A GENERIC PLATFORM FOR BUILDING AIR TRAFFIC ENVIRONMENTAL INTERNET SERVICES

J. Weggemans, J. van Weert
National Aerospace Laboratory NLR
P.O. Box 153, 8300 AD Emmeloord
The Netherlands

ABSTRACT

Communication is getting more important for airports, airlines and Air Navigation Service Providers (ANSPs) for building good relations with the public community.

When bringing air traffic information to the public there is an area of tension regarding openness, transparency, security and privacy. On one hand the public wants as detailed and up-to-date flight information as possible. On the other hand the provider of air traffic data wants a certain level of shielding of privacy related data (pilot, airline) and needs to protect the company data sources against unauthorized access from the outside world.

This paper presents a generic and flexible system architecture, enabling communication of ATC sensitive information to the wide Internet public taking into account the above area of tension. The system architecture was used as a basis to develop the air traffic Internet service for the Deutsche Flugsicherung DFS.

1. INTRODUCTION

Explaining where planes fly and why is one of the challenging tasks in community communications. In close cooperation with the Deutsche Flugsicherung (DFS) the National Aerospace Laboratory (NLR) in the Netherlands developed an application which shows aircraft movements in the vicinity of airports using the Internet. This Internet service, currently deployed for seven German airports, offers functionality for displaying the near real-time traffic picture in airspace and the archived summary flight tracks of the past.

In such an application data from a trusted secure business environment needs to be processed in a way that it is suitable for external delivery to the insecure World Wide Web. A generic and flexible system architecture is introduced to support:

– Protection of data sources which are part of the company infrastructure, such as radar data and flight track monitoring systems, against unauthorized access
– Restrictions to information delivery such as time delay, refresh rate, flight details, area of coverage
– Secure encryption of privacy related data

In addition, the system architecture supports a platform for building Internet based applications which are highly configurable to customer needs.

It is therefore relatively easy to reuse the generic architecture to other domains such as noise and other viewing options. The architecture provides a solid and future proof platform in which data and modules can be embedded from various sources such as different noise calculation modules or GPS oriented data.

For example, the architecture was used to interface with NLR's ATC Research Simulator facility to support research studies focusing on capacity and noise impact of new ATC concepts, with air traffic controllers in-the-loop and real-time visualization of the noise impact.

2. STARTING POINT

DFS had the task to realize a publicly available Internet service showing aircraft movements in the vicinity of German airports. Currently DFS operates a monitoring system at each airport, where flights (flight tracks and administrative information) are stored in a database. The information is used to correlate complaints about aircraft noise to flights. Implementing a publicly available Internet service showing where and how planes fly is an extension to the noise abatement process and also increases the external communication.

For building such a service DFS described specific and demanding requirements for:

– Integrity, data published must have an accuracy similar as the airport's local flight monitoring system (reference system)
– Security,
  • deliver information to the general public with half an hour delay
  • shield company systems from the outside, using a one way communication channel to the web server
– Privacy, encryption of specific data
– Functionality,
  • show near real-time aircraft movements and history flight paths
  • Human Machine Interface must not have the look of an ATC display

However, at the start of the project there were many uncertainties such as: what information has to be shown, for how many airports, what are the data sources, how will the HMI look like, do all the airports have the same requirements. So from a system designers perspective there was a need for a system architecture capable of fulfilling a secure transfer of target aircraft position data to the Internet in a flexible and extensible way.

3. SYSTEM ARCHITECTURE

The system architecture is based on a three-tier model (see figure1).

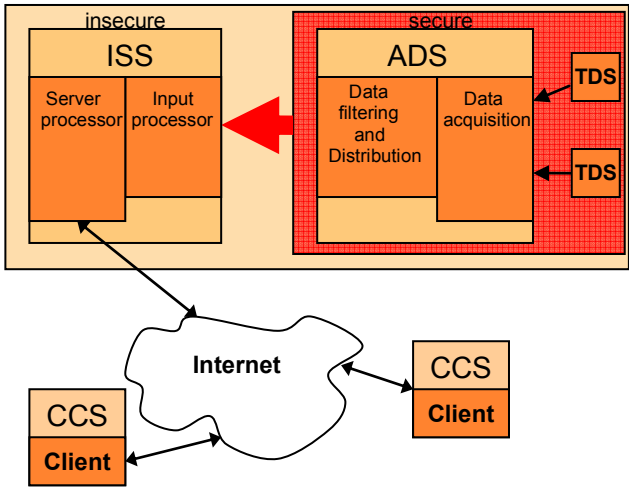

FIG 1. System Architecture

This architecture basically consists of a client server architecture extended with a third tier responsible for gathering and delivery of data. The following systems and main functionality are involved:

