A survey on the usage of XML in Component-based Software Engineering

Michał Rusin
A survey on the usage of XML in Component-based Software Engineering

Michał Rusin

Examinator:
Samantha Jenkins

Handledare:
Steven Kirk

Trollhättan, 2005

2005:DS04
Abstract / Summary

XML and CBSE are being adopted very rapidly in the software industry despite being relatively new technologies. This paper provides an overview of how these technologies work and relate to each other. Its main function is to present XML-based technologies like SOAP, XML-based composition languages, XML functional languages as well as try to answer the question: Does XML impact CBSE? The conclusion of this work is that XML works well in CBSE and together they create a powerful tool for software industry showing a great promise for the future. But it’s not sure if they will be used in their current form or in a more evolved one.
1. Introduction ........................................................................................................... 1
2. Background .......................................................................................................... 1
   2.1. Component-based Software Engineering ..................................................... 1
      2.1.1. What is it? ................................................................. 1
      2.1.2. Why use CBSE? ....................................................... 1
      2.1.3. What is a component? ................................................ 2
   2.2. XML ................................................................................................................... 2
      2.2.1. What is it? .............................................................................. 2
      2.2.2. History ................................................................................. 2
      2.2.3. Why XML? .......................................................................... 2
      2.2.4. Pros and cons ................................................................. 2
      2.2.5. Structure ................................................................. 3
      2.2.6. Example ................................................................. 3
3. SOAP – Simple Object Access Protocol ............................................................ 3
   3.1. What is it? ....................................................................................................... 3
   3.2. History ........................................................................................................... 4
   3.3. Why SOAP? .................................................................................................. 4
      3.3.1. Reasons to use SOAP ...................................................... 4
      3.3.2. Reasons not to use SOAP ................................................. 5
4. XML-based composition languages .................................................................... 5
   4.1. XUL ............................................................................................................... 5
      4.1.1. What is it? ................................................................. 5
      4.1.2. History ................................................................. 6
      4.1.3. Why XUL? .............................................................................. 6
      4.1.4. Pros and cons ................................................................. 6
      4.1.5. Example ................................................................. 7
   4.2. XAML ............................................................................................................. 7
      4.2.1. What is it? ................................................................. 7
      4.2.2. Why XAML? ............................................................... 7
      4.2.3. Pros and cons ................................................................. 8
      4.2.4. Example ................................................................. 8
5. XML functional languages ................................................................................... 8
   5.1. XPath ............................................................................................................. 8
      5.1.1. What is it? ................................................................. 8
      5.1.2. Example ................................................................. 8
   5.2. XSLT .............................................................................................................. 9
      5.2.1. What is it? ................................................................. 9
      5.2.2. Pros and cons ................................................................. 9
      5.2.3. Example ................................................................. 10
   5.3. XQuery ......................................................................................................... 10
      5.3.1. What is it? ................................................................. 10
      5.3.2. Pros and cons ................................................................. 10
      5.3.3. Example ................................................................. 11
6. Discussion and comparison ................................................................................. 11
7. Future .................................................................................................................. 12
8. Conclusion ............................................................................................................ 12
9. References ............................................................................................................ 13
Appendix 1 ............................................................................................................. 15
Appendix 2 ............................................................................................................. 17

A survey on the usage of XML in Component-based Software Engineering
Abstract

XML and CBSE are being adopted very rapidly in the software industry despite being relatively new technologies. This paper provides an overview of how these technologies work and relate to each other. Its main function is to present XML-based technologies like SOAP, XML-based composition languages, XML functional languages as well as try to answer the question: Does XML impact CBSE? The conclusion of this work is that XML works well in CBSE and together they create a powerful tool for software industry showing a great promise for the future. But it’s not sure if they will be used in their current form or in a more evolved one.

1. Introduction

Software Engineering is one of the most important parts of Information Technology and a part of it which is gaining bigger and bigger popularity called Component-based Software Engineering (CBSE). CBSE is something that may change the way we think about programming. Also in this field XML (Extensible Markup Language [1]) is present. And, what’s most important, both of them are also becoming more popular.

There are many concepts of using XML and XML-based tools in Software Engineering. However there are some well-known ones already (ex. SOAP - Simple Object Access Protocol [2]; XPath [3]; XQuery [4]), the future of XML in Software Engineering looks really promising. New ways of using XML as a base for programming languages are becoming more obvious now, especially while Microsoft works on its new platform with code name ‘Longhorn’ [5]. It should be the significant moment in the history of using XML as a base for programming language.

With the development of the concept of using XML as a base for new programming tools many new questions have appeared. Probably the most interesting one is about the impact of XML in Component-based Software Engineering. This paper investigates and answers this question.

In section 2, the background for this paper is presented. A presentation of CBSE and an introduction to XML is provided. Following sections discuss its applications in Software Engineering (Component-based Software Engineering). The next section (3) describes SOAP with all its advantages and disadvantages. In Section 4 XML-based composition languages (XUL – XML-based User Interface Language [6], XAML – Extensible Application Markup Language [7]) are presented. Section 5 is dedicated to XML functional languages (XSLT – Transformation Extensible Stylesheet Language, XQuery). Next section contains discussion and comparison of these techniques while Section 7 presents future of XML in CBSE. The conclusion is situated in Section 8 while Section 9 is a list of references.

2. Background

In this section general information about CBSE and XML are provided. These two concepts form the basis for rest of this paper. First of all, there’s a short description of CBSE, then, in the next subsection XML is presented with its history, pros and cons and an example of using it.

2.1. Component-based Software Engineering

2.1.1. What is it?

Briefly, Component-based Software Engineering (CBSE) is a way of creating large-scale and complex software by reusing already existing pieces of software (components). It also involves development of such reusable parts and improvement of already existing systems by component replacement and customization.

