Proceedings of the 1st Human-Centered Transportation Simulation Conference, The University of Iowa, Iowa City, Iowa, November 4-7, 2001 (ISSN 1538-3288).

A PROTOTYPE TOOLSET FOR RAPIDLY REDUCING AND REVIEWING DRIVING SIMULATOR DATA

Yiannis Papelis
National Advanced Driving Simulator
The University of Iowa
Iowa City, IA, USA 52242-5003
(319) 335-4597
yiannis@nads-sc.uiowa.edu

Matt Schikore
National Advanced Driving Simulator
The University of Iowa
Iowa City, IA, USA 52242-5003
(319) 335-4794
schikore@nads-sc.uiowa.edu

Ginger Watson
National Advanced Driving Simulator
The University of Iowa
Iowa City, IA, USA 52242-5003
(319) 335-4679
gwatson@nads-sc.uiowa.edu

Tad Gates
National Advanced Driving Simulator
The University of Iowa
Iowa City, IA, USA
(319) 335-4752
tgates@nads-sc.uiowa.edu

Submitted August 2nd 2001.

ABSTRACT

In our experience, one of the most challenging aspects of conducting research using driving simulators is the post-processing of simulator data, a process often referred to as data reduction. This step is necessary to produce various numeric measures that in turn are used for statistical analysis. Although commercial software packages can often be used to facilitate the process, such tools are not explicitly aware of the nature of the data and the intricacies of the scenario used to produce it. This knowledge is often necessary to effectively manage the data. These packages also have limited usability for performing simulator-specific calculations (i.e., lane deviation) or linking data based on associations between research participants or treatments. This paper describes an integrated toolset that has been designed explicitly to handle data reduction calculations for driving simulator or similar data. The toolset is tightly integrated with other simulator tools and can intelligently handle binary as well as video, audio, questionnaire, and other data typically collected during simulator-based experiments. The toolset is explicitly aware of the nature of each data file, its contents, and its association with other data files. A prototype of the tool has been constructed and is presented as part of the paper, along with some technical challenges and how they were addressed.

INTRODUCTION AND MOTIVATION

When developing technology associated with driving simulation applications, a lot of emphasis is placed on the raw computing capacity of the hardware or on the cueing capabilities of a simulator necessary to conduct various research studies. There is significantly less work contributing to the ability of a researcher to process, manipulate, and analyze the data after an experiment has been completed. This post processing step, also referred to as data reduction, involves the computation of quantitative variables that can be used to perform statistical analysis on primary and secondary variables of a research study. The difference between data reduction and statistical analysis is

in the data handled by each process. Whereas statistical analysis generally deals with the primary endpoints of a research design (i.e., number of collisions, reaction time, following distance, number of lane excursions, etc.), data reduction is responsible for processing raw data as collected by the system (i.e., vehicle position and velocity, scenario vehicle positions, etc.) and produces the data utilized for statistical analysis.

In working with simulators for 10 years or so, we have identified several key challenges associated with post-simulator data collection activities such as data reduction, data visualization, data handling, and data storage and retrieval. These challenges are not necessarily representative of the environments of all simulators, and there may in fact be partial solutions in commercial tools, such as Matlab [1], or LabView [2] that provide a very flexible general purpose environment for performing arbitrary data calculations and graphing. However, to the best of our knowledge, there is no single set of tools that comprehensively addresses simulator specific requirements. Interestingly enough, these deficiencies appear to affect not only driving simulators, but other research activities that depend on the collection and analysis of data sampled across time such as instrumented vehicles, flight simulators, or similar devices. Work described in this paper attempts to address these issues though the development of a comprehensive toolset.

The remainder of this paper is organized as follows. Section 2 profiles the key issues in data handling and reduction that have been identified as hindering effective use of simulators. Section 3 describes a set of high-level requirements for a toolset that would largely address the aforementioned problems. Section 4 presents an overall architecture and preliminary specification for such a toolset as conceived at the National Advanced Driving Simulator (NADS) for use in upcoming studies involving the NADS itself and other simulators located within the organization. Section 5 presents the current state of development of this toolset. Finally, Section 6 presents a conclusion and near-term plans for the toolset's development.

KEY PROBLEMS ASSOCIATED WITH DATA HANDLING AND REDUCTION

Limited Availability of Useable Variables

Typically, primary and secondary variables of a research design are not directly produced by the simulator during operation. Sometimes the calculations may be too computationally expensive to be performed in real time. Other times, they depend on accumulation of data (i.e., an average value across subjects). Another reason is that frequently the need to have a particular variable was not identified until after data collection was completed. Given the diversity of driving simulation devices, there are exceptions. For example, certain simulators may produce a fixed set of variables that can either be used directly for statistical analysis or can easily be manipulated using common tools such as spreadsheets. However, in high-fidelity simulators used for a variety of research studies in diverse scientific fields, the types and nature of required variables are such that it is not possible to anticipate all of them a-priori. Unfortunately, when not given a choice, as in the case of a “black-box” type simulator that produces a fixed set of data, researchers will use what is available, potentially missing opportunities to identify results that require a deeper look into the experiment data collection.

One way to address this problem is to utilize custom data reduction software or tools that can be programmed as necessary to calculate all necessary variables. This, unfortunately, leads to additional issues described below.

Specification of Data Reduction Procedures

To circumvent the limited availability of variables, data reduction software can be used to produce any necessary variable, provided that enough raw data were collected by the simulator to allow the calculation. When these calculations exceed basic operations that can be implemented using a spreadsheet or within the statistical package, the software has to be developed by someone with intimate knowledge of the simulator output format and access to all databases containing information about the virtual environment. Generally, this is somebody other than the researcher. The problem now is that the researcher needs to articulate the requirements and define a specification that can be used by a programmer. This interdisciplinary communication can be problematic due to the different backgrounds and jargon of the two parties. Another problem is that specifications may be misunderstood or taken literally. For example, when asking for the average speed during a run, the researcher may not necessarily want to include the variable time that a subject remained stopped on the side of the road before starting. If the researcher was coding this calculation, he or she would likely identify this very quickly and adjust the calculation, maybe by eliminating the times when the speed was zero, something that a programmer not familiar with the research design

may not identify as an issue. Worse yet, high-level specifications may allow so much room for improvisation that two programs developed by two different programmers adhering to the same specification produce different results, and they are both right. Consider as typical examples the measures of lane deviation or steering wheel reversals. Both can be interpreted in multiple ways. For lane deviation, one interpretation is the perpendicular distance between the vehicle's CG and the lane centerline; another may be the nearest distance between the vehicle's body and the right lane line. Similarly, in computing steering wheel reversals, one has to pick a threshold of change necessary to consider it a reversal. Picking different thresholds may produce drastically different results.

One way to address this problem is to provide tools that allow researchers to specify data reduction procedures using a formal method that also serves as a software specification.

Correctness and Verification of Results

Another problem with utilizing custom software on a study-by-study basis is the lack of confidence about the correctness of the produced data. Very few people would question the results provided by SAS [3] or SPSS [4] for simple statistical calculations, provided that the inputs were correct. The reasons for this confidence are that these programs have been tested extensively by their developers before release and have been used by millions of people worldwide for several years. It is likely that programming faults affecting commonly used functions would have been identified by now. When using newly developed code, however, the level of trust cannot be as high. Factoring into that the rapid-pace environment under which data reduction code is developed on a project-by-project basis sharply increases the chances of having erroneous output due to programming errors.

