Integrating X-Plane

- Communicating with X-Plane

LTH School of Engineering at Campus Helsingborg
Computer Engineering

Bachelor thesis:
Ricky Djerf
Marcus Hammar

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

Printed in Sweden
Media-Tryck
Biblioteksdirektionen
Lunds universitet
Lund 2012

Abstract

Saab Training Systems develop training systems for civil and military use. The company desires a possibility to connect the flight simulator X-Plane 10 to other simulators with the use of their existing integration platform, WISE.

X-Plane has a built in plug-in manager which makes it possible to use third-party plug-ins for the simulator. There is a free SDK available for writing plug-ins. This SDK is not provided by Laminar Research, the company who develops X-Plane, but two persons working on it as an extra project. Plug-ins for X-Plane are coded using C or C++.

This report describes how information in X-Plane is accessed by a plug-in and how the developed plug-in works. The plug-in is able to extract and inject data with the use of X-Plane’s existing data references. The data references and functions for data access are declared in the existing X-Plane SDK. This report also describes how the plug-in handles communication via UDP socket to send and receive data to and from WISE and some of the limitations and possibilities with X-Plane from an integration point of view.

The plug-in is able to send data such as aircraft type, velocity, orientation, landing gear status etc. for the user’s aircraft. It is also able to receive data of the same type, seize control over an AI aircraft and apply the data on it. The plug-in uses a simple form of dead reckoning to make the aircraft drawing look smoother.

Keywords: X-Plane, Flight simulator, SDK, Plug-in

Sammanfattning

Saab Training Systems utvecklar träningssystem för militärt och civilt bruk. Företaget vill veta vilka möjligheter det finns med flygsimulatorn X-Plane 10 samt koppla ihop flygsimulatorn med andra simulatorer via deras existerande integrationsplatform, WISE.

X-Plane har ett inbyggt stöd för plugins vilket gör det möjligt att utveckla egna plugins. Det finns ett tillgängligt SDK för utveckling av plugins. SDK:et tillhandahålls inte av Laminar Research, utvecklarna av X-Plane, utan utvecklas som ett eget projekt utav två personer. Plugins skrivs i C eller C++ kod.

Rapporten beskriver hur information från X-Plane hämtas ut via plugin och hur ett plugin skrivs. Det utvecklade pluginet är kapabelt att extrahera samt injecera data med hjälp av data referenser ifrån X-Plane. Data referenserna och funktionerna för data tillgång är deklarerade i SDK:et. Rapporten beskriver även hur pluginet, via UDP socket, hanterar kommunikationen mellan pluginet och WISE. Möjligheter och begränsningar med integrationen i fokus tas också upp.

Pluginet kan skicka data så som flygplanstyp, hastighet, position och status för landningsställ för användarens flygplan. Pluginet kan även ta emot denna data, ta kontroll över ett AI kontrollerat flygplan och tillämpa den mottagna datan på detta. Det finns även en enklare form av dödräkning implementerad, detta för att flygplanet skall ritas upp på ett jämnare sätt.

Nyckelord: X-Plane, flygsimulator, SDK, plugin

Foreword

This report and the software developed is the result of a project initiated by Saab Training Systems, Helsingborg.

We would like to give our special thanks to the following:
Sandy Barbour for answering forum threads and developing the SDK,
Fredrik Ullner for support and supervising at Saab Training System,
Mats Lilja for support regarding the report,
Saab Training System for the opportunity to work in our desired field.

Helsingborg June 2012

Ricky Djerf och Marcus Hammar

List of contents

1 Introduction

SAAB Training Systems [sts] is a part of the SAAB group and develops training systems for civil and military use. The company has a client who desires a possibility to connect X-Plane with other simulators and interact between them. SAAB Training Systems provides an integration platform, WISE, which can be used for communication between the simulators. Currently they have no way of extracting information from X-Plane.

X-Plane is a commercial simulator with several versions such as a time-limited Demo version and a professional version, where the user may obtain actual flight hours [xplane info].

The assignment regards extracting and injecting information, such as position and orientation, from and to the X-Plane simulator by communicating with the integration platform. This will be made possible by writing a plug-in for X-Plane, which handles information input and extraction, and writing a WISE driver.

1.1 Questions to answer

The questions that this project will answer are

- What are the possibilities with X-Plane?
    - Is it possible to draw ground objects in X-Plane and if so how to do it?
    - Does X-Plane support weapons and if so how to access weapon data?
- How to develop a basic plug-in for X-Plane?
    - How to access and modify aircraft data?
    - What are the necessary functions in a plug-in?
- How to handle plug-ins in X-Plane?
    - How to install a plug-in in X-Plane?
    - How to remove a plug-in from X-Plane?
- How to get the plug-in to communicate with a server?
- What are the limitations in the X-Plane simulator?

1.2 Project delimitations

The development during this project will focus on aircraft information extraction and injection, information concerning other objects in X-Plane are outside the development scope. The main focus of the plug-in will be communication with the integration server and decent display of other controlled aircrafts. The display of other controlled aircrafts is possible with the information sent via the integration server, at a set interval, and dead reckoning calculations within the plug-in [dead reckoning].

1.3 X-Plane 10

X-Plane 10 aims to be “the world’s most comprehensive and powerful flight simulator for personal computers” [xplane] and ”offers the most realistic flight model available.”. X-Plane 10 is developed and owned by Austin Meyer and Laminar Research. Once bought it includes the flight simulator, plane maker, to make own aircrafts, and airfoil maker, all of which is runnable in Windows, Mac and Linux environments.

X-Plane 10 is a relatively new flight-simulator released December 2011. It has a lot of similarities with its predecessor X-Plane 9 but not all the information in the communities is applicable in X-Plane 10. X-Plane 10 has a completely renewed air traffic control system called ATC which controls all the aircrafts in the system including the users [xplane news]. There are also some data references that are removed and some that are added [datarefs].

2 Method

A SCRUM [scrum] inspired development was intended to be used in a way that sprints and backlog were utilized. This was to create a better structure in the work and to be able to make more accurate time estimations of the work required to meet deadlines. As the project evolved and the knowledge of possibilities of X-Plane expanded the SCRUM inspired development seemed superfluous and was therefore not used. Instead of SCRUM the development turned towards daily meetings to decide what was going to be developed the next day. Development also made use of post-it notes which worked as a backlog and stated what functionality was needed and when it should be completed. The documentation of the project proceeded with Code Commenting and comments on problems in a simple text-file for helping with report writing later on.

Development of the plug-in and analysis/evaluation of X-Plane have proceeded simultaneously. This came natural as questions of functionality and operations came up during development and it would be hard to cover it all in the limited time available for testing X-Plane before the development began. Both the plug-in for X-Plane and the driver for WISE [see Chapter 2.1.3] are coded in C++.

The first tests of the simulator occurred at home using the X-Plane demo version, to get a feel for the different functions available in the simulator and how the simulator worked. Development of the plug-in started after the first tests were completed and workplace was set to Saab Training System's facilities. Some testing was done at STS as the development process proceeded.

The concept of the work model narrows down to: analysis of problem, viable solution to the problem by researching, implementing solution and documenting code and encountered problems.

Before the project started, this project was divided into three stages:

- Research and testing of X-Plane
- Extracting and injecting data with X-Plane
- Communication with WISE

2.1 Research

2.1.1 X-Plane

As X-Plane provides a Demo version it was solely to install and test X-Plane on a personal computer. The main goal with this phase was to try out the possibilities with X-Plane simulation and to gain some understanding of the simulator. To do this a checklist [see Appendix 8.1 and 8.2] was established for structural testing. The checklist concerned matters like “is it possible for a plane to collide with a car or a building?” and “Are there cars and if so, are they moving?”. These tests were made to get a better understanding of how the objects work, interact with each other and if there’s a possibility to extract information from ground objects.

2.1.2 SDK

X-Plane SDK [sdk] was originally developed by Sandy Barbour and Ben Supnik as volunteers and is still maintained. As the SDK is made as a spare-time project there are no obligations to improve the SDK but the community helps each other and the developers by forum or email contact.

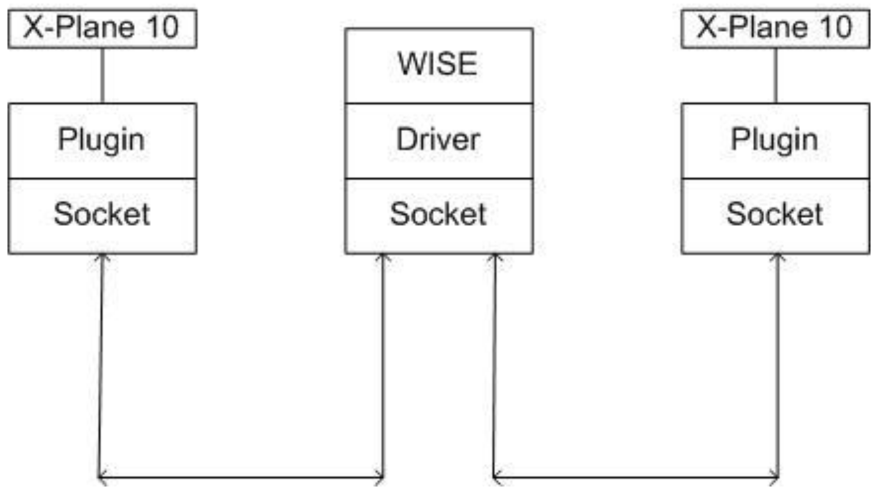
Research concerning the existing X-Plane SDK was also made by going through the different functions available and isolating the ones that might be useful for this project. There are existing code samples in the SDK folder which proved useful when trying to understand the structure of the SDK.