- ADS, the acquisition and distribution system at the back-end, responsible for:
  - connecting to and collecting data from Target Data Sources (TDS) such as flight tracking systems, raw radar data and flight administration data suppliers
  - delivery of data to be published via a uni-directional communication channel to the ISS
- ISS, the Internet Server System responsible for:
  - Receiving and storing data from ADS
  - External delivery of data to CCS
- CCS, the Client Customer Systems. These are the end-user systems (PC) connecting to the ISS, retrieving data (and application) and displaying all data to the end-user. CCS provides the HMI.

The three-tier model supports an architecture of separate independent components where changes in requirements will be isolated to certain components/tiers. To service and implement a flexible system architecture also design patterns are applicable such as:

- Program to an interface and not to an implementation. Generic classes, such as flight(s), track(s), plot(s), are based on a hierarchy with a root class defined as an abstract class/interface defining only the method signatures. All derived classes implement these methods according to specific needs.
- Observer pattern to define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.
- Data Access Object pattern to abstract and encapsulate all access to (external) data sources. The DAO manages the connection with the data source to obtain and store data.

The next section addresses in headlines each system and related main components.

3.1. Acquisition and distribution system

The data acquisition and distribution system (ADS) consists of the main components Data acquisition and Data filtering and Distribution.

3.1.1. Data acquisition

To realize a service that supports tracking of aircraft movements the system needs a target data source delivering aircraft position data. The basic equipment for this is a unit delivering raw radar messages in a standardized surveillance format such as ASTERIX [1].

However, using these raw radar data messages directly has some disadvantages, such as:

- Data suffers from inaccuracies due to:
  - Reflections, signal propagates via high buildings for example
  - Outliers, incorrect radar measurements
  - Gaps, aircraft is temporally not tracked because SSR modeA antenna on the aircraft is hidden for the radar
- General public gets the raw radar data and can get insight in the (proper) functioning of company surveillance data and systems
- Display update rates faster than the turnaround time of the radar are not possible

These disadvantages can be eliminated by using a flight tracking system, such as NLR's flight tracking and aircraft noise monitoring system FANOMOS which is currently operated by DFS at German airports [7]. Such a system uses a flight track reconstruction module where raw radar plots are converted into smoothed flight tracks using a sequence of outlier check, skipping the incorrect radar measurements, followed by a smoothing and data reduction step based on B-splines [2]. The resulting polynomial flight track can be used to generate aircraft position data at any time during the duration of the flight track.

There is time for applying this kind of smoothing algorithm because of the security requirement for having a certain time delay before real-time position data gets propagated to the Internet. To implement this smoothing technique, NLR's flight track reconstruction module was used.

Within the flight track reconstruction module raw radar plots are received and organized into separated raw flight tracks based on equality of identifier (for example SSR modeA code) and on time. In case it is the first plot, a new raw flight track structure is created; otherwise the plot is added to the existing flight track. At each turnaround of the radar the collection of raw flight tracks is checked for growth. In case no data is received anymore for a certain flight (Normal track closure condition: i.e. aircraft has landed or has left area of interest), the raw flight track will be further processed: sequence of outlier check and B-spline curve fitting.

The produced polynomial flight track is stored, and its raw

flight track representation is removed from the raw flight track collection. However, there will be circumstances under which this reconstruction will start too late, beyond the required delay time. This happens when the flight route and thus the flight duration is longer than the required delay time. To guarantee in time delivery of all aircraft position data, the criteria for activating the flight track reconstruction were extended with a time based condition, see figure 2. Every *n* minutes (e.g. 5 minutes) each raw flight track within the raw flight tracks collection is submitted to the flight track reconstruction module producing updated growing track segments. The resulting track segments are made available to the plot generator in a way that existing track segments will be overwritten. Finally, the plot generator is able to generate position data at time *t* using the B-spline algorithm.