2.1.2. Why use CBSE?

There are a number of reasons why CBSE is a significant technology with impressive future. CBSE makes developing applications cheaper (reusing components is cheaper than writing everything from scratch) [8]. Also applications written using CBSE have a competitive edge over commercial-off-the-shelf products. The whole application looks just like client expected and doesn’t have any functions that client didn’t want because only components containing the necessary functionality are used to build it. Another relevant factor is that it is easier to update Component-based Software by changing components [9]. Using CBSE, a development project can be divided into small, clearly-defined subprojects. It shortens the development time [10].
2.1.3. What is a component?

‘Component is a unit of independent deployment. Well separated from environment and other components. It is a sufficiently self-contained unit of third party composition which has no persistent state and cannot be distinguished from copies of itself.’ Definition by C. Szyperski [9].

2.2. XML

2.2.1. What is it?

In short we can say that XML (Extensible Markup Language) is a simple markup language created to structure, store and send information, but not control or specify the presentation of the information.

2.2.2. History

The history of XML starts with the moment of creating a standard for descriptive markup language called SGML in 1978. The next step was to approve it as an international ISO standard in 1986 [11]. Defined by ISO, SGML provides a language for describing markup, not just a set of particular markup elements. Also tagging of content is separated from its presentation and style – a DTD file (Document Type Definition) sets the rules for using tags in a specified SGML document.

Because SGML was found to be an interesting tool but not exactly the tool companies were looking for (it was unsuitable for quick and easy Web publishing [12]), the new standard called HTML has been created. It is a language created in CERN around 1990 to display data and to focus on how data looks. The HTML language is now ubiquitous, being the data format used to hold the vast majority of the pages on the World-Wide Web. But after a few years some defects of HTML became apparent. Hyper Text Markup Language derives from SGML, but in spite of being an SGML application, the HTML DTD emphasizes presentation over structure. There have been some attempts made to fix HTML like creating a Dynamic HTML language – DHTML [13], but it wasn’t the technology that everyone was expecting.

In 1996 World Wide Web Consortium (W3C) in cooperation with SGML specialists started to work on a new international standard which would be a permanent solution for HTML problems and would simplify large-scale electronic publishing. In fact a newly created standard – established in 1998 – is a simplified version of SGML and has been called XML (eXtensible Markup Language). It has a number of advantages over SGML, one such advantage being the relative ease of creating tools which use and process the language by software companies, compared with SGML [14].

2.2.3. Why XML?

The main reason for the creation of XML was to simplify large-scale electronic publishing by providing a language which could be used to structure, store and send information. Many software companies now use XML as their standard data format for this purpose, and the author is not aware of any existing alternative language which would realize these functions in an easier way. In order to use XML, however, software must be written in order to provide the functionality of reading and writing data in XML format. Many such libraries already exist (many of these are open-source, e.g. [15, 16, 17]), and these can be linked with an application program to provide an XML capability.

2.2.4. Pros and cons

There are many advantages of using XML. One of them, and probably the most important one, is that data stored as an XML document is available to everyone. The specifications for how to create a valid XML file are open and freely available for everyone to read and use [1]. Because XML is platform and language independent, it is easy to exchange data between applications written in different languages and working on different platforms. It makes XML really useful in the real world by avoiding dangers of vendor ‘lock-in’ [18]. It is also the most important advantage of XML when talking about using it in Component-based Software Engineering.

An all too common experience of users and developers is encountering computer systems and databases containing data in incompatible formats. Allowing communication between them is a time-consuming task for developers, so XML might be some kind of a “silver bullet” here. If everyone used XML as a standard way of storing and presenting data this problem would be largely eliminated.

Accessing XML data is really simple; even HTML browsers (e.g. Mozilla Firefox, MS Internet Explorer) allow viewing XML documents. Because the structure of this type of document is clear and easy to understand you can view it using text editors and still being able to find the data you’re looking for.

In future XML should be the only way of exchanging data. It would solve most of the problems with converting files to different
formats. But it can be possible only when all of the huge software developers (e.g. Microsoft) will agree.

Nowadays XML is an important part of exchanging data on the Web and, as a part of Component-based Software Engineering, is becoming more important as a way to communicate between components.

2.2.5. Structure

Generally the XML language is divided into three main categories [1, 9]:
- XML elements
- XML attributes
- Unstructured text

XML requires that unstructured text is properly quoted to avoid inclusions of metacharacters. The most important metacharacters are “<”, “>” and “&” (used to introduce a quoting sequence).

Elements of XML are introduced by the form `<tag …>` and closed by `</tag>`. There are also two special kinds of elements introduced by `<!- … ->` and `<? … ?>`. These are called processing instructions and indicate meta-information about the document. Tags are not predefined, each developer has to create his own tags – that’s why it’s called extensible.

Each of the elements may have one or more attributes. Attributes are added according to the schema `<tag attr1="1" attr2="2">`. They are used only when an element is introduced. By closing an element values of attributes revert to a default value. No two attributes can have the same name.

A proper XML document should be created according to the structure [1]:

```xml
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

The DTD (Document Type Definition) may be included in the XML file or may exist as a separate file, usually with a ‘.dtd’ file extension. The full specification for creating DTDs is freely available as a part of XML specification [1]. It is created according to the structure :

```xml
<!DOCTYPE _name [ 
  <!ELEMENT _name (_element)> 
  <!ELEMENT _element (#PCDATA)> 
]>
```

2.2.6. Example

Listing 1 presents a simple example of an XML file. As you can see the file consists of few tags and text between them. It is a message sent from Peter to Gustav – an invitation to a party.

Explanation of tags used:

```xml
<?xml version="1.0" encoding="ISO-8859-1"?> declaration of XML version used
<note> beginning of a note
  <to> name of receiver
  <from> name of sender
  <heading> heading
  <body> text of a note

Listing 1. Example of an XML file.

3. SOAP – Simple Object Access Protocol

This section is created to present SOAP (Simple Object Access Protocol) [19]. This technology is widely used now, and probably in future it will be more popular.

3.1. What is it?

Simple Object Access Protocol (SOAP) is a message-based protocol based on XML. Its main function is accessing services on the Web and allowing communication between applications via Internet.

