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

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

**Part 1: Claims 1, 2, 9**

**Patent Description**

(This document focuses on claims 1, 2, 9)

This patent describes a way to connect up independent software building blocks (components) to create composite components, establishing a compositional hierarchy (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 describe how values flow along connectors between the building blocks from input to output gates. This is known as a dataflow paradigm.

The basic defence I am using against this patent is to demonstrate that all of its structures and techniques were comprehensively described in academic and industrial literature in the 1990s and have been in industrial use for nearly 20 years. In particular, two closely related systems that cover the patent are Darwin and ROOM.

**Background of the Patent and Prior Work**

The academic field of computer science that covers this area is architecture description languages (ADLs). An ADL describes how to connect up components in a structural fashion, and keeps the structure of a system separate from the logic of its computational units. This is an “electronics circuit board and chips” metaphor applied to software. Applying this at multiple levels creates a compositional hierarchy.

A key advance of ADLs over earlier work is the presence of explicit connectors, which wire up component instances together, like tracks in a circuit board.

**Overview of Prior Art**

The papers and books I will use are as follows, and full details of these references are at the end of the document. There are many others but I have tried to keep the list small and focussed.

[REX] 1992  
This describes Darwin, an ADL for configuring distributed systems, and its runtime executive called Rex whic h was later superseded by Regis. I will show that it describes a superset of patent claims 1, 2. The second document will show that it covers claims 3-8.

Distributed systems are a superset of the non-distributed systems that the patent addresses. The patent also alludes to its applicability for distributed systems in “XVII. Message Breaking”. Further, [KOALA] is a variant of Darwin used at Philips showing the technique also works well for non-distributed systems.

[FORMAL] 1995  
This is a formal description of Darwin, which describes the theoretical underpinnings of the hierarchical ADL approach. I have included this because the patent uses a formal description and some of the diagrams in the patent are virtually identical to diagrams in this paper.

[ROOM] 1994  
This describes ROOM, an ADL and graphical modelling toolset which was sold to industry by a company called Objectime for developing real-time systems. It uses a Darwin-like model, but was the first to introduce structural inheritance (claim 9). ROOM provided a comprehensive graphical modelling tool. The ROOM program suite was sold to Rational Software in the 1990s and subsequently purchased by IBM.

[ROOM2] 1996  
This invited paper discusses ROOM, and points out how similar it is to Darwin.

**Ancillary documents (covering the patent body rather than the claims per se):**

[KOALA] 1996  
This describes a Darwin variant called Koala, create at Philips to produce software for their televisions. This shows that the approach was being used in an industrial setting to create non-distributed programs well before the patent was lodged.

[EVOLVING] 1990  
This is a description of an evolution approach in the precursor of Darwin, covering the general claim in the patent body regarding runtime software evolution. The patent uses almost identical operations to describe runtime structural evolution.

[SAA] 1998  
This describes a Darwin-based graphical modelling tool which covers the modelling tool that the patent body claims would be possible using its techniques. Evolve uses the same techniques. This also refers to Darwin’s similarity to ROOM.

**Similarity of Concepts in Darwin and ROOM**

Darwin and ROOM describe the same underlying approach, and both sides were aware of the close similarities.

In Bran Selic's paper: "Modeling Real-Time Distributed Software Systems" [ROOM2] in 1996.  
  
*"Although there is a number of excellent architectural definition languages defined (e.g., [3]), the one that is closest in spirit to ROOM is the Regis/Darwin system developed at the Imperial College in London [6]. The other efforts are generally not targeted at real-time applications (which have stringent demands on time and memory) nor are they oriented towards code generation (hence, they are susceptible to architectural decay). There are, however, several significant differences between Regis and ROOM. Most notable, perhaps, is that ROOM has incorporated the object paradigm and can take advantage of its features at the highest levels. For example, entire architectures can be subclassed resulting in a very effective form of reuse at the highest levels of abstraction. In addition, ROOM provides a powerful behavioral modeling mechanism for dealing with reactive systems. Furthermore, ROOM’s ability to model dynamic structures using multiple containment allows for a dynamic hierarchy as opposed to a purely static tree-structured one. On the other hand, Regis is a more open system that flexibly supports heterogeneous objects and environments in its specifications. We are currently working towards establishing a collaboration with the Imperial College group to incorporate the best of the  
two research thrusts"*

In other words, ROOM is conceptually the same as Darwin (claims 1, 2) except that it provides subclassing and structural inheritance (claim 9).  
  
