A cognitively informed approach to describing product lines in UML

Corin Gurr and Perdita Stevens
Human Communications Research Centre/Software Engineering Programme,
Division of Informatics,
University of Edinburgh
Scotland, UK

Abstract

This paper applies a body of theoretical (though practically motivated and validated) work on the cognitive and semantic aspects of diagrammatic notations to the very practical problem of describing product line architectures and products based on them in UML, the industry standard modelling language. We briefly analyse the tasks that a suitable notation must support; justify using UML as a basis and discuss deviations from it; discuss issues in the light of their different semantic and cognitive characteristics; and propose a notation.

1 Introduction

As the explicit use of product lines—collections of products based on a common product line architecture—is increasingly recognised as a means of achieving business benefits, it becomes increasingly evident that tool support for the associated design activities is required. Just as we need to be able to record the design decisions made in the production of an individual product, we need to be able to record the decisions embodied in a product line architecture (PLA), in a product instantiation of that PLA, and in their relationship.

For many organisations, the choice of UML as the notation in which to record these decision is an inevitable one. The short paper [Ste99] discussed the pros and cons of this decision and showed how some of the important decisions could be recorded in UML using the standard extension mechanisms, without the need for non-standard notations. Like much related work, however, it took a naive approach to the notational choices made. No systematic attempt was made to evaluate the resulting notation or to consider cognitive issues.

In the field of cognitive science, meanwhile, there is a large body of work which discusses the significance of choices like these for the human user of diagrams. Most examples in that field have be taken either from small diagrammatic languages with well-defined, small task sets—Venn diagrams, for example—or from the unconstrained world of graphic design.

In this paper we readdress the problem of [Ste99] with the aid of this work. The contribution of the paper is two-fold: first, we provide a reasoned, consistent proposal for the solution of the practical problem of choosing a notation for product lines; second, in applying the theoretical work to a real-world example, we demonstrate its maturity in use.

The remainder of this paper is structured as follows: in Section 2 we briefly cover the necessary technical background in the fields of semantic and cognitive aspects of diagrammatic notations and of UML and its semantics; in section 3 we discuss users of a notation for PLAs and the tasks in which they should be supported. This concludes the description of the problem we address and the tools we use. In section 4 we discuss options, propose and justify a simple notation for product lines. Finally we discuss related work, conclude and discuss topics for further exploration.

2 Technical background

2.1 Intuition and Comprehensibility in Diagrams

It is often claimed that diagrammatic specification, modelling and programming languages provide natural representations which permit intuitive reasoning. Understanding how, and for what applications, diagrams actually do offer benefits to comprehensibility requires that we examine seriously the meaning and accuracy of the terms ÒnaturalÓ and ÒintuitiveÓ in this context.

Studies such as [Goo99,SY97,ZN94] have indicated that the most effective representations are those which are well matched to what they represent, in the context of particular reasoning tasks. For the purposes of this paper we assert that an ÒintuitiveÓ representation is one which is well matched. Furthermore, we assert that whether a representation is ÒnaturalÓ concerns how it achieves its intuitive matching; and (certain classes of) diagrammatic representations are particularly good at naturally matching their intuitive interpretations. Clearly, these two assertions beg the questions of

what are the intuitive meanings which an effective representation matches, and *how* diagrams achieve such matching in a natural way.

An intuitive, or well matched, representation is one which clearly captures the key features of the represented artefact and furthermore simplifies various desired reasoning tasks. To assess intuitiveness we must consider not only the (formal) semantics of a diagram, but also how well the diagram captures less formalised aspects such as conceptual grouping of elements. Such information may play no role in, say, determining the behaviour of a system, but will substantially aid comprehension of the design. This extra information andÑto a certain extentÑthe inferences a human reader might be expected to draw from it, is termed *pragmatic* content. An intuitive (well-matched) diagram is thus one for which this combined semantic, pragmatic and inferential content closely matches the truth of the represented artefact.

A recent comparison [GLS98] of typical text-based languages and diagrammatic languages was quite revealing about both similarities and differences between the two. The decomposition in [GLS98] of issues in how diagrams capture information permitted the identification, in a subsequent study [Gur99], of the fundamental issues relating to the effectiveness of visual and diagrammatic representations for communication and reasoning tasks. This latter study identified two advantages which effective diagrams exploit in naturally capturing content. The first of these is that intrinsic properties of certain diagram symbols and graphical relations enable them to *directly* capture semantic information. The second is that diagrams offer designers substantial opportunities for exploiting *pragmatic* features to capture useful information about their intentions and the represented artefact.

2.1.1 Diagrams: Direct Representations

One primary difference between diagrams and typical text-based representations is that diagrams may capture semantic information in a very direct way. That is to say, intrinsic features in the diagram, such as spatial layout, directly capture aspects of the meaning of the diagram. For example, in Figure 1 the fact that package A is contained in package C does not need to be consciously deduced from the representations of the inclusion of A in B and B in C: it is immediate.

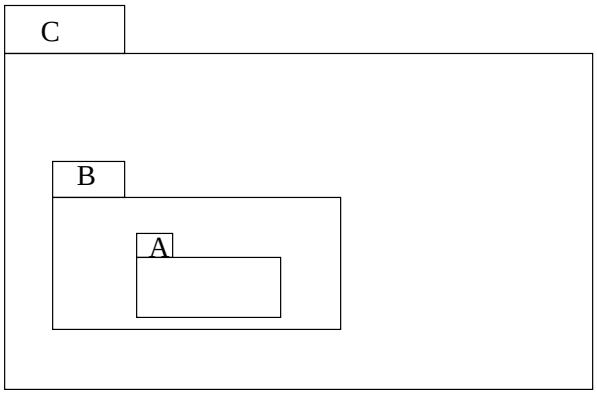


Figure 1: Direct representation of package inclusion

This directness may be exploited by the semantics of diagrams in a systematic way. Such ÒsystematicityÓ is not exclusively the preserve of diagrammatic representations, butÑwith their potential for direct interpretationÑdiagrams have a head start over textual representations in the systematicity stakes. However, to understand what makes diagrams effective, we must consider their interpretation by humans more generally.