The main idea for SOAP came from Microsoft and IBM. SOAP defines how to create messages which are sent between applications. All messages sent and received by SOAP are XML-based, so the syntax of them is similar to the syntax of XML. SOAP as a protocol was created to work over Internet using HTTP. Its main function is similar to DCOM (Distributed Component Object Model) [19] and CORBA (Common Object Request Broker Architecture) [21], but SOAP makes it much easier and doesn’t need that much programming. Also because of its simple structure it may be more easily used to implement a messaging system.

SOAP is supported in COM, DCOM, MS Internet Explorer, MS’s Java implementations and is a key element of MS’s .NET architecture for Internet application development.

```xml
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12
          /soap-envelope"
```
A survey on the usage of XML in Component-based Software Engineering

Soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Header>
  ...
</soap:Header>
<soap:Body>
  ...
<soap:Fault>
  ...
</soap:Fault>
</soap:Body>
</soap:Envelope>

Schema 1. An example of a valid SOAP message.

3.2. History

The year 1998 was decisive in the history of Simple Object Access Protocol. In fact, this is the year when SOAP has started and version 1.0 of XML has been defined. It was the first schema language for XML [22, 23].

The early versions of the SOAP specification (including XML-RPC – XML Remote Procedure Calling) were focused on defining a type system. Firstly, the type system of SOAP did not have many primitive types – only composites that are accessed by name (structs) and composites accessed by position (arrays). Then behavioural types (which describe operations and methods which act on or manipulate the primitive types) were modelled. It was done by defining operations and methods in terms of pairs of structs. Then they were aggregated into interfaces.

But 1998 wasn’t actually the year when SOAP has been defined. Because Microsoft according to its politics was working on other standards like XML-Data and COM, development work on SOAP was held up until Dave Winer defined on his own the XML-RPC specification [24] based on subsetting the original SOAP type system.

The SOAP specification finally was brought in 1999 when the W3C XML specification was completed. Primitive types in XML were a subset of what was needed for SOAP, and in addition, the composite type system was mostly a superset of what was needed for SOAP [25]. The 1999 SOAP specification was explaining how to model type references and arrays in the W3C XML Schema type system. However the XML specification was still changing, so the SOAP specification had to be changed as often as the XML specification.

Summarizing, the first version of SOAP was actually created by Microsoft. There were a lot of bugs which should be removed with introducing version 1.1. But, in fact, IBM (developer of ver.1.1) hasn’t improved it. IBM has only reorganized the specification to make it clearer.

SOAP became an official standard and was recommended by W3C in 2003 when version 1.2 was released [19].

3.3. Why SOAP?

There is number of reasons why you should use SOAP as well as why you shouldn’t use it. In this subsection pros and cons of this technology are presented [26].

3.3.1. Reasons to use SOAP

First of all it uses XML to communicate. This makes SOAP an easy way to accomplish remote object, component or service communications. Actually almost all vocabulary of SOAP is already well-known; if you have any idea about how XML works and looks understanding SOAP shouldn’t cause you any trouble.

Because SOAP is based on XML it makes it a platform and language independent technology. XML can be generated easily, therefore anyone and anything can participate in a conversation, with a relatively low barrier to entry. Also as a platform independent technology SOAP decouples the encoding and communications protocol from the runtime environment. A Web service can receive a SOAP payload from a remote service, and the platform details of the source are entirely irrelevant [25].

One thing that’s most important in SOAP, and makes it so useful and unique is that SOAP uses standard internet HTTP protocol. Thanks to using HTTP there are no problems with firewalls when using SOAP. This cannot be said about objects using DCOM [27] or CORBA (which, unlike HTTP, often do not use a standardized TCP/IP port number).

Also a huge advantage over other ways of communication between for example components is the ease of use of SOAP. Because SOAP doesn’t deal with certain ancillary protocols like DCOM or CORBA (both of them work with dynamically chosen ports; CORBA is much bigger than SOAP and has a part that is responsible for the same functions which SOAP takes care of called IIOP – Internet Inter-Orb Protocol [21]) it makes it easier to use and understand. SOAP takes care about important aspects of remote object systems and doesn’t deal with less important ones.

SOAP is a protocol which can be used to exchange information in a decentralized and distributed environment. It is useful in exchanging data in systems which are geographically remote from each other and use networks such as Internet to connect. Thanks to being an independent transport protocol SOAP can be used in combination with a variety of protocols like
DCOM or CORBA. It is also quite important to note that SOAP, as well as XML, is vendor neutral.

### 3.3.2. Reasons not to use SOAP

SOAP also has some disadvantages. The most important one is that SOAP specification contains no mention of security facilities. Because SOAP uses HTTP as a communication protocol it is possible to attack computer systems by using SOAP [26]. Thanks to HTTP, messages sent by SOAP are not checked when passing firewall. So, there’s no guarantee of safety when using SOAP. Because firewalls let the SOAP message go through without any control SOAP might be used for example to break into system or infect it with viruses (the author is not aware of any existing practical demonstrations of such a security exploit, but the theoretical possibility remains).

Also SOAP specification does not specify a default encoding for the message body. By using encodingStyle attribute a custom coding of the messages or individual parts in it can be defined. The specification defines a default coding, but its use is not compulsory.

SOAP was created to deal with plain text so to send a file by SOAP you have to serialize it (encode a data structure as a sequence of bytes). In this case distributed garbage collection [29] has no meaning. Also clients using SOAP cannot hold any stateful references to remote objects (client holds only the name of object, not its instance).

### 3.4. Example

Simple example - two SOAP messages are presented here, a request and an answer.

```xml
<?xml version="1.0"?>
<soap:Envelope xmlns:soap=
  "http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:m="http://www.calendar.org">
    <m:GetDate>
      <m:Date>Christmas</m:Date>
    </m:GetDate>
  </soap:Body>
</soap:Envelope>
```

Listing 2. An example of request sent using SOAP.

The request from List.2 asks http://www.calendar.org for a date of Christmas.

```xml
<?xml version="1.0"?><soap:Envelope xmlns:soap=
  "http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:m="http://www.calendar.org">
    <m:GetDateResponse>
      <m:Date>26.12</m:Date>
    </m:GetDateResponse>
  </soap:Body>
</soap:Envelope>
```

Listing 3. An answer for request from list.2.

