UNIVERSITY OF WATERLOO
Software Engineering

Analysis of Different High-Level Interface Options for the Automation Messaging Tool

Deloitte Inc.
Toronto, ON M5K 1B9

Prepared By
Matthew Stephan
Student ID: xxxxxxxx
Userid: mdstepha
3B Software Engineering
December 7, 2006

X
X
X

December 7, 2006

William Wilson, Director
Software Engineering
University of Waterloo
Waterloo, ON N2L 3G1

Dear Professor Wilson,

This report, entitled "Analysis of Different High-Level Interface Options for the Automation Messaging Tool", is the fourth work report that I have written. The information presented in the report is based upon my recently completed 3B work term with Deloitte Inc.

Deloitte Inc. provides technical consulting for a myriad of different companies and project types. I was situated in the Technology Integration group, which is responsible for projects involving system integration, strategy, management, architecture, infrastructure, and information dynamics. During my time with that group, I was put on the Quality Assurance team for one of Deloitte's current clients, a bank that, for the purposes of this report and for privacy reasons, will be referred to as Goodbank. My responsibilities included creating automation tools that testers can run, drafting a wide variety of document, and writing test cases to meet provided test requirements.

The following report performs an analysis on the different interfaces that can be implemented on top of a proposed Quality Assurance tool at Goodbank that will allow testers to automate a number of different test cases. This problem is one that I encountered during my work term, as a high-level interface was needed for the automation tool that was to be implemented. My team had to decide which interface was best suited to the task given all the requirements and factors.

I would like to thank my teammates, Elaine Oei and Joshua Guo, for assisting me in understanding the proposed Quality Assurance tool that was to be put into place and for informing me about the different interface options available to us. I also wish to thank Frankie Stephan for proofreading my report and ensuring that it adhered to all of the report guidelines. I hereby confirm that I received no help, other than what is mentioned above, in writing this report. I also confirm that this report has not been previously submitted for academic credit at this or any other academic institution.

Sincerely,

Matthew Stephan
Student ID: xxxxxxxx

Executive Summary

The following report investigates two possible high-level interface options that can be used for the Automation Messaging Tool that Deloitte is creating for Goodbank's Quality Assurance department. The scope of the report encompasses only the interface for the tool and the Java component that will interact with the interface. It does not delve into how the tool automates the messaging process or how it interacts with other systems within Goodbank.

The first interface option presented is the use of XML files. This has the testers create the request messages that they want to send and the expected result messages they should receive. They then place them in directories such that the Java program can locate these files and utilize them. The second option is the use of Excel files. This has the testers specify parameters about a message on one spreadsheet and add rows of data to different spreadsheets that are then used to construct the request and expected XML messages.

Both options are shown to be valid interfaces for the tool and are then compared based on three different criteria. It is concluded that the XML files interface is quicker to implement, is slightly more efficient when it comes to converting the input data into request and expected messages, and requires more work for the Goodbank testers once the interface is put into place. The Excel files interface is simpler for the testers and will require much less manual input than its counterpart. It will, however, take longer to develop and is slightly less efficient than the XML files interface option.

The recommendation put forth is to construct the Excel files interface because of the importance of the end-user experience. It is worth the longer development time and decreased efficiency in creating the messages. If Goodbank decides that they would prefer shorter development time or increased efficiency in message construction at the cost of significantly increasing the amount of work for their testers, then the XML files interface should be developed.

Table of Contents

List of Figures

List of Tables

1. Introduction

Since the advent of computers, people are persistently finding new and innovative ways of automating as many tasks and processes as possible. Automation is a desirable goal for many companies because it is a very effective way of increasing efficiency. Once a process is automated, a company need not allocate a lot, if any, resources to executing that process since it is now performed without any manual intervention. It then follows that Goodbank, a bank whose name has been changed because it is not needed for the report and for privacy reasons, will want to automate a significant number of tasks. This is especially true in the case of quality assurance (QA) because of the importance it holds for financial companies, like banks, and the amount of resources that banks typically require for QA assignments. It is important to note that "(Manual) Testing accounts for 25% to 50% of the total budget in many software projects [1]" which is no exception in the case of Goodbank. One group of processes that Deloitte has been brought in to automate are the QA tasks that involve testing the messages that are sent between the many systems utilized by Goodbank. These messages are used to relay information and commands between the bank's services. All of these messages are in Extensible Markup Language (XML) format but differ considerably in the layout of the elements. All of the respective systems know how to interpret and respond to messages that fit each system's specified format; all other messages are ignored and discarded. Currently, the QA department of Goodbank tests the messaging services by manually creating XML request messages that satisfy certain test requirements, comparing the response messages they receive, also in XML format, and determining if there are any discrepancies between the response message and the expected response message. Manual testing, such as the process just described, is often considered "difficult, tedious, time consuming, and inadequate [1]". This is the task that the Deloitte team has been brought in to automate.

