Real-Time Case Management with Blaise

Leonard Hart and Erin Slyne, Mathematica Policy Research

Presented at the 15th International Blaise Users Conference, September 2013—Washington, DC

To meet the demand for more efficient data collection, Mathematica Policy Research developed an Integrated Survey Management System (ISMS). As part of the ISMS, we are developing a new case management application called the SCI (Survey Contact Interface). This application seamlessly combines the Blaise data collection system with our ISMS. The two systems will share data in real time.

In this paper, we describe the structure of the SCI and the process of integrating Blaise with it. We also discuss the advantages of the SCI as well as the challenges we faced during development.

1. Background

1.1 Real-time processing

Over the years, Mathematica Policy Research has been working to develop a centralized repository for survey data by re-designing our systems so they can share data in real time. Maintaining data centrally decreases the need to duplicate and transfer data between systems and provides instant real-time access. Real-time processes reduce problems like data de-synchronization and issues with data-transfer scheduling, thereby reducing the time spent troubleshooting. Ultimately, we seek to create a flexible system that saves valuable programming resources, provides end users with immediate access to their data, and is easily adaptable so it can be used for all surveys.

1.1.1 Our history with real-time processing

Over the past 10 years, we have extensively researched Blaise's capabilities to see how well they would meet our real-time processing goals. We researched Datalink, alien-router procedures, and event handlers, evaluating each in terms of its ease of implementation and how well it interacts with our other systems.

We explored the Blaise Datalink utility and discovered that, although it has some powerful capabilities, it did not meet our requirements. We needed the capability to store selected instrument information in its own database table, while simultaneously storing the survey management data in the database used by our Survey Management System (SMS). Datalink only allows users to save instrument data in a single database, with no ability to store common data in databases that share this information with different applications. By implementing some backend triggers in the database, we were able to move data between various tables within the databases in real time; however, this process became complicated to set up and maintain, and we concluded this option was not feasible.

We also researched Blaise alien-router procedures but discovered they require a predetermined list of field parameters. Given that our goal was to build a generic process that is flexible enough to use anywhere for any instrument, and that our data models differ for our various projects, this was not a feasible option. We were able to incorporate this process generically by defining a static, ordered list of all possible fields for integer, real, string, and so on; however, not all fields would be used for every instrument, so we had to ensure there were placeholders containing empty values for these fields. Unfortunately, it became too cumbersome to maintain a large list of field parameters in both systems.

Finally, we examined Blaise's event-handler capability. In a previous paper for the International Blaise Users Conference,[1] we discussed how we successfully implemented this feature to call a Component Object Model (COM) method that invokes an external application. The methods in the COM object execute stored procedures in a SQL server database to move data between a Blaise data set and a SQL database and vice versa. An event can be linked in the data model properties file (.bdp) to a field's user-defined data type or to the end of a parallel block. The COM object is automatically called when the user arrives at the associated field or at the end of a parallel block. Based on the capabilities we researched, this out-of-the-box capability is flexible, reusable, easy to implement, and most closely meets our real-time processing need.

1.1.2 Analysis of our current systems

Before designing a new system or implementing Blaise's event handler, we decided to evaluate the advantages and disadvantages of our current systems. Our SMS system already centrally maintains data for multiple instruments including contact names and addresses, respondent incentive information, as well as case and instrument level statuses. However, much of this information is also maintained in the Blaise shell portion of our survey instrument, which transfers between the instrument and the SMS nightly using an overnight process consisting of batch scripts that process data. The Blaise shell, developed by Mathematica, consists of computer-assisted telephone interviewing (CATI) contact modules, the logic for setting case statuses, and a case call history array. The shell acts as a "wrapper" around our Blaise instruments. Occasionally, we encounter problems with the data transfer process including issues with network connectivity. This can cause the transfer of data to fail when the systems get out of sync as well as cause disruptions to interviewing after completion of the process. To minimize these problems, our new system will ideally collect and maintain this information in one centralized database in real time.