Plug-ins for X-Plane are coded in C but the SDK support C++ through “extern C” [extern].

An update [sdk21] for the SDK was released during development and provided some needed functionality and some possibilities to expand the functionality of the plug-in further. Functionality of Flightloopcallbacks [see Chapter 4.3] was expanded and data references for damage were included, providing support for firing at aircraft and registration of hits.

2.1.3 Driver

The driver is a component with the task of sending, receiving and converting data to/from a data model. To get a better understanding of how the driver should be coded a two day education was given by STS. This education gave an introduction in how to write a driver which is working as a middle man between the integration platform and the simulator. The education also provided information on how simulators, drivers and servers are working together to achieve the integration, see picture ***Overview***.



***Picture Overview**: Shows a system overview of an integration between two instances of X-Plane 10*

2.2 Extracting data from X-Plane

By using a plug-in to X-Plane it is possible to extract data values from the simulator. The developers of the SDK provide simple examples [sample code] to get anyone going with plug-in development in X-Plane.

The SDK provides functions which uses data references to access data within X-Plane, there are also functions which are able to write data to X-Plane using the same data references. Writing data does not apply to all data references though, only the ones which are writable [datarefs].

2.3 Communication

The chosen path to communicate with WISE came down to socket communication since it was the only way known to the project members and it is a valid way to communicate in networks. Socket communication is an easy way to decide where to send data and where to receive data. There are several types of sockets available for implementation. The types of sockets considered in this project were c-sockets[c-socket], an implementation of Berkeley sockets [berkeley-socket] called simple-socket [simple socket] and the standard windows socket winsock [winsock].

2.4 Source criticism

The references were chosen as the SDK used is developed and maintained by the administrators of XSquawkBox. The forums used are often replied by the developers of the SDK or people that have worked with the SDK for a long time and have provided valuable feedback to the developers. It should be noted though that when it comes to the decisions of not choosing X-Planes built in way of communicating via UDP it was made only by reading forum [forum] threads on the X-Plane forums, this was due to the time limit of the project.

The fact that decision was based on forum threads means that these things have not been tested in the project and may in fact work even if people on forums said that they did not. Other references than the ones concerning X-Plane were chosen either because they answer a problem in a way that the project members found satisfying, because they are written by known companies/organizations or because the documents were written by the creators of a certain code for example simplesocket created by carrierlabs.

3 Analysis of X-Plane and the SDK

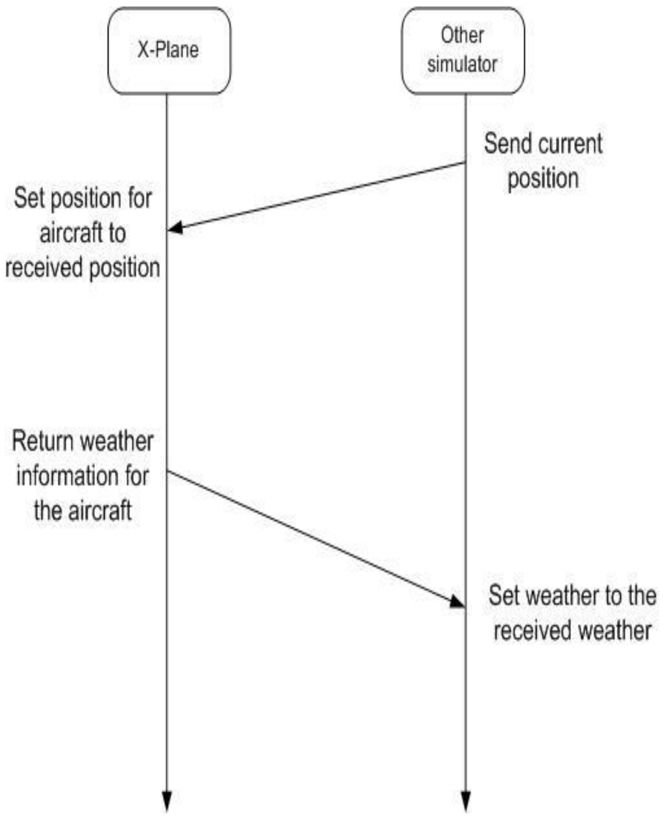
This chapter contains information apprehended concerning the possibilities with X-Plane and functionality of the SDK.

3.1 X-Plane and its possibilities

As mentioned in the introduction X-Plane is a very versatile flight simulator. It can both entertain and educate the user in the various aspects of flying. The simulator includes special scenarios such as entering earth's atmosphere in a space shuttle, landing on an aircraft carrier and simulating air to air combat. Basic flying in X-Plane is fairly easy even with a mouse as controller. The user starts on the runway of a chosen airfield and only needs to apply throttle, loosen the brakes and use the mouse to control the rudders. There is also a more advanced dimension to flying in X-Plane. The user may start with all systems offline and will then need to complete the entire start sequence manually including taxiing and radio communication.

X-Plane does not need a super powerful computer to run. As long as graphics and draw-distance is not an issue, a 256 MB graphics memory card and 3 GB of RAM is sufficient. If the user wants a good experience though the system needs better specifications, at least a 3 GHz dual-core processor, 4 GB RAM and a 1 GB graphics memory card.

There are a lot of possibilities created with X-Plane since it supports network play and third party plug-ins. X-Plane has a lot of data which is available for access and modification, not only for aircrafts but for missiles, bombs, weather and time. This opens up the possibilities with X-Plane, theoretically X-Plane could be used to reproduce weather in a second simulator. This could be done by completing the steps shown in picture ***Weather***.


X-Plane
Other simulator
Send current position
Set position for aircraft to received position
Return weather information for the aircraft
Set weather to the received weather

Picture Weather*: shows a possible way to use X-Plane as a weather controller*

Since data is available for weapons, it should also be possible to shoot missiles and bombs on ground targets in other simulators by extracting the weapon data in X-Plane and cross-referencing it with object data received from another simulator which has ground-units. This has not been covered in the testing process but should not differ much from the aircraft information transfer.

It is possible to draw ground objects such as people and vehicles in X-Plane as long as the objects are valid X-Plane object files. Valid X-Plane objects are .obj files with references to a .png texture map. An .obj file is a 3D object file which can be created in for example Autodesk 3D studio [object files].

There are already a lot of third party plug-ins available. The most renowned is the existing XSquawkBox plug-in [xsquawkbox] which allows the user to connect the simulator to a simulated traffic control network. By running the plug-in the user gets live traffic control information from real persons who are using the VATSIM [vatsim] air traffic control simulator. X-Plane users receive the traffic information whilst flying online with other real persons, the network contains players running both X-Plane and Microsoft flight. There has not been enough time to test the XSquawkBox plug-in so what data is sent and received by the plug-in is currently unknown, it is also unknown how other aircrafts are represented in the plug-in.

The simulator shows elevation in feet but when retrieving data using the SDK the values returned are in meters. The elevation in X-Plane is calculated by the difference between the mass point of the aircraft and the mean sea level, MSL.

The maps in X-Plane appear to be geotypical of the real world but the detail level differs depending on where in the world the aircraft is located. Originally only the airport in Seattle has buildings on it but there are other airports available for download. A possibility to create own airports also exists with the use of WorldEditor program called WED[wed].There are new third party textures available for purchase but not for all airports. Cities are never exact copies of the real ones but the user notices buildings on the positions where cities are located.

If all maps are not installed X-Plane will warn the user when flying outside the installed maps. After the warning all terrain shown will be water until the user reaches a part of the world where a map is installed.

Weapons are own objects in X-Plane, there are several types of weapons in the simulator including: guns, rockets, missiles and bombs. There are data references similar to the ones for the aircraft which access information concerning the weapons for the user’s aircraft. The data references are able to provide information concerning the weapon type, weapon location, fire rate for the gun and if the weapon is in the aircraft, free flying or demolished.

One aircraft may carry a maximum of 25 weapon units (missiles or bombs) and the weapons included in the aircraft are specified in the aircraft file. This means that if the creator of an aircraft has not included bombs in his aircraft, the aircraft will not be able to drop bombs even if it is designed as a bomber. The amount of data accessible for guns is very limited and the information available concerns the firing rate and bullets.

