ABSTRACT

Using Geographic Information System (GIS) in crime mapping and analysis is a new approach adopted by many law enforcement agencies. This technical report details the development and the usage of new database-and-GIS software, called Auto Theft Analyst (ATA), which leads police personnel to store consistent data and produce several maps for auto theft crimes. Many cities in the USA suffer from a severe automobile theft crisis, especially those cities close to the Canadian and Mexican borders such as Corpus Christi. ATA is database-and-GIS based software that implements a set of database utilities for storing and checking input data, and integrates another set of GIS activities for mapping manipulation. ATA uses visualization techniques to present maps of any geographical jurisdictions and store auto theft data in a certain pattern. Its purpose is to enhance the process of investigation and decision-making for auto theft crimes by deploying resources more effectively.
# TABLE OF CONTENTS

Abstract........................................................................................................ ii

Table of Contents........................................................................................... iii

List of Figures................................................................................................. v

List of Tables.................................................................................................. vii

1. Introduction and Background.................................................................. 1

2. Computerized GIS For Combating Auto Theft Crime ...................... 4
   2.1 Graphical User Interface (GUI)......................................................... 5
      2.1.1 Login Form........................................................................... 5
      2.1.2 Auto Theft Analyst Main Form ........................................... 6
      2.1.3 The File Menu.................................................................... 15
      2.1.4 The Data Entry Menu.......................................................... 15
      2.1.5 The Distance Calculation Menu........................................ 20
      2.1.6 The Help Menu.................................................................. 21

3. System Analysis and Design................................................................. 22
   3.1 Anatomy of Current Datasets......................................................... 23
      3.1.1 Stolen Data......................................................................... 23
      3.1.2 Recovery Data.................................................................... 25
   3.2 New Database Modeling.................................................................. 28
   3.3 Database Creation and Software Design Patterns....................... 30
      3.3.1 The Connection Pool Technique Used in ATA..................... 31
      3.3.2 The Factory Method Used in ATA....................................... 34
   3.4 MapObjects-Java Standard Edition in Auto Theft Analyst........... 37
3.5 The Geocoding Tool .................................................. 38
   3.5.1 The Manual Geocoder ........................................... 39
   3.5.2 The Geocoder in ArcIMS ...................................... 41
3.6 Distance Calculation Tool ........................................ 43
   3.6.1 Dijkstra’s Algorithm ......................................... 43
   3.6.2 Java Implementation of Dijkstra’s Algorithm ............. 45
4. Testing and Evaluation ............................................. 47
   4.1 Unit Testing ...................................................... 47
   4.2 Integration Testing ............................................. 47
   4.3 Validation Testing .............................................. 48
   4.4 System Testing .................................................. 48
5. Future Work ....................................................... 49
6. Expected Results and Conclusion ................................. 51
Bibliography and References ......................................... 52
Appendix A. Data Dictionary ....................................... 54
Appendix B. Database Schema ...................................... 56
Appendix C. Forms and Reports ..................................... 57
   - Stolen Vehicle Entry Form .................................... 58
   - Offense Report ................................................ 59
   - Suspect Report ................................................. 60
Appendix D. Source Code ............................................. 61
   - ATASystem.java ............................................... 62
   - ATAUDL.java ................................................ 64
LIST OF FIGURES

Figure 2.1 Login Form........................................................................................................... 6
Figure 2.2 Auto Theft Analyst Main Form............................................................................... 7
Figure 2.3 Select Data Source Panel..................................................................................... 8
Figure 2.4 Layer Properties for the Node Layer................................................................... 9
Figure 2.5 Results After Using the Identify Tool ................................................................. 10
Figure 2.6 The Query Builder Associated with the Active Layer.......................................... 11
Figure 2.7 The Buffer Tool................................................................................................... 12
Figure 2.8 The Attributes Panel Generated Output.............................................................. 13
Figure 2.9 The Attribute Table Generated by the Table View Tool....................................... 13
Figure 2.10 Map Tips Dialog Box......................................................................................... 14
Figure 2.11 The File Menu.................................................................................................... 15
Figure 2.12 The Data Entry Menu......................................................................................... 16
Figure 2.13 Add Theft Incident Form.................................................................................... 17
Figure 2.14 Add Recovery Incident Form............................................................................... 18
Figure 2.15 Add A New User Form...................................................................................... 19
Figure 2.16 Add A New Damage Pattern Form.................................................................... 19
Figure 2.17 Distance Calculation Form.................................................................................. 20
Figure 2.18 About Auto Theft Analyst Form......................................................................... 21
Figure 3.1 A view of CCPD Database File for incidents in 2000.......................................... 24
Figure 3.2 The 2001 CCPD Database After Normalization.................................................. 24
Figure 3.3 Unfilled Data Attributes...................................................................................... 27
Figure 3.4 Recovery addresses attribute values in 2000 data.................. 27
Figure 3.5 Entity Relationship Diagram for the Auto-Theft Database Module.... 28
Figure 3.6 Hierarchy of Coding in ATA.............................................. 30
Figure 3.7 The Factory Design Method Concept in ATA.......................... 35
Figure 3.8 Control flow details between the ATA program and its database..... 35
Figure 3.9 The Node Layer Created for The Manual Geocoder in ArcView........ 39
Figure 3.10 The Database Table of the Node Layer................................ 40
Figure 3.11 GeocodingUI API in MapObjects 1.0................................. 42
LIST OF TABLES

Table 1.1 Top 10 Vehicles Stolen in Texas in 2001................................. 2
Table 3.1 Top 10 Vehicles Stolen in Corpus Christi in 2000...................... 25
Table 3.2 Top 10 Vehicles Stolen in Corpus Christi in 2001...................... 25
1. INTRODUCTION AND BACKGROUND

Nueces County ranked eighth in Texas in auto theft offenses with over 1,485 vehicle theft losses in 2000. Corpus Christi reported 677 (45.6%) vehicle thefts throughout Nueces County in 2000 and 745 thefts in 2001. Commander David Griffith of the motor vehicle theft service, part of DPS' criminal law enforcement division stated: “The most popular vehicles out there are usually the most popular vehicles stolen. They serve as a source of parts.” [Fikac 1998]. The goal of this project is to enhance the forensic investigations through the design and implementation of database and Geographic Information System (GIS) tools to reduce auto thefts and further increase apprehension of auto theft offenders. This can be achieved through a comprehensive mapping system that utilizes GIS crime analysis tools to address the hot-spot areas of theft occurrences and to support the development of effective enforcement and prevention strategies.