There are two ways to solve this problem. One is to expend a significant amount of effort in manually and redundantly verifying output results [5]. Another is to develop tools that are flexible enough to perform reduction based on high-level specifications and use these tools extensively. Increasing use and bug discovery followed by fixes can slowly increase the trust on the tools' output with a corresponding reduction in the effort required to verify results by hand.

Delay in Delivery of Results

In almost all situations involving a scientist or engineer dealing with a design and the resultant data, there is a natural interaction loop between the person, the tools, and the data. For example, a car designer may put together a design, simulate its performance using some tools, observe the data, and use it to modify the design. In case of problems, the analysis tools may be tweaked to provide more insight into the design, allowing the designer to get closer to the desired outcome. Similarly, a researcher dealing with data produced by a driving simulator study often needs to analyze it and observe the results, and, in doing so, may decide to look at the parameters and modify the reduction process in order to get more detailed information or more powerful measures, then try again. Unfortunately, when the researcher cannot rapidly modify the reduction method, this iterative loop is broken. The situation severely limits the scientist's ability to exhaust the usefulness of data or extract as much information as would have been possible if direct and interactive modification of the data reduction process was possible. In a relatively new field such a driving simulation where performance measures have not been established or standardized, this limitation can severely affect the ability to effectively use simulators.

Again, a potential solution to this problem is developing data reduction tools that allow the researcher to specify and implement the data reduction with minimal dependency on programmers.

Lack of Specialized Tools to Handle Data in a Native Context

Effective handling of data requires that available tools are aware of the nature of the data and display it in a format that is easily understood by humans. Although graphing or tabulating data is often enough to gain a good understanding of basic data such as speed or trajectory, there are numerous simulator-produced data that could benefit greatly from customized representations. For example, consider observing the trajectory of the simulator's vehicle during a rear-end collision. Although an x-y plot of the lead vehicle and the simulator vehicle, as shown in Figure 1, may be useful, it does not provide much information about the relative position of the two vehicles when plotted on a common x-y axis. Augmenting this with a plot of the following distance versus time, as shown in Figure 2, does improve delivery; however, an animation of the two vehicles either in a top-down mode or a fully 3D-rendered virtual environment would significantly enhance the other delivery modes. There are a lot of similar

examples where driving simulator data can be best understood when displayed using techniques that explicitly account for the nature of the data.

Addressing this problem requires tools that understand the meaning of the data and can deliver it to the user in ways that augment traditional display methods such as plotting or tabulating.

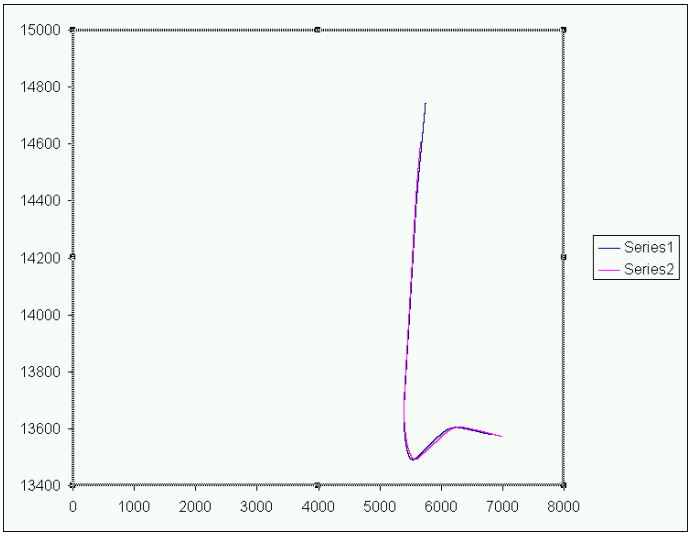

FIGURE 1 X-Y plot of the lead vehicle and simulator vehicle.

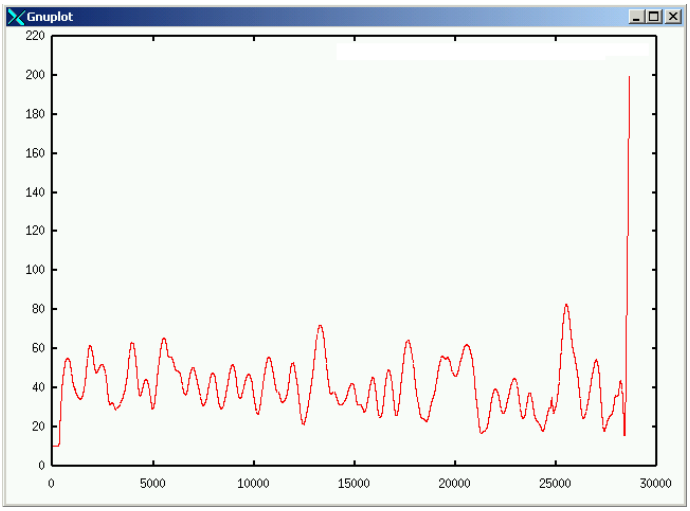

FIGURE 2 Following distance vs. time.

Maintaining Associations Among Distinct Data Streams

An issue related to presenting data in a context-sensitive manner is maintaining proper associations among data. For example, a researcher may want to view the portion of a video tape while the lane deviation exceeds a threshold. Without explicit support for such associations by a tool, a researcher has to manually record the lane deviation time, then find the appropriate tape containing the video and manually search the tape for that instant. Needless to say, this is very time-consuming when considering that even medium-sized simulator studies may involve tens of subjects, each of which often produces several video tapes.

Again, a possible solution involves the use of tools that are aware of the meaning of the data and can maintain proper associations between the various data streams and data files.

Configuration Management

One logistical problem that often hinders effective use of simulator data is the methodology by which data is stored, backed up, and delivered to researchers. Part of the problem occurs because of the sheer volume of data involved in simulator studies. Because of the number of variables and the hours of data and video collection that take place, it is not atypical for the required storage to exceed the capacity of the typical workstation found on a researcher's desk. Even though the continuing advances in hard disk technology may address the storage problem, networking technologies have not kept up and are quickly becoming the bottleneck. Simply copying data around the maze of typical office networks, firewalls, and virus-scanning interfaces can often take longer than the simulator took to collect that data in the first place. Other logistical problems, especially when multiple researchers require read/write access to the data, further complicate the process. Distributing data reduction and analysis activities may appear to ease the problem; however, it has been our experience that it in fact complicates it further because of the requirement of maintaining consistency and the need to coordinate activities. Maintaining access rights and keeping confidential subject information private also becomes an issue. Solutions often contradict the need to keep multiple copies of data to guard against loss. It is often necessary to keep track of the evolution of the data reduction and analysis. This overall problem of configuration management and version control is well known to software development professionals, but it may not be immediately obvious as a problem when dealing with data reduction and analysis activities following a simulator data reduction experiment.

Fortunately, a tremendous amount of effort has gone into addressing this problem in various fields, and the lessons learned, as well as existing software techniques, can be readily applied in driving simulator applications.

SOFTWARE TOOLSET REQUIREMENTS

This section presents a set of requirements for a software toolset that would address the key problems described earlier. There is a mixture of both high-level and lower-level requirements. When possible, examples are given to clarify the requirements.

A few of the requirements go beyond the toolset itself and may involve capabilities associated with data collection. In general, these requirements are based on the assumption that input binary data consists of a series of time-sampled records reflecting the state of the simulator's virtual environment at the time. Another key assumption is that all collected data can be digitized in a form that maintains its original content with no loss. In binary data, this is trivially true, but for other salient data such as questionnaires or consent forms, additional steps may be required to code questionnaires. This issue requires special considerations; however, it is beyond the scope of this paper.

