Creation of Web Services
- All Java classes can be automatically exposed as Web Services (RPC-style).
  - One operation per method
- Automatic generation of SOAP messages
- Automatic generation of WSDL
- Generation of Java code from a WSDL document
Web Service invocation

- Generation of client “stubs” from a WSDL document
  - A local object that automatically invokes the corresponding Web Service
- Automatic generation of SOAP messages
- Error handling
  - Conversation of SOAP Faults to Java Exceptions

2.2.5 Client code example – Apache Axis

Following is the Java code example of a client using the Apache Axis to call a simple Hello World Web Service.

```java
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import javax.xml.namespace.QName;

public class TestClient {
    public static void main(String[] args) {
        try {
            String endpoint = 
                "http://nagoya.apache.org:5049/axis/services/echo";

            Service service = new Service();
            Call call = (Call) service.createCall();

            call.setTargetEndpointAddress( new java.net.URL(endpoint) );
            call.setOperationName(new QName("http://soapinterop.org/",
                "echoString");

            String ret = (String) call.invoke( new Object[] { "Hello!" } );
            System.out.println("Sent 'Hello!', got '" + ret + '"");
        }
        catch (Exception e) {
            System.err.println(e.toString());
        }
    }
}
```

Listing: Apache Axis example, TestClient.java
2.3 SOAP

2.3.1 Introduction

This chapter is a brief introduction to the Simple Object Access Protocol (SOAP) 1.1. The text is focused on the aspects of SOAP that relate to its use as message protocol for Web Services. The information is mainly taken from the SOAP 1.1 specification [6]. The Web Service Architecture Group of W3C has produced a SOAP 1.2 specification that is currently under review and will not be treated in this document. Following is a short resume of SOAP taken from the SOAP 1.1 Specification [6].

“SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined data types, and a convention for representing remote procedure calls and responses. SOAP can potentially be used in combination with a variety of other protocols; however, the only bindings defined in this document describe how to use SOAP in combination with HTTP and HTTP Extension Framework.”

Basically SOAP provides a mean for exchanging structured and typed information between peers in a distributed environment using XML. It does not define any programming model or implementation specific semantics. It defines a simple mechanism for expressing application semantics by providing a modular packaging model and encoding data within modules. The main design goal is simplicity and extensibility. This allows SOAP to be used in a large variety of systems ranging from messaging systems to RPC and it makes SOAP especially suitable as the main protocol for Web Services.

2.3.2 The SOAP specification

There are four main areas covered by the SOAP specification. The first one is a required SOAP Envelope format that defines what the envelope that surrounds the XML content of a SOAP message must look like. The next two parts describe how to represent program data as XML and how to use SOAP to do Remote Procedure Calls (RPC). The optional RPC part of the specification is used to implement RPC-style applications where a SOAP message, containing a callable function and the parameters to pass to the function, is sent from the client, and the server returns a message with the results of the executed function.

Most current implementations of SOAP support RPC applications because programmers who are used to creating COM or CORBA applications, see 2.1.3, easily understand the RPC style. SOAP also supports Document-style applications where the SOAP message is just a wrapper around an XML document.

The last optional part of the SOAP specification defines what a HTTP message that contains a SOAP message looks like. This HTTP binding is important because HTTP is supported by almost all current operating systems. The HTTP binding is optional,
but almost all SOAP implementations support it, being the only standardized protocol for SOAP. This does not mean that SOAP requires HTTP. Some SOAP implementations support Microsoft Message Queue, IBM MQSeries, Simple Mail Transport Protocol (SMTP) or TCP/IP transports, but almost all current Web Services use HTTP because it is ubiquitous. Most organizations have a network infrastructure that supports HTTP and people who understand how to manage it already.

2.3.3 SOAP toolkits

Most people who use SOAP do not write SOAP messages directly but use a SOAP toolkit to create and parse the SOAP messages. These toolkits generally translate function calls from some programming language to a SOAP message. For example:

- Microsoft SOAP Toolkit 2.0 translates COM function calls to SOAP.
- Apache Toolkit translates Java function calls to SOAP.

The most compelling feature of SOAP is that it has been implemented on many different hardware and software platforms. This means that SOAP can be used to link almost any disparate systems. Many attempts have been made in the past to come up with a common communications protocol that could be used for systems integration, but none of them have had the widespread adoption that SOAP has. The main reason is that SOAP is much smaller and simpler to implement than many of the previous protocols. SOAP can use existing XML Parsers and HTTP libraries to do most of the hard work. This is why there are more than 70 SOAP implementations available. SOAP obviously does not do everything that for example COM or CORBA, see note on page 5, do, but the lack of complexity in exchange for features is what makes SOAP so readily available.

2.3.4 SOAP message model

The SOAP message model consists of three parts:

- The SOAP envelope that defines an overall framework for expressing what is a message, which should deal with it and whether it is optional or mandatory.
- The SOAP encoding rules defines a serialization mechanism that can be used to exchange instances of application-defined data types.
- The SOAP RPC representation defines a convention that can be used to represent remote procedure calls and responses.

All elements in a SOAP message are defined in namespaces in order to disambiguate SOAP identifiers from application specific identifiers. SOAP itself defines two namespaces:

- The SOAP envelope, http://schemas.xmlsoap.org/soap/envelope
- The SOAP serialization, http://schemas.xmlsoap.org/soap/encoding

A SOAP message does not contain any Document Type Declarations or any processing instructions. SOAP messages are fundamentally one-way transmissions
from a sender to a receiver, but as illustrated in the examples, they are often combined to implement request and response patterns. The last example also shows how a SOAP message can be carried in a HTTP message. Regardless of transport protocol the messages can be routed between one or more intermediate nodes before it reaches its ultimate destination.

A SOAP message is a XML document that consists of two mandatory parts; the Envelope that gives information about the message and the Body that contains the actual message. It can also contain and one or more optional Headers that are used to add features to a message, for example security issues like user and password. A graphical representation of the SOAP message structure can be seen in Figure 2-4.

![Figure 2-4. SOAP message structure](image)

**Envelope**

The envelope element is the top element of the XML document. It is mandatory and is together with its attributes defined in the namespace mentioned above.

**Body**

The body element is the actual container of the information aimed to the ultimate recipient. It is mandatory and must be an immediate child element of either the envelope element or a header element if present. The body element can contain an arbitrary amount of child elements that may or may not be namespace qualified. One special element that is sent inside the body element is the fault element that is used to carry error and/or status information within a SOAP message.

**Header**

Header elements are optional and must, if present, follow immediately after the envelope element. All its child elements must be namespace qualified. The header is a generic container for added features to a SOAP message in a decentralized manner.
SOAP defines attributes to indicate who should deal with a feature and whether understanding is optional or mandatory.

**SOAP message formats**

Unfortunately for the standardization and interoperability of Web Services the current SOAP specification gives developers wide latitude in SOAP message formats.

The SOAP message style that decides the format of the method or function request within the SOAP body can be of two styles:

- RPC (Remote Procedure Call)
- Document

And the parameter formatting, the format of the input parameters and return value within the SOAP body, can use two serialization formats:

- Encoded (SOAP-Encoding)
- Literal

In practice the combinations used are limited to:

- RPC/Encoded
- Document/Literal

RPC messaging utilizes specific RPC coding conventions within the SOAP message in order to invoke a specific remote method. The XML typically just describes a method call and its parameters and is bound to a strict method signature. A RPC request might specify that one parameter is a xsd:string and that another is a xsd:int. All parameters are encapsulated within a single XML element named after the Web method. Each XML element within that XML element represents a parameter named after the parameter.

The Document-style approach is much more flexible, allowing any type of XML document associated with any XML Schema. Document style is preferred for interoperability because it uses loose coupling of methods and data. Rather than passing the data as arguments to a method call, which is tightly bound by data type, the document style allows for a wide range of data to be passed and interpreted by the server. Document style messaging can be more complex to implement, but it provides a number of benefits. If offers the full capabilities of XML to describe and validate a business document and it does not require a strict contract between the client and the server. Many new Web Services take advantage of this flexibility to build services that would be difficult to implement using RPC.

Here is how each style affects the contents of the `<soap:Body>` element.

- **RPC**: RPC implies that `<soap:Body>` contains an element with the name of the method or remote procedure being invoked. This element in turn contains an element for each parameter of that procedure.
- **Document**: The `<soap:Body>` contains one or more child elements called parts. There are no SOAP formatting rules for what the `<soap:Body>` contains; it contains whatever the sender and the receiver agree upon.

SOAP Encoding is a set of serialization rules defined in section 5 of SOAP 1.1 and is also sometimes referred to as "Section 5 Encoding" or only "Encoding". The rules specify how objects, structures, arrays, and object graphs should be serialized. RPC/Encoded requests specify the data types of their parameters one-by-one using XML Schema basic types. The SOAP encoding must conform to the SOAP 1.1 specification, while a literal encoding gives you more flexibility in using data types that map to XML Schemas.

Document refers to formatting the Web Service method according to a schema. In practice, this schema is usually expressed using a XML Schema. The data is serialized according to this schema. Although there are no prescribed rules for serializing objects, structures, and graphs, etc., the service's schema describes the application-level information set of each of the service's messages. A document-style SOAP request generally applies a XML Schema complex type to its body as a whole. The assumption is that this complex type is specified somewhere, or at least can be generated automatically on demand. The place where it is generally specified is in a WSDL file. This is the "literal" alternative to SOAP-encoding.

Earlier many SOAP toolkits, with the notable exception of Visual Studio .NET, only supported the RPC/Encoded encoding. But after Microsoft adopted the Document/Literal encoding as the .NET Framework’s default SOAP message format the industry appears to be standardizing on the Document/Literal message format.

**SOAP Headers**

SOAP headers provide a modular and flexible way to extend the SOAP message. Typical extensions put in headers are authentication and transaction management. These can be added without any prior communication between client and server. All the elements under the header element are called header entries. They can all contain header attributes that for example determine how a recipient of a message should process the message. One example is the mustUnderstand attribute that indicates whether a header entry is mandatory or optional for the recipient to process or not. Another attribute is the action attribute. A SOAP message can potentially before arriving to the final recipient pass by various intermediaries. A SOAP intermediary is an application that is capable of both receiving and forwarding SOAP messages. Some parts of the SOAP message may not be intended for the ultimate recipient but to some of the intermediaries. In this case the action attribute is used to indicate the recipient of that header element.

**2.3.5 Examples**

In the SOAP examples chapter certain knowledge of XML [1] and XML Schemas [8] is assumed. Three different examples of SOAP messages are given. The first example is meant to show the basic structure. The second example shows the use of SOAP headers. The third and last example shows how a SOAP message can be sent in a
HTTP request.

**Nominalia Web Service example**

Example 1: SOAP Message from the Nominalia checkDomain operation.

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

The first line is the XML declaration that defines the character set used in the SOAP document. The Envelope element (I) is a required element that is always the root of a SOAP message and is always in the http://schemas.xmlsoap.org/soap/envelope namespace. The Body sub element (II) of the Envelope element contains the SOAP Message. In this example the Body contains a complex type `checkDomain` (III) that has two fields `sld` and `tld`.

The response to this message has the same structure and could look like this:

```xml
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <checkDomainResponse
        xmlns="http://www.nominalia.com/operations">
      <checkDomainResult>1</checkDomainResult>
    </checkDomainResponse>
  </soap:Body>
</soap:Envelope>
```

**SOAP Header example**

The listing in Example 2 shows a sample SOAP message that is an alert message to a Web Service. The request contains a text message (in the Body) and is marked to indicate that the message is not interesting after the given time (in the Header). The `alert` element represents a call on an operation named `alert`. The sub elements are the parameters of the method call in this case the `msg` element.
Example 2: SOAP message with headers

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<env:Envelope xmlns:env="http://www.w3.org/2001/09/soap-envelope">
  <env:Header>
    <n:alertcontrol
        xmlns:n="http://example.org/alertcontrol">
      <n:priority>1</n:priority>
      <n:expires>2001-06-22T14:00:00-05:00</n:expires>
    </n:alertcontrol>
  </env:Header>
  <env:Body>
    <m:alert xmlns:m="http://example.org/alert">
      <m:msg>Pick up Mary at school at 2pm</m:msg>
    </m:alert>
  </env:Body>
</env:Envelope>
```

SOAP over HTTP example

In this example taken from the SOAP 1.1 specification, a GetLastTradePrice SOAP request is sent to a StockQuote service. The request takes a string parameter, symbol, and returns a float. The example also illustrates the HTTP binding. It is worth noting that the rules governing XML payload format in SOAP are entirely independent of the fact that the payload is carried in HTTP. In the envelope tag it also specifies that the SOAP Encoding message format is used.

Example 3: SOAP message embedded in a HTTP request

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="Some-URI">
      <symbol>DIS</symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Following is the response message:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:GetLastTradePriceResponse xmlns:m="Some-URI">
      <Price>34.5</Price>
    </m:GetLastTradePriceResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
2.4 Web Service Description Language

“WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate, however, the only bindings described in this document describe how to use WSDL in conjunction with SOAP 1.1, HTTP GET/POST, and MIME.” - from the WSDL 1.1 Specification [7].

2.4.1 Introduction

As communication protocols and message formats are being more and more standardized in the web community, it is important to be able to describe the communications in some structured way. WSDL addresses this need by defining a XML grammar for describing and accessing network services. WSDL service definitions provide documentation for distributed systems and help the automation of applications communication.

A WSDL document defines:
- What operations are available from the service.
- What are the formats of the messages exchanged between the client and the server, both to invoke the services and to interpret their results.
- How the services are accessed: addresses and protocols.

In this chapter certain knowledge of XML [1] and XML Schemas [8] is assumed.

2.4.2 WSDL Document

A WSDL document is simply a set of definitions. Figure 2-5 visualizes the WSDL elements described in this section. It shows a client invoking a Web Service in two different ways: Using SOAP and using HTTP GET. Each invocation consists of a request and a response message.
Hence, a WSDL document uses the following elements in the definition of Web Services:

- **Types** – The data types used in the messages
- **Message** – A typed definition of the in and out messages to an operation
- **Operation** – An abstract description of an action supported
- **Port Type** – An abstract definition of a service, a set of operations
- **Binding** – The communication protocol and data format used
- **Port** – A single network endpoint combined with a binding
- **Service** – A collection of related ports

See Figure 2-6 for a graphical representation of the WSDL structure.

WSDL does not introduce a new type definition language. It recognizes the need for rich type systems for describing message formats, and supports the XML Schemas specification (XSD) [8] as its canonical type system. In addition, it defines a common binding mechanism. This is used to attach a specific protocol or data format or structure to an abstract message, operation, or endpoint.

The WSDL specification introduces specific binding extensions for the following protocols and message formats:

- SOAP 1.1, HTTP GET / POST and MIME
A port is defined by associating a network address with a reusable binding and a collection of ports define a service. In WSDL, the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. The concrete protocol and data format specifications for a particular port type constitute a reusable binding.

![Figure 2-6. WSDL Structure.](image)

### 2.4.3 Document grammar

Here follows the grammar of a normal WSDL document. The following informal document syntax is used:

- Characters are appended to elements and attributes as follows:
  - ‘?’ - 0 or 1 occurrences
  - ‘*’ - 0 or more
  - ‘+’ - 1 or more
- A **qname** is a XML qualified name, for example to another WSDL definition.
- A **nmtoken** is an identifier of an element.
<wsdl:definitions name="nmtoken" targetNamespace="uri">  
    <import namespace="uri" location="uri"/>
      
    <wsdl:types>
      <wsdl:documentation />
    </wsdl:types>
    <wsdl:message name="nmtoken">
        <wsdl:documentation />
        <part name="nmtoken" element="qname" type="qname"/>
    </wsdl:message>
    <wsdl:portType name="nmtoken">
        <wsdl:documentation />
        <wsdl:operation name="nmtoken">
            <wsdl:input name="nmtoken" message="qname"/>
            <wsdl:output name="nmtoken" message="qname"/>
            <wsdl:fault name="nmtoken" message="qname"/>
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="nmtoken" type="qname">
        <wsdl:documentation />
        <wsdl:operation name="nmtoken">
            <wsdl:input/>
            <wsdl:output/>
            <wsdl:fault name="nmtoken"/>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="nmtoken">
        <wsdl:port name="nmtoken" binding="qname"/>
    </wsdl:service>
    <wsdl:documentation />
</wsdl:definitions>
2.4.4 Document elements

As mentioned earlier, services are defined in WSDL using the six major elements:

- **Types** – which provides data type definitions used to describe the messages exchanged.
- **Message** – which represents an abstract definition of the data being transmitted. A message consists of logical parts, each of which is associated with a definition within some type system.
- **portType** – which is a set of abstract operations. Each operation refers to an input message and output messages. This could be seen as a library or a module.
- **Binding** – which specifies concrete protocol and data format specifications for the operations and messages defined by a particular portType.
- **Port** – which specifies an address for a binding, thus defining a single communication endpoint. For example an URL.
- **Service** – which is used to aggregate a set of related ports.

Types

The types element encloses data type definitions that are relevant for the exchanged messages. For maximum interoperability and platform neutrality, WSDL prefers the use of XSD as the canonical type system, and treats it as the intrinsic type system.

```xml
<definitions .... >
   <types>
      <xsd:schema .... />*
   </types>
</definitions>
```

The XSD type system can be used to define the types in a message regardless of whether or not the resulting wire format is actually XML, or whether the resulting XSD schema validates the particular wire format.

Messages

Messages consist of one or more logical parts. The parts can be compared to the parameters of a function call in a traditional programming language. Each part is associated with a type from some type system using a message-typing attribute. WSDL defines several such message-typing attributes using the XML Schema definition.
The syntax for defining a message is as follows.

```xml
<definitions .... >
  <message name="nmtoken"> *
    <part name="nmtoken" element="qname"? type="qname"/> *
  </message>
</definitions>
```

- **element** – Refers to an XSD element using a `qname`.
- **type** – Refers to an XSD simpleType or complexType using a `qname`.

The message name attribute provides a unique name among all messages defined within the WSDL document. The part name attribute provides a unique name among all the parts of the enclosing message.

**Port types and operations**

A port type is a named set of abstract operations and the abstract messages involved. The port types are protocol independent.

