Astronomical Ada

Experience Report

Swiss Ada Event 2017

White Elephant GmbH

Experience report on the problems we encountered getting a program written entirely in Ada to work on three popular PC operating systems:

- Windows (WinXp and later)
- Linux (Ubuntu Tahr)
- OSX (Sierra)

Why Astronomical Ada?

Because the program in question controls astronomical telescopes

Background:

- No Motor

  Whatever you are watching will gradually drift out of view due to the rotation of the Earth

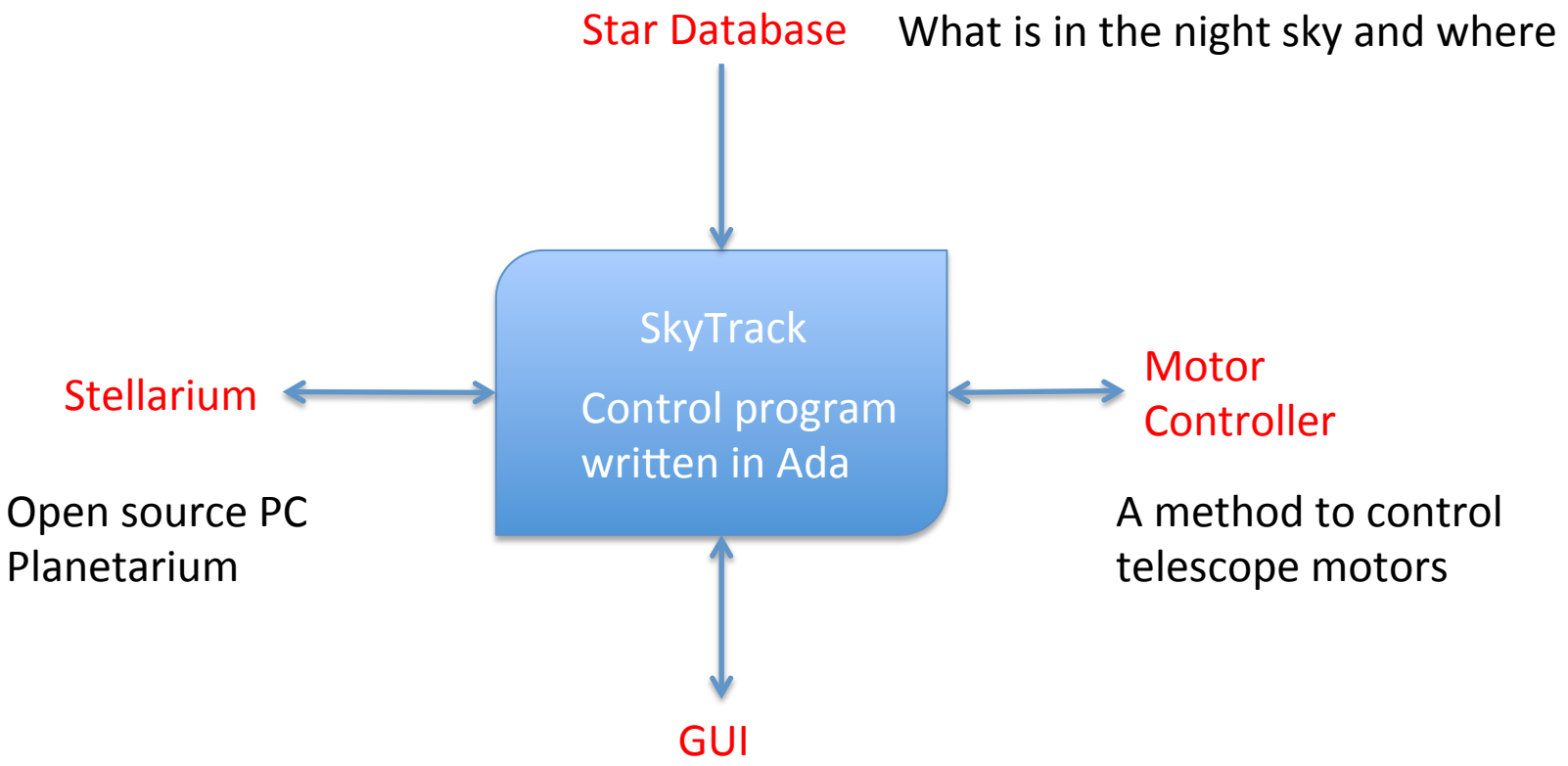
- One Motor

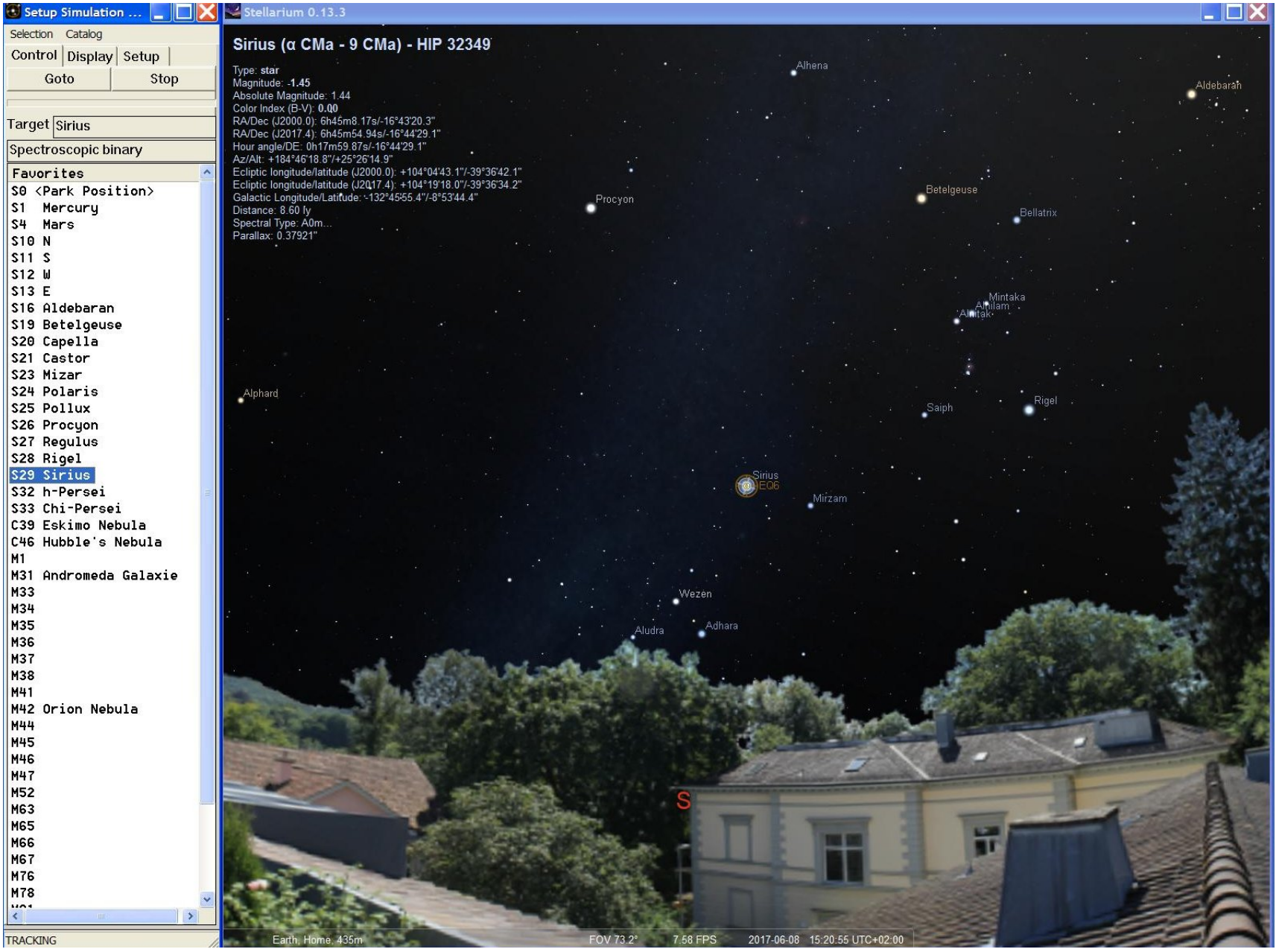
  When aligned with the Earth's axis cancels out the drift due to the Earth's rotation. But some objects move relative to the Earth : Eg. Planets, Comets & Satellites