Listing 3 shows an answer for request presented in List.2. The place where GetDate was earlier has been replaced by GetDateResponse. Also as an answer ‘Christmas’ was replaced by ‘26.12’.

### 4. XML-based composition languages

In this section XML-based composition languages is presented. There are many of them in use or in the development stage, but this section concentrates on XUL and XAML. Although XAML is still in the development phase, both of them are becoming more popular and are a part of success stories in Software Engineering (Mozilla Firefox, XAMLON [30]).

#### 4.1. XUL

##### 4.1.1. What is it?

XUL (pronounced “zool”) stands for the XML User Interface Language [31]. It is a language created to allow creating rich dynamic user interfaces. It is a portable technology which is available to many different platforms such as MS Windows, MacOS, Linux and Unix flavours [32].

XUL includes many existing standards like XPath, XSLT, and DOM functions to manipulate user interface. XUL is a really powerful tool – for example: the user interface in Mozilla WWW browser applications [33] has been written in XUL.

If a programmer will find all built-in tags and functions not suitable enough he can use XBL (Extensible Bindings Language [34]), which is related to XUL. It allows you to create your own stuff like custom tags for XUL.
All of applications written in XUL may be opened directly from a remote Web site as well as they may be downloaded by the user and installed. Mozilla has created a technology called XPInstall [35] which allows an application to be placed on a remote server and installed on your system just by clicking few times a mouse button. There is no need to search for the downloaded file. The huge benefit of such a solution is that application can access files on your hard-drive, write and read them so all your preferences may be saved easily. Every time when this application is opened it knows exactly preferences of the user and everything what it needs about the system configuration.

Although this main function of XUL is to create network applications it can also be used to create standalone applications. The Gecko engine (Mozilla’s layout engine – an embeddable component designed to display user interfaces based on open standards language [36, 37]) may be used in this circumstances, but an existing web browser might be enough – so the application will start as a part of it.

One of the most important advantages of Component-based Software Engineering – the possibility of creating extensions for existing applications – is a huge advantage of XUL. If you’ve used Mozilla Firefox or any other application written in XUL you may know exactly what this means. At present there are many more than 100 extensions available for Mozilla Firefox (probably the most successful one is the extension which blocks adverts [38, 39] – nowadays available in many versions from different authors), and practically everyone can create new ones. The function that is responsible for this in XUL is called an overlay [40]. It was created to allow a third party to create extensions for a browser like toolbars, different menus. But now every XUL application uses it, so for example Mozilla Thunderbird – an email client – also has a number of extensions available.

4.1.2. History

XUL has its beginning in 1998 when Netscape Communications under pressure of the open source community announced that the Netscape Communicator source code would be free (March 31, 1998) [37]. The next step was the creation of Mozilla.org. This organization concentrated many programmers working across the Internet. It made it possible to develop a new Internet browser and connected technologies.

One of these technologies was XUL. It was cut from Mozilla.org’s Cross-Platform Front-End project. The mission was to create environment in which GUIs code could be constructed from XML and JavaScript. It was created as a XML-based language.

All of the components of new Netscape/Mozilla browser were created in XUL. As the language has evolved, different technologies have been added (ex. CSS – Cascade Style Sheets [41], XBL – XML Binding Language [34]).

4.1.3. Why XUL?

Thanks to XUL, creating complicated applications without special tools is possible. As a technology it is a part of Mozilla web browser and related applications and also we can find it in Gecko.

The main reason why XUL has been designed was to create user interface of Mozilla applications (web browser, mail client and page editor). However XUL might be also useful whenever you would like to use a web application. XUL allows you to retrieve documents from the network and require a richer user interface. CSS (Cascading Style Sheet) is also served by XUL, as well as JavaScript – these are main similarities to HTML. XUL, just like HTML allows you to create an interface using a markup language. But XUL offers much more than HTML. It provides a rich set of user interface widgets for creating menus, toolbars, tabbed panels and hierarchical trees. Also XUL allows you to connect with web services.

So why use XUL? All these things like being able to create for example menus just by using XUL makes it easier to use XUL than HTML. Also it eliminates the necessity of looking for a third party component like it is with JavaScript and HTML – for example popup menus – all these things may be done much more easily by using XUL because all of them are built-in. It is even possible to use patterns for design to make an application look and feel like those on the user’s native platform like Windows XP and MacOS X. But if the programmer is not interested in this he can use standard CSS to create his own look.

4.1.4. Pros and cons

This description of pros and cons is mostly based on the mozilla.org site, references [6, 32, 37] and my own experiences with this technology. Actually Mozilla developed it and is the first well-known user of XUL in this scale. Just one view of any Mozilla application may make you realize that this technology is a big success. But it has also weak points.

The advantages of XUL [42]:

- Scriptable – Now there’s no need to interfere in logic and event-handling code if we want to change UI. Imagine
A survey on the usage of XML in Component-based Software Engineering

situation when client has ordered an application and when it’s finished he doesn’t like UI of it. Earlier a big part of source code would have to be changed, now only the UI code can be changed.

- Cross-platform – All files are portable across many platforms (Windows, Mac, Unix). And it is possible to create applications that will feel and look as a native user OS, as well as using the same look for all of them.
- Customizable – All components may be configured either statically or dynamically.
- Ease of use – Everyone who had anything in common with XML or web technologies can easily start programming XUL.

But, because no technology is perfect XUL has also some disadvantages.

- No modelling/development tool currently available.
- Because XUL was developed by Mozilla, all of XUL applications write now can be opened only using Mozilla and Chrome, part of Mozilla platform that deals with XUL files.

4.1.5. Example

An example of a simple calculator UI written in XUL is presented in Appendix 1.

4.2. XAML

4.2.1. What is it?

XAML (pronounced “zamel”) – EXtensible Application Markup Language is a new Microsoft standard being developed especially for Longhorn [11] – the new Microsoft OS which will have its premiere in 2006. Right now there are some tools with which you can use XAML while programming in .NET – XAMLON [30] developed by XAMLON Inc. and MyXaml [43].

