6th AIAA Aviation Technology, Integration and Operations Conference (ATIO)
25 - 27 September 2006, Wichita, Kansas

AIAA 2006-7830

A Data-Centric Air Traffic Management Decision Support Tool Model

James R. Murphy* and Ron Reisman†
NASA Ames Research Center, Moffett Field, CA 94035-1000

and

Rob Savoye‡
Seneca Software, Rollinsville, CO 80474

This paper describes the differences between a data-centered and an algorithmic design model as it applies to air traffic management software. An example of converting an existing algorithmic software baseline to a data-centric model is discussed. Using the data-centric model, a proof of concept flight deviation tool was developed. The deviation tool received inputs from a real-time data manager and a third party database to supply a list of aircraft identified as off their assigned flight path. The results were displayed using a standard web browser. From the development of this prototype it was determined that the use of a database should be coupled with a data manager to ensure a timely interaction with the data.

I. Introduction

THE Center – Terminal Radar Approach Control (TRACON) Automation System (CTAS) is a suite of programs designed to improve the efficiency of the National Airspace System (NAS) while maintaining or exceeding the current level of safe air traffic operations.[1-4] CTAS based tools, such as the Traffic Management Advisor, are among a group of tools included in the Federal Aviation Administration's (FAA) plan for modernization.[5] It is recognized that modernization of the air traffic infrastructure will come from integration of existing and future support tools and data.[6-7] One aspect of the next generation air traffic control system is the collection and dissemination of aircraft flight information from these tools via the Flight Object.[8] It has been seen through the development of CTAS tools such as the Collaborative Arrival Program that connecting to and transferring data out of CTAS can be difficult.[9] CTAS tools have been developed over many years starting with a foundation of a 4-dimensional (4-D) trajectory modeler and adding functionalities such as scheduling, speed advisories, and conflict detection.[2-3] Whether the reason was development time constraints or the need to get the best performance possible from the existing hardware, new CTAS functionalities were typically inserted into the software at the place where the data were accessible, or in some cases, through the addition of new modules with a customized interface to existing data. Each addition caused the underlying software to have a greater interdependence.[2,10] It is this interdependence that complicates sharing of CTAS derived data with outside systems and integration of new functionalities into the existing baseline.

Software design and development is a balancing act among ease of maintenance, scalability, and efficiency. In the near real-time necessities of air traffic control software, efficiency has always been a major concern. However, due to continued advancements in hardware, designing software that is more easily managed can now be the focus. In the data-centric software model, the algorithm is not the primary driver for the design of the software. Instead the data inputs and derived outputs from the algorithmic software modules are stored in a central database for use by any system that can connect to it. This provides several benefits. First, new functions can be added to the system without impacting any of the existing software algorithms. The only thing that is needed is knowledge of the data and a connection to the database. At the same time, the database provides an interface between derived data, such as

* Software Engineer, AFD, MS 210-8, AIAA Member.
† Research Engineer, AFD, MS 210-8, AIAA Member.
‡ Software Engineer.

schedules, and the outside world. In addition, since all of the data are stored in the same place, checks can be added to ensure that any set of derived data is consistent with respect to each other.

This paper provides a description of a data-centric software model for use in the air traffic management (ATM) domain and discusses the transition of the CTAS software from an algorithmic design model to a data-centric model. A proof of concept tool development where CTAS algorithms were used to provide derived data to an ATM database is described. Lessons learned from the database development will also be discussed, including data display techniques that proved to be problematic.

II. Motivation

The replacement to the Host Computer System, which is an air traffic controller's main source of aircraft track and flight information, is currently under development by the FAA.[11] In parallel, ATM decision support tools are under research and development to augment the Host replacement's capabilities by providing controllers with tools to increase efficiency of air travel while maintaining or increasing safety.[7,12] The FAA's System-Wide Information Management project is an attempt to bring value added functionality to the decision support tools by sharing information.[13] As new decision support tools are developed, several questions are commonly raised:

1.) How does an outside program access data from CTAS?
2.) Where are the data needed for a particular CTAS algorithm stored?
3.) Does all of CTAS need to be run to get the data?

Though thoroughly familiar with the CTAS software and algorithmic baseline, these are the same questions asked by the authors when presented with the following task: Show aircraft that are deviating from their flight plans on a simple display.