- Two Motors

  Needs a control system

  To know what to follow and how to follow it


Star Database
What is in the night sky and where
SkyTrack
Control program written in Ada
Stellarium
Open source PC Planetarium
Motor Controller
A method to control telescope motors
GUI
So it can be told where to go and what to follow


Setup Simulation ...
Selection
Catalog
Control
Display
Setup
Goto
Stop
Target
Sirius
Spectroscopic binary
Favorites
S0 <Park Position>
S1 Mercury
S4 Mars
S10 N
S11 S
S12 W
S13 E
S16 Aldebaran
S19 Betelgeuse
S20 Capella
S21 Castor
S23 Mizar
S24 Polaris
S25 Pollux
S26 Procyon
S27 Regulus
S28 Rigel
S29 Sirius
S32 h-Persei
S33 Chi-Persei
C39 Eskimo Nebula
C46 Hubble's Nebula
M1
M31 Andromeda Galaxie
M33
M34
M35
M36
M37
M38
M41
M42 Orion Nebula
M44
M45
M46
M47
M52
M63
M65
M66
M67
M76
M78
TRACKING
Stellarium 0.13.3
Sirius (α CMa - 9 CMa) - HIP 32349
Type: star
Magnitude: -1.45
Absolute Magnitude: 1.44
Color Index (B-V): 0.00
RA/Dec (J2000.0): 6h45m8.17s/-16°43'20.3"
RA/Dec (J2017.4): 6h45m54.94s/-16°44'29.1"
Hour angle/DE: 0h17m59.87s/-16°44'29.1"
Az/Alt: +184°46'18.8"/+25°26'14.9"
Ecliptic longitude/latitude (J2000.0): +104°04'43.1"/-39°36'42.1"
Ecliptic longitude/latitude (J2017.4): +104°19'18.0"/-39°36'34.2"
Galactic Longitude/Latitude: -132°45'55.4"/-8°53'44.4"
Distance: 8.60 ly
Spectral Type: A0m...
Parallax: 0.37921"
Alhena
Aldebaran
Procyon
Betelgeuse
Bellatrix
Mintaka
Alnilam
Alnitak
Alphard
Saiph
Rigel
Sirius
EQ6
Mirzam
Wezen
Aludra
Adhara
S
Earth, Home, 435m
FOV 73.2°
7.58 FPS
2017-06-08 15:20:55 UTC+02:00

Originally written for MS-Windows

Could it be ported to other operating systems?

Majority of code and know-how is not operating system specific so why limit ourselves to Windows?

Should be easy

- Tasks are part of the Ada language (doesn't use OS specific libraries to create threads and protected objects)
- Compilers for Windows, Linux and OSX
- What the language doesn't supply is mostly provided by packages that have been implemented on various platforms.

However it wasn't as easy as we thought it was going to be.

In this presentation I want to describe some of the difficulties and our solutions

Five areas of concern:

1. Accessing the star database
2. GUI
3. UDP/IP communication with motor controller
4. TCP/IP communication with Stellarium
5. Other OS specific difficulties

1. Accessing the star database

SIMBAD is vast (>9’000’000 entries)

However most objects cannot be seen by amateurs or are of little interest.
This reduces the amount of data entries down to around 20’000 entries

Stars do come and go but…

Compared with the human lifespan, the night sky is relatively static

So we solved the database portability problem by not having a database!

Instead we wrote a program
(of course in Ada)
to convert the SIMBAD data into a very large structured constant

This constant is then compiled and linked into the StarTrack program

Advantages:

- Faster
- No external file (that can be lost or corrupted)
- OS Independent

2. GUI

Written using a package that provides a simple interface to create and manipulate common graphical objects.

Originally implemented using direct calls to the Windows API

So all we had to do was re-implement the implementation.

Surprisingly easy to do.

We chose to re-implement our GUI package based on Gtk

because it was available on all the target platforms

and Ada Bindings were available.

Windows

Windows API not task safe.

So use protected objects to prevent concurrent API calls

and an Ada task to process the Windows message loop.

