Patent US 6851 104 B1: Awarded Feb 1st 2005, Submitted 2000

**System and Method for Programming Using Independent and Reusable Software Units.**

**Part 2: Claims 3-8**

As described in part 1, the patent describes a way to connect up independent software building blocks (claims 1, 2). At the lowest level, the primitive blocks each consist of a single method (a function accepting input and providing output). Claim 9 then describes a way to inherit structure when defining new components by adding/deleting/replacing inherited elements for reuse.

Claims 3-8 build on claims 1,2 to describe how values flow along connectors between the building blocks from input to output ports. This is known as a dataflow paradigm. This document will concentrate on the prior art for this paradigm, specifically narrowed to the context of hierarchical component systems (claims 1-2). If I did not narrow the scope down in this way, I could choose from a vast array of work – it is a well-known approach and there is nothing specifically new in the patent in that regard.

**Prior Work**

In part 1, I established that Darwin and ROOM described claims 1-2, and that ROOM (same model as Darwin) covers patent 9.

Darwin (see [FORMAL] paper) actually started off with a dataflow paradigm where messages passed from output to input ports. This is a natural expression of an electronics metaphor, analogous to how electronic signals pass over wires between output and input pins of integrated circuits. Darwin refers to input and output ports as “require” and “provide”. This is historical – they chose these words in preference because they had the same amount of letters each, allowing the programs to line up on the screen when they were writing the programs!

Darwin later changed to the more general service oriented paradigm. In this document, I will look only at the dataflow paradigm. ROOM used a combination of both synchronous (service calls) and asynchronous (actors), but not the dataflow paradigm specifically.

**Synchronous versus Asynchronous**

Darwin used an asynchronous model where the sender of a message can “fire and forget”, and get on with other work while that message is being processed. In a synchronous model, the sender must wait until the receiver has dealt with the message and the sender receives a value back in return.

The patent text discusses that it always uses a synchronous approach. Two points need to be made here. Firstly, the notion of output gates appears to conflict/overlap with the notion of synchronous return values which are really another kind of output. It is difficult to see why the patent includes both, and Darwin (particularly in its early forms) was parsimonious in this regard.

Secondly, asynchronous systems can be made to be equivalent to synchronous systems and vice versa. In the former case, you simply wait asynchronously for the “synchronous” return value. This is a common technique used in enterprise message queuing systems. In the latter case, you return a dummy result synchronously, and then do the work rather than doing the work before returning a result. In other words, people tend to mix and match the two approaches regardless of whether the underlying platform focuses on one over the other.

**Analysis of Patent Claims**

|  |  |
| --- | --- |
| Patent claim | Comments and Prior Art |
| **Claim 3** |  |
| The software ensemble according to claim 2 | i.e. this builds on a hierarchical component model |
| Where the model of the ensemble executive is defined as a model of a software unit augmented with a structure function... | Taking out the tortured language (model of a model?) this really just adds one thing to claim 2 – an order function which indicates in which order actions are invoked. From the claim text itself, it is not clear what the actions are – are they the methods from before, or actions attached to gates?  I have spent a long time thinking about what this ordering could possibly mean. It really isn’t clear. It cannot mean the ordering of methods that is naturally described by the connections in the system, surely? If it did, the claim is “stating the obvious” – i.e. an output is sent to inputs which connect to it thereby invoking associated actions. This is the heart of all dataflow systems – they mimic the flow of values along wires. |
| **Claim 4** |  |
| The ensemble executive according to claim 3, wherein a channel is a 3-tuple defined by: {(i, g), (j, g), (dF, rF)}... | Each channel/link between gates can have a forward and reverse filter, which is a transformation for a value through a channel. It could be used for converting from Celsius to Farenheight.  The use of the reverse transformation is described in claim 7. This transforms any return value received synchronously.  In Darwin, the forward transformation would be achieved by wiring in a converter component., or none if no conversion is required. As such, Darwin copes with the forward part by applying a more general model. The reverse transformation would be achieved by creating a separate path for the return value, and wiring in another converter component. |
|  |  |
| **Claim 5** |  |
| The software ensemble according to claim 3, wherein an input gate and the channels determine the methods invoked at one or more... | This is simply saying that a value output from a gate into a link will invoke a method associated with an input gate at the other end of the link.  This is how Darwin worked, but in a sense there is no other option. If you transmit along a wire, the value is received at the other end. |
|  |  |
| **Claim 6** |  |
| The software ensemble of claim 3, wherein the output gate of a software unit and the channels determine the messages to be sent through the output gates of the software ensemble. | This is quite unclear. It may be saying simply that the channels/links/wires determine which messages are sent to other gates they are wired to.  Of course, in Darwin, the connections determine how the values propagate also. Claims 3, 5, 6 could really be simplified into a single “outputs flow to connected inputs” description. |
|  |  |
| **Claim 7** |  |
| The software ensemble according to claim 4, wherein when a value x is sent from g(i) to gate g(j) by a channel ((i, g(i)), (j, g(j), (dF, rF)) the... | i.e. we send an output, it is received by another input, it performs an action, and then send back the (transformed) return value.  Darwin used an asynchronous model which had no explicit return value in this way – it was “fire and forget”. However, you could easily wire up a return path from the output gate back to the original sender, and pass the return value this way. It is commonly the case that synchronous invocation was emulated in Darwin using two asynchronous channels. |
|  |  |
| **Claim 8** |  |
| The software ensemble according to claim 7, wherein when the receiver software unit is an ensemble, the value dF(x)... | i.e. if a value x is passed down a connection, the value dF(x) arrives at the other end. In Darwin, a separate converter component was used for the same effect. |

In summary, the claims describe how the dataflow paradigm operates where values from output ports propagate to input ports (possibly transformed) and how return values are sent back (possibly transformed). It does this in the context of a hierarchical model, but curiously the claims do not address this in any way, except to state by fiat that this builds on claims describing a hierarchical model.

The claims do not address the conceptual overlap between the return values and the notion of an output gate.