Much of the necessary code to complete this task already existed, embedded in various CTAS modules. However, while developing the proof-of-concept prototype, it was realized that using the existing CTAS baseline also brought in software algorithms that were not necessary for this task, such as generating estimated times at the meter fix. In this case, the capability of CTAS to build functionality from existing software became a liability for a relatively simple, well-defined problem. So instead of using the existing CTAS software baseline, only the algorithms needed to develop a module whose purpose was to determine whether an aircraft was deviating from its flight path were taken from CTAS. This led to reevaluating the design of all CTAS tools to orient them around data storage instead of data generation. This is a basic concept of object-orient programming, which has demonstrated extended re-use of software and algorithms.[14]

III. The Current CTAS Design

To illustrate the benefits of using a data-centric software design, let us first examine the current CTAS software design model. For clarity, the basic flow of information through a CTAS system designed to schedule aircraft at the meter fix of an adapted airport will be described. For a more in-depth discussion of CTAS processes, refer to Erzberger, et al.[2]

One way to think of the CTAS design is as a hub and spoke algorithmic data server. Each of the core algorithms necessary for CTAS output are contained in separate processes that communicate to each other through the hub. Refer to Figure 1 for a schematic representation of the software hub and the spoke processes. It should be noted that the CTAS design allows each of the spoke processes to be run on a separate machine that connects to the hub though a network socket. This allows computation intensive processes to be run on separate machines, which enables users to scale the software to the level needed by a particular application.

As the hub, the Communications Manager primarily acts as a router of data that are calculated by one process, but needed by other processes. Aircraft track and flight plan information from the Host are received via radar parsing processes and passed to each of the spoke

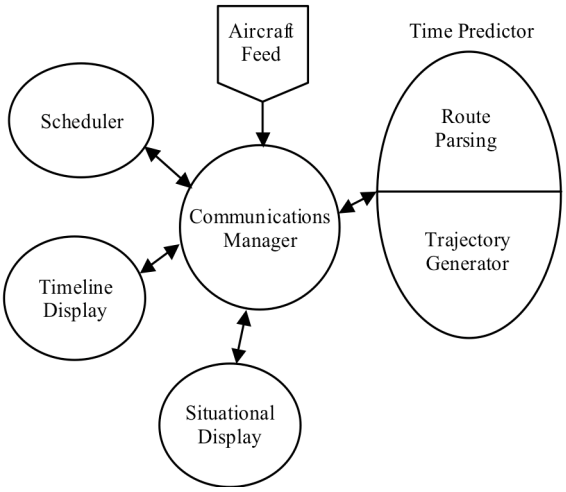

Figure 1. Current CTAS design. *Schematic of the CTAS processes used to provide scheduled times to the meter fix.*

processes, as needed. The Communications Manager also stores various state information, such as active runways and aircraft flow restrictions; so that when a spoke process connects, it will have a snapshot of the current state of the system.

One of the spoke processes that receive data from the Communications Manager is the Time Predictor. This process is divided into two primary sub-tasks. The Route Generator first parses the flight plan data into a list of fixes that define the aircraft's horizontal path, then determines the vertical restrictions the aircraft must maintain based on intent information from the flight plan and the adaptation of air traffic control procedures. The Trajectory Generator utilizes the horizontal and vertical profiles, along with atmospheric and wind data, to calculate a 4-D trajectory. Finally, estimated times to various points along the projected path, including the assigned meter fix and expected runway, are determined through integrating the trajectory segments.[2]

These estimated times are sent to the Scheduler via the Communications Manager.[15] After receiving the estimated times, the Scheduler divides aircraft that are assigned to the same meter fix and have similar altitude restrictions into groups. Aircraft in the same group are then assigned times at the meter fix to assure temporal and spatial separation. These scheduled times provide controllers with an efficient flow of aircraft into the adapted airspace.[15]

The visualization processes receive information from the Time Predictor and Scheduler processes and display the data to air traffic personnel. Timelines, depicting each aircraft's estimated and scheduled times to the meter fix or runway, are displayed in the Traffic Management Unit. CTAS also provides an air traffic situational display that gives an external representation of the current track data for each aircraft and displays various types of air traffic control maps.

As new algorithms are developed, the CTAS tool-set can be expanded to provide further value-added outputs to controllers. Figure 2 shows the expansion of the base scheduling functionality in CTAS to provide trajectories for aircraft to achieve the scheduled times at the meter fix. The Schedule Analyzer, shown outside the core scheduling area, also connects to the CTAS system via the Communications Manager. It receives track, flight plan, and scheduling data for each aircraft and then iteratively generates trajectories for the aircraft that are checked for both successfully meeting the schedule time at the meter fix and conflicts among all other arrival aircraft trajectories.[3]

At the start of the iteration process, the Schedule Analyzer generates a set of trajectories that provide a range of times that the aircraft can be expected to reach the meter fix. This starting set of trajectories is identical to the set calculated by the Time Predictor used to generate the original estimated times at the meter fix. It would be possible to have the Time Predictor module send its trajectories to the Schedule Analyzer, however at the time the schedule manager was developed, the network bandwidth needed to send the original trajectories from the predictor module imposed problems. Re-generating the trajectories in the Schedule Analyzer, though on the surface seeming like a duplication of effort, only negligibly impacts the rest of the CTAS system because the Schedule Analyzer is able to run on a separate machine.

