A Web Compatible Dep

Maurice Martens, CentERdata, Netherlands

1. Abstract

The Survey of Health Ageing and Retirement in Europe (SHARE) is currently in its development phase for its seventh wave. This survey uses a centrally developed source questionnaire and using an online translation management environment this source version extended to 41 country/language versions.

In SHARE wave 7 the sample was divided into subsamples who are assigned two largely different questionnaires. The larger sample is assigned to a life history calendar questionnaire (like was done in wave3), the respondents who already did a life history calendar were assigned to do a regular interview. The SHARE wave 3 history calendar was developed in VB6. After some trails, it was decided that reusing this instrument was not wise. Some alternative approaches were reviewed, including using Blaise 5, but ultimately this redevelopment was done in Java, using a browser component providing the freedom to use web techniques combined with calls to the Blaise 4.8 API.

This Java/browser approach gives us the possibility to implement web techniques in CAPI. This allows for use of jQuery, and JavaScript code for CAPI. It allows for use of interface design using style sheets JavaScript solutions developed for CAWI mode can be reused in CAPI mode. Using this technique we were able to rapidly implement more advanced features like a history calendar, autosuggest lists, and word-recall list in a short time without major changes to the Blaise questionnaire.
This paper discusses the route that led us to this architecture and will showcase some exiting implementations, within the context of the SHARE multi-mode, multi lingual, multi-questionnaire panel study.

2. Introduction

CentERdata has been developing software for the Survey of Health Aging and Retirement in Europe (SHARE) since its first wave in 2004. The SHARE project is a multidisciplinary and cross-national panel database of micro data on health, socio-economic status and social and family networks of individuals aged 50 or over. Currently its seventh wave is under development, which will be fielded in 28 countries, in 28 languages, for which we need to generate 41 local versions. All these versions are derived from one source Blaise questionnaire which is developed in English. A compiled version of this questionnaire is uploaded to a central online database. From this, we identify and import translatable elements. These translatable items are presented in a web-environment for translation, TMT (Translation Management Tool). This tool and it successor LMU, Language Management Utility (See: Managing Translations for Blaise Questionnaires, Martens at al., 2009 and Blaise Translation Challenges: Versioning, Multimode and Exporting, Martens, 2012) allow for a quick flexible decentralized translation process of Blaise questionnaires, hiding programming code from translators.

SHARE interviews are done mainly via face to face interviewing. SHARE uses an ex ante harmonized model. All data which is collected will be collected using the same data model. Each wave fieldwork is done at the same time for all participating countries, the questionnaire is designed once, translations are done centrally via the web system, and country specific versions are generated centrally and distributed to our fieldwork agencies. On a bi-weekly schedule, data collected so far is transferred and collected at a central location. This centralized design has the advantage that the survey can be managed very well and has a quick turnaround, it however needs the support of many tailor-made software solutions.

To support the SHARE survey, CentERdata has developed a rich environment to support the flow of data, to help the interviewer identify respondents and to launch the questionnaires. In chapter 2 this architecture will be discussed.

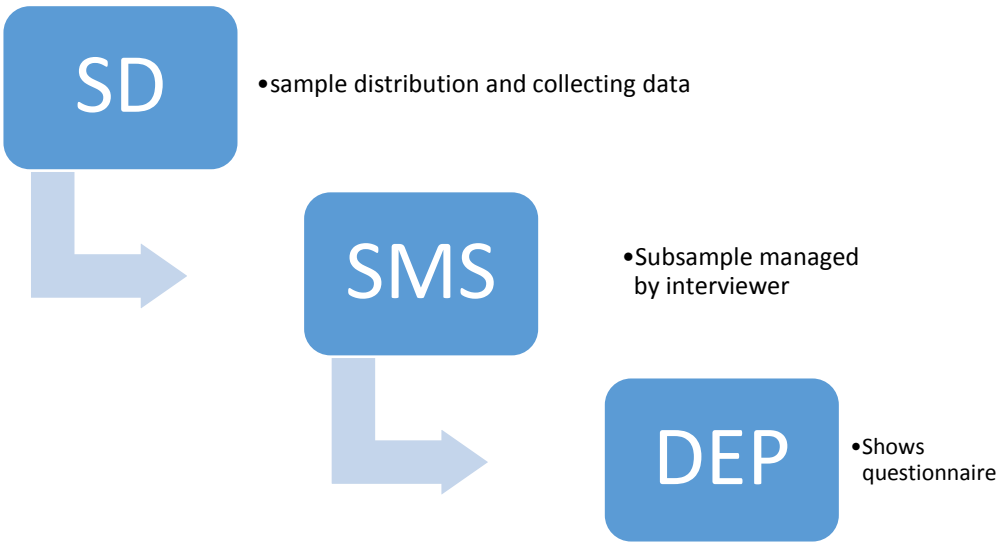
In this latest wave, we needed to present a Life History calendar to part of the sample. The Visual Basic 6 software that was developed in 2006 to support the Life History calendar in wave3 of SHARE was reviewed and deemed not suitable for reuse in the current wave. Since it still is not possible to generate such a tool natively in Blaise we had to develop an alternative solution. In chapter 3 various approaches will be discussed and we will explain why we chose our solution.