It is clear that the actual automation component of sending, receiving, and parsing of messages can be accomplished using Java given the resources of the bank and the skill set of the people creating the tool. The problem/issue arises in determining the high-level

interface that the testers will use to specify the test cases and to identify the expected response messages. The interface will then facilitate the launch of the automated testing of the messages. It is important that an excellent high-level interface is chosen because the scope and context of the automation tool entails that it will be utilized by all Goodbank testers after Deloitte is finished the project and it will be used to automatically send messages to all of the bank's systems. The main design constraint in choosing an interface to implement is that it must satisfy the requirement of allowing users to input data such that the Java program is able to create the appropriate request and expected response XML messages. It must also be flexible enough to allow for users to enter data in a uniform manner for all of the different systems, that is, it will not be required to change the interface in order to allow for messages to be sent to each of the systems. Since there are multiple message types per system, such as adding or requesting a data entry, another constraint is allowing users to create multiple messages of a single message type without manipulating the interface. The design criteria for determining the optimal solution include time and cost of implementing the interface, efficiency in converting output from the interface into XML messages via Java, and difficulty and time required for users with respect to entering data into the interface.

This report is intended for the Deloitte team currently engaged at Goodbank, or anyone else, who needs to decide an appropriate interface to facilitate the construction of XML messages in an environment such as the one at Goodbank. It is for readers who have a basic understanding of XML messages, Microsoft Excel, and Java programming. Beyond those concepts, all other aspects required to grasp the report will be elaborated upon. The report begins by briefly discussing the current process that the QA team uses to test messages and the automation tool that is being proposed by Deloitte. It then continues by investigating two possible high-level interface options for the automation tool and how they can be implemented. A comparison between the two options is performed using the criteria mentioned above as the points of comparison. Lastly, conclusions are drawn and recommendations are made regarding which interface is more suitable for the Automation Messaging Tool.

2. Current QA Process and Proposed Tool

Goodbank currently has in place systems that communicate to one another via the use of XML messages. Each system accepts only messages that fit a certain form/specification, therefore, when inter-system communication is desired, the sending system must be aware of what messages will be accepted by the receiver. Understandably, the QA associated with manually testing and verifying this process is quite cumbersome. As exhibited in Figure 1, it entails creating request XML messages for each system that cover all of the applicable test cases. Each message is sent using the existing Goodbank message-sending framework, which entails sending a single message to a specified system and then receiving a corresponding response XML message. The received message is then compared with an expected/correct XML message, which must also be created or gathered by the tester, and the results are then stored for viewing and evaluating.

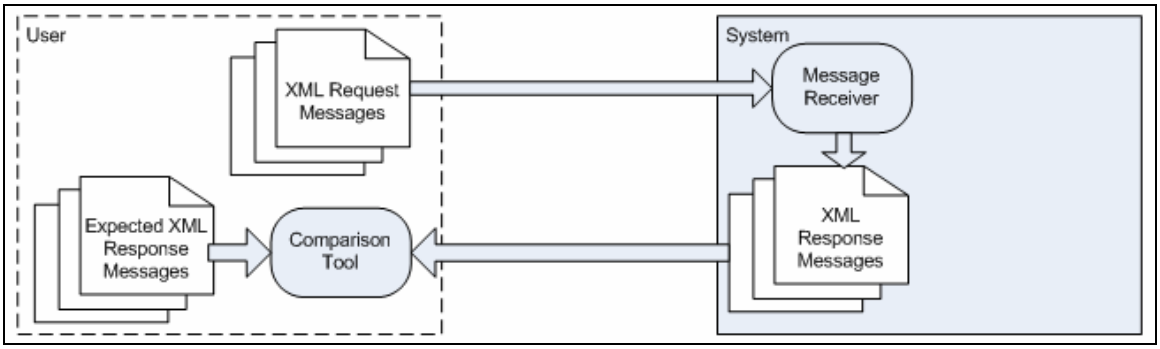

Figure 1: Current method of testing messaging services

The manual execution of this process is quite draining on resources as it requires two to three QA testers to perform these steps multiple times throughout the day, roughly four to five times a day. The Deloitte team has come in to create a tool that can automate this process. The tool, written in Java, will allow testers to run, or schedule a group of runs of, an executable that can kick off a large number of messaging tests. This will be facilitated by having the tool create the different sets of messages using the data input from the testers. It will send and receive the messages to the appropriate systems by having the Java program utilize the existing message-sending framework, which is a

group of Java libraries. The program will then perform XML comparisons between the response XML messages and the expected XML messages, when appropriate. Lastly, the results of the individual comparisons will be output in a format that is easy to summarize and grasp by the testers. The only missing component in the design of the tool is the nature of the high-level interface. As illustrated in Figure 2, the testers would use this interface to specify the data that indicates the message content and destination, the Java program would interpret the output from this interface and create the appropriate request and expected XML messages, and the process would continue as mentioned above. The requirements of the interface are elaborated upon in the subsequent section.