The other paper is a Darwin-related paper, by the Imperial professors and a PhD student, called "The software architect's assistant -- a visual environment for distributed programming" in 1995 [SAA].  
  
The text in that paper says:  
  
*"ROOM [21] and its associated CASE tool ObjecTime is targeted at distributed real-time  
systems. It embodies many of the same concepts as Darwin/Regis, including the separation of system structure from its behaviour. As in Darwin, the system architecture is specified in terms of hierarchically-structured components which communicate through message ports.  
The dynamics of a system is modelled using a variation of Harel’s StateCharts [22]."*

In general, academics are reluctant to talk this way unless the systems are extremely similar (and there is a high level of respect between the parties). It is much more common to emphasise differences and weaknesses in a competing model, for obvious reasons.

**Terminology**

The patent uses a number of unusual terms which are not standard. I will use the standard terms.

|  |  |
| --- | --- |
| Standard Terms for ADLs | Patent Terms |
| Component | Connecton, sometimes Block |
| Component instance | Connecton |
| Connector | Link or Channel |
| Provided or Required Service, also Port | Gate |

The lack of distinction in the patent between component and component instance is limiting, and the use of the word “connecton” which looks very similar to “connector” is confusing. I will use the standard terminology as far as possible.

**Analysis of Patent Claims**

|  |  |
| --- | --- |
| Patent claim | Comments and Prior Art |
| **Claim 1** | This claim covers the “static” structure of the system. i.e. the design of the components and the connectors between the component instances. |
| 1 . A software ensemble stored on a computer readable medium and executable by a computer, the software ensemble comprising: | This phrase describes a general computer program. |
| A plurality of software units each software unit of said plurality of software units including a method and data: | This is quite general, and refers to the collection of components and component instances making up the structure of a hierarchical component model. This is the basis of Darwin described in [REX] and [FORMAL]. It is also the basis of [ROOM]. |
| An executive software unit including links between said plurality of software units | This refers to connections between component instances. Described in [FORMAL] in section 2:  “Darwin allows distributed programs to be constructed from hierarchically structured specifications of the set of component instances and their **interconnections**”.  The use of connectors is perhaps the defining feature of ADLs. Described also in detail in [REX]. |
| Wherein a software unit of the software ensemble is described by a model M, given by  M = (inGates, {inSign},{a}, Q, q, outgates, {outSign}, {outFunction} | NOTE: {} indicates a set of elements.  This says that the system comprises a number of method-level components {a}, each of which has an initial state. Each component specifies the parameters it accepts as input {inSign} and what is produces as output {outSign}. The system starts with a state q (the data values), and each function in the set of {a} has a specified set of input parameters as a signature {inSign} and a set of output parameters {outSign}.  This is the same as the Darwin/ROOM model. [REX] figure 4 shows a primitive component with ports (gates) with provided (input) / required (output) polarity. This has local state as per q in the patent description.    Also, compare and contrast figure 1 of [FORMAL] with figure 2 of the patent. They both describe the same sort of entity.  Figure 1 from [FORMAL] showing a primitive component:  C:\Users\andrew\Desktop\fig1.jpg  Figure 2 from the patent showing a primitive “connecton”:    Let us now turn to composite structures:  [FORMAL] figure 2 shows how component instances are connected together to make a composite component. This is the foundation of the hierarchical approach. Compare and contrast with figure 8 in the patent, they are almost identical.  Figure 2 from [FORMAL] showing a composite component:    Figure 8 from the patent showing a composite “connecton”:      These are slightly different examples, hence the lack of an “output” in the latter figure. However, it makes the point that the patent describes a subset of the pre-existing Darwin language.  In other words, the Darwin configuration language can model the patent system completely. Like the patent, Darwin initially worked at the method level also in a dataflow paradigm. |
| Where inGates... | Just describes the above terms. |
|  |  |
| **Claim 2** | This builds on claim 1, using similar language, and describes the runtime state of the program. i.e. the changeable state while running, which starts as per the initial design (claim 1) which can then deviate. |
| 2 . A software ensemble stored on a computer readable medium and executable by a computer, the software ensemble comprising: | As before, just describes a general computer program in all its forms. |
| A plurality of software units each software unit of said plurality of software units including a method and data: | As before. |
| An executive software unit including links between said plurality of software units | As before. |
| Wherein the software ensemble, E, is described by  E = (inGates, {inSign}, , M, outgates, {outSign}, {outFunction}) | This logic description references M of claim 1. It is basically saying that a running program has an ensemble executive that keeps track of the (possibly changing) structure of the system  The ensemble executive is a reference to the runtime of the system which manages the configuration of component instances and connections. Such a runtime is described in [EVOLVING] as the Darwin configuration manager. The purpose of this was to track the connections during runtime, as well as other aspects.  The operations established by [EVOLVING] for the Darwin executive are described in section B of that paper as:  create(component instance), remove,  link, unlink  In an almost identical fashion, the patent in section XVIII describes the operations as:  addConnecton(component instance), removeConnecton, replaceConnecton  link, unLink  Perhaps the only difference is in the use of replace, which in Darwin was synthesised by a delete/add.  [REX] describes a variant of the Darwin runtime called Rex. [FORMAL] describes a later, improved version, called Regis. |
| Where is... | Just describes the above terms. |
|  |  |
| **Claim 9** | Using structural inheritance to reuse existing components |
| The software ensemble according to claim 2, wherein structural inheritance is utilized to build new software ensembles from existing software ensembles. | The notion of structural inheritance was first developed in [ROOM] and was a key selling point of the ROOM toolset. ROOM was an ADL where structure could be inherited and added to, deleted or overridden (replaced). The patent description of structural inheritance is almost identical to the description in the [ROOM] book figure 9.5. A similar example is used in the patent in figures 11 and 13.  [ROOM] figure 9.5 showing structural inheritance of 2 subclasses of a parent class:    Patent figures 11 and 13 showing structural inheritance of a subclass (13) from a parent class (11). Figure 13 adds the NOT instance and 2 connectors, in the same way that figure (b) above adds the C instance and b3/b4 connectors.    The patent only allows for single inheritance, as per the Smalltalk inheritance approach. ROOM also had this exact limitation. This is perhaps not surprising – both ROOM and the patent describe Smalltalk systems. |