Diagram 1. The two-tier model (UI – User Interface).

With XAML it is possible to develop application in two-tier model (Diagram 1). It allows developers to change the user interface without rewriting the logic and event-handling code. XAML is based on XML. It is used in similar ways to HTML.

In general XAML is a programming language with which it is possible to create user interfaces by using .NET objects.

By combining XAML with the next generation of Windows graphics kernel we receive a really powerful tool in MS Windows environment. It allows creating desktop applications, web applications, animations, print documents and much more [44].

According to Chris Anderson (one of developers of XAML) [45] XAML has four general rules:

- Tag names are class names. The CLR (Common Language Runtime [46]) namespace and assembly is discovered by an XML namespace mapping scheme (more later).
- XML Elements are mapped to CLR properties or events
- Sub-elements of a tag can either be other class instances or ‘compound’ properties (of the form Type.PropertyName)
- The root tag of a markup file is the base type of the class the markup file defines.

4.2.2. Why XAML?

The main idea was to create new user interface language which would be the basic language used to program in new Microsoft platform. So all user interfaces in Longhorn are created in XAML.

But why did Microsoft have to create a new markup language? One of the reasons is functionality [7, 44]. Because Avalon (user interface framework for WinFX [47]) is quite complex and uses many sophisticated tools which are not available in HTML (scaling, rotation) it was easier to create new standard than improve already existing ones. Because XAML applications will run directly on Windows, HTML wasn’t an option here. The biggest difference between XAML and any other markup language is that XAML is designed to integrate directly with WinFX. There is a pretty close relationship between elements in XAML file, and objects at runtime. In a XAML file each element will create a corresponding object at a runtime.

Summarising, XAML is a powerful tool. You may change user interface of application without changing the logic and event-handling code. It might be really useful when writing custom software. If client would not accept the user interface you may change it just in few seconds, don’t have to prepare many versions of the same
application. Most of XML-based languages can do many incredible things. With XAML it might be possible in future to create XAML-based markup language. You can also use for example XSLT to transform normal XML file to XAML.

Some of programmers even say that when Longhorn is released, XAML is going to penetrate all facets of software development [44].

4.2.3. Pros and cons

Comparing XAML to any other “normal” programming language we can find a few advantages of using XAML as well as disadvantages. Because official Microsoft XAML implementation is not finished yet programmers are using XAML implemented by other vendors, but no one can say if this versions are the one that will be in the final MS’s implementation. So it is hard to exactly show pros and cons of this technology.

First of all, the thing that you see right after starting programming in this scripting language is the length of source code. For example we can compare source code of a form with one button written in XAML and C#.

```
<Button Background="LightSeaGreen"FontSize="24pt">Calculate</Button>
```

Listing 4. XAML source code.

```
Button b = new Button();
b.Background = Brushes.LightSeaGreen;
b.FontSize = newFontSize(24,FontSizeType.Point);
b.Content = "Calculate";
```

Listing 5. C# source code.

As it can be seen when comparing source code from Listing 4 and Listing 5 XAML source code is much shorter than the equivalent C# source code. When talking about implementing anything, this might be one of the biggest advantages of a new language. But because it is a scripting language, like all of the XML-based languages, it might be hard to get used to this new way of programming user interfaces.

Microsoft had a vision. If everything will work like they wanted to and programmers will get used to scripting way of programming without bigger problems, it should be the future of programming. Of course it is easier for people who already know such languages like XML, HTTP or PHP [48].

Because XAML is an XML-based language and may be used to implement either web or desktop applications there shouldn’t be any difference anymore between programming these kinds of applications. In this case Windows is the browser kernel and Internet Explorer is just a shell. It makes developers lives easier. They will not have to learn many different programming languages to be able to create different kinds of applications.

4.2.4. Example

An example of a simple calculator UI written in XAML is presented in Appendix 2.

5. XML functional languages

In this section XML functional languages are presented. But first of all the XPath technology has to be introduced because it is the basis everyone should know if one wants to talk about XML functional languages. The section begins with a short description of XPath.

5.1. XPath

5.1.1. What is it?

The XPath language is really important when talking about XQuery and XSLT – because it’s the main part of these technologies.

XPath is a language created for finding information in XML documents. It has built-in standard functions for string and numeric values, Boolean values and other types of data like date and time comparison. It uses path expressions to navigate in XML documents which make it quite similar to using traditional computer file system. Also XPath is the syntax for defining parts of an XML document. XPath became a W3C standard in 1999 [3].

The most important things in XPath code are nodes. XPath distinguishes 7 types of nodes (element, attribute, text, namespace, processing-instruction, comment, and document nodes) [49].

5.1.2. Example

A simple XML file with data about jazz albums:

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<mystore>
  <music category="Jazz">
    <artist>Andre Bush</artist>
    <title lang="en">Art of the Duo</title>
    <year>2002</year>
    <price>100.00</price>
  </music>
  <music category="Jazz">
    <artist>Michal Urbaniak</artist>
    <title lang="en">Jazz Legends</title>
    <year>1993</year>
    <price>80.00</price>
  </music>
</mystore>
```
The XPath expression

```
/mstore/artist/text()
```

will return:

Andre Bush
Michal Urbaniak
Butch Thompson

Here will be returned everything about first album:

```
/mstore/music[0]
```

will return:

```xml
<music category="Jazz">
  <artist>Andre Bush</artist>
  <title lang="en">Art of the Duo</title>
  <year>2002</year>
  <price>100.00</price>
</music>
```

As you can see using XPath is really simple and is helpful when extracting data from XML files.

5.2. XSLT

5.2.1. What is it?

The history of XSLT (Transformation Extensible Stylesheet Language) started with defining by W3C XSL (Extensible Stylesheet Language) [50]. There was a need for creating some kind of equivalent of the CSS language used with HTML but for XML. Because in HTML all of the tags are predefined it makes it easy for browsers to understand and support them. But in XML we use our own custom tags. So XSL became some kind of translator used by the browser to understand XML document.

XSLT is the biggest and most important part of XSL. It stands for XSL transformations - XSLT (EXTensible Stylesheet Language Transformations).

XSLT transforms one XML document into another XML document and uses XPath to navigate in XML documents. Usually XML documents are transformed into (X)HTML documents which can be viewable for web browsers. Thanks to XSLT editing documents is possible – adding/removing of unnecessary parts and many other things. When XML document is being transformed XSLT checks which parts of it are needed in the new one by using XPath.

In Component-based Software Engineering this language is getting more popular with every day. Because components are communicating by XML-based protocols like SOAP XSLT is useful in transforming messages sent by components.

5.2.2. Pros and cons

What are the advantages of XSLT? Well, first of all XSLT allows you to manipulate, select and transform the content of XML document [51]. This is very important when we are talking about Component Based Software Engineering. Most of messages sent between components and applications like web services are based on XML. So if there’s a need to find something it is good to have such a tool as XSLT. Thanks to this tool nobody has to implement this by himself.

Showing XML data is not a problem. One of main reasons why XSLT has been created was to transform XML data into HTML files which can be opened by any web browser. By using different XSLT templates the same data may be shown in many different ways.

XSLT gives you an opportunity to transform any XML file into other format (HTML, XHTML [52] etc.). Custom templates can be created, which allows transforming data into many different formats, not just the commonly-used or popular ones.

Because this section is called pros and cons some disadvantages are shown in the following text.

First of all scripts written in XSLT can grow really fast. Consequently, such large amounts of source code can require substantial amounts of processing time. Even now, when almost every computer is really powerful computation time may be a big problem when it comes to running complicated and big scripts, especially when it has to work with big amounts of data. Of course it means also that debugging is hard and takes lots of time.

Next quite important disadvantage is the complexity of XSLT. It can make maintaining XSLT stylesheets an interesting job all its own. Also according to Roger Costello, the size of the specification (printed pages) of new XSLT – ver. 2.0 is 564% larger than XSLT 1.0 [46]. Because standards are becoming more complex, and there are many more tools included so we should ask one question. Who need all of this stuff? Probably as Kendell Grant Clark [53] says only those who advocated for them in the first place. Actually, almost all standards, when new version is available, are bigger than earlier. So it’s scary to think how it will look in future.
5.2.3. Example

First of all an XML file must be defined. For this purpose the same file as in example for XPath will be used.

```xml
<?xml version="1.0" encoding="ISO-8859-1"?><mstore>
  <music category="Jazz">
    <artist>Andre Bush</artist>
    <title lang="en">Art of the Duo</title>
    <year>2002</year>
    <price>100.00</price>
  </music>
  <music category="Jazz">
    <artist>Michal Urbaniak</artist>
    <title lang="en">Jazz Legends</title>
    <year>1993</year>
    <price>80.00</price>
  </music>
  <music category="Jazz">
    <artist>Butch Thompson</artist>
    <title lang="en">Yulestride</title>
    <year>1994</year>
    <price>90.00</price>
  </music>
</mstore>
```

This is a simple XSLT script

```xml
<?xml version="1.0" encoding="ISO-8859-1"?><xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body>
        <table border="1">
          <tr><th>Title</th><th>Artist</th><th>Year</th><th>Price</th></tr>
          <xsl:for-each select="mstore/music">
            <tr>
              <td><xsl:value-of select="artist"/></td>
              <td><xsl:value-of select="title"/></td>
              <td><xsl:value-of select="year"/></td>
              <td><xsl:value-of select="price"/></td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

This is how the result will look like. In this case it is a HTML file but depending on what is needed you can make it save all the data in any format you want.

5.3. XQuery

5.3.1. What is it?

XQuery is a language developed by W3C. It was designed to query collections of XML data. XPath is an important part of this language. It works not only with XML files, but also other data whose structure is similar to XML files (databases). The semantics of XQuery is similar to SQL. Both of them are query languages but SQL queries relational databases while XQuery queries XML databases and files. It provides a mechanism to pull XML content from multiple sources and dynamically generate new content using a declarative language [54].

XQuery, just like XSLT, can be used to transform XML data to other formats. According to the opinion of many programmers XQuery is less powerful than XSLT, but it is easier to learn it and easier for vendors to optimize it [50].

5.3.2. Pros and cons

Lately there is a big discussion taking place on Internet about XQuery. The main cause is that Microsoft announced that they are dropping the idea of XQuery in the next .NET release (.NET 2.0, code name “Whidbey” [55]). In fact they also declared that probably they will never again support XQuery in .NET platforms [56].

Many companies are fighting to change it. Stylus, for example, has started the project called “XQuery for all” [57] which aim is to convince
Microsoft to continue implementing of XQuery in future projects. Because many of these companies have spent much time on developing XQuery they are disappointed by the attitude of Microsoft.

XQuery as other XML functional languages is based on popularity of XML. It was designed to query XML data. So because it can pull information from many places at once less storage data is required which makes this language a quite useful tool. It also has few types of data defined that makes it work faster.

5.3.3. Example

The same XML file as in the previous examples is used here.

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<mstore>
  <music category="Jazz">
    <artist>Andre Bush</artist>
    <title lang="en">Art of the Duo</title>
    <year>2002</year>
    <price>100.00</price>
  </music>
  <music category="Jazz">
    <artist>Michal Urbaniak</artist>
    <title lang="en">Jazz Legends</title>
    <year>1993</year>
    <price>80.00</price>
  </music>
  <music category="Jazz">
    <artist>Butch Thompson</artist>
    <title lang="en">Yulestride</title>
    <year>1994</year>
    <price>90.0</price>
  </music>
</mstore>
```

Below is a simple XQuery code which will return titles of albums recorded by Michal Urbaniak.