3.2 X-Plane data access via SDK

The SDK contains a set of functions called **XPLMDataAccess**. By using this function combined with data references, will henceforth be called DataRefs. A lot of useful information is available when developing a plug-in. DataRefs are used to access and modify data in X-Plane. There is a lot of data available for access such as aircraft: location, speed, heading etc. DataRefs also provide the possibility to control the airplane remotely using the plug-in. To extract data from X-Plane the functions must be called i.e. the plug-in must “ask” X-Plane for information at every update. X-Plane does not send the information to the plug-in.

The data access functions are fairly easy to understand and may look as follows: XPLMGetDataf(“data reference”), the function name can be broken down to three pieces:
1. XPLM which only states that it is the X-Plane library
2. Get which declares that you want to read data
3. Dataf which declares that the reference contains a float value

All data access functions are built this way so XPLMSetDataf(“data reference”, number) assigns the number to the specified data reference as long as the reference is modifiable which is far from all references.
An interesting example of non modifiable data references is longitude and latitude for airplanes.

If the longitude and latitude coordinates for the aircraft are to be modified the user has to call the function XPLMWorldToLocal(lat, lon, alt, &x, &y, &z) so the coordinates can be modified into OpenGL coordinates(x, y, z) and then set.

If the data reference to be accessed contains an array of integers XPLMGetDatavi is used where the “Datavi” declares that a vector of integers is returned and/or expected.

To find a data reference the function XPLMFindDataRef(“reference name”) is used. There are approximately 3700 data references available, these references contain everything from weather and position of aircrafts to certain buttons in the cockpit. Data references are accessed by search-paths to the specific reference. An example of this is “sim/flightmodel/position/latitude” which gives access to the latitude position for the user aircraft. The search-path can also be broken down to smaller pieces:

1. Sim clarifies that it is a simulator reference and is always used.
2. Flightmodel is the reference for user aircraft.
3. Position states that the reference has something to do with the aircrafts orientation, position, speed, angles etc.
4. Latitude is the specified instance of position.

All data references are built in the same way which makes it easier to obtain the wanted information. For access to the heading of multiplayer aircraft 1 the reference “sim/multiplayer/position/plane1_psi” is used, an important notice here is that there is a maximum of 20 aircrafts in X-Plane. Aircraft number zero is the user aircraft followed by numbers 1-19 which are AI aircrafts. This means that flightmodel references must be used instead of multiplayer references when controlling aircraft number zero.

Not all data references have as long search-paths as the ones for position. An example of this is the data references for weather which may look like “sim/weather/sigma” which contains “the atmospheric density as a ratio compared to sea level”.

X-Plane has a lot of settings for weather, the user may customize the weather to his liking, set random weather, paint weather with the mouse, download real weather from the internet or inherit weather from a master-machine. All these functions are not available remotely but weather can be customized by a plug-in which changes data-references for weather.

When manipulating data references on AI-planes it is important to notice that it is only DataRefs in “sim/multiplayer/position” that modify the outside of the plane, for example “sim/multiplayer/position/plane1_flap_ratio” sets the flap-ratio on AI-plane one but “sim/multiplayer/controls/flap_request” only modifies the lever for the flaps inside the aircraft.

4 Development

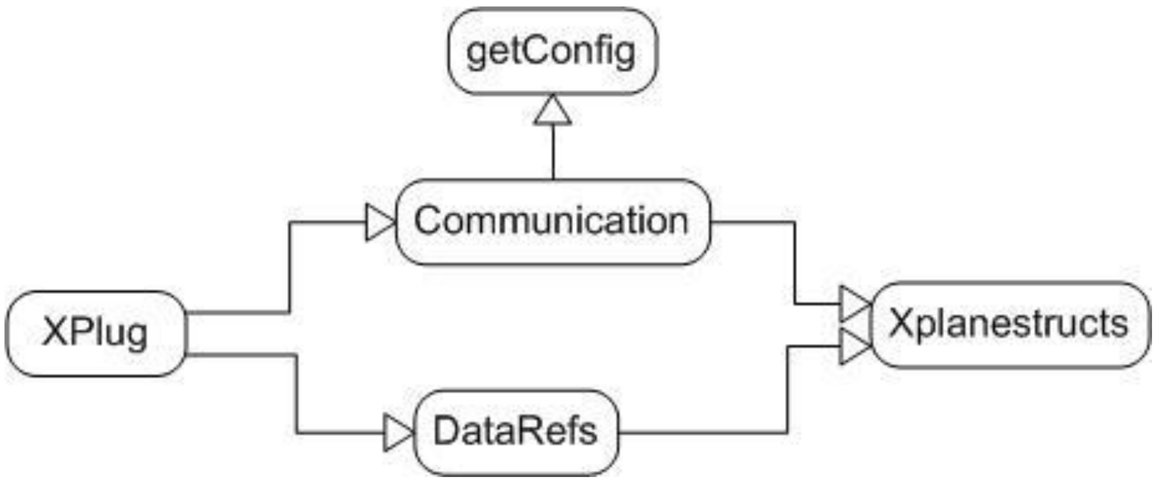
X-Plane is very compatible with plug-ins as it has a built-in function for deploying plug-ins into the game. Plug-in deployment is done with three easy steps:

1. Set the project configuration to "Dynamic Library"
2. Change "target extension" to .xpl
3. Copy the .xpl file from the build directory and paste it in the X-Plane plugins folder, X-Plane 10\Resources\plugins\.

X-Plane does not allow plug-in to be added or removed while the simulator is running. When X-Plane is started the plug-in will be loaded as any other X-Plane resource file, it will also be initialized automatically on start-up. It might take some extra time before the plug-in starts though.

A plug-in for X-Plane must contain five functions if it is supposed to work: `XPluginStart`, `XPluginStop`, `XPluginEnable`, `XpluginDisable` and `XPluginReceiveMessage` [see Appendix 9.3].

The structure of the plug-in is as follows, show in Picture ***Structure*** :
XPlug – Standard X-Plane plug-in, calls functions
Communication – Handles socket communication
DataRefs – Handles everything concerning the data references, setting and getting values to and from X-Plane
getConfig – Handles the configuration file
XPlaneStructs – contains all data structs used by the plug-in and driver



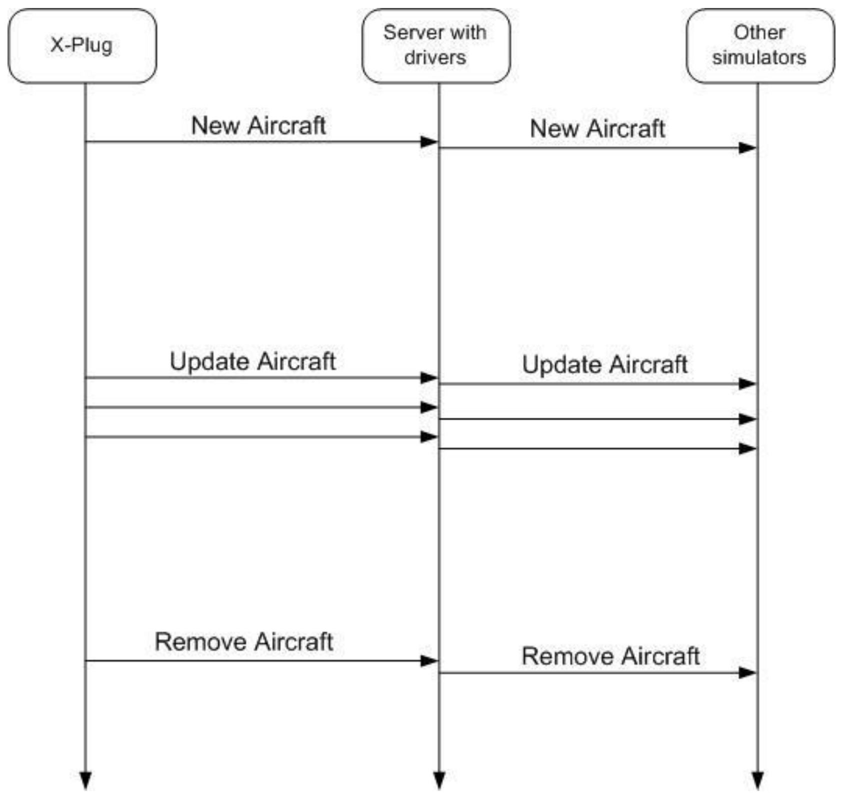
Picture Structure*: How the plug-in is structured with classes.*

The development of data extraction and injection proceeded smoother than first expected. The implementation of communication between plug-in and driver with extraction/injection was not as simple. Much time was spent learning to understand and implement socket communication as well as testing the communication with simulator-to-simulator.