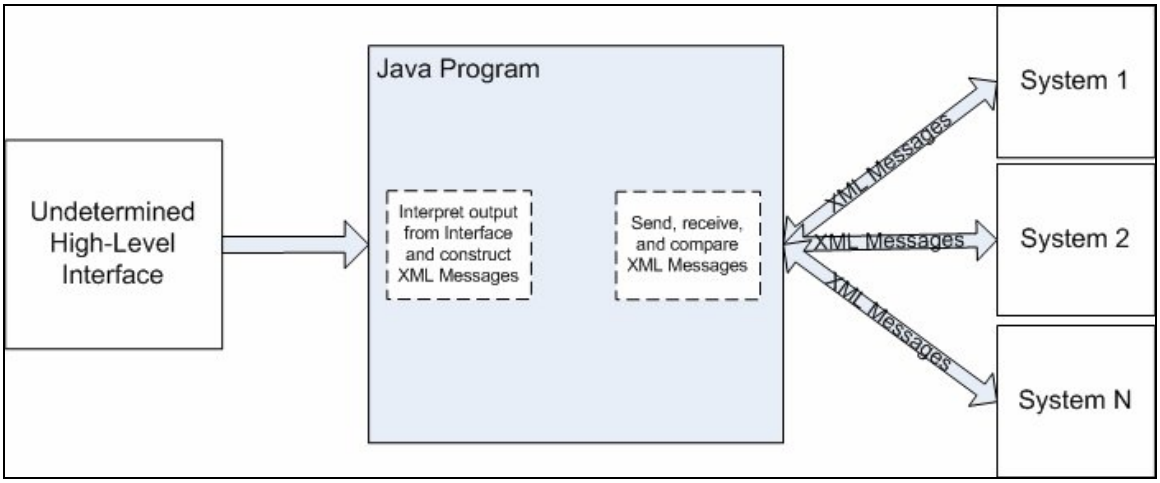

Figure 2: Overview of proposed Automation Messaging Tool

3. Interface Options

As alluded to earlier, there are a number of constraints that must be considered when choosing the high-level interface for the Automation Messaging Tool. Firstly, users need to have the ability to input data in a form that the main program can interpret and eventually convert to XML messages. The interface must also ensure that the entering of data is always done the same way, regardless of what system the messages are being sent to. Lastly, the interface must allow for multiple messages of a single type, but with different data, to be generated. An example of this is requesting a data record with one field, such as date of birth, and attempting to request the same data record using a different field, such as last name. The messages in this instance are the same type, but contain different data. The following subsections describe two possible interface options that are being considered and the way in which each of them satisfies the constraints.

3.1 XML Files

One possible interface that can be utilized to input data to create the messages is using XML files. This entails having each system having its own set of templates, XML files with no data values, which users copy and fill in values for. The templates are made encompassing enough that they cover all the different message types, test cases, and systems. The users fill in the templates to create the actual XML request and expected message files with the appropriate data. The files would then be placed in directories in such a way that the Java program can determine which systems to send to and which files to compare after the request is sent. As shown in Figure 3, one such directory hierarchy has the highest-level folder contain all of the systems that messages are being sent to. Each of these folders contain three subfolders: one for the request messages, one for the expected messages, and one that will be populated with the actual response that was received from each request. The request messages and expected messages must have file names that begin with the same characters, such as update01_request.xml and update01_expected.xml, so the Java program is able to determine which expected response is to be acquired from which request.