This approach to auto theft reduction has significantly worked in other jurisdictions. For example, in Baltimore County, Maryland, auto thefts increased at an alarming rate, having nearly doubled during the 1980s. The county’s auto theft rate was 21 percent higher than the national rate. To attack the problem, Baltimore County Police Department began using computer-based mapping techniques to more accurately identify areas of attacks for stealing vehicles. Maps that showed the hot-spot areas were produced with the Spatial and Temporal Analysis of Crime (STAC) program developed by Illinois Criminal Justice Information Authority. Baltimore County’s Auto Theft Squad later targeted those areas for undercover operations and other intensified enforcement activities. This approach produced almost immediate results: in 1992, Baltimore County reported a 28 percent drop in motor vehicle thefts in the targeted areas [Lewin 1995].
Such a strategy could have a major impact on the auto theft problem in Corpus Christi, as well as in Nueces County. Statistics depicts Corpus Christi as one of main cities suffering from an auto theft problem in Nueces County.

With a vehicle stolen every 5.5 minutes in Texas, major cities are deploying more resources to combat auto theft problems. American vehicles, such as GM and Ford, are considered the favorite-targeted vehicles. Table 1.1 shows the top 10 models of vehicles stolen during 2001 throughout Texas [TxDPS 2002].

<table>
<thead>
<tr>
<th>1. GM Pickup</th>
<th>6. Dodge Pickup</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Ford Pickup</td>
<td>7. Olds Cutlass</td>
</tr>
<tr>
<td>3. Honda Accord</td>
<td>8. Honda Civic</td>
</tr>
<tr>
<td>4. GM Suburban</td>
<td>9. Ford Mustang</td>
</tr>
<tr>
<td>5. Toyota Camry</td>
<td>10. Olds 88</td>
</tr>
</tbody>
</table>

It would be efficient to have a timely useful system for collecting, analyzing and sharing information about vehicle thefts. This technical report discusses the design and implementation of an automated crime analysis and mapping software called Auto Theft Analyst (ATA). ATA is a hybrid application package that uses a Microsoft Access database for storing theft occurrence locations, and some GIS functionality to produce different queries and thematic maps. Such a mapping system will assist in accomplishing the following goals in the future: (1) Enhance the decision-making capabilities of department commanders in deploying their resources more effectively; (2) Perhaps more importantly,
strengthen the investigative capabilities of both auto theft detectives and district patrol
officers in targeting hot spots for motor vehicle thefts and recoveries [Lewin 1995].

Both of these goals are critical for reducing the number of motor vehicle thefts in
Corpus Christi, as well as increasing the apprehension of offenders. Although the number of
motor vehicle thefts that occur each year in Corpus Christi is high, improving the quality of
auto theft data to include accurate addresses for theft and recovery locations, and using GIS
analysis, will lead to exposing valuable information to reduce auto theft crimes in the city.

This report is divided into two parts: part one comprises a detailed review of the ATA
system design and implementation. One section shows the ATA graphical user interfaces
through snapshots of the program. Another section, the design and implementation section,
provides a detailed description of the internal steps and approaches considered at each level
of the system development. Also, it demonstrates the difficulties faced in the implementation
of the geocoding tool, due to MapObjects limitations, and the alternative that was adopted to
overcome such limitations.

Part two reveals the results of the evaluation and testing that was performed in the
process of the project accomplishment. Also, it identifies some of the additional work and
details that can be considered to improve the ATA system. Finally, the conclusion section
briefly summarizes the main outcomes of the project.
2. COMPUTERIZED GIS FOR COMBATING AUTO THEFT CRIME

The Office of Community Oriented Policing Services (COPS) has encouraged the use of computer mapping through its grants programs and has demonstrated a strong interest in fostering problem solving, as well as more concentrated approaches to allocation of police resources. In support of this mandate, COPS has provided funds to the Police Foundation’s Crime Mapping Laboratory to provide a wide range of support and information, including assisting local police agencies in implementing computer mapping technologies and encouraging mapping applications and information systems in the development of community-oriented policing and problem solving in COPS sites [CMLFP 2003].

As the proposal for this project stated, the project theme is adopted from research funded by the Corpus Christi Police Department (CCPD) and conducted by Texas A&M University-Corpus Christi (TAMUCC) in 2001. Therefore, the confidential data and information about offenders, officers and other procedures applied in the process of investigation have been eliminated. Issues associated with the project database and GIS programming procedures are addressed in the following sections.

The graphical user interface for ATA is illustrated in the next section. Several forms and dialog boxes are shown, along with descriptions of how users can access these forms. Auto Theft Analyst (ATA) contains several GIS automated tools and database procedures to produce maps and generate results that might support police investigations in combating auto thefts crimes.
2.1 Graphical User Interface (GUI)

The ATA system is a pure Java application that combines several utilities and programmatic mechanisms for handling database activities, such as system security and data manipulation. In addition, it has a set of predefined GIS facilities for map manipulations, such as zooming, selecting features, and querying feature data.

2.1.1 Login Form

As any secure system, ATA starts with a login interface prompt that permits users to access the system after the authentication process has occurred. Each user of the system is assigned a login and password by the administrator. Logins and passwords are stored in a table called Users, shown in the database Entity Relationship Diagram (ER-Diagram) (Figure 3.5). Authorized users can login to the system after entering their own logins and passwords and clicking the OK button, or they can exit the system by clicking on the Exit button (Figure 2.1). The Default mapping system checkbox, if selected, will dynamically load a set of layers as a default view to display the street map of Corpus Christi. These layers are stored as Shapefiles in the Data sub-directory of the main directory, where the ATA system files are installed. Shapefiles stores non-topological geometry and attribute information for the spatial features in a dataset. The geometry for a feature is stored as a shape comprising a set of vector coordinates. Shapefiles handle single features such as points, arcs or polygons that might or might not overlap each other.