2.1.2 Pragmatics in Diagrams

In linguistic theories of human communication, developed initially for written text or spoken dialogues, theories of ÔpragmaticsÕ seek to explain how conventions and patterns of language use carry information over and above the literal truth value of sentences. This concept applies equally well to the use of diagrammatic languages in practice. Indeed, there is a recent history of work which draws parallels between pragmatic phenomena which occur in natural language, and for which there are established theories, and phenomena occurring in diagrammatic languages [GLS98,MR90,Obe96]. For example, studies of digital electronics engineers using CAD systems for designing the layout of computer circuits demonstrated that the most significant difference between novices and experts is in the use of layout to capture domain information [PG92]. In such circuit diagrams the layout of components is not specified as being semantically significant. Nevertheless, experienced designers exploit layout to carry important information by grouping together components which are functionally related. By contrast, certain diagrams produced by novices were considered poor because they either failed to use layout or, in particularly ÔawfulÕ examples, were especially confusing

through their misuse of the common layout conventions adopted by the experienced engineers. The correct use of such conventions is thus seen as a significant characteristic distinguishing expert from novice users. These conventions, termed "secondary notations" in [PG92], are shown in [Obe96] to correspond directly with the graphical pragmatics of [MR90].

More recent studies of the users of various other diagrammatic languages, notably visual programming languages, have highlighted similar usage of graphical pragmatics [Pet95]. A major conclusion of this collection of studies is that the correct use of pragmatic features, such as layout in graph-based notations, is a significant contributory factor in the comprehensibility, and hence usability, of these representations. For diagrammatic languages, and particularly tools which support them, to be effective, they must permit the ready use of such pragmatic features—both in the form of global conventions (such as programming styles) and of more local (e.g. in-house or even individual) styles.

2.2 UML, its semantics and pragmatics

UML, the Unified Modelling Language, is the rising industry standard notation for describing the design of systems, especially object oriented systems. The OMG has accepted Version 1.3 [UML]; ISO standardisation is underway.

In [Ste99] we argued that, for many organisations, a product line notation should use standard UML, because:

1. Developers are increasingly likely to know UML. Familiarity with a notation is essential to using it correctly and efficiently.
2. There are many tools supporting UML. Using the same tool and notation throughout development is convenient.
3. Despite many justified criticisms of UML's semantics, it is the best specified general purpose modelling language to date. Because UML is widely accepted by a large user community, remaining areas of ambiguity are being rapidly clarified.

UML is defined by two main documents, the Notation Guide and the Semantics.[1] The abstract syntax of UML is described using a metamodel, which, with associated constraints, enforces consistency conditions between UML diagrams representing one system. The semantics is defined informally.

2.2.1 Tool support, UML extensions

UML is built to be extensible. The simplest extension mechanism is the ability to define new stereotypes of metamodel elements. One can also define new properties of model elements via tagged values. Recently more far-reaching proposals have been made for profiles[OMG99] or prefaces[CKM+99] to define extended or modified "dialects" of UML. However, extreme caution is needed; in practice, people use those features of UML which are supported by their tool.

Current tools generally implement only a subset of UML; but this is changing rapidly as the standard matures. The new XML Metadata Interchange format (XMI), an OMG standard-in-progress[OMG98] for UML in XML (among other benefits) enables tool developers to update their tools with respect to the latest version of the UML standard with less effort than would previously have been required.

However, even if many tools will soon support the whole language, this does not imply that they will smoothly support extensions. Particular vendors will presumably incorporate certain extensions in their tools; but then their users risk being "locked in" to a single vendor. If tool vendors offer no support for extensibility, we are obviously forced to stick to basic UML, however cognitively bad the result. In this paper we make an intermediate assumption: that several tool vendors will offer the ability to associate user-defined graphics with user-defined stereotypes. This conservative assumption nevertheless allows us sufficient flexibility.

2.2.2 Remark on generalisation

In the UML semantics many metamodel elements are GeneralizableElements: that is, it is often possible to specify that a model element is a generalisation of another similar one. In the context of classes, components etc. this concept is well understood and carries a connotation of substitutability: an instance of the more specialised model element can be substituted for an instance of the more general model element. UML's intention is to broaden this notion to include, for example, generalisations of associations, stereotypes, and collaborations. The implications of such a relationship

[1] The Semantics takes precedence over the NG; many uses of UML are covered by the Semantics, but not mentioned in the Notation Guide. Such cases are normally obvious extensions of things illustrated in the NG, so notation and meaning are unambiguous. Several previous authors have made statements about the limitations of UML based only on the NG, without considering the Semantics: this is an error. It is a reasonable criticism of UML that it is a large language whose limits take considerable effort to determine.

holding are not, however, explored, and they pose formidable technical difficulties and ambiguities which are at present ill-understood. For this reason we avoid using this concept in the core of our PLA descriptions.

3 Users of a product line notation and their tasks

There are many classes of potential users of a product line notation. [BOS98] identified as stakeholders in the architecture of a product: architects, test engineers, configuration managers, management, maintainers, developers, API users, end-users and QA people. In this paper we will consider two groups:

1. users of PLAs: people concerned (in any role) with the task of building a product using a PLA;
2. maintainers of PLAs: people concerned with the improvement of a PLA.

To choose genuinely useful notation we must understand what tasks it should facilitate. This is hard in the case of a notation for PLAs because the set of tasks is complex.

Each task broadly concerns understanding a software design artefact (product or PLA). Users of PLAs have two kinds of tasks:

1. those where the relationship between the PLA and the product is the focus of interest. For example, we may be considering whether the product makes correct use of the PLA, or whether a particular change in functionality can be provided by using PLA features.
2. those in which the product is the focus, and it is less relevant that the product is built from a PLA.

Maintainers of PLAs have tasks which are more similar to those faced by all developers of software for reuse: for example, evaluating potential changes to a PLA. This requires understanding both existing and potential instantiations.

Next we consider what aspects of a PLA should be documented diagrammatically. Both groups need to understand the hotspots of the PLA; 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. Instantiation by composition is generally more straightforward, so here we concentrate on instantiation by specialisation.) 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.

We remark that we are not assuming a component and explicit connector approach to architecture: although popular in academia, this is not an approach taken by most industrial users of UML. We discuss how the notation can be adapted to such an approach in section 5.