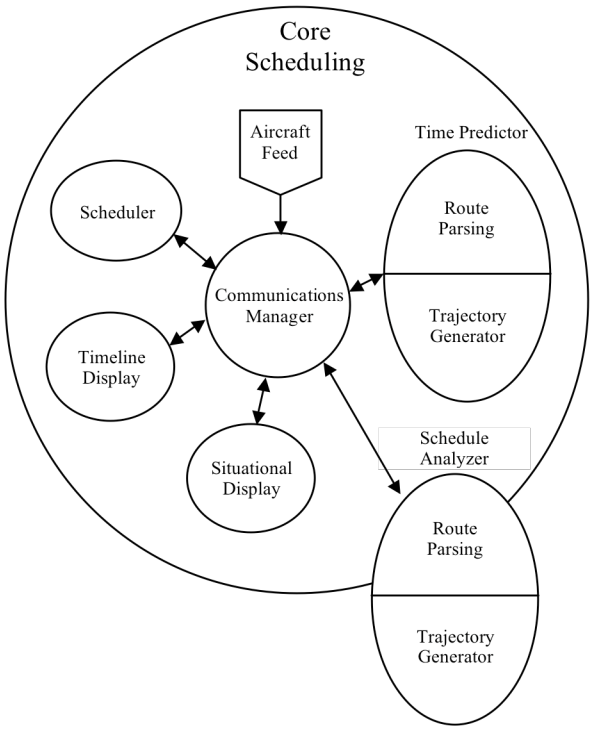

Figure 2. CTAS Expansion. *Diagram of the CTAS processes used to extend the core scheduling functionality to also provide a trajectory to meet the schedule times.*

IV. Proposed Data-Centric Design

Hardware and software techniques have advanced to a state where some of the limitations that drove the current CTAS design are no longer as much of a concern. In fact, the original CTAS scheduling functionality that was running operationally using several machines can now be run on a single machine. However, as more spoke processes are moved to the same machine, duplication of any data calculation can directly impact the performance of the system as a whole. With that in mind, it should be noted that even though using a data-centric approach to software development can remove the duplication of some of the calculations such as trajectory generation, it does not

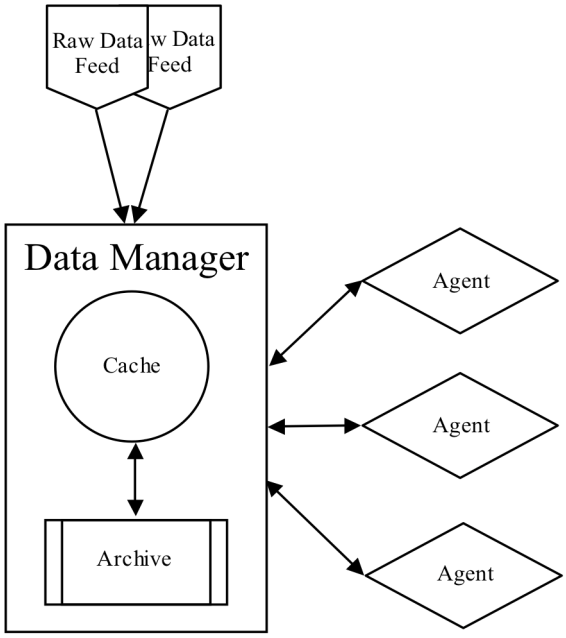

Figure 3. Data-centric Design. *Schematic of the basic data-centric software model.*

guarantee more efficient software. In fact, it is possible that a tightly coupled interdependent system such as CTAS can perform more efficiently. However, with the advancements in memory, computer performance, and network bandwidth, a small loss of software efficiency is a fair trade for ease of programming, extensibility, and maintainability of the software.

A. Data-centric Model

Figure 3 illustrates the basic design of the data flow in a data-centric tool-set model. A data manager at the center stores all of the information generated by each of the raw data feeds and sub-processes. These sub-processes are referred to as agents, which can be thought of as building blocks for any given program. Agents are the smallest subdivision of source code necessary to derive output data. Their complexity can range from a simple function to a full program, depending on the requirements. Furthermore, agents are only concerned with the data they need to generate a particular answer. That answer is sent back to the data manager without regard to who needs it, thus reducing the complexity of the agent's software.

The Data Manager in the model is really comprised of two components, a data cache and an archive. The data storage is separated this way to account for the two main functions of the manager, access to all of the necessary pertinent data used by the agents, and archiving of all data for post processing or expanded meta-analysis. The cache is essentially real-time memory, containing data that are used often by the agents, or needed by agents as quickly as possible. The archive can be considered a standard database, with the data stored on a disk, accessed through query commands. These databases have powerful tools to query and access the information, but since it may not reside in memory, the data can be slow to retrieve, especially with complex queries.

The Data Manager provides basic publish and subscribe functionality to all attached agents. When an agent connects to the Data Manager, it indicates what kind of information it needs to access. As the Data Manager receives that type of data from raw inputs or other agents, it is forwarded to the requesting agent. This allows for agents to build on the output of other agents, essentially creating a program from a group of agents.

In its role of managing the agents, the Data Manager is crucial for data consistency. It is possible that the data required by an agent will come from multiple sources. It is the Data Manager that has the responsibility to ensure that data from each source are consistent with each other before sending it on to the requesting agent.