Logging into the ATA system, without selecting this checkbox, allows the users to load their own data layers for any geographical jurisdictions of their own choice, as long as those layers are in Shapefiles format.
2.1.2 Auto Theft Analyst Main Form

After users login to the system, the ATA console is displayed to let them access the exiting data layer set, for Corpus Christi area, or they can use their own data of choice. Figure 2.2 shows the ATA console, which consists of four menus in the menu bar, 23 buttons in the toolbar, one Table of Contents and one ATA Map Viewer. Table of Contents includes all the open layers that users can load to the system by using the Add Layer button in the toolbar. It also has the capabilities to enable and disable some buttons in the toolbar based on the active or selected layer. In other words, when a user clicks on one of the layers in the Table of Contents, some of the toolbar buttons will be enabled, if their functionalities are applicable on the active layer. Figure 2.2 shows six layers loaded to the ATA console during the initialization process of the system. ATA Map Viewer displays the maps of all layers found in the Table of Contents. In addition, it reflects all the actions that can be invoked by any of the toolbar buttons.
The following is a detailed description of every button in the toolbar of the ATA console:

1. **Add Layer:** pops up the **Select Data Source Panel**, shown in Figure 2.3, for loading a *Shapefile*, as a new layer, to the **Table of Contents**.

![Select Data Source Panel](image)

**Figure 2.3 Select Data Source Panel**

2. **Print:** prints the current map view displayed on the **ATA Map Viewer**.

3. **Layer Properties:** pops up the **Layer Properties Panel** for the active layer in the **Table of Contents**. Figure 2.4 shows **Layer Properties Panel** for the nodes layer, shown in Figure 2.2. This panel can be used for changing the *Layer Legend* that includes the symbol style, color and size.

4. **Previous Extent:** moves the content of the **ATA Map Viewer** one step backward.

5. **Next Extent:** moves the content of the **ATA Map Viewer** one step forward.

6. **Zoom To Active Layer:** displays the whole view of the active or selected layer in the **Table of Contents**.
7. **Zoom To Full Extent:** zooms in to the maximum extent that shows all exiting layers in one view on the **ATA Map Viewer**.

8. **Zoom In:** when this button is selected, a mouse pressed event will cause the map on the **ATA map Viewer** to begin showing a rectangle that has one corner anchored at the mouse-down location. As the mouse is dragged across the screen, this rectangle is continually resized with the opposing corner located at the current mouse position. Once the mouse button is pressed again, the whole content included in this rectangle is zoomed in.

9. **Zoom Out:** provides a tool for clicking or dragging a rectangle on the **ATA Map Viewer**. In order to zoom out, one follows the same procedure in the **Zoom In** tool.
10. **Pan:** when this button is selected, a mouse pressed event will cause the map to begin panning. As the mouse is dragged across the screen, the map display in the **ATA Map Viewer** is moved so that the initial mouse-down location remains underneath the moving mouse. When mouse is released, panning ceases and if the final mouse location is different from the initial mouse-down location, the map extent will be adjusted to reflect the change.

11. **Pan One Direction:** pans the map display in the **ATA Map Viewer** in one of four directions, north, south, east, or west.

12. **Identify:** provides a tool for clicking on features in the **ATA Map Viewer**, and displaying attributes about those features. The information about those features will be shown in a small box (Figure 2.5).

![Figure 2.5 Results After Using the Identify Tool](image)

13. **Search:** opens a dialog for locating features based on a predefined stored query.

14. **Find:** opens a dialog for locating features whose attributes contain an end-user provided string.
15. **Query Builder**: opens a dialog for locating features based on a query that the user constructs. This tool, shown in Figure 2.6, is easy-to-use and can issue queries to select data and join tables. Before using this tool, the user should activate the queried layer by clicking on that layer in the **Table of Contents**. Then, ATA populates **Select a field** list with all the fields that belong to the active layer. The user can construct his query by selecting, in order, the field name, the logical operator and finally the field value from the **Values** list, which will be populated based on the chosen field.

Thereafter, the user can check the generated query in the text field, and then click **Execute** to select all the features, that match the query result, on the active layer.

![Query Builder](image)

*Figure 2.6 The Query Builder Associated with the Active Layer*
16. **Select Feature:** provides a tool for selecting features by a rubber-banding shape on the **ATA Map Viewer**. When this button is clicked, a small drop down menu is display to the user to choose which selection shape requires to be used, such as Rectangle, Circle, Line or Polygon. Thereafter, the mouse can be dragged down on the **ATA Map Viewer** to select all features on the drawn shape.

17. **Clear All Selection:** clears all the currently selected features on the **ATA Map Viewer**.

18. **Buffer:** opens a dialog for constructing a buffer polygon around the currently selected features (Figure 2.7).

![Buffer](Image)

*Figure 2.7 The Buffer Tool*

19. **Attributes:** displays attributes of the currently selected features (Figure 2.8).

20. **Table View:** displays the table of attributes associated to the active layer. This table is a database table, which has “.dbf” format. It contains all the information required to query and select the features associated with its records. Figure 2.9 shows the table of attributes of the node layer, shown in Figure 2.2. The **Table View** tool shows at most 100 records per page, where the user can navigate between pages.
Figure 2.8 The Attributes Panel Generated Output

Figure 2.9 The Attribute Table Generated by the Table View Tool
21. **Locate Addresses**: uses ArcIMS geocoding services to locate addresses. Section 3.4 explains the geocoding functionality and the alternative used to implement the **Manual Geocoder** in the ATA system.

22. **Map Tips**: opens a dialog for choosing map tips for all layers found in the **Table of Contents** (Figure 2.10). The user can select the layer from the **Layers** list, and then the **Fields** list will be populated to show all fields that belong to the selected layer. Finally, the user can click on the **Set Map Tips** button to set the tips for all selected layers. Thereafter, the values of the selected fields in **Map Tips Dialog** will be displayed on the **ATA Map Viewer**, whenever the user hovers over a certain feature on the map.

23. **Measure Tool**: provides a rubber-banding tool for measuring distances on screen between two or more features. It uses the map unit such as feet, miles, meters or kilometers as a measurement.

![Map Tips Dialog Box](image-url)

*Figure 2.10 Map Tips Dialog Box*
In addition, Figure 2.2 shows four menus: **File, Data Entry, Distance Calculation** and **Help**. The following sections explain each of these menus in detail. Figure 2.2 also shows the **Status Bar** in the bottom of the ATA console. This bar determines the latitude and longitude in XY coordinate system, whenever the user moves the mouse over the **ATA Map Viewer**.