- The front-end to the toolset should be a user-friendly graphical interface that allows the user full access to the toolset's functionality. Wizard-type interfaces should be used to help the user through commonly applied tasks such as importing data.
- The toolset should be aware of the nature of the data it is handling and, whenever possible, should provide context-specific operations for the data. For example, when dealing with the simulator's vehicle trajectory, in addition to generic matrix operations, the toolset should allow its use for lane deviation calculations. Conversely, the toolset should warn about operations that do not make sense, such as feeding the position of the gas and brake pedals into the lane deviation calculation. At a minimum, the following classes of data should be handled intelligently by the toolset:
    - Vehicle trajectories (simulator and scenario vehicles) and variables representing relations among them (following distance, time to collision, etc.)
    - Vehicle control inputs (gas, brake, steering, and clutch, if applicable)
    - Vehicle linear and angular velocities and accelerations
    - Relative and absolute time intervals
    - State of traffic lights
    - State of vehicle's instruments

  - Simulator and scenario vehicle light indicators (turn signals, brake lights, etc.)

- The toolset should be aware of units associated with all data with a known context.
- Based on domain-specific knowledge and additional user-specified constraints, the toolset should have automatic error detection, both for the input and output data. Error data is data whose values exceed certain limits in range or rate of change. They occur either due to hardware problems during collection or due to erroneous manipulation or processing by the user. For example, a scalar representing velocity should not exceed reasonable velocities obtained by vehicles. The user should be able to selectively enable or disable warnings about potential errors.
- The toolset should be capable of replaying the evolution of an experiment, including the state of all entities around the driver, either on a 2D top-down view or in a full 3D-rendered view.
- The toolset should have an input module that is flexible enough to import record base data from any simulator or other data of a similar nature (for example, instrumented vehicles). For initial implementation, the toolset should be able to import simple ASCII data and NADS binary data files.
- The toolset should provide interface hooks that allow association of external programs for manipulating internal variables, for which the type is known. For initial implementation, the NADS software tools should be used within the toolset.
- The tool should support coded questionnaire data and scanned images of other forms used in the course of a study.
- The toolset should support various forms of digital video data, both as input and output. Supported formats should include MPEG and AVI.
- The toolset should utilize a formalism that can be used both to specify and to execute data reduction procedures. At the same time, the selected methodology should be simple enough that users with no extensive programming expertise can utilize the toolset.
- The toolset should support automatic generation of documentation based on the same formalism used for specification and execution.
- The toolset should contain a variety of built-in reduction procedures that can be easily used for frequently asked variables but are flexible enough to allow the user to specify arbitrary reduction rules.
- The toolset should allow a user to store existing data reduction specifications in libraries that can be reused later.
- The toolset should explicitly take into account the potential of dealing with large data sets.
- The toolset should support basic data viewing (plot, graph, etc.) of both raw and reduced data.
- The toolset should maintain data in a way that allows time correlation of variables across multiple data streams. Any display of data (plot, video, animation) should be easily time-linked with any other data stream. For example, this will allow a user observing a plot of a variable to link a particular point on the plot with a corresponding animation of the simulator at the time instant corresponding to that point.
- The toolset's output should support ASCII and HTML formatting. In addition, the toolset should produce output formats compatible with common commercial statistical and data-handling programs.
- The toolset should explicitly support the notion of missing data. For example, if partial data for a subject is missing (maybe because the subject did not finish a run), the tool should maintain this information and properly generate files that reflect the missing data, as opposed to simply skipping producing that data.

- The toolset should provide the ability of accumulating new data to an existing set. That data can include new data files, derived data, or any other file that the researcher may want to have as part of the data set and augment any data collected during a simulator run.
- The toolset should support file management that includes enforcement of per-user read/write access rights and version control of pre- and post-reduction data. An audit log database should be maintained that audits all access to the experiment's dataset throughout its lifecycle.
- In addition to quantitative calculations, the toolset should support the interactive specification of qualitative variables and their association with the remaining experiment data. This, for example, will allow a user to request observation of an event and then classify it as benign or severe based on his or her own experience and observation. Once the user has provided this information to the system, this classification should become available for further calculations.
- When user-specified qualitative variables are part of the calculations, the toolset should ensure that all user requests for responses take place as early as possible during the execution cycle. This will ensure that the user can provide their inputs as early as possible, leaving the program to execute in a batch mode for the remainder of the execution time.
- The toolset should support multiple users working on the same experiment at the same time.
- The toolset should provide the capability for the user to override any of the answers produced by the data reduction procedures. The tool should make a note of any overrides. This will allow a researcher to eliminate outliers or other data that should not be included in the final calculations.
- The toolset should have explicit knowledge of the totality of the input data. For example, the tool should know that a study consists of 20 subjects, each having two runs. That way, if only 10 subjects' data is available, the toolset should be aware of the missing items.
- The toolset should be able to manipulate the data even when partial data is available. This allows exercising the data reduction specification even while a study is still running.

TOOLSET ARCHITECTURE

Driven by the requirements described earlier, a system framework has been designed that supports simulator data storage, retrieval, backup, data reduction specification and execution, verification, and visualization. A high-level block diagram of the system is shown in Figure 3.

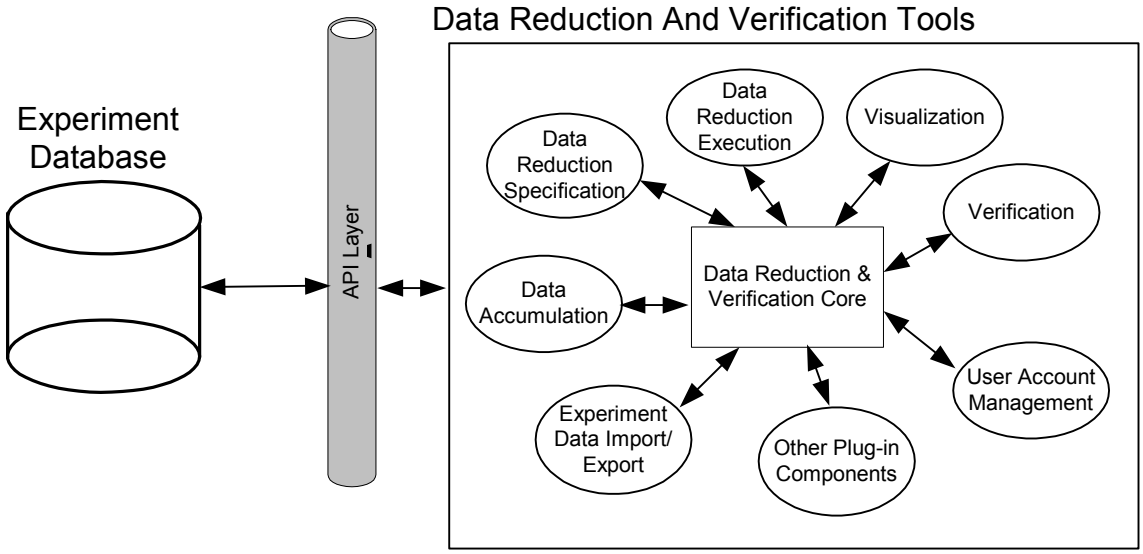

FIGURE 3 Data reduction toolset block diagram.

