Building a Personalized Communication Platform using Open Standards

Nick van den Bleeken
<nick.van.den.bleeken@inventivegroup.com>

Abstract

Communicating with customers in the manner they prefer, with the information they find interesting, is getting ever more challenging with rise of mobile and social media. This paper will discuss how we have built a Personalized Multi-channel Communication Platform using open standards, what extensions to those standards were required and what challenges we faced creating the platform.

Table of Contents

1. Introduction 1
2. Platform Overview 2
3. XML Standards used 2
    3.1. SCXML 3
    3.2. XQuery and XQuery Update Facility 3
    3.3. XSLT 4
    3.4. XSL-FO 5
    3.5. XForms 6
4. Conclusion 6
References - XML 7
References - non XML 7
References - Implementation 7

1. Introduction

With the rise of mobile and social media (e.g.: Facebook, Twitter, LinkedIn, Pintrest), the number of channels over which to contact people keeps growing rapidly. More traditional output channels like print, email and sms keep their value, but depending on the addressee(s) preferences, the content and context, the best communication channel will vary. The importance of creating unique and engaging conversations increases the need for a full overview of all your communications (what is sent, when was it sent, was the communication delivered and/or opened by the addressee, did the addressee(s) interact with communication and how did they interact). Due to the great differences between these types of communication, the platform should be able to generate and manage high-volume batch, on-demand and interactive communications.

To build such a Personalized Communication Platform we used a lot of different open standards. We chose XML as the backbone of the platform because XML isn't limited to a fixed grammar; a lot of services return XML (if not, the data can easily be converted to XML and back); and there are rich query and transformation languages for XML. Additionally most open XML standards allow extensions, which are required to reach our goal. Most of those extensions are already scheduled for a future version of the standards or are being discussed in the appropriate working group.

This paper will discuss the standards that we used to build the platform, how they are combined to create a Personalized Multi-channel Communication Platform, what extensions to those standards were required, and what our challenges were building the platform. The platform is used today as a multi-channel communication platform that helps enterprises worldwide to improve their customer communications, leading to increased customer engagement and loyalty. This paper is absolutely not intended to be commercial in nature. We thought it would be interesting for people to see how technologies they work on for a large part of their life can all be combined to create an enterprise communication platform, and what the challenges were in using and combining these technologies.

2. Platform Overview

During the complete communication process a lot of decisions have to be made (e.g.: what channel to use, when to send the communication, fallback when the communication is unsuccessful, communication approval) based on the personal preference of the addressee, the available data and other context information. This orchestration is done using State Chart XML (SCXML) [SCXML-10].

The data used throughout the communication process is XML. The data can be pushed into the platform (folder drop, SOAP/REST endpoint), pulled by the platform (XQuery, SOAP/REST call), or a combination of both. If manual data entry is required we use XForms to collect the data. We are also using XSLT, in case different data sources need to merged, transformed or enriched.

For the output generation we are using XSLT+XSLFO. We embed SVG, Open Office Charts, and XForms in those formats when appropriate. Those templates are created using a WYSIWIG editor. The output generation part of the platform uses a lot of NON-XML open standards, because certain channels don't support XML formats (e.g.: AFP, PS, PDF, PCL, … for printing/press). An exception to this are 'interactive documents', for which we use XHTML+XForms, which can be used in both desktop and mobile environments. The current work in the Open Web Platform and the rise of mobile are going to make it possible to interact with documents in ways we never imagined. In the delivery part of the platform we don't use a lot of XML because services like Print, FTP, SMTP, Facebook, Twitter, and the Apple Push Notification Service don't use XML as their data/communication format. Most of the modern channels have a REST service, but they either encode all data as parameters or use JSON as a data format. There are exceptions like the Windows Push Notification Services, which use an XML grammar as a data format.

Tracking how people interact with your communication is a really important part of the platform. Did the addressee(s) open the communication, did they respond to it, did they share it? All this information is collected, tracked using XQuery Update Facility, and fed back into the SCXML instance handling the communication, if requested.