In summary, the claims can be broadly divided into the following categories:

1. Claims 1 & 2 which define hierarchical component structures with connectors. This is covered by Darwin [REX] [FORMAL], the original ADL which established the field and Regis the Darwin runtime executive. Koala [KOALA] and ROOM [ROOM] are other examples which were used in industry well before the patent filing date of 2000.
2. Claim 9 which describes the use of structural inheritance to effect component reuse by allowing connectors and component instances to be added, deleted or replaced in a subclass. This technique is identical to that described [ROOM] and was in common industrial use in the early 1990s and beyond.
3. Claims 3-8 which describe a dataflow paradigm where values flow prescriptively along connectors. Darwin started with this paradigm and shifted to a more general service-oriented paradigm later. Part 2 of the defence analyses claims 3-8.

**References**

[REX] 1992  
Configuring Object Based Distributed Programs in REX  
Jeff Kramer, Jeff Magee, Morris Sloman, Naranker Dulay  
Distributed Computing Systems, pp.187–205, IEEE Computer Society Press

[FORMAL] 1995  
Magee, J., Dulay, N., Eisenbach, S., and Kramer, J. 1995.  
Specifying Distributed Software Architectures.  
In Proceedings of the 5th European Software Engineering Conference.  
Lecture Notes In Computer Science, vol. 989. Springer-Verlag, London, 137-153.

[ROOM] 1994  
Real-Time Object-Oriented Modeling  
Bran Selic, Garth Gullekson, Paul T. Ward.  
John Wiley & Sons; 1 edition (April 22, 1994), ISBN-10: 0471599174

[ROOM2] 1996  
Modeling Real-Time Distributed Software Systems  
Bran Selic, Invited talk, Proceedings of the 4th WPDRTS

[KOALA] 1998  
Koala, a Component Model for Consumer  
van Ommering, R.  
Electronics Product Software,  
In Development and Evolution of Software Architectures for Product Families, LNCS 1429

[EVOLVING] 1990  
The Evolving Philosophers Problem: Dynamic Change Management.  
Kramer, J. and Magee, J.  
IEEE Trans. Softw. Eng. 16, 11 (Nov. 1990), 1293-1306.  
DOI= http://dx.doi.org/10.1109/32.60317

[SAA] 1995  
The Software Architect's Assistant-a visual environment for distributed programming.   
Ng, K., Kramer, J., Magee, J., and Dulay, N.  
In Proceedings of the 28th Hawaii international Conference on System Sciences (Jan 04 - 07, 1995).  
HICSS. IEEE Computer Society, Washington, DC, 254.