```xml
<wSDL:definitions .... >
  <wSDL:portType name="nmtoken">
    <wSDL:operation name="nmtoken" .... /> *
  </wSDL:portType>
</wSDL:definitions>
```

The port type name attribute provides a unique name among all port types defined within in the enclosing WSDL document. An operation is named via the name attribute.

WSDL has four transmission primitives that an endpoint can support:

- **One-way** – The endpoint receives a message.
- **Request-response** – The endpoint receives a message, and sends a correlated message.
- **Solicit-response** – The endpoint sends a message, and receives a correlated message.
- **Notification** – The endpoint sends a message.

WSDL refers to these primitives as operations. Although the base WSDL structure supports bindings for these four transmission primitives, WSDL only defines bindings for the One-way and Request-response primitives.
Bindings

A binding defines message format and protocol details for operations and messages defined by a particular portType. There may be any number of bindings for a given portType. The grammar for a binding is as follows:

```xml
<wsdl:definitions .... >
  <wsdl:binding name="nmtoken" type="qname"> *
    <-- extensibility element (1) --> *
    <wsdl:operation name="nmtoken"> *
      <-- extensibility element (2) --> *
      <wsdl:input name="nmtoken"/> *
        <-- extensibility element (3) --> *
      </wsdl:input>
      <wsdl:output name="nmtoken"/> *
        <-- extensibility element (3) --> *
      </wsdl:output>
      <wsdl:fault name="nmtoken"> *
        <-- extensibility element (4) --> *
      </wsdl:fault>
    </wsdl:operation>
  </wsdl:binding>
</wsdl:definitions>
```

The name attribute provides a unique name among all bindings defined within in the enclosing WSDL document.

A binding references the portType that it binds using the type attribute.

Binding extensibility elements are used to specify the concrete grammar for:
- the input and output (3) messages
- the fault messages (4)
Per-operation binding information (2) as well as per-binding information (1) may also be specified.

The soap:binding element has two attributes - the style attribute and the transport attribute. The style attribute can be "RPC" or "Document". The transport attribute defines the SOAP protocol to use, for example HTTP.

For the input and output elements the encoding has to be specified, for example “Encoded” or “Literal”.

A binding must specify exactly one protocol.
A binding must not specify address information.
Ports

A port defines an individual endpoint by specifying a single address for a binding.

```xml
<wSDL:definitions .... >
    <wSDL:service ...... > *
        <wSDL:port name="nmtoken" binding="qname"> *
            <-- extensibility element (1) -->
        </wSDL:port>
    </wSDL:service>
</wSDL:definitions>
```

The name attribute provides a unique name among all ports defined within in the enclosing WSDL document. The binding attribute refers to the binding using the linking rules defined by WSDL. Binding extensibility elements (1) are used to specify the address information for the port.

A port must not specify more than one address.
A port must not specify any binding information other than address information.

Services

A service groups a set of related ports together:

```xml
<wSDL:definitions .... >
    <wSDL:service name="nmtoken"> *
        <wSDL:port .... />*
    </wSDL:service>
</wSDL:definitions>
```

The name attribute provides a unique name among all services defined within in the enclosing WSDL document.

Ports within a service have the following relationship: None of the ports communicate with each other (e.g. the output of one port is not the input of another). If a service has several ports that share a port type, but employ different bindings or addresses, the ports are alternatives. Each port provides semantically equivalent behaviour (within the transport and message format limitations imposed by each binding). This allows a consumer of a WSDL document to choose particular port(s) to communicate with based on some criteria (protocol, distance, etc.).

By examining its ports, we can determine a service’s port types. This allows a consumer of a WSDL document to determine if it wishes to communicate to a particular service based whether or not it supports several port types. This is useful if there is some implied relationship between the operations of the port types, and that the entire set of port types must be present in order to accomplish a particular task.

For a more detailed description of these elements see the Sections 2.2 to 2.7 in the WSDL 1.1 Specification [7].
2.4.5 Examples
Following is an example of each part of a WSDL document mentioned above, all taken from the Nominalia Web Service WSDL document. The complete WSDL document of the Nominalia Web Service can be found at the following URL: https://ws.nominalia.com/ws/NominaliaWS.asmx?WSDL

Types
Here is a snippet of two of the types defined in the WSDL document, both defined using XSD. The first element is a part of the input message to the GetRequestStatus operation that also can be seen in the message example. The second element is the complex type Request that the metadata send with all operations.

```
<types>
  <s:schema elementFormDefault="qualified"
    targetNamespace="http://ws.nominalia.com/operations">
    <s:element name="GetRequestStatus">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="env" type="s0:Request" />
          <s:element minOccurs="1" maxOccurs="1" name="requestId" type="s:int" />
        </s:sequence>
      </s:complexType>
    </s:element>
    <s:complexType name="Request">
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="env_version" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="env_wholesaler" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="env_login" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="env_password" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="env_language" type="s:string" />
      </s:sequence>
    </s:complexType>
  </s:schema>
</types>
```

Messages
Here is a snippet of two defined messages. The first message is the input message to the GetRequestStatus operation that can be seen later in the operations example. The second element is the message sent to the CheckDomain operation.

```
<message name="GetRequestStatusSoapIn">
  <part name="parameters" element="s0:GetRequestStatus" />
</message>

<message name="checkDomainHttpGetIn">
  <part name="sld" type="s:string" />
  <part name="tld" type="s:string" />
</message>
```

Port Types and operations
Here is a part of the portType that specifies the SOAP port type of the Nominalia Web Service. It shows the first two operations: checkDomain and getRequestStatus, both specifying their corresponding input and output messages.
<portType name="NominaliaWSSoap">

<operation name="checkDomain">
  <documentation>Check availability of a domain</documentation>
  <input message="s0:checkDomainSoapIn" />
  <output message="s0:checkDomainSoapOut" />
</operation>

<operation name="GetRequestStatus">
  <documentation>Get the status of a request</documentation>
  <input message="s0:GetRequestStatusSoapIn" />
  <output message="s0:GetRequestStatusSoapOut" />
</operation>
...
</portType>

**Bindings**

This example shows the SOAP binding and the checkDomain operation that is specifying its message format.

<binding name="NominaliaWSSoap" type="s0:NominaliaWSSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
  <operation name="checkDomain">
    <input>
      <soap:body use="literal" />
    </input>
    <output>
      <soap:body use="literal" />
    </output>
  </operation>
  ...
</binding>

**Services and ports**

Finally, all within the NominaliaWS service, the SOAP port is linked together with the SOAP binding and is associated with a communication endpoint, in this case the URL to the Nominalia Web Service.

<service name="NominaliaWS">
  <documentation>Nominalia Webservices</documentation>
  <port name="NominaliaWSSoap" binding="s0:NominaliaWSSoap">
    <soap:address location="http://ws.nominalia.com/ws/NominaliaWS.asmx" />
  </port>
  ...
</service>
2.5 Universal Description, Discovery and Integration

“UDDI is an industry specification for publishing and locating information about Web Services. It defines an information framework that enables you to describe and classify your organization, its services, and the technical details about the interfaces of the Web Services you expose.” – definition taken from the UDDI.org web page [9].

2.5.1 Introduction

UDDI provides a way to locate and understand services provided by other companies.
- It is a kind of "Yellow Pages" on the Internet for the industry.
- It is a platform independent framework for describing services, discovering businesses, and integrating business services by using the Internet.

UDDI is the name of a group of web-based registries that expose information about a business and its technical interfaces or Application Programming Interfaces (API’s). These registries are run by multiple Operator Sites, and can be used by anyone who wants to make information available about some businesses or entity, as well as anyone that wants to find that information. There is no charge for using the basic services of these operator sites. By accessing any of the public UDDI Operator Sites, anyone can search for information about Web Services that are available.

The benefit of having access to this information is to provide a mechanism that allows others to discover what technical programming interfaces are provided for interacting with a service. The benefit to the individual business is increased exposure of its services.

The information that a business can register includes several kinds of data:
- Simple information about a business – such as name, business identifiers and contact information as well as descriptive information about the services that the business makes available.
- Information about the address through which each type of service is accessed: URL, email address or any other address.
- References to information about interfaces and other properties of a given service. These service properties describe a particular software package or technical interface and are referenced as tModels in the UDDI documentation.

The UDDI framework defines a set of Application Programming Interfaces (APIs) that can be used by applications and services to interact with UDDI data directly. For example, services can be developed that publish and update their UDDI data automatically, react dynamically to service availability, or automatically discover interface details for other services with which they interact.
2.5.2 Industry support

UDDI is making a strong effort to get support from the industry by adapting common standards and attracting new important members to the community.

- UDDI is a cross-industry effort driven by all major platform and software providers like Microsoft, Compaq, Dell, HP, IBM, SAP, Intel, Oracle, SUN and others.
- UDDI uses Internet standards such as XML, HTTP and WSDL to describe interfaces to Web Services.
- Additionally, cross platform programming features are addressed by adopting SOAP.
- Over 220 companies are members of the UDDI community.
- Registries
  - IBM – www.ibm.com/services/uddi/
  - Microsoft – http://uddi.microsoft.com/

2.5.3 Registry information

The UDDI XML Schema defines four core data types for business and service information. All the data in a UDDI repository must consist of one of these four types. UDDI defines each type with an XML-based data structure and each contains mandatory and optional fields. The four core data types are shown in Figure 2-7.

Figure 2-7. UDDI core structures.

Broadly, the four data types can be classified into “Business information” and
“Service information” categories.

**Business information**

Business information defines the concept of a businessEntity element, which is analogous to the “White Pages” and “Yellow Pages” sections of a telephone directory. During the discovery phase, the businessEntity element serves as a handle to retrieve pertinent information about a company. A businessEntity element contains the following types of information:

- Company name
- Business description and industry codes
- The products and services offered
- The geographical location the company serves

**Service information**

Service information defines the notion of a businessService element and a bindingTemplate element. The businessService element stores technical and business descriptions of Web Services offered by a business. The businessService element contains a bindingTemplate element that stores the technical description of the Web Service. In other words, the businessService element contains information about the Web Services being offered, but the bindingTemplate element contains the information required to invoke them.

The bindingTemplate element has an accessPoint attribute that contains the address of the Web Service and a tModelInstanceDetails element that contains an identifier called tModels, which are blueprints that can be used to identify compatibility between the requesting agent and the published service.

**2.5.4 Discovery, lookup and invocation workflow**

In short, the process for using the UDDI registry is:

1. Use the UDDI repository to locate a businessEntity element for the appropriate business offering Web Services.
2. Locate the businessService element to identify all the Web Services offered by this businessEntity.
3. Select a bindingTemplate element to retrieve the address of the Web Service and tModel element to ensure technical compatibility between the systems.

**2.5.5 Benefits**

Before UDDI, there was no Internet standard for businesses to reach their customers and partners with information about their products and services. Nor was there a method of how to integrate into each other's systems and processes. Problems the UDDI specification can help to solve:
- Making it possible to discover the right business from the millions currently online.
- Reaching new customers and increasing access to current customers.
- Describing services and business processes programmatically in a single, open, and secure environment.

2.5.6 Limitations

For the moment the actual use of the UDDI registries is quite limited depending mainly on the following reasons.

- Few real services published in the registries.
- A lot of “spam” and incorrect information published.
- Primitive querying language and very limited information published.
3 Application requirement specifications

In the specification phase the functional requirements of the project are formalized. The requirement specifications of the system are documented with UML [10] through use case diagrams and use case specifications. The functionality is documented with class- and sequence diagrams describing the different parts of the system. The requirements of the Web Service application have been specified in various workshops together with professionals from Nominalia. In addition to know what entities that have to be represented in the new platform the existing systems of Nominalia have been studied.

3.1.1 Use case diagram
Use case diagrams are used to specify the functionality and behaviour of the system describing the interaction between the users and the system. Here the use case diagram is used to give an overview of the functionality given by the system.

3.1.2 Use case specification
In the use case specification each use case is defined with a text describing its functionality. In this description the interaction between the actors and the system and the internal functionality are described in more detail.

3.1.3 Class diagrams
A class diagram is a representation of the static elements of a system. The static elements described in this diagram are of the types: class, relation and interface. The classes can be either concrete or abstract. If the class is abstract it cannot be instantiated. This is indicated with the class name written in italics in the diagram. The relations can be of the types association or generalization. Association describes a relation between two classes. The class that in the diagram is pointing to another class has access to that other class. The normal way to implement an association is that one class has an instance of the other class as a variable. Generalization is where one class is the super class of other classes who inherits the methods and attributes of the super class.

3.1.4 Sequence diagrams
A sequence diagram describes the dynamic interactions between the different parts of a system, ordered in time. The diagram shows what information must be sent between the different parts of the system and how the parts are related. In this case more than one use case works in a similar way so they are grouped into common use cases that are described with sequence diagrams. In this report the sequence diagrams are used to give an understanding of architecture and not implementation details.
3.2 Glossary

3.2.1 Domain

On the Internet, a domain consists of a set of network addresses. This domain is organized in levels.

- The top level identifies geographic or purpose commonality (for example .es or .com).
- The second level identifies a unique place within the top-level domain and is equivalent to a unique address on the Internet (an IP address).

Second-level domain names must be unique on the Internet and registered with one of the ICANN-accredited registrars for the COM, NET, and ORG top-level domains, for example Nominalia.

In the Nominalia system a domain is a unique combination of a top-level and second-level domain. For example: nominalia.com.

On the Web, the domain name is that part of the Uniform Resource Locator (URL) that tells a domain name server using the domain name system (DNS) where to forward a request for a Web page. The domain name is mapped to an IP address, which represents a physical point on the Internet that in the Nominalia system called a Host.

3.2.2 Host

In Internet protocol specifications, the term host means any computer that has full two-way access to other computers on the Internet. A host has a specific local or host number that, together with the network number, forms its unique IP address.

In the Nominalia system every host has in addition a responsible technical contact.

3.2.3 Contact

In the Nominalia system there are three kinds of contacts associated to a domain.

- Administrative
- Business
- Technical

These are all persons that are responsible for different aspects associated to the domain. They could in practice be the same person. A technical contact is also associated to every host.
3.3 Use case diagram

Each operation of the Nominalia Web Service is described as a use case. There are also two internal use cases that are used by many of the others. Figure 3-1 illustrates the use case model of these use cases.

![Use case model diagram]

The use cases “GetRequestStatus” and “Create or modify entity” are described with their corresponding sequence diagrams in chapter 3.1.4. As “Get entity info” and “CheckDomain” are simplified versions of the “Create or modify entity”, they are not documented with sequence diagrams.
3.4 Use case specification

Every use case represents an operation of the Nominalia Web Service. All these operations are invoked with an operation header specifying username and password and other environment variables.

The information retrieval operations and the getRequestStatus operation will only allow the retrieval of information belonging to the user executing the operation.

NewDomain

The NewDomain operation allows the client to create a new domain specifying the domain specific attributes and its associated entities described in the class diagram.

ModifyDomain

The ModifyDomain operation allows the client to modify an already existing domain specifying the domain and supplying all the data associated to the domain. As all data, even the one that not will be changed, have to be send as parameters the GetDomainInfo can be invoked before to get the existing data. The domain name can never be changed.

GetDomainInfo

The GetDomainInfo operation retrieves all the information associated with a specific domain.

NewHost

The NewHost operation allows the client to create a new host specifying the host specific attributes described in the class diagram; name, IP and technical contact.

ModifyHost

The ModifyHost operation allows the client to modify the IP or technical contact of an existing host specifying the host’s name or IP and the new data. As all data, even the one that not will be changed, has to be send as parameters the GetHostInfo can be invoked before to get the existing data.

GetHostInfo

The GetHostInfo operation retrieves the information associated with a host specifying the host’s name or IP.
NewContact
The NewContact operation allows the client to create a new contact specifying the contact specific attributes described in the class diagram.

ModifyContact
The ModifyContact operation allows the client to modify the data of an existing contact specifying the contact id and the associated data.

GetContactInfo
The GetContactInfo operation retrieves the information associated with a contact specifying the contact id.

GetContactListInfo
The GetContactListInfo operation retrieves the information about all contacts specifying a contact name.

GetRequestStatus
The GetRequestStatus operation is used to retrieve the status of an operation. Can be used to check if an operation is done after a time out or to get the result of an old operation.

CheckDomain
The CheckDomain operation checks the availability of a domain. Given the domain it returns either true or false.
3.5 Class diagram

Figure 3-2 shows the classes representing the main business entities of Nominalia.
In Figure 3-3 the most important classes that contain the implementation logic of the Web Service are shown. The configuration file of the Web Service module is also modelled to show its attributes.
3.6 Class descriptions

A class is a programming entity that represents a real life object. It has attributes to describe its characteristics. It has methods that execute operations on these attributes or other tasks that the class is responsible for. When a class is instantiated and represents a specific real life object, it is called an object.

Following is a short explanation of the main classes of the Web Service application:

Communicator
The Communicator class is responsible of all communication with the Nominalia back-end system.

Contact
This class encapsulates the data that comprise a contact associated with a domain. A contact type attribute has been added in addition to what the Nominalia system uses. This due to the way that the contacts associated to a domain are represented in the Nominalia Web Service operations. The contact type can be one of: administrative, business or technical.

DatabaseManager
The DatabaseManager class is responsible of all communication with the internal Web Service database.

DBRequest
The DBRequest class is a wrapper class for the data stored in the cache, used by the getRequestStatus operation.

Domain
This class encapsulates the data that comprise an Internet domain and the methods used to manipulate this data.

Entity
This is the abstract base class for the main business objects. It has two abstract helper methods that have to be implemented by classes that inherit from this class.

Host
This class encapsulates the data that comprise a host associated with a domain. The hosts specify the domain name servers associated to the Domain. An order attribute is added in addition to what is needed to represent a Host in the Nominalia system due to the way that the hosts associated to a Domain are represented in the Nominalia Web Service operations.

HostList and ContactList
These two auxiliary classes are added because of the way that .NET serializes arrays
to XML according to the SOAP representation of arrays.

**NameValueCollection**

The NameValueCollection class is provided by the .NET framework and is used to encapsulate values that are sent by HTTP POST.

**NominaliaService**

This class is responsible for the logic of the Nominalia cleaning service.

**Owner**

This class encapsulates the data that comprise an owner of a domain.

**Response**

The response class encapsulates all the environment variables that are returned by the Nominalia system when invoking an operation. It is the base class of all the entity specific responses, which also contain its corresponding entity.

**Request**

The request class encapsulates all the environment variables, for example user, password and language, that are send to the Nominalia system when invoking an operation. It is the base class of all the entity specific requests, which in addition also contain its corresponding entity.

**Timer**

This class is provided by the .NET framework and is used to execute the Nominalia Service on a regular basis.
3.7 Sequence diagrams

Here the Create and modify entity and the GetRequestStatus use cases are described in detail with their corresponding sequence diagram. These diagrams are more meant to explain how the system works than describing some specific operation in detail. The two last sequence diagrams illustrate how the Nominalia cleaning service works.

3.7.1 Use case: Create or modify entity

The sequence diagram in Figure 3-4 shows how the different parts of the system interact when a client invokes an operation to create or modify a business entity.
3.7.2 Use case: GetRequestStatus

The sequence diagram in Figure 3-5 shows how the GetRequestStatus operation is using the cache to minimize the database load.

![Sequence diagram for GetRequestStatus operation]

Figure 3-5. Sequence diagram – GetRequestStatus operation

3.7.3 Nominalia Service initialisation.

Figure 3-6 shows the sequence diagram of the initialisation of the Nominalia Service.

![Sequence diagram for Nominalia Service initialisation]

Figure 3-6. Sequence diagram - Nominalia Service initialisation.
3.7.4 Nominalia Service execution.

The sequence diagram in Figure 3-7 illustrates an execution of the Nominalia Service.

![Nominalia Service execution diagram](image)

**Figure 3-7. Nominalia Service execution.**

3.8 Interface design

In Web Service applications there are no visible user interfaces involving text or graphics. The interface is the way the operations are exposed to the client and the design of the in and out parameters of the SOAP messages for the different operations. In this project the SOAP messages have been designed using an object orientated approach. This is characterized by an extensive use of complex types in the SOAP requests and responses. Encapsulation of the different attributes as objects and the use of inheritance map very well to an object oriented environment like .NET or Java, but is less suitable for other platforms like PERL and PHP.

Currently the SOAP standard has in practice two message formats. One is the document-literal format that is the most general and firmly supported by Microsoft. The other is the RPC-Encoded format that follows a more strict remote procedure call format. See the SOAP chapter for more information about the SOAP message formats. To be accessible by the most possible number of clients the Nominalia Web Service has two different endpoints supporting the two formats.

Normally, and in the case of this project, first the service application code is written and the public methods are defined. Then with the use of some development tool, like Visual Studio .NET, it is possible to automatically publish a WSDL description and create client proxy code. The SOAP message formats used to communicate between the client and the service applications will then conform to the WSDL document,
without having to be concerned about exactly what these message formats are. It is sufficient they are consistent with the implemented code. The other more complex case is that a Web Service implementation has to be built conforming to some predefined SOAP messages. This case presents various additional complications. It requires a careful development of the object model and the use of XML serialization attributes to insure conformant SOAP messages. To do this manually, starting from a WSDL document, a fairly deep understanding of WSDL, SOAP, XML Schema and the SOAP serialization functionality of the actual platform is required.

3.9 Information sources

The information needed for this project has been taken from two main sources. All the information needed to connect the new Web Service platform to the Nominalia backend systems has been taken from existing manuals and documentation provided by Nominalia. The information about the Web Services technologies has been taken mainly from the World Wide Web Consortium (W3C) web page and the Microsoft Developers Network. See biography.

3.10 Constraints

The technical constraints set upon this project are mainly due to the scope of the project. To ensure backward compatibility for all existing clients the actual platform were not to be changed in this first phase of the implementation.

After the choice of technical platform, in this case the Microsoft .NET platform, many subsequent choices (for example development tools and database) were quite obvious for ease of integration reasons.
4 The Web Service application

4.1 Introduction

In this chapter a description of the different modules that comprise the Nominalia Web Service application is given. After the overview of the modules follows a short description of how the development has been conducted. Finally each module is described in more detail: administration, configuration and dependences.

The Nominalia Web Service application is divided into four separate modules.

- **Web Service:** This is the main module of the application, the actual Web Services that run as a Web application inside the Web Server. The operations that it offers are described in chapter 3.3 and 3.4.

- **Database:** The other core module is the database that is used for maintaining state, logging and recovery of failed operations.

The two remaining modules are in charge of the housekeeping of the application.

- **Archiving Scheduled Task:** This a stand-alone application that is used to archive old requests from the database to files. The application is run as a Scheduled Task of the operating system. It archives all requests older than one month.

- **Windows Service:** The last module manages non-successful operations and is an agent that runs in the background as a Windows service. It periodically checks if there is any failed operations that should be invoked again.

4.1.1 Implementation history

Initially one operation, to create a new domain, was implemented to create the framework and the main classes for the integration with the Nominalia backend system. After this first operation and the communication framework was done, the rest of the create and modify operations of all business entities were added. Then the polling operation to get the status of an operation was added together with the database cache. Then all the information-retrieving operations were implemented.

To manage failed operations a cleaning agent that is responsible for resending them to the Nominalia backend system was implemented. To maintain the internal database the archiving application was implemented. After the whole application was finished, a suite of tests was implemented and tests were performed with real data from Nominalia to prove that the application fulfilled the functional and technical requirements. Two example clients on different platforms were implemented and added to the user manual. Finally after the application had been installed and tested at Nominalia it has been installed at a hosting company and is currently in production.
4.2 **Web Service module**

The Web Service module is the main module of the application where the actual Web Services are. It runs as a Web application in the Web Server, Internet Information Server 6.0 (IIS).

There exist two access points to the applications depending on the SOAP message format that the client wants to use. The default access point, using the document literal format, is the *NominaliaWS.asmx* in the root directory of the Nominalia Web Service application. The other, for the clients that use the RPC-Encoding format, resides in the /RPC directory.

The two access points of the application are:
- [https://ws.nominalia.com/ws/NominaliaWS.asmx](https://ws.nominalia.com/ws/NominaliaWS.asmx)
- [https://ws.nominalia.com/ws/RPC/NominaliaWS.asmx](https://ws.nominalia.com/ws/RPC/NominaliaWS.asmx)

See figure B-1 in the User manual appendix for a screen shot of the default access point.

4.2.1 **Architectural overview**

Figure 4-1 illustrates an architectural overview of the Web Service module. It is shown with who and how it interacts.

![Figure 4-1. Architectural overview - Web Service module.](image-url)
4.2.2 Configuration

The Nominalia Web Service has five parameters that can be set in the application configuration file web.config, a XML file that can be found in the application directory. These parameters are shown in Table 4-1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectionString</td>
<td>The connection string to connect to the SQL Server database.</td>
</tr>
<tr>
<td>dna_url</td>
<td>The URL of the dna.cgi script. This can be set to the DNANominalia.aspx page for local testing.</td>
</tr>
<tr>
<td>answer_url</td>
<td>The URL of the answer_url page to which the dna.cgi script will send its response</td>
</tr>
<tr>
<td>whois_url</td>
<td>The URL of the PublicWhois.v4.php script.</td>
</tr>
<tr>
<td>asinc_timeout</td>
<td>The TIMEOUT of the Web Service call to the dna.cgi script. If the dna.cgi script has not responded within the given time the Web Service returns an operation id.</td>
</tr>
</tbody>
</table>

Table 4-1. Configuration parameters

Following is an example listing of the Nominalia Web Service configuration file. The parts that do not concern the Web Service application are not shown.

