Development of an Accident Visual Analysis 
and Data Integration Tool

State Job No. 14706(0)

FINAL REPORT

Prepared in cooperation with the Ohio Department of Transportation and the U.S.
Department of Transportation, Federal Highway Administration

February, 2001

by
Ping Yi, Yingcai Xiao,
Greg Frommer, Saroja Devarakonda, Anthony Ciccolini.

The University of Akron
This project develops a visualization and integration tool for crash data analysis. Based on a client server data communication system between the central database and distributed users, safety engineers at local districts can download crash data for a specific roadway section in a given time window. They can view crash characteristics through data mapping and attribute listing, and analyze the data using nested query or sorting operations and statistical graphs. A visualization module in the system can be used to construct the collision diagram according to the types, movements, and points of impact of the vehicles involved in the crash. Preliminary tests using crash data obtained from the Ohio Department of Public Safety have shown the effectiveness and efficiency of this tool in crash data analysis.
Development of an Accident Visual Analysis
and Data Integration Tool

State Job No. 14706(0)

FINAL REPORT

Prepared in cooperation with the Ohio Department of Transportation and the U.S.
Department of Transportation, Federal Highway Administration

February, 2001

by
Ping Yi, Yingcai Xiao,
Greg Frommer, Saroja Devarakonda, Anthony Ciccolini.

The University of Akron
DISCLAIMER STATEMENT

The contents of this report reflect the views of the authors who are responsible for the facts and the accuracy of the data presented herein. The contents do not necessarily reflect the official views or policies of the Ohio Department of Transportation or the Federal Highway Administration. This report does not constitute a standard, specification or regulation.
Intranet-based data communications enable the statewide crash data files to be stored and maintained at the Central Office of the Ohio Department of Transportation (ODOT) in Columbus. Computer technologies have revolutionized the data storage and processing power of desktop PCs today, which makes it possible to create district-specific data applications locally by ODOT safety engineers.

ODOT has sponsored a research project at the University of Akron to design an “Accident Visual Analysis and Data Integration Tool” (AVADIT). The tool provides the visual display and data integration functions needed in crash data analysis. This tool can automatically construct collision diagrams, present detailed information for individual crashes, and generate data distributions, charts and figures for intersections, roadway segments, counties, and the entire district.

AVADIT is designed in a modular format. The software consists of four modules: Crash Data Acquisition module, Collision Diagram module, Data Analysis module, and Report module. Based on a client-server data communication framework, the Crash Data Acquisition module includes program features and a user interface for extracting data from the central database; the user can define data limits based on county, dates, route number, the roadway log number and/or other data fields. Once downloaded to the local system, the data can be queried further for more detailed and advanced analysis. The Collision Diagram module transforms the data selected for visualization into graphic symbols depicting the crashes in the form of a collision diagram, where various pre-designed pictures and graphic templates can be added to the diagram such as traffic signs and lane arrows, etc. The Data Analysis module allows the user to define the type and features of charts based on selected data, and print out the artwork or save it as images for reporting. Finally, the Report module allows the user to generate reports where tables, graphs, collision diagrams created during the analysis can be inserted into a report file along with descriptive text.

The entire project produced two products, this project report and a companion software CD with user’s manual. This project report describes the system architecture and development of the functional modules and rule base; the User’s Manual, explains how to use the tool in crash data analysis and includes a case study as an example. Preliminary testing by the project investigators indicated that AVADIT is an effective and efficient tool and very user friendly, although additional functions and enhancements may be built into the model in the near future.

For Copies of this Report, Contact:
Ohio Department of Transportation, Ms. Monique Evans, Office of Research and Development, (614) 728 - 6048, mevans@odot.state.oh.us
ACKNOWLEDGEMENTS

Our sincere thanks and gratitude go to the AVADIT project team, led by project manager Ms. Shireley Shokouhi for their guidance and suggestions throughout the development of the project. The other project team members include

Ron Chesla  Julie Cichello  Kerry Dean  Joseph DeFuria
David Griffith  Lou Hazapis  Matt Parrill  JD Sturgill

Our special thanks go to Mr. Mohammad Khan, Mr. Leonard Evans, Mr. Wallace Richardson, and Mr. Angelo Serra for their invaluable help and encouragement in making this project possible.
TABLE OF CONTENTS

1. **Objectives**  
   1.1 Description of Problem  
   Figure 1.1 Crash Data Sample File  
   1.2 Solution to the Problem  
   1.3 Software Objective

