Tommy Ingulfsen
July 30, 2002

Pushing Tin Project - First Design Draft

1 Some notes

This is the first iteration of the design for the Pushing Tin Project. The INFORMED method of Praxis Critical Systems is being used for the safety critical component, the Automatic Collision Detection System. Note that this is the only part of the software that is considered here, the rest (including GUI, file I/O etc.) is not taken into account. This is partly because I am uncertain about what design method is best for the "Ada" part of the program.

Question: Do you have any ideas for design methods for the non-ACDS part of the system, the part written in ordinary Ada?

This wee document restates quite a lot of what has already been agreed. Please let me know if there is something in here that contradicts our earlier discussions!

I am trying to sum up our initial ideas for the system and take the first step toward something that can be programmed, so the level of detail is not high.

2 Brief system overview

At least to begin with, the system will not be interactive. Its behaviour will be determined by a script that defines flight plans and movements, together with the ACDS calculations. The script will be in the form of one or two files. The user will only be able to see the output in some way on the screen, not really being in charge of system operation.

The script information will be fed into the system in some way and analysed by a scheduler. The scheduler will be responsible for handling time in the system, that is, it updates the application with new information from the script at certain time intervals. So the program will not be real time, but use a clock with a "wait" mechanism to progress.

The scheduler will update a global data structure known as "The World", or just "The global-overall-datastructure-thing". "The World" will hold all information necessary to calculate any potentially dangerous flight paths.

The safety critical part is the ACDS, which will be encoded in SPARK. It will use data from "The World" to find out if any planes are on a colliding course. It is not known exactly how this will operate.If any hazardous situations arise, then some warning is going to be issued to the user.

3 INFORMED steps

3.1 Identification of the system boundary, inputs and outputs

This first steps is concerned with (I think) the **entire** system. The boundary of the system is the script(s) file that define flights and movements, and the display. The basic inputs are the script file(s) and the output is information displayed on the screen.

*Notes in this series are for ϵ-baked ideas, for $1 \geq \epsilon \geq 0$. Only exceptionally should they be cited or distributed outwith the NuSPADE project (EPSRC GR/R24081).

3.2 Identification of the SPARK boundary

"The World" is the SPARK boundary, because that is what the ACDS will interact with.

I am a little uncertain of the exact choice of boundary variables, but it seems straightforward to use "The World" as a variable package to SPARK. I suppose I could provide SPARK annotations for the specification of "The World" and then use the -# hide directive to conceal the (Ada) implementation. This is a point where some comments from you would be good.

I don't think we will need any boundary abstraction layers, since the data to be stored is already in an appropriate form. See section 3.3.

Question: It could be argued that there is another boundary, namely the screen, since the user must be informed. I think that could be a part totally separate from the ACDS. What do you think?

3.3 Identification and localization of system state

I think this step is concerned with the entire system, even though this analysis/design becomes slightly messy that way.

The following information must be stored on each aircraft:

- Callsign
- Heading
- Current position
- Flight level (FL)
- Speed

As mentioned above, there will be 1 data structure, "The World". This will take input indirectly from the script, and store callsign, heading, current position and FL. Speed will be constant within the system, and should be stored within the ACDS since that is the only place where it will actually be used.

"The World" is probably going to be a variable package, global to the SPARK Automatic Collision Detection System. Some of the data items will be defined in terms of Ada type packages. The current position will be a set of 3-D co-ordinates. "The World" specification may be annotated in terms of SPARK. I imagine that speed is just going to be an ordinary integer constant.

The ACDS will need to use all the data above, plus the previous FL. This is to be able to work out if an aircraft is climbing or descending.

Question: It was mentioned that a history of positions could be useful, but is that really necessary. Should not the current position, speed, heading and FL be enough? Admittedly I don't know how the ACDS will work yet.

3.4 Handling initialization of state

"The World" has a valid value prior to ACDS execution. In terms of annotations this implies that some state or variable must be mentioned in an –# initializes (?).

Speed will obviously be initialized by the ACDS.

3.5 Handling secondary requirements

In the INFORMED document, secondary requirements are mentioned as testing requirements and implementation specific requirements. See page 31. I cannot see that we have any such secondary requirements to take into consideration.

3.6 Implementing the internal behaviour of components

I have not got this far yet, and I guess it's a bit too early as well.

4 Concluding remarks

This is just meant as a basis for discussion, and a first step on the way. A major point to look further into is what level of abstraction should be used in the SPARK implementation. At the highest level, you can imagine something like this being used in the specification of the ACDS system:

–#derives SafetyState from AircraftSystem.State

where SafetyState is the internal state of the ACDS and AircraftSystem.State is the state of "The World". But would it not be better just to leave it as it is? I mean, simply to use the heading, position and so on, to get a useful level of detail in the annotations?

That I don't know yet how to implement an algorithm for collision detection is a bit of a hole. I will take some trips to the library to consult the maths books there and revise the vector maths we had in first and second year. If I don't get anywhere, I'll consult some lecturer in the maths departement.