Gtk

Restriction:

All Gtk calls MUST be executed from the SAME task

Solved using Ada:

- Protected types
- Task requeuing
- Abstract types and procedures
- Guarded entries

Gtk – Two types of call

Synchronous:

Calls to Gtk that return a value

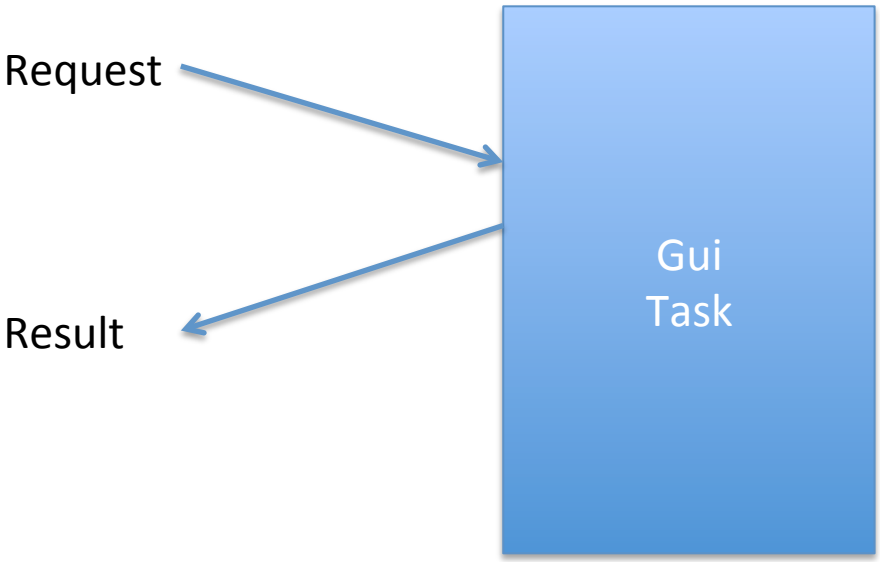
For example, get the contents of an edit box

Asynchronous:

Calls to Gtk that do not return a value

For example, write into a list view

Synchronous



Coding

```
type Request_Data is abstract tagged null record;
procedure Synchronous_Service (Data : in out Request_Data) is abstract;
```

Extend the Request_Data type to include the data to be sent to Gtk and to be received from Gtk.

```
type Check_Enquiry_Data is new Gui.Router.Request_Data with record
    Check_Box  : Gtk.Check_Button.Gtk_Check_Button;
    Is_Checked : Boolean;
end record;
```

Define the routine that the Gtk task should call to process the data

```
overriding
procedure Synchronous_Service (Data : in out Check_Enquiry_Data) is
begin
    Data.Is_Checked := Data.Check_Box.Get_Active;
end Synchronous_Service;
```

Coding

```
type Request_Data_Ptr is access all Request_Data'class;
protected Gateway is
    entry Synchronous_Request (Data : in out Request_Data'class);
private
    entry Serviced (Unused_Data : in out Request_Data'class);
    State : Gateway_State := Idle;
    Data  : Request_Data_Ptr;
end Gateway;
```

To perform a Gtk synchronous operation:

- Create a variable of the extended data type and initialise it
- Pass the data into the Gtk Gateway (a protected type)
- When control is returned, extract the results and pass them back to the caller.

```
function Is_Checked (The_Check_Box : Check_Box) return Boolean is
    Data : Check_Enquiry_Data := (Gui.Router.Request_Data with
        Check_Box  => The_Check_Box.The_Box,
        Is_Checked => False);
begin
    Gateway.Synchronous_Request (Data);
    return Data.Is_Checked;
end Is_Checked;
```

Coding

- The Synchronous request blocks until the Gtk task is ready to process the request (State=Idle)
- Then it makes a private pointer to the request data and changes the change to Busy;
- It then blocks until the Gtk task makes a rendezvous with it to signal that the Synchronous procedure has completed.

```
entry Synchronous_Request (Data : in out Request_Data'class)
when State = Idle is
begin
    Gateway.Data := Data'unchecked_access;
    State := Busy;
    requeue Serviced;
end Synchronous_Request;

entry Serviced (Unused_Data : in out Request_Data'class)
when State = Ready is
begin
    State := Idle;
end Serviced;
```

Coding

```
protected Gateway is
    entry Complete_Synchronous_Service;
    entry Check;
end Gateway;

function Synchronous_Data return Request_Data_Ptr;
```