```xml
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.web>
    <!-- The connection string to connect to the SQL Server database. -->
    <add key="ConnectionString" value="Integrated Security=SSPI;Initial Catalog=nominalia;Data Source=(local)" />
    <!-- The URL of the dna.cgi script. This can be set to the DNANominalia.aspx page for local testing. -->
    <add key="dna_url" value="https://secure.nominalia.com/ws_gateway/ws_gateway.php" />
    <!-- The URL of the answer_url page to which the dna.cgi script will send its response -->
    <add key="answer_url" value="http://147.83.59.181/WSNominalia/Answer.aspx" />
    <!-- The URL of the PublicWhois.v4.php script. -->
    <add key="whois_url" value="https://secure.nominalia.com/utils/PublicWhois.v4.php" />
    <!-- Time in milliseconds that the Web Service waits for a response from the dna script before it returns an id to the client. Minimum value 1000 -->
    <add key="asinc_timeout" value="1000" />
  </system.web>
</configuration>
```

Listing: Nominalia Web Service configuration file, web.config (partial)
4.2.3 Administration

The Nominalia Web Service is administrated as a normal Web application from the Internet Services Manager of the IIS, see Figure 4-2, that is accessible from the administrative tools of the control panel of the operating system. Here it can be started and stopped. The default page should be set to the main page of the application, NominaliaWS.asmx.

![Image: The IIS administration of the Web Service module.]

Figure 4-2. The IIS administration of the Web Service module.

Security

The access to the Answer.aspx page must be limited to only accept calls from the backend machine of Nominalia.

Dependencies

The Web Service module uses the following stored procedures from the database module:

- `insertBeginRequest`
- `updateRequest`
- `insertEndRequest`

The Web Server process (ASPNET) must have access right to these stored procedures.

4.2.4 Error and logging

User-friendly error messages are returned to the user explaining the source of the error if possible. In the case of more severe errors or system failures, a general “Please try again later” error is returned to the client and the internal error is logged to the Application Log of the operating system. This log can be accessed via the Event Viewer found in the Administration Tools of the control panel, see Figure 4-8.
4.2.5 Files

Here follows a list of the files that the application consists of. The files are grouped by directory where they reside on the file system.

Application root directory

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NominaliaWS.asmx</td>
<td>Web Service file, access point of the application. Public</td>
</tr>
<tr>
<td>NominaliaWS.asmx.cs</td>
<td>Source code behind of all the methods of the NominaliaWS.aspx page.</td>
</tr>
<tr>
<td>Answer.aspx</td>
<td>Web page used receive the response from the Nominalia backend system.</td>
</tr>
<tr>
<td>Answer.aspx.cs</td>
<td>Source code behind the Answer.aspx page.</td>
</tr>
<tr>
<td>Comunicator.cs</td>
<td>Source code of the Comunicator class responsible of the communication with the Nominalia backend systems.</td>
</tr>
<tr>
<td>DatabaseManager.cs</td>
<td>Source code of the DatabaseManager class responsible of the communication with the internal database.</td>
</tr>
<tr>
<td>Whois.cs</td>
<td>Source code of the Whois class that is responsible for all the communication with the PublicWhois.v4.php script.</td>
</tr>
<tr>
<td>Web.config</td>
<td>XML configuration file</td>
</tr>
</tbody>
</table>

Bin directory

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NominaliaWS.dll</td>
<td>Dynamic link library. This is the binary created from all the source code of the Web Service module.</td>
</tr>
</tbody>
</table>

RPC directory

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NominaliaWS.asmx</td>
<td>Web Service file, access point of the application for RPC-Encoded clients. Public</td>
</tr>
<tr>
<td>NominaliaWS.asmx.cs</td>
<td>Source code behind the NominaliaWS.aspx page. References to the corresponding methods of the root NominaliaWS.</td>
</tr>
</tbody>
</table>
### Entity directory

<table>
<thead>
<tr>
<th>Entity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity.cs</td>
<td>The base class of the entity classes.</td>
</tr>
<tr>
<td>Contact.cs</td>
<td>Source code of the class that encapsulates the information of this entity. Contains methods to serialize and deserialize the entity to and from a querystring.</td>
</tr>
<tr>
<td>ContactList.cs</td>
<td>-</td>
</tr>
<tr>
<td>Domain.cs</td>
<td>-</td>
</tr>
<tr>
<td>Host.cs</td>
<td>-</td>
</tr>
<tr>
<td>HostList.cs</td>
<td>-</td>
</tr>
<tr>
<td>Owner.cs</td>
<td>-</td>
</tr>
</tbody>
</table>

### Request directory

<table>
<thead>
<tr>
<th>Request</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request.cs</td>
<td>Base class of the Request classes. Contains the environment information sent with an operation and methods to serialize it.</td>
</tr>
<tr>
<td>ContactRequest.cs</td>
<td>Source code of the class that encapsulates the in parameters of the corresponding operations.</td>
</tr>
<tr>
<td>HostRequest.cs</td>
<td>-</td>
</tr>
<tr>
<td>DomainRequest.cs</td>
<td>-</td>
</tr>
<tr>
<td>OwnerRequest.cs</td>
<td>-</td>
</tr>
<tr>
<td>DBRequest.cs</td>
<td>Source code of the class that encapsulate the information stored in the cache of the getRequestStatus operation.</td>
</tr>
</tbody>
</table>

### Response directory

<table>
<thead>
<tr>
<th>Response</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response.cs</td>
<td>Source code of the base class of the Response classes. Contains the environment information returned by an operation and methods to deserialize it.</td>
</tr>
<tr>
<td>ContactResponse.cs</td>
<td>Source code of the class that encapsulates the out parameters of the corresponding operations.</td>
</tr>
<tr>
<td>ContactListResponse.cs</td>
<td>-</td>
</tr>
<tr>
<td>HostResponse.cs</td>
<td>-</td>
</tr>
<tr>
<td>DomainResponse.cs</td>
<td>-</td>
</tr>
<tr>
<td>OwnerResponse.cs</td>
<td>-</td>
</tr>
</tbody>
</table>
4.3 **Database module**

To address the various issues mentioned in the pre-study, for example concerning the two step communication between the Web Service, Nominalia and the name registrars, a database is used to ensure a high reliability. The three main purposes of the internal database of the Nominalia Web Service are:

- The main purpose is to save the state of an operation between it has been started and it has finished.
- Second it is used to be able to query the status and result of a request after it has been executed.
- And third it is used to facilitate recovery and the retrying of failed operations.

4.3.1 **Administration**

The database used is the Microsoft SQL Server of Nominalia. It can be administrated from the SQL Server Enterprise Manager, see Figure 4-3. The ASPNET process of the web server must have access rights to the stored procedures.

![Figure 4-3. The SQL Server Enterprise Manager.](image)
4.3.2 Tables

The Web Service database consists of only one table “PETICION”.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>peticion_id</td>
<td>int</td>
</tr>
<tr>
<td></td>
<td>IDENTITY (1, 1)</td>
</tr>
<tr>
<td>is_done</td>
<td>bit</td>
</tr>
<tr>
<td>begin_petición</td>
<td>datetime</td>
</tr>
<tr>
<td>end_petición</td>
<td>datetime</td>
</tr>
<tr>
<td>operation</td>
<td>varchar (50)</td>
</tr>
<tr>
<td>query string</td>
<td>varchar (4000)</td>
</tr>
<tr>
<td>status</td>
<td>varchar (100)</td>
</tr>
<tr>
<td>wholesaler</td>
<td>varchar (50)</td>
</tr>
<tr>
<td>type</td>
<td>int</td>
</tr>
<tr>
<td>retries</td>
<td>int</td>
</tr>
<tr>
<td>login</td>
<td>varchar (50)</td>
</tr>
<tr>
<td>password</td>
<td>varchar (50)</td>
</tr>
</tbody>
</table>

The reasons of having the database structure this denormalized is the short storing time of the data and to avoid synchronization with the Nominalia backend system.

Field descriptions:

**peticion_id:**
This is an unique operation id, given by the SQL Server database.

**is_done:**
This is a Boolean flag that indicates if the operation is done or not.

**begin_petición:**
This is the date and time when the operation has been initialised. The Web Service sets it before sending the request to the backend system.

**end_petición:**
This is the date and time when the operation has finished. The Answer.aspx page updates this after receiving the response from the backend system.

**operation:**
This field is thought to contain the type of operation. Not used in this release.

**querystring:**
This is initially the querystring that is created from the SOAP message received and later sent to the backend system. After getting the response from the backend system it is changed to the response received. It can be used to retry a failed operation or get the result of a successful operation.
status:
This is the status of the operation either returned by the backend system or an internal error message set by the Web Service. Possible values are:
- “OK” – The operation has been successfully executed.
- “ERROR” – Backend error message.
- “Not done” – The operation has been started but has not yet finished, TIMEOUT.
- “Failed” – The operation has failed to contact the backend system. The user has to redo the operation manually.

wholesaler:
This contains the identifier of the wholesaler that has executed the operation.

type:
This is the type of entity that the operation concerns. It is used for the GetRequestStatus to know what kind of response to return. Possible values are:
- 1 – Domain
- 2 – Host
- 3 – Contact

retries:
This is the number of retries that has been made for this operation in case of failure.

login:
This is the login of the client that executes the operation. Together with the password it is used to authenticate the client when he requests the result of the operation with the GetStatusRequest operation.

password:
This is the password of the client that executes the operation. Together with the login it is used to authenticate the client when he requests the result of the operation with the GetStatusRequest operation.

4.3.3 Stored procedures
All access to the database is done via stored procedures. It is done this way to separate the database logic from the application code and also to restrain what the user can do with the database. The only access right the user has, is to execute the stored procedures, so he cannot directly access the table. As the stored procedures are precompiled there is no parsing of SQL and therefore a considerable gain of performance is obtained.

The stored procedures are documented with their name and parameters followed by a
short description and who is using them.

**insertBeginRequest** (begin_peticion, type, wholesaler)
Marks the beginning of a new request. Returns a unique identifier for this request.
Used by all the new and modify Web Service operations.

**updateRequest** (peticion_id, querystring)
Stores the querystring of a started request.
Used by all the new and modify Web Service operations.

**insertEndRequest** (peticion_id, end_peticion, status, querystring)
Updates a started request. Sets the status and the response from the backend system. Used by Answer.aspx.

**getRequestResult** (peticion_id)
Gets the status of a specified request. Used by the GetRequestResult operation.

**checkFailedRequests** (maxRetries)
Sets the status to Failed to all operations that have been retried the specified maximum amount of times and that are not done.
Used by the Windows Service that manages non-successful requests.

**getRetryRequests** (maxRetries, retryInterval)
Gets all requests that:

- Have been started more than the specified (retryInterval) minutes ago
- Have not yet been retried the (maxRetries) maximum amount of times
- Are not done and increases the value of the field “retries”.
Used by the Windows Service that manages non-successful requests.

**getOldRequests**
Get all requests older than one month.
Used by the archiving application.

**cleanOldRequests**
Delete all requests older than one month.
Used by the archiving application together with getOldRequests.

### 4.3.4 Database access privileges

The ASP.NET (Web Server) user, the Cleaning Service user and the Archiving Application user have to have:

- Login and access rights to the database
- Execute rights to its corresponding stored procedures. See dependencies of each module.
4.4 Nominalia Cleaning Service

If a request from the Web Service that is sent to the Nominalia backend system fails or “times out”, the result will be that the Web Service returns the operation id and a “Not done” status message to the client, see Figure 4-4. The client can then use the getRequestStatus operation to later check the status of the operation.

The Cleaning Service is a Windows Service running as an agent in the background; see section 4.4.2 for a short introduction to Windows Services. It is in charge of retrying failed operations and after having reached the maximum amount of retries marking the status of the operation as “Failed”. The database is queried if there exist requests that are not done and older than a configurable amount of minutes. That is, that they have been started and are still not finished. In this case there are two scenarios that can occur:

- If the maximum amount of retries has been reached the Cleaning Service sets the status to “Failed” so that the client will know that the operation has failed.
- If not the Cleaning Service tries to execute the operation one more time.

A graphic representation of the workflow described above is given in Figure 4-6.

![Graphic representation of the workflow described above](image-url)

Figure 4-4. Status workflow of a Web Service request.
4.4.1 Overview

Figure 4-5 shows the architectural overview of the Nominalia Service module. The basic workflow inside the Nominalia Service is illustrated in Figure 4-6.

![Figure 4-5. Architectural overview - Nominalia Service](image1)

4.4.2 Introduction to Windows Services

A service in Microsoft Windows is a program that runs whenever the computer is running the operating system. It does not require a user to be logged on. Typically these services provide system-level support, including the system event log and the task scheduler. They can also be used for user-independent tasks as in the Nominalia Cleaning Service. Specific tasks for an application can be done on a regular basis. As there might not be any user logged on, errors and warnings should be reported to the system or application logs. The services can be controlled through either a separate
application or the Services tool provided by the Windows operating system.

### 4.4.3 Configuration

The Nominalia Cleaning Service application has five parameters that can be set in the application configuration file `NominaliaService.exe.config`, a XML file that can be found in the application root directory. These parameters are shown in Table 4-2.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conn.ConnectionString</td>
<td>The connection string to connect to the database.</td>
</tr>
<tr>
<td>dna_url</td>
<td>The URL of the backend dna.cgi script.</td>
</tr>
<tr>
<td>timer.Interval</td>
<td>The interval in which the service should run.</td>
</tr>
<tr>
<td>MaxRetries</td>
<td>The maximum number of retries of an operation before it is considered failed.</td>
</tr>
<tr>
<td>RetryInterval</td>
<td>The time that the service waits before an operation is retried.</td>
</tr>
</tbody>
</table>

Table 4-2. Configuration parameters

Following is an example listing of the Nominalia Cleaning Service configuration file.

