Reform:
A Domain Specific Language

Dustin Graves

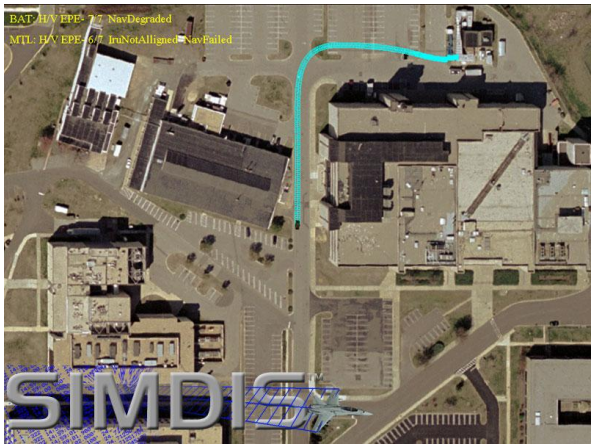
October 5, 2007

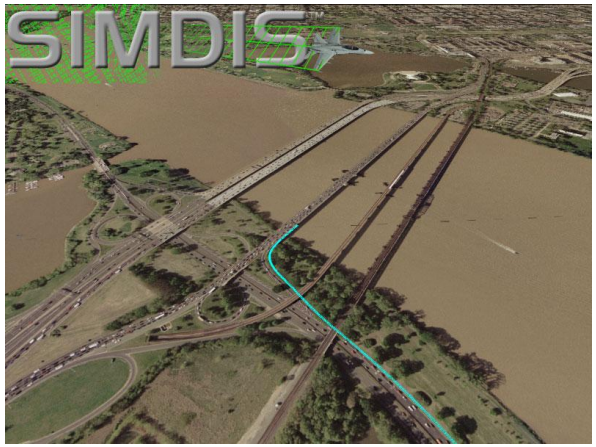
Overview

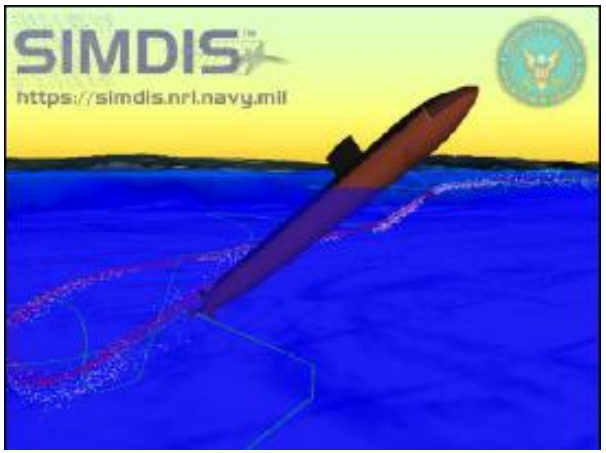
- Scripting language
- Monitors and manages data streams
  - Network, File, RS-232, etc
- Reformats and redirects data
- Contains keywords for data source and format definition, creation, and manipulation
- Operates within a distributed environment
  - Remote communication and manipulation
- Authenticates and controls remote access to resources

Background

- Designed for use with large scale situational awareness and vehicle tracking systems
  - Receive and process Time-Space-Position Information (TSPI) data for consumption by visualization and analysis applications
  - Military/Coast Guard applications, FAA flight tracking systems, commercial vehicle tracking systems