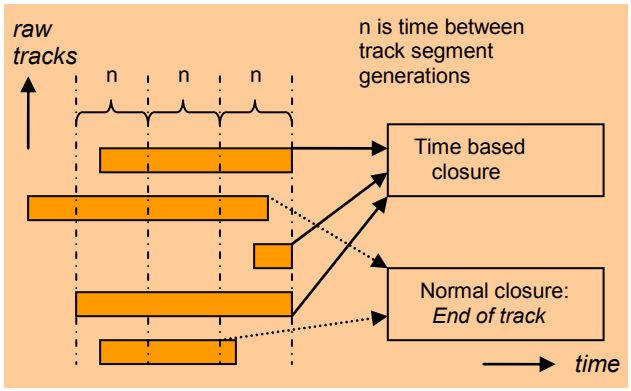

FIG 2. Track segment generation

To support not only aircraft movements derived from a flight tracker module, but also other movements such as real-time radar plots, a loose coupling between the consumer and the data source implementation is needed. The DAO design pattern is applied to handle this data source abstraction and encapsulation. Figure 3 shows a consumer (RTPlots) and a producer (Collect Tracksegments) communicating via a shared data source (RTplotsDS), where the specific data source implementation is created via a factory.

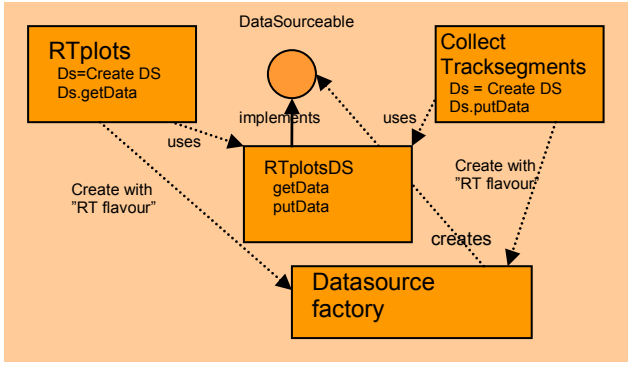

FIG 3. DAO Pattern and Interfaces

The aircraft movements are delivered as generic plot objects consisting of general attributes such as position, timestamp, speed, identifier etc. The plot objects are part of the generic object model for flight(s), track(s) and correspond to the design pattern of programming to an interface and not the implementation. Besides of radar data sources other kind of data sources must be consulted such as databases containing flight tracks and flight administration data. Flight administration data are collected with the purpose to extend plots with flight details such as aircraft type.

3.1.2. Data filtering and distribution

The data filtering and distribution component is responsible for:

- Filtering of data
- Delivery of filtered data to the Internet Server System

One important requirement is the shielding of company systems. The data acquisition and distribution system ADS, part of the secure company environment, fulfills an important role in this. The ADS communicates to the outside by means of a uni-directional channel. This means ADS will setup a communication link towards the Internet Server System ISS, where the ISS will only listen. There is no way where the ISS, located in an insecure environment, can request for information to the ADS.

To get a flexible way of data communication, objects and data will be converted into XML-messages. The transfer characteristics depend on the kind of object/data to be sent and can be divided into dynamic objects, such as plots, flights, tracks, noise, and the more static data like geographical background images that will change occasionally. Besides the distinction in dynamic and static objects/data, a further discrimination is made according to the priority of objects to be sent. Real-time data needs a high send priority. Looking to the XML-based data stream the main data element will be real-time radar plots where gaps are filled with lower priority data elements such as flight tracks.

Before objects and data travel to the Internet Server System they flow through a *chain of filters* [3]. The filter chain fulfills the task of what and how things will be delivered. There are filters that support encryption of attributes of objects, filters for geographical transformation for example from UTM coordinates to WGS84 latitude/longitude coordinates, and filters for object skipping based on time and unique identification (e.g. callsign, SSR modeA). The chain of filters makes the system architecture flexible and well organized. Flexible because new filters can easily be added to the chain (not in runtime), and existing filters can be activated or deactivated in runtime. Well organized because all transformations are located at one place just before distribution.

