Technical Report:
Design and Implementation of DANTi

Paolo Masci
NASA Langley Research Center
paolo.m.masci@nasa.gov

November 17, 2023

Abstract

This technical report describes the design and implementation of the DANTi prototype (version 2.0), a Detect-And-Avoid application intended to run on an Electronic Flight Bag.

1 Introduction

DANTi is a Detect-And-Avoid application (app) intended to run on an Electronic Flight Bag. Detect-And-Avoid (DAA) refers to the ability to see, sense or detect conflicting traffic or other hazards and take the appropriate action. [4] An Electronic Flight Bag (EFB) is a device that includes applications allowing flight crews to perform a variety of flight management tasks such as trip scheduling, take off and en-route calculations (a list of EFB applications is in Appendix A of AC 120-76D [7]).

DANTi computes route conflicts, traffic alerts and maneuver guidance in real-time using live traffic data. Feedback on route conflicts, traffic alerts and maneuver guidance is presented to the flight crew using an interactive display with a navigation moving map and flight display instruments (see Figure 1). The system is meant to be used by the pilot in command to augment information available on the standard flight displays of the aircraft. It is not a substitute or replacement for any cockpit equipment required by airworthiness or operating regulations. The original design concept was introduced in [3]. Flight tests and experiments with preliminary prototypes are described in [1] and [2].

2 Design

DANTi is an interactive application designed to run on a table. The prototype includes several display elements that are typically part of primary flight displays in a cockpit.

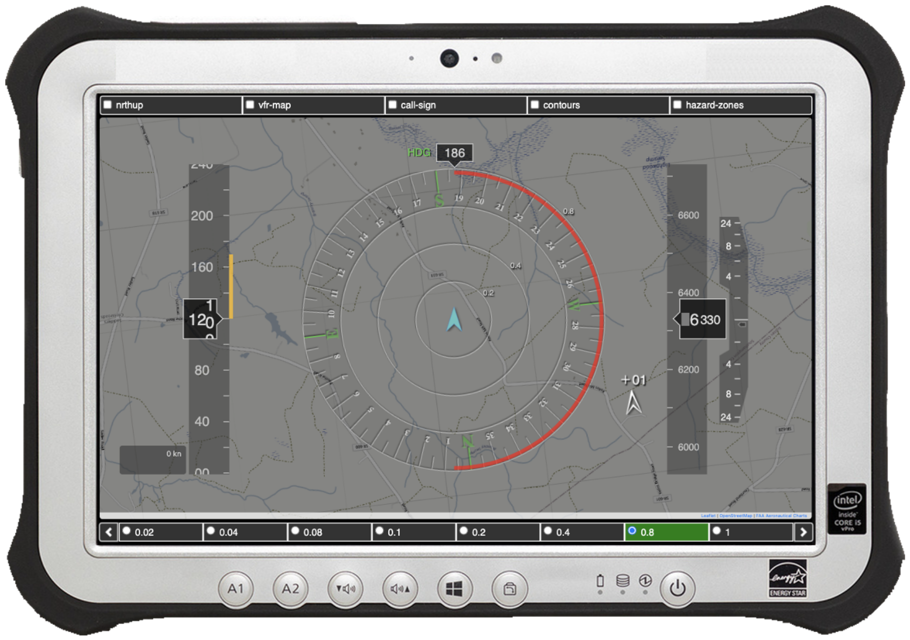

Figure 1: DANTi prototype executed on a tablet.

The ownship is shown at the center of the display, as a blue chevron. A moving map is shown in the background and follows the position of the ownship. The map can be in street mode or VFR charts mode. When in street view mode, the map displays landmarks and city names. When in VFR chart mode, the map shows sectional charts used in aeronautics for navigation under visual flight rules. Traffic aircraft are shown using standard DO-365 symbols. Each traffic aircraft has a label, indicating the relative altitude of the aircraft wrt the ownship, as well as the climb/descend trend. For example +01 indicates that the aircraft is 100 feet above the ownship. If the label does not include an arrow, as in this case, it means that the aircraft is on a level flight. Call sign and tail number can also be displayed as part of the label of each traffic aircraft.

A compass indicates the heading of the ownship. The compass can be in track-up mode, where the ownship points up and the compass rotates, or in north-up mode, where the compass is fixed and the ownship rotates.