### 2.1.3 The File Menu

Figure 2.11 shows the contents of this menu. It has 5 menu items:

- **Add Layer**: functions as same as the **Add Layer** button in the toolbar.
- **Remove Layer**: removes the active layer from the **Table of Contents**.
- **Layer Properties**: functions as same as the **Layer Properties** button in the toolbar.
- **Print**: functions as same as the **Print** button in the toolbar.
- **Map Units**: sets the map unit in the **ATA Map Viewer** container. Three units, as shown in Figure 2.11, are considered: Decimal Degrees, Feet and Meters.

![Figure 2.11 The File Menu](image)

### 2.1.4 The Data Entry Menu

There are three items in this menu: **Add Theft/Recovery**, **Add User** and **Add Damage Pattern** (Figure 2.12).
This menu allows the user to add data to the Auto-Theft database, such as a new theft or recovery incident, a new user to access the ATA system or a new damage pattern to describe the condition of the stolen vehicle at the time it was recovered, (refer to the database in section 3.2). Each item in the Data Entry menu is attached to a form that is linked to one of the Auto-Theft database tables.

Figure 2.13, Add Theft Incident Form, is the generated output from clicking on Add Theft/Recovery. Add Theft Incident form has 19 fields that include information such as case number, vehicle data and incident location. This form should be used, whenever a new theft entry needs to be added to the database. This form is a user-friendly form in which the user can easily switch between text fields by using the Tab key. After filling all the fields available for a new theft incident, the user should click on Submit Data button to save the results in the Thefts table in the database. In case of providing any invalid data such as a invalid date, invalid time, alphabetic case number or alphabetic location number, the appropriate error message will be fired to inform the user about the correct format of the wrong data field.
Figure 2.13 Add Theft Incident Form

To add new recovery data for any theft incident already existed in the database, the user can enter the relevant Case Number in the Add Theft Incident form, and then press Enter to retrieve the data associated with such a case from the database. The Recovery button will be highlighted, where the user can click on to display the Add Recovery Incident form (Figure 2.14). In this form, all entered fields are submitted to the Recovery table in the Auto-Theft database.
To add a new user to the system, the **Add A New User** form, shown in Figure 2.15, should be accessed by the administrator, whose login is *Admin*. This form is called, when the administrator clicks on the **Add User** item in the **Data Entry** menu (Figure 2.12). The **Add A New User** form is linked to the **Users** table in the Auto-Theft database. The administrator has the right to add and delete users from this table. Other users are not allowed by the ATA system to access this form.

Figure 2.16 shows a form to add new damage pattern to the **DamagePatterns** table in the database. These patterns show the ways of breaking into the vehicles and how the offenders stole those vehicles, more details in section 3.2.
Figure 2.15 Add A New User Form

Figure 2.16 Add A New Damage Pattern Form
2.1.5 The Distance Calculation Menu

This menu is used to calculate the distance between the theft location and the recovery location, after the user fills up the system with the sufficient data. This menu calls the Distance Calculation Form, shown in Figure 2.17. The form consists of one combo box for Case Number and two text fields for Theft Location and Recovery Location. Case Number is automatically populated from the Auto-Theft database. The Theft Location and Recovery Location values are assigned automatically by the Manual Geocoder, explained later in section 3.5.1. The user can change these values, while the required update takes place on the database when the OK button is clicked. Section 3.6 explains in details the approach considered for implementing the Distance Calculation Tool.

![Distance Calculation Form]

Figure 2.17 Distance Calculation Form
2.1.6 The Help Menu

This menu includes the About menu item. It provides information about the Auto Theft Analyst system. It has a single form, shown in Figure 2.18.

![About Auto Theft Analyst Form](image)

**Figure 2.18 About Auto Theft Analyst Form**

Appendix D of this report includes all the source code for the ATA system and its forms.
3. SYSTEM ANALYSIS AND DESIGN

Several steps and procedures were considered, at this stage of the project, to understand the requirement specifications. Since, as stated previously, the main research was adopted from a CCPD funded project, several interviews and meetings were held at different administrative levels in CCPD and TAMUCC. These are some of the methods used for gathering information from CCPD:

- Interviewed administrators, police personnel, and employees using the current system. Several questions were asked to collect more information and data about the current system, existing database and the expectations of the new system.

- Analyzed reports and forms accessed when vehicles are reported stolen or recovered, such as the “Stolen Vehicle Entry Form”, “Suspect Report” and “Offense Report” (Appendix C). Such forms must be filled out by police officers or CCPD authorized employees. They are used to obtain information that police officers seek about owners, vehicles and thefts occurred. These forms are used later in the process of data entry and manipulation. For the purpose of this project, those forms and reports have been accessed to create the main database model and ER-Diagram of the Auto Theft Analyst system.

- Interacted with the current database used in CCPD, and gathered information about data entry process and procedures used to update records based on the changing events.

- Acquired a sample of real-time dataset and data dictionary to understand, in depth, the structure of the current database.
Section 3.1 describes the CCPD dataset sample, and section 3.2 explains the new database model considered for the ATA system.

3.1 Anatomy of Current Datasets

The vital datasets were acquired after the last meeting held with CCPD representatives on February 25, 2002 at TAMUCC. Two Excel spreadsheet files were received on May 15, 2002. The first file includes all auto thefts, which occurred in 2000 "HS_FILE_2000", while the second has all thefts during 2001 "HS_FILE_2001".

3.1.1 Stolen Data

As a start, the acquired files were converted to dBase IV for compatibility with ESRI GIS software. CCPD maintains good informative records about the stolen vehicles such as type, make, year, model, style, stolen location and date, and vehicle VIN...etc (Figure 3.1). Only problem faced was the existence of multi-values in the Stolen_Loc attribute. This attribute was normalized later to three different columns in both database tables (2000 and 2001) to maintain the data consistency (Figure 3.2)
Figure 3.1 A view of CCPD Database File for incidents in 2000

Figure 3.2 The 2001 CCPD Database After Normalization
In 2000, CCPD reported 677 stolen vehicles. Table 3.1 shows the top 10 vehicles stolen during that year.