Clearly a diagrammatic modelling notation can only be a partial solution to the PLA documentation problem. For example, a concise textual description of the intention of the PLA is indispensable.

4 Proposals and evaluation

We consider the main UML diagram types in turn.

4.1 Externally visible behaviour

This includes the coherent tasks which figure in the requirements of the system; or indeed of any other classifier, such as a subsystem or a class. 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. The PLA designer might use a diagram like Figure 2 to describe this intention.

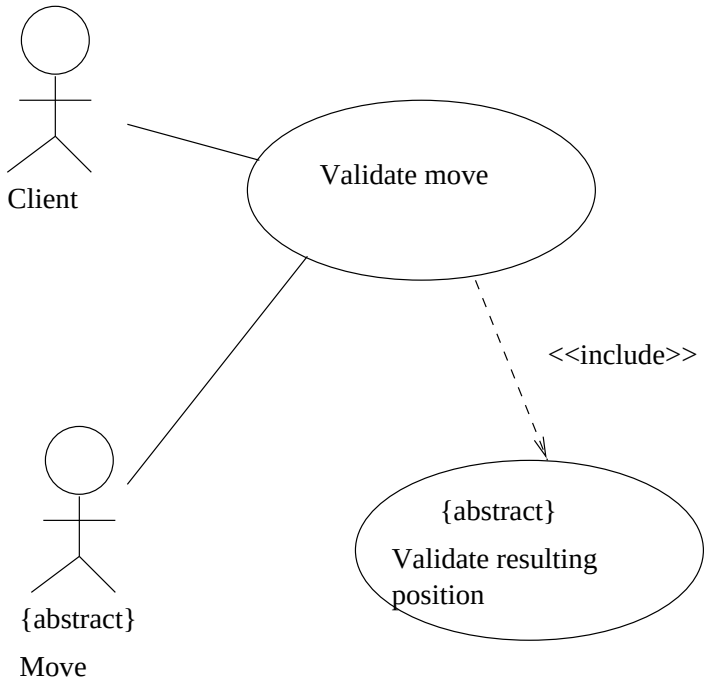


Figure 2: A use case diagram showing variability

To begin with, the product designer needs to be able to tell whether the PLA provides appropriate facilities and flexibility; this is relatively straightforward using this representation. Once a product-level use case model has been defined, there are two different tasks. The product designer needs to

1. check that the product makes valid use of the PLA;
2. make use of the product-level use case diagram as in any other development.

The first task is most easily achieved if the structure of the product-level use case diagram mirrors that of the PLA use case diagram exactly: in our example, if the product reuses `Validate move` and provides an instantiation (say `Validate resulting chess position`) of the hotspot use case.

The second, however, may be hindered by this representation: one of the advantages of use case diagrams in development is that they can be kept simple enough to be used in discussion with customers, who are not experts in the notation. This is harder if the diagram splits use cases as shown. It is straightforward in principle for a tool to temporarily suppress non-primary use cases which are related to primary use cases by extend and include relationships, which would provide a simple view of the use case model. In fact this ÒfoldingÓ of diagrams extends naturally to the other diagram types; for reasons of space we will not discuss it again.

4.2 Static structure

The notation for static structure diagrams raises most of the issues we wish to consider, so we will go into greatest depth here.

4.2.1 What information do we need to represent?

A static structure diagram shows the different kinds of static software entities that are used in the design of a system, together with the relationships between them. For example, depending on the designerÕs style it may show:

- components, such as classes[2] (together with their operations and attributes) or subsystems; these may be used to represent connectors explicitly if required;
- interfaces provided by classes;
- associations between classes: that is, relationships expressed by objects of one class containing references to objects of the other, or by objects of one class sending messages to objects of the other;
- aggregation or composition relationships between classes: formally hard to pin down ([HSB99]) these are particular kinds of association, used when there is a (weaker or stronger) Òis a part ofÓ relationship between objects of the classes;

[2] In this paper we focus on classes. However, this is inessential.

- generalisations between classes: that is, relationships expressed by objects of one class being regardable as objects of the other class;
- dependencies not classifiable as associations or generalisations;
- packages: that is, the grouping of other elements into namespaces.

These are the main elements of static structure diagrams provided by UML. In the context of PLA documentation we also need to show what the PLA provides, what the PLA instantiator must provide, and what flexibility is available. Specifically, when a product is based on a PLA, some of its classes will be provided by the PLA, others by the product instantiator. Some of the former may be optional, for use in certain products when appropriate; this includes the common situation in which the PLA is accompanied by a component library. The latter may build on PLA classes by generalisation or composition or both. More interestingly, some of its relationships will be consequences of decisions made by the PLA designer, whereas others will be determined by decisions of the product designer. For example, certain associations may be required to be present in any instantiating product; others may be optional. The product designer may add associations between elements which, though present at the PLA level, were not associated there. There may be cases where it would be desirable to forbid the product instantiator to do so, in order to avoid adding undesirable dependencies (or such additional associations may be forbidden as a blanket rule; but this is probably too draconian in practice). Similarly, some PLA elements may be refined at the product level—for example, a class may be specialised, or a simple association which in the dynamic model carries a single message may be replaced by a more complex protocol, or by another connection such as an association class.

The exact flexibility required will again depend on the designer's style, but the space of possibilities may be described using two axes:

1. the *existence* dimension: we need to represent whether a component or a connection must necessarily appear in any product, whether it may optionally appear, or perhaps in the case of a connection whether it is forbidden to appear;
2. the *specialisation* dimension: we need to represent whether an element, if it is present at all in a given product, must necessarily be specialised (because, as with an abstract class, it is not fully defined at PLA level), whether it may optionally be refined, or whether it must be used as is.

These dimensions are sometimes confused, but we think it is useful to consider them separately.

4.2.2 Semantic considerations

Thus we must represent certain information which is not routinely represented in UML diagrams. Some of this information will be carried in the pragmatics of the diagram; that is, though present for the human reader it will have no impact on the UML semantics of the diagram. For anything we would like recorded at the semantic level, we need to make use of one or more of UML's extensibility mechanisms. In section 2 we discussed the implications for tool support of the various possibilities, and settled in general for a mild customisation, the use of user-defined stereotypes along with new graphic notations to represent stereotyped model elements. The choice of notation is a cognitive issue, so we postpone its consideration to the next subsection. Here we discuss the choice of what metamodel elements should acquire new stereotypes, and what they should be.