2. **Functional Review**  
   2.1 Software Benefits  
   2.2 System Architecture and Functionality  
   2.2.1 Data Flow  
   Figure 2.1 Logical Data Flow Chart  
   2.2.2 Client Server System  
   Figure 2.2 Client Server Model

3. **Program Modules**  
   3.1 Startup Menu  
   3.2 Crash Data Acquisition module  
   3.3 Collision Diagram module  
   3.4 Data Analysis module  
   3.5 Report module

4. **Rule Base and Data Library**  
   4.1 Data Dictionary  
   4.2 Display Rule Base  
   4.2.1 North Arrow Rotation Rules  
   Figure 4.1 North Arrow Rotation  
   4.2.2 Vehicle Direction Rules  
   Figure 4.2 Virtual Direction Based on the Main-Stream Traffic Direction
4.2.3 Collision Display Rules
Figure 4.3 Graphical Representation of Vehicle at Fault
Figure 4.4 Graphical Representation of Crash Severity
4.2.4 Zoning Rules
Figure 4.5 Designation of Collision Zones for a Four-Leg Intersection
Figure 4.6 Zonal Designation of Collision
Figure 4.7 Traffic Conflict at a Four-Leg Intersection
Table 4.1 Coding Conventions of Collision Characteristics
4.2.5 Validation Rules
4.3 Preference Values
4.4 Column Mapping

Conclusion and Future Work

References
CHAPTER I
OBJECTIVES

1.1 DESCRIPTION OF PROBLEM

Highway safety remains to be the number one priority addressed by highway engineers and safety program managers at all levels of the transportation authorities. Although crash rates per million vehicle miles traveled (VMT) remained nearly unchanged in recent years, there has been a continuous increase of motor vehicle crashes since 1992 in the total number of accidents involving Fatalities, Injuries, and Property Damages, etc. To address this problem, public safety officials are stepping up their effort to enforce traffic safety laws, and traffic engineers are striving to improve the safety standards of our highways.

Effective treatment of roadway safety problems depends on accurate and timely identification of the deficiencies with the design and control of our highways and intersections. Because traffic safety is affected by many factors, including roadway design, traffic flow and control, vehicular characteristics, human factors, weather and lighting conditions, the study of the cause of a crash by traffic safety engineers is usually complex. Although the crash records have already been compiled electronically for easy and quick access in many states, safety engineers at the local level do not have the most effective computer tools to work with as they process the crash data. In Ohio, for example, the engineers have to work with crash data files prepared on a state-wide basis in which all the accidents are coded in numbers based on established coding rules (Figure
1.1). This makes it very cumbersome to manually decode (translate) the numbers in many data fields of the files to investigate crash characteristics.

Secondly, to obtain a clear view of the collision characteristics, safety engineers frequently need to construct the collision diagram, which graphically depicts all the crashes that occurred at a selected intersection or a roadway segment, and the details involved in each crash. Such details include the number of vehicles or pedestrians involved in the crash, the direction of travel of each vehicle, the road surface conditions, the date and time of collision, etc. Many safety engineers have found this task to be extremely time consuming without using some type of automated data visualization tool.

Furthermore, it is often desired that summaries of accident statistics and graphic displays (tables and charts) by type, location, vehicle size, or severity be prepared for comparing differences between locations, routes, or days. This becomes very difficult for the safety engineers because of the lack of a data integration tool for query and sorting application and a data presentation tool for displaying figures and charts.

ODOT is one of only a few states in the country that have made possible automatic and computerized crash data analysis. Up until several years ago, ODOT safety engineers had used a computerized program, TRACPLOT, to perform crash data analysis. With a number of functions, this program was able to perform statistical analysis on crash data and construct a non-graphics based collision diagram for engineers to use. However, the
### TRAFFIC ACCIDENT LISTING

(Jan-Dec, 1997)