Tape indicators on the side of the display indicate ground speed (on the left side of the display), altitude and vertical speed (on the right of the display).

DAA maneuver guidance is provided using color-coded bands. Bands displayed on the compass provide information on heading maneuvers. For example, a red band on the compass is called peripheral band, and indicates heading maneuvers that should be avoided because they would result in loss of well-clear. Another example is a yellow band on the groundspeed tape. This is called preventive band, and indicates speed changes that should be avoided because they

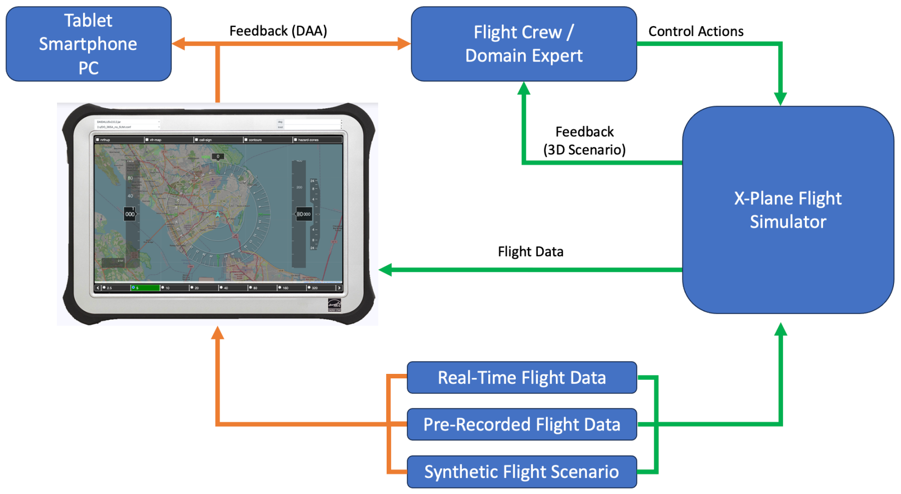

Figure 2: DANTi connection options.

would result in loss of well-clear. When the entire compass is red then there was a loss of well-clear, and a green wedge on the compass is used to indicate a range of heading directions that can be used to regain well-clear in the fastest and most efficient way.

Other ways to provide DAA maneuver guidance include contours and well-clear volumes. Contours are regions of the airspace that should be avoided to prevent a loss of well-clear. Well-clear volumes are protected areas around the aircraft. They are cylindrical volumes elongated in the direction of the relative velocity between the ownship and the aircraft. Voice feedback can be used to signal traffic alerts to the pilot. All display elements and alerts/guidance can be fully customized.

A range of different connection options are supported, which can be used to set up both fast-time simulations and interactive human-in-the-loop experiments (see Figure 2).

DANTi can receive flight data from different data sources, including:

- **Synthetic flight scenarios** generated by computer programs to study specific encounter geometries.
- **Pre-recorded flight data** based on pre-recorded traffic information from real-world flight scenarios.
- **Real-time flight data** obtained from the ADSB equipment of the ownship or streamed live from an aircraft flight simulator.

DANTi can be used to show visual feedback to human operators, as well as to send DAA results to other software applications. Examples include:

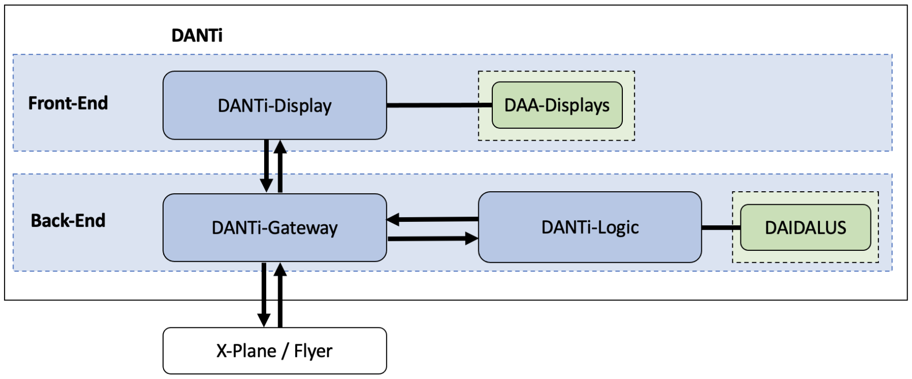

Figure 3: DANTi architecture.