There are two primary components of the system: the Experiment Database (ED) and the Tools. The database consists of a set of storage servers and an API that controls access to all data stored there. The tools consist of a central core component and a collection of tools that interface to the core. Note that this decomposition is hidden from the user. In fact, even the interaction with the ED through the ED API is largely hidden from the user. From the user's standpoint, interacting with the whole toolset involves using a single GUI to build and execute data reduction specifications, along with a second administrative interface that allows creating new experiments and adding data to these experiments.

The remaining sections overview each of these components in more detail.

Experiment Database

The ED stores all data files associated with all experiments. The ED can contain an arbitrary number of experiments, constrained only by storage limitations. In addition, the ED maintains a directory of registered users and their credentials, which are used to authenticate users before allowing any type of access to the data files.

Each data file stored in the ED is identified according to a hierarchical triple Experiment, Subject, Run (ESR) that obeys a transitive inclusive relationship from left to right. Each experiment contains any number of subjects, and each subject can contain any number of runs. The transitive relationship implies that a file associated with an experiment only is automatically associated with all subjects and their runs. A file associated with an experiment and a subject is automatically associated with all runs of that subject. Finally, a file associated with a full ESR triple is associated only with that run.

In addition to their ESR association, each file is tagged according to the data it contains. Supported data types include binary data collection files containing time-sampled data, video files, and other meta-data consisting of text files, scanned document files, and questionnaire data files. Each file has certain properties and is treated in a manner consistent with its contents. For example, binary data files can be used to plot variables, perform calculations, etc. In addition to known types, user-defined file types can be specified within the meta-data classification, although the ED will simply manage them consistent with their ESR without any additional type-specific capabilities. Figure 4 illustrates how the hierarchical specification for an imaginary experiment would associate files of known types with the experiment.

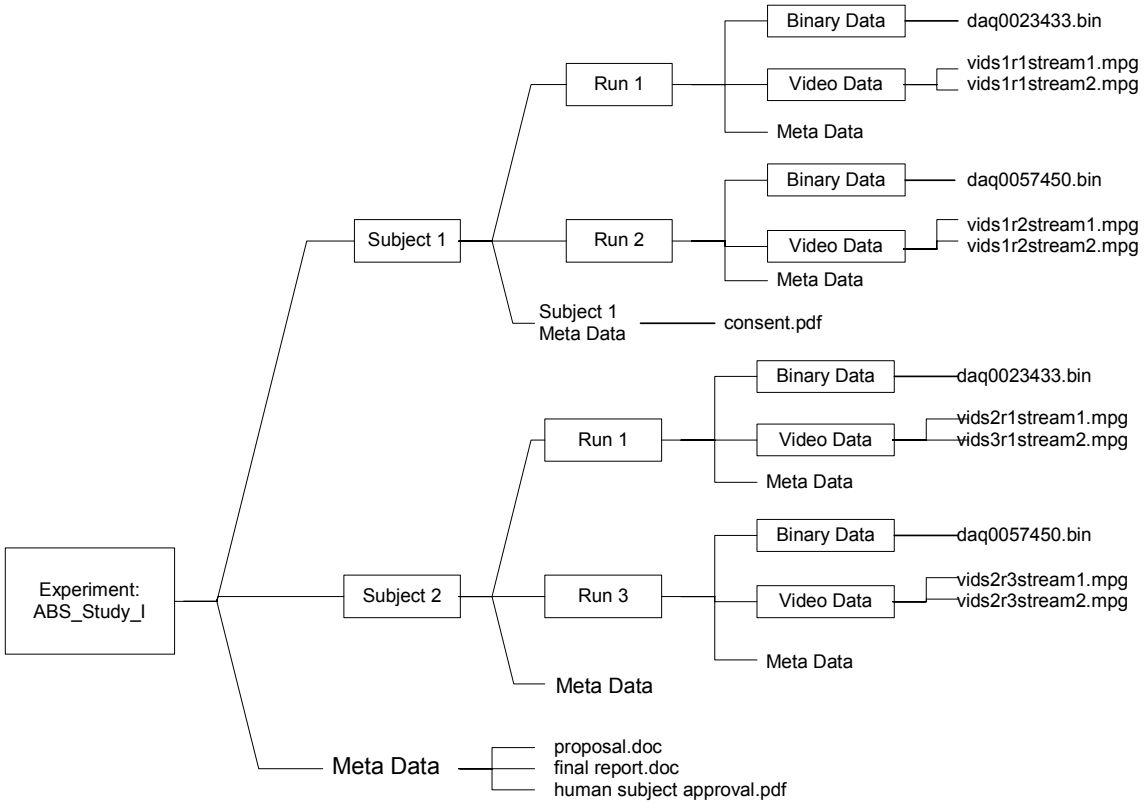

FIGURE 4 Example of hierarchical association.

In this particular example, the file "proposal.doc" is associated with the whole experiment, the file "consent.pdf" is associated with Subject 1, and the file daq0023433.bin is the binary data file associated with (ABS_Study_1, Subject 1, Run 1).

The ED is designed to meet all requirements that pertain to security, redundancy, and version control. This is achieved by routing all file create/delete/read/write/modify requests through an Applications Programming Interface (API) that validates access rights, enforces usage policies, and maintains versions of modified or deleted files.

Redundancy is maintained by keeping multiple copies of an experiment on multiple servers, all within the ED. Figure 5 illustrates how this duplication is achieved. The ED actually consists of multiple servers, each of which can maintain any number of experiments online. One of the servers is designated as the Master, and all other servers are designated as secondary. Any requests for initial access to an experiment are routed to the Master, which maintains a directory of existing experiments and their location among any available servers. The Master decides which server will handle the request and locks access to that experiment from any other server. Upon completion of the session, the Master is notified and initiates a propagation of changes to the remaining servers containing that experiment.

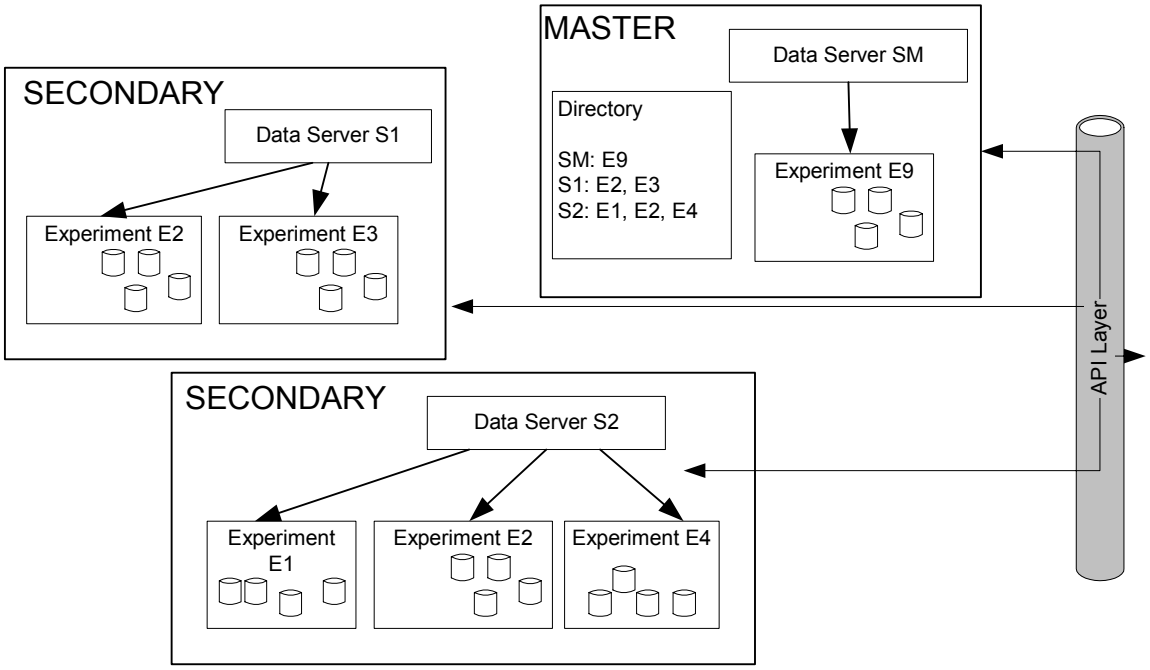

FIGURE 5 Redundancy in experiment data storage.