The answers are, happily, almost obvious: in general we stereotype the metamodel element of which an instance is expected to appear in the product, and the stereotypes that we need correspond to the values on the two dimensions of existence and specialisation identified above. For example, if we expect there to be classes in our diagrams, we need stereotypes of the metamodel element Class (or perhaps better one of its supermetaclasses such as Classifier) such as <<must exist>>, <<must be specialised>> etc.[3]

There are a few semantic points which deserve further comment:

4.2.2.1 Representing potential associations

There is an unavoidable underlying question of what exactly an Association between two UML Classifiers is. Here we assume that associations between classifiers model static associations: roughly, a connection which is somehow modelled in data of the classifiers concerned, so that it would be correct, for example, for a code generator to provide instance variables for one or more of the associated classes to store the implementing reference(s). We do not necessarily show two classifiers as being associated whenever an instance of one may dynamically invoke a service of the other. There are sound reasons for making this choice—and it has been defended as the best choice by Rumbaugh in [RBP+91] —but it has 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.

[3] More elegantly, we could use two new tagged values, one for each dimension: but this is perhaps less likely to be supported by tools. The choice is inessential here.

(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 resolvable with respect to the current specification. This is another reason why we skirt the issue of representing connectors in this paper.)

The way out is to show dependencies between components, with a dotted arrow. 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.

4.2.2.2 Namespace control

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.

4.2.2.3 Interdependent constraints

Specialisations have to be compatible (“animals eat food, lions eat meat, cows eat grass”). Compatibility is partly a semantic and partly a type-theoretic notion, 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.

4.2.3 Cognitive considerations

A product designer concerned with the relationship between a product and a PLA must understand: (i) how to validate a product against the PLA; (ii) what must be done to instantiate the PLA; and (iii) whether it is an appropriate PLA for the required product. These tasks involve a comparison of the PLA and product, suggesting that a natural representation would offer separate diagrams for each. An intuitive notation is one in which, as far as possible, these tasks can be performed purely by visual inspection of the PLA and product diagrams.

4.2.3.1 Utilising diagram layout

Consider firstly the task of comparing a product and PLA diagram to check whether the former is a valid instantiation of the latter. A visual inspection of the product diagram should confirm that every required element appears in the product and that every element requiring specialisation has been specialised in the product. Ideally it should be immediately obvious which elements of the product diagram represented the corresponding elements in the PLA diagram. The most obviously natural means of achieving this is to make the visual structure—the layout of key elements—identical in both PLA and product diagrams.

PLA diagrams are often designed by encapsulating or deleting parts of product diagrams, so fortunately there is a tendency for them to share layout. However, typically some elements of the PLA diagram will be replaced by more complex sub-diagrams in the product diagram. For example, an abstract class in the PLA may be replaced (instantiated) in the product diagram by a more complex arrangement of several classes and associations. Checking that this sub-diagram is a valid instantiation is a user task that goes beyond what is directly supportable in a diagram. However, the diagram must make it obvious that the sub-diagram is intended to replace the original PLA element. The PLA diagram should ideally contain sufficient free space for such a sub-diagram to be directly inserted without perturbing other PLA diagram elements. This use of layout carries further advantages, as this pragmatic feature encourages the PLA designer to record information concerning possible instantiations. For example, a designer might add the comment “Your CurrentPosition here” in a space left blank for the instantiation. This also helps distinguish accidental from deliberate free space.

We illustrate this approach with a na ve use of UML, taken from [SP99]. Figure 3 shows the PLA for a family of computerised games, while Figure 4 shows an instantiation of this for noughts-and-crosses.[4]