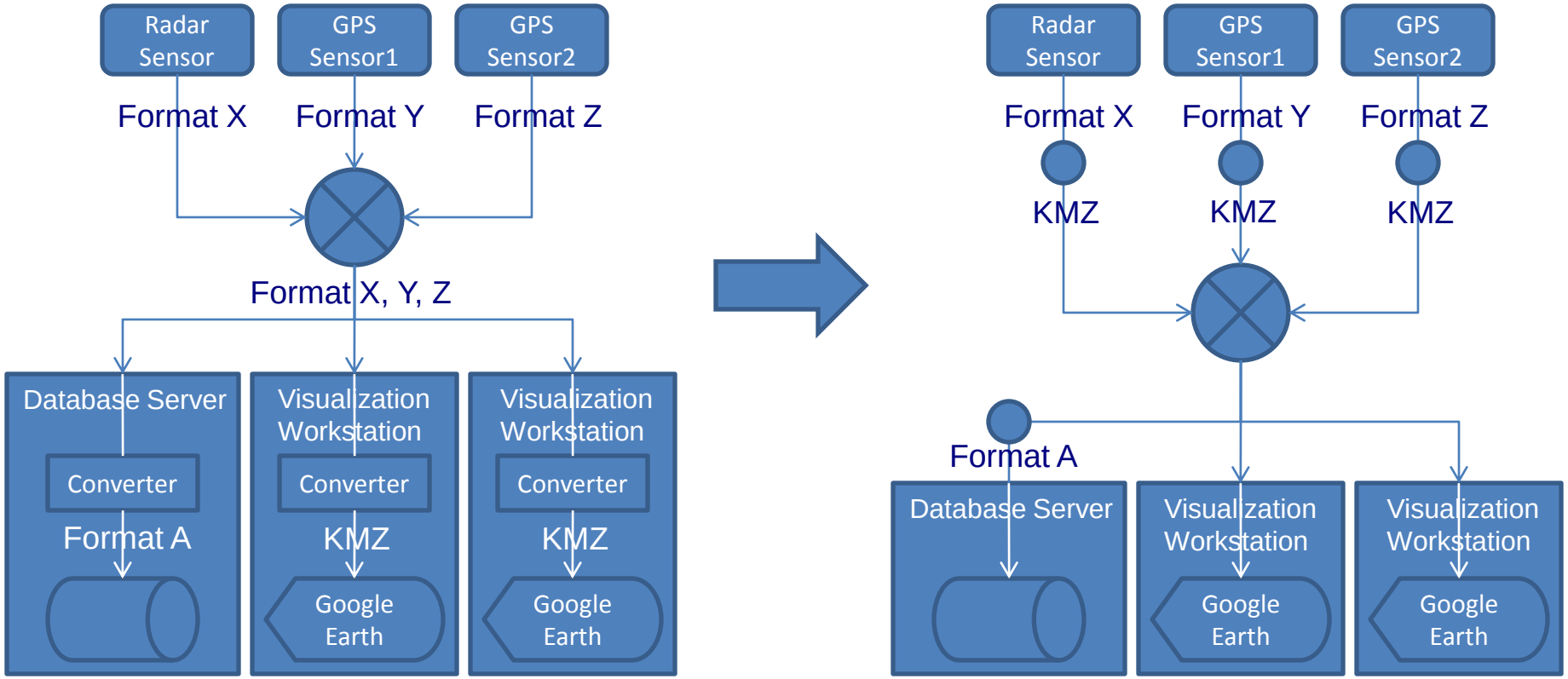
Background (cont.)

- General design allowing application to many other domains
  - Distributed data processing
  - Streaming compression and encryption
  - Importing/exporting non-native file formats
  - Flight simulators and on-line gaming

Goals

- Improve efficiency and scalability for large scale systems
- Reduce the amount of data conversion work required by the “consumer”
- Simplify process for adding support for new data formats
- Remote management of system components from a central location

Goals (cont.)


Radar Sensor
GPS Sensor1
GPS Sensor2
Format X
Format Y
Format Z
Format X, Y, Z
Database Server
Converter
Format A
Visualization Workstation
Converter
KMZ
Google Earth
Visualization Workstation
Converter
KMZ
Google Earth
Radar Sensor
GPS Sensor1
GPS Sensor2
Format X
Format Y
Format Z
KMZ
KMZ
KMZ
Format A
Database Server
Visualization Workstation
Google Earth
Visualization Workstation
Google Earth

Interpreter

- Each active interpreter instance acts as a component of a larger distributed network of interpreters
  - An active interpreter instance is referred to as a Reformer
- Contains three key elements
  - Formats – A collection of data formats recognized by the Reformer
  - Sources – A collection of data sources for receipt and transmission of data
  - Channels – A collections of channels for remote Reformer manipulation
- Authenticates remote procedure calls
  - Remote manipulation can be restricted to limit access to data
  - Commands may only be executed by authorized Reformers

Key Elements: Formats

- Data formats provide a concrete structure to raw stream data
  - Allow high-level manipulation of data
  - Used to define mappings between records of different types
  - Can be predefined or defined at runtime
  - One or more Data Formats are registered with a data source
    - Specify how data source should decode and encode raw data
    - Characteristics of the raw data are used to determine the specific format type when decoding
  - Individual format types are mapped to other format types
    - Convert data for use with other streams
  - The interpreter provides an array, *source*, containing all active sources and a print statement, *sources*, which prints information about each active source

Format Definition an Mapping

- Format definition syntax:

define format *format-name* as
(*type-name field-name ;)+*
end (*format-name*)? ;

- Format mapping syntax:

map *format-name1* to *format-name2*
begin
((*format-name2* :)? *field-name* =
(*format-name1 : field-name* | *expr* ;)+
end (map)? ;

Format Definition Sample

- Format definitions describing a radar measurement and the position of a vehicle with a known ID value:

```
define RAE as
        uint32 sec;
        uint32 msec;
        double range;
        double azimuth;
        double elevation;
         uint32 status_bits;
end;
```

```
define format Track as
        uint32 ID;
        double time;
        double latitude;
        double longitude;
        double altitude;
        uint8 active_status;
end Track;
```

Format Mapping Sample

- Format mapping for raw byte array from stream to RAE format:

```
map raw to RAE
begin
   RAE:time = raw[0:7];                    # Map bytes 0-7 to time field
   RAE:range = raw[8:15];
   RAE:azimuth = raw[16:23];
   RAE:range = raw[24:31];
   RAE:status_bits = raw[31:35];
end map;
```

Format Mapping Sample (cont.)