1.2 Planning the new system

Based on our experience using Blaise's event handler to make real-time updates to an SQL database, we planned to build a large scale process to maintain management and survey data in separate databases and tables. Part of the plan was to move the section of Blaise code that collects management data into the Integrated Survey Management System (ISMS) through the SCI. This includes maintaining items such as respondent contact data and instrument status logic separately from our Blaise instrument, and thereby allowing us to remove this code from the Blaise shell. We still needed to maintain a few fields to be used by the call scheduler including telephone numbers, time zone definitions, and status codes. The SCI collects and maintains this information in ISMS and updates in the Blaise database in real time.

To illustrate how the ISMS will operate with a Blaise instrument (see Figure 1-1), we composed a preliminary outline. Blaise calls a generic COM object that invokes the SCI. Blaise would continue to act as the main survey data collection component with the exception of collecting the survey management level data. We incorporated the CATI contact modules and most of the demographic data collection into the SCI in order to reduce the amount of data being transferred nightly between the two systems, thereby eliminating redundancy and possible data discrepancies.

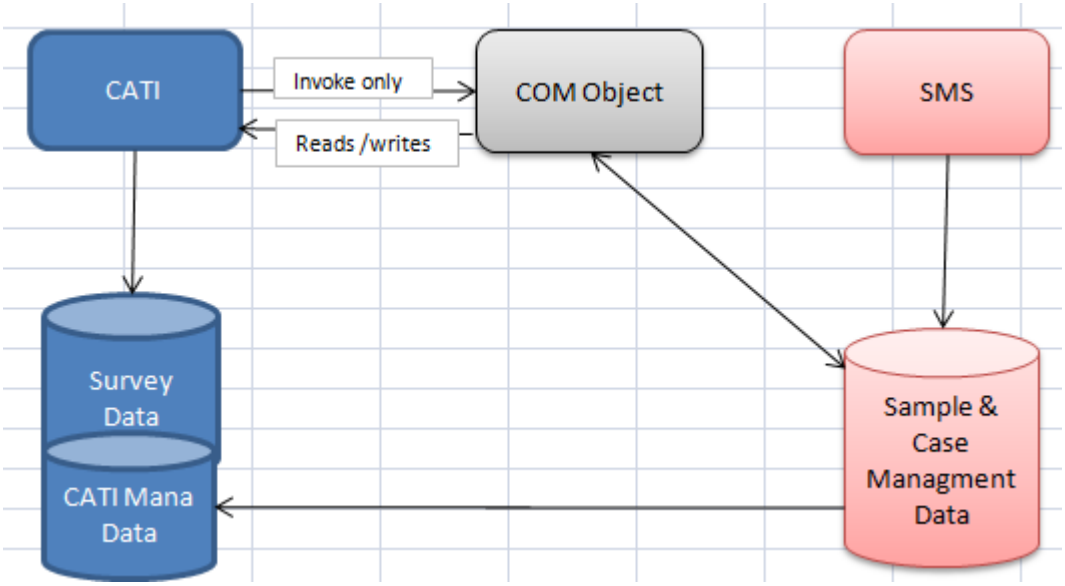
Next, we tested some of the new concepts to ensure they could work as intended. The SCI would be updating the Blaise dataset including the CATI Mana block, which is used by the call scheduler. We verified that we could modify this block externally without the Blaise system overriding its values.

[1] Hart, Leonard, Scott Reid, and Erin Slyne. "The Challenge in Balancing Data Collection Innovations, Remaining Practical, and Being Cost Effective" Blaise Users Group website. Available at http://www.blaiseusers.org/2012/papers/01a.pdf. Accessed June 14, 2013.

We also wanted to prove that we could seamlessly transition between the SCI screens and the Blaise Data Entry Program (DEP). After developing and testing a small instrument, we confirmed we could easily switch between the different application screens.

We built a small prototype that executed one specific path of logic and combined the proof of concepts described above. This prototype showcased the continuous transition between the two systems and proved that data could be read and written to the Blaise dataset using the Blaise Application Programming interface (API). As we developed and tested our prototype, we became confident that we could build this system but understood we might encounter unexpected challenges. However, we felt these challenges were not large enough to deter us from our ambitious goal of creating a centralized repository of all data that is real time.