```xml
<?xml version="1.0" encoding="Windows-1252"?>
<configuration>
  <appSettings>
    <!-- This is the interval that the service checks the database for failed operations. -->
    <add key="timer.Interval" value="60000" />

    <!-- The connection string to connect to the SQL Server database. -->
    <add key="conn.ConnectionString" value="Integrated Security=SSPI;Initial Catalog=nominalia;Data Source=(local)" />

    <!-- This is the number of retries for an operation before it is considered failed -->
    <add key="MaxRetries" value="0" />

    <!-- This is the amount of minutes before an operation is retried -->
    <add key="retryInterval" value="5" />

    <!-- The Windows service uses the Comunicator from the NominaliaWS module but as it is run inside this module it needs to have it's own configuration file -->
    <add key="dna_url" value="https://secure.nominalia.com/ws_gateway/ws_gateway.php" />

    <!-- Required by the Comunicator class, but not used here -->
    <add key="answer_url" value="" />
    <add key="asinc_timeout" value="0" />
  </appSettings>
</configuration>
```

Listing: Nominalia Web Service configuration file, NominaliaService.exe.config
4.4.4 Administration

As all Windows services the Nominalia Cleaning Service is administrated from the Services tool found in the Administration tools of the Windows operating system, see Figure 4-7. Here the service can be started, stopped and paused. It can also be specified if the service should start automatically when the operating system starts or if it has to be started manually.

A Windows service cannot be run like a normal application. After building the executable, it must be installed as a running service.

![Figure 4-7. The Windows Service administration applet.](image)

**Install your service**

Run the InstallUtil tool that comes with Visual Studio, using the following command line:

```plaintext
InstallUtil NominaliaService.exe
```

**Uninstall the service**

To remove the service use the Services dialog box to stop the service. Run InstallUtil with its /u parameter to uninstall the service. The Windows Services tool also needs to be shut down to completely remove the service from memory.
4.4.5 Error and logging
As a Windows Service does not have any user interface, error messages and logging are made to the Application Log. This log can be accessed with the Event Viewer of the Administration Tools of the Windows operating system, see Figure 4-8.

![Figure 4-8. The application log of the Event Viewer.](image)

4.4.6 Dependencies
The Cleaning Service uses the database module and specially the stored procedures:
- checkFailedRequests
- getRetryRequests

The Cleaning Service user must have execute rights to the stored procedures above. It also uses the WSNominlia.dll to resend the failed operations.

4.4.7 Files
The Nominalia Cleaning Service consist of the following files:

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NominaliaService.exe</td>
<td>Executable</td>
</tr>
<tr>
<td>NominaliaService.exe.config</td>
<td>XML configuration file</td>
</tr>
<tr>
<td>WSNominlia.dll</td>
<td>The dynamic link library containing the communicator class used to communicate with the dna.cgi.</td>
</tr>
<tr>
<td>NominaliaService.cs</td>
<td>C# source file</td>
</tr>
<tr>
<td>ProjectInstaller.cs</td>
<td>C# source file of helper methods to install the service, Generated by Visual Studio .NET.</td>
</tr>
</tbody>
</table>
4.5 Archiving Application

This is a stand-alone application that deletes requests older than one month from the database and logs them to a log file on the file system to constrain the database size. It can be run manually but it is thought to be run the first of every month by the Scheduled Tasks of the Windows operating system.

4.5.1 Overview

Figure 4-9 shows the architectural overview of the Nominalia database archiving module.

Figure 4-9. Architectural overview - Database archiving module.

4.5.2 Administration

The Archiving Application is installed in the Scheduled Task manager where it can be configured when and in what interval it should run. It can also be specified as what user the service should run. Figure 4-10 shows an example view of the Scheduled Tasks with the Nominalia Archiving Application installed.

Figure 4-10. The archiving module in the Scheduled Tasks of Windows
4.5.3 Dependencies
The archiving application uses the database module and must have access right to the stored procedures:
- `getOldRequests`
- `cleanOldRequests`

4.5.4 Files
The Archiving application consist of the following files:

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NominaliaDBArchive.exe</td>
<td>Executable</td>
</tr>
<tr>
<td>DBArchive.cs</td>
<td>C# source code</td>
</tr>
<tr>
<td>NominaliaDBArchive.exe.config</td>
<td>XML configuration file</td>
</tr>
</tbody>
</table>

The log files are plain text files that contain the values of the fields of the PETICION database table separated with a tabular.

The naming convention of the files is:

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
</table>
5 Conclusions

5.1 Objectives reached
The implementation and documentation of the application have been delivered to the customer. The application has been successfully tested and accepted. It is now deployed live and in production. Consequently the main objective to develop a new Web Service platform for Nominalia has been reached. Following is an evaluation of the work that has been conducted and an evaluation of the additional objectives.

5.1.1 Objectives of the student of the work conducted
Listed below are all the objectives from the student’s point of view. Each objective is followed by an evaluation of its final results.

- Overcome the software design and system architecture problems and challenges of developing a Web Service project.

  During four month the Nominalia Web Service application was designed and developed together with professionals from Spenta and Microsoft. After the application had been finished two weeks of installation, integration and testing was conducted together with Nominalia. Now the application is deployed live and in production.

- Evaluate the different architectures applicable to the actual project and chose the optimal one.

  Prototypes of different architecture alternatives were developed and evaluated before choosing the optimal to finalize the work.

- Explore the maturity of this technology.

  Two month of research were conducted to get an exhaustive comprehension of the Web Service technologies. During the implementation and testing of different clients of the Applications various issues like security and interoperability had to be investigated and solved.

- Research the evolution of the standards on which Web Services are based.

  The standards upon which Web Services are based are rapidly changing. There exist many implementations but there are still many questions that have to be solved, for example security and interoperability. There is an enormous effort being made to make the Web Service standards and protocols accepted by the software industry.
- Obtain a real vision of how Web Services can help a company to improve its processes and reduce its costs.
  - At this point it is too early to evaluate the business success of the Nominalia Web Services.

5.1.2 Objectives of Nominalia concerning the implementation of the Web Service application

Listed below are all the objectives from Nominalia’s point of view. Each objective is followed by an evaluation of its final results.

- Increase the volume of business through new resellers, offering them a stable solution that is easy to use and easy to integrate with their actual systems.
  - As the application is conformant to both of the two most common SOAP formats any reseller, independent of its technical platform, can use the Nominalia Web Service. Although at this point it is too early to evaluate the business success of the Nominalia Web Services.

- Reduce the costs associated with reseller support, permitting Nominalia to offer more competitive prices.
  - The platform is well documented with examples of all operations. And the processes are totally synchronous without any need of human support.

- Avoid the development costs of tailor made solutions to its resellers.
  - As the Web Service platform is very easy to use and SOAP toolkits exists for almost all technical platforms, the resellers can easily develop their own client applications.

- Build a new solid technology base for a future migration of the backend systems of Nominalia to an open and scaleable platform based on the Microsoft .NET technology.
  - The technical staff of Nominalia has all been trained in the Web Services technologies and the Microsoft .NET platform and has successfully integrated the new technology into their organization.

- Media impact and technology positioning.
  - The marketing department of Nominalia is making a big effort in promoting and selling the new Web Service platform, both to its existing net of customers and in various technical magazines.
5.1.3 General conclusions

When working on a big project there exists a need of discipline and planning that does not exist in a smaller project, which always can be solved in the end, making one big effort.

The decision to conduct this project together with a company has not only had the positive effects desired, it has also implied many other unexpected both positive and negative effects. Having your work depending on other people forces you not only to do your own work good, but also to communicate and collaborate with other people to make the work as effective as possible.

To avoid project delays and specification changes always procure that the requirements are agreed on before starting the project. To avoid financial drawbacks it is better to be paid by the hour than project based and always have a big margin in the time and cost planning.

5.2 Planning and costs

5.2.1 Initial planning

Initial reading and studying the Web Services technologies, not being paid by the customer (Nominalia): 40 hours.

<table>
<thead>
<tr>
<th>Phase – Task</th>
<th>Hours</th>
<th>Persons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-study</td>
<td>160</td>
<td>3</td>
</tr>
<tr>
<td>Analyst – 80 (other professionals of Spenta)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Technical Architect – 40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Microsoft Consultant – 16 (Microsoft)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requirements</td>
<td>320</td>
<td>2</td>
</tr>
<tr>
<td>Design</td>
<td>160</td>
<td>1</td>
</tr>
<tr>
<td>Implementation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Web Services Framework</td>
<td>160</td>
<td>1</td>
</tr>
<tr>
<td>- Operations</td>
<td>160</td>
<td>1</td>
</tr>
<tr>
<td>Test</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>Documentation</td>
<td>80</td>
<td>1</td>
</tr>
<tr>
<td>- User manual</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Administration manual</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deployment</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>- Test environment</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Production environment</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The technical part of the pre-study, appendix A, was entirely done by the student. The business part that is not included in this report was done by two other consultants.
from Spenta Consultants. A two days workshop together with the Microsoft consultant was held in the middle of the pre-study.

In total the implementation project was planned to last 100 working days, which correspond to 5 months of work.

5.2.2 Costs

Implementation

The cost of the project was calculated with an average price of 40 Euros an hour. The pricing model for the final customer was project based and set to a fixed price based on the initial calculations. The Microsoft consultant was free of charge due to a convention between Spenta Consulting and Microsoft.

Hardware and software

The hardware cost of the project is included in already existing costs of Nominalia as existing servers and Internet hosting are used. The licensed software used are also shared with actual applications, for example the SQL Server 2000 database. The .NET Framework is free of charge.

5.2.3 Comparison of initial and final planning

The 5 months initially planned to execute this project have been exceeded with 2 months. Below follow some possible reasons why this occurred.

- Lack of priority from the technical department of Nominalia, due to heavy workload on other projects and the maintenance of the existing systems.
- Functional requirements not complete before starting the implementing phase.
- Lack of documentation, real data and access to the backend systems of the client.
- Lack of continuity of the work, due to long response times from the client.

As the price was fixed there are no differences between the planned and real cost of the project for Nominalia. But the margin of the benefits for Spenta Consulting has been severely reduced, although some of the changes made to the requirements have resulted in penalties and increased budget.
5.3 Evaluation

To test that the application works, a test suite of operations has been created with real data provided by Nominalia. This test suite has been used to get the final approval from the client that the application is done. The test suite consisted of the following operations with different values both correct and incorrect. The complete test suite was programmed in C# in Visual Studio .NET. There was also programmed a limited version in PHP. Parts of the test suite are used as examples in the user manual (see Appendix B). Below is a list of the methods tested in the test suite.

- Create a new domain
  - With existing contacts and owner
  - Creating a new owner in the same operation
  - Creating a new contact in the same operation
- Modify an existing domain
- Get info about an existing domain

- Create a new host
- Modify an existing host
- Get info about an existing host

- Create a new contact
- Modify an existing contact
- Get info about an existing contact
- List all contacts with a specific name

- Get request status
  - With the correct user and password
  - With the incorrect user and password

5.4 Future

Negotiations are held for phase two of the Web Service application that will comprise the following improvements:

- More operations
- Refined security management
- Pre-made client APIs
5.4.1 More operations

These two action paths are thought to lead to a progressive migration of all the business logic now encapsulated in the CGI and PHP scripts to a more stable and scalable based on Web Services and .NET, see Figure A-4 in Appendix A.

- Management and registering of new domains:
  - Incorporation of the management and registering of the remaining synchronous domains, and also the functionalities that involves the management and registering of asynchronous domains.
    - .name
    - .info
    - .biz

- New functionalities and services:
  - Design, development and incorporation of new functionalities to be published as Web Services.
    - Statistics
    - Listings

5.4.2 Refined security management

In this first version the security is totally based on the transportation layer using SSL over HTTP. This solution is the simplest to integrate and only requires that the SOAP toolkit used supports SSL.

In the following version the use of SOAP Headers conformant with the Web Service Security (WS-Security) [11] specification will be studied. WS-Security describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. It proposes a standard set of SOAP extensions that can be used when building secure Web Services to implement integrity and confidentiality.

This solution implies a lot more work both on the server and the client side. In short this means that only the body of the SOAP message will be encrypted and the encryption details will be sent in the SOAP Headers. This implies that both the server and the client must be able to encrypt and decrypt parts of the messages. In addition there will be the management of public and private keys and certificates.

5.4.3 Pre-made client APIs

To facilitate the development of clients of the Nominalia Web Services, pre-made client APIs could be developed for all the operations offered by the Web Service ready to be used. APIs should be developed for the most common programming languages: C#, Java, PHP, Perl and C++. 
Bibliography

References

[1] Extensible Markup Language (XML)
   http://www.w3c.org/XML
   The World Wide Web Consortium’s web site containing the formal technical specifications of XML.

   http://msdn.microsoft.com
   The official developer site from Microsoft.

   http://java.sun.com/j2ee
   The official site of the J2EE platform.

   http://www.ws-i.org

   The official site of the Apache SOAP implementation.

   The World Wide Web Consortium’s web site containing the technical specifications of SOAP 1.1.

   The World Wide Web Consortium’s web site containing the technical specifications of WSDL.

   The World Wide Web Consortium’s web site containing the technical specifications of XML Schemas.

   Universal Description, Discovery and Integration Specification

    The UML helps you specify, visualize, and document models of software systems, including their structure and design.

    An article about the WS-Security specification.
Web Services technologies

HTTP – Hypertext Transfer Protocol:
http://www.w3.org/Protocols/

SOAP – Simple Object Access Protocol (W3C Note 08 May 2000):
http://www.w3.org/TR/SOAP/

SOAP Toolkits
http://www.soapware.org/directory/4/implementations

UDDI - Universal Description, Discovery, and Integration:
http://www.uddi.org/

WSDL – Web Services Description Language (W3C Note 15 March 2001):
http://www.w3.org/TR/wSDL

W3C WS – Web Services Activity:
http://www.w3.org/2002/ws

XML – Extensible Markup Language XML (W3C):
http://www.w3.org/XML/

XML Schema
http://www.w3.org/XML/Schema

Software development

PHP
http://www.nusoap.net

Web Services
http://www.webservices.org
http://www.w3.org/2002/ws

Books

Appendices

Appendix A

A Project pre-study

Actual situation of Nominalia

The implementation of a new domain name registering system based on Web Services presents various impact areas on different business processes. The new platform will have to be integrated with the already existing architecture of Nominalia. Therefore a thorough research of the actual situation and organization of Nominalia has been conducted. The project has been divided into various phases differentiated in time, aiming to avoid interactions with the day-to-day work of Nominalia. The current situation of Nominalia is a not entirely stable architecture with proprietary applications that are not easily integrated. There is a need for some work towards a future robust platform that is scalable and easy to integrate with other systems. This can be done with the adaptation of standard protocols and Web Services.

Technical Architecture

All the business logic of Nominalia concerning the registering of domain names are exposed via a Common Gateway Interface (CGI) script, dna.cgi. This script is run on a Silicon Graphics machine with the operation system IRIX. The functionalities concerning the management of domain names, for example updates, listings and statistics are implemented with various PHP scripts and in some cases by the previously mentioned CGI script. The resellers have the following two options to manage their domain names at Nominalia:

- The Nominalia extranet
- Direct communication with the CGI script via HTTP requests.

The direct HTTP requests must be made with predefined parameters supplied by Nominalia. The CGI script supports many operations with dozens of parameters. Its maintenance is very expensive and changes and incorporation of new functionalities are complicated. The CGI script communicates via different proprietary protocols with each of the domain name registrars, for example Verisign. The registration of newer top domains, for example info, biz and aero, is done via a domain name industry standard protocol called EPP (Extensible Provisioning Protocol). See Figure A-1 for an architectural schema of the actual systems of Nominalia.
Actual architecture

![Diagram of Nominalia's actual architecture]

Figure A-1. The actual architecture of Nominalia

Particularities and differences between different domains

On an operational level, the two biggest differences between the different types of domains that are managed by Nominalia are:

- **Type of process - Synchronous or Asynchronous**
- **The communication with the registrars**

In the first category, process type, we define as synchronous registers those whose management is handled automatically, and therefore completely on-line, without the need of manual processes or waiting periods to execute validations or verifications off-line. In the Nominalia case, the following top-level domains belong to this category:

- .com, .org, .net
- .name, .info, .biz

The registering of any of these top-level domains can be done automatically from the interfaces that Nominalia provides its users.

The asynchronous registers are those that are not immediately processed and
approved by the registry server, but require manual processes or/and waiting periods in their processing. In Nominalia, the following top-level domains belong to this category:

- The country code top-level domains belonging to any country in the world, except .tv and some other special cases.
- .museum, .aero, .coop

The registering process of any of these domains contains both automated and manual parts. Examples of manual parts are physical deliveries of documents, translations, etc.

In the communications between Nominalia and the different registrars there exist a variety of levels of digitalisation and communication protocols depending on each registrar. For example, registrars like ESNIC (the registrar of .es) has a very low level of adaptation of automatic digital processes and has the biggest part of its processing off-line. On the other hand the communications with some of the other registrars are totally automatic, digital and standardized, using protocols like RPP (Registry Registrar Protocol) for .com, .org or .net and EPP for the newer domains mentioned earlier.

When designing the new platform careful consideration has to be taken to the different nature of the domains concerning process types and communication protocols.

**Technical architecture**

The Nominalia system interacts both with the internal Informix database and with the different backend systems of the Name Registers, for example Verisign. The communication between Nominalia and the Name Registers is, for the moment, done over proprietary protocols specified by the different Name Registers.

All operations concerning adding or changing data of a certain domain or associated entities like owner, contacts or hosts are done via the CGI and PHP scripts of Nominalia and then sent further to the Name Registrars. This two-step process creates a potential problem with delays and service failures that has to be taken into account when designing the Web Services using these services.

The information retrieval operations only query the internal database of Nominalia.

**Design considerations**

When a client calls the Web Service, the Web Service transforms the input of the operation to the format of the Nominalia system. It then calls the corresponding script in the Nominalia system or in the case of a query operation the database. If the operation needs to call the Name Registers the CGI script can return the result value in two different manners. The default is returning a HTML page showing the results. The alternative manner is a type of callback functionality where the CGI script returns the result to an answer URL that is supplied as a parameter.
Evaluation of the two result retrieving alternatives

Alternative 1:
Parsing the HTML that is returned and returning the information to the client. This alternative is discarded because of the strong coupling with the format of the HTML. We do not want to be dependent on the format and design of the HTML returned.

Alternative 2:
Supplying an answer URL to the Nominalia system and having an ASP-page receiving the results returned. However this creates two new problems. First, how to get the results from the receiving ASP-page to the Web Service so it can return it to the client, as they run in different threads and possibly different processes. The other problem is if the Nominalia script returns before it posts the results to the Answer page. Then the Web Service has to wait until the results have arrived before being able to continue and return the results to the client.

As the second alternative is considered to be less coupled and easier to maintain, it is chosen to be continued working on.

To solve the problem of passing the returned results between the answer ASP-page and the Web Service, one approach could be storing it in the shared Application object of the Web Server (IIS). If the ASP-page and the Web Services are running in the same IIS instance then they will both have access to the Application object. In this solution the Web Service sends a unique identifier with the request to the CGI script. The identifier is sent via the environment variable HTTP_REFERER that is forwarded unaltered by the CGI script to the answer ASP-page. The ASP-page then puts the results in the Application object with this unique identifier as the key and subsequently the Web Service can retrieve the results and return them to the client.

The solution using the Application object that is stored in memory has a major drawback of not being failsafe. If the server goes down, the Application object and all of its information are lost and the results can never be returned to the client.

Therefore there exists a need for a more permanent storage that is accessible from both the ASP-page and the Web Service. The natural choice is a database. The Web Service stores an unique operation identifier and additional data in the database and sends this identifier to the CGI script. When the ASP-page gets the posted results it updates the database with the results of the operation and some additional information. This solution is more failsafe and gives more flexibility to address another problem, namely the problem of latency and lost connections between Nominalia and the Name Registrars. By having the request stored in the database, operations can be re-invoked to the CGI script if they have failed.
Implementation

Problems with the CGI script and the initially proposed solution:

The way that the CGI works implies some difficulties in integrating it with the Web Service platform. Here follows an explication of this problem and the solutions proposed.

As mentioned in the previous section, to avoid having to parse the HTML returned by the CGI and to be dependent on the design of the HTML, the ANSWER_URL parameter of the CGI script is used and the returned HTML is ignored. However with this situation two new problems occur:

- How does the Web Services get the results from the ANSWER_URL to be able to return them to the client?
- How will the Web Service know when the CGI has returned the results to the ANSWER_URL?

Initially a ‘Semaphore’ and the ‘Application’ object, that is shared by all the applications of the Web Server, were used to solve this problem:

1. The Web Service waits at the ‘Semaphore’ after the CGI to process the request.
2. The CGI sends the results to the ANSWER_URL.
3. The ANSWER_URL puts the results in the ‘Application’ object and notifies the ‘semaphore’.
4. When being notified, the ‘Semaphore’ wakes up the Web Service that gets the results from the ‘Application’ object and returns them to the client.

As mentioned earlier the Semaphore and Application object solution is not failsafe so some other alternatives where investigated to solve these problems. Using a database as storage instead of the Application object makes the solution much more failsafe. Instead of an explicit semaphore we use an asynchronous call to the CGI script from the Web Service.

To solve the time-out problems with latency and lost connections, various communications alternatives were considered. Both between the client and the Web Service and also between the Web Service and the Nominalia backend systems as illustrated in Figure A-2.

![Figure A-2. Client-Server-Backend communication.](image-url)
Client server alternatives

1. Client Synchronous – Server Synchronous

This is the simplest solution considered. The client calls the Web Service synchronously and the Web Service calls the Nominalia system synchronously and when it returns the values the Web Service returns the results to the client.

Advantages:
- This is the simplest alternative and also the easiest to implement.
- It is conformant to the way the Nominalia system now works for a user accessing it with a normal browser.

Disadvantages:
- While the client is waiting for the results to be returned; one thread of the IIS (Web Server) is occupied waiting. This affects the performance and limits the amount of concurrent users of the Web Services.
- If there is a lost connection or a timeout between Nominalia and a Name Registry the connection between the client and the Web Service is also lost.
- Even if the connection is not lost the client might have to wait a relatively long time for the results.

2. Client Asynchronous – Server Synchronous

One way to avoid having the client waiting and to free the connection of the IIS, is that the Web Service returns an identifier of the operation as the result of the operation. The client then calls a second operation that gets the result of the specified operation (Polling). The state of the requests will be stored in a database and in this way the problem with the ‘no-recovery’ of time-outs is also solved. This new polling operation checks the database to see if the results have been returned. If returned they are consequently returned to the client and if not, the client is asked to try again later. To prevent that the client invokes a large amount of database calls that could lower the performance of the server, a cache can be implemented on the server side. The cache is invalidated when the results have arrived and in this way maximum two database calls are made per operation. On the client side a kind of smart polling can be made, for example a routine on the answer page that refreshes in increasing intervals and that only calls the polling operation a certain amount of times.

Advantages:
- Improves the concurrency performance avoiding that the Web Server holds up threads.
- Solves the ‘no-recovery’ problem.

Disadvantages:
- Client side development of polling mechanism necessary.
3. Client either synchronous or asynchronous – Server asynchronous
   This is a hybrid solution where the Web Service creates a callback object for an
   asynchronous communication with the Nominalia back-end system. The Web Service
   waits for the callback object to get the response for a certain time, TIMEOUT. If the
   back-end system does not respond within this time the Web Service returns an
   identifier of the request to the client. In this case the behaviour is very similar to the
   second alternative, which in fact corresponds to the special case having the waiting
   time set to 0. If the results are returned by the back-end system before the TIMEOUT,
   the Web Service gets the results from the callback object and returns them to the
   client. In this case the behavior is identical to the first alternative. This third
   alternative can be seen as synchronous with a TIMEOUT. If the TIMEOUT occurs it
   acts in an asynchronous manner and if not, synchronously. The TIMEOUT parameter
   is defined in a configuration file.

Advantages:
- We avoid the need of client polling in some cases.
- We have absolute control of the behavior of the system (synchronous or
  asynchronous).
- We solve the ‘no-recovery’ problem.

Disadvantages:
- The threads of the IIS are not freed, but we can define arbitrary small timeouts
  and then each connection will be relatively short.

4. Client – Server Asynchronous Web Service
   The last option is to let the IIS implicitly create a callback object that returns the
   result to the user but liberates the connection so the concurrency performance is not
   affected. However the user will not notice any difference and the problem with the
   latency or lost connections from the client’s perspective are not solved. In this option
   explicit delegates used in asynchronous calls from the Web Service to the Nominalia
   system cannot be used. If they would, they would occupy a thread and it would be the
   same effect as without the asynchronous Web Service implementation.

Result
   Test implementations of these alternatives have been made to test the performance
   with different loads and concurrent users accessing the Web Service. The results of
   these tests have resulted in the choosing of the second alternative.

Client either synchronous or asynchronous – Server asynchronous
   Following in Figure A-3 a sequential diagram describes the suggested solution.
   and Figure A-4 a class diagram illustrates the initially suggested architecture.
Figure A-4. Initial class diagram
Test environment
Nominalia has in this moment two different environments, test and production.

- Test environment
- Production environment

There will be only one Web Service system implemented that will act on the two different environments of Nominalia depending on the parameters received. It is recommended to design and implement a test system with some kind of test or exam where clients can test their applications. The testing system would check if the requests done by the resellers’ applications are done according to the specifications and if the parameters send to the Web Service are the ones expected. Only after the reseller has passed this testing platform he will get access to the production environment. The process of testing the resellers’ applications to get access to the production environment should be totally automated. Hence, the support needed by the resellers would be minimized and additionally erroneous calls to the production system would be avoided.

Access to the Web Services system
Initially the resellers that decide to use the Web Services of Nominalia will only have one interface: the SOAP protocol. The publication of the communication protocol of the Nominalia Web Services is orientated to resellers with a certain technical level that are able to realize some client development. Depending on the response of the resellers and future resellers on the new platform and its Web Service interface, Nominalia will consider developing clients and application programming interfaces (API) in various programming languages.

Security
The SOAP protocol defines a format to interchange XML documents between applications in a distributed environment. The simplicity of this protocol has lead to a rapid adaptation from the software industry, but has also obliged the developers to implement some complex concepts themselves that are not included in the SOAP specification, for example security. However most platforms, like .NET, give the developers the necessary tools to add security mechanisms to their Web Services.

The most common way is to use the security mechanisms of the transportation layer, most commonly HTTP:

- Basic Authentication
- Windows Authentication
- Secure Socket Layer (SSL), etc.

Another option is to manage the security in the application layer. With XML encryption it is possible to include the security in the SOAP messages or in custom SOAP Headers and only encrypt parts of the message. SSL encrypts the whole message including the headers and other metadata so that for example routing of messages does not work. This second approach requires more from developers and
clients.

In the last year, Microsoft, IBM and Verisign have published various additional specifications to the SOAP protocol. These specifications treat some of the complicated concepts of distributed applications that are not included in the main specification: security, transactions, routing, etc. Recently Microsoft has released a Web Services Development Kit (WSDK) that contains a set of classes that supports for example the WS-Security specification to develop secure Web Services. Because of the diversity of alternatives to create secure Web Service and the early face of the new specifications some time was spent studying the different options now available.

Mainly the following two paths were considered:

- Basic authentication – The user and password are sent in SOAP Headers following the WS-Security specification over SSL.
- Public Key Identification - Especially interesting is XML Key Management Service specification (XKMS) and XML digital signature (XMLDSig) that could open a new line of business for Nominalia.

A third option also considered is a product recently released by Nominalia, called ‘Digital Identity’, that consists of an authentication system of users. Once and if this product has any success on the market this should be studied as a security solution for the Web Services.

After an analysis made together with Microsoft of the different alternatives considered, it was decided to implement the security in this first phase using the transport layer, SSL over HTTP. Which is currently used by Nominalia.

**Hardware and software**

**Machine dimension**
To host the Nominalia Web Service applications the minimum need is a server with a Pentium III 800Mhz with at least 256 MB RAM.

**Hosting**
To obtain an ideal configuration and optimal performance it is desirable that the Web Service server is in the same internal red as the other systems of Nominalia. In this way we avoid having to make HTTPS requests internally and improve the response time to the clients.

**Software**
The software installed on the production server should be:

- Web Server - IIS (Internet Information Server) 6.0
- .NET Framework v 1.0
**Choice of platform**

On the market there exist various platforms to facilitate the development and hosting of Web Services. All major software companies that offer products in relation with application servers, for example Microsoft, IBM, Sun, HP, Borland and IONA, also offer a platform for Web Services. Before taking the decision which technology to use a study of the two biggest alternatives now on the market was done:

- Java J2EE
- Microsoft .NET

It is important to remark that J2EE is a specification and not a product. There are many vendors that offer implementations of this specification, for example the IBM WebSphere and Sun ONE.

Both J2EE and Microsoft .NET base their Web Service offerings on existing technologies of distributed applications. Both share the idea that much work behind Web Services should be facilitated by the platform to save the programmer a lot of work. For example the generation and interpretation of XML (Extensive Markup Language) messages, load balancing and transaction management. Both technologies aim at allowing the programmer focus on the application logic and having the platform do all the specific work concerning the Web Service technologies.

**Proposed technology**

When constructing an information platform, in general, it is preferred to chose a solution coming from one single provider, obtaining a more stable environment that is robust, interoperable and with less integration errors between modules.

The Microsoft .NET platform contributes a complete solution offered by a single proved provider. The tools included in the .NET framework offer applications that guarantee a rapid development of Web Services, automatically generating them from now existing systems.

Arguments in favour of Microsoft .NET:

- Microsoft started to work with Web Services before all other providers, and therefore now has a certain benefit.
- .NET benefits of being absolutely integrated with the operating system.
- One single provider of operating system, application server, database and development tools.
- Minor cost of licenses.
- Collaboration and support from Microsoft.

Based on the motives given above, with emphasis on the fact that we benefit from a near support and collaboration with Microsoft Spain, the project will be development with the Microsoft .NET technologies.
Study of the competitors of Nominalia

Before deciding how to implement the new platform, an extensive study of solutions of some of Nominalia’s competitors has been made. In detail the following alternatives were studied.

The OpenSRS platform from TuCows

TuCows is the world's largest wholesale supplier of domain name registrations. They offers domain names, digital certificates and email via the OpenSRS service delivery platform. OpenSRS is a registration and management platform for domain names, security products and e-mail. It offers several ways of integration:

- QuickStart - A web-based registry interface that can be used immediately.
- Full Access - An API with which an automated registration and management system can be built. A client library is provided to facilitating the creation of own interfaces in the programming language of choice.

Extensible Provisioning Protocol (EPP)

To enable Internet registrars that sell online identity services to access central domain name registry data more efficiently, VeriSign has developed EPP to support an XML-based domain name management utility. EPP enables accredited registrar partners to sell domain names, telephone numbers, and other identity assets. EPP is a connection oriented, application layer client-server protocol for management of objects stored in a shared central repository. The protocol defines generic object management operations and an extensible framework that maps protocol operations to objects. It is based on the schema notation of XML.

Conclusions

A comparison of the architectures and functionalities of the OpenSRS, EPP and Nominalia has been conducted. The full result of this study is a report given to Nominalia and it is not included in this document. Here follows the most important conclusions from that report.

The three platforms all offer the basic functionalities concerning domains, hosts and contacts. In addition the OpenSRS platform and Nominalia offer some additional services like Web Certificates and Web Forwarding. A difference can be seen in the execution of actions. The OpenSRS platform and Nominalia are based on commands while the EPP protocol is objects oriented. There also exists a certain difference how the security is managed by the three platforms. Nominalia and EPP base all their security on SSL over HTTP while OpenSRS use asynchronous encryption on the application layer.

Here follows a short resume of each one of the three platforms studied:

Extensible Provisioning Protocol (EPP):
The EPP protocol provides seven basic operations:
“hello/greeting, login, logout, poll, commands, responses, framework extension”. In the context of any of the three base classes of EPP: domain, host or contact the following commands can be executed:
    “check, info, create, update, delete, transfer, renew”.
The security of EPP entirely rests on the use of SSL on the transport layer.

OpenSRS:
The API of TuCows uses a series of commands to manage domains and name servers. It also permits the management of users, and an additional service to contract Web Certificates. The security of TuCows is message based and is realized through asynchronous encryption with public and private keys on both the server and the client side. It is used for the authentication of the client and for the encryption of the messages. TuCows also offers some of their data to be consulted via their extranet.

Nominalia:
The resellers of Nominalia can be divided in function of the modality of the access to their services:
    § Advanced - who realize calls directly to the dna.cgi script and in that way integrates the Nominalia services into their systems.
    § Standard - who use the extranet and its PHP scripts to access the services of Nominalia.
Nominalia provides similar functionality as the OpenSRS platform and EPP. In addition they also offer access to a lot more information to be consulted through their extranet. In terms of security, Nominalia base their security on the SSL protocol over HTTP.

**Scope of the project**
The scope of the project has been limited after an analysis of the different challenges that Nominalia are facing:
    § the business and technological objectives
    § the competition
    § the market and its actual circumstances

**Application scope**
Based on the business goals and the actual architecture of Nominalia initially two alternatives were proposed to reach the goals required. The main difference between the two alternatives is the dimension and scope of the project in a short term, although the long-term goal is the same for both alternatives.

**Alternative 1**
The first platform alternative consists of the Web Service acting as an interface to the current entry point of the actual backend system of Nominalia. In this way the users can access the Nominalia services both via Web Services standards and the old proprietary protocols. When the Web Service platform receives a petition from a
client, the petition is passed on to the CGI script via the web server through a HTTP POST request. The Web Services will receive the client petitions via the SOAP protocol, and will translate these to the format accepted by the CGI script. This solution allows Nominalia to acquire a Web Service platform without changing its current business logic. The Web Service will be a new communication interface, maintaining all the actual business logic unchanged.

Alternative 2
The second alternative is a totally new platform based on Web Services, whose function would not be limited to act as an interface to the actual systems of Nominalia. It would replace a part of the old business logic now contained in the CGI script and various PHP programs. It would communicate directly with the Nominalia database and with the different registrars through the EPP protocol. This alternative also considers the redesign and optimisation of the business logic of Nominalia, making it more effective and easier to maintain. In this manner, initially Nominalia would maintain two parallel systems depending on the access of their clients, and in the future migrate all users to one platform based on Web Services. This solution implies a much bigger investment of both time and resources and a project that would be considerably more complex than the first alternative.

Proposed alternative
After considering the two alternatives earlier exposed, Nominalia, Spenta and Microsoft, have decided that in order to minimize the initial technical impact the best option is the first alternative. Building a Web Service platform that acts as an interface on top of the current systems. This option permits a faster development and implementation of the Web Services platform. It will also allow a step-by-step transmission of the business logic to the .NET platform and Web Services with the experience of the first part of the system already in operation.

As shown in Figure A-4, the final goal of the project is to totally eliminate the CGI script and all the proprietary internal protocols at all levels. It is previewed to implement all the business logic in the .NET platform and expose all functionalities as Web Services. Some of these methods will be public and others will be private and for internal use only.

Initially there will be a Web Service layer situated between the client and the CGI script. The resellers will make SOAP requests to the Web Service, that will transform and forward these to the CGI script via its proprietary protocol and return its result. The final goal is to, step by step, migrate all the logic now encapsulated in the CGI and PERL scripts to the Web Service. The final result being the CGI totally replaced by the Web Service platform.
Figure A-4. Proposed architecture
Functions to be developed

It has been decided together with Nominalia, that in the first development phase only the functionalities that imply a direct mapping of the parameters and a call to the CGI script will be developed. These functionalities are the ones associated with the following entities and objects:

- Domain
- Host
- Contact

The first implementation phase of the Web Services platform will also be restricted to the management of domains that have a synchronous registering process and information management. These domains are:

- .com
- .org
- .net.

Future suggestions

Once successfully implemented the Web Services for the domains mentioned earlier the expansion of functionalities are previewed orientated in two directions:

- Management and registering of new domains:
  
  Incorporation of the management and registering of the remaining synchronous domains, and also the functionalities that involves the management and registering of asynchronous domains.

- New functionalities and services:
  
  Design, development and incorporation of new functionalities to be published as Web Services.

These two action paths are thought to lead to a progressive migration of all the business logic now encapsulated in the CGI and PHP scripts to a more stable and scalable platform based on .NET and Web Services.

Analysis of the impact

The impact of the construction of a new Web Service platform at Nominalia has been analyzed in three perspectives.

Technical

- The adaptation of a new technology can have an impact on:
  
  - Maintenance: In the case of Web Services the maintenance is easy and minimal. The Web Service is a web application that is run upon already existing infrastructure in which the professionals of Nominalia already have experience in administrating and maintaining.
Coexistence of the two technologies (dna.cgi / Web Service): In the first phase of the proposed implementation the two technologies will be coexisting. Moreover the Web Service will be totally dependent on the backend systems.

- The incorporation of the Web Service platform will bring with it an improvement of the robustness of the systems and functionalities of Nominalia.
- Improvement of the possibilities of integration with other systems, both internal and external.
- Minimization of the costs of new developments for clients.

Organisation

- Internal education/formation: The Web Service technology is standardized and follows traditional architectural and programming concepts and should therefore be easy to learn and adapt. Specially as the technical department of Nominalia have a good background on classical Web technologies.
- Personal motivation: Giving the technical staff of Nominalia the opportunity to get to know and mastering one of the latest tendencies in a new technology should be highly motivating.

Market

- The possibility to augment the service offerings of Nominalia.
- The possibility to operate on the international market with minimum costs.
- Improve the service to clients and resellers by facilitating the integration with their platforms.
- A clear positioning as a leader in technology in the domain registering business by Nominalia.

Risks

There are two risks that should be considered when implementing a Web Service platform.

- Delay in the market of the adaptation of the new technology, given by the recession of the domain name registering sector.
- The Web Service standards might not evolve in a unified manner and conforming to the path chosen by Nominalia.
Conclusions

With the implementation of a Web Service platform and the adaptation of its standard protocols Nominalia will set the base for a solid future growth that will permit a possible expansion to other markets with a certified technology. The proposed architecture of this platform is the result of a rigorous work conducted by the professionals of Nominalia, Spenta Consulting and consultants of Microsoft.

After analysing, evaluating and testing various design alternatives of the platform to be implemented, a solution that is a compromise between technology (robustness, solidness, liability, scalability), implementation and exploitation costs and a rapid time-to-market has been chosen. The first development phase should be considered as the foundation on which future applications will be built. Without any doubt the new platform will facilitate the integration of the Nominalia systems with the ones of their partners. Giving in this way a competitive advantage and an important reduction of costs. This first investment of Nominalia in the Web Service technologies will position the company as a leader in innovation and the adaptation of new technologies. This positioning will give Nominalia an unbeatable situation to profit from the business opportunities that will surge around the Web Service technologies.
Appendix B

B User manual

Introduction
This user manual is an addition to the documentation the .NET Framework automatically generates for a Web Service that is accessible from the main page of the Web Service (see Figure B-1). It contains a list of available operations and an example of a request and a response message for each operation. It also contains the WSDL document that formally describes the Web Service (see Figure B-2).

The Nominalia Web Service can be found under the following end points.

Document/Literal message formats:
https://ws.nominalia.com/ws/NominaliaWS.asmx

RPC/Encoded message formats:
https://ws.nominalia.com/ws/rpc/NominaliaWS.asmx

Figure B-1. Nominalia Web Service main page
The Web Service Description Language Document describing the operations can be found at (Document/Literal):

https://ws.nominalia.com/ws/NominaliaWS.asmx?WSDL

In this user manual, all operations and their parameters, the base elements, are described.

Base elements

Each element is described in detail with descriptions of valid and default values of its attributes. There is also specified which fields that are mandatory. Each element is described with a XML Schema. There are also SOAP examples of each business entity as a part of a message. The following elements are documented.

- The base request and response objects.
- All business entities
  - Host
  - Domain
  - Contacts and Owner
- All operation specific Response and Request objects.
Operations
All public operations accessible to the client are each documented with the following structure:

Contract
The contract of an operation describes the changes of the state of the system after an operation and what information the operation returns.

Attributes
Description of the in and out parameters of the operation. Valid values and default values, if omitted and if not mandatory.

Examples
An example of a SOAP message of a request and a response of the operation.

The examples of the operation messages are all in the Document/Literal format. For examples of RPC/Encoded message formats, refer to the online documentation: https://ws.nominalia.com/ws/rpc/NominaliaWS.asmx

At the end of the document there are code examples made in C# in Visual Studio .NET and PHP using the NuSOAP toolkit.

General message structure
In the majority of the operations the messages send and received are made up by two parts. The first part is the Request object for the message being sent and the Response objects for the message being returned. These two objects contain the environment-, authentication- and status information concerning the operation. The second part is structurally a part of the specific Request or Response object and is a corresponding entity of the specific operation. It can be a Domain, Host or Contact object depending on the operation. It specifies an existing entity or data of a new entity.

For example, in the newDomain operation a DomainRequest object is sent as a parameter and a DomainResponse is returned. The DomainRequest class inherits from the Request class and therefore contains all the environment variables in addition to the actual Domain object that contains the data for the new domain to be created. Correspondingly the DomainResponse inherits from the Response object and also has a Domain object that contains the data of the newly created Domain.
Workflow for building a message
There are many different ways to discover how to construct a message to invoke an operation.

- The first and easiest is to use a SOAP toolkit and specify the URL of the Web Service Description Language (WSDL) document generated for the Web Service. Then the toolkit will generate an operation ready to be invoked as any local operation.
- The second is to use this manual and adapt the desired example message given with the help of the documentation of the different elements and their attributes.
- The third and most complex way is to use the WSDL document manually. This is described in details in the following section.

Using the Web Service Description Language Document
Here is an example how to discover the structure of the message that is required to invoke the newDomain operation using SOAP.

- In the WSDL document find the operation with the name newDomain under the portType NominaliaWSSoap. Here the operations input and output messages are specified together with a short description of the operation. The input message specified is in this case is the newDomainSoapIn.
- In the specification of this message, the parts that it is comprised of are given. In this case there is only one part that is a parameter named newDomain.
- This parameter can be found under types, it is a complex type that only contains one element, a DomainRequest. The DomainRequest type specifies that it inherits from the Request type and that it contains a Domain element. Investigating the Request and the Domain types we will know exactly what attributes these types have. And finally we have all the information we need to send the message to invoke the newDomain operation.

This is the same procedure that a SOAP toolkit would do to create the client proxy, which is normally the only part that the developer sees.

To know what the attributes of these final elements represents and what values the can have, find the corresponding element in the Base attributes section below. For each operation there is a SOAP example given.
Base attributes

Request

This class encapsulates the general environment variables for an operation. It is the base class of all entity specific requests (i.e. the DomainRequest) that are used as parameters to all the Web Service operations. All entity specific requests inherit from this class and therefore also contain all its attributes.

XML Schema Definition