Adding WISE in the mix complicated the development as the driver was developed during the plug-in development and uses different syntaxes. The functionality of the driver and plug-in communication is the same but the implementation is slightly different. As the project members have no prior experience with WISE the introduction course given at STS complemented the documentation and eased the driver development. Still minor mistakes have been made and required more time than estimated.

Several different implementations of socket communication [see Chapter 2.3] were tried and evaluated based on the amount of code, inclusions and linking needed. In this project Windows was used as platform, this contributed to the decision to choose standard Windows socket communication as it was well documented and could be implemented.

4.1 Plug-in and driver sequence



***Picture Message Sequence**: Shows the message sequence for X-Plug*

The message sequence begins with the plug-in, called X-Plug, sending a newAircraftMessage when the simulator has been fully initialized. The driver receives the message and passes it on to the server which stores the information. A newAircraftMessage is then sent from the server to the drivers of other simulators connected.

After the plug-in has sent its newAircraftMessage it immediately starts to update the information by sending updateAircraftMessages, these messages are sent periodically at a specified interval, default setting is once per second. The plug-in continues to send these messages until the simulator is shut down or the plug-in is disabled.

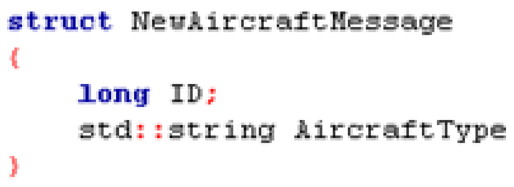
When the simulator is shut down or the plug-in is disabled data needs to be removed from the server, this means that a removeAircraftMessage is sent which erases the references for the object in both the drivers and the server.

Illustration see Picture ***Message Sequence***.

4.2 Structuring information

When sending information to and from the plug-in the structs within XPlaneStructs are used to structure the data. This simplifies the communication by sending the message by a specified structure and receiving the message by the same structure. Which structure to use is specified in the first bytes sent, which contains the header with message type. The different message types makes it easier for the plug-in/driver to process the message as each message type needs to be processed in their own way.

A new aircraft message only adds the id and the search-path, within X-Plane 10 folder, to the integration server. This makes the driver send a new aircraft message to other simulators plugged into WISE. A new aircraft message looks as shown in Picture ***new aircraft message***.



```
struct NewAircraftMessage
{
    long ID;
    std::string AircraftType
}
```

Picture new aircraft message: Content of a new aircraft message.

The update message contains all position data of the aircraft such as coordinates, elevation, heading, pitch, roll and also the velocity and acceleration of the aircraft. Some minor details such as landing gear, flaps and speedbrake are also included. The velocity and acceleration is used for the dead reckoning calculations, basically using the velocity and acceleration to determine where the aircraft should be at the next update. This makes the aircraft appear to fly smoothly until the next position update, when the aircraft needs to be positioned correctly, to reduce stuttering.

4.3 Communication

Communication with WISE is made with the plug-in for X-Plane. The plug-in has a socket which communicates with a driver on the integration server. The driver translates information and handles communication in both directions. The plug-in socket uses a send-function which runs periodically so that the information flow is constant.

The plug-in sends an information package at a set interval, modifiable in the configuration file. The configuration file also contains the information of which port to connect on, IP address of receiver, flight id and dead reckoning rate.

The implementation in the plug-in is Server-Client communication, the plug-in acting server and the integration server acting client. At first TCP was used for communication, requiring a connection to be established before communicating. This would freeze X-Plane as the plug-in awaited connection from the integration server. By threading the receive process this problem should be averted though this has not been tested due to the projects time constraints.

To avoid the lock of X-Plane at every start-up and enable of the plug-in, UDP was chosen instead. UDP only need to know which socket port to listen to and which IP address to look for. This means that X-Plane could start with the plug-in, without the need to establish a connection, while still being able to receive messages once started.

When both the plug-in and the integration server have started they are able to send and receive data to and from each other. The plug-in will listen for data until the plug-in is disabled, either through the plug-in administrator or by shutting down X-Plane.

The concept of socket communication is easy to grasp and once you understand it, not too complicated to implement. If one has never programmed network communication before it can take some hours to understand and build a socket that is able to establish a valid connection.

X-Plane has a built-in way of communicating via UDP-socket. It was decided not to use the built-in UDP communication [see Chapter 4.10].

Before implementing the socket communication in the plug-in a short server- and client-program was coded and compiled. With these programs the socket properties were observed and the structs were tested to make sure everything was working correctly.The server sent a message to the client with fictive aircraft data. The client receives the message. This test was meant to make sure the socket communication worked as planned and the structs were correctly implemented. By displaying the message on both client and server, by printing the values on screen, it showed that while the message was successfully sent and received, the information did not look the same at both ends. This proved to be a handling error as the structs were of different versions. Correcting the mistake showed that, while using same version of the structs, the aircraft information got through in a correct way.

After testing the server and client the socket communication code was implemented in the plug-in. Another test was made to measure the send and receive limits of the plug-in. To test this, the client from before was modified to receive and bounce the message with aircraft information. At first the send- and receive functions was put in respective DrawCallback, which are called each frame. This was done to see if the plug-in could send and also receive as often as every frame. This hurt the performance of the simulation and only worked “well” for one aircraft.

It was decided to send information at an editable rate [see Chapter 4.10].

This is where FlightLoopCallback, as previously mentioned, comes into the development. FlightLoopCallback is a periodic callback with a modifiable value and had added functionality with the SDK update. FlightLoopCallback is used for scheduling periodic tasks, such as sending information. The return value of the callback is the number of seconds until the callback will be called again. If the return value is negative then the function will be called after the absolute value number of X-Plane cycles. This value can be edited in the configuration file supplied with the plug-in.

The receive function is still in a DrawCallback as the integration server could send information at any moment and the plug-in need to process the information as soon as possible. By setting the socket option to non-blocking it is possible to receive and send at the same time. If the socket is set to blocking, the plug-in would lock up when it is trying to receive data. As data is not sent very often the plug-in would lock the entire simulator, thus the socket is set to non-blocking. No major performance dips due to the receive-Callback have been noticed during testing.

With every update the plug-in receives information about position, velocity and acceleration of the aircraft as well as speedbrakes, landing gear and flaps. The velocity and acceleration is used in the dead reckoning calculations to make the aircraft move smoothly and minimizes the jump needed when updates arrive with new position information.

As the plug-in can not be feed a constant stream of updates dead reckoning was implemented and brought some problems. The first tests did not include acceleration in the calculation as the information was not yet implemented in the messages. At low speeds the aircraft jumps forwards at updates and at high speed the aircraft is calculated to fly too far and is therefore pulled backwards at updates.

After implementing the acceleration the representation of another aircraft is good at a distance. When watching another aircraft closely or following it small jump can be seen when updates are received.

It is possible to get a better simulation of the received aircraft by implementing interpolation or two-step Adams-Bashforth Integration [bashforth]. These functions are much more mathematically advanced than the current dead reckoning implementation and they require extra variables to save the last position. These methods were therefore not considered in this project due to the projects time limit.

4.4 Dead reckoning

Dead reckoning was first implemented with a DrawCallback seeing as drawing every frame would be desirable. The dead reckoning implementation uses the current position, velocity and acceleration to determine where the aircraft should be at the next update. This however made the simulated aircraft move faster than it should and was exponentially increased as the aircraft moved faster. At high flight speed the simulated aircraft would position itself at the correct position as the update was received and then “slingshot” forward. This is most likely something with the DrawCallback being called more often than every frame.

A FlightLoopCallback was used instead of the DrawCallback as it is possible to set the period of the interval at which the FlightLoopCallback is executed. The period is not guaranteed but it will come very close. By setting the period close to frame rate or a bit higher the simulated model will look smooth and will not slow down the simulator more than needed.

The formula for dead reckoning, used in the plug-in, is as follows [drcalculation]:

$$Position = Position + Velocity * time + (Acceleration * time^2)/2$$

$$Velocity = Velocity + Acceleration * time$$

The position, velocity and acceleration are updated with every update message. Acceleration is not stored internally in the plug-in as position and velocity are, it is only used in the dead reckoning calculation.