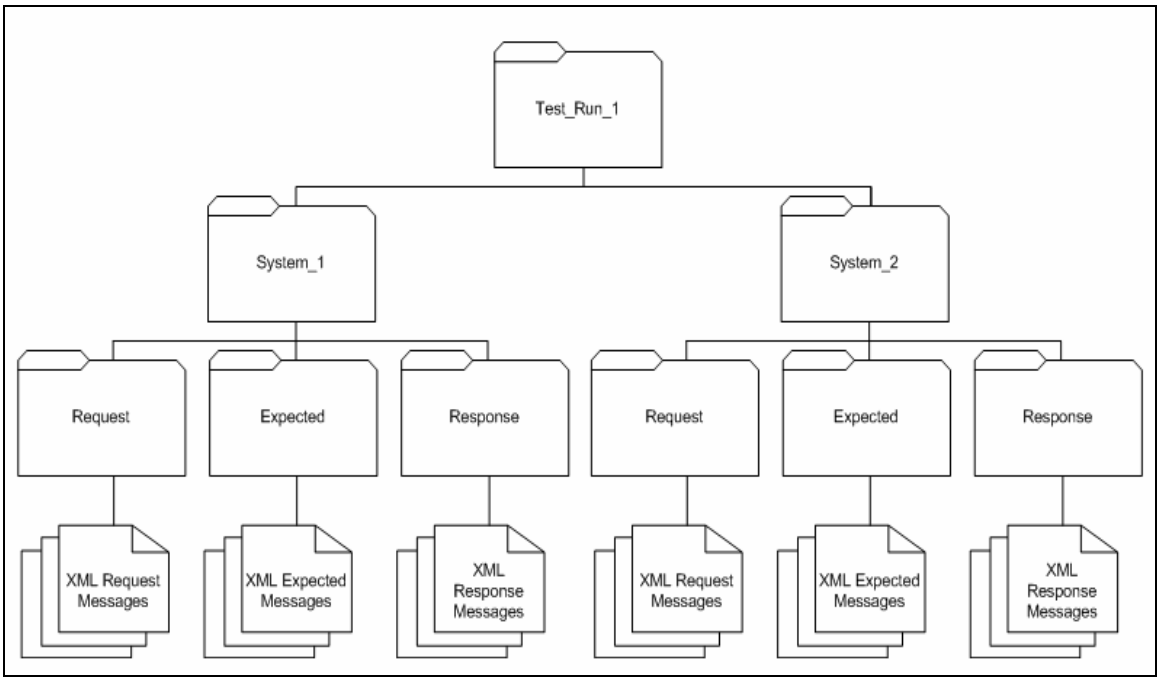

Figure 3: Sample layout of directories for XML file implementation

The Java program will be provided the location of the highest-level folder containing the entire test run, traverse the system directories, interpret, evaluate, and send each of the request messages to the respective systems. The response messages received will be placed in the appropriate folder and compared to the corresponding messages in the expected folder.

Testers will be required to create the XML request and expected messages only once. Rather than sending messages one at a time and performing comparisons, the program will be able to grab all of the request messages in the provided test run directory and perform comparisons between the expected and response messages.

3.2 Excel Files

Another high-level interface option for the Automation Messaging Tool is to use Microsoft Excel spreadsheets. The Java program can read and interact with Excel by using any number of widely available open-source distributions. The most common example of this is the Java Excel API, also called the JExcel API, which is an "open-

source Java API which allows Java developers to read Excel spreadsheets and to generate Excel spreadsheets dynamically [2]." Using this API allows for a workbook and its corresponding spreadsheets to be open, read, and even modified [2]. The Deloitte team has made the assumption that it would be much less effort to use an open-source tool rather than write their own. Once installed, the API would be utilized by the Automation Messaging Tool by having one general Microsoft Excel workbook, a file containing many spreadsheets, created by the developers of the tool and provided to each of the testers to fill in. Each spreadsheet in the workbook would be used to represent a different message type except for the first sheet. Figure 4 exhibits a very simplified version of the message-type specific spreadsheet. As shown, it has columns that represent all of the possible fields that can be populated for the particular message type. Each message type would have two spreadsheets: one spreadsheet would be for the request data and one spreadsheet would be for the expected data. Each row in the spreadsheet would be filled in with data that will be placed in the message for the field specified by the column. The "Message DATA ID" column is used to uniquely identify the data row while the "ROW #" column is used to provide a direct link from the first sheet, explained below, to the message-type specific sheet.

	A	B	C	D	E	F	G	H	I
1	Message DATA ID	ROW #	First Name	Last Name	Address	City	DOB	Customer Type	Active
2	TD-1	2	Fox	Mulder	1 Blue Street	Toronto	4/12/1974	3	Y
3	TD-2	3	Bob	Funland	3 Fake Street	Quahog	7/15/1984	1	Y

Figure 4: Simplified version of data sheet for a specific message type

Figure 5 showcases a minimal version of the first spreadsheet in the workbook. It contains one row for each message that is going to be sent by the application. The columns required for each row include the system that the message is being sent to, which the tester will select from a drop down list; the message type, which refers and links to the appropriate sheet in the workbook and is also selected from a drop down list; the row of data inside the message-type specific spreadsheet; and the name of the XML request template and XML expected template, which will be automatically filled depending on what message type is selected. These templates differ from the templates mentioned in the XML file implementation and will be explained below. Having this sheet gives the Java program the ability to iterate through these rows and use the

information to send, receive, and compare the sets of messages. The values stored in each row are used also when the Java program needs to create the internal representations of the messages because it needs to grab the data from the appropriate row in the correct message-type spreadsheet.