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>COUNTY</th>
<th>ROUTE</th>
<th>LOG</th>
<th>MONTH</th>
<th>DATE</th>
<th>DAY</th>
<th>HOUR</th>
<th>CONT</th>
<th>ACC. TYPE</th>
<th>SIDE</th>
<th>PED</th>
<th>VEH</th>
<th>SPEED</th>
<th>DIR</th>
<th>TYPE UNIT</th>
<th>DRUG</th>
<th>ALCO</th>
<th>TYPE 1</th>
<th>TYPE 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>99129</td>
<td>LAK</td>
<td>20R</td>
<td>2549</td>
<td>10</td>
<td>15</td>
<td>1</td>
<td>15</td>
<td>2</td>
<td>06</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>57</td>
<td>04</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>99148</td>
<td>LAK</td>
<td>20R</td>
<td>2549</td>
<td>11</td>
<td>22</td>
<td>4</td>
<td>16</td>
<td>1</td>
<td>2</td>
<td>06</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>6</td>
<td>73</td>
<td>2</td>
<td>1</td>
<td>01</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>99227</td>
<td>LAK</td>
<td>20R</td>
<td>2550</td>
<td>06</td>
<td>29</td>
<td>5</td>
<td>16</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>6</td>
<td>73</td>
<td>05</td>
<td>1</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>99086</td>
<td>LAK</td>
<td>20R</td>
<td>2587</td>
<td>07</td>
<td>07</td>
<td>6</td>
<td>11</td>
<td>1</td>
<td>1</td>
<td>02</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>37</td>
<td>02</td>
<td>1</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>99210</td>
<td>LAK</td>
<td>20R</td>
<td>2610</td>
<td>02</td>
<td>18</td>
<td>7</td>
<td>20</td>
<td>1</td>
<td>1</td>
<td>09</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>6</td>
<td>37</td>
<td>03</td>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 1.1 Crash Data Sample File
software, written many years ago, is incompatible to today's computer hardware. According to the engineers in most of ODOT districts, with whom the principal investigators have discussed this problem, an extraordinary amount of time is spent constructing the diagrams manually now, as they need to do column-row matching by reading off the data in the ROSCOE data files, followed by manual code interpretation.

It is apparent that not having an automated data integration tool can result in discontinuous data flows between applications. Manual manipulation of the data is usually fragmented, complex and cumbersome, leading to work inefficiency and even errors. To effectively minimize manual data manipulation, a customized, automated and dynamic data integration and analysis tool is needed.

1.2 SOLUTION TO THE PROBLEM

Advances in networking and computer technologies in recent years have substantially enhanced information exchange and data processing capabilities for crash data analysis. Intranet communications in ODOT enable the statewide crash data files to be stored and maintained at the Central Office in Columbus. This makes it very convenient to update these files periodically with new records and yet keep most current data easily accessible by safety engineers at the local districts.

The purpose of this project is to provide an automated visualization and data integration tool for ODOT safety engineers to perform district-specific crash data analysis and graphic representations. By designing the tool specifically for automated
data flow in data acquisitions and analyses the engineers can be freed from the time-consuming data manipulation work to focus their main effort on investigating the causes of crashes and identifying weaknesses in the design and control of the roadway system.

1.3 SOFTWARE OBJECTIVE

The objective of the AVADIT program is to create collision diagrams and provide data integration functions for crash data analysis. Specifically, this tool is capable of automatically constructing collision diagrams, and generating data distribution tables and charts and figures for intersections, roadway segments, counties, and the entire district. Specifically, this program is developed to:

1. Provide a mechanism to build a system through which different clients located at different sites running on different platforms can download the required crash data from a central database to their local database.

2. Build a database management system that allows users to access and manipulate their local database.

3. Implement data integration that links the data fields to graphic elements and charting functions.

4. Create an object-oriented, event-driven, platform-independent program that allows users to control the tool by easily pointing-and-clicking a mouse [5].

This tool is written in JAVA, which makes this program platform-independent, so it can be easily installed on mainframes, UNIX workstations, or PCs. The program is easy to follow and most of its functions are self-explanatory through drop-down menus.
CHAPTER II
FUNCTIONAL REVIEW

2.1 SOFTWARE BENEFITS

The greatest benefit of this software is to automate and streamline the data flows in crash data studies. By creating an any-to-any internal data structure, AVADIT enables a dynamically coherent data flow between functions of the program, even though they may reside in different modules.

Initial testing of the program had shown that a total of 100 crash records can be processed in five minutes, including running a database query, generating collision diagrams, and displaying charts off selected data, etc. Although the way in which the correctness of data is checked and graphics presented may be further customized to better address the needs of the users, it is anticipated that the AVADIT will serve as an effective and efficient tool to save hundreds of hours of time and minimize frustration due to manual or semi-automated data handling.