**Table 3.1 Top 10 Vehicles Stolen in Corpus Christi in 2000**

<table>
<thead>
<tr>
<th>Vehicle Make</th>
<th>Vehicle Style</th>
<th>Number of Thefts</th>
<th>Preferable Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chevrolet</td>
<td>Pickup</td>
<td>70</td>
<td>1994, 1992</td>
</tr>
<tr>
<td>Chevrolet</td>
<td>4 Doors</td>
<td>34</td>
<td>2000, 1999, 1984</td>
</tr>
<tr>
<td>Ford</td>
<td>Pickup</td>
<td>31</td>
<td>1995, 1994</td>
</tr>
<tr>
<td>Chevrolet</td>
<td>2 Doors</td>
<td>29</td>
<td>1993</td>
</tr>
<tr>
<td>DODGE</td>
<td>4 Doors</td>
<td>25</td>
<td>2000</td>
</tr>
<tr>
<td>Ford</td>
<td>4 Doors</td>
<td>23</td>
<td>1994, 1991</td>
</tr>
<tr>
<td>GMC</td>
<td>Pickup</td>
<td>21</td>
<td>1995</td>
</tr>
<tr>
<td>Ford</td>
<td>2 Doors</td>
<td>20</td>
<td>1999</td>
</tr>
<tr>
<td>DODGE</td>
<td>Pickup</td>
<td>20</td>
<td>1992</td>
</tr>
<tr>
<td>BUICK</td>
<td>4 Doors</td>
<td>16</td>
<td>1987</td>
</tr>
</tbody>
</table>

Table 3.2 shows the top 10 vehicles stolen during 2001, where 745 auto thefts were reported in that year.

**Table 3.2 Top 10 Vehicles Stolen in Corpus Christi in 2001**

<table>
<thead>
<tr>
<th>Vehicle Make</th>
<th>Vehicle Style</th>
<th>Number of Thefts</th>
<th>Preferable Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>DODGE</td>
<td>4 Doors</td>
<td>37</td>
<td>2001</td>
</tr>
<tr>
<td>Chevrolet</td>
<td>2 Doors</td>
<td>30</td>
<td>1987, 1995</td>
</tr>
<tr>
<td>Ford</td>
<td>4 Doors</td>
<td>29</td>
<td>1998</td>
</tr>
<tr>
<td>DODGE</td>
<td>Pickup</td>
<td>25</td>
<td>2001, 1999</td>
</tr>
<tr>
<td>Chevrolet</td>
<td>4 Doors</td>
<td>23</td>
<td>1994, 1992</td>
</tr>
<tr>
<td>Olds</td>
<td>4 Doors</td>
<td>18</td>
<td>1986</td>
</tr>
<tr>
<td>Ford</td>
<td>2 Doors</td>
<td>15</td>
<td>Between 1990-2000</td>
</tr>
<tr>
<td>DODGE</td>
<td>VAN</td>
<td>14</td>
<td>Between 1990-2001</td>
</tr>
</tbody>
</table>

**3.1.2 Recovery Data**

Keeping records of the recovery data, such as vehicle locations when they were recovered, and later analyzing this information might lead to some conclusions about when and how those vehicles were abused, and where the offenders prefer to get rid of those stolen vehicles.
vehicles. Jerry Johns, president of Southwestern Insurance Information Service, an insurance trade group, said “The recovery rate for stolen vehicles in Texas during 1997 was 67% which illustrates that your stolen car may very well be in a foreign country” [Johns 1999].

In 2000, 64% (= 432) of the stolen vehicles were recovered, while 61% (= 455) were recovered in 2001. CCPD maintains information about the location where a vehicle was stolen from, but not the recovery location, in case the vehicle was recovered. Vehicle recovery efforts can be improved by recording addresses of recovered stolen vehicles. More data about auto theft and recovery locations might enhance GIS analysis of auto theft prevention. Spatial analysis of auto theft recovery data can assist in:

1. Identifying the hot-spot areas, where stolen vehicles recovered.

2. Presenting the distribution of vehicle thefts and recoveries across the city.

3. Tracing the vehicles to check if some were recovered on the Mexican border.

4. Redistributing patrol officers by targeting hot-spot areas for recoveries. Perhaps, this will enhance the probability of arresting more offenders, since it is more convenient for them to leave the vehicles in uncovered areas.

Figure 3.3 demonstrates a view of CCPD provided dataset that shows how recovery incidents enter to the system. Since the stolen and recovered vehicles are maintained in same table, records blank (unfilled) in the recovery attributes are treated as non-recovered vehicles.
Figure 3.3 Unfilled Data Attributes

Figure 3.4 represents a way of data entry for the recovery addresses, which were entered as "CCPD". In this situation, no analysis for recovery data can be undertaken. A better solution is to record exact addresses, where officers find the stolen vehicles.

Figure 3.4 Recovery addresses attribute values in 2000 data
3.2 New Database Modeling

Instead of interacting with the currently used database from CCPD, it was decided to build a new Auto-Theft database module that served the project purpose and narrowed the database domain. This generic approach was helpful to identify Auto Theft Analyst system as a standalone platform that will be running under Windows. Figure 3.5 presents the Auto-Theft Database ER-Diagram. The main database consists of four entities: Users, Thefts, Recoveries and DamagePatterns. In addition, the ER-Diagram shows a single relationship entity, named Logs, between Users and Thefts. Each of these entities is described in this section. Appendix A presents the data dictionary that addresses all fields found in the Auto-Theft database. Appendix B shows the database schema.

![Diagram of Entity Relationship Diagram for the Auto-Theft Database Module](image)

Figure 3.5 Entity Relationship Diagram for the Auto-Theft Database Module
*Users* entity contains information about legitimate users such as first and last names, positions, logins and passwords. It is mainly used for log and authentication purposes. Authentication happens, whenever a user enters his login and password in the *Login Interface Prompt*, where there is a validation method, written in Java, that authenticates the user and retrieves his information from this entity. *Users* entity is in a many-to-many relationship with *Thefts*. This relationship constructed the *Logs* entity, to keep a record of each user who logs into the system and adds a new theft incident. In the future, log reports can be generated, whenever it is required, throughout this log entity to show the users’ names, the case numbers they have previously added and the time of their transactions.

*Thefts* entity includes data about the vehicle descriptions, incident times and locations. This entity is accessible, whenever a new theft incident is reported. It is in one-to-one relationship with the *Recoveries* entity, because every auto theft might have at most one recovery case. *Recoveries* entity maintains information about the recovery date, time and location, also a description of the vehicle damage at the recovery time. In addition, this recovery entity maintains the distance value between the theft and recovery locations. The distance calculation tool, described in section 3.6, calculates this value and maintains the result as an attribute in this entity. Finally, *Recoveries* is in one-to-many relationship with *DamagePatterns* that maintains all the damage patterns for all reported vehicles. This tells that many recoveries might categorize under one pattern of damage for vehicles.
3.3 Database Creation and Software Design Patterns

Microsoft Access was the tool used to create each of these entities, and to build the database file. Thereafter, two techniques, Connection Pool and Factory Method, for object-oriented programming were adopted to provide elegant and reusable media between the data and the source code. Figure 3.6 depicts the hierarchy of the internal coding process in Auto Theft Analyst (ATA).

