Situational Awareness on Android
Team SAndroid

Evan Charlton          Mike Roberts          James Tymann          Brian Wong

Harris RF Communications
Nate Ransom
Tim Lee

Faculty Coach
Tom Reichlmayr
Project Overview

Harris RF Communications designs, tests, and manufactures military radios. The United States Department of Defense has recently expressed an interest in cell phone apps, specifically those built on Android. The goal of this project will be to create several military-focused applications on the Android Platform. Situational Awareness Proximity and Tactical Chat are two separate parts of an application that offers ease of communication. Situational Awareness Proximity will allow the user to easily locate, and begin chat with other devices. Tactical Chat will allow the user to communicate with other users on the same application. The GPS information gathered from other phones for the location data will remain available as a service to future applications that may build upon this.

Basic Requirements

The system as a whole had requirements which were associated to neither component

- Must run on Android 2.2 and newer
  - In addition to working on Android 2.2-compliant devices, the application needed to be demonstrated using the Harris-provided Google Nexus One devices (which were running Android 2.2)
- Must expose collected data for third-party applications to use
  - Third-party applications must be able to access the collected data in a standardized format.

The situational awareness component ("SA Prox") had several key requirements:

- Interoperate with existing Harris system
  - Application must be able to parse the data from the existing MITRE-created Cursor-on-Target data format which Harris’ radios use.
  - Application must store all parsed information in a way which will let other applications use it.
- Display devices on a radar view to show relative bearing and location
  - Radar view must orient correctly to reflect the device’s current compass orientation
  - Radar view must be able to be zoom-controlled
  - Radar view must display devices with correct bearing
  - Radar view must display devices with correct relative distance
- Display devices on a Google Maps overlay to show absolute location
  - Devices must be displayed in their correct positions on top of a Google Maps view
  - User’s current location must be show on the Google Maps view
The TacChat portion of the application required the following functionality:

- Interoperate with existing Harris system
  - Application must transmit and receive TacChat data packets which conform to the TacChat protocol (supplied by Harris)
  - Application must transmit and receive data packets which conform to the FCP data transfer format (supplied by Harris)
- Store TacChat addresses
  - User must be able to manually enter TacChat addresses
- Manage TacChat addresses
  - User must be able to edit stored TacChat addresses
  - User must be able to delete stored TacChat addresses
- Conduct a text-based chat with a chat partner
  - User must be able to send and receive text-based TacChat messages to and from other Android phones running this application
  - User must be able to send and receive text-based TacChat messages to and from PCs running the Windows-based TacChat application (supplied by Harris)
- Send and receive files via TacChat protocol
  - User must be able to send and receive files from other Android phones running this application
  - User must be able to send and receive files from PCs running the Windows-based TacChat application (supplied by Harris)

**Constraints**

The main constraint with this project was the fact that it needed to run on Android 2.2. This presented a hardware challenge since some members on the team did not own Android phones. Furthermore, since this application required device-to-device communication, each team member would need more than one phone. Furthermore, the application’s domain dictated that the system use features unavailable in the Android emulator (such as GPS, compass, and accelerometers), which meant physical devices were required. Harris was able to supply official test phones (Google Nexus One) and the faculty coach was able to supply an adequate number of development phones (Motorola Droid).

In order to deliver the situational awareness component correction, the application needed to integrate with Harris’ existing systems. These radios conformed to the Cursor-on-Target specification which was developed by MITRE. The Cursor-on-Target specification is an extremely large specification which conveys information about device positioning, types, bearing, routing information, and much more. The application had to be able to handle this wire format. Fortunately, Harris only uses a subset of this information, so the application does not have to handle every case specified in the format. Harris supplied an example packet which was captured from one of their radios, and this was used as a reference implementation for development. However, it was critical that the application be able to receive any valid Cursor-on-Target packet and be able to pull out as much desired information as possible without crashing.
The TacChat portion of the application presented an entirely new set of challenges. The first issue which was encountered was that there were contradictions within the specification itself. For example, some fields were labeled as having one role, but then the description of that field described a different role. Additionally, there were unspecified implementation details, such as the fact that the Windows-based TacChat client performs a GZip operation on outgoing packets in order to minimize traffic. Finally, the last major hurdle we encountered was due to the communication between different implementation languages. The Windows-based implementation was written in C/C++ (which has support for unsigned types) and this application’s implementation was written in Java (which does not support unsigned types).