In chapter 4 the solution we developed will be presented. We added a browser object to the SMS system that communicates with the Blaise API. An html webpage is built to display the questionnaire. This allows for the use of (web-) techniques like cascading style sheets, jQuery, JSON objects, and html5 in a CAPI environment. It provides the freedom to develop your interface and questionnaire behavior to your own liking. With the web-compatible DEP we can reuse many of the tools we already developed for our websites. Since web techniques are well known, and quite easy to learn, we hope that Blaise will allow for a similar model natively, so it will be quicker and the full power of Blaise could better be used.

3. SHARE architecture

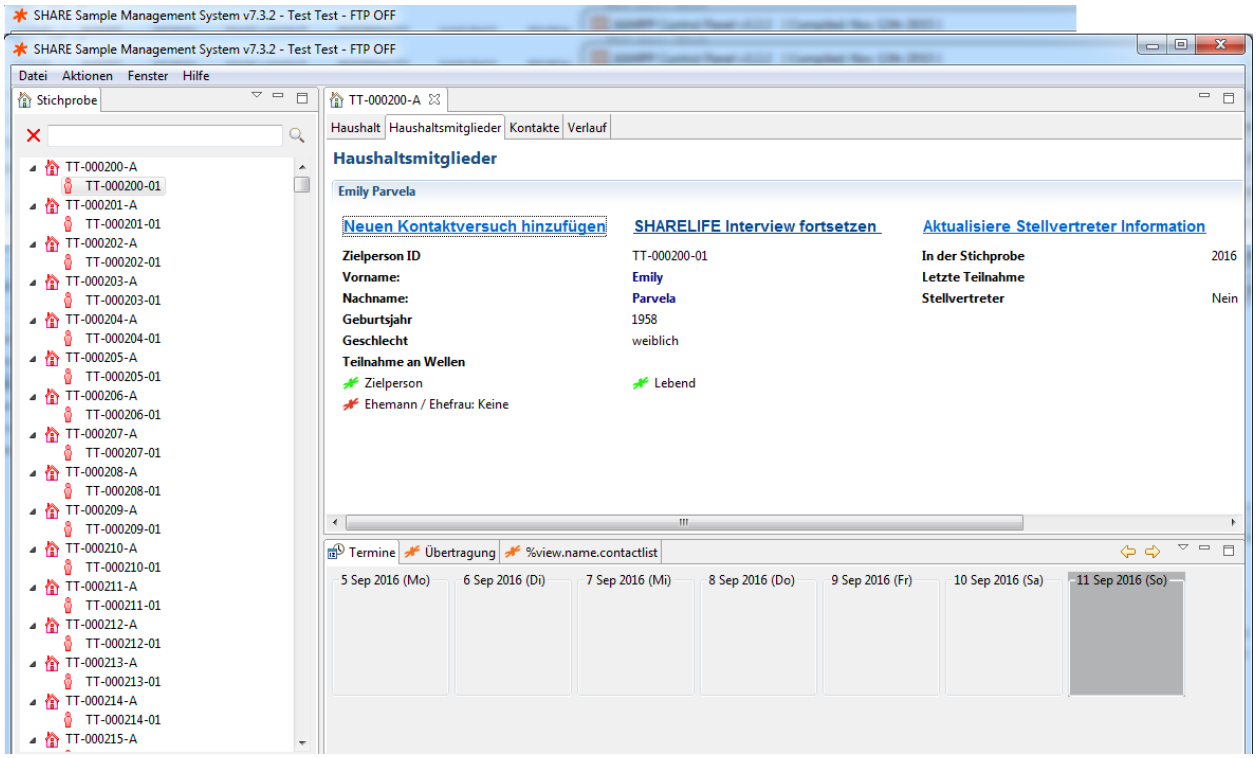
Various tools that need to interact were developed to support the SHARE survey. Since it is a panel study, in many cases the respondent has been visited before, therefor we often need to preload some of the information, to reduce interview length. When available, both a respondent and their partner are interviewed. To optimally identify changes in the household and manage the sample from a central location we developed a Sample Management System (SMS), which is installed on the interviewers' laptop. This program connects to a central server in each country at which subparts of the sample are assigned to interviewers.