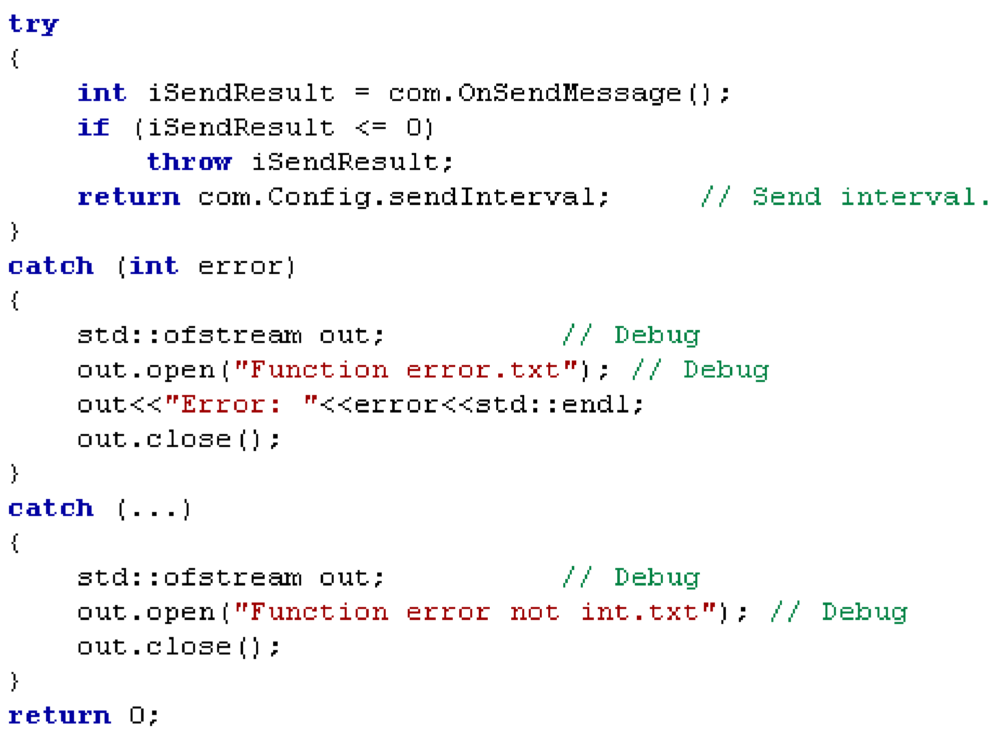
4.5 Debug

Debugging the plug-in with “Attach to process” via Visual Studio 2010 did not yield the expected results since even when the plug-in crashes it is shown that X-Plane.exe has crashed as it was the process running the plug-in.

To be able to debug and find flaws in the plug-in a simple debug method was implemented by surrounding the troubling code parts with try-catch blocks [trycatch], see Picture ***Debug***. The try block would check most functions and validate their return-value. If this value was out-of-bound or unexpected, an error was thrown and the catch block would create a file and write the error in the file. This way the errors could be checked even if X-Plane crashed. The error codes were noted in a separate document for easy follow-up.

The same process was used for debugging and checking the communication of the plug-in.

Example: When sending the aircraft type, the search-path within X-Plane 10 folder was specified. By printing the path being sent and the path being received it was possible to determine if the problem was located in the plug-in or the driver.



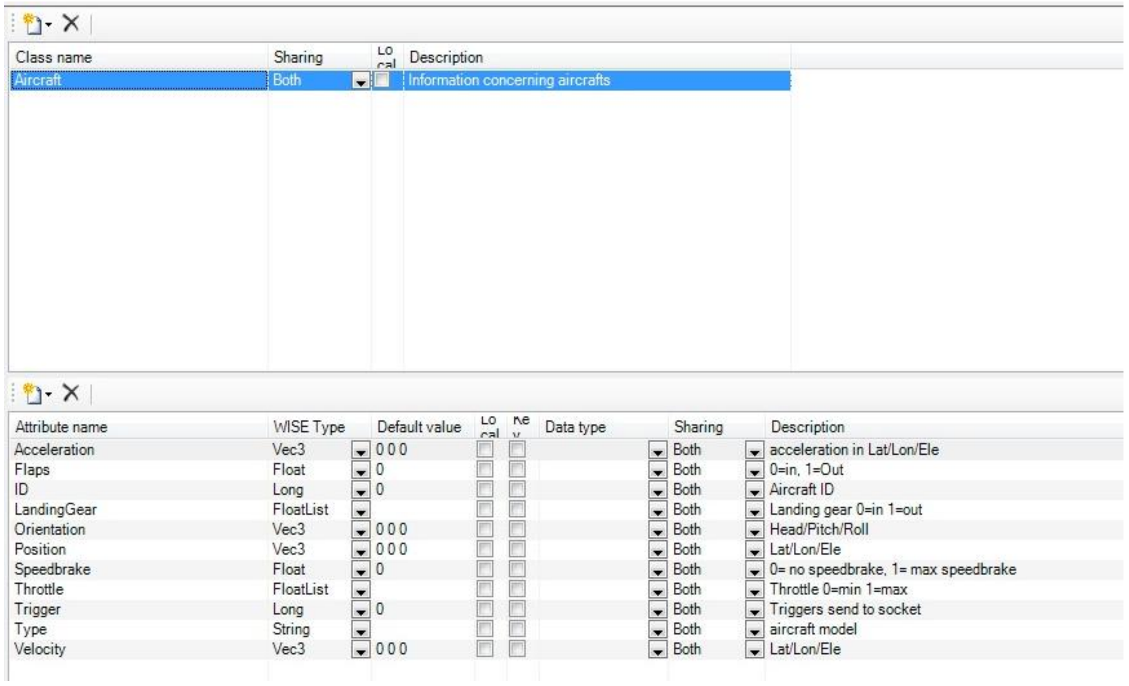
```
try
{
    int iSendResult = com.OnSendMessage();
    if (iSendResult <= 0)
        throw iSendResult;
    return com.Config.sendInterval;     // Send interval.
}
catch (int error)
{
    std::ofstream out;          // Debug
    out.open("Function error.txt"); // Debug
    out<<"Error: "<<error<<std::endl;
    out.close();
}
catch (...)
{
    std::ofstream out;          // Debug
    out.open("Function error not int.txt"); // Debug
    out.close();
}
return 0;
```

Picture Debug *: Example of try-catch used for debuging.*

4.6 Driver implementation

All data sent from and to the plug-in is handled by a driver. The driver uses STS own classes and because of the cross platform compatibility some data needs to be converted. This is due to X-Plane 10 using char and string while WISE uses wchar and wstring. By simply converting the information either when sent to WISE or when received in WISE the format issue is solved.

The driver uses a template database model containing all concerned attributes. The driver is able to receive data concerning attributes that are not mapped in the template database, these attributes will however not be stored. In order for WISE to receive data from the plug-in a template database containing all attributes sent from the plug-in was created for the driver. The database is shown in picture ***Database***.



Class name	Sharing	Lo cal	Description
Aircraft	Both	☐	Information concerning aircrafts

Attribute name	WISE Type	Default value	Lo cal	Ke y	Data type	Sharing	Description
Acceleration	Vec3	0 0 0	☐	☐		Both	acceleration in Lat/Lon/Ele
Flaps	Float	0	☐	☐		Both	0=in, 1=Out
ID	Long	0	☐	☐		Both	Aircraft ID
LandingGear	FloatList		☐	☐		Both	Landing gear 0=in 1=out
Orientation	Vec3	0 0 0	☐	☐		Both	Head/Pitch/Roll
Position	Vec3	0 0 0	☐	☐		Both	Lat/Lon/Ele
Speedbrake	Float	0	☐	☐		Both	0= no speedbrake, 1= max speedbrake
Throttle	FloatList		☐	☐		Both	Throttle 0=min 1=max
Trigger	Long	0	☐	☐		Both	Triggers send to socket
Type	String		☐	☐		Both	aircraft model
Velocity	Vec3	0 0 0	☐	☐		Both	Lat/Lon/Ele

Picture Database*: shows the objects and attributes in the WISE template database*

The remove aircraft message only contains the id of the aircraft as that is the only relevant information for removing an aircraft. In the driver this will remove the aircraft from the database of the integration server and notifying the other driver that an object with the specified id will no longer be simulated. As this message is sent to the plug-in the specified aircraft will no longer be updated by the plug-in. The aircraft ID reference is also removed in the driver so that the ID becomes reusable.

After the message has arrived at the driver it is decoded by reading the header type of the message. The header type is a parameter in a switch-case function which processes the message based on header type. The new aircraft message contains only id and aircraft type. The id is stored as a long type and the aircraft type is stored as a char array. The information is stored in the integration server. The integration server notifies the other drivers that a new aircraft is registered so that the drivers can send the information to their respective simulators/plug-ins/applications.

The update message is stored in grouping of position (latitude, longitude, elevation), orientation (heading, roll, pitch), velocity and acceleration in each vector of the coordinate system. Position is stored as double type arrays with the other numeric arrays are stored as float type arrays. This makes it easier for the plug-in to receive data from the driver as it is formatted in the same way as X-Plane handles data.

A yet unresolved problem occurred when testing plug-in driver communication with TCP. As X-Plane on machine one got connection with the driver the user interface on machine two locked up and started shaking. This seems to be related to the driver as it did not happen anything like this while testing with the self implemented server acting driver. This problem disappeared when the protocol was changed to UDP, the reasons for the shaky picture have not been investigated due to time constraints.