2.2 SYSTEM ARCHITECTURE AND FUNCTIONALITY

AVADIT is designed in modules. The modular design of this software ensures the independence and flexibility of this model. For example, should the format of the accident data files be changed in the future, the tool can still serve its full capability after the corresponding changes are made in the data dictionary and the rule base.

Functionally, this model can:

1. Display the detailed collision characteristics for crashes.
2. Construct collision diagrams depicting crashes for an intersection or a segment of roadway.

3. Perform data distribution analysis, and generate charts.

4. Provide report generation capabilities and allow the users to select tables, charts and collision diagrams to be included in the report.

5. Allow the users to print out the art-work or save it into images for other applications.

2.2.1 DATA FLOW

The architectural and data flow design of the AVADIT model is shown in Figure 2.1. There are four data flows among functional units and each flow is depicted differently from the others, including:

(1) *On-line network access flow* – to download requested data from the remote central database into the local database.

(2) *Local database operation and data manipulation flow* – to access and manipulate the data in the local database, organize and prepare data for further applications.

(3) *Supporting data and graphic flow* – to transport the graphics prepared by the Collision Diagram module and Data Analysis module to the Report module for report printing or file saving.

(4) *Command and control flow* – not shown but embedded in all flows to communicate and execute commands for various types of applications.
The above flows represent all the major information exchange and integration involved in the initiation and application of the model functions. Obviously the effectiveness of the model depends on efficient acquisition of data from the central database.
Figure 2.1 Logical Data-Flow Chart
2.2.2 CLIENT-SERVER SYSTEM

A client-server data communication system was selected as the enabling platform for data acquisition [2]. Such a system has the advantage of sharing data and device, and has the flexibility and reliability necessary for frequent download of various kinds of crash data from the central data system.

The primary units of this client-server model include a data-acquisition unit, a local database and a main user interface that integrates the component modules of the model (Figure 2.2), where,

- **Data Management System (DMS) Interface** is to display the client interface if the user starts the client-server model for data download.
- **Client Interface** is responsible for
  1. building parameters (location, time duration, type of crashes etc.) used for the client routine to specify a certain set of data to be downloaded and
  2. calling the client routine to send data download request to the remote server where the central database is maintained.
- **Client Routine** is to
  1. packet the parameters obtained from the client interface to form a download data request message;
  2. build a connection to the remote server and send the request to the server;
  3. create a local database and save the download data into the local database.
- **Server** executes a set of programs running at the central data system to provide
  the following services:
  
  (1) maintaining the central database;
  
  (2) accepting client’s connections and data download requests and sending the
data to the clients.

- **Central Database** is the database that contains the crash records. These records
  are archived and cannot be changed by the user for the purpose of data
  consistency and integrity.

- **Local Database** is the dataset downloaded from the central database and resides
  on the Client’s machine. This database can be modified, queried, sorted, plotted,
  and graphed.
Figure 2.2 Client-Server Model
CHAPTER III
PROGRAM MODULES

The individual modules interact with each other and each can be activated/executed depending on the type of applications selected by the engineers. The AVADIT model was written in Java to ensure platform independence. The program modules are discussed in detail in the following sections.

3.1 STARTUP MENU

The startup menu allows the user to access from one main menu the Crash Data Acquisition module, the Collision Diagram module, the Data Analysis module, and the Report module. An Exit button exits the program while the other buttons in the startup menu execute separate modules.

3.2 CRASH DATA ACQUISITION MODULE

The primary function of the Crash Data Acquisition module is to extract data from the main data source. Based on the county, date of interest, route number, and straight-line-mileage (SLM) limits selected for the study, appropriate data are identified and sorted out for download and review. The crash data can be either in the single table format or multi-table (Crash, Units, People) format in the central database. A local database is created by the module for the engineers through the following:

- Developing Structured Query Language (SQL) scripts according to the data specifications of the engineers.
• Executing the SQL scripts to pull the selected data and creating a local database to save the downloaded dates into the local database. In the entire data downloading process, there is no need for operator's intervention at the central data location. An ordinary Pentium PC at a local district will have enough memory power and disk storage space to run multiple applications and save the data files locally.

With the local database, the engineers can manipulate the data in many ways. For example, data queries can be run based on specific features such as Manner of Collision (Head-on, Sideswipe, Rear-end), Type of Units (Heavy trucks, Pedestrians, etc.,) and Severity (Fatal, Injury, Property Damage). Collision characteristics can be further grouped and studied through single or nested sorting operations on any type of data. The crash data can be exported either as a text file or an image to be included in any word processor.