- The Gtk task calls check to wait for something to do
- When released it performs the synchronous routine associated with the extended type pointed to by its private pointer
- Then it calls Complete_Synchronous_Service to change the state to Ready thereby releasing the caller.

```
loop
    Gateway.Check;
    Synchronous_Service (Gateway.Synchronous_Data.all);
    Gateway.Complete_Synchronous_Service;
end loop;
```

Coding

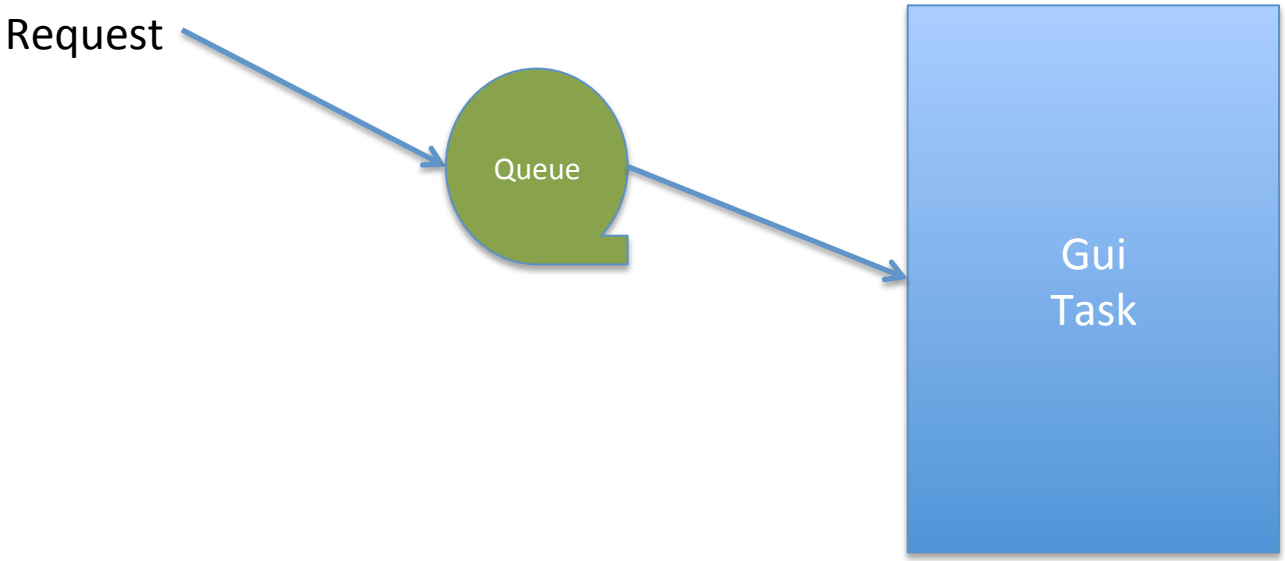
```
entry Check
when (State = Busy) is
begin
    null;
end Check;

function Synchronous_Data return Request_Data_Ptr is
begin
    return Gateway.Data;
end Synchronous_Data;

procedure Complete_Synchronous_Service is
begin
    State := Ready;
end Complete_Synchronous_Service;

loop
    Gateway.Check;
    Synchronous_Service (Gateway.Synchronous_Data.all);
    Gateway.Complete_Synchronous_Service;
end loop;
```

Asynchronous



Coding

```
type Message_Data is abstract tagged null record;
procedure Asynchronous_Service (Data : Message_Data) is abstract;
```

Extend the Message_Data type to include the data to be sent to Gtk.

```
type Set_Check_Data is new Gui.Router.Message_Data with record
    Check_Box : Gtk.Check_Button.Gtk_Check_Button;
    Is_Set    : Boolean;
end record;
```

Define the routine that the Gtk task should call to process the data

```
overriding
procedure Asynchronous_Service (Data : in out Set_Check_Data) is
begin
    Data.Check_Box.Set_Active (Data.Is_Set);
end Asynchronous_Service;
```

Coding

To perform a Gtk Asynchronous operation:

- Create a variable of the extended data type and initialise it
- Append the data to a queue (accessed via the gateway protected type)

The Asynchronous operation does NOT block.

```
procedure Set (The_Check_Box : Check_Box) is
    Data : Set_Check_Data := (Gui.Router.Message_Data with
        Check_Box => The_Check_Box.The_Box,
        Is_Set    => True);
begin
    Gateway.Asynchronous_Request (Data);
end Is_Checked;
```