![Diagram showing hierarchy of coding in ATA]

**Figure 3.6 Hierarchy of Coding in ATA**

First, the database content and tables are totally hidden and wrapped with the JDBC Driver and Connection Pool. The connection pool technique synchronizes the process of opening and closing connections with the main MS Access database using the JDBC 2.0 Driver. Then, the Factory Method acts as an intermediary to split all the SQL queries from the main source code of all GUIs utilized in ATA. The Factory Method contains a wrapper class for each entity in the database, plus all the SQL queries from insertion, updating and deletion. On the top of this method, the source code and GUI of ATA are placed.
3.3.1 The Connection Pool Technique Used in ATA

Connecting to any database is a time consuming activity, since the database must allocate communication and memory resources as well as authenticate the user and set up the corresponding security context. The exact time varies, of course, but it is not unusual to see connection times of one or two seconds. Establishing the connection once and then using the same connection for subsequent requests can, therefore, dramatically improve the performance of any database driven software. This feature has a major impact on the system. That impact occurs due to eliminating the overhead in creating and destroying different objects by the database resource manager process [JavaWorld 2000].

The Java Database Connect API (JDBC) is supported by all major database vendors, such as Microsoft Access through the JDBC-ODBC driver manager. To access a database through JDBC, one needs to open a connection to the database, resulting in a Connection object. A Connection object represents a native database connection and provides methods for executing SQL statements. The database connection pool has a manager class which provides an interface to multiple connection pool objects. Each pool manages a set of JDBC Connection objects that can be shared by any number of processes [Bergsten 1999].

The database connection pool class ConnectionPool.java is a Java thread object that implements the Runnable Interface. It provides methods for doing the following:

- Get an open connection from the pool or create a new connection if there is no availability through the getConnection() subroutine.

    ```java
    public synchronized Connection getConnection()
    throws SQLException {
        if (!availableConnections.isEmpty()) {
            Connection existingConnection = 
                (Connection)availableConnections.lastElement();
            int lastIndex = availableConnections.size() - 1;
    ```
availableConnections.removeElementAt(lastIndex);  
// If connection on available list is closed (e.g.,  
// It timed out), then remove it from available list  
// and repeat the process of obtaining a connection.  
// Also wake up threads that were waiting for a  
// connection because maxConnection limit was reached.  
if (existingConnection.isClosed()) {
    notifyAll(); // Freed up a spot for anybody waiting  
    return(getConnection());
} else {
    busyConnections.addElement(existingConnection);  
    return(existingConnection);
} else {
    // Three possible cases:  
    // 1) You haven't reached maxConnections limit. So  
    // establish one in the background if there isn't  
    // already one pending, then wait for  
    // the next available connection (whether or not  
    // it was the newly established one).  
    // 2) You reached maxConnections limit and waitIfBusy  
    // flag is false. Throw SQLException in such a case.  
    // 3) You reached maxConnections limit and waitIfBusy  
    // flag is true. Then do the same thing as in second  
    // part of step 1: wait for next available connection.  
    if ((totalConnections() < maxConnections) &&  
        !connectionPending) {
        makeBackgroundConnection();
    } else if (!waitIfBusy) {
        throw new SQLException("Connection limit reached");
    }
    // Wait for either a new connection to be established  
    // (if you called makeBackgroundConnection) or for  
    // an existing connection to be freed up.  
    try {wait();} catch(InterruptedException ie) {}  
    // Someone freed up a connection, so try again.  
    return(getConnection());
}

- Return a connection to the pool through free(connection) method:

    public synchronized void free(Connection connection) {  
        busyConnections.removeElement(connection);  
        // System.out.println("Busy Connection = "+busyConnections.size());  
        availableConnections.addElement(connection);  
        // System.out.println("Available Connection" +  
        availableConnections.size());  
        // Wake up threads that are waiting for a connection  
        notifyAll();
    }


- Close all connections at shutdown through `closeAllConnections()` method:

```java
public synchronized void closeAllConnections() {
    closeConnections(availableConnections);
    availableConnections = new Vector();
    closeConnections(busyConnections);
    busyConnections = new Vector();
}
```

- Handle connection failures, such as timeouts, communication failures, etc.
- Limit number of connections in the pool to a predefined max value [Hall 2003].

The manager class, `APDbPool.java`, is a wrapper around the `ConnectionPool` thread that manages multiple connection pools in the ATA system. It does each of the following:

- Load and register all JDBC drivers.
- Create `ConnectionPool` objects, to connect to the Auto-Theft database, based on properties defined in the properties file called `ConnectionDb.properties`. This properties file contains configuration parameters that can be adjusted during the installation process of the ATA system.
- Register all connection pool instances that are created as long as the ATA system is running.
- Keep track of connection pool processes to gracefully shut down all pools’ objects such as statements and result sets, when the last process is completed. This happens by calling `closeResources(connection, statement, resultSet)` method. See Appendix D for the source code.
3.3.2 The Factory Method Used in ATA

One of the goals of object-oriented design is to delegate responsibility among different objects. This kind of partitioning is good since it encourages encapsulation and delegation. Sometimes, an application (or framework) at runtime, cannot anticipate the class of object that it must create. The application (or framework) may know that it has to instantiate classes, but it may only know about abstract classes (or interfaces), which it cannot instantiate. Thus the application class may only know when it has to instantiate a new object of a class, not what kind of subclass to create. On the other hand, a class may delegate responsibility to one of several helper subclasses so that knowledge can be localized to specific helper subclasses [Raj 1999].

The Factory Method, also known as Data Access Object (DAO), is a creational pattern. This pattern helps to model an interface for creating an object, which at creation time can let its subclasses decide which class to instantiate. It was named Factory Method, because of its responsibility for "Manufacturing" an Object. This method helps instantiate the appropriate subclass by creating the right object from a group of related classes [SunDOA 2002].