Graphical User Interfaces (GUIs) have also been used in this module for the user to interact with the database. The local database is displayed as a two dimensional (rows and columns) grid. User interactions with the grid are handled as events and the GUIs are event-driven functions for various kinds of data manipulations such as described above.
3.3 COLLISION DIAGRAM MODULE

The primary function of this module is to construct collision diagrams. This module shares data from the Crash Data Acquisition module and interprets the collision code to depict graphically the conflict of vehicles involved in the crashes. Visualization not only brings human perceptual processes to bear in organizing and understanding data about traffic crashes, but is also required to provide an interactive environment in which the user can modify, rearrange, and seek additional information for an enhanced study of crash characteristics. A major challenge is therefore to develop a coordinate system, transformation methods, or structures that meaningfully organize graphics to represent traffic crashes.

A rule component is first built into the system and it has a rule base and a rule interpreter. The rule base captures the coding rules of the crash data. The rule interpreter uses the rules in the rule base to encode the graphical elements. The rule interpreter in the system is implemented to parse the rules in the following format [1]:

\[
\begin{align*}
<\text{rule}> & \ ::= \ <\text{premise}> <\text{action}> \\
<\text{premise}> & \ ::= \ ($\text{AND} <\text{condition}> ....<\text{condition}>)$ \\
<\text{condition}> & \ ::= \ ($\text{OR} <\text{condition}> ....<\text{condition}>)$ | ($\text{RELATION} <\text{field}><\text{value}>$) \\
$\text{RELATION} & \ ::= \ < | \ = | \ >= | \ <= \\
<\text{action}> & \ ::= \ (\text{set control variable})
\end{align*}
\]

For each crash record in the database, its field values are evaluated in the \<$\text{condition}$\> part of the rules. The evaluation of the conditions of a premise yields a TRUE or NIL
value for the premise. Any “TRUE” premise results in an action to set the values of a control variable. The collection of all the control variables is then used by the system to construct and display an element that corresponds to the data record.

A graphical element is composed of graphics primitives – points, lines, shapes, pixel maps of vehicle icons, traffic signs, etc., to represent any application through a combination of them. The generation of the graphical element is controlled by the construction rules in the rule base. A constructed element is placed into the final display according to the display rules, which control the element’s location, size, color, orientation, and its position relative to other elements and other displayable properties.

With this module the engineers can review automatically generated crash displays from selected data. Specifically, the engineers can:

- Display the collision diagrams for an intersection, or crashes that occurred along a segment of roadway.
- Display detailed collision related information upon selecting any single crash in the collision diagram, such as vehicles/pedestrians involved, direction of travel, road condition, driver condition, etc.
- Create customized graphics with a library of picture templates including traffic signs, pavement markings, and roadway/intersection configurations.
- Modify the diagram according to the user’s preferences or due to data error or insufficient data.
3.4 DATA ANALYSIS MODULE

The AVADIT model is capable of visualizing the crash data statistics in many ways. The Data Analysis module summarizes data and generates crash data statistics through a variety of different charts and graphs. Many of the graph properties can be changed, such as the colors, height and depth, and the titles of the axis. Different types of graphs require different numbers of columns from the database. For instance, a regular Bar chart only requires a single column where as a Stacking Bar chart requires two columns to be plotted against each other.

In order to clearly display data characteristics in graphs, a data mapping is used to convert raw number values to meaningful text representation of each value. This mapping is achieved through creating a column alias file, column.map. More discussion on this file is given in section 4.4.

The JClass from the KL Group was used to create the graphs in AVADIT. JClass Chart functions are versatile and ease of use. They support a wide variety of scientific and standard graphs and there are a variety of different ways to package the data into a graph. The charts produced are of similar quality to those generated by a spreadsheet software such as Microsoft’s Excel.
3.5 REPORT MODULE

The reporting module is designed to generate rich graphic reports that include various output from the AVADIT model. The user can include images of the crash diagrams from the Collision Diagram module and graphs from the Data Analysis module. Text elements of various fonts and sizes, can be inserted anywhere into the report. The database data can be included into the final report by saving a specific table from the Crash Data Acquisition module as a file. Since the user might not want all of the columns of the database file included in the final report, we allow the user to specify which ones are to be included by using the Column Selection Dialog as seen in the Crash Data Acquisition module. After the various elements are added into the Report Editor, the user can press the Generate Report button to create a report file in the HTML format.