4.7 WISE Communication

To communicate between X-Plane and the integration platform the X-Plane plug-in needs a socket which handles communication with a driver. Both the driver and the plug-in are able to send and receive data as byte streams. The byte stream is decoded in two steps, first to see which kind of data the package contains by converting the byte stream to an XPlaneHeaderMessage struct pointer to retrieve the header which declares what type of message it is. After that the byte stream is converted to the type of message-struct pointer declared by the header, for example a NewAircraftMessage struct pointer which states that it is a new aircraft which has not appeared in the game yet. Information from the received message is now accessible by using the message pointer to access its variables in the following way: message->variable.

During test of the implementation a maximum of two updates per second was used as a delay is seen when using a higher update frequency. This problem probably lies in the implementation of the driver since both the sockets and servers have been tested successfully with more updates per second. It should be noted though that WISE is usually able to handle up to 60 updates per second so the problem lies in the code for the driver. The update frequency is not a big problem though since many simulators are not able to send and receive data more often anyways.

4.8 Problems during development

4.8.1 Representation of aircrafts in X-Plane

When developing the plug-in some problems concerning the plug-in were encountered and dealt with. The first problem that occurred was how to draw the aircrafts in X-Plane. The SDK has a built in function called XPLMDrawAircraft which seemed suitable. The function requires parameters such as position, angles for the aircraft and aircraft type. The problem however was that when the function was called no aircraft showed up on the given coordinates. This problem was due to the fact that the XPLMDrawAircraft function only draws the aircraft for one frame. If the aircraft is to be drawn for a longer period of time a draw callback function must be used. The DrawCallback is a loop which is called when every frame is drawn.

When the Aircraft was drawn another problem showed up, the drawn aircraft was just a picture of an aircraft, not a real aircraft as far as X-Plane was concerned. This meant that the aircraft did not show up on radar and had only the essential data available (position, angles, type). An important notice here is that the speed is not an input for XPLMDrawAircraft, This means that the aircraft will be drawn on one position only and it will not move. To get the aircraft moving, the speed variable may be passed from another simulator and then used to calculate the next position on which the aircraft will be drawn. This means that a function for calculating the aircrafts movement must be implemented and used for the aircraft to move. It is also important to notice that when XPLMDrawAircraft is used, the aircraft drawn will not show on the radar since the aircraft is basically just a 3D image which is drawn and not seen as a true aircraft from X-Planes point of view.

The alternative to the XPLMDrawAircraft function is to make use of the AI-aircrafts in X-Plane. When X-Plane runs in single-player mode it creates other aircrafts controlled by an AI, these aircrafts fly along with the user and have data references which can be modified. The user of the simulator may specify how many aircrafts are to be shown by changing a setting within the simulator. A backside when using these aircrafts however is that X-Plane only supports a maximum of 20 aircrafts which might be considered too few, especially compared to the XPLMDrawAircraft which has no limit other than the computers performance.

When trying to change the position of an AI controlled aircraft by setting new real world coordinates in the data references the aircraft did not move. This problem was due to the fact that real world coordinates are not writable data references in X-Plane. To get past this problem one must convert the real world coordinates to OpenGL coordinates and then write those to the data references. Converting coordinates in degrees to coordinates in meters can be tricky, especially if you do not know where the 0,0,0 OpenGL coordinate is located on the map. This is however not a problem in X-Plane since the SDK has a function called XPLMWorldToLocal which handles this computation.

After obtaining the OpenGL coordinates attempts were made to change the aircrafts position. This was however unsuccessful, the problem now turned out to be XPLMAcquirePlanes which is a SDK function that gives the plug-in access to modify aircraft data. This function has to be called before trying to modify the AI-aircrafts.

The decision to modify AI aircraft was made [see Chapter 4.10].

When changing the position for AI aircrafts a few new problems appeared. When the position and heading for an AI aircraft was changed the aircraft moved to the correct position and it had the correct heading, however after moving the aircraft it continued towards its previous destination and not in the given heading. Sometimes the aircraft that was moved would also get some strange looking behaviour where it crashed and bounced off the ground. There was also a problem with aircrafts standing on the ground since they would not move when changing the data references. These problems were easily solved by using XPLMDisableAIForPlane to disable the AI for a specific aircraft.

When the AI is disabled for an aircraft, physics for that aircraft is also disabled. This means that the aircraft stops to move as soon as the AI is disabled. With the AI disabled one can move the aircrafts as one likes but the drawing must be done manually using functions. To enable the AI again XPLMReleasePlanes must be called which releases control over all the aircrafts and not just a specific one. If the aircraft data references are to be modified the function XPLMAcquirePlanes must be called again after XPLMReleasePlanes has been called.

If the AI aircrafts are being used when running a multiplayer session which uses the plug-in, a problem is that AI aircrafts show up on different locations for every player. This is quite a large problem a player might collide with aircrafts on another players screen even if he does not do this on his own screen. It is also difficult to use air traffic control since the controller does not know where aircrafts are located. This problem has not been solved in the plug-in due to time constraints. There are several solutions to this problem, one is that the first simulator that is registered on the server sends data concerning all AI aircrafts. This data is then sent to every new simulator that is registered on the server. In this way, the first registered simulator controls all AI aircrafts for the other simulators. The problem when using this method is that if the first registered simulator shuts down first, all aircrafts which are not user controlled will stop to move.

A second solution to the AI aircraft problem is to have an extra computer running an instance of X-Plane. This computer would not have a user, instead it sends data for all aircrafts except the ones being controlled by other computers/users. By using this method a more rigid solution is achieved but at a higher cost economically since an extra computer is needed.

A necessary feature in the plug-in was the ability to set and change the aircraft model for each user. This is important not only because it might look strange with a commercial airliner flying as a fighter jet but also because the input data needs to be correct. An example of this correctness might be that if the simulator receives data for a fighter which has three sets of landing-gear and that data is applied to a commercial airliner which has six sets of landing-gear only half the landing-gears will be affected. This makes it important to be able to set the aircraft model.

All aircraft models in X-Plane are located in the “X-Plane 10\Aircrafts” folder where each aircraft has its own folder and a file with the file extension .acf. It is important to know the location of this file since it is needed when loading the aircraft-model in X-Plane.

The first problem with setting the aircraft type was to retrieve the search-path for the current aircraft. This was easily done since the SDK contains a function called XPLMGetNthAircraft (int AircraftNbr, char* buffer) which saves the search-path for the aircraft with AircraftNbr into the char array buffer. This search-path was however not the one wanted since it was a complete search-path and the one wanted was “X-Plane 10\Aircrafts\...\aircraftType.acf”. This is desirable as the installation path should not matter when using the plug-in. This problem was solved by iterating through the char array and finding the first notation of the number zero. This zero is the one in “X-Plane 10”, when this is found one can create a substring containing all data from zero+1 to the end of the char array.

It was decided to only send the aircraft type with a NewAircraftMessage [see Chapter 4.10].

To set the aircraft type the SDK function XPLMSetAircraftModel(int index, char* search-path) is used where index is the aircraft ID and search-path is the search-path for the aircraft model. This function is called when the plug-in receives a newAircraftMessage, the message contains the aircraft ID and the correct search-path for the aircraft which that ID uses.

During the SetAircraftModel tests one machine running Windows XP Service pack 3 the simulator continuously crashed after enabling and disabling the plug-in two times. The Windows 7 machine also experienced this problem at later tests. After restructuring the plug-in, by moving the XPLMAcquirePlanes function from XPluginStart to XPluginEnable, this problem seems to be solved.

4.8.2 Other objects in X-Plane

When trying to draw objects using the XPLMDrawObjects function the object chosen for drawing could not be seen on the screen. Since this problem occurred when using the XPLMDrawAircraft function as well, conclusions were that this function also needed to be in a loop that is called before every frame is drawn. When the loop function was used the correct object showed up on the screen as expected but when starting to fly the object moved along with the aircraft. This turned out to be quite an embarrassing problem which was easily fixed. To be able to see the drawn object it needed to be within a certain distance from the aircraft. A decision was made that ten meters beside the aircraft would be a sufficient distance for the user to be able to see the drawn object easily. The problem however was that the position for the object was set inside the loop function and every time the loop function was called it read the aircraft position and then positioned the object. Therefore the object moved along with the aircraft. The problem was solved by using global variables and setting the position for the object when registering the loop function.

4.8.3 General problems concerning X-Plane and the plug-in

X-Plane has a built in multiplayer function where users may fly together without AI-aircrafts. When running X-Plane in this mode with a plug-in that controls aircrafts active X-Plane has tendencies to freeze. If two players are running X-Plane with plug-ins controlling aircrafts and it freezes one player needs to inactivate the plug-in. Most likely the reason for this is that when the plug-in is active both users claim access to modify the aircraft data which X-Plane does not allow. This is however not a problem in the finished implementation of the plug-in since it runs X-Plane in single-player mode and all communication is done via the plug-in instead of X-Plane.

