A framework for UML consistency

John Derrick and David Akehurst and Eerke Boiten

Computing Laboratory, University of Kent, Canterbury, CT2 7NF, UK
{J.Derrick,D.H.Akehurst,E.A.Boiten}@ukc.ac.uk

Abstract. In this paper we discuss a framework by which one might approach questions of consistency in UML. The framework derives from work undertaken in the Open Distributing Processing (ODP) standardisation initiative which looked at consistency checking across the ODP viewpoints.
This work has resonance with some of the problems facing those using the many different aspects of UML, and the purpose of this paper is to discuss how the existing work could be applied in a UML context.

Keywords: UML, Consistency, Viewpoints, Open Distributing Processing, Refinements.

1 Introduction

The UML is rather unique as a modelling language, since it combines multiple notations in a loosely coupled framework. Indeed this combination of multiple notations defined in a framework which is non-prescriptive (i.e., a designer is free to use and interpret these notations with some freedom) has many potential benefits, and has helped its popularity.

There is a flip side, however: the loose coupled nature means there does not exist a unified semantics. This lack of semantics means that it is not immediately clear *how* to determine the consistency of a UML specification, or indeed *what* consistency means in the language.

This situation contrasts with most notations, even wide-spectrum languages such as RAISE [22] or ones containing more than one sub-notation as in SDL [11]. In these situations a unified semantics is defined as part of the language, and this allows a smooth transition to be defined between the sub-notations in the language, and a well-defined notion of consistency to be given.

However, there are existing techniques and a wide-ranging body of work that offer ways to approach the problem of defining and checking consistency for UML. In particular, we focus here on the work on viewpoints, or partial specifications, which has investigated ways in which to check consistency for combinations of disparate notations.

The problems facing UML can, we believe, usefully draw upon many of these ideas and the purpose of this paper is to draw out some of these ideas and comment on how UML might build upon this existing work. We thus offer less

of a solution to the issue of consistency in UML, but rather tentatively put forward a framework by which techniques might be developed.

In Section 2 of this paper we discuss the ideas of viewpoints and partial specification. The central thesis is that a viewpoint represents one particular perspective of a system, that is, it is a *partial* specification. There are a number of approaches to partial specification but, like UML, they all bring an attentive problem, viz. how to check whether the partial specifications are consistent with one another.

In this paper we focus on work related to one particular viewpoints approach, namely that which has taken place in the Open Distributed Processing (ODP) domain. ODP [25] was initially developed as a standardisation framework for distributed systems, but has more recently been proposed as a *development architecture* for the construction of distributed systems [12]. In Section 3 we describe the approach to consistency in ODP, with pointers to its relevance to work on consistency in UML. This is expanded in Section 4 where we sketch some of the techniques that are available to check the consistency of specifications. Section 5 then describes the issues that would need to be resolved for this work to be applied to a UML context.

2 Viewpoints and partial specification

To understand the relevance of the work on viewpoints and partial specification consider the UML specification given in Figures 1, 2, and 3.

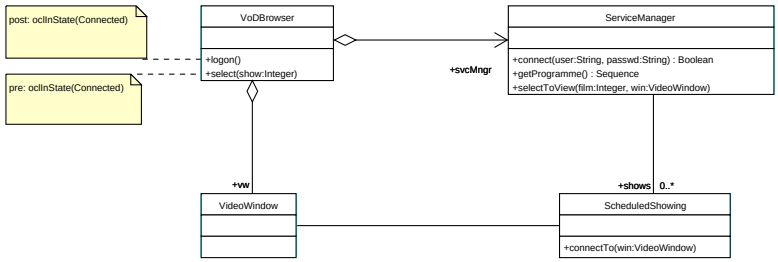

Fig. 1. Class diagram of a video on demand browser

These consist of a class diagram to define some of the structural aspects of the system, a state chart to define the behaviour within one particular object, and a sequence diagram to define some of the allowable interactions between the components in the system.

No single diagram defines the behaviour of the complete system, this is only given by their collective constraints. Indeed, the philosophy of UML is that complex systems are best specified via a number of different views of a model.

Thus each diagram represents a *partial description* of the complete system. Such a partial description is usually called a partial specification or a viewpoint [18].

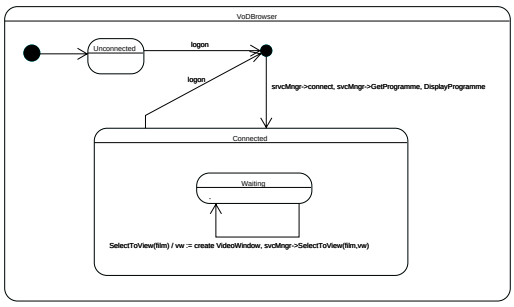

Fig. 2. State chart for the video on demand browser

Work on viewpoints goes back many years, with relevant work being done in requirements engineering [20], formal system development [36], distributed systems [24] and in software engineering in general [31]. The idea is that rather than having a single thread of system development, in the style of the classic waterfall approach, multiple partial specifications (i.e. viewpoints) of a system are considered. Each particular specification represents a different perspective on the system under development and, in fact, may well be written by a different specifier.