Another constraint with TacChat is that there were actually two separate protocols to implement, and both behaved very differently. The TacChat protocol describes the actual chat information (such as chat messages, files, and so on). This data is then encoded into Harris’ proprietary FCP protocol, which is used to actually exchange data across a low-bandwidth, low-reliability network. It may be helpful for the reader to think of FCP as a reimplementation of the widely-used TCP/IP communication protocol, but with lower overhead requirements.

The final constraint which needed to be considered when designing the TacChat system was that the system needs to be easily upgradeable. This was tested in the development of the application, because the version of the TacChat protocol specification documents which Harris provided was version 0, but it was later discovered that their Windows-based TacChat client was using version 2. Due to the design of the system, upgrading the parser was straightforward and didn’t cause much of a delay.

**Development Process**

For this project, the development methodology was an implementation of Scrum. Since high numbers of requirements changes were not expected, many development processes would have been applicable, such as an incremental waterfall model. However, since there were large unknowns around the two systems and their protocols, one of the risk mitigation strategies for this was to go with a quick iteration model which would also allow for the flexibility of negotiating requirements if necessary.

The scrum implementation used was based around two-week sprints. These short sprints allowed for a high level of client visibility since a functional deliverable was ready at the end of each two-week sprint. At the end of each sprint, there was a sprint review conducted with the sponsor, where the deliverable for the past sprint was explained, demonstrated, and feedback was given. Then, the upcoming sprint’s plans were explained and feedback was sought.
The scrum methodology requires daily stand-up meetings. These were performed every weekday, though some were virtual meetings. On Monday, Wednesday, and Friday, the team would hold a conference call at a time when all team members were available. On Tuesday and Thursday, a stand-up meeting would be held in person prior to the start of the regularly-scheduled meeting. This constant contact helped the team stay updated and informed about what the rest of the team was doing.

The team adopted the following roles:
James Tymann - Scrum master, benevolent dictator
Evan Charlton - Tech lead
Mike Roberts - Testing coordinator
Brian Wong - Sponsor point of contact

Code reviews One of the cornerstones of the development process was mandatory code reviews. To enforce this, the Gerrit code review tool was used. Gerrit uses the Git source control system, and allows for asynchronous code reviews. Through Gerrit, team members were able to review changes on their own schedule and provide feedback, verification, and approval.

For each change, there were two requirements which had to be met. The first is that each change had to be verified. This means that someone pulled down a clean copy of the change, compiled it, installed it, and ran all the tests. If all of the tests passed, the change was marked as verified. Next, the code had to be approved. This meant that a team member other than the submitter read through all of the code, understood it, and agreed with all of the changes. Once this was completed, the change was marked as approved. Until the change had both verification and approval, Gerrit would not let the change be merged into the repository. It should be pointed out that Gerrit makes it impossible for code to get into the master repository without having gone through the review process.

The other major component of the development process was the use of the JIRA issue tracker. When combined with the GreenHopper plugin (which provided agile project management), this tool suite proved to be invaluable. This system ensured that bug reports were never lost, tasks never forgotten, and progress accurately tracked. Furthermore, the system managed all of the process artifacts such as the task board, burndown charts, and backlog. JIRA and GreenHopper removed much of the manual labor involved with the process and allowed the team to focus on product development.

**Project Schedule: Planned and Actual**

The initial project plan was two-fold. Since there were two main systems (SA Prox and
TacChat), the goal was to make progress on both fronts simultaneously. The initial plan was to have both systems at least feature-complete by the start of the second quarter, so that the second quarter could be focused on cleaning up the code, refactoring, fixing defects, system integration, and overall application polish.

Initially, the project plan was analyzed and potential risks were identified. The risks which were identified, in order of decreasing severity:

1. Problems with TacChat interoperability
2. Problems with CoT interoperability
3. Misinterpreting project requirements
4. Not understanding business use case
5. Asynchronous communication with project sponsor

In order to mitigate these risks, different strategies were developed in order to prepare for these potential disruptions most appropriately.

The risks identified with the two systems were very similar: not understanding the protocol and specification. As a way to meet these challenges head-on, it was decided to begin work on these as soon as possible. By starting development on these protocols early, any potential problems would be identified as soon as possible, and could be addressed appropriately.

Similarly, misinterpreting project requirements and not understanding the business use case were also related to each other, and could be addressed in a similar fashion. In order to mitigate these risks, a significant amount of time was spent with the sponsor to understand their vision of the system, where and how it would be deployed, why they wanted to have this application, and so on. Extracting this information from the sponsor gave great insight into the project and allowed changes to the architecture, process, and prioritization as necessary.