Figure 1-1. The ISMS



2. Designing the SCI

After we proved these concepts, our next step was to design the ISMS application. We organized the design documentation into two parts: (1) specifications that included screen layouts and detailed logic for setting a case or call status and (2) requirements that consisted of the business rules and case flow process diagrams.

2.1 Defining the SCI and Blaise specifications

We used our Blaise shell as the foundation for designing the SCI because it serves as the backbone for all of our Blaise survey instruments. Unfortunately, the specifications for the Blaise shell were not detailed enough for this document and we were tasked with updating the written specifications to meet the new processes. This process, although time consuming, was advantageous to non-Blaise programmers because it provided a thorough understanding of how the Blaise shell worked before starting to program. The programmers also spent time exploring the Blaise DEP interface to get familiar with how it operates from a user's perspective and for graphic user interface (GUI) development research.

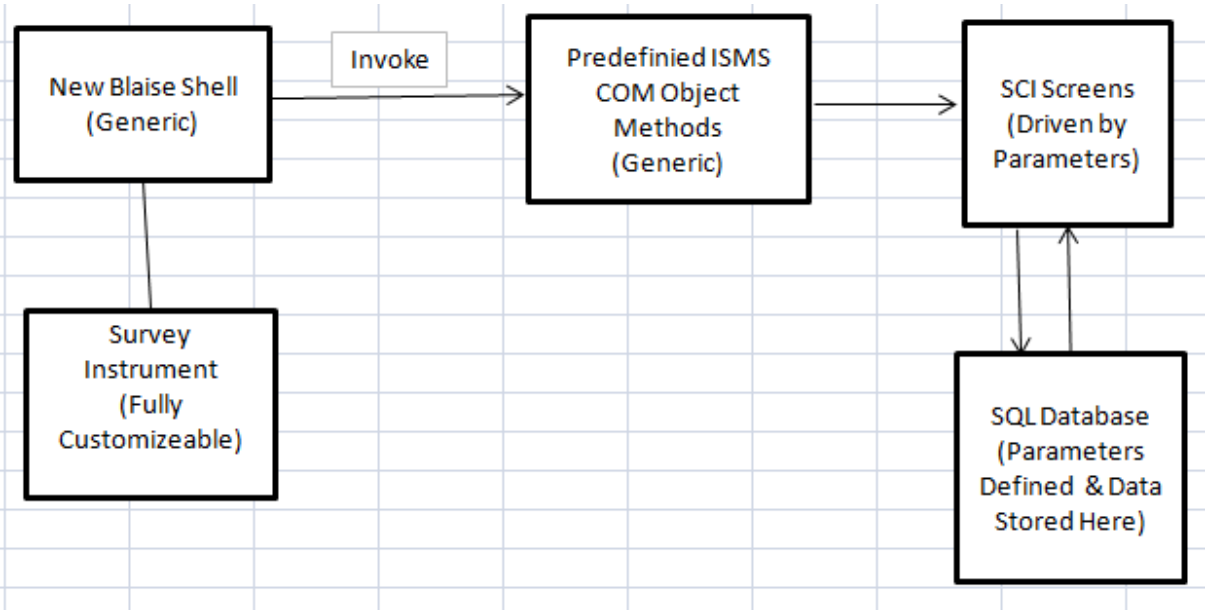
Because we wanted data collection to operate seamlessly between both systems, the SCI would use the same CATI contact screens and logic for setting case and call statuses as our current Blaise shell.

Aesthetically, we wanted to make the SCI screens look and operate as identically as possible to the CATI screens in the Blaise DEP. This included a keyboard driven interface and an info and field pane-type screen design.

2.2 Defining the SCI and Blaise requirements

To meet our goal of developing a generic, parameter-driven system that would require minimal customization, our requirements needed to be specific. We developed a detailed diagram outlining which parts would be kept generic, which would require parameters, and where those parameters would be defined (see Figure 2-1).

