Draft description for a Maddog IOC Server

Author: Frank Sørensen (frank@jkelloggs.dk)
September 2009
Revision: 0.2

Abstract

This paper presents a number of issues faced by MD-80 simpit builders trying to interface with Leonardo SH's FlyTheMaddog. Interfacing with the product is difficult because the product contains a closed software model consisting of the variables that are specific to the Maddog product. The variables are not available for reading via standard protocols such as FSUIPC. After presenting the issues a draft description for a standalone software service implementing the IOCP protocol is elaborated as a possible solution for the described issues.

Revision history

0.2	Changed title from Maddog Middleware to Maddog IOC Server. A solution as a standalone TCP/IP software service implementing the IOCP protocol is elaborated.
0.1	Initial description of problem cases and a draft for possible implementation of solutions.

Problem

In the add-on market for MS Flight Simulator there is really only one option if you want to get the simulated experience of flying the McDonnell Douglas MD-80: Leonardo SH's "FlyTheMaddog", or just "Maddog". The way this product simulates all the various subsystems of the aircraft is superb. However, you will face a number of challenges if you are a simpit builder, trying to build specialized hardware and software that can interface with the Maddog product. You will probably find that you need a bridge between your hardware devices on the one side and the MS Flight Simulator + Maddog on the other side. There are already general purpose products available that allow you to write the logic of this interfacing in fairly high-level scripting languages or ini notation, e.g. FSIUPC and SIOC from OpenCockpits.com.

A major problem is, however, that none of the mentioned technologies will let you interface with the information that is specific to the Maddog part of the simulator. They will only give you access to standard variables that are not part of the Maddog model such as gear position, altitude, engine EGT etc. These variables can be read via FSUIPC regardless whether you are flying the MD-80 or most other aircrafts. So getting the state of basic aircraft variables is not the issue, thanks to FSUIPC by Peter Dowson. The issue is getting information such as the selected flight plan in the FMS, the selected mode on the MCP or the indication of the APU RPM PERCENT gauge in the overhead panel. I will present a number of problem cases that I am trying to address in this document. They are examples of different categories of problems that can be generalized to other similar cases. The problems of the different categories may need different approaches in their solutions, which I will present in the later part of the document.

To clarify: I will be using the word "model" a lot in this document. It is to be understood in the abstract sense, not as a physical model or 3D model. Simply as an abstract collection of a number of variables that can have specific values. An example of such a variable could be the state of the Start Pump switch.

Cases and suggested principles for solutions

General considerations

It is generally not possible in an FSUIPC-like way to read the values of variables residing inside the Maddog model. At least not by any method that I know of. I hope one day that a method may be discovered. But it is a fact that is it possible create a piece of software that can *write* a value into most of the variables residing inside the Maddog.[1] And this may turn out to be not such a bad thing after all considering the following proposition: As a simpit builder you are probably more interested in making the state of the software model reflect the state of the hardware model (knobs, dials, switches etc.), rather than vice versa. This, I believe, will be true in the vast majority of the cases. In some cases, however, you would want the information flow to go from the software model to the hardware devices, e.g. if the auto throttle is turned off by pressing AT Disengage button on the throttle handle the switch on the MCP should automatically snap to the OFF position. But as stated in most cases you will probably just want to get the state of the hardware devices and then force the Maddog software model to reflect that state.

Problem case 1: Obtaining access to information about the selected flight plan

In the Maddog a flight plan is created by typing information into the MCDU. In the MCDU you can select an already stored flightplan (.mdr files) or you can type the name of each part of the flightplan individually. Each time a modification is made the new flightplan will be in a kind of buffer zone state until it is either executed or rolled back to the state it had before it was modified. As a normal end user of MS Flight Simulator it is not difficult to get visual information about the currently active flightplan because it is shown in the MCDU and the Navigational Display and sometimes the Primary Flight Display. But for a third party developer to get digital information about the currently active flightplan is practically not possible because it resides in the closed part of the Maddog software model.

Possible solutions for case 1

The problems of this category share the common attribute that they are driven by specific user input events. So if the input events could be intercepted before they are passed on to the Maddog model it should be possible at runtime to construct a kind of buffer model outside the Maddog model that reflects the state of the model inside the Maddog. The accuracy of such a model will be based on a principle of probability and some emulation of the CDU input logic. Here is an example:

1. Powering up the APU after starting a flight in a cold and dark configuration, the MCDU will normally start with the IDENT page.
2. Pressing LSKR6 will bring the MCDU in a state of showing POS INIT page. This is equivalent to pressing the INIT REF button.
3. The POS INIT page consists of three pages of which the first is shown as default. If the user presses PREV PAGE button the MCDU will show page 3 of

1 Special command IDs can be sent via the FSUIPC offset 0x3110. The IDs can be seen here: http://www.flythemaddog.com/download/2006/Maddog_commands.txt

3 of the POS INIT PAGE.