Finally, most communication needed to happen asynchronously through email. These delays could potentially cause problems if speed of response was important. The mitigation strategy for this was to be in contact with the sponsor on a frequent and regular basis. This was a key driver in the decision to use two-week sprints in the development methodology.

Some of these risks were, in fact, manifested during the duration of the project. The main risk encountered was problems with the TacChat protocol. This protocol is very complicated and robust, and this was underestimated. Implementing the protocol itself, let alone ensuring interoperability with the existing Windows-based TacChat implementation, turned out to take more time than previously anticipated. However, since this had been anticipated, it was possible to adapt and not be caught off-guard.

**System Design**
The overall system architecture can be seen in Figure 1, Appendix.

Both the SA Prox and TacChat architectures are split into three tiers: background services, content providers, and the user interface. The background services handle sending and receiving of data over the network as well as communication between devices. Data from the services is handled and stored by the content providers; this includes information about the position and type of other devices as well as a database of network addresses and messages. Finally, the user interface observes data from the content providers and displays it to the user, updating screens and views according to received data.

SA Prox is currently split into two different services, one for receiving situational awareness data (SA Prox Receiver) and one for broadcasting (SA Prox Broadcaster). The situational awareness data is received in the Cursor-on-Target (CoT) format. The services are split into a broadcaster and receiver so that they can be separated easily (for example, if receive-only functionality is desired the broadcaster can be decoupled). These two services interact with a central CoT Processor to encode and decode CoT data to and from a standardized format. This means that users can interact with the system without having to re-implement the CoT format.

All of the received data is stored in databases, which are managed by Android’s content provider framework. By hooking into the native Android content provider framework, the delivered application can insert and receive data in a standardized format. This means that developers familiar with existing Android conventions will be able to maintain the application in a familiar way. However, this system has another important benefit.

By placing the aggregated data in storage mechanisms managed by content providers, it can be accessed by third-party applications. Third-party applications have access to the data in exactly the same way that the first-party application does. This provides Harris with the opportunity to expand upon the delivered application’s platform. These additional applications can be developed completely separately from Tactical Compass and run in their own process space.

Finally, the UI is split up in a way that’s similar to the division of the rest of the application: separate stacks between SA Prox and TacChat. However, these two halves are integrated into one seamless end-user experience through the use of quick actions. These quick actions are implemented globally on device icons. When an icon is selected, a context-sensitive, device-sensitive dialog is displayed (anchored to the tapped icon) which allows the user to jump around the application. For example, the user can switch from a TacChat conversation to the Google Maps view and send TacChat messages without ever having to leave the Google Maps view.

It should be additionally noted that the application user interface’s information is pulled from the application’s content providers. This means that third-party applications would receive the same
data and that they would live as first-class citizens in this ecosystem.

**Process and Product Metrics**

In order to track project progress, the team opted to track the following metrics:

- Acceptance tests written
- Acceptance tests passing
- Unit tests written
- Unit tests passing
- Burndown progress

The aggregated metric data (see Figure 2, Appendix) shows the progress of the project as measured by tests and issues resolved. As the number of tests created and passed increases it correlates with functionality implemented thus increasing complexity and quality of the product. The burndown progress shows goals achieved within a particular sprint as issues are complete or pushed off to the next.

The metrics were used to estimate future project progress. For instance, if the last sprint had poor performance in terms of burndown progress, it could be assumed that there were tasks carried over and the next sprint should have a smaller estimate. Additionally, if any of the tests failed then the review process had failed because failing tests had slipped through. Finally, if the test count did not increase proportionally with development, it would be an indication that the application wasn’t being thoroughly tested.

**Product State at Time of Delivery**

At the time of delivery, the project meets all of the sponsor’s requirements. The required functionality of both Situational Awareness Proximity and Tactical Chat have been implemented and interoperability between Harris’ existing radio devices and PC implementation is possible. The application is stable and bugs besides several corner cases have been fixed.

The Tactical Chat protocol is not fully implemented as of delivery; there are features and capabilities within the protocol that were not addressed during development. While interoperability has been tested with a provided version of the Tactical Chat PC implementation, full blanket testing has not been done. There are also several corner cases that are not handled as well as a limitation on the data fields that Situational Awareness Proximity is storing.

Beyond the basic features requested, the delivered application is capable of further functionality. File transfers between clients are possible (although limited in size). Quick actions have been
implemented to facilitate easy access between SA Proximity and TacChat. A unified view of TacChat conversations and messages have been added in order to give the user the option to choose their desired interaction method. Finally, a demonstration application has been included to provide a reference implementation for how third-party applications can access the data stored within the content providers.