The Factory Method promotes loose coupling by eliminating the need to bind application-specific classes into the code [Garcia 2001]. In this project, the ATA application's classes access database catalog of Thefts, Recoveries, User, Logs and DamagePatterns using the DOA interface called APDbAble. Recreating APDbAble for another data access mechanism (to use a new object such as Vehicles, for example) would have a little or no impact on any classes that use the APDbAble interface, because only the implementation class, which is called APDbAble would change. Each potential alternate implementation of APDbAble would access data for the entities in the database catalog in its
own way, while presenting the same API to the interface that utilizes that implementation.

Figure 3.7 illustrates the roles of the Factory Method used in this project.

![Diagram of Factory Design Method](image)

**Figure 3.7 The Factory Design Method Concept in ATA (Adopted from [Garcia 2001])**

Figure 3.8, below, shows the control flow details of component creation from the main program using the Factory Method in the ATA software.

<table>
<thead>
<tr>
<th>Program Source Code</th>
<th>APDbAble (DOA Interface)</th>
<th>APDbFactory (Factory Method)</th>
<th>APDbImpl (Interface Implementation)</th>
<th>Database</th>
</tr>
</thead>
</table>

![Control Flow Diagram](image)

**Figure 3.8 Control flow details between the ATA program and its database**
As shown in Figure 3.8, the program first asks for an instance of \textit{APDbAble} Interface. Then, the program must query a factory class, called \textit{APDbFactory} in the ATA system, to invoke the \textit{getImpl()} method. This method returns the correct and required implementation of the \textit{APDbAble} instance. It returns an \textit{APDBImpl} instance, which can send the required query to the database and retrieve the correct results back to the main program.

The following shows the source code of the \textit{APDbAble} interface that declares eight abstract methods as interfaces to interact with the Auto-Theft database for updating and query data. The implementations of these methods are found in the \textit{APDbImpl} class. Refer to Appendix D for the source code of the ATA software.

```java
public interface APDbAble {
    // Four Methods To Retrieve Different Objects from each database table
    public abstract User findUser(String username);
    public abstract Theft findTheft(String caseNo);
    public abstract Recovery findRecovery(String CaseNo);
    public abstract DamagePattern findDamagePattern(int patternId);
    // Two Methods To Handle updating Objects in the main database
    public abstract boolean updateUser(User user, int action);
    public abstract boolean updateTheft(Theft theft, int action);
    public abstract boolean updateRecovery(Recovery recovery, int action);
    public abstract boolean updateDamagePattern(
        DamagePattern damagePattern, int action);
}
```

Finally, the factory class, \textit{APDbFactory}, is a separate class to factorize the \textit{APDbAble} objects, any time they are acquired by callers. In this situation, there is no need to identify which implementation should be referred to, since this will be identified during the program execution runtime. The code of \textit{APDbFactory} class is

```java
public class APDbFactory{
    private APDbFactory() { }
    public static APDbAble getDbImpl()
    {return new APSqlDbImpl();}
}
```
3.4 MapObjects-Java Standard Edition in Auto Theft Analyst

MapObjects-Java Edition is a developer toolkit consisting of more than 900 Java developer components that can be used to build custom, cross-platform GIS applications, servlets or applets. It provides a robust, Java-based API that allows programmers to design applications to perform a wide variety of geographic-based display, query, and data retrieval activities. It also includes predefined Java Beans that are easily used in the integrated development environments, such as Borland's JBuilder™ software [ESRI 2003].

MapObjects allows users to create applications that perform activities such as labeling map features, thematic mapping, panning and zooming through multiple map layers, querying spatial and attribute data, performing geometric operations, measuring distances, displaying real-time geographic data, and much more. MapObjects-Java Standard Edition is built entirely on the Java 2 platform. It consists of a set of Java Archive (JAR) files containing pure Java components that can be referenced and used to develop custom Java-based client standalone applications or applets [ESRI 2002a].

The suite of visual JavaBean contained in MapObjects-Java Standard Edition can be referenced in an integrated development environment (IDE) such as JBuilder™ and Forte™ for Java. These beans can be used in a drag-and-drop environment for building graphical user interfaces. All of the visual components in MapObjects-Java Standard Edition extend from the Java Swing components. Using these components, developers can build applications that include functional toolbars; overview maps, and map legends that make custom applications easy to use [ESRI 2002b].

However, MapObjects is not the perfect approach to create a complete Java mapping system, due to its limitations in the APIs, such as no geocoding APIs found to locate
addresses on a separate street map layer without assistance from an external server. In addition, MapObjects lacks online tutorials and documentations. This is because it is an ESRI software product, and therefore, developers need to attend special classes to learn the APIs in MapObjects.

In this project, the MapObjects 1.0 APIs were used to build the map and layer components that construct the GUI of the ATA system, which explained in section 2 of this report. The following APIs were used for this purpose:

- ToolBar APIs: To create the ATA toolbar and its 23 buttons, refer to Figure 2.2.
- ToolBarAction APIs: To define the action invoked by each of the 23 buttons in the toolbar.
- JavaBeans APIs: to add the Map and Layer Components to the main system.

Section 3.5 describes the ATA manual geocoder, and what other alternatives can be taken to implement better geocoding. Section 3.6 explains the distance calculation tool design and implementation.

3.5 The Geocoding Tool

In this project, the geocoding tool or geocoder is a required procedure to locate the incidents, thefts and recoveries, on the street map. MapObjects Java version lacks the required libraries or APIs for implementing this GIS feature in any GIS Java application without using the geocoding service provided by ArcIMS. For the project purpose, a manual geocoder was created as an approach to overcome the MapObjects limitations, and later to integrate the distance calculation tool to the ATA system, described in section 3.6.
3.5.1 The Manual Geocoder

The process of geocoding is to find the latitude and longitude, in XY coordinate system, to a certain address, and then locate this address as a point on the map. The geocoding tool takes the block number and the street name of a target address as inputs, and then it calculates the XY coordinates for that address, and later draws the point on the road map. The **Manual Geocoder** in the ATA system was implemented by creating a point *Shapefile* using ArcView 3.2a that represents all the street intersections in the City of Corpus Christi. Each intersection between two or more streets is a node that has latitude and longitude. These nodes have been saved on a text file, *latlong.txt*, and later have been geocoded on ArcView to create a new layer or *Shapefile* called *nodes.shp*. This layer was loaded to the ATA system after applying some data manipulation on ArcView (Figure 3.9).