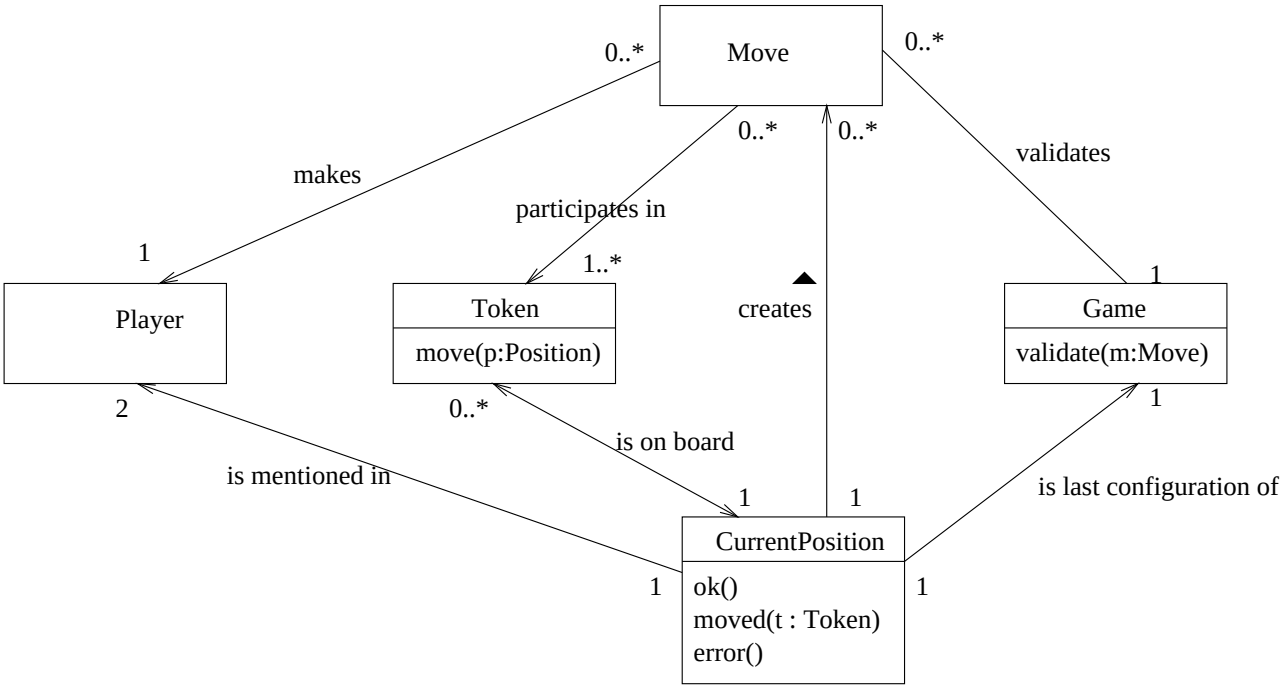


Figure 3: Class diagram for games PLA

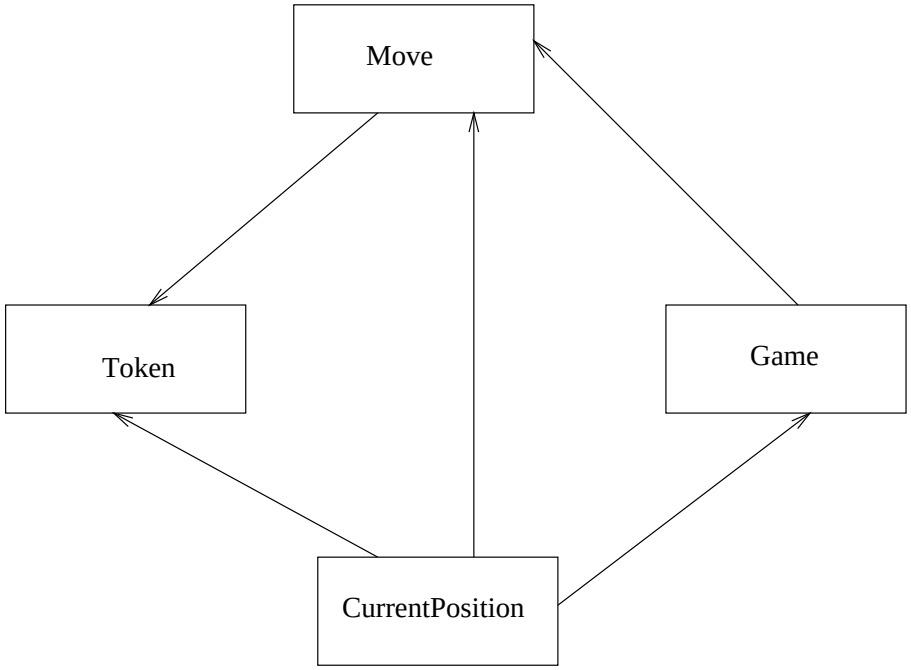


Figure 4: Class diagram for noughts and crosses

Note that the PLA diagram of Figure 3 fails to reflect the structure of elements in the product diagram. A more intuitive representation of the games PLA is given by Figure 5. This alternative diagram also contains further features which carry semantic information, which we discuss next.

[4] Ò Tic-tac-toeÓ for American readers.

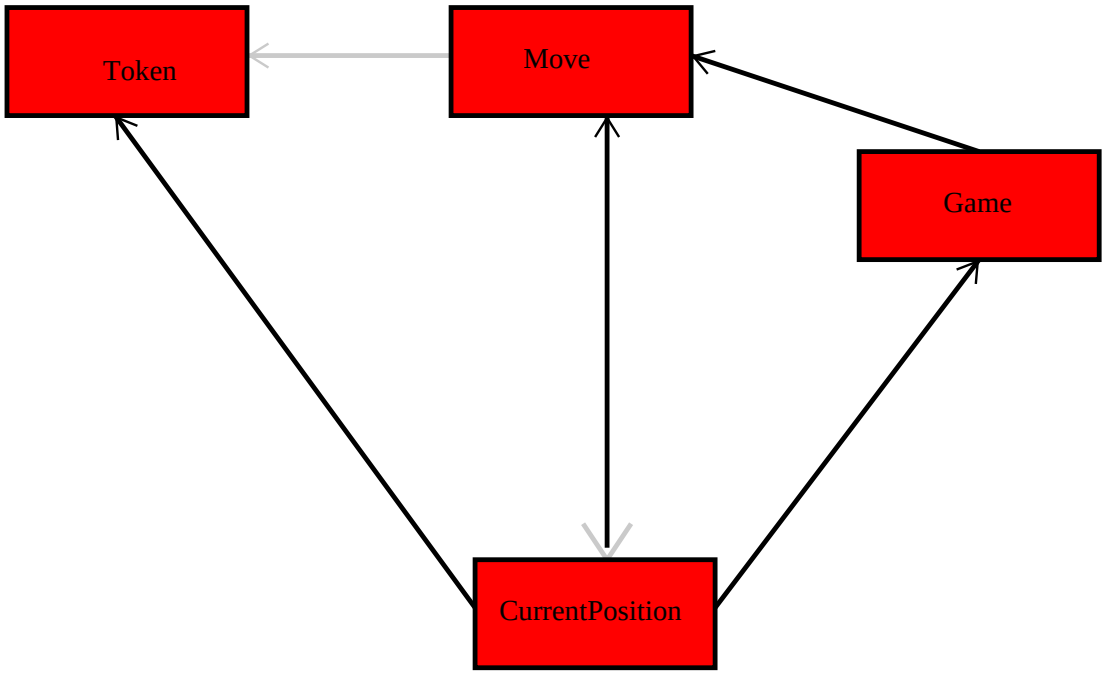


Figure 5: Alternative diagram for games PLA

4.2.3.2 Making distinct semantic dimensions visually distinct

As we have stated, checking that a product is a valid instantiation of a PLA involves checking existence and specialisation. These two issues are orthogonal for particular elements in a PLA; whether an element is required to exist, may be viewed independently of whether it need be specialised. An intuitive representation should reflect this distinction; the notation chosen to represent the dimension of existence (e.g. optional) should be clearly distinct from that which represents the dimension of specialisation (e.g. abstract).