Figure 1. Diagram of SHARE Architecture



The SMS uses a wizard style interview to determine the current composition of the household. Once done, it can launch the correct questionnaire for the respondent.

Figure 2. SMS Wizard



In wave 7, two big challenges were presented to us. It was decided that part of the sample, should get the wave 3 instrument again, which was a history calendar. We developed this software in 2010, using VB6 and the Blaise 4.8 API (See SHARELIFE Methodology wave3, Chapter 3). The respondents who already completed the history calendar in wave 3, will now get the regular (or rather updated wave 6) questionnaire. A second challenge was that at a very last moment, it was decided to add 8 new countries, which gives SHARE full EU-28 coverage. This put an extra burden on time and personnel.

4. Approaches

Given the time constraints and complexity of our task, our initial hope was to reuse the software we developed in wave 3. This was a VB6 application, which used the Blaise API, to show a history calendar. This software was developed in 2010, and we weren't really sure, to what extend it would still run on newer versions of windows, and how easy it would be to introduce right-to-left support for Hebrew and Arabic versions. It could also be problematic to integrate other tools we developed since then, for example, we need to support the display of movies, and allow for look-up tables to code, countries and occupations.

To examine the feasibility of reusing this software, we started further developing this version of the VB6 software, and tested it in a pretest in 25 of the SHARE countries. To a certain extend this approach actually seemed to work, it was very slow in some cases, and it was a big hassle to find the correct libraries to install, but it behaved above expected.

In our review we decided that we could not properly support this anymore; if problems occurred during fieldwork, we could not guarantee a timeframe in which problems could be solved. It was also reported by interviewers that the tool was very much outdated.

This presented a new challenge, how were we to display this calendar if we could not use the old tool anymore? Could we think of a trick to do it in Blaise 4.8 DEP? Would Blaise 5 present a solution? Should we develop something like the VB6 tool using another programming language? Should we

maybe strip functionality of the calendar down to only displaying images? We had some experiences with an online version of the history calendar (See Martens, 2013. Jumping around in Blaise IS), but this backend was programmed in PHP and used BlaiseIS, maybe that could help. We would need to install a server on every laptop, which seemed a bit overdone. Should we leave Blaise altogether, develop something ourselves, or look for another vendor that might support the features we need in their software? It was clearly time to make some sort of overview and properly discuss what path we should take to solve the challenge within the given time. In the following table we will present our thoughts and findings:

Table 1. Possible Solutions

Solution	Pro	Con
Reuse VB6 build on Blaise 4.8 API	• Proven • Works with current questionnaire	• Outdated • Limited recent experience • Bad to support • No future solution
Blaise 4.8 DEP	• Stable	• Can't develop a rich interface
Blaise 5 DEP	• New Blaise features seem to fit the regular questionnaire very well	• Can't develop a rich interface • Limited experience • Complete redesign of all tools involved
Build something on Blaise 4 API	• Experience	• What IDE?
Build something on Blaise 5 API	• Future proof • Build up experience	• Complete redesign of all tools involved • What IDE?
Use the earlier developed web solution	• Proven • Easily adapted to work with current questionnaire	• Questionnaire in browser • Install Blaise IS on every laptop
Other vendor	• Could try to find something that fully supports all our requests	• No experience • Complete redesign of all tools involved • Can't reuse already developed code
Develop own survey system	• Full control	• Can't reuse already developed code • Would take very long

Since we only had a few months to find a working solution, the directions that would cost too much time were ignored. For both Blaise native solutions we decided that I would not be possible to create the tools we needed. This left us with either develop something on top of the Blaise 4.8 API again, or find a way to get the web tool working on laptops.

The code from the web tool we developed earlier was an html frontend, with a style sheet for the layout, which was called from a Blaise IS questionnaire. The calendar was generated by a PHP-script, driven by JavaScript calls hidden in the Blaise question texts.