A separate controller tool, ran by an authorized user (administrator), controls how filters operate within the chain. The controller settings and other parameters whose values may vary depending on the airport are stored in a permanent store. All parameters are stored in a kind of tree starting with a root, consisting of company (e.g. DFS), airport identifier (e.g. EDDF), system selector (ADS, ISS, CCS), data tag (e.g. flightplan, map), and finally the attributes. To support fast responses to clients a memory caching is available that will be refreshed in case administrative parameters have been changed.

A special treatment exists for the background images needed within the visualization system (see 3.3). Original background images are transformed into a map structure according to Google's KML format [5]. The KML standard is XML based and easy to parse within the visualizer. The ADS also supports a sub task for splitting images into smaller segments based on a certain tile size. Each image or tile will be stored as KML XML definition file containing descriptive information such as image boundary box coordinates and a reference to the file with binary contents representing the real image/tile. By doing the image processing at the ADS the ISS load will be reduced.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <GroundOverlay>
    <name>DFS%2FEDDF_Normal_112</name>
  - <Icon>
      <href>DFS%2FEDDF_Normal_112.jpg</href>
    </Icon>
  - <LatLonBox>
      <north>50.3071</north>
      <south>50.00317060194175</south>
      <east>9.240857</east>
      <west>8.9073944</west>
    - <Lod>
        <minLodPixels>64</minLodPixels>
        <maxLodPixels>350</maxLodPixels>
      </Lod>
    - <LookAt>
        <longitude>8.9073944</longitude>
        <latitude>50.3071</latitude>
      </LookAt>
    </LatLonBox>
  </GroundOverlay>
```

FIG 4. KML XML example

To support some kind of load balancing on the Internet Server systems, the data filtering and distribution component supports multiple data feeds to multiple Internet Server systems.

3.2. Internet server system

The Internet server system (ISS) consists of two main components. The first is the Input Processor

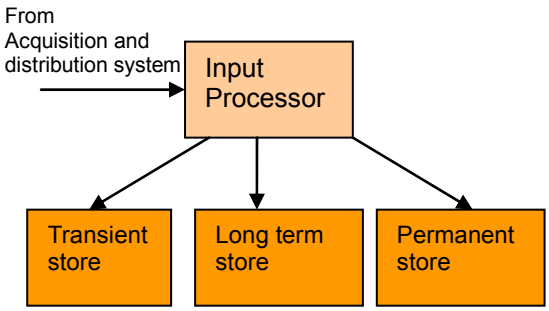

FIG 5. Input Processor component

3.2.1. Input Processor

The Input Processor is responsible for:

- Retrieving data from the ADS acquisition and distribution system
- Storing the data in one of three different data stores

The data retrieval from the ADS Data acquisition system is a one-way data stream. The input processor listens for incoming connections and parses the retrieved XML based data stream. Depending on the type of data, the data will be stored in one of the available data store families: transient, long term or permanent.

3.2.1.1. Transient store

The most volatile data store: the data will be essentially stored in memory. The data which qualifies for the transient store must have a temporary and/or temporal nature, such as real-time flight information, or information for the Client system which is updated within a reasonable timeframe. Whether a data type belongs to the Transient or Long term store depends also on the consequences of a possible crash of the physical Internet server system. If data loss could be a problem, the data should be stored in the Long term store.

3.2.1.2. Long term store