There are two aspects of data consistency that make its assurance non-trivial. First, the latest data are not always the data that are required. For example if an agent requires track and trajectory information to perform conflict prediction, the track data that are provided to the agent must correspond to the trajectory data that are sent over. Hence, it may not be sufficient for the Data Manager to send off the track data as they come in from the raw data source. Instead it must wait until a complete set of data, i.e., the track and corresponding trajectory, are obtained before sending it out. Coupled with that issue is the need for the Data Manager to keep track of dependency issues among all data elements. The Data Manager must know that the trajectory is dependent on track, flight plan, wind, etc. If any of these are updated or changed, the Data Manager must be able to either invalidate the existing trajectory so that it is not used with inconsistent data or store the data that were used to generate it. How the Data Manager handles these issues is dependent on the requirements of the system, but in the end it must keep a consistent snapshot of required data for each agent.

The Data Manager also serves as the main connection to the archive. This allows for programs to connect to the data cache, but still access archived information easily. In fact, an agent needs only make a request for data from the Data Manager. If those data have been archived, the manager can query the archive automatically. This is hidden from agents of the system. As the data age, a snapshot of the archive can be taken and the database purged. These snapshots can be useful for post-run analyses, because they can be kept for an indefinite period of time and are a complete record of the information that was in the Data Manager. It should be noted that using a data manager does not preclude an agent, or any program, from accessing the archive directly. The archive is essentially a standard

database and can be accessed through direct queries from any program. However, when controlled by a data manager, access can be handled so that the archive is not overwhelmed with queries that have the potential to slow it down dramatically.

The real power of this model is the flexibility it affords to the developer for the creation of the agents. Since the agents are stand-alone modules that only interface with the Data Manager, they can be written in any programming language that can connect to it. The agents can also be as complex as dictated by the requirements of the system. So whether due to reasons of data consistency or software efficiency, an agent can be written with multiple complex functionalities. In addition, a documented mechanism and infrastructure now exists to then break apart the complex agent when development resources become available. This promotes software designed by asking what data are needed, instead of how can the necessary data be generated or retrieved.

Using a data-centric model during program development provides the ability to use a simplistic agent as a stub, while a more detailed agent is developed. As long as the interface to the Data Manager stays the same, the detailed agent can be seamlessly swapped into the system upon completion. This allows for easier concurrent development among separate programming teams.

B. Data-centric CTAS example

Consider again the case of the basic CTAS scheduling functionality described above but this time through the data-centric model. Figure 4 illustrates how the CTAS algorithms can be incorporated into this model. The aircraft track and flight plan data still come into the system, but they are now stored into the Data Manager's cache and archive. The Data Manager determines which attached agents require the track and flight plan data and forwards this information appropriately. The example could stop there; the Data Manager could be a simple replacement for the original CTAS Communications Manager and each of the agents, though complex, could be the original Time Predictor, Scheduler, and displays. This is where the most significant changes can be made.

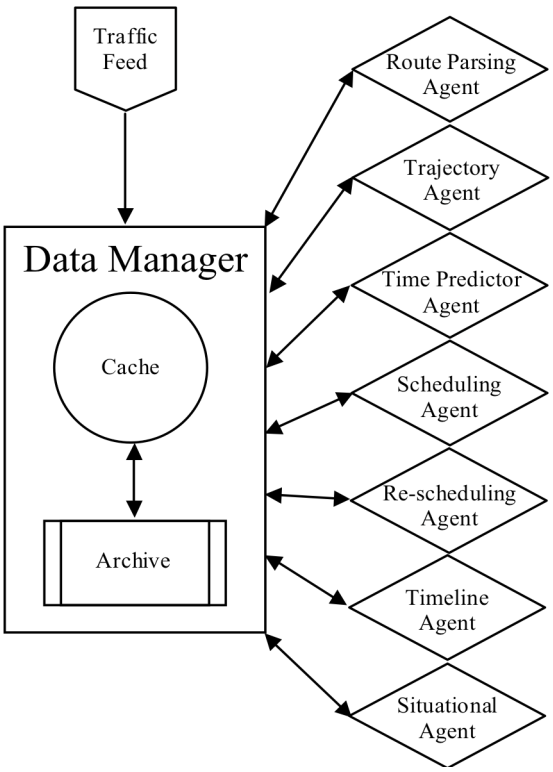

Figure 4. Data-centric CTAS design. *Schematic of the CTAS system using a data centered design model.*

Consider a functionality that takes the flight plan route and returns the list of fixes that define the horizontal path. This functionality could be taken out of the Time Predictor module of the current CTAS implementation and moved into a route-parsing agent. Likewise the trajectory generator functionality can become a separate agent. The Time Predictor now becomes an agent that subscribes to the Data Manager for trajectories. As the Data Manager receives new trajectories, they are forwarded to the time predictor, which then calculates the time at the meter fix. The new scheduling agent connects to the Data Manager and periodically sends a request to receive the latest snapshot of estimated times at the meter fix for each aircraft to generate the schedule. The timeline and situational display agents at this point remain for the most part unchanged. They connect to the Data Manager and subscribe to the necessary track, flight plan, time estimates, and schedule data.