```xml
<s:complexType name="Request">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="env_version" type="s:string" />
    <s:element minOccurs="0" maxOccurs="1" name="env_wholesaler" type="s:string" />
    <s:element minOccurs="0" maxOccurs="1" name="env_login" type="s:string" />
    <s:element minOccurs="0" maxOccurs="1" name="env_password" type="s:string" />
    <s:element minOccurs="0" maxOccurs="1" name="env_language" type="s:string" />
  </s:sequence>
</s:complexType>
```

SOAP example

```xml
<request>
  <env_version>1.0</env_version>
  <env_wholesaler>DEMO</env_wholesaler>
  <env_language>SP</env_language>
  <env_login>DEMO</env_login>
  <env_password>OMED</env_password>
</request>
```

Attributes

- **env_version**: Describes what version of the Web Service is used. For the moment there is only one version, the ‘1.0’. If omitted the default value ‘1.0’ is used.

- **env_wholesaler**: Indicates which reseller that makes the request.
  Mandatory.

- **env_language**: Indicates in what language the response should be given. For the moment the following 3 options exist:
  - CT for Catalan
  - SP for Spanish (default value)
  - EN for English
  If omitted the default value ‘SP’ is used.

- **env_login**: Indicates the login of the person who conducts the operation.
  Mandatory.
**env_password:** Indicates the password of the person who conducts the operation. Mandatory.

**Response**
This is the base class of all responses given by the Web Service. All entity specific responses (i.e. the DomainResponse) inherit from this class and therefore also contain all its attributes.

**XML Schema Definition**

```xml
<s:complexType name="Response">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="req_id" type="s:string"/>
    <s:element minOccurs="0" maxOccurs="1" name="ord_id" type="s:string"/>
    <s:element minOccurs="0" maxOccurs="1" name="env_version" type="s:string"/>
    <s:element minOccurs="0" maxOccurs="1" name="env_language" type="s:string"/>
    <s:element minOccurs="0" maxOccurs="1" name="status" type="s:string"/>
    <s:element minOccurs="0" maxOccurs="1" name="err_source" type="s:string"/>
    <s:element minOccurs="0" maxOccurs="1" name="err_code" type="s:string"/>
    <s:element minOccurs="0" maxOccurs="1" name="err_desc" type="s:string"/>
  </s:sequence>
</s:complexType>
```

**SOAP Example**

```xml
<Response>
  <env_version>1.0</env_version>
  <env_language>SP</env_language>
  <status>ERROR</status>
  <err_source>NOMINALIA</err_source>
  <err_code>250010</err_code>
  <err_desc>El service esta temporalmente indisponible, por favor intenta mas tarde.</err_desc>
</Response>
```

**Attributes**

- **req_id:** This field contains the identifier of the Web Service request.
- **ord_id:** This field contains a reference id to an operation that has to be paid for, for example the newDomain method.
- **env_version:** Describes what version of the Web Service is used. For the moment there is only one version, the ‘1.0’.
- **env_language:** Indicates in what language the response is given. For the moment the following 3 options exist:
  - CT for Catalan
  - SP for Spanish
  - EN for English
**status**: This field contains the status of the operation. If the operation has been successfully executed it contains 'OK' and if some error has occurred it contains 'ERROR'. If the field contains ‘Not done’ then the operation has timed out and the result can be retrieved with the getRequestStatus operation. An operation that has failed for some reason will return ‘Failed’ and must be re-executed by the client.

**err_source**: This field contains the source of the error, for example: 'NOMINALIA' if the error is on the server side. ‘CLIENT’ if for example the client has forgotten some mandatory parameter.

**err_code**: This field contains the error code of the occurred error.

**err_desc**: This field contains a textual description of the occurred error.

### Host

#### Xml Schema Definition

```xml
<s:complexType name="Host">
    <s:complexContent mixed="false">
        <s:extension base="s1:Entity">
            <s:sequence>
                <s:element minOccurs="0" maxOccurs="1" name="hst_order" type="s:string" />
                <s:element minOccurs="0" maxOccurs="1" name="hst_id" type="s:string" />
                <s:element minOccurs="0" maxOccurs="1" name="hst_name" type="s:string" />
                <s:element minOccurs="0" maxOccurs="1" name="hst_ip" type="s:string" />
                <s:element minOccurs="0" maxOccurs="1" name="hst_tcon_id" type="s:string" />
            </s:sequence>
        </s:extension>
    </s:complexContent>
</s:complexType>
```

#### SOAP Example

Taken from a newHost operation.

```xml
<host xmlns="http://www.nominalia.com/entities/Host">
    <hst_name>www.nominalia.com</hst_name>
    <hst_id>10.10.10.10</hst_id>
    <hst_tcon_id>62195</hst_tcon_id>
</host>
```

#### Attributes

**hst_id**: This field contains the id of a new host created with the newHost operation. It is also used to specify an existing host in the newDomain operation.
**hst_name**: This field contains the name of the host. Mandatory.

**hst_ip**: This field contains the IP address of the host. Mandatory.

**hst_tcon_id**: This field contains the identifier of the technical contact of the host. Mandatory.

**hst_order**: This field is used when the host is associated to a Domain. For example hst_order=1 means that the Host is host1 of the Domain. Mandatory when associated to a Domain.

**Contact**

**XML Schema Definition**

```xml
<s:complexType name="Contact">
  <s:complexContent mixed="false">
    <s:extension base="s1:Entity">
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="con_id" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="con_name" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="con_org" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="con_dep" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="con_tit" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="con_add" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="con_city" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="con_state" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="con_post" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="con_iso" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="con_country" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="con_phone" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="con_fax" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="con_mail" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="con_nif" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="con_mailing" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="con_login" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="con_password" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="con_type" type="s:string" />
      </s:sequence>
    </s:extension>
  </s:complexContent>
</s:complexType>
```

**SOAP Example**

Taken from a newContact operation.

```xml
<Contact xmlns="http://www.nominalia.com/entities/Contact">
  <con_name>Charles Bronson</con_name>
  <con_org>Nominalia Internet S.L</con_org>
  <con_dep>Comercial</con_dep>
  <con_tit>Key Account</con_tit>
  <con_add>Cr Doc Dou</con_add>
</Contact>
```
Attributes

con_id: This field identifies a Contact. It is returned when creating a new Contact. It can also identify a Contact when associated to a Domain in the domain operations.

con_name: This field contains the name of the Contact. Mandatory.

con_org, con_dep, con_tit: These fields contain the organisation and department that the contact belongs to and his title within the same. Mandatory (put ‘None’ if there is none).

con_add, con_city, con_state, con_post: These fields contain the address information of the Contact. Mandatory.

con_country: This field contains the country of the Contact, given in English. Mandatory (Only one of the two fields, con_country and con_iso is mandatory).

con_iso: This field contains the ISO-3166 country code of the Contact. Mandatory (Only one of the two fields, con_country and con_iso is mandatory).

con_phone: This field contains the phone of the Contact in international format: +<int><number> (i.e. +34 93311452323). Mandatory.

con_fax: This field contains the fax of the Contact in international format: +<int><number> (i.e. +34 93311452323)

con_mail: This field contains the email of the Contact. Mandatory.

con_nif: This field contains the NIF of the Contact. Mandatory (Only applicable when the contact is Spanish).
**con_login, con_password:**
These fields contain the login and password of the Contact. Output from the newContact operation.

**con_type:**
This specifies the type of the contact when the Contact is associated with a domain. Can be of the following types:
- a = administrative contact
- b = billing contact
- t = technical contact
Mandatory when associated with a domain.

**Owner**

**XML Schema Definition**

```
<s:complexType name="Owner">
    <s:complexContent mixed="false">
        <s:extension base="s2:Entity">
            <s:sequence>
                <s:element minOccurs="0" maxOccurs="1" name="own_id" type="s:string" />  
                <s:element minOccurs="0" maxOccurs="1" name="own_name" type="s:string" />  
                <s:element minOccurs="0" maxOccurs="1" name="own_add" type="s:string" />  
                <s:element minOccurs="0" maxOccurs="1" name="own_city" type="s:string" />  
                <s:element minOccurs="0" maxOccurs="1" name="own_state" type="s:string" />  
                <s:element minOccurs="0" maxOccurs="1" name="own_post" type="s:string" />  
                <s:element minOccurs="0" maxOccurs="1" name="own_country" type="s:string" />  
                <s:element minOccurs="0" maxOccurs="1" name="own_iso" type="s:string" />  
                <s:element minOccurs="0" maxOccurs="1" name="own_mail" type="s:string" />  
                <s:element minOccurs="0" maxOccurs="1" name="own_nif" type="s:string" />  
            </s:sequence>
        </s:extension>
    </s:complexContent>
</s:complexType>
```

**SOAP Example**
Taken from a newDomain operation.

