UML for describing product-line architectures?

Perdita Stevens*
Division of Informatics, University of Edinburgh
JCMB, King's Buildings, Edinburgh EH9 3JZ, Scotland, UK

June 4, 1999

Abstract

This position paper reports some work in progress on the use of UML, the industry standard object oriented modelling language, as a notation for describing product line architectures. We draw on a great deal of previous work on notations for architectures and PLAs, and on UML as a notation for architectures and for frameworks, attempting to identify the salient points of this particular question.

1 Scope

As UML becomes more widely accepted, it is inevitable that some organisations that use product line architectures will describe them using UML as their diagrammatic language, alongside other description elements. This paper attempts to identify advantages and disadvantages of doing so. As far as possible, it tries to consider the issues specific to PLAs, and to steer clear of more general architecture description issues. For reasons of space we assume considerable familiarity with UML.

2 Is this the right question?

For example, why should we not put all our effort into inventing the optimal notation for describing product line architectures and use that?

Developing optimal notations may indeed be a useful thing to do, and this paper does not aim to discourage it. As always, the points for and against using a special purpose language must be considered in context.

*Perdita.Stevens@dcs.ed.ac.uk

However, the following factors suggest that it is worth investigating whether it is feasible to use UML, an industry-standard general purpose system modelling language.

1. The users of product line architectures – those instantiating the architecture to build a product – are increasingly likely to be familiar with UML. Familiarity with a notation is an essential ingredient in using it correctly and efficiently.

2. There are many tools supporting UML. If we want tool support for PLAs – and surely we do – it is likely to be easier to get extra functionality incorporated into existing tools than another set of tools built. Even if it were possible to get another set of tools built, using the same tool throughout the process is easier. (Current tools generally implement only a subset of UML; but this is changing rapidly as the standard matures.)

3. Although many authors (including this one) have complained about the lack of clarity in UML's semantics, in fact it is probably the best specified general purpose modelling language ever. Because UML is widely accepted by a large user community, a great deal of effort is being put into removing the remaining areas of ambiguity. There is strength in numbers.

Version 1.1 of UML has been accepted by the OMG. However, that standard contained many ambiguities which have since been dealt with by the OMG's Revision Task Force. Therefore in this paper we use the most recent version of UML at the time of writing (1.3 alpha 5)[13]. Some aspects of UML have been changed, rather than just clarified, but only one such change matters in this paper. We note that the Semantics document takes precedence over the Notaion Guide; the significance is that many uses of UML, including some suggested here, are covered by the Semantics, but not explicitly discussed or illustrated in the Notation Guide. Such cases are normally obvious extensions of things that are covered by the Notation Guide, so that the intended notation as well as its meaning is unambiguous. Several previous authors have made statements about the limitations of UML based only on the Notation Guide, without considering the Semantics: this is an error. It is, of course, a reasonable criticism of UML that it is a large language whose limits take considerable effort to determine.

3 What does the question mean?

That is, what are the criteria against which UML should be evaluated?

Let us briefly review the reasons for documenting a PLA with a diagrammatic notation. There are two main reasons:

- to help users of the PLA, that is, people instantiating it for a particular product
- to help maintainers of the PLA, that is, people who are evolving it for future use.

(Many other views are also important though we will not consider them explicitly here: participants at the First Nordic Software Architecture Workshop [3] identified as stakeholders: architects, test engineers, configuration managers, management, maintainers, developers, API users, end users and QA people.)

Let us consider the aspects of a PLA which may reasonably be documented diagrammatically. Both groups of people need to understand where the hotspots of the PLA are; what kinds of modification, omission or extension are permitted, expected or required. (There are various different kinds of hotspots: for example, instantiation of a PLA may be by composition, by specialisation or a mixture. In general instantiation by composition is easier to handle, and we concentrate on instantiation of PLAs by specialisation in this paper.) Users need especially to understand the dynamics of the PLA; how does it work to achieve its aims? Maintainers will be relatively less concerned with dynamics and more concerned with the dependencies between the parts of the architecture. Many of these concerns are common to all architecture development: the chief difference is the concern for identifying hotspots explicitly.

It is clear that a diagrammatic modelling notation can only be a partial solution to the problem of documenting PLAs. For example, a concise textual description of the intention of the PLA is indispensable. Nevertheless there is normally some diagrammatic component and it is for this role that we examine UML.