The basic symbols of static diagrams are boxes and arrows which connect them. Following Graphic Design recommendations [Hor98,Tuf90], we see that such symbols may be represented via a number of basic line types (including solid, dashed and wavy). Each of these line types may be further varied across a number of graphical dimensions (such as thickness, colour and shading). We directly represent the semantic dimensions of existence and specialisationÑand the distinction between themÑby assigning to them distinct graphical dimensions. The most appropriate dimensions are those which appeal to an appropriate visual metaphor. We represent the dimension of existence by saturation. Thus must-exist is represent by a bold line, while the weaker may-exist is represented by a paler, and hence less imposing, shaded line. We use colour to represent the dimension of specialisation as it has the most obviously appealing visual metaphor[5]. That is, ÒhotÓcolours such as red and orange are used to represent strong requirements to specialise (c.f. hotspots), whilst other, ÒcoolerÓcolours represent less stringent requirements to specialise. Figure 5 illustrates this.

4.3 Interactions and collaborations

A UML collaboration diagram documents the relationships between roles that may be played by instances (objects etc.). 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. Some consistency between the static structure and the interactions is ensured, because in UML both are views of the same model. The use of these diagrams for documenting architectures is discussed in [RMRR98]. Here we consider documenting hotspots; especially, how to show abstract collaborations and interactions which may be instantiated or refined later.

We may use mechanisms consistent with those described for static structure to describe the hotspots in a collaboration or interaction documenting a PLA. For example, in Figure 6 (i) message a is pale because it is optional (it involves

[5] The choice of colour leads to further issues such as monochrome printing and colour-blind users. We raise this issue, but do not address it here.

communicating with an optional element of the architecture) and message b is on a Ò redÓ background because it must be replaced by a more complex protocol. In the specialisation case two different situations arise:

1. instances which exist at PLA and at product level might carry out a more complex interaction at the product level; for example, a single message may be replaced by a (perhaps conditional) sequence of messages. Figure 6 (ii) illustrates.
2. there may also be more instances involved at the product level, because a single instance at PLA level has been reified by a collaboration of instances at the product level.

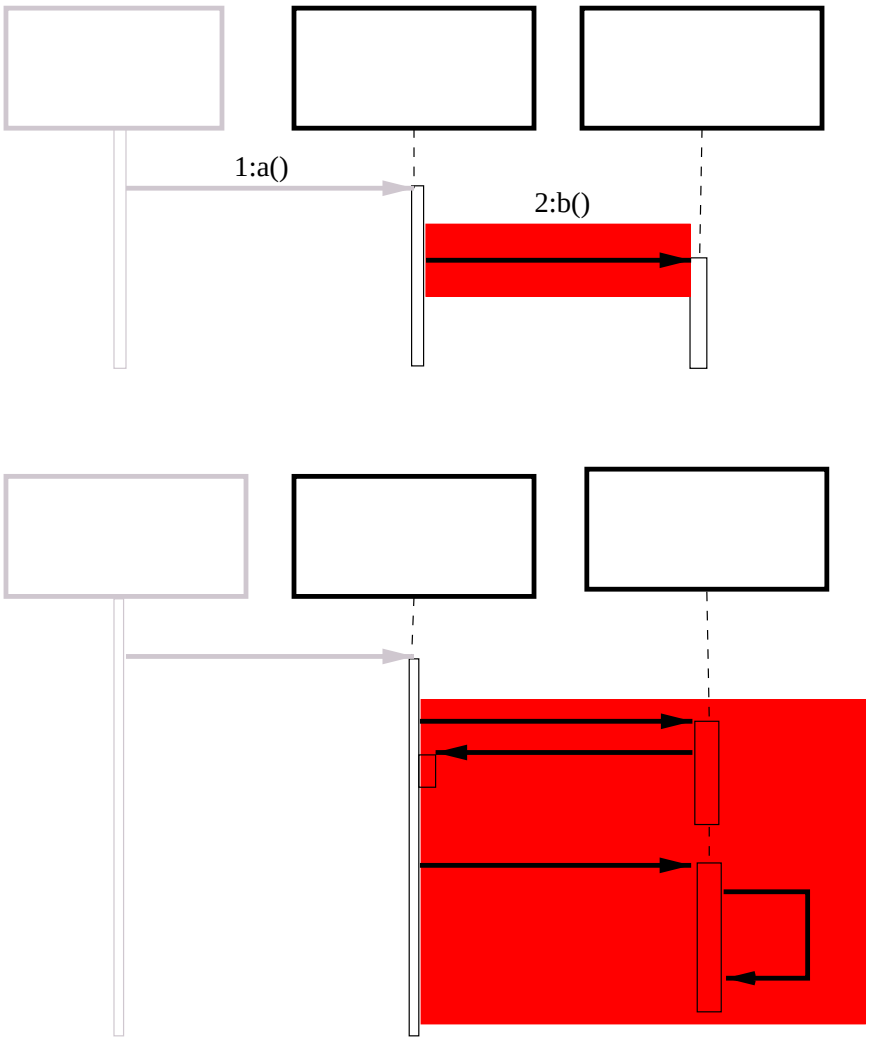


Figure 6: Protocol Specialisation

(We avoid using generalisation of Collaborations here not only for the technical reasons mentioned earlier but also because we consider the oval Collaboration notation more useful for demonstrating the use and refinement of small design patterns or frameworks, which is what it was intended for.)

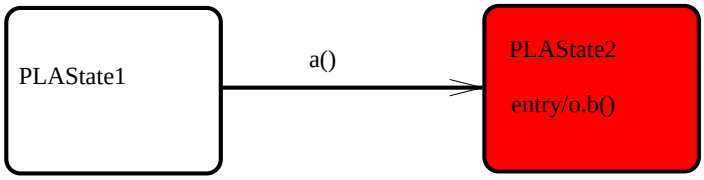
Some architectures will require sophisticated documentation of real-time and synchronisation aspects. UML includes a reasonable set of features to support this, in the form of notation for timing constraints on interaction diagrams and support for different threading models. However this is still an area of active work, particularly in the embedded system community, e.g. [DlJ99].