```xml
<results>
  { 
    let $a := doc("mstore.xml")
    for $artist in distinct-values($a)
      where $artist/artist = "Michal Urbaniak"
    order by $artist
    return 
      <result>
        { 
          for $b in doc("mstore.xml")
            /mstore/music
            where some $ba in $b
            satisfies ($ba/artist = "Michal Urbaniak")
          return $b/title
        }
      }
  </result>
</results>
```

The answer looks like this:

```xml
<results>
  <result>
    <title lang="en">Jazz Legends</title>
  </result>
</results>
```

6. Discussion and comparison

There are few things to discuss about XML and all its related languages and tools in Software Engineering. First of all, we have to ask if there is still a place for XML in Software Engineering. As we can see, the amount and variety of these tools is significant. Some of these concepts are available without cost for everyone while for others you have to pay. We have to realize that not all of these techniques gained a big part of the market but it may be caused by short time they are available. Some of them, like XAML, are even not finished yet.

According to all earlier presented techniques we can definitely say that XML is becoming more present in CBSE and SE in general with every day. But how do these tools use it? Some of them use XML as a programming language (XAML, XUL), others are tools which work with XML data (XQuery, XSLT) while SOAP is a protocol using XML as a base of its message structure.

When talking about scripting languages (XUL, XAML) we find some unanswered questions. The first, and probably most important, is if any of these languages will become a widely used standard. XAML probably has bigger chances to do it because most of computers use Windows. So XAML as a language defined for the new Windows OS gains a “pole position” when talking about MS’s environment. Even though XUL can run only on Mozilla applications, it may still be an important opponent for XAML because Mozilla applications are available in versions for different operating systems. And what about components? Both XAML and XUL make it possible to reuse components. XAML works generally with dynamic link libraries (DLLs) while XUL uses JavaScript [58] packages. However both of them are continuously developed so we cannot say if it’s the only way you can reuse components in these technologies.

XML functional languages are also a good subject for discussion especially according to recent Microsoft announcements about abandoning the XQuery implementation in the next .NET platform [56]. So is there a need for having such a language as XQuery? It is the only query language for XML data, which is what makes it really useful, but in this case why would MS abandon its implementation in “Whidbey” [55]? They say that the reason is that there is no W3C recommendation for XQuery yet. We have to believe them and hope that as soon as XQuery gets the recommendation it will be implemented.
There is also XSLT which is used to transform XML files. Is it really useful? It was mainly created to work with the Web, to present XML files in browsers-comprehensible formats. But as with many other things people found it useful also in different tasks. It may be used to translate messages sent between components. So it is a nice tool which may be useful not only to transform and show data in web browsers.

When talking about SOAP we have to remind the reader that when this standard was created it was the attempt to have the first standardized way to exchange objects via a network such as Internet using already existing protocols (HTTP). Did it work out? The answer is ‘yes’; it’s been few years since SOAP was officially defined and it is still used and probably will gain more popularity in approaching years. It may have a few disadvantages but the ease of use of SOAP puts all competing technologies in the shadow.

Finally, one question arises. Does XML or any XML-based technology have any impact on CBSE? On one side we have XML-based composition languages, on the other… nothing. We have also SOAP – the only net protocol using HTTP based on XML, used in many systems all over the world.

7. Future

It’s hard to describe the future of XML and it’s much harder to try to describe future of XML in CBSE. XML as a technology is present in many branches of software industry and in each of its uses are evolving all the time. For each of them there are different tools based on XML.

When talking about using XML as a standard to store and present data it must be noticed that it’s almost impossible to make vendors finally use it as the only way of storing data [59]. Maybe they will use self-created standards based on XML, but probably it will still be needed to export/import data and convert it between applications made by different companies.

According to Tom Bradford [60] future of XML is quite difficult to describe especially because of W3C. W3C already has developed few standards based on XML (XPath, XQuery) and with new versions they are becoming more complex and complicated in use. It might be possible that in few years some of these standards will be responsible for the same kinds of tasks. So as Tom Bradford noticed, it’s possible that by the year 2010 XML will be a legacy data format, having been replaced by a new ‘simple’ solution. And this is not a single opinion – for example take a look at Norman Walsh’s [61] or Liam Quin’s [62] blog.

CBSE is probably the environment where XML will gain much more appreciation. The systems which are developed are becoming bigger, more complex and some of them are using already existing components. XML should become more popular – a good example for reuse are web services which are a powerful tool in exchanging data over computer networks [63]. It already has become a very popular tool.

With development of new XML-based functional languages such as XAML or XUL, XML is going to be much more involved in programming than before. Probably XML-based languages will become the most important programming languages in future.

There are many projects taking place now where the main aim is to create languages and tools based on XML to implement component-based applications. One of many groups working in this field is Dr Uwe Assmann’s group [64] which is working on few projects based on XML.

There are also attempts to standardize binary XML but there must be defined rules how to do it before starting [65]. The main problem here is that no single format will cover everyone’s needs. So in future there might appear a situation where binary XML will be available in few versions.

Summarizing, XML should become more popular in next few years. It is hard to say which branch of software industry will gain most by using XML, but probably all of them will get something.

8. Conclusion

Already XML has changed some parts of SE. Nowadays it is hard to think about programming web or Component-based applications without possibility of using XML.

So has XML impact in any way CBSE? Yes, and it has bigger influence on CBSE with each day. Most of the languages based on XML (XAML, XUL) are going to be a big step in (CB)SE when finally developed. XAML will definitely change the way of programming (there will be almost no difference between web and desktop applications), as well as XUL if only will become independent from Mozilla browser.

Unquestionably XML is a significant technology already. The only thing that can be done now is hoping that W3C will not improve XML by complicating it and making it more complex.

This paper provides an overview of usage of XML in CBSE. It is done in not very detailed way, but provides a good overview of XML-based technologies that are used in CBSE. Better access to the library sources would increase the level of details in sections about the XML functional languages and XUL. XAML as a relatively new technology is not well documented yet, so it wouldn’t have big influence on section about
XAML. A good idea when investigating this subject and gain some more information (especially about XAML) is to contact developers involved in creating XML-based technologies. If I could start this study again I would certainly contact more of them.

9. References

[7]. Component Software: Beyond Object-Oriented Programming” C. Szyperski – 14th April
[12]. Presentation of XML, [http://www.ohsu.edu/library/staff/zeigen/xml/present/sld005.htm - 19th May
[13]. SAXon project homepage [http://saxon.sourceforge.net/ - 19th May
[15]. ezXML project homepage [http://ezxml.sourceforge.net/ - 19th May
[17]. W3C school: SOAP, [http://www.w3schools.com/soap - 16th April
[19]. CORBA homepage, [http://www.corba.org/ - 19th May
[20]. SOAP history, Don Box [http://webservices.xml.com/pub/a/ws/2001/04/04/soap.html - 18th April
[23]. Introduction to XML and SOAP, [http://www.ecnc.net/papers/xml200/slides - 22nd April
[25]. Garbage collection definition, [http://www.memorymanagement.org/glossary/g.html#garbage_collection - 13th May
[26]. Xulm Inc. homepage, [http://www.xaml.com - 4th May
[29]. Mozilla.org homepage, [http://www.mozilla.org - 19th May
[30]. XBL homepage, [http://www.w3.org/TR/xbl/ - 19th May
[31]. XUL Planet, [http://xulplanet.com - 19th May
[33]. Gecko project homepage, [http://www.mozilla.org/Gecko/Home_Page - 19th May
[34]. XUL Planet definition of an overlay [http://www.xulplanet.com/definitions/ref_overlay.html - 14th May
[35]. CSS homepage, [http://www.w3.org/Style/CSS/ - 19th May
[37]. MyXAML homepage, [http://www.myxml.com - 26th May
[38]. XAML: Turn .NET into a Browser, Gerald Bauer, [http://xml.sourceforge.net/talk/dotnet-dec-2004/slides.html#xml/4 - 9th May
[39]. Chris Anderson’s blog, [http://www.simplegeek.com - 12th May
A survey on the usage of XML in Component-based Software Engineering

[49]. W3C school – Xpath http://www.w3schools.com/xpath – 7th May
[51]. W3C schools homepage about XSL, http://www.w3schools.com/xsl - 12th May
[53]. How (not) to grow a technology, Kendell Grant Clark, http://www.xml.com/pub/a/2003/06/25/deviant.html - 11th May
[60]. Tom Bradford’s blog, http://www.tbradford.org/FutureOfXML.txt - 14th May
[64]. Uwe Assman’s Research Group homepage, http://www.ida.liu.se/~uweas/research.html-14th May
Appendix 1.
A screenshot and a source code of a simple calculator UI written in XUL.