Adding a new functionality like generating a trajectory to meet a scheduled time has now been simplified. A new re-scheduling agent can take the output from the route parsing, time predictor, and scheduler agents and generate advisories for the aircraft to meet the scheduled time. This trial meet-time trajectory can either be generated by the trajectory agent or generated internally by the re-scheduling agent, depending on the requirements of the system. In addition, regardless of the decision made at the time the agent is developed, with the data-centric design model infrastructure in place, it can be changed in the future.

One of the major benefits of the data-centric model is that it allows for easier access to CTAS-specific data by non-CTAS programs. Agents generating output data such as estimated and scheduled times do not need to know how those data are used; only the manager has the data requirement knowledge of the agents. This removes implicit dependencies among the agents. For example, estimated and scheduled times at the meter fix, trajectories, and even conflict predictions are now accessible by any program that can readily connect to the Data Manager (or query the

data archive directly). This promotes the ability for CTAS systems to inter-operate with existing and future ATM decision support tools.

V. Proof of Concept

The purpose of the proof of concept was to create a national display of the CTAS flight path deviation algorithm for aviation security. This algorithm determines when an aircraft is off its assigned flight plan and flags this condition on the CTAS situational display. It also has a prediction component that checks trajectories for intersections with special use airspaces such as military operation areas.

The current flight path deviation algorithm is embedded within the conflict prediction functionality of CTAS. To create a national display using the current CTAS software baseline, twenty instances of CTAS would need to be run to cover the NAS airspace, one for each en-route Center. Even though a functional prototype of the twenty-Center CTAS system has been demonstrated, since each instance of CTAS must run on at least one computer, the hardware needs alone became prohibitive.[16] In addition, it was still necessary to develop a national display that could show deviating aircraft from each separate CTAS system, as well as a mechanism to get the deviation information out of CTAS.

With this in mind, it was decided to develop a new program based on the deviation functional components from CTAS. The intention was to build a system that would utilize the Enhanced Traffic Management System (ETMS) Aircraft Situational Display for Industry (ASDI) national data feed and output the desired deviation predictions while running on a single machine. The data would be accessed through a database, with the display provided via web browser. By keeping the system simple, software development requirements would be manageable in the time frame allotted. Through designing this software and building the prototype, the model for data-centric ATM software was conceived.

A. Proof of Concept: Design

Figure 5 illustrates the basic process connectivity of the Deviation Web Utility. An open source database (MySQL), which is readily available for our primary development platforms, is used for data archiving. The database daemon serves as the data cache for the test-bed. Upon startup, MySQL loads the national waypoint and sector boundary adaptation data. When the daemon connects to MySQL, it downloads the necessary adaptation from it. It also connects to the live ASDI data feed, parses and formats the aircraft and track messages, and stores the messages in MySQL.

The flight path deviation code was taken out of the CTAS conflict prediction module, along with any necessary support functions from the CTAS libraries. These functions were used to create the deviation agent. Only the portion of the flight path deviation code that checks for flight path deviations was used.

The deviation agent connects to the database daemon and waits for track and flight plan information. The deviation agent was designed to be memory-less, so that each time a track message is received by the database daemon, it queries MySQL for the flight plan data for that aircraft. It then sends a request to the deviation agent with all of the data necessary for it to determine a deviation. The deviation agent simply returns whether or not a deviation was flagged for that aircraft (and the reason if positive).

The display of the deviation list is handled by direct access to MySQL through web based query scripts. These scripts are used due to their relative ease of accessing MySQL and capability for drawing simple interactive displays on a standard web browser. By selecting the appropriate internal web page in the browser, a script performs a query into the database and

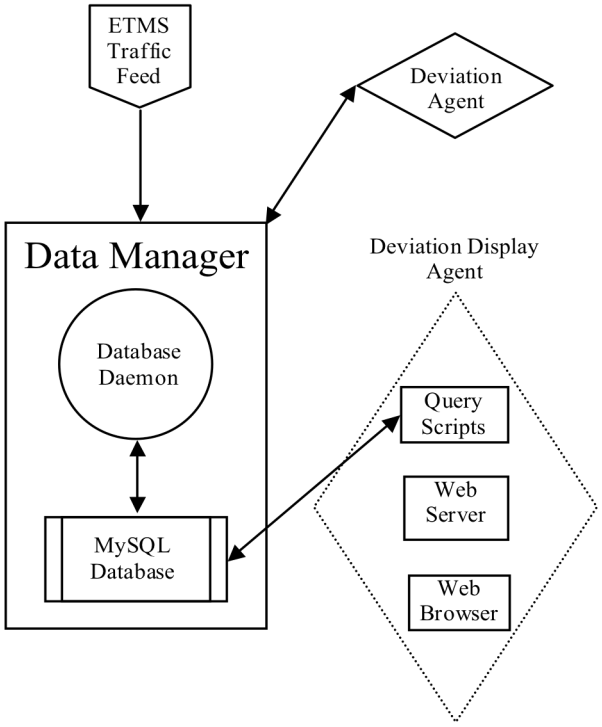

Figure 5. Deviation Web Utility. *Schematic of the proof of concept flight deviation prototype.*

produces a list of deviating aircraft. Figure 6 shows an example of an interactive deviation list. Each deviation contains the aircraft call sign, the reason the aircraft as flagged as a deviation, the controlling Center, and the timestamp of the track message that produced the listing.

