WORKING NOTE AC124

Alan Creak
1998 June 10

PDL MEETS MIDDLEWARE

In recent years there have been several publications in which the merits of middleware architectures (CORBA, etc.) as communication and coordination mechanisms in software for manufacturing systems have been explored. As this is precisely the area for which PDL was designed, it is obviously of interest to ask how they are related. This note is stimulated by a recent short paper[1] in which a convenient brief survey of the use of frameworks in a CIM context. It isn't a particularly exciting paper, but does describe a few components of the system in terms which make it easy to match to PDL.

THE CIM FRAMEWORK, WITH NOTES ON RELATED PDL FEATURES.

The CIM framework is intended as an industry standard basis for constructing CIM systems. The usefulness of such a standard is argued as a means of supporting systems which can easily be changed, and in which different programmes, independently developed, can readily be combined; agile manufacturing systems are specifically mentioned. That all sounds very like PDL. They call them Manufacturing Execution Systems.

A framework is defined as a "collection of specifications" of software "components". The specifications are definitions of interfaces, written in an interface definition language (from the Object Management Group); everything in PDL is described in terms of the messages which it can send and receive. It is interesting to read of "translation layer code to convert the CIM-level instructions into instructions executed by the tool control software", as this sort of translation is just what PDL already provides for implementing instruction-set interfaces with different underlying software in different ways.

Behaviour specifications for the components are also provided in the CIM framework; it isn't stated how these are formulated, but in PDL corresponding material appears in the Machine Type Database[2], where it is intended to be used in the higher planning activities as a part of the system design function, and also by a diagnostic system in dealing with production failures.

The product is identified as a *callable* framework (as opposed to a *calling* framework), providing services (as opposed to scheduling programmes). PDL is neutral so far as such matters of precedence are concerned; it supports the communication mechanism, and the precedence is determined by the behaviour of the code.

To use the framework, software for a specific type of machine must be made to look like an appropriate corresponding interface specification by providing suitable interface code. It is then in a form in which it can be connected to a CORBA system, which it can then use for communication with other machines similarly connected to the system and with other CORBA services.

PDL AND THE CIM FRAMEWORK.

There is clearly a great deal of common ground between PDL and the CIM framework. This can be seen as a natural consequence of two features which PDL shares with components of the framework.

First, there is the shared concern of PDL and of object-oriented methods in general for careful and precise definition of the interface. The reason in both cases is much the same : given the static interface definition, the implementation behind the interface can be changed without affecting the connection to other parts of the system. In the case of the object-oriented systems, this policy is adopted to maintain the principle of encapsulation[3], an important feature of object-oriented programming and design. In PDL, the policy arose naturally as a means of ensuring the robustness of an implementation against change[4].

In addition, PDL and object request broker middleware systems agree in their concern that the function of an interface should be carefully defined, though in this matter the reasons are somewhat different. The middleware interfaces must be specified so that requests for service can be directed to an appropriate destination[5], while the primary purpose of the PDL definitions is as an aid to system design[6], with the secondary function of providing information for failure diagnosis.

Given this common ground, it is perhaps interesting to inquire how the two techniques for software construction might work together. While a number of possible combinations could be imagined, here I

discuss three which together seem to cover most of the ground. (I cannot justify that statement by any sound analysis, but it feels right.)

- **A PDL system connected to a CIM framework.** We imagine that one component of those which comprise some integrated system is implemented using PDL. In this case the language to be used in the component's interactions with the rest of the system is precisely defined by the CIM framework standards, and can be used as the definition of the set of messages which the PDL software must produce and accept.

- **PDL used to implement the interface.** We imagine that a PDL module is used as the "translation layer code to convert the CIM-level instructions into instructions executed by the tool control software" (quoted from the discussion by Doscher and Hodges[1]). This is PDL's home ground.

- **The CIM framework used to implement PDL communications.** We imagine a large-scale PDL system in which the CIM framework is used to link the modules together. Such an architecture has the potential of simplifying the PDL communications significantly, by providing a ready-made standard for the device addresses which must appear in the Machine Database, and which are used in communication between PDL entities. A further advantage is that the use of a standard convention of this sort will in principle make it possible to link easily to other systems, perhaps operated in different parts of the organisation, and not necessarily implemented in PDL. This opens the path to more effective CIM – which is, of course, the whole point of the CIM framework.

REFERENCES.

1 : D. Doscher, R. Hodges : "Sematech's experiences with the CIM framework", *CommACM* **40#10**, 82-84 (October, 1997).

2 : Reference (7), page 27.

3 : B. Stroustrup : *The C++ programming language* (Addison-Wesley, 2nd edition, 1994), page 41.

4 : Reference (7), page 4.

5 : P.A. Bernstein : "Middleware : a model for distributed system services", *Comm. ACM* **39#2**, 87 (February, 1996).

6 : Reference (7), page 33.

7 : G.A. Creak : *Information structures in manufacturing processes*, Auckland Computer Science Report **#52** (Computer Science Department, Auckland University, February, 1991).