```xml
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin" type="text/css"?>
<!DOCTYPE window>
>window title="Simple calculator UI" xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">

<menubar id="mainMenu">
  <menuitem id="Exit" label="Exit" command="Exit" />
</menubar>

<vbox id="main">
  <hbox>
    <textbox id="display" style="align: right; color: yellow; background: black; font: 22 bold monospace; width:378px" />
  </hbox>

  <hbox>
    <button label="Backspace" style="width: 119px; color:red" />
    <button label="C" style="width: 119px; color:red" />
    <button label="CE" style="width: 119px; color:red" />
  </hbox>

  <hbox>
    <button label="7" style="width: 50px; color:blue" />
    <button label="8" style="width: 50px; color:blue" />
    <button label="9" style="width: 50px; color:blue" />
    <button label="/" style="width: 50px; color:red" />
    <button label="sqrt" style="width: 50px; color:blue" />
  </hbox>

  <hbox>
    <button label="4" style="width: 50px; color:blue" />
    <button label="5" style="width: 50px; color:blue" />
    <button label="6" style="width: 50px; color:blue" />
    <button label="*" style="width: 50px; color:red" />
    <button label="%" style="width: 50px; color:blue" />
  </hbox>

  <hbox>
    <button label="1" style="width: 50px; color:blue" />
    <button label="2" style="width: 50px; color:blue" />
    <button label="3" style="width: 50px; color:blue" />
    <button label="-" style="width: 50px; color:red" />
    <button label="1/x" style="width: 50px; color:blue" />
  </hbox>
</vbox>
```

15
A survey on the usage of XML in Component-based Software Engineering
Appendix 2.
A screenshot and a source code of simple calculator UI written in XAML (MyXAML, executed by MyXAMLLoader).

<?xml version="1.0" encoding="utf-8"?>
<UI xmlns="System.Windows.Forms, System.Windows.Forms, Version=1.0.5000.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
xmlns:def="Definition">
<Form Name='AppMainForm' Text="Simple calculator UI" Size="300,210" FrameBorderStyle="FixedSingle">
<Controls>
<TextBox Name="MainTextBox" Dock="Top" Height="25" Text="0.00" TextAlign="Right"/>
<Button Background="Blue" Text="Backspace" Location="0, 30" Width="90"/>
<Button Text="C" Location="100, 30" Width="90"/>
<Button Text="CE" Location="200, 30" Width="90"/>
<Button Text="7" Location="0, 60" Width="60"/>
<Button Text="8" Location="60, 60" Width="60"/>
<Button Text="9" Location="120, 60" Width="60"/>
<Button Text="/" Location="190, 60" Width="50"/>
<Button Text="sqrt" Location="240, 60" Width="50"/>
<Button Text="4" Location="0, 90" Width="60"/>
<Button Text="5" Location="60, 90" Width="60"/>
<Button Text="6" Location="120, 90" Width="60"/>
<Button Text="*" Location="190, 90" Width="50"/>
<Button Text="%" Location="240, 90" Width="50"/>
<Button Text="1" Location="0, 120" Width="60"/>
<Button Text="2" Location="60, 120" Width="60"/>
<Button Text="3" Location="120, 120" Width="60"/>
<Button Text="-" Location="190, 120" Width="50"/>
<Button Text="0" Location="0, 150" Width="60"/>
<Button Text="+/-" Location="60, 150" Width="60"/>
<Button Text="1/x" Location="120, 150" Width="60"/>
<Button Text="=" Location="190, 150" Width="50"/>
<Button Text="=" Location="240, 150" Width="50"/>
</Controls>
</Form>
</UI>