Figure 2-1. A Generic, Parameter driven system



As previously mentioned, the Blaise instrument would use events to invoke COM object methods using the Blaise API. The COM object methods would also update the Blaise dataset so that the user would be routed to the correct Blaise field upon returning from the SCI. The CATI Mana fields would also need to be updated to control the case delivery. Exposing the entire Blaise instrument in this way could potentially risk data integrity. For example, a programmer could accidentally assign an incorrect field value or inadvertently alter the data. To help avoid mistakes, we defined a list of all COM object method calls that Blaise could make at specific points and the fields the COM object could update. Any changes to these generic methods would have to go through tight code review before deployment.

Each method invoked from Blaise calls a set of SCI screens. However each project often has different requirements including project specific question text and the number of fields. Since the SCI can easily read and write to the SQL database, we specified parameters in the SCI and defined their values in SQL tables, rather than making text updates for each project. For the non-text modifications, we built extensible modules using Windows' Managed Extensibility Framework (MEF). Each SCI page calls out to a module when it first loads.

Because the majority of the new system is controlled by the SCI, we decided to pare down the current Blaise shell to a minimum number of Blaise management fields that need to be shared with the SCI while retaining only those used to maintain the call scheduler. We also defined the points of exit from the Blaise

instrument where the COM object was to be invoked and which fields it would update. Possible points of exit include interview break offs, setting up appointments for callbacks, completed surveys, or collecting new information about the respondent. For each point of exit, the call would be to a specific, predefined COM object method with an associated predefined set of business rules.

2.3 Bridging the ISMS and Blaise systems

The smooth transition between the two systems was critical to the success of the ISMS. From the CATI interviewer's perspective, the screens need to flow seamlessly, without disruptions or hesitations. To accomplish this, we used Windows message queuing. The COM object puts messages into the queue while the ISMS acts as a listener to read and process each message. From the Blaise side, we thoroughly tested the system to ensure all calls to the COM object methods were valid. These methods also had to assign the correct values to the Blaise fields including the CATI Mana block. Because we were unsure of the CATI Mana block field value assignments that Blaise performs internally, we ran several tests of all the various call results in our current Blaise shell with the watch window turned on before recording them in our specifications.

3. Challenges faced during development

We experienced several challenges while developing the ISMS including programming the SCI user interface, communication between .NET and Blaise programmers, bridging the gaps between the two different systems, and defining specifications for the complex Blaise shell. We were able to resolve the majority of issues by communicating using telephone, email, and face-to-face meetings. We conducted 15-minute scrum meetings to keep track of progress.

3.1 The SCI design

One challenging part of the SCI development was accurately replicating the keyboard-driven Blaise DEP interface. We wanted both applications to look as similar as possible to avoid creating any potential interviewer bias in the data collection or confusion when interviewers are switching back and forth between applications. Our current Blaise DEP application design includes an info pane where the question text is displayed and a field pane where a response is entered. The return key is pressed to allow a brand new info pane to appear. The field pane often remains on the same page so that responses to other questions are visible. Initially, the SCI did not capture the info/field pane concept, and a mouse was needed to select a response. There was no field pane box where the user could enter or edit a response with the keyboard. Eventually, the screen was developed so it operated almost identically to the DEP, with no need to use a mouse. Additionally, the designers wanted to maximize screen space in the SCI application by placing multiple questions on each page. Once the enter key is pressed after entering a response, the question is disabled and becomes re-enabled when the user presses the up arrow key.

3.2 Seamless operation

Joining the two systems so they operate seamlessly was another difficult challenge. In particular, we experienced a "focus" problem when returning to the Blaise instrument from the SCI. Instead of routing to the next unanswered field and being able to type a response, the user had to click on the screen first. This problem was solved by incorporating Windows system commands into the .NET code (see Figure 3-1).