```
<Owner xmlns="http://www.nominalia.com/entities/Owner">
    <own_name>Patrik Wallén</own_name>
    <own_add>Cr Doctor Dou 19 1-2</own_add>
    <own_city>Bcn</own_city>
    <own_state>BCN</own_state>
    <own_post>08015</own_post>
    <own_iso>ES</own_iso>
    <own_mail>jackychan@nominalia.com</own_mail>
</Owner>
```
Attributes

own_id: This field contains the identification of the owner. Used to specify an existing owner in the newDomain operation.

own_name, own_add, own_city,
own_state, own_post,
own_country, own_iso, own_mail:
These fields contain the contact information of the owner, see corresponding fields of the contact object for more details. Mandatory (Only one of the two fields own_country and own_iso is mandatory).

own_nif: This field contains the NIF of the owner. (Only applicable when owner is Spanish).

Domain

XML Schema Definition

```xml
<xs:complexType name="Domain">
  <xs:complexContent mixed="false">
    <xs:extension base="s1:Entity">
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="1" name="dom_id" type="s:string" />
        <xs:element minOccurs="0" maxOccurs="1" name="dom_sld" type="s:string" />
        <xs:element minOccurs="0" maxOccurs="1" name="dom_tld" type="s:string" />
        <xs:element minOccurs="0" maxOccurs="1" name="dom_per" type="s:string" />
        <xs:element minOccurs="0" maxOccurs="1" name="dom_date" type="s:string" />
        <xs:element minOccurs="0" maxOccurs="1" name="dom_expires" type="s:string" />
        <xs:element minOccurs="0" maxOccurs="1" name="dom_autorenew" type="s:string" />
        <xs:element minOccurs="0" maxOccurs="1" ref="s2:owner" />
        <xs:element minOccurs="0" maxOccurs="1" name="contacts" type="s1:ArrayOfContact" />
        <xs:element minOccurs="0" maxOccurs="1" name="hosts" type="s1:ArrayOfHost" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="ArrayOfContact">
  <xs:sequence>
    <xs:element minOccurs="0" maxOccurs="unbounded" name="Contact" nillable="true" type="s1:Contact" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="ArrayOfHost">
  <xs:sequence>
    <xs:element minOccurs="0" maxOccurs="unbounded" name="Host" nillable="true" type="s1:Host" />
  </xs:sequence>
</xs:complexType>
```
SOAP Example
Taken from a newDomain operation.

```xml
<domain xmlns="http://www.nominalia.com/entities/Domain">
  <dom_sld>lets gobaby</dom_sld>
  <dom_tld>net</dom_tld>
  <dom_per>1</dom_per>
</domain>

<owner xmlns="http://www.nominalia.com/entities/Owner">
  <own_name>Patrik Wallén</own_name>
  <own_add>Cr Doctor Dou 19 1-2</own_add>
  <own_city>Bcn</own_city>
  <own_post>08001</own_post>
  <own_iso>ES</own_iso>
  <own_mail>chucknorris@nominalia.com</own_mail>
  <own_form>S.L.</own_form>
  <own_nif>01111111H</own_nif>
  <own_edate>19690212</own_edate>
  <own_tm>Priv</own_tm>
  <own_gdate>20020212</own_gdate>
</owner>

<contacts>
  <Contact>
    <con_id>1000</con_id>
    <con_type>t</con_type>
  </Contact>
  <Contact>
    <con_id>2000</con_id>
    <con_type>b</con_type>
  </Contact>
  <Contact>
    <con_id>3000</con_id>
    <con_type>a</con_type>
  </Contact>
</contacts>

<hosts>
  <Host>
    <hst_order>1</hst_order>
    <hst_id>11111</hst_id>
  </Host>
  <Host>
    <hst_order>2</hst_order>
    <hst_id>22222</hst_id>
  </Host>
</hosts>
</domain>
```

Attributes

**dom_id:** This is the identification id of the domain. Returned when creating a new domain.

**dom_sld:** This is the second level domain name. For example: “nominalia” in www.nominalia.com Mandatory.
Format according to the RFC-1163.

**dom_tld:** This is the top level domain. For the moment can only be of the types: com, net, org. Mandatory. Format according to the RFC-1163.

**dom_per:** This is the registration period of the new domain. Mandatory. Values 1-9.

**dom_expires:** This is the expiring date of the domain. Output from the newDomain and getDomainInfo operation.

**dom_date:** This is the date when the domain has been created. Output from the newDomain and getDomainInfo operation.

**dom_autorenew:** This is the option autorenew of a domain registration, can be 0 or 1.

**Owner:** This is the owner of the domain. See Owner for details.

**hosts:** This is an array of the hosts associated with the domain. Specifying id and order of each host.

**contacts:** This is an array of the contacts associated with the domain. When creating or modifying a domain only the id and type of a contact can be specified to use an existing contact. If creating a new contact see Contact for details.

**Domain operations**

All domain operations take as parameter a DomainRequest that is a specialization of the Request element with the addition of a Domain element. See the Request and Domain elements for details.

All domain operations return a DomainResponse that is a specialization of the Response element with the addition of a Domain element. See the Response and Domain elements for details.

**NewDomain**

**Contract**

After a successful operation a new domain will have been created and a log of the operation will have been inserted into the database. If an error has occurred a user friendly message will be returned to the client and the real error will be written to the
application log.

**Attribute constraints**

See the Base elements section for details about the Request, Response and Domain elements.

**Owner:**

If the own_id is specified, an existing owner is used. If not, all the data of the new owner must be supplied and a new owner is created.

**Contact:**

If the con_id and con_type are specified an existing contact is used. If they are not specified all the data of the new contact must be presented and a new contact is created.

**Host:**

The hst_id and hst_order of an existing host must be specified.

**SOAP**

Request

```xml
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap=http://schemas.xmlsoap.org/soap/envelope/
 xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <soap:Body>
  <newDomain xmlns="http://ws.nominalia.com/operations">
   <request>
    <env_version>1.0</env_version>
    <env_wholesaler>DEMO</env_wholesaler>
    <env_language>SP</env_language>
    <env_login>DEMO</env_login>
    <env_password>OMED</env_password>
    <domain xmlns="http://www.nominalia.com/entities/Domain">
     <dom_sld>letsgobaby</dom_sld>
     <dom_tld>net</dom_tld>
     <dom_per>1</dom_per>
    </domain>
    <owner xmlns="http://www.nominalia.com/entities/Owner">
     <own_id>1001001</own_id>
    </owner>
   </request>
   <contacts>
    <Contact>
     <con_id>1000</con_id>
     <con_type>t</con_type>
    </Contact>
    <Contact>
     <con_id>2000</con_id>
     <con_type>b</con_type>
    </Contact>
    <Contact>
     <con_id>3000</con_id>
     <con_type>a</con_type>
    </Contact>
   </contacts>
  </newDomain>
 </soap:Body>
</soap:Envelope>
```
Response

```xml
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <newDomainResponse xmlns="http://ws.nominalia.com/operations">
      <newDomainResult>
        <req_id>428</req_id>
        <ord_id>4121228</ord_id>
        <env_version>1.0</env_version>
        <env_language>SP</env_language>
        <status>OK</status>
        <domain xmlns="http://www.nominalia.com/entities/Domain">
          <dom_id>1234</dom_id>
          <dom_sld>letsgobaby</dom_sld>
          <dom_tld>net</dom_tld>
          <dom_per>1</dom_per>
          <owner xmlns="http://www.nominalia.com/entities/Owner">
            <own_id>1001001</own_id>
          </owner>
        </domain>
        <contacts>
          <Contact>
            <con_id>1000</con_id>
            <con_type>a</con_type>
          </Contact>
          <Contact>
            <con_id>2000</con_id>
            <con_type>b</con_type>
          </Contact>
          <Contact>
            <con_id>3000</con_id>
            <con_type>c</con_type>
          </Contact>
        </contacts>
        <hosts>
          <Host>
            <hst_order>1</hst_order>
            <hst_id>1111</hst_id>
          </Host>
        </hosts>
      </newDomainResult>
    </newDomainResponse>
  </soap:Body>
</soap:Envelope>
```
ModifyDomain

Contract
After a successful operation the specified domain will have been modified with the new data supplied and a log of the operation will have been inserted into the database. If an error has occurred a user-friendly message will be returned to the client and the real error will be written to the application log.

Attribute constraints
See the Base elements section for details about the Request, Response and Domain elements.

The domain to be modified is identified with its dom_tld and dom_sld. All the data about the domain have to be supplied in the modify domain operation, not only the data to be changed.

Owner:
If the own_id is specified, an existing owner is used. If not, all the data of the new owner must be supplied and a new owner is created.

Contact:
If the con_id and con_type are specified an existing contact is used. If con_id is not specified, all the data of the new contact must be presented and a new contact is created.

Host:
The hst_id and hst_order of an existing host must be specified.

SOAP
Request
```xml
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema">  
  <soap:Body>  
    <modifyDomain xmlns="http://ws.nominalia.com/operations">  
```

<request>
  <env_version>1.0</env_version>
  <env_wholesaler>DEMO</env_wholesaler>
  <env_language>SP</env_language>
  <env_login>62195</env_login>
  <env_password>62195</env_password>
  <domain xmlns="http://www.nominalia.com/entities/Domain">
    <dom_sld>letsgobaby</dom_sld>
    <dom_tld>net</dom_tld>
    <dom_per>1</dom_per>
  </domain>
  <owner xmlns="http://www.nominalia.com/entities/Owner">
    <own_id>1001</own_id>
  </owner>
  <contacts>
    <Contact>
      <con_id>1</con_id>
      <con_type>t</con_type>
    </Contact>
    <Contact>
      <con_id>2</con_id>
      <con_type>b</con_type>
    </Contact>
    <Contact>
      <con_id>3</con_id>
      <con_type>a</con_type>
    </Contact>
  </contacts>
  <hosts>
    <Host>
      <hst_order>1</hst_order>
      <hst_id>12</hst_id>
    </Host>
    <Host>
      <hst_order>2</hst_order>
      <hst_id>11</hst_id>
    </Host>
  </hosts>
</request>

Response
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <modifyDomainResponse xmlns="http://ws.nominalia.com/operations">
      <modifyDomainResult>
        <req_id>42812</req_id>
        <env_version>1.0</env_version>
        <env_language>SP</env_language>
        <status>OK</status>
        <domain xmlns="http://www.nominalia.com/entities/Domain">
          <dom_id>12121</dom_id>
        </domain>
      </modifyDomainResult>
    </modifyDomainResponse>
  </soap:Body>
</soap:Envelope>
GetDomainInfo

Contract

After a successful operation the information about the specified domain will be returned. The domain is specified by its dom_tld and dom_sld.

Attribute constraints

See the Base elements section for details about the Request and the DomainResponse elements. The domain must belong to the user executing the operation to have authorisation.

SOAP

Request

```xml
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    ...
  </soap:Body>
</soap:Envelope>
```
<getDomainInfo xmlns="http://ws.nominalia.com/operations">
  <env>
    <env_version>string</env_version>
    <env_wholesaler>string</env_wholesaler>
    <env_login>string</env_login>
    <env_password>string</env_password>
    <env_language>string</env_language>
  </env>
  <dom_sld>string</dom_sld>
  <dom_tld>string</dom_tld>
</getDomainInfo>
</soap:Body>
</soap:Envelope>

Response
<?xml version="1.0" encoding="utf-8"?>
  <soap:Body>
    <getDomainInfoResponse xmlns="http://ws.nominalia.com/operations">
      <getDomainInfoResult>
        <domain xmlns="http://www.nominalia.com/entities/Domain">
          <dom_id>string</dom_id>
          <dom_sld>string</dom_sld>
          <dom_tld>string</dom_tld>
          <dom_per>string</dom_per>
          <dom_date>string</dom_date>
          <dom_expires>string</dom_expires>
          <dom_autorenew>string</dom_autorenew>
          <owner xmlns="http://www.nominalia.com/entities/Owner">
            <own_id>string</own_id>
          </owner>
          <contacts>
            <Contact>
              <con_id>1</con_id>
              <con_type>a</con_type>
            </Contact>
            <Contact>
              <con_id>2</con_id>
              <con_type>b</con_type>
            </Contact>
            <Contact>
              <con_id>3</con_id>
              <con_type>c</con_type>
            </Contact>
          </contacts>
          <hosts>
            <Host>
              <hst_order>1</hst_order>
              <hst_id>11</hst_id>
            </Host>
            <Host>
              <hst_order>2</hst_order>
              <hst_id>12</hst_id>
            </Host>
          </hosts>
        </domain>
      </getDomainInfoResult>
    </getDomainInfoResponse>
  </soap:Body>
</soap:Envelope>
Host operations

All host operations take as parameter a HostRequest that is a specialization of a Request with the addition of a Host element. See the Request element for details.

All host operations return a HostResponse that is a specialization of a Response with the addition of a Host. See the Response element for details.

NewHost

Contract

After a successful operation a new host will have been created and a log of the operation will have been inserted into the database. If an error has occurred a user friendly message will be returned to the client and the real error will be written to the application log.

Attribute constraints

See the Base elements section for details about the Request, Response and Host objects.

SOAP

Request

```xml
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<soap:Body>
<newHost xmlns="http://ws.nominalia.com/operations">
<request>
<env_version>1.0</env_version>
<env_wholesaler>DEMO</env_wholesaler>
<env_language>SP</env_language>
<env_login>62195</env_login>
<env_password>62195</env_password>
<host xmlns="http://www.nominalia.com/entities/Host">
<hst_name>dns.serv.com</hst_name>
<hst_ip>10.10.10.100</hst_ip>
<hst_tcon_id>22222</hst_tcon_id>
</host>
</request>
</newHost>
</soap:Body>
</soap:Envelope>
```

Response
ModifyHost

Contract

After a successful operation the specified host will have been modified with the new data supplied and a log of the operation will have been inserted into the database. If an error has occurred a user-friendly message will be returned to the client and the real error will be written to the application log.

Attribute constraints

See the Base elements section for details about the Request, Response and Host objects. All the attributes of the host must be specified in the modifyHost operation.

SOAP

Request

<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <modifyHost xmlns="http://ws.nominalia.com/operations">
      <request>
        <env_version>1.0</env_version>
        <env_wholesaler>DEMO</env_wholesaler>
        <env_language>SP</env_language>
        <env_login>62195</env_login>
        <env_password>62195</env_password>
        <host xmlns="http://www.nominalia.com/entities/Host">
          <hst_name>host.serv.com</hst_name>
          <hst_ip>10.10.10.100</hst_ip>
          <hst_tcon_id>6</hst_tcon_id>
        </host>
      </request>
    </modifyHost>
  </soap:Body>
</soap:Envelope>
Response

<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <modifyHostResponse xmlns="http://ws.nominalia.com/operations">
      <modifyHostResult>
        <req_id>428</req_id>
        <env_version>1.0</env_version>
        <env_language>SP</env_language>
        <status>OK</status>
        <host xmlns="http://www.nominalia.com/entities/Host">
          <hst_id>898</hst_id>
          <hst_name>host.serv.com</hst_name>
          <hst_ip>10.10.10.100</hst_ip>
          <hst_tcon_id>6</hst_tcon_id>
        </host>
      </modifyHostResult>
    </modifyHostResponse>
  </soap:Body>
</soap:Envelope>

GetHostInfo

Contract

After a successful operation the information about the specified host will be returned.
The host is specified with either hst_id or hst_name.

Attribute constraints

See the Base attributes section for details about the Request, HostResponse elements.
The host is specified with either hst_id or hst_name. The domain must belong to the
user executing the operation to have authorisation.

SOAP

Request

<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope">
  <soap:Body>
    <getHostInfo xmlns="http://ws.nominalia.com/operations">
      <env>
        <env_version>string</env_version>
        <env_wholesaler>string</env_wholesaler>
        <env_login>string</env_login>
        <env_password>string</env_password>
        <env_language>string</env_language>
      </env>
    </getHostInfo>
  </soap:Body>
</soap:Envelope>
Response

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
>
  <soap:Body>
    <getHostInfoResponse xmlns="http://ws.nominalia.com/operations">
      <getHostInfoResult>
        <host xmlns="http://www.nominalia.com/entities/Host">
          <hst_order>string</hst_order>
          <hst_id>string</hst_id>
          <hst_name>string</hst_name>
          <hst_ip>string</hst_ip>
          <hst_tcon_id>string</hst_tcon_id>
        </host>
      </getHostInfoResult>
    </getHostInfoResponse>
  </soap:Body>
</soap:Envelope>

Contact operations

All contact operations take as parameter a ContactRequest that is a specialization of a Request with the addition of a Contact. See the Request element for details.

All conact operations return a ContactResponse that is a specialization of a Response with the addition of a Contact. See the Response element for details.

NewContact

Contract

After a successful operation a new contact will have been created and a log of the operation will have been inserted into the database. If an error has occurred a user-friendly message will be returned to the client and the real error will be written to the application log.

Attribute constraints

See the Base elements section for details about the Request, Response and Contact elements.

SOAP

Request

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<soap:Body>
  <newContact xmlns="http://ws.nominalia.com/operations">
    <request>
      <env_wholesaler>DEMO</env_wholesaler>
      <env_login>1</env_login>
      <env_password>2</env_password>
      <contact xmlns="http://www.nominalia.com/entities/Contact">
        <con_name>Patrik Wallen</con_name>
        <con_org>Spenta Consulting S.A.</con_org>
        <con_dep>Technology</con_dep>
        <con_tit>Architecto de sistemas</con_tit>
        <con_add>Dr Dou 19 1-2</con_add>
        <con_city>Barcelona</con_city>
        <con_post>08001</con_post>
        <con_iso>ES</con_iso>
        <con_phone>+34630161953</con_phone>
        <con_mail>patrik.wallen@spenta.es</con_mail>
        <con_nif>00000000P</con_nif>
      </contact>
    </request>
  </newContact>
</soap:Body>
</soap:Envelope>

Response
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <newContactResponse xmlns="http://ws.nominalia.com/operations">
      <newContactResult>
        <req_id>421</req_id>
        <env_version>1.0</env_version>
        <env_language>SP</env_language>
        <status>OK</status>
        <contact xmlns="http://www.nominalia.com/entities/Contact">
          <con_id>81</con_id>
          <con_name>Patrik Wallen</con_name>
          <con_org>Spenta Consulting S.A.</con_org>
          <con_tit>Architecto de sistemas</con_tit>
          <con_add>Dr Dou 19 1-2</con_add>
          <con_city>Barcelona</con_city>
          <con_post>08001</con_post>
          <con_iso>ES</con_iso>
          <con_phone>+34630161953</con_phone>
          <con_mail>patrik.wallen@spenta.es</con_mail>
          <con_nif>00000000P</con_nif>
        </contact>
      </newContactResult>
    </newContactResponse>
  </soap:Body>
</soap:Envelope>
ModifyContact

Contract
After a successful operation the specified contact will have been modified with the new data and a log of the operation will have been inserted in the database. If an error has occurred a user-friendly message will be returned to the client and the real error will be written to the application log.

Attribute constraints
See the Base elements section for details about the Request, Response and Contact elements. All the attributes of the contact must be specified in the modifyContact operation.

SOAP
Request
```xml
<?xml version="1.0" encoding="utf-8"?
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<soap:Body>
<modifyContact xmlns="http://ws.nominalia.com/operations">
 <request>
  <env_wholesaler>DEMO</env_wholesaler>
  <env_login>1</env_login>
  <env_password>2</env_password>
  <contact xmlns="http://www.nominalia.com/entities/Contact">
   <con_name>Patrik Wallen</con_name>
   <con_org>Spenta Consulting S.A.</con_org>
   <con_dep>Technology</con_dep>
   <con_tit>Architecto de sistemas</con_tit>
   <con_add>Dr Dou 19 1-2</con_add>
   <con_city>Barcelona</con_city>
   <con_state>Bcn</con_state>
   <con_post>08001</con_post>
   <con_iso>ES</con_iso>
   <con_phone>+34630161953</con_phone>
   <con_fax></con_fax>
   <con_mail>patrik.wallen@spenta.es</con_mail>
   <con_nif>00000000P</con_nif>
  </contact>
 </request>
</modifyContact>
</soap:Body>
</soap:Envelope>
```

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

Contract

After a successful operation the information about the specified contact will be returned. The contact is specified with con_id.

Attribute constraints

See the Base elements section for details about the Request, ContactResponse elements. The contact must belong to the user executing the operation to have authorisation.

SOAP

Request