![Diagram of the node layer created for the Manual Geocoder in ArcView](image)

*Figure 3.9 The Node Layer Created for The Manual Geocoder in ArcView*
The *Nodes* layer has a database table that contains information about each node in attributes such as latitude, longitude, right block numbers (*R_NREFADDR* and *R_REFADDR*), left block numbers (*L_NREFADDR* and *L_REFADDR*) and street name (*ST_NAME*) (Figure 3.10). The geocoder, in the ATA system, reads the location number and street name from the user input provided through the theft and recovery forms shown in Figures 2.13 and 2.14, and then searches through this table to find all points that matches the given street name and the location number is between left-block-number-start and left-block-number-end or between right-block-number-start and right-block-number-end. The Geocoder SQL statement is:

```
SELECT I, LATITUDE, LONGITUDE
FROM NODES
WHERE ST_NAME = :location_street_name
AND (R_NREFADDR <= :location_number AND
    R_REFADDR >= :location_number) or
    (L_NREFADDR <= :location_number AND
    L_REFADDR >= :location_number)
```

![Table View](image)

**Figure 3.10** The Database Table of the Node Layer
The values of latitude and longitude returned from the above SQL query are the latitude and longitude of the geocoded location. These values might not be so accurate, but they are still located in the area where the real address is. The ATA Manual Geocoder is not the ideal method to geocode locations, but it overcomes the limitations in MapObjects 1.0 to geocode locations without using ArcIMS Geocoding Service. The next section explains how MapObjects communicates with ArcIMS for this geocoding purpose. This ArcIMS geocoding version can be implemented in the future versions of the Auto Theft Analyst software.

3.5.2 The Geocoder in ArcIMS

ArcIMS provides the foundation for distributing high-end GIS and mapping services via the Internet, and enables users to integrate local data sources with Internet data sources for display, query, and analysis in an easy-to-use Web browser [ESRI 2003].

GeocodingUI, in Figure 3.11, is the geocoding API shipped with MapObjects for implementing a geocoding tool. It extends JDialog and provides a convenient interface for users to input address information and apply a geocode request to a layer whose layer source comes from an ArcIMS Server. When users enter the target address in both text fields, Street and Zone, this request is sent from the Java application to the specified ArcIMS road layer in XML file format. ArcIMS Geocoding server processes the request and returns another XML file that contains latitude and longitude in XY coordinates of the target address. Thereafter, the MapObjects-Java application can extract those XY values from the XML file. Finally, transformWorldToPixel() method in the Map Class can be invoked to present the XY coordinates on the selected road map.
Figure 3.11 GeocodingUI API in MapObjects 1.0

The pseudocode of this process of communication between ArcIMS service layer and a MapObjects application is:

- **Enter Address. Format:** Street (number streetName) and Zone;
- **Encode address in an XML file. Use:** Java XML Encoder;
- **Send request to ArcIMS Server;**
- **Process request and extract address from XML file;**
- **Return XML file from ArcIMS that includes XY coordinates;**
- **Extract XY coordinate from the return XML file. Use:** Java XML Decoder;
- **Draw the point on the map by using transformWorldToPixel() method;**

To implement the GeocodingUI functionality into an application using ArcIMS Server, an action listener for a swing component, such as JButton or JMenuItem can be created to call a method from within the listener to open the GeocodingUI (Figure 3.11). This is the required code to accomplish such a task [ESRI 2002a]:

42
Private void displayGeocodingUI()
{
    Com.esri.mo.ui.qry.GeocodingUI gui =
        new com.esri.mo.ui.qry.GeocodingUI(this);
    Gui.setMap(aMap);
    Gui.setVisible(true);
    Gui.requestFocus();
}

3.6 Distance Calculation Tool

The distance calculation tool is to calculate distances between theft and recovery locations, after they have been geocoded and drawn on the road map layer. Section 2.1.5 shows what GUI used for this tool and how it can be used. This tool is built based on the Manual Geocoder implemented in the previous section. However, this tool can easily be integrated with other geocoding tools in the future. It has a routine for calculating the distance between two points using Dijkstra’s Algorithm. The following explains how Dijkstra’s Algorithm serves this tool and demonstrates the Java program written for implementing this algorithm in the ATA system.

3.6.1 Dijkstra’s Algorithm

Dijkstra’s Algorithm solves the problem of finding the shortest path from a point in a graph (the source) to a destination. It turns out that one can find the shortest paths from a given source to all points in a graph at the same time, hence this problem is called the single-source shortest paths problem [Cormen 1998]. This problem is related to the spanning tree one. The graph, representing all the paths from one vertex to all others, must be a spanning tree. It must include all vertices. There should not be any cycles, as a cycle would define more than one path from the selected vertex to at least one other vertex.
In a directed graph

\[ G = (V,E) \] where \( V \) is a set of vertices and \( E \) is a set of edges \( d, \] \hfill (3.1)

Dijkstra’s Algorithm maintains two sets of vertices:

- \( S \): the set of vertices whose shortest paths from the source have already been determined \textit{and}
- \( V-S \): the remaining vertices, maintained in a priority queue \( Q \) keyed by \( d \) values.

The other data structures needed are:

- \( d \): array of best estimates of shortest path to each vertex.
- \( \Pi \): array of predecessors for each vertex

The implementation assumes that graph \( G \) is represented by adjacency lists. Here is the pseudocode for Dijkstra’s Algorithm [Cormen 1998][Morris 1998].

**DIJKSTRA\((G,w,s)\)**

1. Initialize-Single-Source\((G,s)\);
2. Set \( S \) to empty;
3. Set \( Q \) to All vertices in Graph \( G \)
4. While there are still vertices in \( V-S \),
   - i. Sort the vertices in \( V-S \) according to the current best estimate of their distance from the source,
   - ii. Add \( u \), the closest vertex in \( V-S \), to \( S \),
   - iii. For each vertex \( v \) adjacent to \( u \)
       - do Relax\((u,v,w)\);

**Initialize-Single-Source\((G,s)\)**

1. For each vertex \( v \) in \( V[G] \)
   - do \( d[v] \) is infinity;
   - \( \pi[v] \) is 0;
2. \( d[s] = 0; \)

**Relax\((u, v, w)\)**

if \( d[v] > d[u] + w(u,v) \)
then \( d[v] = d[v] + w(u,v); \)
\( \pi[v] = u; \)