4.4 State diagrams

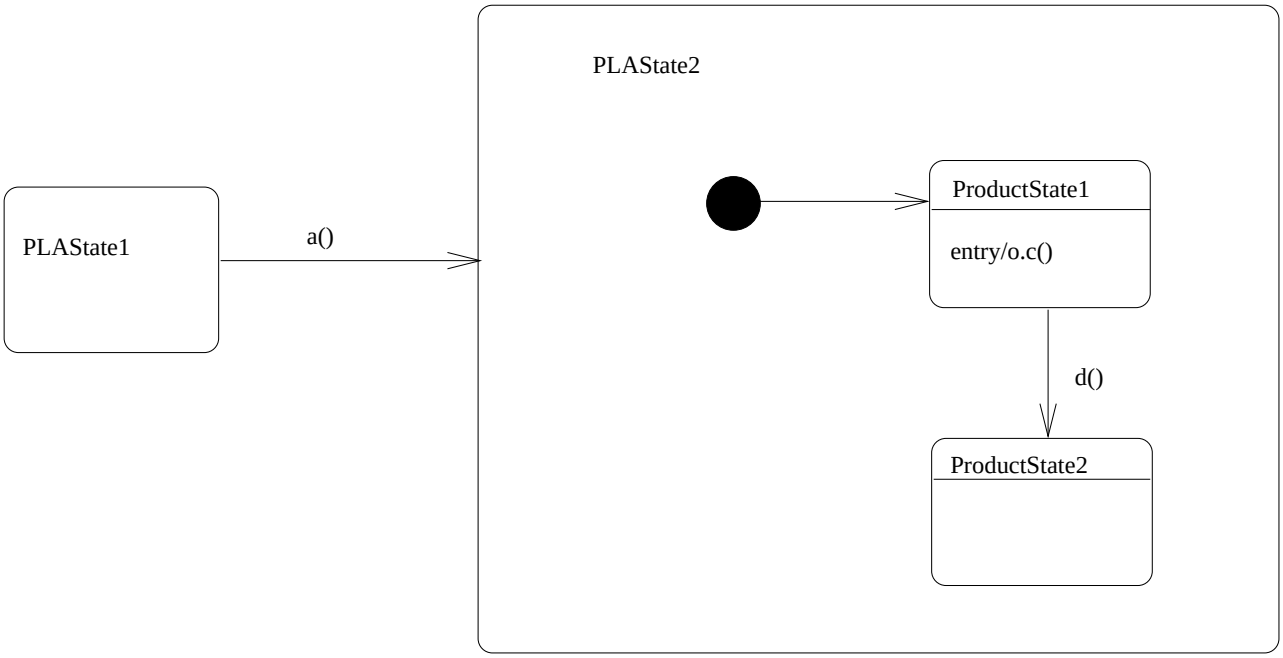
The most likely use of state diagrams in connection with PLAs is as an adjunct to interaction diagrams to demonstrate protocols which might be refined in products using the PLA. In order to support the PLA maintainer in understanding uses of the PLA, or the product developer in understanding the relationship between the PLA and the product, the structure of the PLA state diagram needs to be recoverable from the representation of the product state diagram. In case

1 this is easily achieved by using nesting of states: the PLA state appears as a superstate, with product-specific substates, as shown in Figure 7. (We have chosen to omit the existence/specialisation information in the product diagram to illustrate the possibility of doing so.)

In case 2 this is more challenging. The collection of instances at the product level needs to be represented as an instance of some suitable classifier in its own right; then its state diagram can be represented using nested states as in case 1. Whether this is desirable is dubious.



(i)



(ii)

Figure 7: UML Statechart

4.5 Further Diagrams

The notations described are easily extended to activity diagrams; and deployment and component diagrams do not raise any further interesting issues. For reasons of space we omit making the notation explicit here.

4.6 Non-diagrammatic elements

We have mentioned that there is naturally a need for supporting text to document things like the purpose of a 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. [Hof99] 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, as suggested by the standard [UML].

5 Related Work

Previous studies of diagrams have typically examined two differing dimensions through which diagrammatic representations "naturally" embody semantic information. Firstly logical analyses such as [MK95,Shi95,Sow93] and the collection in [AB96] have examined the inherent constraints of diagrams (topological, geometric, spatial and so forth) to explicate their computational benefits. The second dimension which has been studied concerns features and properties which impact upon the cognition of the user. These more cognitive studies have typically studied either users of highly specialised diagrams (such as Venn diagrams, and simple visual programming languages [BWGPss,Goo99,SO91,ZN94]), or generic HCI issues [BG99,Gre89,Pet95]. Our approach is based on the studies of [GLS98,Gur99], which seek to unify the two dimensions of these earlier studies. More recent work has demonstrated the potential, and benefits, of formalising certain pragmatic aspects of diagrammatic software engineering languages [GT99].

We are not aware of any other published work discussing the use of UML in notating product lines, though we are informally aware that a number of companies are making use of UML and a variety of ad hoc extensions to it and conventions in it for this task; this was of course part of the motivation for this work. Before discussing previous work applying UML to the description of architectures and frameworks, which is relevant, we should deal with a possible objection: it would be absurd to claim that UML cannot be used to model architecture in the sense in which the term is used by most practitioners. 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 [JBR99] 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 [DW98] 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.

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 [RMRR98, MR99, Ber98, Hof99], though doubtless more remains to be done.

Without duplicating their work, a couple of comments are:

1. [RMRR98, MR99] 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. [BGK+97] 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.

   Similarly, it is in principle possible to use a subsystem to represent a connector in a component/connector view of architecture. However, this is to use the elements in a way not envisaged by the inventors of UML and not well supported by the semantics.
2. Section 6.2 of [MR99] 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 [Mat96]. We do not know of any previous work addressing ÒpureÓ UML for this purpose.

[DW98] (which heavily modifies UML) is relevant; the draft paper [dFdLAC] 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.

6 Conclusion

We have demonstrated how using a cognitive and semantic theory of diagrammatic languages we can evaluate ways of using UML to represent product line architectures and their instantiations, and have proposed a coherent set of notation. We stay within standard UML, making use of stereotypes as our customisation mechanism. For a CASE tool to allow the user to use our notions the main thing that would be required is the implementation of UMLÕs existing stipulation that users can define new graphic elements to correspond to stereotyped model elements. This ability is not mainstream in tools at the time of writing, but it can be expected to become so very soon. Our plans for future work include applying cognitive theory to tool support for extending UML.

The work reported here is based on theory of diagrammatic languages, backed by an understanding from the literature and our experience of the needs of PLA. The next step is for somebody to take the recommendations given here and put them to use in a real product line situation, investigating to what extent our theoretically-based expectations are borne out in practice and what unexpected problems exist. We would be most interested to hear from practitioners interested in collaborating on such a venture.