To be able to get a full overview of all your communications, we are tracking all individual steps of the communication process using XQuery and XQuery Update Facility. The User Interface to manage, consult and monitor the complete system is written in XForms in which we use XQuery to retrieve the data and XQuery Update Facility if modifications to the data are required (e.g.: approval, set channel/ communication on hold).

3. XML Standards used

In this section we will discuss for each standard:

- Why we have chosen that particular standard
- What extensions to the standard were needed to meet our needs
- What the challenges were using that standard
- The implementation of the standard (either off-the-shelf or custom-made)

3.1. SCXML

State Chart XML (SCXML) [SCXML-10], a general-purpose event-based state machine language that combines concepts from CCXML and Harel State Tables. It is used in the platform to define the communication process at a high level. In the state chart you can define for example the approval process, the channel(s) to use for the communication, fallback states used when the message couldn't be delivered, delivery schedules, ...

3.1.1. Extensions to SCXML

We registered one custom invoker to send a communication.

3.1.1.1. Send communication invoker

This invoker sends a communication over the specified channel (print, email, SMS, ...). You can optionally specify the delivery intervals (on which days and between which hours) and how long the message is valid for delivery. When the message is sent, can't be send, or times out events are send to the state machine, which can be used to trigger specific behavior in the state machine. Depending on the output channel extra events can be sent (e.g.: Delivery notification for SMS)

3.1.2. SCXML challenges

The state machines for for the communication process typically stay alive for multiple days or even weeks. The Apache Commons SCXML implementation by default keeps all running state machine in memory and does not have support for multiple servers handling the same state machines.

We have implemented a high available event dispatcher that supports persisting running state machine instances and clustering of multiple servers which automatically distributes the work over all servers. When the load of one server gets to high, the running state machine instances are automatically distributed over the other servers.

3.1.3. SCXML implementation

We are using the Apache commons SCXML implementation [apache-scxml].

3.2. XQuery and XQuery Update Facility

XQuery [XQUERY] is an XML query language, capable to extract and manipulate the information content of diverse data sources including structured and semi-structured documents, relational databases, and object repositories.

XQuery is used for the platform to retrieve the personalization data and obtain the data to represent in the management UI. With personalization data we mean the data that is used to create the communication, but also the data that is used to decide which communications to send, when and how.

In the platform we also need to be able to update that data. We have chosen XQuery Update Facility for this. XQuery Update Facility [XQUERYUPDATE-10] is an extension to XQuery, which adds support for making persistent changes to instances of the XQuery and XPath data model. It supports operations like inserting, deleting and changing nodes.

3.2.1. XQuery challenges

Customers prefer to store their data in a relatational database because they already have the infrastucture and expertise to manage the relational database. We developed an XQuery engine which supports a subset

of XQuery and store its data in a relational database. We store every element as a row in a table specific for every element type.

Working around database quirks is another challenge. An example of such a quirk is that an empty string on Oracle becomes NULL in the database. Since we consider a NULL value as the absence of the attribute on the element this poses a problem. We chose to prepend every string with a space character in oracle, but this of course complicates all operations on string in orcale because we have to counter the extra space character.

For performance reasons you want to map one XQuery expression to ideally one SQL query. The hierarchical structure of XML makes it really challenging to only use one SQL query, with the current feature set of XQuery that we support, we are able to map one XQuery expression to one SQL query.

It is an ongoing effort to support more of the XQuery specification. Mapping everything to SQL is hard (or maybe even impossible), some parts of the spec are difficult to implement without sacrificing performance in either the retrieval or insert/update process.

XQuery knowledge is not that common, but because XPath is used as a selection language in XQuery we have seen that the learning curve isn't as steep as we feared. Nevertheless we created a wizard to construct XQuery expressions for the most common use cases. This allows business users to create the queries.

