Atrus AED Maintenance

A technical report of the AED maintenance tracking and reporting features of the National AED Registry project.

Introduction

First, a brief description to provide context about the key business problems that must be solved and the current approach to solving them. Next, review the user reported issues. Then, an analysis to identify the parts of the system that are involved. Further analysis is performed to validate user claims, identify issues, and search for clues that will help implement solutions. Next, an in-depth look at the proposed solutions. Finally, a summary is provided and conclusive statements are given.

1.0 Background Information

Atrus is supposed to help keep track of registered AEDs and the expiration dates of their associated batteries and electrodes. Site-managers and inspectors assigned to AEDs should be notified, on interval, when AED components are soon to expire. The time between notification interval should decrease as the expiration date gets closer. Site-managers, or inspectors, are supposed to inspect the AED, update the AED battery/pad data, and then file a maintenance report indicating the AED components have been replaced, and maintenance issue has been resolved. Upon filing this report, the system should resolve the maintenance issue and cease notifying associated parties.

There is some confusion among registered users and a disconnect in the process of replacing components and resolving maintenance issues created by Atrus. Clients are complaining about receiving messages even though the AED is in good standing. The issues reported by the users are vague and not very descriptive. However, the general consensus among users is that notifications are being sent out when the AED is in good service. It is not clear whether or not the actual state of system reflects the issues reported by users. Their claims will have to be reviewed the data inspected.

2.0 Review User Reported Issues

One user, jane.tokar@rolchurch.net, reported:

```
“I tried several times to update your
records but get booted out every
time.”
```

Jane, did not specifically mention receiving email notifications.

Another user, Janell.Senft@rqhealth.ca, reportedly filed a maintenance report for each AED that was updated. She stated her homepage is showing 5 AEDs with expired batteries, and the maintenance log only shows 2. Customer service reported 5 AEDs with expired batteries and 2 with expired electrodes.

Another user, lward@devoeauto.com, specifically stated that they are receiving notifications every day for expired pads that they have already been replaced. This user also complained about not being able to file a maintenance report.

3.0 System Analysis

There are three main parts of the system which deal with tracking AED component expiration dates and notifying associated parties. These are:

- a scheduled job in the Atrus database labeled, “Atrus AED Maintenance Notifications”
- an asynchronous message queue and queue processor in the project labeled

"Messenger"

- the MaintenanceReportEdit presenter contains most of the logic for processing maintenance reports.

The Atrus database has a scheduled job that executes 5 stored procedures, which collectively generate maintenance issues and populate a message queue with email notifications. These email notifications are picked up by a message queue processor and sent out via an SMTP relay (In this case a third party, SendGrid is used). The message processor will continue to send out notifications as long as they are being generated by the scheduled job in the Atrus database. Only after a maintenance report is filed, will the maintenance issue be marked as resolved. Then the batch job will no longer queue up email notifications for the maintenance issue.

3.1 Scheduled Job

The Atrus AED Maintenance Notifications job is scheduled to execute every day at 06:00 with no exceptions. This job sequentially executes 5 stored procedures. These are:

- pBatchMaintenanceIssueCreate
- pBatchElectrodeExpiredMessageCreate
- pBatchBatteryExpiredMessageCreate
- pBatchSiteManagerReminderCreate
- pBatchInspectorReminderCreate

These are configured to run as a transaction, the results of which will only committed if all the stored procedures execute successfully. Each of these procs require a single parameter, @session_id, the value of which is hard-coded throughout them. An additional stored procedure is executed by the others:

- pSessionKeepAlive

This proc serves to support the other stored procs in the transaction by maintaining session data necessary for sanity during the transaction run-time. It does things like validate and update the special session that is used for this transaction.

3.1.1 pBatchMaintenanceIssueCreate

Executes pSessionKeepAlive, and if execution completes successfully, begin a transaction. Then, a record is created in the MaintenanceIssue table for every battery with an expiration date less then 90 days from the current date. The same for electrodes. The results of each operation is tracked and neither is committed unless both complete successfully. Otherwise the transaction is rolled back.

This stored procedure may attempt to to too much in one query. An insert statement uses a subselect, with a predicate contingent upon NULL values to determine which expired components to create maintenance issue records for. It is generally a bad practice to rely upon NULL values.