	A	B	C	D	E	F	G	H
1	Message ID	Message #	System	Message Type	Data Row #	Request Template	Expected Response Template	Comment
2	MSG_System_A	1	System A	Update Record	2	Update_Request.xml	Update_Expected.xml	
3	MSG_System_B	2	System B	Delete Record	3	Delete_Request.xml	Delete_Expected.xml	
4	MSG_System_C	3	System C	Add Record	3	Add_Request.xml	Add_Expected.xml	

Figure 5: Simple example of first spreadsheet in workbook

As mentioned earlier, the first spreadsheet in the workbook contains columns that are automatically populated with the names of the request and expected response XML templates for the message types. These templates contain XML messages with values that represent column names in the message-specific spreadsheet, allowing the Java program to place values in the correct locations within the template. The process that the Java program executes after all the appropriate data has been entered is illustrated in Figure 6. As shown, the Java program is able to interpret these column names from the template. It acquires the values that the testers entered from the corresponding message-type specific spreadsheet and uses these values to construct request and expected messages that fit the system-specific format, which is specified in the first spreadsheet in the workbook. Then, it iterates through the list of messages on the first spreadsheet and sends each of them to the appropriate system.

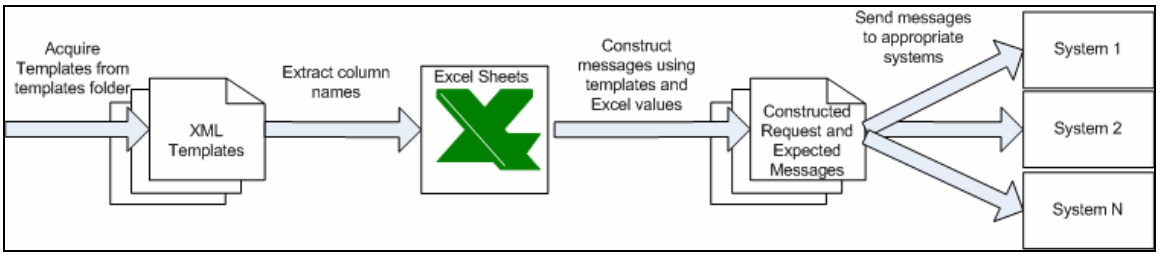

Figure 6: Steps taken by the Java program

4. Comparison

Both of the options discussed satisfy the constraints necessary to be considered an acceptable solution. For the XML files interface, the interpretation of the input data is accomplished by reading the XML files, parsing the data using existing Java libraries, and constructing internal representations to send to the different systems. Since users place the XML messages in the folders under the system folder that they want to send the messages to, consistency is achieved in the entering of message data for all of the systems' respective messages. Lastly, many messages of a single type and with different data can be created and sent to a single system by placing the data-filled messages in the request folder under different names and running the program. On the other hand, the Excel files interface meets the constraints through the combination of Microsoft Excel and the JExcel API, which facilitates the entering of data in a manner that the Java program can use. It gives the testers the ability to enter data and send messages for different systems on a single spreadsheet by having the Java program use the System column on the first spreadsheet to determine how to proceed. Since the message types and destination system are specified on the first spreadsheet and there is no restriction on the types that can be entered, users can construct many messages of one type and send them to a single system without needing to manipulate the interface or the program.

Now that it has been established that both of the options are acceptable solutions, it is necessary to contrast them on the basis of the design criteria. As mentioned earlier, they will be compared on the basis of time and cost of implementation, effectiveness in using data from the interface to create the XML messages, and the difficulty and time testers will have to invest in entering data.

4.1 Time and Cost of Implementation

Since neither of the interfaces are created at this point, it is prudent to determine the amount of time and the cost required to actually implement them. The following sections will evaluate each of the options in this area.

4.1.1 XML Files

There are two main tasks required in implementing the XML files interface option. Firstly, the XML templates that the testers will use as a baseline must be created. This entails gathering existing request and response XML messages for each message type and for each system. The template developers then edit these messages in a format that makes it easy for the testers to copy the files and fill in the correct values. While this task may seem simple at first glance, Table 1 shows that the amount of XML tags that need to be converted to template tags is significant. As demonstrated, each system, who is referred to by number instead of name for privacy reasons, has a varying number of lines per message. Although each system has the same six types of messages, the difference in the amount of system-specific lines is notable. The overall average number of lines for all messages, calculated by taking the average of the systems, is 144. This means that, since we have 30 messages, the developers creating the templates will have to modify close to 4500 lines of XML messages. While the Deloitte team developers believe that it will take two days to create and verify the templates created in this manner, they also believe that it will take at least three days to acquire all of the correct request and response messages from the various Goodbank systems. Continuing on this assumption, it means that it will take at least one week to gather, create, and validate the templates.

Table 1: Average lines per message for each system

System Number	Average Lines Per Message
System 1	127
System 2	220
System 3	85
System 4	103
System 5	184

The second task required in implementing this solution is to create the Java component that is able to pick up and interpret the messages and that will place all responses received in the appropriate response directory. It has to be developed in such a way that it can be given a test run as a parameter, traverse the different systems, and send and receive the appropriate messages. The directory structure must also be provided to

the testers because the program will be looking for specific folders that have the names of systems. This task is not as much effort as creating the templates; as such, the Deloitte team estimates that it will take roughly 2 days to create the functionality and thoroughly test it. Therefore, the XML files interface will take close to a week and a half to implement, which implies a cost of paying the Deloitte QA team on the Goodbank project for that time.