- Format mapping for RAE format to Track format:

```
map RAE to Track
begin
   # Assign a constant to ID field
   ID = 42;

   # Compute milliseconds since midnight from RAE:Time  (seconds since week start)
   time = (RAE:sec % 86400) + (RAE:msec * 0.001);

   # Employ external function to compute LLA from RAE relative to radar position
   latitude = computeLat(RAE:range, RAE:azimuth, RAE:elevation, radar_position);
   longitude = computeLat(RAE:range, RAE:azimuth, RAE:elevation, radar_position);
   altitude = computeLat(RAE:range, RAE:azimuth, RAE:elevation, radar_position);

   # Extract active status bit from
   active_status = RAE:status_bits & ACTIVEBIT;
end;
```

Key Elements: Sources

- Data Sources receive and transmit data through existing streams
  - Can be created, destroyed, and linked
    - Assigned IN, OUT, or INOUT type
  - Send and receive data to and from a stream
    - Data formats are registered with streams to describe the structure of stream data
    - Procedures for transforming, filtering, and monitoring data are applied after reading and before writing
      - Transforms modify data
      - Filters remove unwanted or corrupt data from the stream
      - Monitors trigger actions based on data values
  - Can have caches with adjustable size
  - The interpreter provides an array, *source*, containing all active sources and a print statement, *sources*, which prints information about each active source

Source Manipulation

- Source creation syntax:

  import *module-name* ;

  create [*module-name*.*type-name*, in|out|inout (, *expr*)*] ;

- Source destruction syntax:

  source: *id-name* | source[*expr*]

  destroy [(*source* (, *source*)*) | sources[INT:INT]] ;

- Source linking syntax:

  format-list: ((*format-name*,)* *format-name*|others)?

  source format-list -> *format-list sources* ;

- Format registration syntax:

  criteria: (size *equality-op expr* | raw[INT: INT] *equality-op expr*) (,
  raw[INT: INT] *equality-op expr*)*

  register *format-name* with *source* (when *criteria*)? ;

Source Manipulation Sample

- Creation, linking, format registration, and destruction of an in and an out source:

```
import net;

# Create a unicast socket to read from port 2000 and
# a multicast socket subscribed to group 224.0.0.1:2111
rae_source = create [net.unicast, in, 2000] ;
create [net.multicast, out, 2111, '224.0.2.111'];

# Register formats with sources
register RAE with rae_source when size == 36;
register Track with source[1];

# Link sources with RAE mapped to Track and other formats sent as raw data
rae_source (RAE, others) -> (Track) source[1];

# Destroy the sources
destroy [ source[0:1] ];
```

Source Manipulation Procedures

- Transform, Filter, and Trigger procedures can be attached to sources for higher-level format manipulation
  - Operations for managing procedures include inserting, appending, removing, and relocating procedures within a procedure lists
  - Procedures contained by the procedure list are executed sequentially for each data format object
  - Used to perform more complex data conversions that can not operate on individual format fields
    - A conversion from a Geodetic to Geocentric coordinate system will require the LLA and XYZ format components to be evaluated at the same time
      - Accomplished by simply copying LLA to XYZ with a format mapping and using a transform to complete the conversion process
  - Potential for optional data processing such as data compression
    - Append compression transforms to the output source and attach the inverse compression transforms to input source

Key Elements: Channels

- Channels communicate with and manipulate remote Reformers
  - Can be opened and closed
  - Transmit language commands for the remote Reformer to process
  - The interpreter provides an array, *channel*, containing all active channels and a print statement, *channels*, which prints information about each active channel
- A Listener is required to accept connections from remote Reformers
  - Listeners can be activated and deactivated
  - A trigger can be set to execute a specified command when a new channel connection is received
  - The interpreter provides an array, *listener*, containing all active listeners and a print statement, *listeners*, which prints information about each active listener

Channel Manipulation

- Channel creation syntax:

  import *module-name* ;

  open [*module-name.type-name* (, *expr*)*] ;

- Channel destruction syntax:

  channel: *id-name* | channel[*expr*]

  close [(*channel* (, *channel*)*) | channel[INT:INT]] ;

- Listener creation syntax:

  listen [*module-name.type-name* (, *expr*)*] ;

- Listener destruction syntax:

  listener: *id-name* | listener[*expr*]

  deafen [(*listener* (, *listener*)*) | listener[INT:INT]] ;