4. Pressing the LSKR6 button in all three POS INIT page will take the user to the RTE page which is the really interesting page in terms of intercepting the selected flightplan.
5. On the RTE page two things can happen:
   5.A Origin and destination can be typed specifically by entering the names of the airport in the scratch pad and pressing LSKL1 and LSKR1. Input validation may be necessary to validate the typed airport names as valid ICAO codes, but after these input events we should know the origin and the destination airport.
   5.B A route can be loaded from an .mdr file. In this case the filename without the suffix will be entered in the scratchpad after which the LSKL2 is pressed. To get the information from the mdr file may require constructing an MDR parser.
6. Knowing at least the origin and the destination airport, and maybe more if an mdr file has been selected, the next step would be to emulate the LEGS page, and so on.

Emulating this logic in a buffer software system is not easy, but on the other hand not impossible.

Other cases in the same category: Active AP flight mode, most of the overhead switches and dials.

Problem case 2: What is the state of APU Power Available LEDs?

During the aircraft startup procedure a gauge shows the rpms of the APU heating up. When it reaches appr. 90 PERCENT RPM two blue lights in the overhead panel light up to indicate that APU power is now available. Again the state of the APU pressure is part of the closed (non-accessible) part of the Maddog software model. The question is now if it is possible to get digital information about the state of the APU gauges in some way?

Other cases in the same category: Most LEDs that light up as a result of internal aircraft logic, e.g. Master Caution. Analog gauges with needles that move independently, i.e. not directly as a result of user input events. The TCAS subsystem.

Possible solutions for case 2

So far the best solution that I have been able to come up with is to programmatically make a screenshot of the visual MS Flight Simulator user interface and then read the RGB color value of specific pixels. This is no way an elegant or efficient solution. And I am not even sure that it will work for analog spinning needle gauges, but for lamps changing between lit and not lit, it should work.

As mentioned earlier the best solution would be to get access to an Application Programming Interfacing (API) that can provide third party software developers with digital information about the state of the Maddog model – very much in the way that the basic aircraft variables can accessed with the FSUIPC API. I am hoping that Leonardo SH will one day make such an API available.

Suggested software solution

I am planning to develop a piece of software that can act as a software system "in the middle". So far the best description I have been able to find for this type of software is "middleware". The middleware will have three main areas of responsibility:

1. Intercept all input events that are specific to Maddog and then of course pass them on to the Maddog via the FSUIPC offset 0x3110.
2. Construct a buffer model of the state of the Maddog, based on the inputs, guessing and emulation of the Maddog logic. This will provide a kind of probabilistic map of the Maddog model.
3. Make this buffer model of variables easily accessible to third party hardware and software developers.

Part 2 is of course the difficult part as described in problem case 1. It may require modeling a substantial part of the Maddog software logic.

Suggestions for implementation

So far these are my considerations for the new Maddog middleware software system:

- The system must have a high degree of availability so that other parties can interface with the system. So implementing a system that runs as some kind of service in a TCP/IP seems sensible. After some investigation I believe that there will be a lot to gain from basing such a service on the IOCP protocol which is known from the products of OpenCockpits.com. This is described in details in the next part of this document.
- The system is by no means a replacement for the general FSUIPC / WideFS setup because that system is already efficient and stable as it is. So the Maddog Middleware system is meant to run alongside FSUIPC as a complementary system. In fact, as already described, the middleware will rely on FSUIPC for the sending the commands to the Maddog. This means that the middleware server may not even need to run on the primary Flight Simulator PC. Running on a secondary PC that has WideFS will be sufficient.

Elaboration of a Maddog IOC server

IOCP (IOCards Protocol) is a lightweight network communication protocol.[2] Most of the products from OpenCockpits.com have interfaces for this protocol. The protocol is lightweight in more than one sense: First of all, it is simple. Only a few message types need to be implemented and the structure of the messages is simple too. At runtime the protocol is lightweight too because a given client has told the server which variables it wants to be notified about, and subsequently the client will only receive notifications when those specific variables change their value inside the server data model. So data will only be sent when needed. This is very much unlike communication via the FSUIPC protocol. When communicating with the FSIUPC server the client software will usually poll the server for the value of variables at an interval, e.g. 20 times a second. It means that in FSUIPC there is network communication between server and client, regardless whether any variables have changed their value. So in this sense FSUIPC is not a lightweight protocol.

2 http://translate.google.es/translate?u=http%3A%2F%2Fwww.iocpserver.net&hl=es&ie=UTF8&sl=es&tl=en

Gauge Composer compatibility

The Gauge Composer tool from OpenCockpits.com[3] communicates with Flight Simulator and other data sources via the IOCP protocol. The maximum number of data sources is 2, but all the examples seen so far use only one, which is normally the default implementation of IOCPServer that runs as a module inside FS9 or FSX (as IOCPServer.dll). The IOCP module provides access to approximately the same set of variables and commands that are covered by FSIUPC, i.e. most of the basic Flight Simulator variables. But like FSUIPC the IOCPServer will give you access to the variables events that are specific to the Maddog product. So in this case the problem is the same as described in the problem cases above.

If the Maddog Middleware is implemented as a TCP/IP server socket communicating in the IOCP protocol it will be possible to offer a secondary data source for gauges composed in the Gauge Composer. So the plan is the make the Maddog Middleware such a secondary data source. However, communicating with the Maddog Middleware will not be restricted to gauges made in Gauge Composer. Since the IOCP protocol is openly specified and quite simple to implement it should not be big issue to implement other clients in ofter software environments.

3 http://www.opencockpits.com/modules.php?name=Downloads&d_op=viewdownload&cid=32