Search-paths for all DataRefs are built in the same way and may look like “sim/multiplayer/position/plane1_lon” which retrieves the longitude for the first multiplayer-plane. This makes it fairly easy to create a universal string and then only change the aircraft number. By using the “sprintf” function one is able to have a model-string and change certain elements inside it.
The function is used as follows:

```
sprintf (strBuf, “sim/multiplayer/position/plane%i_lat”, iPlaneNbr);
XPLMGetDataf(strBuf);
```

In the example the integer iPlaneNbr which contains the aircraft-number is applied to the string instead of “%i” and the new string is then saved into strBuf which is a char array. This means that if iPlaneNbr is 1, strBuf will contain “sim/multiplayer/position/plane1_lat”. A consequence of this is that when XPLMGetDataf(strBuf) is called it will return the latitude value for multiplayer-plane one. By using this function in a for-loop beginning on one and ending on 19 the code shrank from 423 lines to 25 which combined with comments is much more foreseeable.

4.9 Problems outside development

Here follows some problems encountered that did not derive for plug-in development.

A problem was encountered that made X-Plane crash on initialization. This crash was not related to the plug-in as the plug-in was removed from the X-Plane folder. X-Plane would start up as normal but once the screen with UI was expected to appear everything crashed. This had something to do with the aircraft being a seaplane and starting on water. It was found that it is possible to change the aircraft type by editing the search-path in the file X-Plane.prf located in “X-Plane 10\Outputs\Preferences\”.

It is possible to change the aircraft model for all 20 aircraft by modifing the search-paths in the X-Plane.prf file. To change which airport the user starts at is possible through X-Plane Binary.prf located in the same folder.

Another issue with the driver was that it was sending data to the simulator a bit too often. This issue was solved by using a variable which stated when it was ok to send data. The variable was set to one when the data update was completed and was then immediately set to zero again.

When an attribute is updated in WISE the server immediately sends the update to the other simulators connected. This was a problem since the plug-in sends several attributes in one message. If the message received from the plug-in contains four attributes it will make the server send four messages to the other simulators which is not desirable since the plug-in wants to receive information in the same way that it sends information. To resolve this problem a trigger attribute was added to the data model. As long as the trigger attribute is zero no data will be sent from the server and when all attribute updates are completed the trigger will be set to one and then immediately set to zero. When the trigger is set to one the server sends a message and since the trigger is set to zero again, only one message is sent.

4.10 Design decisions during development

The following decisions were made during development:

Which method to use for displaying aircrafts, modifying the AI aircrafts or draw the aircrafts with the XPLMDrawAircraft function? The decision fell upon modifying DataRefs for AI aircrafts, mostly because there is a lot more information to be given and set when using DataRefs and one of the tasks with the project was to extract and import as much information as possible.

To minimize the amount of data being sent over the socket it was decided that the aircraft type would only be sent with the NewAircraftMessage struct which means that it will be sent only once when X-Plane is started. There is however a possibility to send a newAircraftMessage again but then the user will have to disable and then enable the plug-in.

The decision not to use the built-in UDP communication was made. This was based on several threads on the X-Plane forum [forumudp] which stated that the syntax for X-Planes UDP communication could change between updates and versions of the simulator. This was unwanted since the user of the plug-in needs to be able to update the system with the plug-in still working. By implementing the SDK instead of X-Plane’s UDP communication it is possible to form the structure of the data sent. This enables more control over the information and is safer to use due to persistence during SDK development.

During a meeting with the supervisor it came up that other simulators might not be able to send updates at high rates such as 50 Hz. This led to the design decision that the plug-in will not send as often as every frame but instead at a configurable rate. The rate can be configured in the included configuration file. At that moment this made the aircraft updates look bad as the aircrafts would jump to the new position at every received update. This is to be solved with dead reckoning.

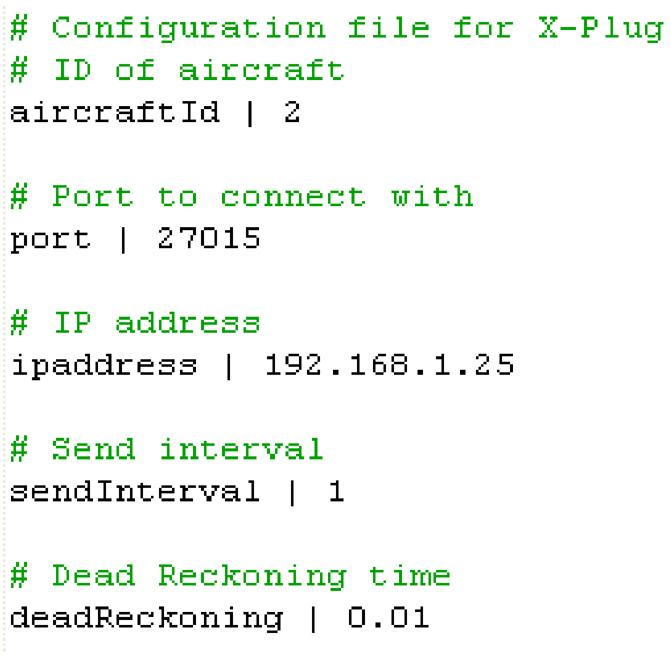
5 Result

This project resulted in a plug-in for X-Plane 10 and a driver for WISE, which is associated with a database developed with WISE.

The plug-in for X-Plane is able to send and receive information via socket communication. The data sent and received is position, orientation, velocity, acceleration, id of the aircraft, flaps, speedbrakes, landing gear and aircraft type. These attributes are sent in three different kinds of messages : newAircraft, updateAircraft and removeAircraft. newAircraft contains a unique id for the user and the aircraft type for that user. updateAircraft contains all aircraft information except the aircraft type and removeAircraft only contains the id that is to be removed.

The developed driver for WISE receives the information sent from the plug-in and applies it in the database. The driver uses a self made datamodel to map the different datatypes sent from and to X-Plane, this datamodel was also created during the project. The driver is also able to receive information from the database and sends such information to the plug-in where it is applied in the simulator which completes the integration.

During startup the plug-in reads data from a configuration file to get the ip-address and port to connect on. The configuration file also contains the users id and the interval for sending data to the driver. Example see Picture ***Config***



```
# Configuration file for X-Plug
# ID of aircraft
aircraftId | 2

# Port to connect with
port | 27015

# IP address
ipaddress | 192.168.1.25

# Send interval
sendInterval | 1

# Dead Reckoning time
deadReckoning | 0.01
```

Picture Config *: Contents of the configuration file, the variable deadReckoning specifies the interval at which the DR function will be called*

When the simulator starts, newAircraft is sent to the driver to register the id and aircraft-type. When this is completed the plug-in immediately starts to send updateAircraft at the specified interval and when the simulator shuts down a removeAircraft is sent. During this whole procedure the plug-in listens for data on the specified port and applies data in the simulator as it is received.

6 Conclusion

X-Plane is a very competent flight simulator and there are lots of possibilities with it when it comes to integration. Considering the flight physics, the advanced weather model, weapons and easy access to data for almost everything concerning these things one could easily say that the sky is the limit for what you can do with X-Plane. This is not the whole truth though since X-Plane has its limitations such as only being able to draw a certain piece of the map and the 20 aircraft limitation. The degree to which the limitations affect the integration possibilities vary. If simulation only occurs on a limited area of the map and less than twenty aircrafts are being used the limitations does not matter but if the goal is to simulate realistic air traffic over a whole country X-Plane’s limitations affect the possibilities to do it. It is also a flaw that no default airport scenery exists for Europe since it makes all airports look the same except for the runway layouts.

Aircraft data is accessed by using the SDK functions and data references. There are two main types of SDK functions, read and write, and they all have a data reference as parameter. The difference between read and write is that write also has the value to be written as parameter. The weapons in X-Plane are accessed in the same way as aircraft information. This information mainly concerns missiles and bombs such as their position and velocity.

When creating a plug-in there are five functions which must exist: start, stop, enable, disable and receive message [see Appendix 9.3]. If these functions do not exist the plug-in will not work. There are different access functions for data which depends on if the data is to be read or written and also depends on the type of data that is to be accessed. These functions have similar names but the last letter/letters decide what type of data the function concerns. It is not allowed to access data before the simulator is fully initialized since it may cause the system to crash. There are loop functions in the X-Plane SDK which may schedule the function calls to retrieve or write data, these loop functions are called callbacks and are not activated until the simulator is initialized. The callbacks are efficient since they do not slow down the simulator performance more than what is necessary and they can also be used to write and/or access data in the simulator as close to initialization as possible.

Plug-ins are easily installed and removed in X-Plane. When creating the plug-in one must simply change the file extension from .dll to .xpl. To install the plug-in the .xpl file must be put in X-Plane’s plug-in directory [see Chapter 4]. To remove a plug-in the simulator needs to be shut down and then it is simply to remove the .xpl file from the plug-ins directory.