3.2.2. XQuery implementation

The query engine and type of the data store is pluggable. Currently we have written our own XQuery engine which supports a subset of XML and store its data in a relational database. It is possible to use an off-the-shelf XML database and use its XQuery engine.

3.3. XSLT

XSLT [XSLT-20] is a language for transforming XML documents into other XML documents. We use it both for transform and merge personalisation data, and for conditional structural formatting of the contents of the communication.

3.3.1. Extensions to XSLT

Customers can plug-in their own custom function library containing business and project specific extension functions. Customers use this to abstract complex expressions and business logic. The functions in the custom function library become available in our XPath query builder, which simplifies the creation of XPath expressions.

It is a common practice to re-use XSLT+XSL-FO blocks in multiple XSLT+XSL-FO documents. There are use cases in which you don't know upfront which XSLT+XSL-FO blocks you want to re-use in specific XSLT+XSL-FO document, because they are selected by the input data (e.g.: Creating a contract based on standardized paragraphs. The number of paragraphs changes over time, and you don't know all the paragraphs while designing the contract template.). To facilitate this use case, we have created an extension element which returns the ouput of another XSLT+XSL-FO stylesheet, and supports passing in arbitrary parameters.

3.3.2. XSLT challenges

Memory consumption while processing large XML source documents is currently our main challenge. The introduction of the streaming mode in XSLT 3.0 is going to make life a bit easier for us. In order to use this new feature in the output generation part of the product we will need to make adjustments to

our WYSIWYG XSL-FO editor, and we are not yet completely sure how to integrate this in our product without introducing extra complexities for the document author.

3.3.3. XSLT implementation

We use Saxon PE [saxon-pe] as our XSLT engine.

3.4. XSL-FO

XSL Formatting Objects [XSLFO-20], or XSL-FO, is a markup language for XML document formatting designed for paged media with features like advanced page layouts with citations, cross-references, running headers/footers, and running totals. In combination with XSLT it allows powerfull conditional structural formatting.

When we generate output (PDF, PS, AFP, HTML5, ...) we always generate XSL-FO first, then format the content and finally render to the specific output format. We currently have 18 different output formats, ranging from mainstream print and press formats, over web and mobile to less common formats like ZPL and TCPL.

3.4.1. Extensions to XSL-FO

For creating more graphical and interactive documents, we mix-in other open standards like SVG, XForms and Open office charts. We've also added some extensions to support features like rounded corners on all block level elements and a barcode markup language to represent barcodes in XSL-FO.

Because most output formats support meta-data (e.g.: TLE's in AFP or PDF annotations) we also added an extension for meta-data. This extension was added before RDFa existed, which we would have used if it existed when we added the extension.

We also added extensions for specifying finishing features like duplex printing, input and output trays.

Some output specific features are so important that we created extensions that are only applicable for one output format. An example of this is the AFP overlays extension. It allows a user to specify the position and the AFP page or medium overlay (resource and library) to use when printing the page.

At XSLT time pagination information is not available, therfore we created an extension called 'Page Data'. It creates an XPath data model, while formatting the XSL-FO, on which XPath expressions can be excuted. This extension can for example be used to calculate the sum of all item prices on the current page.

3.4.2. XSL-FO challenges

When we started writing our own XSL-FO formatter in 2002 there were no other implementations we could use that met our performance, memory and quality requirements. We have to be able to generate small on-demand documents but also large batch documents with millions of pages. In those large batch documents we have to support nested tables of which the cells of the outer table span hunderds of pages for example. This required us to serialize parts of the XSL-FO tree because we can't keep the complete table row in memory while rendering it. There are of course a lot of other XSL-FO related challenges like resolving the correct page masters, keeps/orphans/widows, and cross-references.

We have chosen to separate the XSLT and XSL-FO formatting processes. This means a lot of the markup is repeated in the generated XSL-FO document, which introduces performance and memory challenges. To work around this challenge we are first optimizing the XSLT+XSL-FO template by removing superfluous XSL-FO attributes and afterwards replacing groups of attributes with attribute group references. The second optimization is optional because other XSL-FO formatters won't know this extension.