- **Flight Crew**, human operators and personnel performing interactive human-in-the-loop simulations and flight tests.
- **Virtual Pilot**, a computer program that automatically reacts to alerts and maneuver guidance according to a set of rules and performance requirements.

Feedback loops can be created to perform interactive flight simulations with DANTi:

- **Kinematic Models** of the ownship can be used to compute in real-time position and velocity of the ownship based on control actions provided by a pilot (either virtual pilot or human operator).
- **3D Scenario** visualizers can be used to show to the flight crew spatial information about the current flight scenario. This would mimic what the flight crew may see from the window of the cockpit. It can be used to create interactive simulations where the flight crew can control in real-time speed and direction of the ownship.

3 Architecture

The architecture of DANTi includes three main software components and two libraries (see Figure 3):

- **DANTi-Display** is the software component responsible for rendering the visual elements of the display. This component uses the DAA-Displays [5] library for creating the display elements.
- **DANTi-Logic** is the software component responsible for computing route conflicts and maneuver guidance. This component uses the DAIDALUS [6] library to perform the computations.

- **DANTi-Gateway** is the software component responsible for events routing between components and management of connections with external components.

3.1 DANTi-Display

DANTi-Display renders a flight display with the following visual elements:

- **Compass**. Display element indicating the heading of the ownship.
- **Interactive Map**. Navigation moving map showing the current position of the ownship in the airspace and nearby terrain, as well as traffic data. The ownship is represented as a blue chevron at the center of the map. Traffic data is presented using chevrons and a label presenting the altitude of the aircraft (relative to the ownship) and the vertical speed direction[1]. The direction of the chevron indicates the direction of the corresponding traffic aircraft. Soft buttons at the top of the map can be used to select different visualization options:
  - *North-Up*. When this option is selected, the upper side of the navigation moving map is the magnetic north and the chevron representing the owship rotates according to the current heading of the ownship.
  - *Call Sign*. When this option is selected, traffic labels in the navigation moving map show the tail number or flight number of the aircraft.
  - *Terrain*. When this option is selected, the navigation moving map displays a satellite image of the terrain.
  - *Contours*. When this option is selected, the navigation moving map shows areas of the airspace that should be avoided as they will lead to a loss of well clear. The areas resemble those produced by a weather radar service. A precise definition of contours is in the MOPS.
  - *Hazard Zones*. When this option is selected, the navigation moving map shows cylinder shaped areas defining the well clear volume of each aircraft. A precise definition of the hazard zones is in the MOPS. A mathematical definition of the well-clear volume is in [6].
- **Airspeed Tape**. Display element located on the left side of the screen. Shows the current airspeed of the ownship, in knots.
- **Altitude Tape**. Display element located on the right side of the screen. Shows the current altitude of the ownship, in feet.
- **Vertical Speed Tape**. Display element located on the right side of the screen. Shows the current vertical speed of the ownship.

The following chevrons can be used for traffic aircraft:

[1]No arrow indicates level flight; $\Uparrow$ indicates climbing; $\Downarrow$ indicates descending.

- **Nearby Traffic.** White chevron.
- **Alert Level 1 (Preventive Alert).** Circled yellow chevron.
- **Alert Level 2 (Corrective Alert).** Full yellow circle with inner chevron.
- **Warning.** Red chevron within a red square frame.

The following bands and indicators can be used to provide feedback on maneuver guidance:

- **Conflict Bands**. The bands are yellow. They represent warnings, and indicate maneuvers that can lead to loss of well clear for the ownship. For example, a yellow band from 12 to 26 degrees on the compass indicates a potential conflict for any heading in the range [12..26] degrees.
- **Resolution Bands**. These bands are dashed green. They indicate maneuvers that can be used by the pilot to recover from loss of well clear.
- **Resolution Indicators**. These indicators can be used in conjunction with resolution bands to provide additional feedback on preferred resolutions that optimize given resolution costs.

