Tommy Ingulfsen
August 16, 2002

The SPARK Ada Experience

1 Introduction

What is it like to learn SPARK and Ada in 8 weeks? This is how it was for me. If you spot any errors in this text, then I probably have not understood Ada and SPARK properly! Informally written, this is exactly how I feel about SPARK/Ada.

2 What is this?

At first sight, SPARK seemed rather tricky but not impossible. I mainly read John Barnes' book on SPARK and his book on Ada95 at the start of the project. These texts are quite heavy going, but they do explain (almost) everything in minute detail.

I read most of the SPARK book and more or less just browsed through the Ada book. After about 2 weeks of intense study of both programming and Air Traffic Controller topics I felt ready to start coding for real. Mainly, the files I created to begin with were little packages doing things I thought could be useful for the project.

3 Does it work?

Using Ada and SPARK was not as hard as I initially expected. With experience of Java, C and C++ the transition to other high-level languages is seldom tough. The parts I found difficult were how packages work and how to use the Examiner.

In Ada (and SPARK) the "dot" notation is used quite a lot, for example `Proximity.Min_Distance` refers to the `Min_Distance` procedure in package `Proximity`. This is much like Java and C++, where the notation refers to objects. But packages and classes/objects are not quite the same. Admittedly, the difference is still not clear to me. I looked at object-oriented programming in Ada (Barnes' Ada95 book), but decided that it looked too hard and was not needed. Therefore, the Pushing Tin program is mostly procedural.

It also took me quite long to figure out how to make "persistent" data structures in Ada or SPARK. By "persistent" I mean data structures that keep their data between calls to procedures. For example, the `Situation` package holds data on previous aircraft positions and this information must be available on the next call, thus the data is "persistent". The solution is to declare a global variable, known as an "own" variable in SPARK. John Barnes shows this at a very early stage in his SPARK book, with the stack example. I did not understand it properly the first time I read it and not until week 5 of the project did I comprehend how this programming concept works. When it finally dawned for me, I found global variables and "own" variables convenient and easy to use however.

The main frustration throughout the project has been to understand and use the SPARK Examiner. There are times when it seems like this "compiler" is working against the programmer rather than with him/her. Mostly these hard times occur when there is some subtle strange way of doing things that is not pointed out clearly in the SPARK documentation from Praxis Critical Systems or Barnes' book.

An example is array initialisation. Plain, ordinary for-loops are not accepted for this purpose by the Examiner. Instead an ugly notation is adopted where the programmer (poor me) has to battle with types until his eyes get red and sore. Another example is procedure names. I wanted to use the name

[*]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).

"Update" for procedures in SPARK several times, but every time I did, the Examiner came up with some seemingly unrelated error message. Upon changing the procedure name to e.g. "New_Update" the error disappeared, mysteriously.

4 Survival is possible

Having voiced my complaints, I would like to focus on some of the things I thought was quite nice about Ada and SPARK.

The Examiner usually gives very good error messages. At rare occasions it breaks and responds with something out of hand, but of all the languages I have seen, SPARK by far has the best error messages. Also, the Examiner tends to spot errors that ordinary compilers does not see, which is of great help when programming. Especially the `ineffective statement` message impressed me.

Annotations like `--# derives` and `--# global` are very helpful not only when reading code, but also when writing code. Annotations force the programmer to make explicit his assumptions about use of global variables, initialisation, operation and so on. This requires more thinking, but the reward is clarity.

It is not hard. As one of my lecturers once said - "Don't come and tell me that this is hard. It demands a bit of thinking, but it's not hard." I did not quite agree with him when he spoke of SML in this way, but had he stated the same for SPARK and Ada, I would not have had second thoughts about consenting.

5 What to tell students - and what NOT to tell them!

Since SPARK and Ada is apparently going to be taught to third year computer science students at Heriot-Watt in the first term this autumn, I would like to give some advice:

1. Remember that students tend to forget quite a lot of programming over the summer holiday. Give them some really easy exercises to do early on in the course. This will fool them into thinking it is dead easy, and might even make them feel happy for a short while.

2. Explain how own variables and global variables work in SPARK and Ada, especially how you can declare a variable in the "is" part of a package.

3. Array initialisation is, um, complicated. If students need to know this, give them at least a couple of good examples and explain how you must reference types when you have an array of records.

4. The Examiner likes to see files organised neatly in packages, with separate .ads and .adb files.

5. You can never have more than one `--# inherit` annotation, which caught me out more than once. Make sure that the `syntax` of annotations is understood.

6. When debugging it is useful to consult gnatmake, since it produces different types of messages than the Examiner and therefore can clarify the problem.

7. Do explain how to make meta files for the Examiner, so that you do not have to type the names of lots of files every time. The order in which the Examiner is presented with files matters!

8. SPARK is compilable using gnatmake and is therefore Ada at the low level. Hence it is easy.

6 Conclusion

It is not so bad.