For more information on this stored procedure, and a simplified example designed to more simply illustrate the design, see this sqlfiddle:

http://sqlfiddle.com/#!3/d7bb4/2/0

The notes and SQL necessary to reproduce it are in the attachment.

3.1.2 pBatchElectrodeExpiredMessageCreate

This proc first executes pSessionKeepAlive, beginning transaction if successful. Next, it declares a number of variables used for templating the email notification. Then, it begins to update the MaintenanceIssue table setting the NotificationFrequency to 7 day where the ExpirationDate is between 30 and 90 days out.

Next a cursor is declared. This is used for iterating over a collection, which is created from selecting all the open maintenance issues where:

- the maintenance issue type is for Electrodes
- the LastNotificationDate + NotificationFrequency <= the current date
- an Inspector is assigned to the associated AED

The cursor, containing a collection of electrode maintenance issues with an inspector assigned that have not had notifications go out recently, is then iterated upon. During this process an email body template is created and information associated to the current AED is retrieved. Then, pAsyncMessageInsert is executed, effectively adding a message to the async message queue with a status code of 1 (queuing – not ready for delivery). pBatchExpiredVariablesInsert is then executed, which populates the email template with the data previously retrieved in the current iteration.

Next, some additional business logic, embedded in the cursor iterator, is executed. Basically, @days_to_go is created by calculating the difference between the current date and the expiration date. If the days to go is less than 90 days from the expiration date, and the inspector is not the site-manager, an additional message is created on the queue for the site manager.

Next, the message is “closed” by calling pAsyncMessageVariableInsert, effectively changing the status from queuing to queued (ready for delivery). Finally, the MaintenanceIssue is updated setting the LastNotificationDate to the current date.

The result of executing this proc:

- updated MaintenanceIssue to reduce the notification frequency to 7 days where the expiration date is between 30 and 90 days.
- email notifications are created and queued for inspectors (and site managers, if the inspector is not the site manager and if the expiration date is less than 90 days from @days_to_go)
- update MaintenanceIssue set LastNotificationDate to the current utc date

3.1.3 pBatchBatteryExpiredMessageCreate

pBatchBatteryExpiredMessageCreate is almost identical to pBatchElectrodeExpiredMessageCreate. The same procedure is performed with only slight modifications to the variables which indicate the maintenance issue is for a battery. In other words. Additional messages are generated and queued up for battery related maintenance issues.

The after executing this routine:

- messages created and generated according to the same rules as described for pBatchElectrodeExpiredMessageCreate.
- Now there are messages, for both expired electrodes and batteries, waiting to be processed further.

3.1.4 pBatchSiteManagerReminderCreate

As with the other stored procs in the scheduled job, pSessionKeepAlive is executed and only if it completes successfully, is a transaction is begun. Next, various variables are declared for necessary data. Notable variables are the email body, site manager id, the principle id of the site manager, the site id, and additional variables for the email.

Again, the cursor iteration pattern is used. Here a collection is created from selecting the site managers with the following criteria:

- active account
- SiteManagerReminderFrequency + LastReminderDate <= current utc date

During each iteration an email is created reminding site managers to update their AED data in the system if any changes have been made to the registered AEDs.

These messages is not related to expired parts or maintenance issues. They are reminders for site managers to keep updating the data associated to their registered AEDs to reflect any changes happening in real life.

As with the previous message creating procs, messages are templated and queued. And, only if the the proc executes successfully will the results be committed.

The results of executing this subroutine:

- Additional messages are queued up for site manager reminding them to keep on top of their data.

3.1.5 pBatchInspectorReminderCreate

This proc is almost identical to pBatchSiteManagerReminderCreate. There are two notable differences. The first being, the collection defined for the iteration cursor is selected from Inspectors. Second, only inspectors with AEDs associated to them will get a reminders generated for them.

The results of executing this subroutine:

- Additional reminders queued up for the inspectors who have AEDs associated to them.

3.2 Messenger

The Messenger component is a ServiceBase application that consists mostly of a MessengerService and QueueMonitor, and a Supervisor. The service base derivative, AtrusMessenger, starts and stops the service core, MessengerService.

MessengerService, when started generates a MessageQueues list from each queue defined in the applications configuration, and adds each item in this list to an _activeQueue. Then, for every MessageQueue in _activeQueues a QueueMonitor is created. The MessengerService is designed to be multi-threaded. So, a thread is created and started for each QueueMonitor that is created. MessengerService is also designed to use events to signal start and stop events so that the QueueMonitor threads can be managed.