We should deal with a possible objection: in the sense in which the term "architecture" is used in most of industry, it would be absurd to claim that UML cannot be used to model it. The inventors of UML are also enthusiastic advocates of "architecture-centric development" and descriptions of architecture are integral parts of methodologies developed alongside UML. The Unified Process book [8] uses standard UML; but it does not address

PLAs, and its view of architecture is (simply) as a strict subset of the model of the system, containing only the "architecturally significant" elements. (Note that the Unified Process is *not* standardised by the OMG: UML is a modelling language intended to be free of process.) Catalysis [6] considers frameworks explicitly, using a conception of architecture with explicit connectors. However it uses a version of UML which is different in many ways from the standard, so we will not consider it further here.

Related questions There are two related questions whose answers may be helpful:

1. What are good notations – and how good is UML – for describing architecture (as understood by the architecture community, which arguably has more stringent requirements than most of industry at present)?

2. What are good notations – and how good is UML – for describing frameworks?

Work on Question 1 has been done by [12, 10, 2, 7], though doubtless more remains to be done.

Without duplicating their work, a couple of comments are:

1. [12, 10] use the UML metamodel element Class to represent components (and connectors) in an architecture, acknowledging that this is not a perfect fit but not explicitly considering alternatives. The UML metamodel element Subsystem may well be more suitable. Like a Class, a Subsystem is a Classifier and a GeneralizableElement, so a Subsystem may realise interfaces, may be refined, may take part in collaborations etc. It is also a Package – that is, a namespace – with the ability to be instantiable or not; and UML builds in the idea that a Subsystem may have specification and realisation parts (cf e.g. [1] for use of such a feature in architectural modelling), with collaborations to demonstrate how the realisation matches the specification. On the whole, this seems like a more natural match for an architectural component, and less likely to lead to confusion. In other circumstances Component itself may be worth considering, although UML's intention is that a Component is a separately distributable, run-time replaceable entity. Of course this choice points up one of the disadvantages of using a general purpose rather than a special purpose language: more choices to make.

2. Section 6.2 of [10] cites as a disadvantage of UML that it forces messages shown in a collaboration diagram to be ordered, which is inappropriate if in fact the participants in a collaboration may have their own threads of control. This is incorrect; UML provides quite sophisticated and extensible notation for concurrent activity, which seems adequate for their purposes.

These papers naturally do not address the special issues of PLAs, in particular the need to model hotspots.

A good survey of approaches to Question 2 is Chapter 5 of [9]. The author does not know of any previous work addressing "pure" UML for this purpose. [6] (which heavily modifies UML) is relevant; the draft paper [4] says that it is proposing an extended version of UML, but in fact many diagrams in the paper are in OMT, not UML, and the proposal is not related to the UML standard. In any case the issues are not identical, for example because in the case of PLAs there is less need for potential readers to be able to evaluate whether to use a PLA fast; they are not likely to be considering large numbers of competing PLAs.

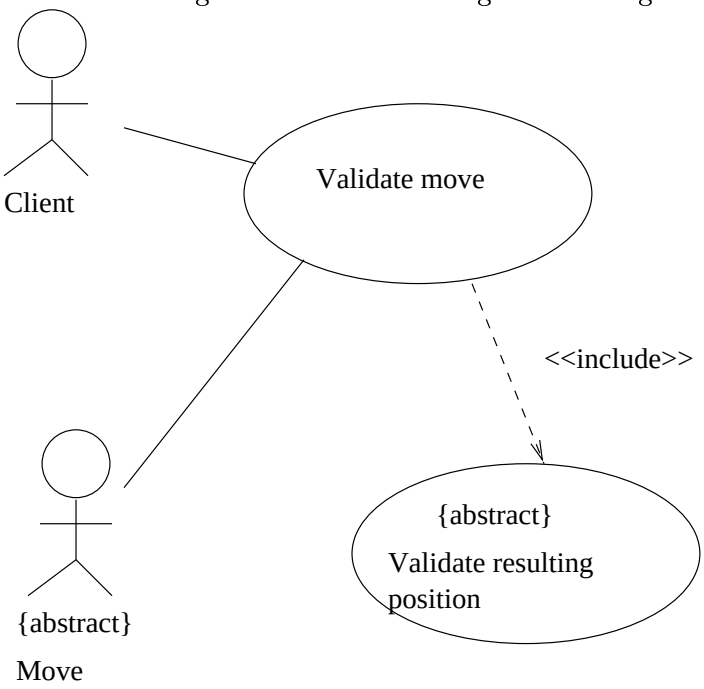
4 Evaluation

How can UML as a notation for product line architectures allow the representation of the variability which is and is not permitted? We will discuss separately modelling the externally visible behaviour of a system, its static structure, the collaborations and interactions, the state diagrams, and the deployment of the system on hardware.