The HTML format was chosen as our report format because of its extensive industry support. Most computers are accompanied by a web browser, which can view the HTML format. The viewers can also use basic editors to further customize the final report. Because of its HTML format, the finished report can be added to the World Wide Web to maximize its viewing audience.
CHAPTER IV
RULE BASE AND DATA DICTIONARY

To make possible data integration and visualization, a data dictionary and a comprehensive rule base are used to determine the data characteristics, and the type, location, and relative position of the graphic elements. This Chapter describes the details of the data definition and visualization rules.

4.1 DATA DICTIONARY

The AVADIT model is capable of handling a variety of different data types when retrieving the crash data from a remote database. To accommodate different types of data format used in the remote data, the data are reformatted after they are transferred to the local site. To accomplish this we have created a mapping between the field names and the needed data types. When AVADIT retrieves the data from the remote site, it will convert the data from whatever format they are in to the format specified in the mapping. The mapping between field name and data type is held in the file “Avadit2.ddf” in the main program directory. The file is laid out with field names on the left side of the equal sign and the data type to the right. Here is an example of data mappings from the file:

```
  ROWNUM = INTEGER
  Local_Report = VARCHAR (100)
  County      = INTEGER
```

4.2 DISPLAY RULE BASE

A display rule base is used to preprocess the data before generating collision diagrams. This rule base contains a number of rules that are developed according to
traffic operation regulations and the coding standards, and are automatically applied once the Collision Diagram module is launched. These rules can be classified based on their function into following types,

1) North arrow rotation rules
2) Conversion of vehicle direction to "virtual direction" based on the mainstream traffic direction.
3) Collision display rules.
4) Zoning rules.
5) Validation rules.

4.2.1 NORTH ARROW ROTATION RULES

As a user preference and in order to simplify graphic display variations, the mainstream traffic flow is always displayed as if the traffic was moving from the left side (West) to the right side (East) of the display in the collision diagram, irrespective of its actual from and to directions. To correctly represent the traffic directions, a North arrow is used which rotates counterclockwise to point the actual north direction in the collision diagram. The default North arrow is designed as \( \uparrow \). Figure 4.1 shows the one-to-one relation of main-stream traffic direction to North arrow display.
### Figure 4.1 North Arrow Rotation

<table>
<thead>
<tr>
<th>Main Stream Traffic Flow Direction</th>
<th>North Arrow Display After Rotation</th>
<th>Rotation Angle</th>
</tr>
</thead>
<tbody>
<tr>
<td>West to East</td>
<td>↑</td>
<td>0°</td>
</tr>
<tr>
<td>North-West to South-East</td>
<td>←</td>
<td>45°</td>
</tr>
<tr>
<td>North to South</td>
<td>←</td>
<td>90°</td>
</tr>
<tr>
<td>North-East to South-West</td>
<td>←</td>
<td>135°</td>
</tr>
<tr>
<td>East to West</td>
<td>↓</td>
<td>180°</td>
</tr>
<tr>
<td>South-East to North-West</td>
<td>←</td>
<td>225°</td>
</tr>
<tr>
<td>South to North</td>
<td>→</td>
<td>270°</td>
</tr>
<tr>
<td>South-West to North-East</td>
<td>←</td>
<td>315°</td>
</tr>
</tbody>
</table>

4.2.2 VEHICLE DIRECTION RULES

In order to minimize the number of graphic icons that depict the vehicles coming from different directions, a “virtual direction” is used to describe a vehicle so that the same icon can be used to represent all the vehicles having the same pre-crash action (i.e., going straight →, for example). This is done by considering the main-stream direction as from west to east and changing the direction of each vehicle into its virtual direction. For example if the roadway is coming up SLM increasing from south to north, then it is considered as if it was coming from west and the other directions are changed accordingly, as shown in Figure 4.2.
Before the Directions are Changed

Virtual Directions for Vehicle from South to North

Figure 4.2 Virtual Direction Based on the Main-Stream Traffic Direction

4.2.3 COLLISION DISPLAY RULES

The purpose of these rules is to select the appropriate graphic icons and position them based on the direction of movement and Pre-Crash Action of each vehicle. The relative positions of the icons to each other are determined by the Point of Impact of each vehicle involved in a crash. The severity of each crash along with the vehicle at fault are also to be displayed. Specifically, the collision display rules include:

a) Identifying the type of vehicle or motorists or objects involved in the accident and the Pre-Crash Action of each unit involved, then obtaining the corresponding graphic element from the library of templates.

b) Based on the main-stream traffic direction and the direction in which each vehicle is coming from, the icon selected is rotated to an appropriate angle.
c) Considering the point of impact of each colliding vehicle, relative positioning of its icon to others is determined and the type of conflict displayed.

d) The vehicle at fault is displayed by attaching a line to the tail of its icon as shown in Figure 4.3.

e) The severity of accident is represented by

   i. an open circle when severity is injury;
   ii. a filled circle when severity is fatal;
   iii. displaying nothing when injury is PDO—property damage only.