Coding

```
package Message_List is new Ada.Containers.Indefinite_Doubly_Linked_Lists
    (Message_Data'class);

protected Gateway is
    procedure Asynchronous_Request (Data : in Message_Data'class);
private
    The_Messages : Message_List.Item;
end Gateway;

protected body Gateway is
    procedure Asynchronous_Request (Data : in Message_Data'class) is
    begin
        The_Messages.Append (Data);
    end Asynchronous_Request;
end Gateway;
```

Coding

- The Gtk task calls check to wait for something to do
- Either the state indicates that there is a synchronous operation to perform or if the message queue is not empty then there is an asynchronous operation to perform.

```
entry Check (The_Data_Type : out Data_Type)
when not (State = Busy) or else (The_Messages.Count > 0) is
begin
    if The_Messages.Count > 0 then
        The_Data_Type := Asynchronous;
    elsif State = Busy then
        The_Data_Type := Synchronous;
    end if;
end Check;
```

Coding

If there is something in the message queue then retrieve it and perform the Asynchronous routine associated with it.

```
loop
    Gateway.Check (The_Data_Type);
    case The_Data_Type is
    when Synchronous =>
        Synchronous_Service (Gateway.Synchronous_Data.all);
        Gateway.Complete_Synchronous_Service;
    when Asynchronous =>
        Asynchronous_Service (Gateway.Next_Message);
        Gateway.Delete_First_Message;
    end case;
end loop;
```

Coding

The Gtk processing loop:

```
loop
    while Gtk.Main.Events_Pending loop
        Unused_Boolean := Gtk.Main.Main_Iteration;
    end loop;
    select
        Gateway.Check (The_Data_Type);
        case The_Data_Type is
        when Synchronous =>
            Synchronous_Service (Gateway.Synchronous_Data.all);
            Gateway.Complete_Synchronous_Service;
        when Asynchronous =>
            Asynchronous_Service (The_Messages.First_Elelemt);
            The_Messages.Delete_First;
        end case;
    or
        delay The_Period;
    end select;
end loop;
```

But...

Whilst this worked for both Linux and Windows it didn't work for OSX

For some reason, under OSX, the Gtk task **MUST** be the main task rather than a task (thread) created by Ada.

This restriction meant we had to modify not only our GUI interface but also how the program setup the Gui Objects.

Before:

1) Create Gtk task
2) Create GUI objects using interface to Gtk task
3) GUI Close terminates Gtk loop and exits Gtk task

After:

1) Execute Gtk procedure from main task
2) Gtk procedure calls start-up procedure that creates all the GUI objects
3) GUI Close terminates the Gtk loop and exits the Gtk procedure.

However the start-up procedure cannot be executed by the main task

Because synchronous calls made from the procedure would deadlock and even if this were circumvented, synchronous events might require Gtk event processing

So the start-up procedure has to be executed by a separate task.

Although not difficult − it is an added complication!

3. UDP/IP

SkyTrack uses UDP/IP Ethernet datagrams to communicate with the motor control

UDP is used because it is more suitable for real-time.

SkyTrack sends a steady stream of datagrams to the motor controller instructing it what it should do in the near future. It doesn't matter if some of these get lost. More important is that they are sent immediately and not buffered and sent at some arbitrary time in the future.

Gnat Sockets

Gnat sockets is implemented on all of our target platforms

The Gnat sockets API very similar to the native Windows API we had been using.
So converting to Gnat Sockets was trivial.

But unfortunately it only worked under Windows!

To get SkyTrack working under Linux and OSX required a bit more effort.

The problem was due to the concept of UDP connections.

UDP connections

UDP is inherently connectionless

UDP datagrams are sent to a destination IP address and port.
There is no guarantee that it will get to the recipient, nor is there any guaranteed way of knowing whether the recipient received the datagram or even if the recipient exists.

TCP has connections but UDP doesn't
However Windows likes to pretend that it does.