The Supervisor watches the QueueMonitors and if any of the threads die, or otherwise error out, it is replaced with a new thread. The QueueMonitor Thread processor, or MonitorThreadProc, is the worker method that is used to start the thread. This is where the messages are processed, with the help of a data access object that was created with the thread.

While a thread is running, the process will attempt to get messages from the database and deliver them using the SendGridSendHtmlMail method. The process calls GetNextMessage(), which in turn uses the dao to execute the database stored procedure, pAsyncMessageSelectNext, and then returns the results. This stored proc selects the first record in the AsyncMessage table, where the status is queued and the QueueID matches the QueueID specified in QueueMonitor thread configuration. Next, it updates the record, setting the status to 3(sending). Finally, it returns the values for the @message_text, @sender_address, and @subject variables for processing.

Back in the GetNextMessage() call, the message recipients are retrieved by way of calling GetNextRecpient() which in turn executes pAsyncMessageRecipientSelectNext as many times as necessary until there are no more recipients associated with the current message.

After the message and it's recipients are retrieved from the database, the email is constructed and sent out via SendGrid one at a time for each recipient. At this point the recipient is marked as "finished" indicating the message was delivered. Finally, the stored procedure pAsyncMessageRecipientUpdate is executed marking the message as either delivered successfully, or failed, depending on whether finished was called with true. Failed messages will not be re-queued.

3.3 MaintenanceReportEdit Presenter

The MaintenanceReportEdit presenter is the controller behind the page for filing maintenance reports. The logic for processing the maintenance report creation request is contained mostly with the MaintenanceReportEditPresenter. There are a number of caveats in the process of filing a report which users muse be aware of.

There are two perquisite conditions which must be met for a report to be filed successfully. First, the correct Maintenance Issue Type must be selected. For examples, if a battery was expired then IsBatteryChange must be checked indicating the maintenance issue was for an expired battery. Second, the AED must be updated to reflect the new expiration dates prior to filing the maintenance report. If the component data was not updated first, the request will fail and the user will be presented with an error. Only if the expiration dates have been updated, and the correct maintenance issue type selected, can a report be filed for a maintenance issue.

4.0 Analyzing User Reported Issues

Issues reported by the user will be researched by looking into the database and website to review the circumstances of their claims.

When tracking maintenance issues in the database:

- examine relations for maintenance issues, expiration dates, AEDs, components, and important dates.
- select key data related to the AsyncMessage created for the maintenance issue. Examine the message ids, dates, recipients, and status.

Since we have the algorithm to generate message authentication codes for Atrus, we can hijack the user account in a staged environment by replacing the kept hash with one of our creation. This method is very useful because it allows us to log in as the user and test their claims directly, seeing exactly what they would see.

Attempting to generate a replacement password hash, using a well known native sql, I uncovered a potentially undesirable flaw in the code that is used for computing the hash from the user provided password. The hash generation code used by Atrus drops the leading 0 on low hex values when building the hash string from a binary array. For example, 01 will be appended to the hash string as 1, 09 will be appended as 9, and so on.

See the user issue sql for more details, and example queries to generate the hash using sql.

4.1 jane.tokar@rolchurch.net

This user reported that they tried several times to update the records but kept getting booted out. Perhaps, they are waiting too long to to submit the form and their session is timing out. Or maybe the user did not follow the proper steps and update the battery or electrode information before filing a report.

Selecting all the data in the Principal table for this user yielded some interesting clues: The user has not logged in since 11/12/2012 3:31:52 PM. Their account is active, their registration was confirmed, and the legal agreement has been accepted. Judging by these factors, the session handler is not going "boot" them out of the system.

Things to check:

- application logs for all activity related to the user. Maybe there are clues in the logs such as error messages or other exceptions that would prevent the user from saving AED related data.
- Find the AEDs associated to the user
- Look for any maintenance issues in the database for the AED registered to the user.

In a staged environment attempt to log in as the user and update their AED information.

4.1.1 Database Analysis

The user, id 1465, is the site manager of Site 1221. The last login date was 11/12/2012 3:31:52 PM. There is only one Location, id 2972, associated to the site. The Location description is River of Life Church and it has one AED, id 3031, registered to it.