4.1.2 Excel Files

The first task that must be accomplished in order to have an Excel files interface is to create the basic workbook that all of the testers will copy and use. Each of the message-type specific spreadsheets must be created in such a way that all of the possible XML tags in that message type are represented by a column. Similar to the XML files implementation, this entails gathering valid request and response messages in order to fill the columns. Creating the columns would take a day of development by the team and gathering the many messages from Goodbank's systems would take at least three days. It is also necessary to go through the request and response messages and convert them into templates that the Java Program can use to replace the tag values with values retrieved from the spreadsheets, which is estimated to take a day or two by the Deloitte team.

Another important component of the workbook is the large number of macros, which is the automation of one or more commands that respond to certain actions in an Excel sheet, which are necessary to make the testers' jobs easier. As discussed earlier, this includes having the template file names automatically filled in when a message type is selected, generating the test data row number and linking it back to the first sheet, and a number of others. The Deloitte developers estimate that these macros will take two days to create and test. Another significant task necessary in employing the Excel files interface is to create the Java module that will interpret the Excel files and extract the appropriate test data. This includes becoming familiar with the JExcel API and using it to iterate through the messages situated on the first spreadsheet, as mentioned previously. Given the complexity of the task, the Deloitte team has estimated that it will take two

weeks to develop and write the unit tests. In total, the Excel files implementation will take a bit more than three weeks to put into place.

4.2 Efficiency in Creating Messages using Interface Data

Since the Java program will either be called by testers when a test run is desired or called many times by an automated batch process, it is important to consider the efficiency in having the Java program interact with each of the different interface options. An efficient solution is desirable because it will ensure that the testers running the tool can minimize the time they are waiting for the program to execute.

4.2.1 XML Files

The computation required in creating messages from the XML files interface is quite minimal because the messages are practically created to begin with. Figure 7 illustrates the main low-level processing that the Java program will be required to execute in order to use XML files as the interface. The program first needs to determine the correct system folders that exist under the test run. It then must parse both the request messages and their corresponding expected messages and create internal representations that can be used by the Java program. It then sends the request message and constructs a response XML file from the response message received

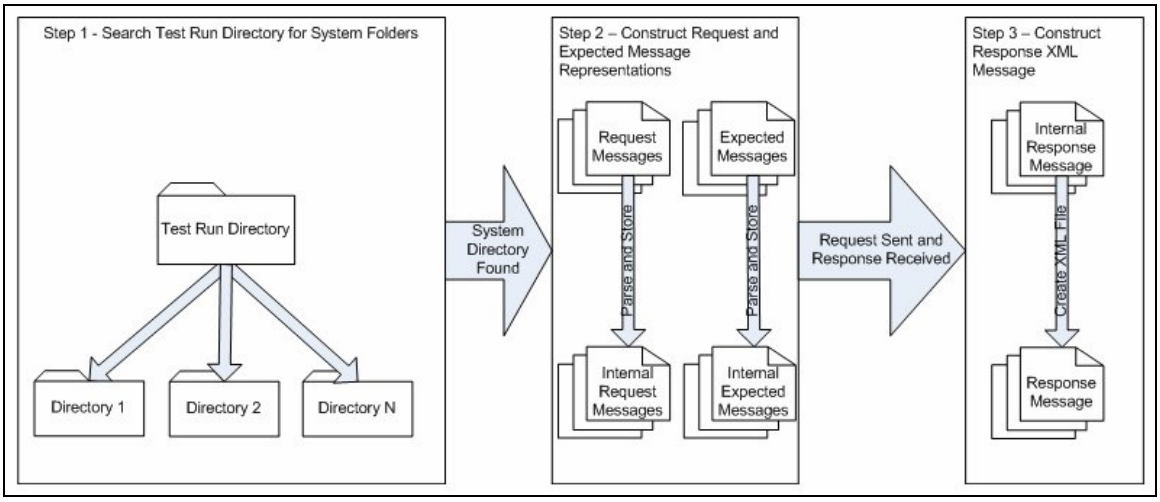

Figure 7: Main processing in XML files implementation

As mentioned earlier, the average length of a message that will be used in the Automation Messaging Tool is 144 lines. It will be assumed that messages will not be more than two levels deep, which is consistent with the Goodbank messages that the Deloitte team has seen up to this point. As such, the processing that will be required to parse the messages will necessitate very little recursion and will run in linear time. In order to traverse the system directories, it would be prudent to look for a pre-defined list of directories that map to systems rather than check each directory under the test run and determine if that matches the list. This requires a few simple calls and can also be accomplished in linear time because only the first level directly under the test run is searched. The only significant waiting time that may occur is when the request message is sent to the system. However, this has nothing to do with the interface and is out of scope for the interface to deal with. Constructing a response message is similar to constructing the request and expected messages and can be accomplished in linear time.