Figure 3-1. Windows commands to fix focus problem

```
Application.Current.MainWindow.Visibility = Visibility.Visible
Application.Current.MainWindow.WindowState = System.Windows.WindowState.Maximized
Application.Current.MainWindow.Activate()
Dim proc As Process = Process.GetCurrentProcess()
Dim hWnd As IntPtr = proc.MainWindowHandle

SetForegroundWindow(hWnd)
Application.Current.MainWindow.Topmost = True
```

Set ForegroundWindow is defined as:

```
Declare Function SetForegroundWindow Lib "User32.dll" (ByVal hWnd As Integer) As Int32
```

When using the Blaise API, we also encountered a floating point unit error when trying to update several different fields in the Blaise instrument. Statistics Netherlands' resolution was to use the latest BlAPI3A.dll and the 3.5 .NET framework. Because the ISMS application was written to use the 4.0 .NET framework, reverting back to 3.5 was not a feasible option, making it difficult to overcome the floating point problem. So far, we have been able to program around this issue while anticipating that it can be addressed in future releases of the Blaise API.

We also experienced some issues with executing parallels blocks using the Blaise API. With help from Statistics Netherlands, we discovered we needed to cross reference the parallel with its associated numerical index in order to ensure the correct parallel is called from the SCI. For the time being, if the parallel block changes, we will have to change the parallel index number in a configuration file. Currently, the parallel index solution works but it is not generic; we will need to revisit this during development for a future release of the SCI. Our goal is for the application to be capable of determining the proper parallel index.

3.3 Specifications

One challenge in particular was to create and edit specifications for the new system based on the complex Blaise shell. Capturing and formatting all the variable assignments with corresponding screenshots was difficult and time consuming. The .NET programmers frequently needed clarification because they did not understand the methods of specifying Blaise logic. Through increased communication we were able to work together to resolve misconceptions.

4. Conclusion and Future of ISMS

Developing the initial version of ISMS has been challenging, but we have learned several important strategies for improving future releases. First, good communication is imperative to implementing a quality system. A project can consist of competent programmers that successfully create separate modules, but if they do not collaborate, the desired outcome will not be achieved. Good communication involves selecting the most effective mode to accomplish a particular task. For example, email is more effective for lists and reminders, while face-to-face and telephone conversations are good for talking through problems. Meetings are an effective way for multiple people to disperse and gather information. As mentioned in Section 3, we used scrum through most of the development stages. During these 15-minute meetings, key development staff would give a brief update of their progress and implementation

plans for the immediate future. This kept everyone focused on his or her task at hand without having to go into great detail.

Second, we learned that the procedure for defining requirements was as important as the content. The SCI and Blaise specifications were combined into one document rather than maintained separately. In hindsight, this was a mistake, but at the time we thought this was the best solution. We thought we could modify our current documentation instead of creating another document. Unfortunately, this added confusion to the process of understanding what items belong in the old system versus what should be included in the new system.

Third, we discovered that the experience of developing the SCI has allowed its programmers to learn different technologies and applications. At times, they had problems communicating with other programmers with different expertise. During development, some struggled to understand how other parts of the system would work. This experience has broadened their knowledge of other systems and improved communication for future projects.

Finally, during planning and development stages, we discovered that due to cost and complexity, we needed to postpone several features we wanted to incorporate into the first version of the SCI. To avoid complication, we decided to build the SCI for CATI data collection only but kept in mind that computer-assisted web interviewing (CAWI) and computer-assisted personal interviewing (CAPI) would eventually be incorporated. As we encountered items during the development process that could affect multiple data collection modes, we either addressed them immediately or noted them for future releases. We also noted the capabilities we wished to add to the next version and possible advancements to research for feasibility review.

For future versions of ISMS, our primary goal will always be a centralized repository of data for all applications used to conduct surveys. After successfully implementing several CATI surveys, we plan to start modifying the SCI for CAWI surveys to accommodate our multi-mode surveys that utilize CATI and CAWI.

As of writing this paper, Mathematica is developing its first production CATI survey using the ISMS with the SCI. At the next Blaise conference, we hope to report on our successes with implementing the system, including any additional lessons learned.