While there were no features eliminated, there was some additional functionality included in the deliverable. These extra features were included because of expressed stretch goals which were thought to be feasible (such as file transfers) or due to team judgement (e.g. unified conversation view, quick actions). None of these additional features were included at the cost of agreed-upon functionality.

**Project Reflection**

At the conclusion of the project, several factors were identified as having identified to the success of the project. Additionally, given the opportunity to do it again, several choices would be reconsidered.

The tools which were chosen worked extremely well. The use of Gerrit and Git proved to be a very good choice as it significantly reduced the possibility of having only one person know a particular area. Additionally, Gerrit allowed for many errors and bugs to be caught before they were became defects.

JIRA with the GreenHopper plugin was another great choice. By automating the development process, more time was freed up for actual development instead of spending time generating and maintaining process artifacts. Furthermore, assignment (and reassignment) of tasks was very easy and helpful. Finally, the ability to track progress at a glance was extremely useful.

The division of labor for this project also worked out very well. By developing these two systems (SA Prox and TacChat) independently, it was possible for the team to split up and tackle different sections without blocking each other. This allowed for very rapid development, which was required by the choice of two-week sprints.

The choice of two-weeks sprints was another good choice. These short sprints allowed for a high level of client visibility into the project, without being overbearing. Additionally, it allowed for fast detection of failure and rapid readjustment as necessary.

However, given the opportunity to do it all over again, two-week sprints would be reconsidered. Since these sprints were so short, it was sometimes tough to include visible changes into the
deliverable. For instance, some sprints were focused on backend work (such as the content providers) which meant that the client couldn’t see any changes during the next sprint review meeting.

Additionally, the chosen metrics were under-utilized. The best metric for project progress that could have been tracked was how long it took to review code. Code reviews were the main bottleneck in the development process and was completely within the team’s control. Tracking this metric would have been incredibly useful for gauging project velocity. However, this information could not be extracted automatically from Gerrit, so the metric had to be abandoned.

Finally, the division of labor would be reconsidered. Since developers tended to stay within their comfort zones, there was a missed opportunity to transfer knowledge to other members of the team. This meant that not only was the “bus factor” rather high (though partially mitigated through the use of Gerrit), but valuable learning experiences were missed.
Appendix

Figure 1:
Below is a very high-level diagram representing the architecture of the system. Represented in the diagram are the main components which make up each of the three application tiers.
**Figure 2:**
The aggregated metric data:

<table>
<thead>
<tr>
<th>Sprint</th>
<th>Acceptance tests written</th>
<th>Acceptance tests passed</th>
<th>Unit tests written</th>
<th>Unit tests passed</th>
<th>Burndown progress</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sprint 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>21/22</td>
</tr>
<tr>
<td>Sprint 1</td>
<td>0</td>
<td>0</td>
<td>13</td>
<td>13</td>
<td>30/30</td>
</tr>
<tr>
<td>Sprint 2</td>
<td>3</td>
<td>3</td>
<td>49</td>
<td>49</td>
<td>16/23</td>
</tr>
<tr>
<td>Sprint 3</td>
<td>3</td>
<td>3</td>
<td>49</td>
<td>49</td>
<td>9/10</td>
</tr>
<tr>
<td>Sprint 4</td>
<td>3</td>
<td>3</td>
<td>88</td>
<td>88</td>
<td>22/24</td>
</tr>
<tr>
<td>Sprint 5</td>
<td>3</td>
<td>3</td>
<td>91</td>
<td>91</td>
<td>3/5</td>
</tr>
<tr>
<td>Sprint 6</td>
<td>5</td>
<td>5</td>
<td>127</td>
<td>127</td>
<td>14/14</td>
</tr>
<tr>
<td>Sprint 7</td>
<td>7</td>
<td>7</td>
<td>155</td>
<td>155</td>
<td>12/12</td>
</tr>
<tr>
<td>Sprint 8</td>
<td>8</td>
<td>8</td>
<td>175</td>
<td>175</td>
<td>9/10</td>
</tr>
<tr>
<td>Sprint 9</td>
<td>10</td>
<td>10</td>
<td>201</td>
<td>201</td>
<td>12/15</td>
</tr>
<tr>
<td>Sprint 10</td>
<td>13</td>
<td>13</td>
<td>233</td>
<td>233</td>
<td>8/8</td>
</tr>
</tbody>
</table>