4.2.2 Excel Files

There are two notable processing tasks, shown in Figure 8, that comprise creating messages using the data entered in the Excel files. Once it is determined which XML template to use, as retrieved from the first spreadsheet, the corresponding request and expected template is parsed and the tag names are extracted. The Java program then opens the appropriate spreadsheets and searches them for the columns that match the tag names that were extracted. It can then store them in a mapping object and use this to construct the correct request message to send and expected message to compare against.

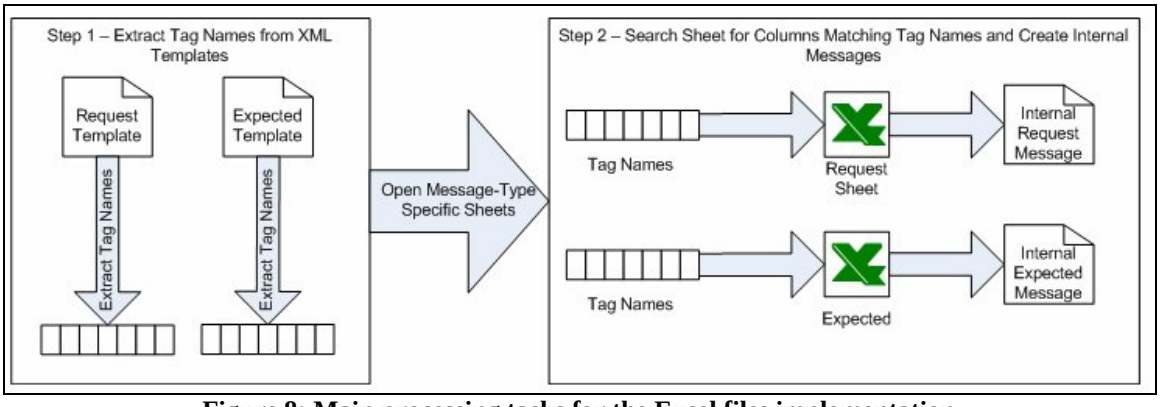

Figure 8: Main processing tasks for the Excel files implementation

Extracting the tag names from the XML request and expected messages begins by parsing the template. Once completed, the Java program will need to traverse the internal representation of the template and store a list of the tag names while disregarding any other values. This can be accomplished in linear time because parsing the file requires linear time, as explained earlier, and extracting the tag names requires iterating through each line in the XML file only once.

Acquiring the data from the Excel spreadsheets is not as straightforward. When the JExcel API searches a spreadsheet for a specific cell by its contents, "The search is performed on a row by row basis, so the lower the row number, the more efficiently the algorithm will perform [3]." This means that on sheets that have many entries, such as the first spreadsheet having many messages or a message-type specific sheet having many test data entries, the search can become noticeably slower. The JExcel API will search each column in each row until it finds the cell specified. If a sheet has m columns and n rows, then the worst-case runtime would occur when using the last row of data. The first entry in this row would cost time m, the second would cost m + 1, the third m + 2, and so on until m + n-1. The last row of data will always be accessed in the case of the first spreadsheet and is likely to be accessed in the message-type specific spreadsheets. Since each cell will only need to be accessed only once, the runtime for retrieving data from the Excel spreadsheets is m+n, where m and n are the numbers of columns and rows, respectively, in the largest spreadsheet being accessed.

4.3 Difficulty and Time Required for Testers Entering Data

Testers are the true end users of this product, so the amount of time and the difficulty in entering data is an important consideration. The difficulty level becomes an issue if one or more of the testers become frustrated with complexity of entering data. The time required for entering data can become a problem if the testers find the work too tedious and too drawn out. The following section will consider both of these for each of the interface options.

4.3.1 XML Files

In the XML files version of the interface, testers are required to create one XML request message and expected message for each message they want to send. They will be working off the templates created by the Deloitte team and need only to fill out the values in the XML tags. The time required for the testers to enter all the data depend on how many messages they want to send. Preliminary inquires with the Goodbank testers suggest that there will be close to 200 messages sent. This means that 400 separate XML files will need to be created and placed in the appropriate folders. Since there is no way of creating more than one file at a time, this may become quite time consuming. With 400 files to be created, including messages of the same message type and system, and an average of 144 lines per message, this means the testers will have to manually edit around 57600 XML lines from the templates. The difficulty in doing this is not very high, but the testers must have a good understanding of where to place the files. Regarding the time required, the Deloitte team was able to successfully edit and verify an XML message from a template in an hour. This translates into roughly 400 hours that the testers, as a whole, will have to invest towards creating the files. Once done, however, they can simply call the Java program and the all of these messages will be sent and compared.