Externally visible behaviour This includes the coherent tasks which figure in the requirements of the system; or indeed of any other classifier, for example a subsystem or a class. I am grateful to a referee for the reminder that use cases can be designed to guide application designers to hotspots. For example, in the case of a PLA for a family of computerised games, it might be the case that part of the functionality of a Game component, whose primary responsibility is to enforce the rules of the game, is provided by the PLA and part must be provided by the product designer; Figure 1 illustrates.

Static structure At the simplest level we may add user-defined stereotypes to apply to the model elements, documenting the PLA's constraints on

Figure 1: A use case diagram showing variability



whether the elements may/must be replaced. This form of extension is mild within UML, and permits for example the use of different graphic elements to represent differently stereotyped elements. It does not automatically allow for automatic checking of correct use, however.

Thus we may document the structure of a PLA by recording the components and connectors which are provided, using stereotypes to indicate where extensions are permitted or expected. How though can we document the relationships among the elements, particularly the "knows about" or association relations? The underlying question is what it should mean to record an Association between two UML Classifiers. The options are:

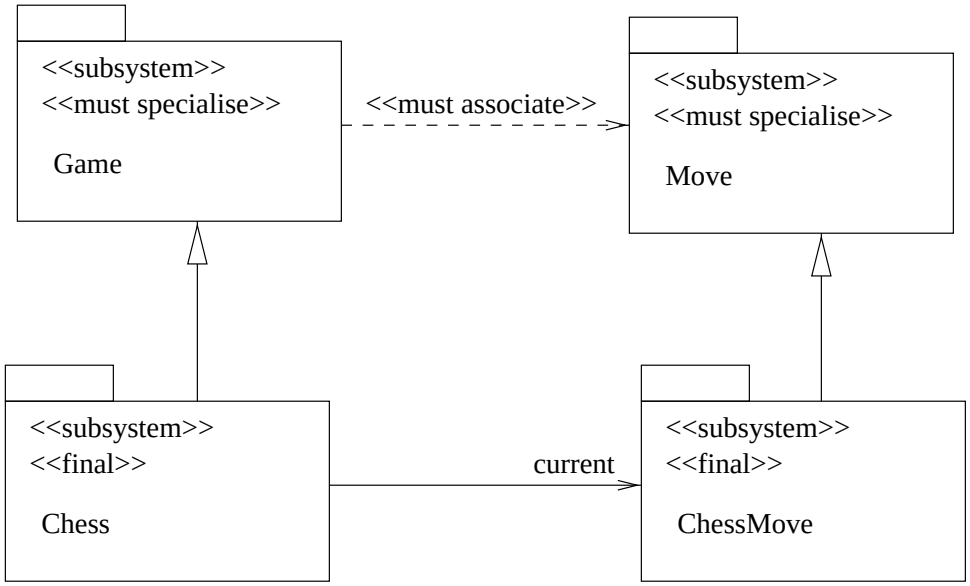
1. the dynamic view of association: classifiers A and B are associated if an instance of A may be associated with an instance of B. For example, classes A and B are associated if an object of class A sends a message to an object of class B.

2. the static view: classifiers A and B are associated if there is a data dependency between them; roughly speaking, if each instance of A has a "data slot" for an instance of B or vice versa.

The UML standard itself does not currently specify what the intended meaning of an association is (hence the discussion of the issue in the author's textbook [11]). However, James Rumbaugh, one of the originators of UML, explained in a recent article [14] that 2 is what is intended. This is reasonable from the point of view of the drawer of a class diagram; so for the purposes of this paper we assume that this is how UML associations should be interpreted, and hope that the next official standard will incorporate this view. It has, however, some unfortunate consequences for the PLA designer. Most notably, it is arguably incorrect to show an association which is not implemented in the classifiers, even if a requirement of correct use of the PLA is that the instantiations of the classifiers should always be associated. On the other hand, since an Association is a GeneralizableElement, it may in principle be abstract, and we may think that an abstract association is what we need. The meaning of an abstract association is not as clear as it might be, since elsewhere in the standard an Association is said to represent a relation, and does not itself carry an implementation. The related question of whether an association can in general play the role of a connector is an interesting one not fully resolveable with respect to the current specification.