The degree of duplication is user-selectable on an experiment-by-experiment basis. For example, when the degree of duplication is set to 2, then two servers will maintain a copy of the experiment.

Data Reduction And Verification Tools

The Data Reduction And Verification (DRAV) tools component is a collection of components that implement the remaining functionality associated with the toolset. Specifically, a central Graphical User Interface (GUI) tool provides a consistent view of all available data and allows access, either directly or indirectly, to the remaining tools. All tools access the data associated with an experiment through the ED API. The decomposition of functionality among the various tools has been designed based on the typical sequence of actions that takes place in the course of an experiment. For example, the Experiment Data Import/Export tool is a stand-alone tool used for importing data from the simulator into the ED and for creating archival backup copies of an experiment on a CD or other removable media. This is the only tool that allows the creation of new experiments. Other tools are integrated within the front-end GUI (for example, various visualization tools are transparently integrated with the specification and execution tools). Despite the relatively large collection of tools, the presentation to the user is seamless, which ensures ease of use.

In order to satisfy the requirement of using a common framework that supports both specification and execution of the data reduction procedures, the decision was made to utilize a dataflow-based visual language for that purpose. The user specifies a dataflow diagram, which is then used to derive and execute data reduction procedures, which in turn produce the required data. The dataflow diagram approach to specifying computations is similar to what is used in Labview [2], however, the actual set of calculations in this toolset is focused on driving simulation applications. To avoid confusion with other dataflow diagrams, we use the term Data Reduction Dataflow Diagram (DRDD) to refer to the diagram supported by the toolset. An example DRDD is shown in Figure 6.

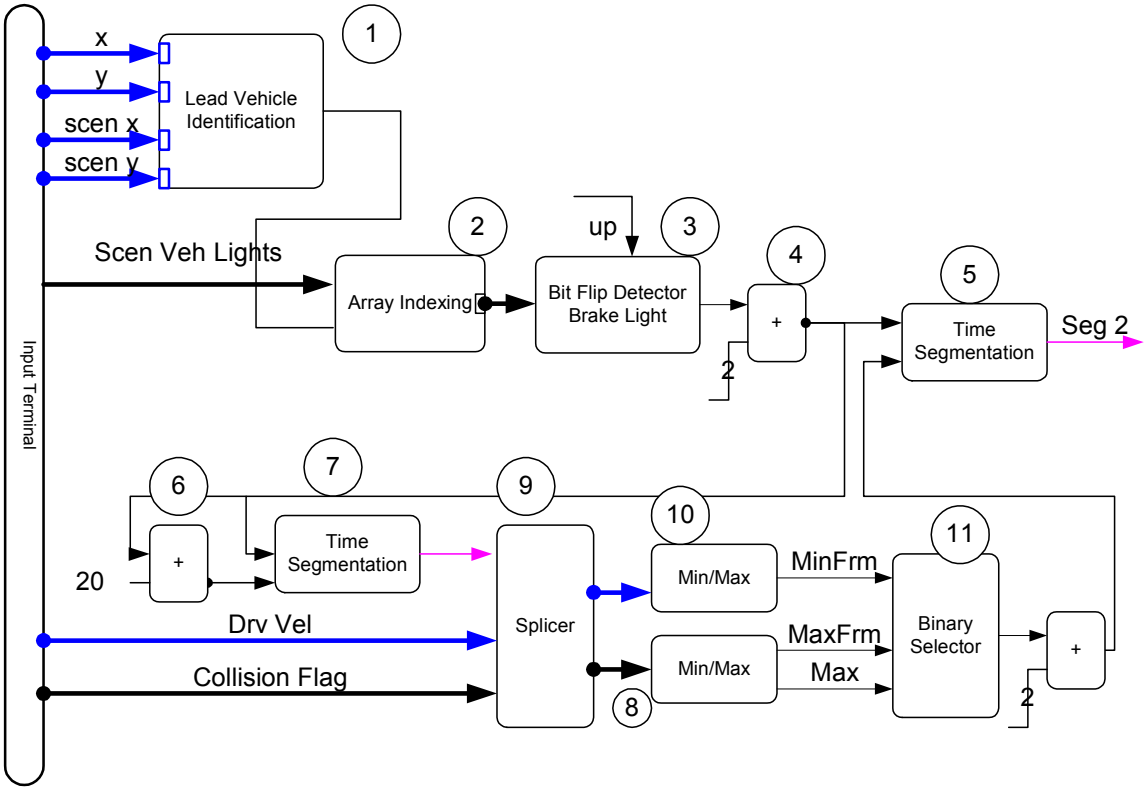

FIGURE 6 Example of DRDD.

There are two types of entities in the DRDD: nodes and arcs. As in all dataflow diagrams, nodes represent computations and arcs represent flow of data between computations. In the DRDD diagrams, the shapes of nodes are used to denote their function. Similarly, the thickness and color of arcs are used to denote the type and cardinality of data flowing through the arcs. Arcs can be thought of as carrying zero or one token. An arc with no tokens indicates that its data has not been computed yet, but an arc with a token has obtained its data from its input node. Nodes can have zero or more input arcs and zero or more output arcs. Nodes with no input arcs either provide constant data (like the constant nodes represented by circles as shown in Figure 6), or act as input terminals, which are nodes that import binary data from the input files. Nodes with no output arcs either produce output to a file or produce some interaction with the user.

Executing a DRDD is the key behind implementing arbitrary data reduction procedures, along with additional data tasks such as verification, data overrides, outlier removal, visualization, and assigning qualitative measures. All such activities are incorporated into the toolset though a set of DRDD nodes that, when executed, provide the required functionality. For example, numerous tools and data-specific computation procedures are all integrated through the common Data Reduction and Verification Core and appear to the user as different nodes within a DRDD. A database administrator's tools are primarily responsible for interfacing with the ED and allow creation of new experiments, perusal of existing files, and the addition or editing of files associated with an experiment. When combined with controlled access to the data through the ED API, this integrated toolset meets all requirements described earlier. Further information on the DRAV core software and how it meets these requirements is provided in the remainder of this paper.

Data Reduction and Verification Core

The DRAV core software consists of two logical components. The first is the front-end graphical user interface that allows DRDD editing and supports all interaction with the graph during both editing and execution. Figure 7 shows a picture of that tool's main window. The second component is the execution engine that can execute a DRDD specification along with the various plug-in components that perform customized functions such as plotting, displaying movies, etc.