```xml
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <getContactInfo xmlns="http://ws.nominalia.com/operations">
         <env_version>string</env_version>
         <env_wholesaler>string</env_wholesaler>
         <env_login>string</env_login>
         <env_password>string</env_password>
         <env_language>string</env_language>
      </getContactInfo>
   </soap:Body>
</soap:Envelope>
```
Nominalia Web Services

Response

<?xml version="1.0" encoding="utf-8"?>
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getContactInfoResponse xmlns="http://ws.nominalia.com/operations">
      <getContactInfoResult>
        <contact xmlns="http://www.nominalia.com/entities/Contact">
          <con_id>string</con_id>
          <con_name>string</con_name>
          <con_org>string</con_org>
          <con_dep>string</con_dep>
          <con_tit>string</con_tit>
          <con_add>string</con_add>
          <con_city>string</con_city>
          <con_state>string</con_state>
          <con_post>string</con_post>
          <con_iso>string</con_iso>
          <con_country>string</con_country>
          <con_phone>string</con_phone>
          <con_fax>string</con_fax>
          <con_mail>string</con_mail>
          <con_nif>string</con_nif>
          <con_mailing>string</con_mailing>
          <con_login>string</con_login>
          <con_password>string</con_password>
          <con_type>string</con_type>
        </contact>
      </getContactInfoResult>
    </getContactInfoResponse>
  </soap:Body>
</soap:Envelope>

GetContactListInfo

Contract

After a successful operation an array of contacts with the specified con_name will be returned.

Attribute constraints

See the Base elements section for details about the Request, Response and ContactArray (contacts) objects.

SOAP

Request

<?xml version="1.0" encoding="utf-8"?>
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<getContactListInfo xmlns="http://ws.nominalia.com/operations">
<env>
<env_version>string</env_version>
<env_wholesaler>string</env_wholesaler>
<env_login>string</env_login>
<env_password>string</env_password>
<env_language>string</env_language>
</env>
<con_name>string</con_name>
</getContactListInfo>
</soap:Body>
</soap:Envelope>

Response

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<getContactListInfoResponse xmlns="http://ws.nominalia.com/operations">
<getContactListInfoResult>
<contacts xmlns="http://www.nominalia.com/entities/Contact">
<con_id>string</con_id>
<con_name>string</con_name>
<con_org>string</con_org>
<con_dep>string</con_dep>
<con_tit>string</con_tit>
<con_add>string</con_add>
<con_city>string</con_city>
<con_state>string</con_state>
<con_post>string</con_post>
<con_iso>string</con_iso>
<con_country>string</con_country>
<con_phone>string</con_phone>
<con_fax>string</con_fax>
<con_mail>string</con_mail>
<con_nif>string</con_nif>
<con_mailing>string</con_mailing>
<con_login>string</con_login>
<con_password>string</con_password>
<con_type>string</con_type>
</contacts>
</getContactListInfoResult>
<contacts xmlns="http://www.nominalia.com/entities/Contact">
<con_id>string</con_id>
<con_name>string</con_name>
<con_org>string</con_org>
<con_dep>string</con_dep>
<con_tit>string</con_tit>
<con_add>string</con_add>
<con_city>string</con_city>
<con_state>string</con_state>
<con_post>string</con_post>
<con_iso>string</con_iso>
<con_country>string</con_country>
<con_phone>string</con_phone>
<con_fax>string</con_fax>
<con_mail>string</con_mail>
<con_nif>string</con_nif>
<con_mailing>string</con_mailing>
<con_login>string</con_login>
<con_password>string</con_password>
<con_type>string</con_type>
</contacts>
</getContactListInfoResponse>
</soap:Body>
</soap:Envelope>
General operations

GetRequestStatus

Contract
After a successful operation the status of the specified operation will be returned. If the operation specified has terminated successfully its result will also be returned. The return type will in this case be dependent of the type of the specified operation. For example a GetRequestStatus of a newDomain operation will return a DomainResponse if successful or a simple Response if not done or unsuccessful.

SOAP
Request

```xml
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <GetRequestStatus xmlns="http://ws.nominalia.com/operations">
        &lt;env
            &lt;env_version>1.0</env_version>
            &lt;env_wholesaler>MAMI</env_wholesaler>
            &lt;env_login>AuStIn</env_login>
            &lt;env_password>pOWeRs</env_password>
            &lt;env_language>SP</env_language>
        &lt;/env>
        &lt;/GetRequestStatus>
    </soap:Body>
</soap:Envelope>
```

Response

```xml
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <GetRequestStatusResponse xmlns="http://ws.nominalia.com/operations"/>
    </soap:Body>
</soap:Envelope>
```
Response (XML Schema description)

The response given by this operation will be the same response that the operation in question would have given. In the case that the operation has not yet terminated or in the case of error the response will be the base Response. For examples see corresponding operations and the base entity Response.

```xml
<s:element name="GetRequestStatusResponse">
  <s:complexType>
    <s:sequence>
      <s:choice minOccurs="1" maxOccurs="1">
        <s:element minOccurs="0" maxOccurs="1" name="HostResponse" type="s0:HostResponse" />
        <s:element minOccurs="0" maxOccurs="1" name="DomainResponse" type="s0:DomainResponse" />
        <s:element minOccurs="0" maxOccurs="1" name="Response" type="s0:Response" />
        <s:element minOccurs="0" maxOccurs="1" name="ContactResponse" type="s0:ContactResponse" />
      </s:choice>
    </s:sequence>
  </s:complexType>
</s:element>
```

CheckDomain

This is the only public operation of the Nominalia Web Services that does not need authorisation. It can be used to test the connection to the Web Service.

Contract

Checks the availability of a domain. This operation uses the Whois operation of Nominalia. If the domain is available the answer is 1 and if not the answer is 0. The domain is specified with dom_sld and dom_tld.

SOAP

Request

```xml
<?xml version="1.0" encoding="utf-8"?>
  <soap:Body>
    <checkDomain xmlns="http://ws.nominalia.com/operations">
      <sld>string</sld>
      <tld>string</tld>
    </checkDomain>
  </soap:Body>
</soap:Envelope>
```

Response

```xml
<?xml version="1.0" encoding="utf-8"?>
```
**Error messages**

**status**
If the status field of the Response objects indicates an error, in most cases the error is described in the env_err_source, env_err_code and env_err_desc fields.

**env_err_source, env_err_desc**
The env_err_source can either be NOMINALIA or CLIENT. In the env_err_desc there is a text describing the possible cause of the error.

**error_code**
The error_code corresponds to the following structure:

**10000 – 130000**
The error code is a numeric value of 6 numbers, formed by the concatenation of 3 pairs of numbers. XXYYZZ where:

<table>
<thead>
<tr>
<th>XX</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Environment</td>
</tr>
<tr>
<td>2</td>
<td>Domain</td>
</tr>
<tr>
<td>3</td>
<td>Owner</td>
</tr>
<tr>
<td>4</td>
<td>Contact</td>
</tr>
<tr>
<td>5</td>
<td>Contact admin</td>
</tr>
<tr>
<td>6</td>
<td>Contact tech</td>
</tr>
<tr>
<td>7</td>
<td>Contact billing</td>
</tr>
<tr>
<td>8</td>
<td>Host</td>
</tr>
<tr>
<td>10</td>
<td>Mail forwarding</td>
</tr>
<tr>
<td>11</td>
<td>Web forwarding</td>
</tr>
<tr>
<td>12</td>
<td>Request</td>
</tr>
</tbody>
</table>

YY indicates the field that contains the error, depending on the section.

<table>
<thead>
<tr>
<th>ZZ</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>The field is empty and mandatory.</td>
</tr>
<tr>
<td>02</td>
<td>The value of the field is unknown for NOMINALIA.</td>
</tr>
<tr>
<td>03</td>
<td>The format of the field is erroneous.</td>
</tr>
<tr>
<td>04</td>
<td>The format of the field is erroneous for ESNIC.</td>
</tr>
<tr>
<td>05</td>
<td>The domain already exists.</td>
</tr>
<tr>
<td>08</td>
<td>The format of the field is not yet supported.</td>
</tr>
</tbody>
</table>
[09] > Internal error.

Example:
20303 The name of the domain is not correct
41101 The country of the contact is empty

200000-
Internal Web Service errors used for internal debugging.

WSDL
The .NET Framework automatically generates a WSDL file accessible from the main page of the application formally describing the interfaces and endpoints of all the operations of the Nominalia Web Service according to the W3C standard of the Web Service Description Language. For an extensive documentation of the WSDL syntax please refer to the W3C webpage.

WSDL document URL:

WSDL document URL (RPC-Encoded):
Client Code Example

Microsoft .NET

The following code snipped is an example of a newDomain operation. It assumes that a Web Reference of the Nominalia Web Service has been imported as NominaliaWS.

```csharp
DomainRequest request = new DomainRequest();
request.domain = new Domain();

// Environment variables
request.env_language = "SP";
request.env_version = "1.0";
request.env_wholesaler = "DEMO";
request.env_login = "X";
request.env_password = "Y";

// Domain info
request.domain.dom_tld = "net";
request.domain.dom_sld = "letsgobaby";
request.domain.dom_per = "1";

// Create new owner
Owner own = new Owner();
own.own_name = "Patrik Wallén";
own.own_add = "Cr Doctor Dou 19 1-2";
own.own_city = "Bcn";
own.own_state = "BCN";
own.own_post = "08015";
own.own_iso = "ES";
own.own_email = "$p@private.com";
own.own_nif = "011111111H";
own.own_edate = "19690212";
own.own_form = "S.L.";
own.own_gdate = "20020212";
own.own_tm = "Private";
request.domain.owner = own;

// Use existing contacts
Contact[] clist = new Contact[3];
clist[0] = new Contact();
clist[0].con_type = "t";
clist[0].con_id = "195";
clist[1] = new Contact();
clist[1].con_type = "b";
clist[1].con_id = "295";
clist[2] = new Contact();
clist[2].con_type = "a";
clist[2].con_id = "395";
request.domain.contacts = clist;

// Use existing hosts
Host[] hlist = new Host[2];
hlist[0] = new Host();
```

124
hlist[0].hst_id = "11";
hlist[0].hst_order = "1";
hlist[1] = new Host();
hlist[1].hst_id = "12";
hlist[1].hst_order = "2";
request.domain.hosts = hlist;

// Create the Web Service client proxy and call the newDomain method
NominaliaWS ws = new NominaliaWS();
DomainResponse resp = ws.newDomain(request);

// Show parts of the result
Console.WriteLine("Id: "+resp.ord_id);
Console.WriteLine("Status: "+resp.status);
Console.WriteLine("Error desc: "+resp.err_desc);
Console.WriteLine("Domain id: "+resp.domain.dom_id);

Listing: Code snippet of a newDomain operation (C# .NET)
The following PHP examples use the NuSOAP toolkit that can be found at:
http://www.nusoap.com

An excellent text about Web Services with PHP, that is an example chapter of the book: “Professional Open Source Web Services” from WROX, can be downloaded from: http://www.phpgroup.org/pdf/7469_Chap08.pdf

First there is an example using the Document/Literal SOAP message style of a newDomain operation.
Second there is an example of a client realising a getDomainInfo and a getContactInfo using the RPC/Encoded message style.

**NewDomain (Document/Literal)**

```php
<?PHP
  // Include the SOAP-toolkit
  require_once('nusoap-0.6.1\nusoap.php');

  // Specify where the webservice is located
  $wsdlfile='http://ws.nominalia.com/ws/NominaliaWS.asmx?WSDL';

  // Using the document-literal SOAP messages
  $msg = "<newDomain xmlns="http://ws.nominalia.com/operations">
  <request>
    <env_version>1.0</env_version>
    <env_wholesaler>DEMO</env_wholesaler>
    <env_language>SP</env_language>
    <env_login>secretlogin</env_login>
    <env_password>moresecretpassword</env_password>
    <domain xmlns="http://www.nominalia.com/entities/Domain">
      <dom_sld>patrikwallen</dom_sld>
      <dom_tld>com</dom_tld>
      <dom_per>10</dom_per>
      <owner xmlns="http://www.nominalia.com/entities/Owner">
        <own_id>1111111</own_id>
      </owner>
    </domain>
    <contacts>
      <Contact>
        <con_id>1</con_id>
        <con_type>t</con_type>
      </Contact>
      <Contact>
        <con_id>2</con_id>
        <con_type>b</con_type>
      </Contact>
      <Contact>
        <con_name>Patrik Wallen</con_name>
        <con_org>Spenta Consulting S.A.</con_org>
      </Contact>
      <Contact>
        <con_add>Dr Dou 19 1-2</con_add>
      </Contact>
  </request>
</newDomain>"
```
<?php

// create SOAP-client and call method
$soapclient = new soapclient($wsdlfile,'WSDL');
$arr = $soapclient->call('newDomain',array($msg));

// Print results
print 'REQUEST:<xmp>'.$soapclient->request.'</xmp>);
print 'RESPONSE:<xmp>'.$soapclient->response.'</xmp>);

// Use results in $arr
echo 'Status: '.$arr['status'];
echo 'New domain id: '.$arr['domain']['dom_id'];
?>
GetDomainInfo and GetContactInfo (RPC/Encoded)

```php
<?PHP
// Include the SOAP-toolkit
require_once('nusoap-0.6.1\nusoap.php');

// Specify where the webservice is located
$wsdlfile='http://ws.nominalia.com/ws/rpc/NominaliaWS.asmx?WSDL';

// Specify parameter for the getDomainInfo method
$env = array('1.0','wholesaler','login','pass','SP');
$dom_sld = 'internetnominalia';
$dom_tld = 'net';

// create SOAP-client and proxy and call method
$soapclient = new soapclient($wsdlfile,'WSDL');
$proxy = $soapclient->getProxy();
$arr = $proxy->getDomainInfo($env,$dom_sld,$dom_tld);

// Print Request and response XML for debug
print 'REQUEST:<xmp>'.$proxy->request.'</xmp> ;
print 'RESPONSE:<xmp>'.$proxy->response.'</xmp> ;

// Use results in $arr
echo 'Status: '.$arr['status'];
echo 'Domain id: '.$arr['domain']['dom_id'];

// Specify parameter for the getContactInfo method
$env = array('1.0','DEMO','wholesaler','login','pass','SP');
// call method
$arr = $proxy->getContactInfo($env,'62195');

// Print Request and response XML for debug
print 'REQUEST:<xmp>'.$proxy->request.'</xmp> ;
print 'RESPONSE:<xmp>'.$proxy->response.'</xmp> ;

// Use results in $arr
echo 'Status: '.$arr['status'];
echo 'Contact name: '.$arr['contact']['con_name'];
echo 'Contact organization: '.$arr['contact']['con_org'];
?>
```