4.3.2 Excel Files

The entire test run to be executed by the Java program can be specified in one Workbook given the fact that a Microsoft Excel Workbook can hold 256 columns and 65536 rows and none of the request and response message have more than 256 tags [4]. For each message that a tester wants to add to the test run, they will add one row on the first spreadsheet, one row on the request message-type specific spreadsheet, and one row on the expected message-type specific spreadsheet. They need not concern themselves with where the files are to be placed because only internal representations of XML messages are used. The difficulty should not be of any concern to the testers since all of the Goodbank computers come with Microsoft Excel installed and all of the employees are familiar with the product. Any advanced details will be handled by the macros that were discussed earlier. The Deloitte team created a prototypical workbook based on a

request and a response message that were available, both containing close to 144 XML lines. They were able to fill out the spreadsheet and manually verify the entries in 45 minutes. Since there will be nearly 200 messages to create, there will be 200 rows of message information in the first spreadsheet, 200 rows of request data added, and 200 rows of expected data added in the spreadsheet. Assuming 45 minutes to fill out the spreadsheet for each message to be sent and 200 messages, it means the testers, as a whole, will need to devote 9000 minutes, or 150 hours, to filling out the workbook to run the test suite.

5. Conclusions

Both of the proposed high-level interface options for the Automation Messaging Tool satisfy the minimum requirements to be considered acceptable solutions. They allow for data to be interpreted by the Java program and for messages to be entered in the same manner, regardless of what system they are being sent to. They also allow for multiple instances of a single message type to be sent to a specific system at one time.

Using the estimations made by the Deloitte team regarding the amount of time necessary to implement the different solutions, the Excel files interface will take more than double the amount of time to put into place. This is mainly due to the complexity required in extracting the information from the Excel files via the Java program. Compared to simply retrieving the messages from directories, as is done in the XML files method, having the test data in spreadsheets means significantly more programming and testing is required.

Both of the interface solutions incur a runtime cost of linear time. Both of the options require parsing the request and expected messages. However, in instances where the Excel spreadsheets are very large, which is a likely scenario since nearly 200 messages will need to be sent to the different systems, the runtime of the Excel solution becomes noticeably longer. Since the runtime is a function of the columns and the rows, the first spreadsheet and any message-type specific sheets that are frequently used are likely to cause some degradation in performance.

The XML files interface option requires testers to manually edit XML files and place them in the appropriate directories. This translates into nearly 400 hours of manpower that Goodbank's testers will have to devote to entering test data. The Excel files interface is significantly less time consuming to use and only requires editing in one file. Estimated at close to 150 hours of work required to enter data, it is more than two and a half times more efficient than the XML files solution.

6. Recommendations

Due to the importance of having the end users of the interface as content as possible, the Excel files implementation should be chosen. Goodbank should be informed about how long it will take to develop the solution and, more importantly, that it will take longer than the XML files interface option because it will make their testers' jobs' a lot easier in the future. Goodbank will only have to invest 150 hours of work into entering data, which means that can assign either less time or fewer resources to accomplish the one-time entering of data. The Deloitte team should immediately begin development on the interface, the Java component necessary to interact with it, and the message-type specific XML templates. They should also speak to the Goodbank testers to see if there are any suggestions they have regarding the Excel files that will be created. While it is unlikely that any of the testers who will use the Automation Messaging Tool are unfamiliar with Excel, this should be verified. After it is developed, the Deloitte developers should run a tutorial for the Goodbank testers to inform them how to fill in the workbook and begin the test runs.

If Goodbank indicates that they want an interface developed sooner than the time it will take to develop the Excel files interface or they want the program to execute quicker at the cost of having the testers doing more work, then the XML files interface should be implemented. In this case, Goodbank will need to be made aware that it will require more manual editing of XML files for their testers in the long run. It also means that the Goodbank testers will have to be made familiar with the requirements of placing the XML files in the correct folders and any other issues, perhaps via a user guide.

References

[1] K. Li and M. Wu, *Effective Software Test Automation: Developing an Automated Software Testing Tool*, Sybex, 2004.

[2] A. Khan, “Java Excel API - A Java API to read, write and modify Excel spreadsheets,” April 2006; http://www.andykhan.com/jexcelapi/. Last accessed May 4, 2006.

[3] A. Khan, “Sheet Interface – JavaDoc”, http://jexcelapi.sourceforge.net/resources/javadocs/2_6/docs/index.html. Last accessed May 4, 2006.

[4] UNL Shared Computing Services, “The Basics of Microsoft Excel”, August 2005; http://itg.unl.edu/resources/documents/ExcelBasics.pdf. Last accessed May 4, 2006.

Acknowledgements

I would like to take this time to acknowledge two of my teammates, Elaine Oei and Joshua Guo. They assisted me in achieving a good understanding of the current processes used at Goodbank as well as the Automation Messaging Tool that was about to be implemented. They also came up with some very helpful suggestions for both the XML file and Excel file interfaces.

I also wish to thank Frankie Stephan for proofreading my report and ensuring that it adhered to all of the report guidelines put forth by the Software Engineering department.