3.2 DANTi-Logic

DANTi-Logic provides the following DAA functions:

- **Conflict Detection**. Determines the current pairwise well clear status between the ownship and a traffic aircraft.
- **Traffic Alerting**. Determines pairwise alert level between the ownship and a traffic aircraft.
- **Maneuver Guidance**. Computes maneuver guidance for the ownship to maintain or regain a well clear status.

3.3 DANTi-Gateway

DANTi-Gateway provides the following functions:

- **Events/Data Routing** between internal software components.
- **Receive Traffic Data** from external data sources, e.g., ADSB equipment.
- **Send DAA Data** to external data consumers, e.g., a virtual pilot module.

4 Implementation

DANTi is implemented as an Electron[2] application. Electron is an open-source framework for creating cross-platform applications based on Web Technologies. Each Electron application has a *front-end component* dedicated to rendering, and a *back-end component* implementing the business logic of the application:

- The front-end component uses the APIs provided by Chromium[3], the open source version of Google's Chrome Web Browser, to render graphical windows and execute JavaScript code.
- The back-end component uses the APIs provided by NodeJS.[4], an event-based JavaScript runtime environment, to access a computer's native functions to create processes, read/write files and folders in the file system, and use input/output devices.

In the implementation of DANTi, the front-end component is DANTi-Display. The back-end component includes DANTi-Logic and DANTi-Gateway.

DANTi-Display is a Web Browser app implemented in TypeScript[5]. A class `DantiDisplay` defines the data structures and APIs necessary for rendering the display elements. Each display element is an instance of a widget from the DAA-Displays library. The main API functions are:

- `activate`. This function activates the connection with the gateway.
- `render`. This function allows to render/refresh the visual appearance of the display elements. The function takes one argument of type `DantiData`, which includes the following information:
  - *Ownship state*, including: aircraft ID, position (lat, lon, alt), 3D velocity vector, altitude, heading, airspeed, vertical speed, wind data;
  - *Traffic state*, including the following information for each traffic aircraft: aircraft ID, call-sign, position (lat, lon, alt) and 3D velocity vector of each traffic aircraft;
  - *Alerts*, including for each alert: alert level, alert region, daidalus alerter, and ID of the alerting aircraft;
  - *Conflict and resolution bands* for heading, horizontal speed, vertical speed, and altitude;
  - *Contours and hazard zones.* The shapes are represented using a set of points representing 2D polygons in space.

[2]`https://www.electronjs.org/`
[3]`https://www.chromium.org/`
[4]`https://nodejs.org/`
[5]`https://www.typescriptlang.org/`

DANTi-Logic is implemented in Java[6] and TypeScript. The Java code defines functions and data structures necessary to execute on-demand the DAA computations. The TypeScript code creates a NodeJS process that runs the Java code and keeps the Java code active in memory. Keeping the Java code active in memory is necessary for the correct use of DAIDALUS (version 2.0 or greater), as maneuver recommendations computed by the library depend on maneuver recommendations computed in previous time instants (hysteresis). The Java code of DANTi-Logic is now further described.

A Java class `DAABandsREPL` creates an interactive Read-Eval-Print Loop (REPL) for interacting with DAIDALUS: commands are entered at a command prompt, DAA computations are performed based on the entered command, and a result is returned at the command prompt. Two class variables, `onwship` and `traffic`, are used to store information on the current state of the ownship and traffic aircraft. The `traffic` table uses aircraft identifiers as keys. Storing the current state is necessary because DAIDALUS is memoryless and state information needs to be provided at each DAA computation. State information in the `traffic` table includes timestamps. A mechanism is implemented to purge *stale* aircraft states from the `traffic` table. An aircraft state is stale if its timestamp is older than `T` seconds with respect to the ownship's time (by default, `T` is 10 seconds).

The set of commands accepted by the REPL of DAABandsREPL are:

- `ownship`. This command allows to update the ownship state. The new state is provided as command argument. The ownship state is a string containing a list of comma-separated values. The default label and units of the elements of the list are: aircraft name, latitude (deg), longitude (deg), altitude (feet), 3D velocity vector (x: knot, y: knot, z: fpm), and time (sec). Different labels/units can be specified (see commands `labels` and `units`).

- `traffic`. This command allows to store a traffic state in the `traffic` table. The traffic state is provided as command argument, and is specified as a string containing a comma-separated list of values, as done for the `traffic` command.