Data which will be sent only once (or must be protected against data loss) must be stored here, because of the one-way data stream the Internet Server system has no means of requesting a resend of data. It is the responsibility of the Input processor to detect data loss. Candidates for this type of storage would be flight data history which can be searched by the Client system.

3.2.1.3. Permanent store

Candidates for the permanent store are static data such as landscape maps or coordinate data which aren't subject to change in a nominal situation. In general it will receive data which will be stored in case of structure changes. Most of the time the (re)sending of this type of data is triggered by human (Administrator) intervention.

It is the responsibility of the Data filtering and Distribution component to ensure that the data streams are optimized according to their size. When for example a lot of data is sent in a small timeframe there is a possibility of data loss. The Input processor has no means to request for a resend.

3.2.2. Server Processor

The second component is the Server Processor

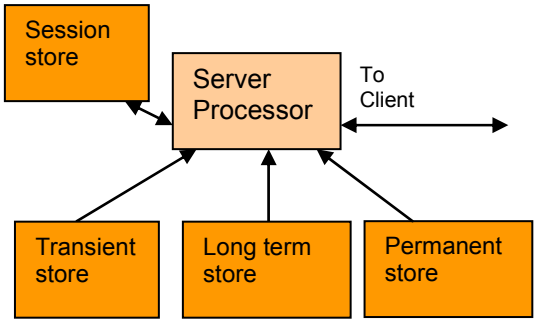

FIG 6. Server Processor component

The Server Processor is responsible for:

- Maintaining communication with the Client
- Delivery of data to the Client
- Preventing “bad boy” Client to hog up the external distributions

The Server Processor only acts on requests from the Client (pull algorithm). The Client takes the initiative to start a session. The session will be put into the Session store.

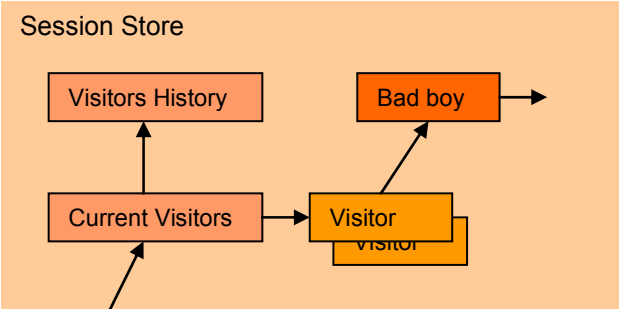

FIG 7. Session Store

The Session Store is responsible for:

- Maintaining a visitor history (for statistics)
- Maintaining a list of current visitors
- Managing potential “Bad Boy” Clients
- Logging information

A “Bad Boy” Client is a non official Client which communicates with the Server Processor. Typically this will be a third party created Client. The Session Store will fingerprint the information send from the Client, and tries to detect invalid messages. When such is detected the visitor will be forced to logout.

The session store will also provide detailed logging information about the sessions. First there is the “current visitors”, a simple list of the current users of the system. Besides that the computational load of the system is logged. There is also a more long term logging facility which logs all relevant communications. The format being used is the Apache Combined Log Format [6]. This format is a standard and often used for Web server access logs. The advantage is that standard tools can be used to analyse the written log files.

The Server Processor delivers the data of all three data stores to the Client.

3.2.2.1. Transient store

The Client will frequently ask (pulls) for information from the Transient store, the frequency itself is part of the Transient store. The Server processor tracks for every session if there is new data available. In this way the data is send only once and the Server processor load is kept within limits. The type of communication is based on a RESTful web service, which essentially means that the Client sends HTTP POST URL requests, and the Server processor replies with XML documents.

3.2.2.2. Long term store

The Long term store is accessed for more static data, e.g. summary aircraft tracks. The Client first sends a REST request for a list of available long term store Unique Identifications (UID). For example: arrival flights of a certain airfield in a specific time range. After that the Client will ask for the content data for every UID. The challenging task for the Server processor is to split and construct e.g. aircraft tracks according to the given timeframe.