The Battery associated to this AED, id 3046, is expired as of 2/19/2007 12:00:00 AM and has never been updated.

The Electrode associated to this AED, id 5472, is expired as of 2/19/2007 12:00:00 AM and has never been updated.

There are two MaintenanceIssue records for the AED, one for each of the expired components listed above. The ids for the maintenance issues, respective to their positions above, are 5346 and 5375.

There is no relation in the database between AsyncMessages and MaintenanceIssues. The only way to find notifications related to the maintenance issue, is to correlate the time stamps between the AsyncMessage and the MaintenanceIssue. This analysis can be done by first pulling all the maintenance issues for a particular principal id and then cross referencing the closed date of the maintenance issue to the modified date of the components associated to that user. In other words, get the principal id from AsyncMessageRecipients table. Get all the components associated to the AEDs for the principal id where the modified date is the same around the same time as the closed date. Optionally, you can search for the email address/login using like clause on the

RecipientAddress column, and narrow in on the affected component that way.

At no times does the user appear in the AsyncMessageRecipients table. This is particularly discerning because the user should have at least received email messages related to registering the system. It is possible that their account was setup by hand.

4.1.2 User Experience Analysis

After logging in as the user and reviewing the situation it was clear the user experience reflected the same state as discovered in the database analysis. There were two maintenance issues, one for the battery and one for the electrode. I was able to update the components and file the maintenance reports without any problems.

However it should noted that a bug exists in the procedure for filing a maintenance report. The default option, “Send a copy of this maintenance report to the site manager”, is checked and this does generate an error on occasions where the site manager does not have an email address associated to them, that will prevent a maintenance report from being filed. The work around this is to un-check that option. The More on that issue in

4.1.3 User Issue Resolution

Without analyzing the application logs, it is unclear what error, if any the user encountered. It seems like all their claims were baseless, stemming from frustration and a lack of understanding for the Atrus AED maintenance process. The user should be advised of the proper procedures and walked through the process.

Additional analysis should be performed to determine why there were no async messages in the database for this user. These may have been deleted previously by another technician or developer, or it's possible the user's account was provisioned by manually by setting values directly in the database.

See attachment 'user issue analysis 1.sql' for sql queries related to conducting this analysis.

4.2 Janell.Senft@rqhealth.ca

The login for this user is actually, pad@rqhealth.ca.

This user reported 5 maintenance issues were listed on the home page, however only 2 were seen on the maintenance issue page.

A quick look in the database indicates that the user has many AEDs associated with them. It may be better to start with the User Experience Analysis to narrow down the list of affected AEDs. The same technique, as with the previous case, of temporarily replacing the password hash can be use here as well (in a replicated environment, not production).

4.2.1 User Experience Analysis

The SiteManager home page, when logged in as the user, indicates there are 5 AEDs with expired batteries. The maintenance log shows 4 open maintenance issues. Two, one for an expired electrode and one for an expired battery, belong to the same AED. The other two, an expired electrode and an expired battery, belong to two different AEDs. So, the maintenance issue page indicates 3 AEDs with maintenance issues but 4 open maintenance issues. The View AEDs page indicates 5 AEDs with expired batteries, and 3 with expired electrodes, for a total of 8 would-be maintenance issues. Compared to the value 4 in the maintenance log.

The first AED reviewed was Deloitte and Touche. This is the AED with two open maintenance issues, one battery one electrode. By clicking “Edit Installed Battery” it was determined the expiration date of the battery was 2/28/2013. The battery is not yet expired. The same holds true for the electrode which also has the same expiration date. Since neither of these components are expired, it would better to list this as such in the maintenance log.

The next AED, at Orr Center, is reporting an expired electrode. This AED has two electrodes associated with it. However, only one is installed. The expiration date for both electrodes is 1/28/2010. Maintenance is long over due and associated parties should be getting daily email notifications regarding this AED. The battery is in good standing.

The final AED, reported to have an open maintenance issue on the maintenance log, is located at Lakeview Golf Course. This AED is reporting an expired battery. The expiration date listed in the battery details is 11/10/2012.