We wanted to avoid installing servers on laptops, so the PHP part had to be replaced by some other part. The display of the questionnaire should not be done in a browser. The standard menus, shortcuts, and other behavior would need to be deactivated. Since our sample management system already launched the DEP to start up an interview, a logical step was to use a standard browser class to launch the browser window from within the SMS.

Once this idea was born, everything fell in place. The Java environment using a library called com4j needed to call the Blaise API, much like our VB6 solution had done before. The browser window could display the html-solution. In the next chapter we will discuss this new solution more in depth.

5. The webdep

Figure 3. Display

As described in chapter 3 we decided that we needed to integrate the functionality of the DEP in our SMS system. Java is able to show a browser component. We would need to show only one webpage, and all submitted actions done by this website will be detected by the system and translated into Blaise actions. The system will call the Blaise API, and transfer the information needed to be displayed back to the website.

A set of high level operations was defined:

- NextQuestion
- PreviousQuestion
- GotoLastQuestion
- GotoFirstQuestion
- GotoQuestion

When one of these actions needed to be done by the webpage, a JSON structure would be formulated like:

Listing 1. JSON Source Code

```
PreviousQuestion
      Key1
      Key2
      Answer
      Status
      RemarkText
      Suppress
```

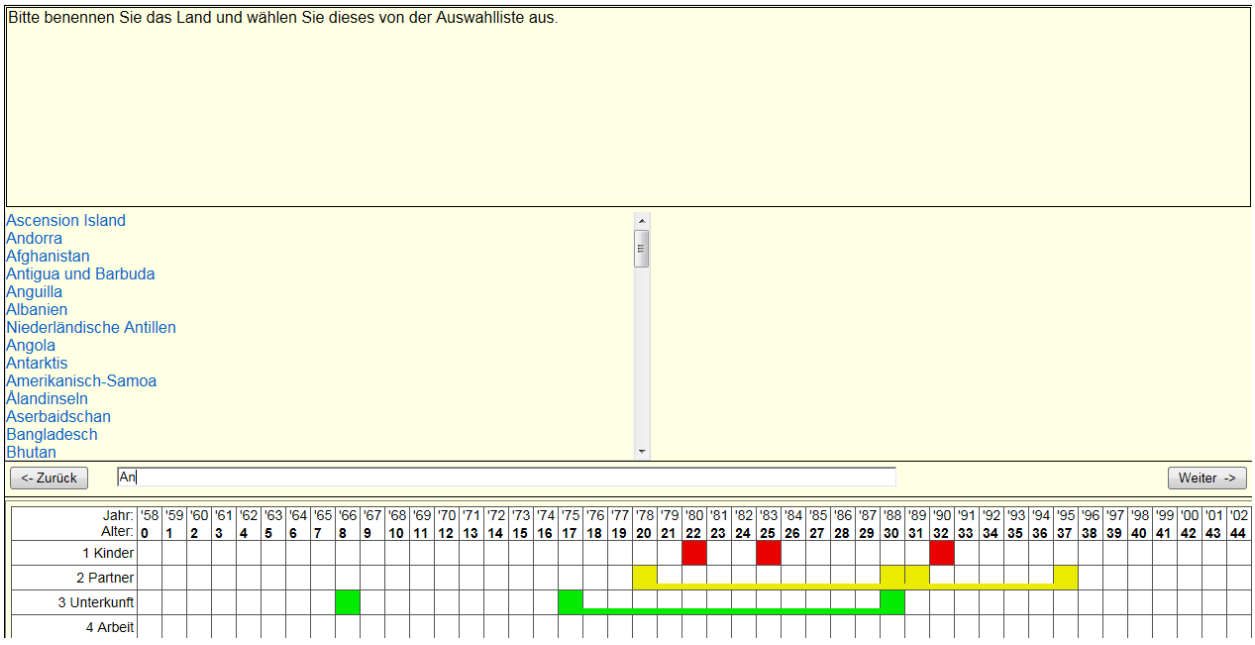
This minimum information is transferred from the webpage to the browser and tunneled through to the Blaise API and generates a new state of the interview. The new state is detected by the Java environment and a JSON object is returned to back the website:

Listing 2. JSON Source Code

```
BlaiseObj
        Key1
        Key2
        Name
        Text
        questionText
        Remarked
        RemarkText
        Interface
        ErrorMessage
        HardErrorMessage
        HardErrorInvolved
        SoftErrorMessage
        SoftErrorInvolved
        DontKnowAllowed
        RefusalAllowed
        Required
        Value
        Status
        FieldDef
                MinValue
                MaxValue
                DataType
                Categories
                IsSet
                TextAsSetString
```

This set allows us to display the questionnaire and provide us with interaction with the questionnaire. JavaScript reads out the new values and adapts the webpage accordingly.