- `labels`. This commands allows to specify the labels to be used to decode aircraft states. Labels are provided using a comma-separated list of values.

- `units`. This command allows to specify the units to be used to decode aircraft states. The units are provided as a comma-separated list of values.

- `wind`. This command allows to set wind information, including wind direction and magnitude. Wind information is provided using a JSON object with two fields, `deg`, indicating the direction in degrees, and `knot`, indicating the magnitude in knots. Default is no wind.

[6] `https://openjdk.java.net/`

- `compute-bands`. This command triggers the computation of bands, alerts and maneuver guidance. All DAA computations are performed using DAIDALUS. The default DAIDALUS configuration used in the computations is `DO_365B_no_SUM.conf`. A different configuration can be selected (see command `config`). The results of the computation is saved in file `REPL.json` contains ownship state, traffic state for each aircraft, alerts, conflicts and resolutions bands, contours, hazard zones, and maneuver recommendations.

- `config`. This command allows to select a configuration file for DAIDALUS.

- `precision`. This command sets floating point precision of the DAA computations. The default precision is 10 decimal digits.

- `stale`. This command sets the stale threshold used for identifying stale traffic information. The default threshold is 10 seconds.

- `show-table`. This command prints the current ownship's state and the current traffic states. A formatted output is produced where: the first line is the labels; the second line is the units; the third line is the ownship state; the following other lines are traffic aircraft states. An example output produced by the command is as follows:

```
NAME     lat      lon        alt      trk    gs      vs      time
[none]  [deg]    [deg]      [ft]     [deg]  [knot]  [fpm]    [s]
N416DJ, 39.5767, -104.8968, 7100.00, 11.00, 140,    0,       0
LYM970, 39.6753, -104.8798, 7782.14, 89.64, 147.50, -214.29, 0
N683SP, 39.6026, -104.8702, 6490.91, 0.550, 95.64,  272.73,  0
```

DANTi-Gateway is WebSocket server implemented in TypeScript. A class `DantiGateway` implements the API functions necessary to connect to the gateway and send/receive data and commands using a publish-subscribe mechanism. This architecture has been chosen to enable dynamic loading of components and reconfiguration of communication pathways between components. This can be used to introduce new specialized components for receiving data from specific data sources, e.g., read the ownship's state from ADSB equipment connected via bluetooth. It can also be used to send data to external data loggers, so data produced during interactive simulations can be saved and analyzed off-line, after the simulation run.

A data type `DantiRequest` defines the structure of the messages that can be sent to the gateway. Each messages includes three main fields: `id`, which is a unique message identifier; `type`, which indicates a message type; and `data`, an optional field used to specify message parameters. Message types currently supported are:

- `register-danti`: This message type is used by DANTi-Display to connect to the gateway. The command has one parameter, representing a unique identifier of the DANTi-Display instance to be connected to the gateway.

- `register-virtual-pilot`: This message type is used by virtual pilot modules to connect to the gateway. The command has one parameter, representing a unique identifier of the virtual pilot instance to be connected to the gateway.
- `render-request`: This message type can be used to send DAA data to DANTi-Display.
- `traffic`: This message type is used by receiver modules to communicate traffic state updates to DANTi-Logic.
- `ownship`: This message type is used by receiver modules to communicate ownship state updates to DANTi-Logic.
- `labels`: This message type is used by receiver modules to communicate label information to DANTi-Logic.
- `units`: This message type is used by receiver modules to communicate units information to DANTi-Logic.
- `epoch-end`: This message type is a synchronization event provided by external data sources to signal that all current traffic data have been provided and DAA computations can be triggered in DANTi-Logic.

4.1 Connection with X-Plane / Flyer

The connection with X-Plane / Flyer is realized using a micro-server implemented in NodeJS. The micro-server uses WebSockets to exchange data and commands with the DANTi-Gateway, and the X-Plane Connect[7] library to communicate with X-Plane / Flyer.

5 Example

An example is now illustrated where DANTi receives pre-recorded flight data from an external data provider. The data provider is implemented in NodeJS, and communicates with DANTi through a WebSocket connection.

The flight scenario includes three aircraft: the ownship (LYM970) and two traffic aircraft (N683SP and N416DJ).
The sequence of interactions between the data source and DANTi are as follows (see Figure 4):

- Data source:
    - Open a WebSocket connection with DANTi.
    - Send labels and units to DANTi, using the `labels` and `units` messages, respectively.