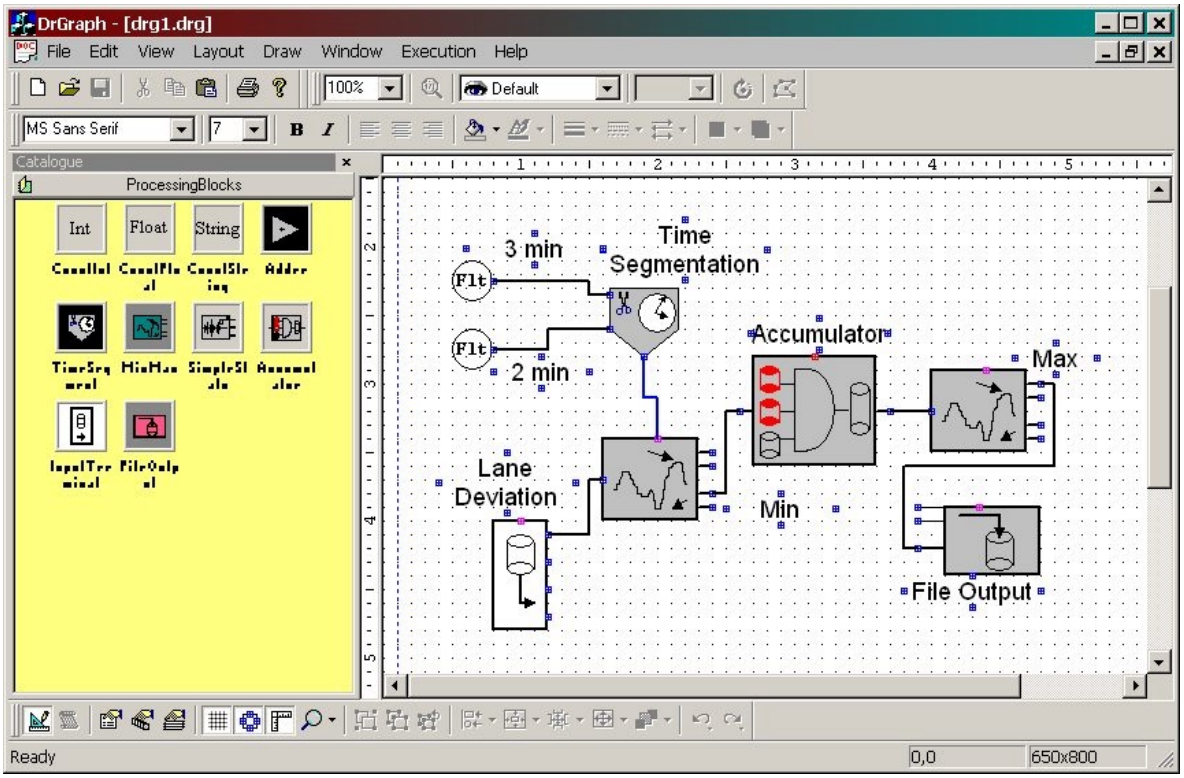

FIGURE 7 Example of DRAV Graphical User Interface.

The DRAV GUI allows the user to build DRDDs and performs basic error-checking of the developed graph. The concept behind the tool is that it guides the user toward building semantically correct diagrams. Whenever possible, the tool provides access to options through pop-up menus and lists so the user has little opportunity to create invalid graphs. For example, the tool will not allow the user to connect the output arc of a node to a mismatched input arc of another node. A detailed description of the DRAV GUI is beyond the scope of this paper, however, so the focus of the remaining sections is on the execution semantics of the DRDD and the capabilities of the various nodes that a user can use in the DRDD.

Execution Semantics The DRDD is always executed in the context of an experiment. This association provides input data files for input terminals. For example, consider an experiment consisting of two subjects, each with one run on the simulator. There would be a total of two input files, each containing time-sampled records. Executing a DRDD such as the one shown in Figure 6 after associating it with the example experiment would result in two sets of executions, one per input data file.

Execution of the DRDD proceeds by continuously scanning the nodes in the graph, finding all *ready* nodes and then executing them. A node is ready when all of its input arcs have tokens in them. Execution of a node means that the node uses the data associated with each input arc to produce data associated with each of its output arcs. After a node executes, all of its output arcs have tokens in them. Note that at any given time, more than one node could be ready. The actual choice of which node executes does not make any difference in the output and is made internally by the execution engine. Once all nodes have executed, the graph is said to have also completed execution.

Nodes representing constant values or input terminals are always ready because the data they require is available before execution begins. For an example of a DRDD execution, see Figure 6.

This example computes a segment that starts 2 seconds before the lead vehicle begins braking on the rural highway and ends either 2 seconds after a collision or 2 seconds after the driver has reached a minimum speed.

The block identifies the time that the lead vehicle puts on its brake lights and then subtracts two seconds from that time instance. Block 1 identifies the lead vehicle, and Block 2 produces a vector containing the light state of the lead

vehicle. The bit flip detector (3) produces the frame on which the brake lights turn on. The adder (4) subtracts 2 seconds from that time instant, and the resultant frame number is the beginning of the segment. For the end of the segment, the graph first partitions the data into a second segment that consists of the 20 seconds following the start of the braking maneuver (which is also the start of Segment 2). The driver velocity and collision flag are spliced so only the 20 seconds after the brake are available at the outputs of Block 9. Block 10 computes the frame on which the minimum driver velocity occurred. Block 8 computes the maximum value of the collision flag and when it occurred. Because the collision flag is 0 when there is no collision and 1 when there is a collision, the maximum will be 1 if there is a collision. That value is used as a selector in Block 11 that will pick either the frame on which the minimum driver speed occurred or the frame on which a collision occurred. The end of the segment is that value incremented by 2 seconds.

Minimizing Data Transfers Typically, data collection files originating from a simulator, instrumented vehicle, or similar source is their relatively large size. For example, consider a simulator study requiring 80 subjects, each exposed to 3 simulator runs of approximately 20 minutes each. Assuming that the simulator collects data at 30 Hz and each data record contains about 400 bytes, all data files would consume approximately 3.2 GBytes. The data can easily be stored on a modest-capacity workstation, but accessing that data can get very lengthy. For example, using today's typical workstation, going through all records to identify the minimum value of a data stream for each data collection period can easily take a few minutes. More complex computations require larger data movements, which in turn require more time. Generally, having to wait for an hour or two for data crunching after such an experiment would not be a problem; however, if a researcher is experimenting with different reduction methods or wants to slightly change the computation, then waiting for a couple of hours after each change becomes prohibitively expensive. The DRDD execution engine uses a special method for caching results that ensures that computations that already have taken place will not be redone unless something in the input data or in how the input data is handled has changed.

This technique depends on each node being able to compute a string that identifies its output before it actually produces its output. That way, before execution, the node can produce this string and compare it with the actual string associated with the output arc. If the strings match, then the node does not have to recompute its output.

Specifically, before each node executes, it creates an identity string that incorporates the identity strings of all of its input arcs and all parameters that affect how it would perform its calculations. It then transforms the input string into an output tag and checks the output arc to determine if it has a token that is labeled with the same tag. If the output arc does not contain a token or contains a token that is labeled differently, the node performs its execution as usual and then deposits the token to the output arc using the computed tag. Arc tokens and their tags are persistent across executions so that once a user has executed a DRDD, the tokens and tags for each node remain in the system (in fact, they are stored on the file system).