3.4.3. XSL-FO implementations

We created our own XSL-FO processor see Section 3.4.2, “XSL-FO challenges” for which we created our own processor.

3.5. XForms

XForms [XFORMS-20] is a cross device, host-language independent markup language for declarativly defining a data processing model of XML data and its User interface. It uses a model-view-controller approach. The model consists of one or more XForms models describing the data, constraints and calculations based upon that data, and submissions. The view describes what controls appear in the UI, how they are grouped together, and to what data they are bound.

We use XForms for interactive communications as wel as the management User Interface. For the interactive communications we first embed XForms in XSL-FO and convert it to XForms in XHTML. Until a couple of years ago we had a native client that accepted XSL-FO + XForms, but we discontinued it in favour of using only XHTML+XForms. Due to the recent changes in web browsers, driven by the HTML5 effort, there were no advantages anymore of having a native client.

3.5.1. Extensions to XForms

We used a lot of extensions which are standardized in XForms 1.1 or are in the XForms 2.0 specification which is about to go to last call. Examples of those are Attribute Value Templates on both XForms elements and on host langague elements, variables, multiple binds for the same MIP on the same node, XPath 2.0 expressions, and repeats over sequences of atomic values and nodes.

For creating the management UI and interactive communications we use a lot of custom components. We are using components that abstract a paging system to browse through large sets of data, auto complete controls, tab controls, ...

The custom components are using an enhanced version of XBL, but we never managed to standardize a custom control framework for XForms in the Forms WG.

3.5.2. XForms challenges

There are a lot of different web browsers which support different features and all have their own quirks, which makes creating an XForms processor that correctly works in all different browsers challenging. Recent versions of all browsers are implementing the standards better, which makes life easier for those, but we have to still support older versions too.

3.5.3. XForms implementations

We are using off-the-shelf XForms implementations. When we added XForms support to the platform we decided to use Chiba [chiba]. We contributed a lot back to the chiba project. For the management UI we are using Orbeon [orbeon], because they have better custom components support and they do some clever performance enhancements.

4. Conclusion

Using open standards where possible and discussing the rough edges and ‘missing’ features in the current standards with their standardization bodies, ensures that the platform is future proof and easy to integrate in/with other systems. We collaborated in the XSL and XForms working groups at the W3C and other non-

XML standards like AFP to ensure this. Working together with other people on those standards helped us a lot in better understanding the best practices and what is possible with those standards.

Other advantages of using open standards are:

- Documentation: there already is a lot of good documentation written on the internet and in books on how to use the standard.
- Re-use of off-the-shelf components: we have re-used a lot of open source implementations and commercial implementations. This decreased our time to market and allowed us to focus on the platform.
- Prevents vendor lockin: Our customers are free to replace one (or all) of the components with there own.

The platform is used today as a multi-channel communication platform that helps enterprises worldwide improve their customer communications, leading to increased customer engagement and loyalty.

References - XML

[SCXML-10] http://www.w3.org/TR/scxml/ .

[XFORMS-20] http://www.w3.org/TR/xforms20/ .

[XQUERY] http://www.w3.org/TR/xquery/ .

[XQUERYUPDATE-10] http://www.w3.org/TR/xquery-update-10/ .

[XSLFO-20] http://www.w3.org/TR/xslfo20/ .

[XSLT-20] http://www.w3.org/TR/xslt20/ .

References - non XML

[AFPC-0004-08] http://www.afpcinc.org/site/assets/files/1120/modca08.pdf .

[JSON] http://www.json.org/ .

[PS] http://www.adobe.com/products/postscript/pdfs/PLRM.pdf .

[ISO 32000-1] http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=51502 .

References - Implementation

[apache-scxml] http://commons.apache.org/scxml/ .

[chiba] http://chiba.sourceforge.net/ .

[orbeon] http://www.orbeon.com/ .

[saxon-pe] http://www.saxonica.com/ .