The alternative is to show dependencies between components, with a dotted arrow. The designers of UML intended that dependency should be free of connection with particular instances, which in this case is not true.

Figure 2: Part of a static structure diagram for a product using a PLA



Rumbaugh cited this as probably a mistake for approximately this reason, and we may expect the restriction to go away in future. The best we can do is to use a stereotype on dependency to indicate the particular kind of dependency, namely one that should turn into an association at lower levels. However, doing so means that CASE tools will not be able to check that a use of the framework is correct.

Figure 2 shows an example of part of a static structure diagram showing two components Game and Move of a PLA, modelled as subsystems, and specialisations of them. It is standard UML with user-defined stereotypes "must specialise", "may specialise" and "final".

We will probably want the whole PLA to have its own package – and hence namespace – separate from, but imported by, a package containing the models of the system which uses the PLA; this is routine in UML but not shown here.

Just what associations are legal instantiations of our abstract association? A rather obvious point, sometimes known as the "animals eat food, cows eat grass" phenomenon is that specialisations have to be compatible. Compatibility is partly a semantic and partly a type-theoretic notion; ongoing work investigates it type-theoretically but that is outside the scope of this paper. The underlying point is that of course the PLA does not consist

solely of the reusable assets; it also includes rules about how it should be used; for example, of what it means for specialisations to be compatible. An unsolved question which UML does not address is: How is this requirement for compatibility to be reflected in the diagrammatic notation? It can of course be represented by adding OCL constraints to the diagrams, but this is not ideal. The most important point is that representing semantically important information in OCL runs the risk of its being accorded less importance by the reader than the more obviously salient diagrammatic information. More specifically, OCL is vulnerable to criticism as a specification language. Its admirable aim is to be a language which is easily usable by any developer whilst still being formal. There is no consensus that it achieves the first aim; for example, the decision to automatically flatten all collections (so that, for example, it is not possible to describe a set of sets in OCL), which was apparently taken in the interests of simplicity, seems confusing to some. On the other hand the language is not formally defined – it has a formal syntax but no formal semantics – so it does not completely achieve the second aim either.

Interactions and collaborations A UML collaboration diagram documents the relationships between instances of types, or more usually (at the specification level not the implementation level) between the roles that may be played by instances of types. An interaction may be shown on a collaboration diagram by adding messages. The resulting interaction diagram is normally used to demonstrate how the functionality documented in a use case diagram is realised by the designed system. A certain level of consistency between the static structure and the interactions modelled for a system is ensured, because in UML both are views of the same model. The use of these diagrams for documenting architectures is discussed in [12]. Here we are interested in documenting hotspots; the main concern is the ability to show abstract collaborations and interactions which may be instantiated or refined later.

There are two possible approaches. The first is to observe that UML does not impose any completeness condition on a collaboration or interaction; so at an informal level it is in order to use stereotypes again to describe the parts of a collaboration or interaction documenting a PLA where refinement or instantiation might be required.

More satisfactory particularly for documenting the use of a PLA, e.g. in exemplars, is to observe that in UML a Collaboration is a GeneralizableEle-

ment[1]. The intended semantics is what we would wish – the specialising Collaboration achieves a specialisation of the task achieved by the more general Collaboration. However the semantics of such a Generalization are not specified by UML, and this might prove challenging, given that Messages in an Interaction know their predecessors and successors and that Interactions are not themselves GeneralizableElements. A Collaboration may be represented as a dashed ellipse connected by dashed lines to the Classifiers that play the various roles in it; several such ellipses may then be related by generalisation. This is conceptually appealing; however, the notation probably makes it more useful for demonstrating the use and refinement of small design patterns or frameworks (which is what it was intended for).

Some architectures will require sophisticated documentation of real-time and synchronisation aspects. UML includes a reasonable set of features to support this, in the form of notation for timing constraints on interaction diagrams and support for different threading models. However this is still an area of active work, particularly in the embedded system community, e.g. [5].

State diagrams UML has taken these over more or less wholesale from the statechart notation; standard mechanisms suffice. For example, a classifier in the PLA (the whole system, or a component of it) may have its behaviour documented using a state diagram, which is then refined in a particular product by the insertion of more detailed state diagrams nested in the states of the PLA state diagram.