4.2.2 Code Analysis

The site manager home page has, among others, public properties with getters and setters for both AEDExpiredBatteryCount and AEDExpiredElectrodeCount. These values are set in the OnViewLoad event handler in the SiteManagerHomePresenter. The values are set when the presenter loads by calling Site.GetSiteSummary(), which in turn calls the stored procedure pSiteSummarySelect. This stored proc is not very complex. It simply selects the count of AEDs, associated to the site, with an expiration date in the past.

4.2.3 Database Analysis

First check the MaintenanceIssue table. Find all the open maintenance issues for the site. Next, find all the expired components. Correlate expired components with the open maintenance issues. Determine why there are not maintenance issues for expired components that were not found in MaintenanceIssues.

Inspecting the data in the MaintenanceIssue table showed the same open issues as listed on the open issues section of the maintenance log. There are only 4 maintenance issues, as listed previously. The sql query to select this data uses a join on the Location data, effectively only displaying results for the given site. In other words, for all the AEDs associated to this site, select the details of any open maintenance issues.

Next. The Battery data was inspected. All batteries with expiration date in the past were selected for the given site. This yielded 5 results, only one of which corresponds with the maintenance issues. 3 of the 4 open maintenance issues have now been verified. However, this count of 5 does correspond with the values displayed on the site manager home page and the view AEDs page. An obvious question arises here, “Why are there expired batteries for which there are no open maintenance issues?” To answer this, pull all the maintenance issue records for AEDs associated to expired batteries. Do the same for electrodes.

Pulling all the maintenance records from the last 90 days, only for expired batteries, yielded some interesting results.

The reason why maintenance issues are not being created for the additional expired batteries is because of the SQL statement that is used to generate maintenance issues. The query in the stored proc does an insert from a select. In other words, the results of a select statement are inserted into the MaintenanceIssue table.

The select statement includes a join subclause, which matches maintenance issues to batteries on 3 predicates:

- AEDID value between joined tables match
- ExpirationDate value between joined tables match
- MaintenanceIssueTypeID matches 2 (for battery)

The integrity of the maintenance issue to battery relation is based upon the premise that maintenance issue data should not exists for relations that do not have a matching ExpirationDate. This is because the where clause stipulates that the MaintenanceIssueID should be null. The system expects, since the ExpirationDate of the battery matches that of the MaintenanceIssue record, they are the same maintenance issue. This presumption is made because the system should not be able to close a maintenance issue (that has an id) without first updating the expiration date of the battery, thus breaking the match in the MaintenanceIssue/Battery relation. The value for mi.MaintenaceIssueID is expected to be NULL since the consequence of using a left join operation on a relational set is resulting NULL values on the right side of the join where there is no matching row. In other words, since no maintenance issue records match the join on predicate, the maintenance issue data will be marked NULL.

See attachment 'user issue analysis 2.sql' for sql queries related to conducting this analysis.

4.2.4 User Issue Resolution

If is unclear how this data came to be in a broken state such as it is. The solution is to update the expiration dates of the affected AEDs. Why is the scheduled job not creating maintenance issues for the expired batteries? The answer is because the left join is able to match battery data (left side) against maintenance issue data (right side). Since maintenance issue data is on the right-side of the left join, it is expected to have null maintenance issue data. This is based on the premise, built into the system, that maintenance issue expiration dates should be older than component expiration dates. Inferring that a maintenance issue record with an expiration date matching that of the joined battery, and an id, is an in-sane record and is not factored into the MaintenanceIssueCreate procedure.

4.3 lward@devoeauto.com

This user reported they are receiving email notifications daily for components which have already been replaced.

All the open maintenance issues were pulled for the AEDs associated to the site. Cross-referenced with the components associated to the AEDs. The user has 4 open maintenance issues, all for expired electrodes. This was reflected in the database and the user interface.

One AED, 17cf63f1-c512-4e15-a88f-7798f9d1c01b Devoe Corporate Office, had two electrodes associated with it. One of which was expired.

A new bug was discovered, as mention in the first user analysis, which will prevent users from filing maintenance reports. This is most likely the issue that the user was trying to explain in his emails.

4.3.1 User Issue Resolution

It was not necessary to conduct much analysis for this issue. It was immediately obvious there were four expired electrodes which corresponded 1-to-one with the open maintenance issues. When replacing the user's password hash and logging in as them, it was possible to update the Electrodes associated to all the AEDs.