References

[AB96] G Allwein and J Barwise. *Logical Reasoning with Diagrams*. Oxford University Press, New York, 1996.

[Ber98] Lodewijk Ber98. A notation for describing conceptual software architectures. In *Electronic Proceedings of the First Nordic Software Architecture Workshop*, 1998. bilbo.ide.hk-r.se:8080/ bosch/BOS9898.

[BG99] A F Blackwell and T R G Green. Does metaphor increase visual language usability? In *15th IEEE Symposium on Visual Languages (VLÕ99)*, pages 246--253, Los Alamitos CA, 1999. IEEE Computer Society.

[BGK+97] Dirk Baumer, Guido Gryczan, Rolf Knoll, Carola Lilienthal, Dirk Riehle, and Heinz Zullighoven. Framework development for large systems. *Communications of the ACM*, 40(10):52--59, October 1997.

[Bos] Jan Bosch. Electronic proceedings of the first nordic software architecture workshop. bilbo.ide.hk-r.se:8080/ bosch/BOS9898.

[BWGPss] A F Blackwell, K N Whitley, J Good, and M Petre. Cognitive factors in programming with diagrams. *Artificial Intelligence Review*, In press. To appear in special issue on Thinking with Diagrams.

[CKM+99] Steve Cook, Anneke Kleppe, Richard Mitchell, Bernhard Rumpe, Jos Warmer, and Alan Cameron Wills. Defining UML family members using prefaces. In *Proc. TOOLS 32*. IEEE, 1999.

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

[dlJ99] Clement de la Jonquiere. An introduction to embedded software for system-on-chip 1. A Cadence Design Systems Inc. seminar given to partners in the Institute for System Level Integration on June 2 1999.

[DW98] Desmond DÕSouza and Alan Cameron Wills. *Catalysis: Objects, Frameworks and Components in UML*. Addison-Wesley, 1998.

[GLS98] C Gurr, J Lee, and K Stenning. Theories of diagrammatic reasoning: distinguishing component problems. *Mind and Machines*, 8(4):533--557, December 1998.

[Goo99] J Good. VPLs and novice program comprehension: How do different languages compare? In *15th IEEE Symposium on Visual Languages (VLÕ99)*, pages 262--269. IEEE Computer Society, 1999.

[Gre89] T R G Green. Cognitive dimensions of notations. In Sutcliffe and Macaulay (eds) *People and Computers V*. pages 443Ñ460, CUP, 1989.

[GT99] C Gurr and K Tourlas. Formalisating pragmatic aspects of visual languages. In *15th IEEE Symposium on Visual Languages (VLÕ99)*, pages , Los Alamitos CA, 1999. IEEE Computer Society.

[Gur99] C A Gurr. Effective diagrammatic communication: Syntactic, semantic and pragmatic issues. *Journal of Visual Languages and Computing*, 10(4):317--342, August 1999.

[Hof99] Hofmeister et al. Describing software architecture with UML. In *Proc. WICSA1*, 1999.

[Hor98] R E Horn. *Visual Language: Global Communication for the 21st Century*. MacroVU Press, Bainbridge Island, WA, 1998.

[HSB99] Brian Henderson-Sellars and Frank Barbier. Black and White Diamonds *Proceedings UML'99*, 1999.
[JBR99] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*. Addison-Wesley Longman, 1999.
[Mat96] Michael Mattson. *Object-Oriented Frameworks: a survey of methodological issues*. PhD thesis, University College of Karlskrona/Ronneby, 1996.
[MK95] K Myers and K Konolige. Reasoning with analogical representations. In J Glasgow, N H Narayan, and B Chandrasekaran, editors, *Diagrammatic Reasoning: Cognitive and Computational Perspectives*, pages 273--302. MIT Press, 1995.
[MR90] J Marks and E Reiter. Avoiding unwanted conversational implicature in text and graphics. In *Proceedings of AAAI-90*, pages 450--456, 1990.
[MR99] 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.
[Obe96] J Oberlander. Grice for graphics: pragmatic implicature in network diagrams. *Information design journal*, 8(2):163--179, 1996.
[OMG98] OMG. Xml metadata interchange (XMI). ftp://ftp.omg.org/pub/docs/ad/98-10-05.pdf, October 1998.
[OMG99] OMG. White paper on the profile mechanism. ftp://ftp.omg.org/pub/docs/ad/99-04-07.pdf, April 1999.
[Pet95] M Petre. Why looking isn't always seeing: Readership skills and graphical programming. *Communications of the ACM*, 38(6):33--45, June 1995.
[PG92] M Petre and T R G Green. Requirements of graphical notations for professional users: electronics CAD systems as a case study. *Le Travail Humain*, 55:47--70, 1992.
[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modelling and Design*. Prentice-Hall Inc., 1991. ISBN 0-13-630054-5.
[RMRR98] 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.
[Shi95] S-J Shin. *The Logical Status of Diagrams*. Cambridge University Press, 1995.
[SO95] K Stenning and J Oberlander. A cognitive theory of graphical and linguistic reasoning: logic and implementation. *Cognitive Science*, 19:97--140, 1995.
[Sow93] J F Sowa. Relating diagrams to logic. In *Conceptual Graphs for Knowledge Representation: Proceedings of 1st International Conference of Conceptual Structures*. Springer-Verlag LNAI 699, 1993.
[SP99] P Stevens and R Pooley. *Using UML: Software Engineering with Objects and Components*. Addison Wesley, Updated edition, 1999.
[Ste99] Perdita Stevens. UML for describing product-line architectures? Proc. ECOOP Workshop on Object Technology for Product Line Architectures, 1999.
[SY97] K Stenning and P Yule. Image and language in human reasoning: a syllogistic illustration. *Cognitive Psychology*, 34(2):109--159, 1997.
[Tuf90] A R Tufte. *Envisioning Information*. Graphics Press, Cheshire, CT, 1990.
[UML] OMG Unified Modelling Language Specification 1.3. OMG 99-06-08, June 1999.
[ZN94] J Zhang and D Norman. Representations in distributed cognitive tasks. *Cognitive* Science, 18:87--122, 1994.