Figure 4. Questionnaire Display



This BlaiseObj object was extended to support some optional extra objects on top of this:

BlaiseObj['Movie']; a list of words with their duration is defined in the Blaise questionnaire. Using the setTimeout-function we can trigger when the words are shown. Before we used a movie to show this list, which we needed to generate for all translations, by hand.

BlaiseObj['Coder']; several lookup-tables were defined as JavaScript arrays, countries, job titles, languages, currencies. The searching in these tables can now be performed in JavaScript giving us full control of the behavious of these tools.

BlaiseObj['Calendar']; the information that needs to be displayed in a calendar, is translated into a html structure, that displays the calendar.

The html page where we load the questionnaire in, is very basic, it is a set of divs. Javascript controls the content of these divs, and stylesheets control the layout.

Listing 3. HTML Source Code

```
<div id="completedep">
      <div id="questiontext-outer">
            <div id="questiontext">
            </div>
      </div>
      <div id="moviediv"></div>
      <div id="errordiv"></div>
      <div id="helptext"></div>
      <div id="answeroptions" class="answeroptions"></div>
      <div id="navigation" class="navigation">
            <div class="navigation-inner">
                  <input name="previousbtn" value="&#8592; Back" />
                  <div style="display: inline-block;">
                        <div id="remarked" class="remarked"> </div>
                        <div id="answerdiv"  class="answerdiv">
                              <input type="text" name="answerfield" />
                        </div>
                  </div>
                  <input name="nextbtn" value="Next &#8594;" />
            </div>
      </div>
      <div id="CalendarDiv"></div>
      <div id="remark">
            <span id="remarklbl"></span>
            <textarea id="remarkText"></textarea><br>
            <input type="button" name="savebtn" />
            <input type="button" name="cancelbtn" />
      </div>
      <div id="console"></div>
</div>
```

This approach gives us the possibility to implement web techniques jQuery, and JavaScript code for CAPI. It allows for use of interface design using style sheets. JavaScript solutions developed already for CAWI mode can be reused in CAPI mode. Using this design we were able to rapidly implement more advanced features like a history calendar, autosuggest lists, and word-recall list in a short time without major changes to the Blaise questionnaire.

An added bonus is that we can directly present the questionnaire over the internet, we could present the calendar to our respondents as feedback, or could attach the questionnaire to our translation environment and present questions in context to our translators.

While implementing, two problems were detected. There was an issue with speed. It took a long time before we got a response from the API. When tracking it down, we found that this was due to the storing of answers. To solve this, we save the answers after the response is returned to the browser.

A bigger problem presented the use of open answers. We wanted the instrument to be completely UTF-8 encoded. However, internally Blaise 4.8 stores strings in ANSI, but depending on your windows installation, it will use different code pages for this. When we want to use open answer responses as a fill in a following question, the open answers that were coded with this different code page, would show as gibberish. To solve this we needed to recode the open answers strings to something that could be formulated in the first 128 characters of the code pages, because these are all identical. Since we now presented the questions in a browser a logical candidate for this would be to encode special characters into htmlchars (Ӓ). For each non-latin character this however takes 7 characters to encode. Since we sometimes need to integrate these open answers in fills we now encounter that these fills can become too long easily. Since these composed fills break off when they exceed length 256, we could still encounter Gibberish texts when open answers are too long. If we could somehow enforce open answers to be stored in a specific code page or if the limitations on string length would disappear, we would appreciate it. As we understood these issues are not relevant for Blaise5 anymore, so these problems may solve themselves in time.

We hope the use of internet techniques in Blaise CAPI, integrating JavaScript, designing interfaces with Style sheets is something that could be supported natively in future versions of Blaise.

References

Martens at al., 2009. Managing Translations for Blaise Questionnaires

Schröder, M. (ed.), 2011. Retrospective Data Collection in the Survey of Health, Ageing and Retirement in Europe. SHARELIFE Methodology. MEA, Mannheim.

Martens, 2012. Blaise Translation Challenges: Versioning, Multimode and Exporting,

Martens, 2013. Jumping around in Blaise IS