3.2.2.3. Permanent store

The permanent store contains timeless data, e.g. background maps. The Client will typically ask for these data in the startup phase, or when more structural change is required, e.g. show a different map. Currently the permanent store is responsible for:

- storing and sending map structure
- storing and sending map representation

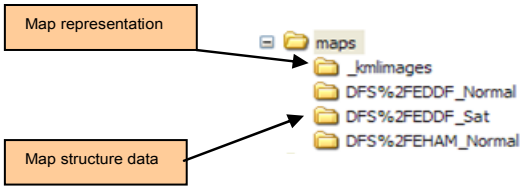

FIG 8. Map directory structure

The map structure data is constructed and delivered by the ADS Acquisition and Distribution system. This prevents extensive calculations for the Internet Server system. The standard used to define the map structure data is Google’s KML (Keyhole Markup Language) format [5].

The KML standard is XML based and will be easy to parse for the Client system, which will be Java v1.5+ (embedded XML support). By choosing the KML standard it is fairly easy to perform integration with other KML based applications, such as Google Earth.

The KML separation in the Internet Server system is based on company (e.g. DFS), the airfield (e.g. EDDF) and map type (e.g. Normal/Sat). The Data Distributor will create extra zoom levels (XML files).

The map representation will be the binary content of the map structure data. The data is already „tiled“ by the Acquisition and Distribution system. The map representation server has the possibility for caching the data in memory in case of heavy load on disk access.

3.3. Client

The Client system consists of three parts:

- Data retriever
- Buffer manager
- Visual manager

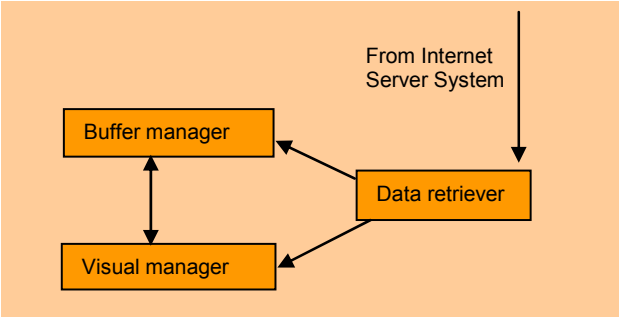

FIG 9. Client system

3.3.1. Data retriever

The Data retriever maintains the communication with the Internet Server system. Depending on the view of the Visual manager (which data display(s) are enabled) it will ask for relevant data. When data is required from the Transient store it will first retrieve the preferred communication message interval, this to prevent from becoming a “bad boy”. The “bad boy” factor is related to the update time of the requested data. Experiments were conducted and the conclusion was that this factor should be half the value of the update time. E.g. when real-time aircraft data is requested and their plots are updated every 10 seconds, the factor will be 5 seconds. This will cover possible “hick-ups” in the connection to the Internet Server system. The transient data is transferred to the Buffer manager.

The Long-term store data is retrieved sequentially. After every transfer a small delay is build in, in order to prevent a possible server overload (when multiple Clients are retrieving long-term data). The long term store data is directly send to the Visual manager, because of its non-temporal state there is no reason to pass it to the Buffer manager.

The Permanent store data, such as background maps, is retrieved as bulk data from the Internet Server system. The Server processor component involved has no Session store attached and acts like a session-less REST interface. The data is directly sent to the Visual manager.

3.3.1.1. Communication

The Client communicates with the Internet Server system using HTTP URL calls (POST calls). POST is being used instead of GET because of possible message size limitations. An example POST body would look like:

```
?tag=Cmd=GetObjects&tag=Root=3dflight_rt%3A
%2FDFS%2FEDDF%2FOnline&
```

The information is URL encoded and is equal to the format used for URL GET calls. This example asks for aircraft position data from the transient store (Online) of airfield EDDF. The Internet Server replies with a `text/xml` content type document; see figure 10.

```
<?xml version="1.0" encoding="utf-8" ?>
- <wowserver>
  - <reply>
    - <result>
      <object no="0" Root="3dflight_rt:/DFS/EDDF/Online/9855611B"
        Name="9855611B" time="1181896483172" x="8.716599" y="50.0715" h="2300.0"
        head="-2.7856479" type="Def" actype="B772" />
      <object no="1" Root="3dflight_rt:/DFS/EDDF/Online/DF70FF11"
        Name="DF70FF11" time="1181896483172" x="8.651199" y="50.0607" h="1200.0"
        head="-2.7840316" type="Def" actype="B735" />
      <object no="2" Root="3dflight_rt:/DFS/EDDF/Online/C9A7901D"
        Name="C9A7901D" time="1181896483172" x="8.8616" y="50.113" h="4000.0"
        head="-2.6207478" type="Def" actype="A319" />
      <object no="3" Root="3dflight_rt:/DFS/EDDF/Online/FB44E8AC"
        Name="FB44E8AC" time="1181896483172" x="8.1008" y="50.0687" h="8100.0"
        head="2.554114" type="Def" actype="B744" />
      <object no="4" Root="3dflight_rt:/DFS/EDDF/Online/1DFFEC63"
        Name="1DFFEC63" time="1181896483172" x="8.4094" y="49.900097" h="7000.0"
        head="-2.3655787" type="Def" actype="A306" />
      <object no="5" Root="3dflight_rt:/DFS/EDDF/Online/36669AB4"
        Name="36669AB4" time="1181896483172" x="8.5264" y="49.9913" h="1600.0"
        head="-1.6668601" type="Def" actype="B735" />
    </result>
  </reply>
</wowserver>
```

FIG 10. Server reply with aircraft position data

3.3.2. Buffer manager

The buffer manager is responsible for a fluent feed of data. It will buffer the data to eliminate lagging effects of the communication between the Client and the Internet Server system (i.e. delays in Internet traffic). The buffer time itself is fixed and will remain so during the lifecycle of the system. Currently only the data of the Transient store is send to the Buffer manager.

3.3.3. Visual manager

The visual manager is the HMI and represents the actual presentation of all data to the user. It consists of four main parts:

- Transient store data display
- Long term store data display
- Options and info display
- Permanent store display (KML viewer)

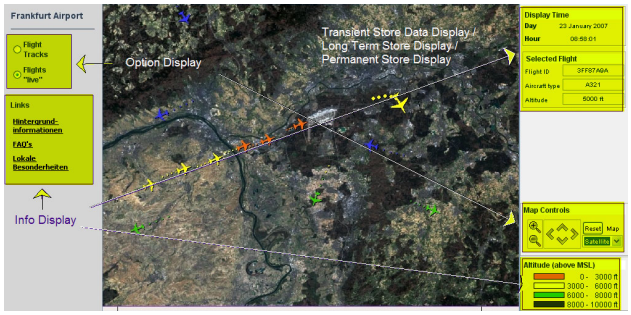

FIG 11. Composition of the displays

3.3.3.1. Transient store data display

This data display presents the data from the Transient store. How it is displayed depends partly on the data of the Option and info display. Because of the temporal properties of the data the HMI checks the state after every update of the Transient store. For example, the user can select data which in time will be removed from the display. Any extra data shown in the Option and info display are then removed also. The Observer design pattern is used to handle this kind of one-to-many dependencies.

The prime example of the Transient store data is shown in figure 12. Aircraft data are presented to the user as aircraft symbols with a certain heading.

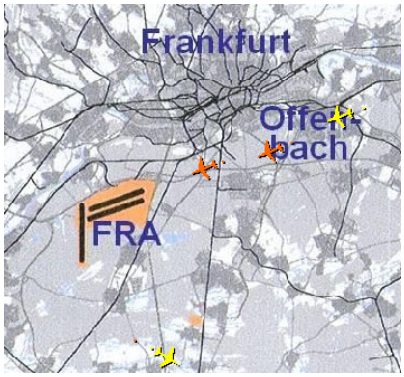

FIG 12. Transient store data display

3.3.3.2. Long term store data display

This data display will present the data from the Long term store. Like the Transient store data display its presentation depends partly on data of the Option and info display. An example of the Long term store data display is given in figure 13, showing flight tracks.

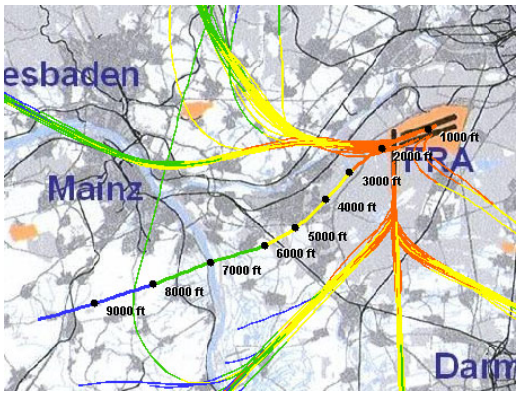

FIG 13. Long term store data display

3.3.3.3. Option and info display

The option and info display holds the GUI elements to control the application and to show secondary information.

Option displays contain interactive GUI elements which control the look or behaviour of the application. The Info display has no interaction and only shows data about the total context or the currently selected object. The content of the info displays are retrieved from the Transient store and can be changed by the Administrator.

3.3.3.4. Permanent store display (KML viewer)

The permanent store display is basically a 2D KML viewer. Its primary task is to show a map on which the Transient store data and Long term data displays can render their information. The Permanent store display simulates the behaviour of Google Earth in two dimensions.

3.3.3.5. Pluggable displays

The Visual manager can hold any number of data displays, so it is possible to have multiple data displays synchronized to one and the same Data retriever.

The additional display could display the data in e.g. 3D, or show the data in their numerical presentation. The Client is responsible for sending all data to the pluggable displays.

4. APPLICATIONS