The MySQL query is automatically repeated every twelve seconds to keep the deviation list and web page up to date. Notice that the call sign from Figure 6 is selectable on the web browser. By selecting an aircraft from the list, the web browser will bring up a web page that displays two pictures. As seen in Figure 7, the picture on the left shows the complete flight plan path for the aircraft, including all previous tracks, as well as the air traffic control sectors the aircraft has flown through. The picture on the right shows a zoomed-in view, with the flight plan and the last twenty track hits. Note that the last track segment was colored red to help indicate the position of the aircraft in deviation.

Deviating Flights

Aircraft ID	Deviation Flag	Center	Timestamp
SKW6409/344/KZOA	Route Deviation	KZOA	10:02:32
JBU469/622/KZMA	Route Deviation	KZMA	10:02:33
N699DA/251/KZMA	Route Deviation	KZMA	10:02:33
SWA449/988/KZLA	Route Deviation	KZLA	10:02:33
N947QS/673/KZBW	Route Deviation	KZBW	10:02:34
NWA827/653/KZME	Route Deviation	KZME	10:02:35
N7601/273/KZAU	Route Deviation	KZAU	10:02:36
SYX2296/70/KZAU	Route Deviation	KZAU	10:02:36
DAL645/340/KZOB	Route Deviation	KZOB	10:02:38
FLG5995/97/KZOB	Route Deviation	KZOB	10:02:38

Figure 6. Deviation list. *Actual snapshot of the web page that displayed a list of deviating aircraft.*

The rudimentary aspect of the proof of concept displays seen in Figures 6 and 7 come from the selection of display hardware. For the demonstration, the web browser was run on a wireless hand-held Personal Data Assistant (PDA), which provided a 3.5" display screen.

B. Proof of Concept: Lessons Learned

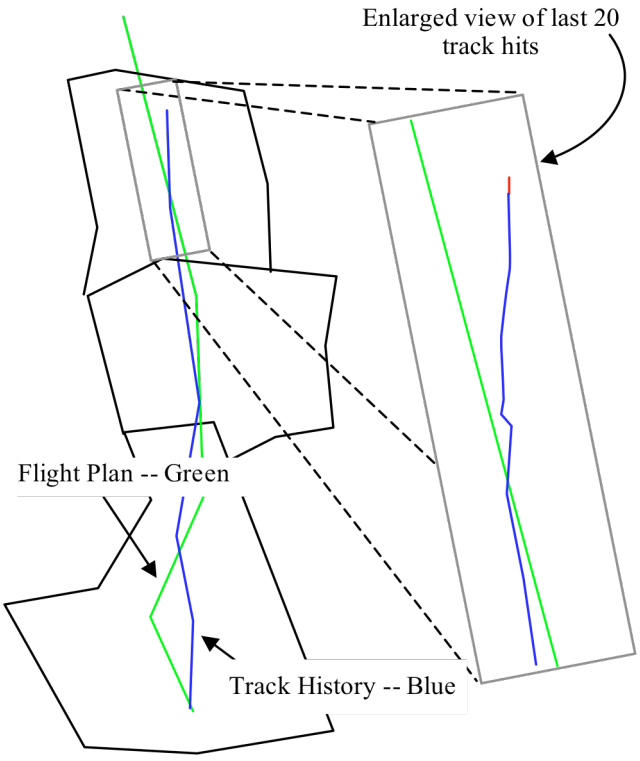

Figure 7. Deviation Display. *Display of the flight plan and track history of a deviation flight. The drawing on the left represents the full flight plan and track history. The drawing on the right shows the same flight zoomed into the last 20 track hits.*

The proof of concept design was not a result of, but a precursor to, the data-centric model described in this paper. From the development of the Deviation Web Utility and the lessons learned from the prototype, the concept of the data-centric model was formed. The discussion of the lessons learned from the proof of concept should be prefaced by noting that the development time frame for the prototype was approximately three weeks and the researchers, though proficient in ATM decision support tool development, were novices in website programming. It should also be noted that early in the proof of concept development stage, Host flight and track information was used to augment the existing ASDI data. Though not part of the end prototype, much insight from this merging exercise was gained.