Figure S can be used to further illustrate this example. The lane deviation input terminal obtains the lane deviation stream from the input database. Before actually performing the transfer from the ED to its output arc, it would query the token tag from the ED, then augment that tag with a string containing any applicable parameters (in this case, none). It then generates a new output tag based on the composite string and compares it to the tag associated with the arc connecting its output to the Min/Max block. If the tags are the same, the node will do nothing; otherwise, it will contact the ED, obtain the data, and write it to the output arc's token. Similarly, the Min/Max block will first create a string consisting of the tags of the token in all of its input arcs (in this case, from the input terminal and segmentation blocks), augment it with any customized parameters and then generate its output tag, which it compares with its output arc's token.

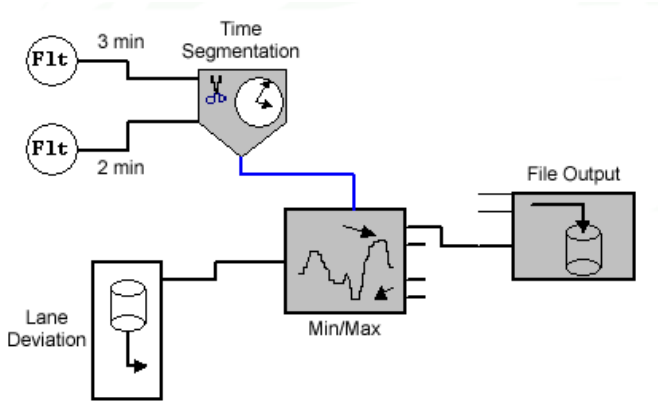

FIGURE 8 Graph Used to Illustrate Caching.

In the above example, note that once it executes all tokens will be produced (and remain on the system), and their tags will be defined. If the user reruns the code, there will be no data transfer from the ED and no new computations other than the production and comparison of the tags. Now let us consider that the user changes the 2-minute designation on the input of the time segmentation block to a new value. The tag produced from the time segmentation block will be different now. The composite string created by the Min/Max block for generating its output tag will also be different, thus triggering the actual computation. Still, no data will need to be transferred between the ED and the lane deviation input terminal. Overall, the time savings when dealing with large data sets where little of the graph changes can be tremendous, thus affording a researcher the ability to change the data reduction specification and obtain new data with as little re-computation as possible.

The only problem in implementing the above algorithm is finding a transformation function that, given an arbitrary-length string, will produce a short tag that uniquely identifies the input string, i.e., if any of the string changes, then the output tag should also be changed. In fact, the above problem is very common in cryptography [6], where such transformation functions (also referred to as *hash* functions) are used to produce digests that can be used to digitally sign documents, produce passwords for public key encryption algorithms, and similar functions. In fact, even the typical cyclic redundancy checksum (CRC) fits this criteria and could be used as the transformation key. However, improved algorithms exist that reduce the likelihood of producing duplicate tags for two different input strings to astronomically low levels of probability.

Segmentation Nodes Simulator data collection typically proceeds immediately after the commencement of the simulator run and terminates when some condition dictates the end of the run. However, when reducing the simulator data, one may only be interested in values collected for only a subset of the duration. For example, one may want the maximum lane deviation while the simulator driver was on a one-mile section of a highway, even though the whole drive is 20 miles in length. Alternatively, the brake and accelerator pedal position may be needed only during the two seconds after a lead vehicle brakes. We call the process of narrowing down the range of data records to be analyzed *segmentation.* Three types of segmentation are supported by the DRAV toolset. The first, time segmentation, is the simplest. It narrows down the data between two points in time. The second, and the one specified in the first example above, is position segmentation. In position segmentation, the endpoints of a range are specified based on the geographical position of the simulator driver within the virtual environment. The last, and the one specified in the second example above, is event segmentation. In this type of segmentation, the endpoints or a range are given relative to an event that occurred within the virtual environment.

Note that all segmentation types can be translated into two record numbers that provide the first and last record of the range. However, it is important to keep in mind that segmentation used in a DRDD applies uniformly to all input data files and, if events happened at different times for each subject, then the record range may not be the same for all data files. For example, consider position segmentation. Different subjects may reach the entry of a curve after a varying amount of time from the start of the simulation, so a fixed time will not work. By using position segmentation, the system itself will determine the frame numbers that mark the entry into and exit from the curve, but the frame numbers would be different for each subject.

The first two types of segmentation are supported by Time Segmentation and Position Segmentation, respectively. Event Segmentation is supported indirectly in that several blocks produce outputs representing record positions and can be used as inputs to a time segmentation block. The inputs to the time segmentation block are two times, expressed in seconds, that mark the start and end of a segment measured from the start of the scenario's execution. The position segmentation block allows the user to click on a top-down map of the virtual environment to indicate the actual locations that will mark the endpoints of the range. An example of using the Time Segmentation block is shown in Figure 12.

Input Terminals Input Terminals (ITs) provide access to the raw input data. An IT has no mandatory input arcs and, as a result, is ready to execute immediately after the start of execution. Upon execution, the IT deposits a token to each of its output arcs containing the corresponding data. Generally, an input data file will contain multiple streams of data. The IT allows the user to specify which stream of data to produce. A segment specification can also be provided as input to this block if, for example, only a subset of the data records is required. Figure 9 shows the graphical representation of an IT.

Computation Nodes Computational nodes allow general mathematical computations to be performed on scalar or vector quantities. In addition to general arithmetic operations, computational nodes can perform basic statistics such as average, median, standard deviation, and min/max. All computation nodes accept optional segmentation inputs that limit the range of their inputs. Figure 10 shows the graphical representation of various computation blocks.

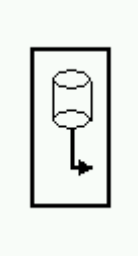

FIGURE 9 Input terminal.

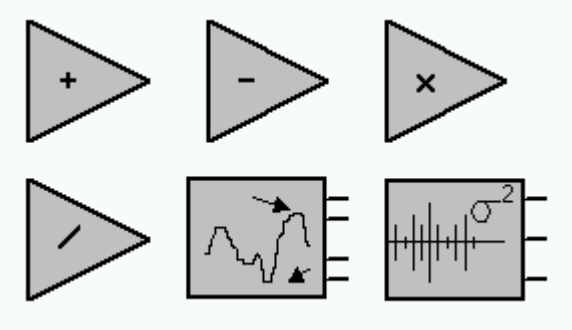

FIGURE 10 Computational nodes.

Output Nodes Output nodes are responsible for writing into files the contents of their input arcs using various formatting options. Usually, output nodes can be used to produce the final output files, which can be used as-is or imported into specialized statistical packages. Output nodes do not produce tokens and have no output arcs. The user can specify various options for how the data will be formatted. Several templates exist for supporting commercial statistical and spreadsheet software packages. Figure 11 shows the graphical representation of an output node.

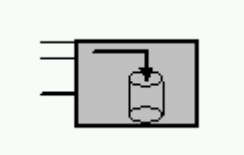

FIGURE 11 – Output node.

Data Visualization Nodes Visualization nodes are responsible for creating various plots and graphs that can be viewed interactively while a DRDD executes. Simply put, a visualization node accepts one or more input arcs and some configuration information and, when it executes, it creates a plot on the workstation's window of the input data. Special emphasis has been placed on facilitating verification activities, where a researcher may want to get quick plots of key variables for sanity checks or to independently verify quantitative results. Even though this functionality is not meant to replace plotting using specialized commercial packages, there are numerous options that allow customization of plots, including the ability of a user to specify where the plots will appear on the screen and how different subjects' data will stack across the screen (so repeated runs will produce plots in the same window location in a deterministic manner).

Visualization nodes do not halt execution, i.e., when a visualization node executes, it will create the window and render the plot while execution continues.