The generic system architecture was used as a basis to develop the air traffic Internet service for the DFS. This publicly available service offers functionality for publishing the near real-time traffic picture in airspace and the archived summary flight tracks of the past. This service, called STANLY_Track [9], is currently deployed for seven German airports. Besides of this operational service other domains and applications were explored.

With relatively low effort the system was extended with a noise calculation and visualization module. For each real-time aircraft position or flight track a simplified fast noise calculation model is activated. The noise results are added to the plot/flight track. This application was further explored at NLR by connecting to NLR's ATC Research Simulator (NARSIM) facility. Interfacing to NARSIM enables research studies focusing on noise impact. The connection provides a fast-time visualization of noise impact for simulated traffic, and shows impact of changes in flight route and procedure.

It is also possible to use the application data as a continuous feed of aircraft movements which can be fed to other applications such as Google Earth. By combining the data of multiple airports it is possible to get an overview of the air traffic for larger areas.

5. CONCLUSIONS AND FUTURE WORK

NLR developed a flexible framework that has proven in real practice that it can be used to implement services to make ATC data available to the Internet public. In close cooperation with the DFS the so-called STANLY_Track application was developed dealing with the constraints for bringing ATC data from the company to the wide Internet.

As part of future work, the authors foresee extension of the framework to other domains (emission, external safety) and environmental issues as prediction of traffic, noise. Also further developments can take place related to the KML-viewer to view elements as tracks, plots and noise which have to be defined by means of KML.

6. ACKNOWLEDGEMENTS

This work was carried out by the National Aerospace Laboratory NLR in the Netherlands in close cooperation with the Deutsche Flugsicherung DFS. Many thanks are to the employees of the Lage- und Informationszentrum LIZ of the DFS for their support and their initiative for introducing such a service as the first one in Europe.

7. GLOSSARY

ADS	Acquisition Distribution System
ATC	Air Traffic Control
CCS	Client Customer System
DFS	Deutsche Flugsicherung DFS
DAO	Data Access Object
HMI	Human Machine Interface
ISS	Internet Server System
KML	Keyhole Markup Language
NLR	National Aerospace Laboratory
SSR	Secondary Surveillance Radar
XML	Extensible Markup Language

8. REFERENCES

[1] ASTERIX documents available at Eurocontrol web site: http://www.eurocontrol.int/asterix/public/standard_page/documents.html
[2] Pavlidis, T, *Algorithms for Graphics and Image processing*, Bell Laboratories, Springer-Verlag, Berlin-heidelberg, 1982.
[3] Flow-based programming http://en.wikipedia.org/wiki/Flow-based_programming
[4] RESTful web service http://www.xfront.com/REST-Web-Services.html
[5] Google KML definition available at the Google api website: http://code.google.com/apis/kml/documentation/index.html
[6] Apache combined log available at the Apache website http://httpd.apache.org/docs/1.3/logs.html#combined
[7] Flight track and Aircraft Noise Monitoring System FANOMOS, see NLR web site: http://www.nlr.nl/documents/flyers/f158-04.pdf
[8] The Java web site: http://www.java.sun.com
[9] STANLY_Track tool hosted at the DFS web site: http://www.dfs.de

Johan Weggemans was born in the Netherlands in 1964. In 1987 he received the Bachelor degree in technical Informatics from the Informatics Institute for Higher Education Hogeschool Drenthe.
Since 1989 he is software engineer at the National Aerospace Laboratory NLR. His work focuses on development of information systems to determine the impact of aviation in relation to environment and safety. These systems are used for environmental research studies at NLR, but have also been realized as turn-key projects for airports (civil, military), the Department of Civil Aviation and Air traffic Control in the Netherlands and abroad. Examples are systems for flight track monitoring, capacity and noise management, and for visualization of air traffic and environmental impact either as stand-alone or within a distributed client/server enterprise or web environment.

Jacco van Weert is Senior Software Engineer at the National Aerospace Laboratory NLR. He has a Bachelor degree in computer science from Higher Education Hogeschool Enschede. His work covers the whole computer science arena, from 3D visualization systems, web based services and applications to data models and content repositories.
Jacco was born in the Netherlands in 1968.