[7] `https://github.com/nasa/XPlaneConnect`

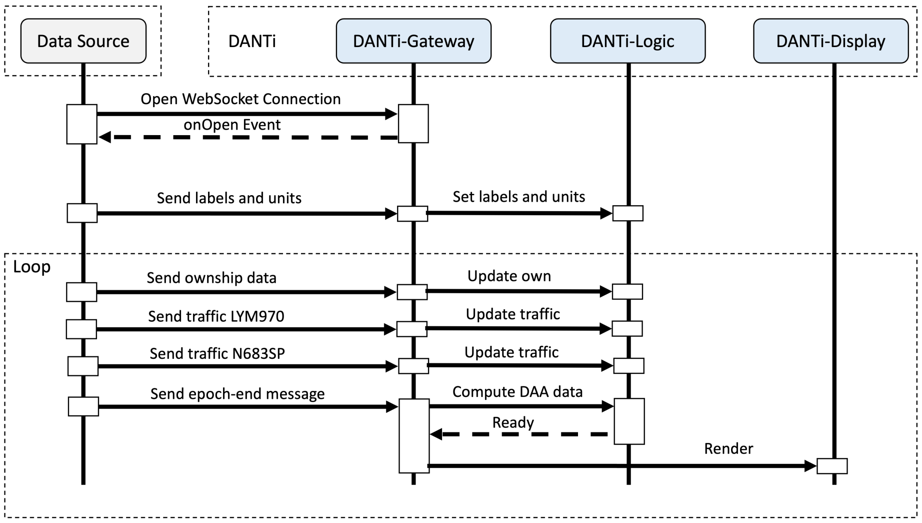

Figure 4: Sequence diagram representing the interactions between data source and DANTi during the test with pre-recorded flight data.

  - Periodically send traffic information to DANTi, using the `ownship` and `traffic` message types, respectively. Whenever the timestamp of the ownship changes, send an `epoch-end` message to DANTi.

- DANTi:
  - Upon receiving `labels` and `units` messages, set labels and units according to the values specified in the messages.
  - Upon receiving `ownship`, update ownship state.
  - Upon receiving `traffic`, update traffic table.
  - Upon receiving an `epoch-end` message, perform the following actions:
    * Compute DAA data, including: bands, alerts and maneuver recommendations.
    * Refresh the display, by rendering traffic data and the computed DAA data.

References

[1] Víctor Carreño. Evaluation, analysis and results of the DANTi flight test data, the DAIDALUS detect and avoid algorithm, and the DANTi concept for detect and avoid in the cockpit. Contractor Report NASA/CR-20205004594, NASA, Langley Research Center, Hampton VA 23681-2199, USA, August 2020.

[2] Víctor Carreño, María Consiglio, and César Muñoz. Analysis and preliminary results of a concept for detect and avoid in the cockpit. In *Proceedings of the 38th Digital Avionics Systems Conference (DASC 2019)*, San Diego, CA, US, September 2019.

[3] James P. Chamberlain, Maria C. Consiglio, and César A. Muñoz. DANTi: Detect and Avoid in The Cockpit. In *17th AIAA Aviation Technology, Integration, and Operations Conference*, 2017.

[4] International Civil Aviation Organization (ICAO). Rules of the air – annex 2 to the convention on international civil aviation, 2005.

[5] Paolo Masci and César A. Muñoz. A Graphical Toolkit for the Validation of Requirements for Detect and Avoid Systems. In Wolfgang Ahrendt and Heike Wehrheim, editors, *Tests and Proofs*, pages 155–166, Cham, 2020. Springer International Publishing.

[6] César Muñoz, Anthony Narkawicz, George Hagen, Jason Upchurch, Aaron Dutle, María Consiglio, and James Chamberlain. Daidalus: detect and avoid alerting logic for unmanned systems. In *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*, pages 5A1–1. IEEE, 2015.

[7] Federal Aviation Administration (FAA) U.S. Department of Transportation. Faa advisory circular ac120-76d: Authorization for use of electronic flight bags document information. https://www.faa.gov/documentlibrary/media/advisory_circular/ac_120-76d.pdf, 2017.