Channel Manipulation Sample

- Creation and destruction of a channel and a listener:

```
import net;

# Open a TCP channel to remote Reformer listening to port 3000 on the localhost
remote = open [net.socket, 3000, '127.0.0.1'] ;

# Print the remote Reformers sources and destroy the first source
remote.sources;
remote.destroy [ source[0] ];

# close the channel
close [remote];

# Listen for channel connections on port 3001
listen [net.socket_server, 3001];

# Destroy the sources
deafen [ listener[0] ];
```

Authentication

- Authentication Modules control access from remote Reformers
  - Authentication certificates can be attached to Reformers, functions, and data fields
    - High level access control at the Reformer level
      - Remote Reformers provide credentials when establishing communication channels
    - Low level access control at the function and field level
      - Remote Reformers provide credentials with remote requests
  - Custom Authentication modules can be supplied to authenticate requests through different methods

System Design

- The Reform interpreter consists of three major sub-systems
  - The Data Manager receives data from sources; applies format conversion, transformation, and filtering; and transmits data through linked sources
  - The Source Monitor monitors sources of IN and INOUT types for incoming data
    - Receipt of data is performed asynchronously
    - Sources, Channels, and Listeners ready to receive data are placed in a queue for the data manager to process
  - The Parser processes language commands
    - Sources with IN and INOUT types are registered with and removed from the Source Monitor when created and destroyed
    - Listeners and channels are registered with and removed from the Source Monitor when opened and Closed

Parser

- The Parser consists of a Reform language parser and an AST parser
  - Reform programs consist of a series of definitions and statements that are converted to Abstract Syntax Trees by the language parser
  - The AST parser process the AST generated by the language parser and executes the appropriate commands
    - AST nodes representing variable initialization and function definitions are stored in a symbol table for future access
    - AST nodes containing remote commands are transmitted to the remote Reformer for execution
    - An iterative statement processor is recursively invoked when processing scoped functions, control structures, and explicitly defined blocks
    - Statements are iteratively processed until a statement requiring a new scope is encountered, when the current state is pushed onto a stack and the new scope is entered

Source Monitor

- The Source Monitor waits on “handles” to data sources
  - When a source is ready to receive data, it is placed in a queue for processing by the Data Manager
    - The UNIX select and Windows WaitForMultipleObjects system calls can be used to monitor file handles and object handles
    - Interrupts could be used for embedded systems
  - The Source Monitor may be threaded, to take advantage of multi-core systems, such that one or more threads can be assigned to monitor one or more sources

Data Manager

- The Data Manager performs data manipulation and transmission
  - The Data Manager monitors the queue containing readable sources with pending data
  - The sources are removed from the queue and processed
    - Pending data is read from the source and any associated transforms, filters, and triggers are applied to the received data
  - Processed data is transmitted to any existing linked output sources
    - Because each source may have a set of transformations which alter the data, a copy of the data must be provided to each source
  - The Data Manager may be threaded, to take advantage of multi-core systems, such that one or more threads can be assigned to process one or more sources

Scalability

- The system design has been chosen to allow implementation for single-processor hardware and multi-processor hardware
  - When dealing with systems limited to a single processor, processing time may be serially divided between the three sub-systems:
    - Execute the parser to process one or more blocks of statements
    - Execute the source manager to test for pending data from one or more sources; add any pending sources to the queue
    - Execute the data manager to process one or more pending sources contained by the queue
    - Repeat until stopped
  - When dealing with multi-processor and/or multi-core systems, thread pools can be employed to take full advantage of all available processing resources
    - Each source is treated as an independent task
    - Tasks are submitted to the thread pool
    - One or more available threads will be assigned one or more pending tasks
  - The official position regarding order of execution is that all data transformations are guaranteed to be processed, but there is no guaranteed order of processing

Optimization of Remote Processing

- Synchronous processing of remote statements can lead to long periods of time where the Reformer is waiting to receive the result of the remote statement execution
- Remote statement processing can be optimized by rearranging the AST such that statements independent of a remote statement are executing while waiting for the result of an asynchronously executed remote statement
  - Reorganize statements as:
    - Execute statements upon which the remote call depends
    - Execute the remote call asynchronously
    - Execute statements that are independent of the remote call
    - Wait for receipt of the remote call result
    - Execute the statements that depend on the remote call

Implementation

- The Reform language will be implemented with a mix of C and C++ code
  - This is a deviation from the original intent to develop a prototype with C#
- ANTLR v3 will be used to generate a C parser for the Reform language
  - Many improvements from ANTLR v2.7
    - Better language constructs
    - Nice syntax for AST specification
    - ANTLRWorks GUI development environment with syntax highlighting and grammar debugger
- Intel's Threading Building Blocks library will be leveraged to obtain optimal multi-core performance
  - C++ template based
  - Provide thread safe data structures and parallel control structures
  - Available for Windows, Linux, and Max OS X
  - A Solaris version is currently under development