There are a few ways to get the plug-in to communicate with a server. X-Plane has a built in UDP function to send and receive data, this was however not chosen after consulting different forum threads which said that the syntax for UDP-communication could change between different versions/updates of X-Plane. Socket communication using UDP is fairly easy since the plug-in only has to send information to the specified port on the specified IP address.

When creating the plug-in several types of messages were created. A message is a struct containing the interesting variables and a header saying what kind of message it is. The message types are new aircraft, update aircraft and remove aircraft. When sending messages like this the receiver only has to check the message type to know the content of the message. Knowing the content the referred variables can then easily be set.

6.1 Future Development

With the use of the SDK functions one is able to draw objects wherever one wants. If one wants a custom created object that is ok as well as long as it is in the .obj format and is a 3D object [see Chapter 3.1]. Objects are drawn with the use of the SDK function XPLMDrawObjects which takes the objects search-path as a parameter. The chosen object is then displayed for one frame. If the object is to be persistent a callback function needs to make the draw function call so that the object is drawn every frame.

It is possible to customize the scenery in X-Plane by creating liveries for airports and adding more detailed maps. Laminar Research, the developer of X-Plane, provides an open source world editor program [see Chapter 3.1].

There are lots of possibilities for the future with plug-ins. It is possible to draw objects such as persons and vehicles which expand the list of simulators that can be integrated with X-Plane. A plug-in can export and import data for missiles and bombs which enables both air-to-air and air-to-ground combat. It is also possible to register hits on aircrafts with the use of data references. As X-Plane evolves and the community and SDK along with it, the possibilities can reach new heights.

7 Dictionary

AI	Artificial Intelligence which makes the airplane fly by itself
ATC	Air traffic Control
Char	Standard C++ one byte character
DataRefs	Data references: search-paths for data in X-Plane
Dead Reckoning	Calculates where an object should be at a later time based on the object’s position and velocity
DLL	Dynamic Link Library
Float	Standard C++ 4 byte decimal number
Frame	One drawn picture on the screen
RAM	Random Access Memory
SDK	Software Development Kit refers to the X-Plane SDK provided at www.squawkbox.net/xpsdk/
String	Standard C++ string consisted of one byte characters
Struct	A collection of variables used to structure data, useful for communication
STS	Saab Training Systems
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

8 References

[bashforth] Information of Adams-Bashforth integration
http://math.fullerton.edu/mathews/n2003/AdamsBashforthMod.html
(2012-05-25 11:10)

[berkeley-socket] Information about Berkeley-socket
www.on-time.com/rtos-32-docs/rtip-32/programming-manual/programming-with/berkeley-socket-api.htm
(2012-04-12 11:50)

[c-socket] Information about the Csocket class.
http://msdn.microsoft.com/en-us/library/wxzt95kb(v=vs.80).aspx
(2012-05-17 12:38)

[datarefs] Link to available data references in X-Plane
www.xsquawkbox.net/xpsdk/DataRefs.html
(2012-03-21 15:29)

[dead reckoning] Explanation of dead reckoning
http://www.merriam-webster.com/dictionary/dead%20reckoning
(2012-05-17 11:20)

[dll] Information regarding the PLUGIN_API declaration
http://msdn.microsoft.com/en-us/library/a90k134d(v=vs.80).aspx
(2012-04-12 13:03)

[drcalculation] Calculation used for dead reckoning
http://www.gamasutra.com/view/feature/3230/dead_reckoning_latency_hiding_for_.php
(2012-05-18 10:34)

[extern] Explanation of Extern usage
http://msdn.microsoft.com/en-us/library/0603949d(v=vs.80).aspx
(2012-05-17 15:23)

[forum] Link to the X-Plane forum used during this project
http://forums.x-plane.org/index.php?showforum=23
(2012-05-08 15:33)

[object files] Information concerning object files in X-Plane
http://wiki.x-plane.com/Creating_Airplanes_and_Object_Files_for_X-Plane
(2012-05-18 09:21)

[sample code] X-Plane plug-in sample code
http://www.xsquawkbox.net/xpsdk/mediawiki/Category:Sample_Code
(2012-05-17 15:29)

[scrum] Explanation of SCRUM
http://www.mountaingoatsoftware.com/topics/scrum
(2012-05-17 12:03)

[sdk] Main page of X-Plane SDK
http://www.xsquawkbox.net/xpsdk/mediawiki/Main_Page
(2012-02-23 13:20)

[sdk21] Information about the updates in SDK 2.1
http://www.xsquawkbox.net/xpsdk/mediawiki/XPLM_2.1_Release_Notes
(2012-04-01 15:43)

[simple socket] Carrierlabs Berkley-socket implementation
http://sockets.carrierlabs.com/
(2012-05-17 12:39)

[sts] Homepage of Saab Training Systems
http://www.saabgroup.com/Training-and-Simulation/
(2012-04-25 12:05)

[trycatch] Explanation of the try-catch statement
http://msdn.microsoft.com/en-us/library/6dekhbbc%28v=vs.80%29.aspx
(2012-05-18 11:00)

[vatsim] Link to the VATSIM homepage
http://www.vatsim.net/
(2012-05-18 09:30)

[wed] WorldEditor information
http://wiki.x-plane.com/Scenery_Tools
(2012-05-21 12:58)

[winsock] Functions for Winsock
http://msdn.microsoft.com/en-us/library/windows/desktop/ms741394(v=vs.85).aspx
(2012-05-17 16:01)

[xplane] Information about X-Plane 10
http://www.x-plane.com/desktop/meet_x-plane/
(2012-05-17 15:43)

[xplane info] Information about X-Plane 10 Professional
http://www.x-plane.com/pro/certified/
(2012-02-23 11:05)

[xplane news] X-Plane 10 version information compared to X-Plane 9
http://wiki.x-plane.com/What's_New_in_X-Plane_10
(2012-05-17 11:35)

[xsquawkbox] Information concerning the XSquawkBox plug-in
www.xsquawkbox.net
(2012-03-21 15:09)

9 Appendix

9.1 X-Plane tests checklist

- ☐ Time it takes to start X-Plane 10?
  Depends on the computer performance and how much scenery that is to be loaded but between 1 and 5 minutes.

- ☐ Performance demands?
  X-Plane is able to run on low settings using a 256 MB graphics card, 3GB RAM and a 2.4GHz quad-core, it doesn’t look very nice though.

- ☐ Is it possible to view other aircrafts in X-Plane?
  Yes, there are other aircrafts, both on the ground and in the air, amount of aircrafts in the air is changed in the simulator settings but there seems to be a maximum of 20 aircrafts.

- ☐ Is it possible to view buildings in X-Plane?
  Yes there are buildings which the user may see in X-Plane.

- ☐ Is it possible to view ground objects in X-Plane?
  Yes there are ground vehicles in X-Plane, both trees, cars, trucks, boats, birds and deer.

- ☐ Are cars moving?
  Cars are moving on the roads.

- ☐ Is it possible for bombers to drop bombs?
  Yes, bombers may drop bombs if the aircraft-model supports it.

- ☐ Is it possible to collide with buildings and ground objects?
  No, a collision between objects isn’t supported, the aircraft only flies through the object, the same goes for buildings.

- ☐ Are the controls different when flying with a joystick?
  No the aircraft behaves in the same way whether the user is flying with joystick or mouse but the joystick improves the experience since it isn't that sensitive.

- ☐ Is there a fast forward button?
  No, such a function has not been found. Time can be modified but the movement is not affected.

9.2 SDK test checklist

- ☐ Is it possible to extract information from ground objects?
  No, no way of extracting this information has been found.

- ☐ Do objects have some kind of code to specify what kind of object it is?
  Objects are specified by the complete search path to the folder where the object is located.

- ☐ Is it possible to inject objects into X-Plane?
  Yes, there are functions in the SDK to draw objects.

- ☐ What kind of aircraft data is accessible? (coordinates, velocity, height etc)
  A lot of data is available, everything from position to levers inside the cockpit for the users aircraft, a bit more limited for AI aircrafts though.

- ☐ Is it possible to access information concerning ground objects such as cars?
  No, not by default.

- ☐ Is information concerning weather available?
  Yes, information of weather is available, such as cloud density, air humidity, rain level etc.

- ☐ Is it possible to set the weather?
  Yes, it's possible to set the weather remotely.

9.3 Essential plug-in functions

- *PLUGIN_API int XPluginStart (char* outName, char* outSig, char* outDesc) the input variable are name, signature and description for the plug-in*
- *PLUGIN_API void XPluginStop (void)*
- *PLUGIN_API void XPluginDisable (void)*
- *PLUGIN_API int XPluginEnable (void)*
- *PLUGIN_API void XpluginReceiveMessage(XPLMPluginID inFromWho, long inMessage, void* inParam)*

PLUGIN_API which stands prior to the function return types is a definition. The definition declares that it's a dll-export file and adds the export directive to the object file so that a definition file (.def) is unnecessary [dll].