The creation of the database daemon and deviation agent to utilize a database uncovered many areas of concern when developing a data-centric software system, most notably:

1. Modular Code Reuse

Based on the SLOCCount source lines of code counting program, had the CTAS software been taken as is and the hooks to retrieve the appropriate deviation data added, our system would have been over 300,000 lines of code. For the prototype, the total count of the deviation agent source code was less the 4000 lines. For the

most part, the core functions that comprise the deviation algorithm were stripped from CTAS. However, several supporting CTAS library functions were also taken. It was the intention to link the prototype software with the existing CTAS libraries, but due to the libraries interdependencies, it was easier to pull out the code. Due to this exercise, work to make the CTAS libraries less interdependent and more able to be used outside the core CTAS processes has begun.

2. MySQL Database Access

It was found that if too many queries were sent to MySQL in a short period of a time, the system would slow down considerably. Since the ASDI feed contains track messages for aircraft on a one-minute update rate, this was not an immediate concern. However when twelve-second Host data were sent to MySQL, the insertions into the database were not timely enough. Using a data daemon to bundle the track messages together for a single large insertion into MySQL helped alleviate the access slow down.

Database access was also an issue when it came to retrieving data for display. For a system running with two hours worth of data, the query for the list of deviating flights took 3-4 seconds, while the accessing the information for a particular deviating flight took 3-6 seconds. This is not practical for a near real-time ATM display. Though not implemented in the prototype, use of the database daemon to store the current list of deviating aircraft and information on each of these aircraft would alleviate this problem. To display this information, the daemon would then need to be queried instead of the MySQL database, but this step would be necessary for timely interaction. If fact, the use of a data cache and subsequently the addition of it into the data-centric model were due, in part, to the MySQL access problem.

3. Disparate data storage

Flight plan information from Host and ASDI data can come from multiple flight plan and flight information message types. Hence, the latest flight plan amendment may contain an updated flight path, but not the filed airspeed, etc. However, when the flight information is needed, the complete set of the latest information, regardless of its source, is usually required. Either the code that inserts the flight data into MySQL needs to merge all relevant flight plan information before insertion, or the code that queries the aircraft flight plan information needs to be able to synthesize the relevant fields from each of the latest flight plan message types into a complete superset of information. It is possible to let MySQL merge the relevant data fields from the latest of each of the flight plan message types. This is known as a join. However it was our experience that MySQL joins can take a significant amount of time, and it is better to keep the queries as simple as possible. So if real-time access to data is a requirement, a process to manage data outside of MySQL is likely to be necessary.

It should also be considered that the database could be used as a record of each of the messages that came into the system. If feasible, it may be advantageous to archive both sets of information, the raw message as it came in, and a synthesized data message that contains the latest information for that aircraft. This would provide the capability of using the same MySQL archive to post-process the data, as well as playing back the scenario.

4. Unique Aircraft Queries

A major problem with the prototype was the identification of unique aircraft. When a track or flight plan update message came into the system, it needed to be identified with the appropriate flight segment, defined as a take off and landing, for that aircraft. As mentioned above, complicated queries slow down MySQL, as well as make it more difficult to interpret the returns from it. To increase the speed of the queries for a particular aircraft, it was found that a unique aircraft identifier would be necessary.

The key is to use information from the aircraft that does not normally change during a flight segment. In the current version of CTAS the call sign, departure airport, and to a limited extent the Host computer id are used. However, the archive should be designed to handle a national traffic data source, so the Host computer id would change as the aircraft flies from Center to Center. In addition, call signs can change for various reasons and airlines routinely use the same call sign for particular flight segments every day. This presents a problem both in real-time and archival MySQL access. As done with the ETMS, it is recommended that each flight segment be assigned a unique number the first time an aircraft message is received by the system for that aircraft. For each additional aircraft message received by the data manager, the aircraft can be matched to the appropriate flight plan message already received for that aircraft. Depending on the incoming message and data source, the information used to identify the matching flight segment flight plan can be one or more of the following:

1) Call sign
2) Source computer id
3) Source name

4) Departure airport
5) Arrival airport
6) Beacon code
7) Time and date of flight
8) Aircraft active flag.

The data manager will need to store enough information from each of the previous flight segments seen in the system to either find the appropriate flight segment to the incoming message or assign a new unique id number.

It is estimated that 40,000 unique flight segments are assigned each day in the ETMS.[16] The design of the system called for the cache to stop writing to the active MySQL archive and start a new one each day. The cache would then seed the new MySQL archive with information on all aircraft that have not yet landed. The number of unique id values would need to be large enough to at least cover this transfer between archives. This would give us the ability to track aircraft flights between multiple archives.

5. Use of Web Browser

A main focus of the prototype demonstration was the use of a web browser to access the deviation data. The scope of the proof of concept prohibited a long development period for the display and the use of a web server allowed for rapid prototyping. This also supported the ability to run the demonstration on the wireless PDA. However, the web browser itself was a problem.