Examples of crashes with different severities are shown in Figure 4.4.

<table>
<thead>
<tr>
<th></th>
<th>Vehicle not at fault</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Diagram" /></td>
<td>Vehicle at fault</td>
</tr>
</tbody>
</table>

Figure 4.3 Graphical Representation of Vehicle at Fault
### Figure 4.4 Graphical Representation of Crash Severity

<table>
<thead>
<tr>
<th>Accidents</th>
<th>Severity</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image.png" alt="Diagram" /></td>
<td>Injury</td>
</tr>
<tr>
<td><img src="image.png" alt="Diagram" /></td>
<td>Fatal</td>
</tr>
<tr>
<td><img src="image.png" alt="Diagram" /></td>
<td>Property Damage</td>
</tr>
</tbody>
</table>

#### 4.2.4 ZONING RULES

It is necessary to display the collision icons at a place in the collision diagram where a certain type of collisions is most likely to occur. Based on this an intersection is divided into a number of zones for displaying the collision icons. For example, a four-leg intersection is divided into twelve zones which include four center zones (with the same SLM number) and eight approaching zones (as shown in Figure 4.5). Allocating the collision icons to the collision zones follows a set of established zoning rules as in Figure 4.6. These rules are developed according to the type of vehicle movements [3] summarized in Figure 4.7. For example in case of a four-leg intersection the following steps are used:
Zones - 1, 2, 3, 4 : Center Zones

Zones - 5, 6, 7, 8, 9, 10, 11, 12 : Approaching Zones

Figure 4.5 Designation of Collision Zones for a Four-Leg Intersection
Figure 4.6 Zonal Designation of Collision
Figure 4.7 Traffic conflict at a Four-Leg Intersection
Step 1 – get Vehicle Type, Direction of Travel and Pre-crash Action for the first vehicle; identify the graphic icon to represent the first vehicle involved in the crash;

Step 2 – get Vehicle Type, Direction of Travel and Pre-crash Action for the second vehicle and Number_of_Units involved in the collision; identify the graphic icon to represent the second vehicle involved in the crash;

Step 3 – get Point_of_Impact for each of the vehicles involved in the crash, and position the graphic icons relative to each other to depict the collision.

Step 4 – get Vehicle Factor of the vehicles to identify whether they were at fault.

Step 5 – get the Direction of Travel and graphic icon number of the first vehicle to identify the angle of rotation to be provided;

Step 6 – get the Direction of Travel and graphic icon number of the second vehicle to identify the angle of rotation to be provided;

Step 7 -- get Direction of Travel, Vehicle Type, and Pre_Crash_Action of both the vehicles to identify the zone where collision may occur. If no zone is found to display a crash, it will be placed in the Unplottable Zone;

Step 8 – get crash Severity;

Step 9 – display the selected graphical icons at the selected zone;

Step 10 – stop if all crash records have been analyzed. Otherwise, back to Step 1.

The above procedure was implemented on the local database. For example, following the denotations for vehicle movements in Figure 4.7 and the coding convention for Pre-crash Action, Point of Impact, as well as Crash Severity in Table 4.1, a crash involving
the front center of an eastbound vehicle going through the intersection, hitting the right side of a westbound left turning vehicle, resulting in injuries, would be displayed as

Display: zone (4)