Movie Generation Nodes Movie generation nodes allow the creation of movies for visualization, verification, and documentation purposes. There are three movie generation nodes.

The first node is the movie extraction node, and its job is to create a movie file by subdividing another existing movie file. This is often needed when a researcher wants to create a collection of movie clips that show a specific maneuver or a key instant in a scenario provided that a data file containing a movie of the whole scenario already exists. For example, out of a 20-minute video clip, one may want to extract a 10-second clip that includes a collision avoidance maneuver. The movie extraction node, in conjunction with a segmentation node, can be used to extract this 10-second clip very rapidly.

The other two nodes create movies containing animations of the simulator's virtual environment for a specified duration. Unlike the movie extraction node, which depends on the existence of a movie file, these nodes produce a new movie file by rendering the simulator's virtual environment frame by frame and then combining these frames into a movie file. The difference between the two nodes is in the type of rendering used to show the simulator's virtual environment. The first type provides a top-down, two-dimensional rendering of the environment. The second type provides a fully 3D-textured view of the environment. Both nodes allow a segment specification along with various parameters to specify eye point and zoom control.

User Input Entry Points Blocks in this category are responsible for allowing the user to provide data that either augments (Data Augmentation) or replaces (Data Replacement) data produced by calculations that take place within the execution environment.

A Data Augmentation block has no inputs. Various optional parameters allow the user to specify the format and layout of a dialog box that includes feedback and input fields. When the Data Augmentation block executes, it displays the pop-up window and waits for the user to fill in the dialog box. Once the user fills in all required input fields, the data appears in the output arcs of the block and can be used in other calculations. The main goal of the augmentation block is to allow the user to provide numeric values that require human interpretation and cannot easily be computed. For example, a movie generation block can be authored before a Data Augmentation block to allow the user to classify the subject's behavior.

The Data Replacement block allows the user the opportunity to replace data in its input arcs. The modified, or untouched if the user didn't change them, data then appear on the output arc. This block can be used to allow the user to remove outliers or perform other such changes for whatever reason.

Both user input entry blocks are blocking, i.e., they halt execution until the user performs some action. To ensure that all blocks that require user interaction appear as early as possible during execution, the execution engine performs a special prioritization of the various nodes to ensure the earliest possible execution of these nodes. This ensures that even if the overall data reduction execution takes several hours, all nodes requiring user interaction will execute as soon as possible, thus minimizing the time the user has to wait before letting the workstation operate in batch mode.

Accumulators All nodes that have been specified to this point have the capability of performing calculations or interacting with data on a subject-by- subject basis. This is due to the nature of execution of the DRDD, which involves running the specified graph once per available input file. During that execution, all variables, either input or

output, are with respect to the current run. What is missing, however, is the capability of performing calculations involving variables across subjects. Consider a simple example of wanting to compute the maximum lane deviations across all subjects' minimum lane deviation during the third minute of a scenario. For this, we need to compare the minimum deviation within each subject and find the maximum across the subjects. The solution to this is a block called an accumulator. An accumulator collects the values presented in its input for each input run and only produces its output when its input data for all subjects' data has been computed. Furthermore, its output is a vector, each element of which is the value at the accumulator input for the respective data set.

An illustration of using an accumulator is given in Figure 12. A time segmentation block, which is connected to a min/max block, which in turn is connected to the lane deviation input stream computes the minimum lane deviation of the current subject. Let us assume that in this example the input experiment consists of 10 data sets. During execution, the DRDD shown in Figure 12 would execute 10 times, and the value on terminal between the min/max block and the accumulator would contain the min lane deviation corresponding to the respective data set. However, the output of the accumulator will not produce a token until after all 10 data sets have been processed, in which case the accumulator output will contain a vector of 10 numbers, each representing the minimum lane deviation of each input set. At that point, the min/max block located at the output of the accumulator will execute and will find the maximum value of its input vector, which in turn will be sent to the file output block for writing to a data file.

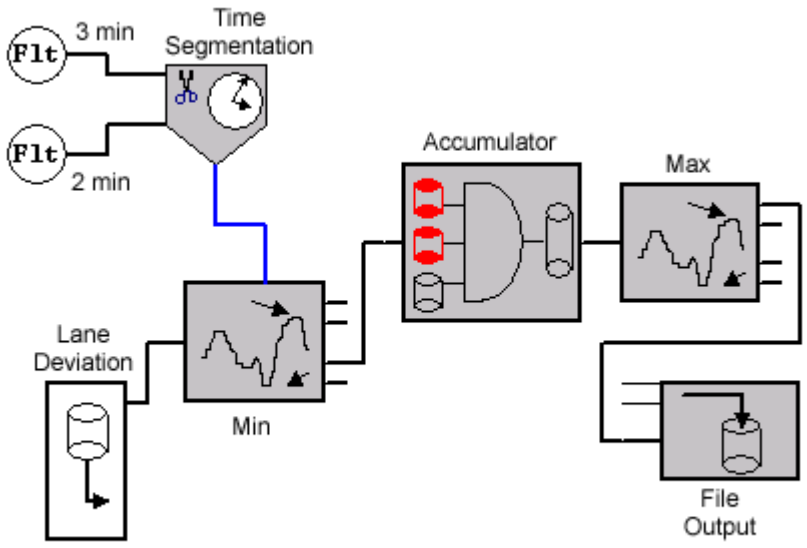

FIGURE 12 Example of using an accumulator.

CURRENT STATUS

Significant effort has gone into the design of the toolset over the last two to three years. More recently, a focused development effort has been taking place with the goal of producing a minimal skeleton of software that includes the execution engine and a few blocks that can be used to support actual data reduction activities for NADS-based research. Our goal is to enhance the software toolset by adding new functionality based on the requirements of NADS studies.

At the time of this writing, the DRAV GUI has been developed to the point where it allows the user to edit DRDDs. These diagrams can be saved and reloaded as necessary. The DRDD execution engine has been completed and tested extensively with a core set of nodes that includes approximately 50 percent of the nodes described in this paper. The remaining types of nodes are currently under development. Detailed design of a networked version of the ED is currently underway. In order to support diagram execution and, specifically, the ability to build the Input Terminals node, we have developed a version of the ED that provides all input/output and data set change detection capabilities but does not provide user access limitations, version control, or access audit trail. Finally, an input converter that understands the native NADS data acquisition system format has been completed.

CONCLUSION

This paper described the problems typically associated with the post-processing of large amounts of record-based data, with a focus on driving simulator applications. Numerous requirements were specified, along with a specific toolset architecture that addresses such requirements. Status on the development of the toolset was provided.

REFERENCES

1. *Matlab User's Guide*, The Mathworks Inc., 2001.

2. *Labview User's Manual*, National Instruments Inc., 2001.

3. *SAS Statistical Analysis Software*, The SAS institute Inc., 2001.

4. *SYSTAT Version 10*, SPSS Science, 2001.

5. Weiler, John M., John R. Bloomfield, George G. Woodworth, Angela R. Grant, Teresa A. Layton, Timothy L. Brown, David R. McKenzie, Thomas W. Baker, and Ginger S. Watson. Effects of Fexofenadine, Diphenhydramine and Alcohol on Driving Performance: A Randomized, Placebo-Controlled Trial in the Iowa Driving Simulator. *Annals of Internal Medicine*, Vol. 132, No. 5, 2000, pp. 354-363.

6. Schneier, B. *Applied Cryptography* (2nd edition). Wiley, New York, 1996.