The purpose of our display was to present a list of deviating aircraft and allow a user to see plots of the flight plan and track history for that aircraft. A typical web browser has many features that can either take the focus away from the deviation utility, or go to a different site completely. Web browsers also have a built in caching of data that needs to be controlled. In addition, the implementation outlined above displayed only images of the track and flight plan information, with no zooming or panning interaction capability. While it is possible to create a web utility that is more interactive, it appears that it would be more appropriate to invest in the development of a thin display client that would interface with the Data Manager and display the appropriate data, with sufficient ability to zoom, pan, and interact with the data. This would also provide the developers with the ability to completely control the display environment.

VI. Conclusions

The data-centric design model shifts the question from how a particular problem is solved to what information is needed to solve that problem. Implicit in this statement is that the algorithm will still need to be developed, but focusing on the data allows for compartmentalization of the solution. Storing derived data in a centralized database or manager allows for small isolated parts of the overall algorithm to be developed separately, as needed, without impact to the rest of the system. This increases the re-use of separate modules and decreases the maintenance of the finished system.

Another benefit is that over time, individual modules, such as route parsing or trajectory modeling can be re-engineered and replaced as new technology is developed. This is possible due to the focus on the data itself and not the algorithm. As long as the interface to the data stays the same (or only changes slightly), the impact of an algorithm change would be lessened.

In a data-centric model the management of the data is of primary importance. Well-defined access to the data is necessary for the algorithms to perform properly. A system that revolves around the data will need to ensure the quality of that data. One aspect of this is the dual nature of the data access, real-time data and archival data. Access to the data needs to be well defined so that requests for an aircraft's flight plan can be uniquely identified between yesterday's, today's, and tomorrow's flight of the same name.

As seen from the flight deviation proof of concept, web-based access to the data can be problematic. It should be stated however that this was one implementation. There are many examples in industry where web-based access to data can be managed in such a way as to provide near real-time access to the information. Those should be explored, but were out of scope for the proof of concept as developed.

Built into the data-centric model is the use of a database, such as MySQL. Though a data manager is recommended when using the database for real-time processing, the ability to store daily (or even hourly) snapshots of all of the data allows for the development of powerful analysis capabilities. Processes will no longer need to output specialized analysis files; all data will be archived on a daily basis and can be stored for as long as needed.

Acknowledgments

The authors would like to thank David A. Wheeler for the use of SLOCCount source code counting program.

References

[1] Erzberger, H., and Nedell, W., "Design of Automated System for Management of Arrival Traffic," NASA, TM 102201, Ames Research Center, June 1989.

[2] Erzberger, H., Davis, T. J., and Green, S., "Design of Center-TRACON Automation System," AGARD Meeting on Machine Intelligence in Air Traffic Management, Berlin, Germany, May 11-14, 1993.

[3] Green, S. M., Vivona, R. A., "En Route Descent Advisor Concept for Arrival Metering," AIAA 2001-4114, Guidance, Navigation, and Control, Montreal, Canada, August 2001.

[4] Davis, T.J., K.J. Krzeczowski, and C. Bergh, "The Final Approach Spacing Tool," IFAC Thirteenth Symposium on Automatic Control in Aerospace, Palo Alto, CA, September, 1994.

[5] "Blueprint for NAS Modernization (2002 Update)," FAA, Oct. 2002.

[6] Dillingham, G. L., "Air traffic control: role of FAA's modernization program in reducing delays and congestion," General Accounting Office, GAO-01-725T, May 2001.

[7] "Next Generation Air Transportation System: Integrated Plan," Department of Transportation, Joint Planning and Development Office, December 2004.

[8] Viets, K. J., Taber, N. J., "An Overview of a Flight Object Concept for the National Airspace System (NAS)," The MITRE Corporation, McLean, VA, MTR 00W0000085, Sept. 2000.

[9] Quinn, C., Zelenka, R. E., "ATC / Air Carrier Collaborative Arrival Planning," *2nd USA/Europe Air Traffic Management R&D Seminar*, Orlando, Florida, December 1998.

[10] Laudeman, I. V., C. L. Brasil, and P. Stassart, "An Evaluation and Redesign of the Conflict Prediction and Trial Planning Planview Graphical User Interface," NASA TM-1998-112227, April 1998.

[11] Mead, K. M., "Next Steps for the Air Traffic Organization," Department of Transportation, CC-2005-022, April 2005.

[12] Post, J., Knorr, D., "Free Flight Program Update," *5th USA/Europe Air Traffic Management R&D Seminar*, Budapest, Hungary, June 2003.

[13] "Operational Evolution Plan Version 8," FAA, May 2006.

[14] Rentsch, T., "Object oriented programming," ACM SIGPLAN Notices, v.17 n.9, p.51-57, September 1982.

[15] Swenson, H. N., Hoang, T., Engelland, S., Vincent, D., Sanders, T., Sanford, B., and Heere, K., "Design and Operational Evaluation of the Traffic Management Advisor at the Fort Worth Air Route Traffic Control Center," *1st USA/Europe Air Traffic Management R&D Seminar*, Saclay, France, June 1997.

[16] Clayton, J., Murphy, J., "Traffic Flow Automation System (TFAS) Analysis Report," NASA AATT RTO 57 final report.