Table 4.1 Coding Convention of Collision Characteristics

<table>
<thead>
<tr>
<th>Pre-Crash Action</th>
<th>Vehicle Impact</th>
<th>Severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - going through</td>
<td>1 - no impact</td>
<td>1 - fatal</td>
</tr>
<tr>
<td>2 - backing</td>
<td>2 - center front</td>
<td>2 - injury</td>
</tr>
<tr>
<td>3 - changing lanes</td>
<td>3 - right front</td>
<td>3 - property damage</td>
</tr>
<tr>
<td>4 - overtaking/passing</td>
<td>4 - right side</td>
<td>4 - unknown</td>
</tr>
<tr>
<td>5 - turning right</td>
<td>5 - right rear</td>
<td></td>
</tr>
<tr>
<td>6 - turning left</td>
<td>6 - center rear</td>
<td></td>
</tr>
<tr>
<td>7 - making U-tum</td>
<td>7 - left rear</td>
<td></td>
</tr>
<tr>
<td>8 - entering traffic lane</td>
<td>8 - left side</td>
<td></td>
</tr>
<tr>
<td></td>
<td>9 - left front</td>
<td></td>
</tr>
</tbody>
</table>

4.2.5 VALIDATION RULES

Validation rules represent a very important part of the procedure in collision diagram construction. These rules are all based on the new OH-1 format (as of 10/1999), and are used to check crash data for several possible types of errors shown below.

- Values exceeding predefined limits -
  
Since each field in the database can take only a given set of values according to the new OH-1 format, any value outside this range is an error.

- Inconsistency in motorist and non-motorists specifications -
  
Vehicle Type, Vehicle Factors and vehicle Pre-Crash Action are classified into two categories Motorist and Non-Motorist. A vehicle which is classified as in the
motorist category should have its values for Vehicle Type, Vehicle Factors and Vehicle Pre-Crash Action in the defined limits for the same category. The same rule applies for non-motorist also. Any crash record which does not satisfy this rule is considered containing an error in the data.

- Inconsistency between Point of Impact and Manner of Collision -
  Each type of Manner of Collision allows only certain points of impact. For example, for Manner of Collision "rear-end to rear-end", there would be an error when the Point of Impact of either or both vehicles indicates a front or side.

- Inconsistency between Vehicle Pre-Crash Action and Direction of Movement -
  Collision of vehicles while moving in certain directions occur only in a limited number of cases. For example, an error in the data is obvious if both vehicles are coming from opposite directions but one Pre-Crash Action is specified as overtaking.

- Inconsistency between Direction of Movement and Point of Impact -
  The Point of impact of vehicles in a crash is affected by its direction of movement. For instance, vehicles moving in the same direction will rarely have a head-to-head collision.

4.3 PREFERENCE VALUES

AVADIT is flexible enough to handle changes to the field names of the remote database. It also supports the feature to change the display of the collision diagram as preferred by the user. These features are provided in the preference file, Avadit.prf.
Avadit.prf is a text file located in the root program directory, and contains a list of the various options and configurations used in the program.

4.4 COLUMN MAPPING

It is very helpful to have a data mapping when browsing the database so that text instead of numbers can be viewed. We have developed a parser that reads in the “Column.map” file in the main program directory and provides the conversion between the data and meaningful text strings. The Column.map file is formatted in blocks, each block contains the mapping for one column of data. Here is a sample block from the “Column.map” file:

```
Column = DAY_OF_WEEK
1 = Monday
2 = Tuesday
3 = Wednesday
4 = Thursday
5 = Friday
6 = Saturday
7 = Sunday
0 = UnSpecified
-1 = N/A
END
```

Each block starts with the word Column followed by an equal sign and the name of the column to be mapped. Each line following contains one mapping with the data value on the left side of the equal sign and the meaningful text to the right. A single END keyword marks the end of the mapping block. Every block has the 0 and -1 data value.
Any data not specified in the police report is considered as UnSpecified. All the related fields to other vehicle units, in the case of single vehicle accidents, are considered not applicable, and assigned -1.

Column.map is a text file and can be modified should a change in the mapping become necessary.
CONCLUSION AND FUTURE WORK

The AVADIT model was developed in this project based on database theory and computer graphics techniques and traffic operation principles. It is capable of automatically generating the collision diagram and performing integrated data analysis.

With its modular design, the model can easily adapt to changes in the format of crash data or the data coding rules. The use of the Java programming language also makes the model platform independent.

Although in our limited testing the model has proven to be effective and efficient in data processing and graphical analysis, additional work is expected to further increase its functional capabilities and reduce possible programming bugs that cannot be avoided for such a comprehensive program in Java today. Additional validation rules may be developed and graphic presentations further customized to better address the needs of user. As many states are moving towards the use of latitude and longitude references in crash data reports, revisions should be incorporated in the near future to integrate the model with a GIS database instead of using the current SLM reference system.
REFERENCES