It was necessary to delete some expired Electrodes that were not installed. This is because multiple expired components are currently not supported by the current implementation of the maintenance tracking system. Specifically, the issue is caused the code in IsElectrodeExpired() and IsBatteryExpired() in the AED class of the Registry.Model. Here the collection of components associated to the AED is iterated upon and if any of the components are expired, the method returns false. Thus indicating the maintenance issue cannot be filed.

Users are currently not permitted to have multiple expired components associated to an AED. However, deleting batteries or electrodes can cause data-integrity issues as the database grows. One example is that maintenance issues will exist for components that were deleted. If a battery is deleted, the maintenance issues associated to that user will be left around. This is because the relation between components and maintenance issues is indirect.

In summary, removing the extra not installed electrodes, updating the expiration dates, and filing maintenance reports for each expired component cleared up all maintenance issues. However, it was necessary to de-select the email site-manager option when filing maintenance reports.

5.0 Conclusion

The AED Maintenance tracking features in Atrus are implemented in a number of loosely coupled components who share state via the database. The Web application, Messenger service, and Daily batch job are the primary components which are responsible for tracking expiration dates and notifying the associated parties. There are a number of shortcoming, in the system architecture, where further research and optimization is required. There were bugs identified in the system, for which hot-fixes need to be rolled out immediately. Furthermore, and perhaps most importantly, the user issues need to be resolved in production.

5.1 Hot-Fixes to Roll-out

There are two primary hot-fixes that should be rolled out ASAP. The first is to fix the bug identified in user analysis 1. The second is to modify the Is*Expired() calls in the AED model to not match against AEDs that are not installed.

5.1.1 Fix 1 – NULL user data

The first fix deals with the bug, discovered in user analysis 1, where an error is displayed in some cases when opting to email the site-manager when filing a maintenance report. The error can be reproduced by creating a site manager account but don't edit the profile. Leave the email address and name values NULL. When filing a maintenance, after updating an expired component, select the option to email the site manager. Since the FirstName and LastName are NULL and error is generated.

The problem is in the stored procedure pAsyncMessageInsert, starting with the select statement on line 63. This select is used to set the value of the recipient string. However, in cases where NULL values are provided for FirstName or LastName, the result will be NULL, rather then the value for Login as expected. This would explain some inconsistencies spotted between the AsyncMessage data, maintenance issues, and the LastNotification date on the AsyncMessageRecipient.

A proper solution would involve developing a more robust routine to generate the recipient address string. However, that has not been done at this time. As a temporary work around, users can be advised of the issue and to have their site-managers update their profile data to reflect accurate information. A more long term solution would include making it requirement to provide this information during the registration process.

5.1.2 Fix 2

This problem is with the AED model class in the Registry.Model project. This class contains two methods, IsBatteryExpired() and IsElectrodeExpired() which iterate through every battery or electrode associated to the AED and returns false if any one of them are expired. This needs to be updated to not return false if the AED is not installed. This will ensure that users with multiple batteries registered don't have to keep updating the same record and the system can keep a history of the actual batteries.

5.2 Resolving User Issues in Production

These are the actions that need to be taken to finally resolve the user reported issues in production.

5.2.1 jane.tokar@rolchurch.net

There was no real problem, other than the issue fixed in a previously explained hotfix. This user need to be walked through the process of resolving maintenance issues.

5.2.2 Janell.Senft@rqhealth.ca

First this user should update the AEDs and clear the current maintenance issues. Then, the user must search for and update the AEDs associated to the following locations:

- Dr. Paul Schwann Centre
- Corp of Commissionaires
- Lakeview Golf Course
- Greenall High School
- Kipling School
- Travelodge
- St. Peter School
- Orr Centre

Any extra, not installed, expired components should be deleted. All other expired components should be updated.

5.2.3 lward@devoe

This user should be advised to fill out the site-manager profile, including the FirstName, LastName, and EmailAddress. Then customer service should walk the user through the current procedure for resolving maintenance issues. As with the previous resolution, any extra, not installed, expired components should be deleted. All other expired components should be updated.

5.3 Attachments

Supplemental this report is a zip package that contains the following:

- database backup from 01/14/2013
- separate sql file for each user issue analysis
- and sql files for sqlfiddle to simplify the maintenance issue creation problem
- http://sqlfiddle.com/#!3/d7bb4/2/0