UDP connections

The villain in all this is the Gnat Socket routine Connect_Socket

```
procedure Connect_Socket (Socket : Socket_Type;
                          Server : Sock_Addr_Type);
--  Make a connection to another socket which has the address of Server.
--  Raises Socket_Error on error.
```

Which allows the destination IP address to be associated with a socket. Once this socket has been connected you can send datagrams to the Server without having always to specify the Server's address

```
procedure Send_Socket (Socket : Socket_Type;
                       Item   : Ada.Streams.Stream_Element_Array;
                       Last   : out Ada.Streams.Stream_Element_Offset;
                       Flags  : Request_Flag_Type := No_Request_Flag);
```

UDP connections

We used this because it seemed convenient for the socket to hold the server's IP address so that it didn't need to be specified every time a UDP datagram was sent.

And it worked under windows!

We naively thought that all Connect_Socket did was store the server's IP address in the socket structure.
After all – UDP is connectionless, what more could it do?

Well a lot more –
and this extra only works under Windows!

UDP connections

Connect_Socket uses NetBios to attempt to determine whether or not the Server specified by its IP address exists.

Unfortunately NetBios is a Windows protocol that Linux only supports if an add-on has been installed and OSX doesn't support it at all.

The solution is easy.

Don't use Connect_Socket.
Remember the Server's IP address in your own structure and use the send variant that requires the destination IP address

```
procedure Send_Socket(Socket : Socket_Type;
                      Item   : Ada.Streams.Stream_Element_Array;
                      Last   : out Ada.Streams.Stream_Element_Offset;
                      To     : Sock_Addr_Type;
                      Flags  : Request_Flag_Type := No_Request_Flag);
```

UDP connections

If you need to know if the destination exists then use a protocol.
Send the destination a datagram and check the response.
Don't rely on the system to do this for you.

Lesson learnt:
Just because a package offers a capability doesn't mean that it works on all platforms.

4. TCP/IP

SkyTrack acts as a TCP/IP server for Stellarium to connect to.

A task opens a port using a TCP/IP socket and waits indefinitely for connections.

```
procedure Accept_Socket (Server  : Socket_Type;
                         Socket  : out Socket_Type;
                         Address : out Sock_Addr_Type);
```

When Skytrack was shutdown it closed the socket.

Under Windows this released the task waiting on the Accept with an exception.

But not under Linux.
Under Linux the task waiting on the Accept was NOT released.

TCP/IP

The Solution is to use Socket Selectors.

```
Net.Set (R_Socket_Set, The_Listener.The_Socket);
Net.Check_Selector (Selector     => The_Listener.The_Selector,
                    R_Socket_Set => R_Socket_Set,
                    W_Socket_Set => W_Socket_Set,
                    Status       => The_Status,
                    Timeout      => The_Select_Timeout);
```

Wait until something becomes available on the socket and then accept it

When SkyTrack is shutdown it aborts the selector which releases the task waiting on Check_Selector returning Status = Aborted

```
procedure Abort_Selector (Selector : Selector_Type);
```

TCP/IP

Lesson learnt:
Packages can behave differently on different target platforms.

5. Other difficulties

- Who am I?
- Am I the only one?
- Where shall I write?

Who am I?

How should we remember settings and configurations?

Under Windows we used the registry.
To be platform neutral we should write these either as an .ini or XML file.

But:

- Where should configuration files be placed?
- What should the file be called?
- Who am I – What am I called?

Could be hardcoded.
After all we know the name of the program.
However this is a poor solution for code that may be shared between many programs.

Unfortunately there doesn't seem to a Platform neutral method of finding out what the currently executing program is called.

Some have suggested that

Ada.Command_Line.Command_Name

returns the name of the program.

However it is relatively easy to demonstrate that this is not the case.

It returns the command used to start the program.

Which is not the same thing at all.

Because there is no Ada way of finding out the name of the program we had to make our own package to provide this function and then make three implementations.

One for each platform.

Likewise we had to implement a target specific function to return whether or not the current program was already executing.

We also had to write OS specific code to decide where to place application data.

This also required the program to know what operating system it was running on.

Conclusion

We were successful – StarTrack is available on all three of our chosen platforms with over 99% of the code platform independent.

But...
Unfortunately NOT 100%
We think that routines are missing from the Ada library.

And...
Gnat.Sockets was far from ideal.

1) Poor design (based on Berkeley sockets?)
2) No support for IP6

We think that networking should be part of the Ada language.

END

Questions?

Demonstration : Using SkyTracker

Apéro – Generously sponsored by Paranor AG