Deployment UML's deployment diagrams are a frequent source of confusion, and indeed they are much less well explained in the standard than other elements of UML. However, nodes – the UML element which represents processing elements – are Classifiers in UML, which means that they can have instances, play roles in collaborations, realise interfaces, etc. They can also contain instances of components. (There is a little remaining confusion here between the Notation Guide, which suggests that a Node can contain an instance of any Classifier, and the Semantics, which limits this to Components, the element which UML regards as having a run time existence. Comments on what is actually required would be interesting.)

Non-diagrammatic elements We have mentioned that there is naturally a need for supporting text to document things like the purpose of a

[1] This is a change since UML1.1

PLA, the details of a use case, etc., as well as any other constraints or decisions not conveniently documented in UML. There is also a need to maintain the connections between the different diagrams; for example, to show what collaboration implements a given use case. [7] recognising this suggests tables as the solution. In a computer supported environment there is an additional possibility which may be useful instead of or as well as tables: hyperlinks. The standard [13] writes (3-8):

> A notation on a piece of paper contains no hidden information. A notation on a computer screen may contain additional invisible hyperlinks that are not apparent in a static view, but that can be invoked dynamically to access some other piece of information, either in a graphical view or in a textual table. Such dynamic links are as much a part of a dynamic notation as the visible information, but this guide does not prescribe their form. We regard them as a tool responsibility. This document attempts to define a static notation for the UML, with the understanding that some useful and interesting information may show up poorly or not at all in such a view. On the other hand, we do not know enough to specify the behavior of all dynamic tools, nor do we want to stifle innovation in new forms of dynamic presentation. Eventually some of the dynamic notations may become well enough established to standardize them, but we do not feel that we should do so now.

5 Conclusions and ongoing work

This short paper describes a small part of work in progress. The interim conclusion is that it is reasonably to use UML as the diagrammatic component of the description of a product line architecture, provided that its drawbacks are understood and provided that the UML description is not the only form of documentation, but is supplemented by appropriate text tabular and other elements as necessary; however, that UML is not so well adapted to the task that other approaches are superseded.

References

[1] Dirk Bäumer, Guido Gryczan, Rolf Knoll, Carola Lilienthal, Dirk Riehle, and Heinz Züllighoven. Framework development for large systems. *Communications of the ACM*, 40(10):52–59, October 1997.

[2] Lodewijk Bergmans. A notation for describing conceptual software architectures. In *In Electronic Proceedings of the First Nordic Software Architecture Workshop*, 1998. bilbo.ide.hk-r.se:8080/ bosch/NOSA98.

[3] Jan Bosch. Electronic proceedings of the first nordic software architecture workshop. bilbo.ide.hk-r.se:8080/ bosch/NOSA98.

[4] Marcus Felipe M. C. da Fontoura, Carlos José P. de Lucena, Paulo S. C. Alencar, and Donald D. Cowan. On expressiveness: Representing frameworks at design level. Draft paper from Web.

[5] Clement de la Jonquiere. An introduction to embedded software for system-on-chip 1. A Cadence Design Systems Inc. seminar given on June 2.

[6] Desmond D'Souza and Alan Cameron Wills. *Catalysis: Objects, Frameworks and Components in UML.* Addison-Wesley, 1998.

[7] Hofmeister et al. Describing software architecture with uml. In *Proc. WICSA1*, 1999.

[8] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Software Development Process.* Addison-Wesley Longman, 1999.

[9] Michael Mattson. *Object-Oriented Frameworks: a survey of methodological issues.* PhD thesis, University College of Karlskrona/Ronneby, 1996.

[10] Nenad Medvidovic and David S. Rosenblum. Assessing the suitability of a standard design method for modeling software architectures. In *Proc. First IFIP Working Conference on Software Architecture*, 1999.

[11] Rob Pooley and Perdita Stevens. *Using UML: software engineering with objects and components.* Addison-Wesley Longman, 1998.

[12] Jason E. Robbins, Nenad Medvidovic, David F. Redmiles, and David S. Rosenblum. Integrating architecture description languages with a standard design method. In *Proc. International Conference on Software Engineering*, 1998.

[13] OMG RTF. Uml specification 1.3 alpha 5. available from uml.systemhouse.mci.com.

[14] James Rumbaugh. Modeling and design. *Journal of Object Oriented Programming*, 11(4), 1998.