Work in distributed systems is of particular relevance since there has been significant progress on viewpoint consistency in that domain, and in particular within the Open Distributed Processing standardisation initiative.

The Open Distributed Processing is a joint ITU/ISO standardization framework for constructing distributed systems in a multi-vendor environment. The architecture has reached a level of maturity, with the main ODP document, the Reference Model for Open Distributed Processing (RM-ODP), an established international standard. For introductions see [28, 29, 2] and the standards documents themselves [24]. Significant features of ODP include *object based* specification and programming, use of *transparencies* to hide aspects of distribution and *viewpoints*. The latter has relevance to UML since the viewpoints provide a basic separation of concerns, enabling different participants to observe the system from suitable perspectives and at suitable levels of abstraction.

A central tenet of ODP is the use of these viewpoints to decompose the task of specifying distributed systems, and this has resonance with the use of different diagrammatic notations within UML. ODP defines five viewpoints, *Enterprise*, *Information*, *Computational*, *Engineering* and *Technology*. It is beyond the scope of this paper to give a full introduction to ODP viewpoints modelling. The interested reader is referred to [27]. In contrast to many other viewpoint models,

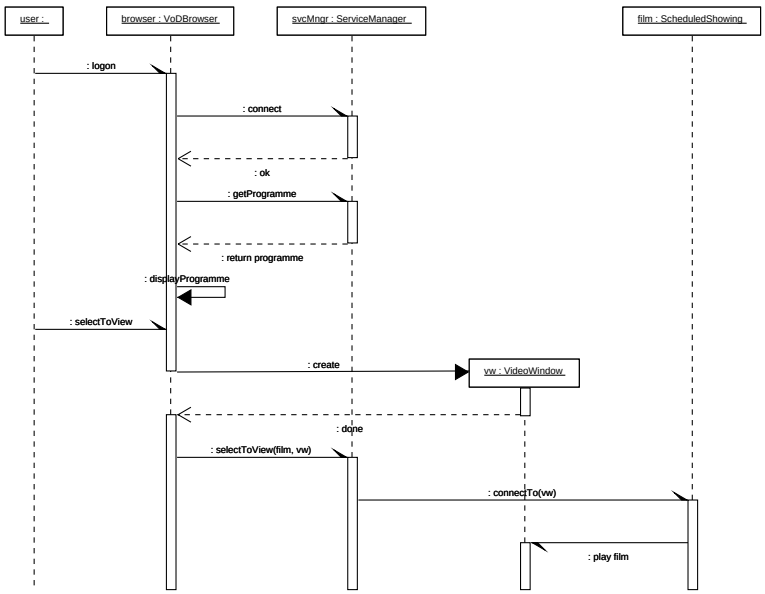

Fig. 3. Sequence diagram for the video on demand browser

ODP viewpoints are predefined and in this sense *static*, i.e., new viewpoints cannot be added, and this contrasts with how they arise in other models, e.g., [20]. Each of the viewpoints has a specific purpose and is targeted at a particular class of specification.

One of the consequences of ODP adopting a multiple viewpoints approach to development is that descriptions of the same or related entities can appear in the different viewpoints and must co-exist. Thus, the different ODP viewpoints can impose contradictory requirements on the system under development and consistency of specifications across viewpoints becomes a central issue. The problem is complicated by the fact ODP viewpoint specifications can be written in different languages, that is, languages particularly suited for the viewpoint at hand, e.g., Z [32] for the information viewpoint and LOTOS [5, 23] for the engineering viewpoint [7].

A significant amount of work has taken place on consistency in ODP [8, 1, 17]. In particular, a number of research projects were undertaken at the University of Kent at Canterbury in order to explore this issue. The scope of the work was broad, including theoretical investigations of the nature of consistency checking [10, 6], techniques for consistency checking within specific formalisms, e.g., LOTOS [33, 10, 34] and Z [4, 16] and, techniques for consistency checking across specification languages [14, 15]. The rest of the paper explains the approach and discusses how the solutions developed might be applied in a UML context.

3 Consistency in ODP

The complete specification of any non-trivial distributed system involves a very large amount of information. Attempting to capture all aspects of the design in a single description is generally unworkable. Most design methodologies aim to establish a coordinated, interlocking set of models each aimed at capturing one facet of the design, satisfying the requirements which are the concern of some particular group involved in the design process. In ODP, this separation of concerns is established by identification of five *viewpoints* [24]. For each of the five viewpoints, the reference model presents a set of definitions and a set of rules which constrain the ways in which the definitions can be related, this is known as a viewpoint language.

Furthermore, since different languages are applicable to the specification requirements of different viewpoints, e.g., LOTOS for the engineering viewpoint and Z/Object-Z for the information viewpoint, consistency checking for ODP specifications must be performed both within and across specific languages.

The relevance to UML should be obvious. The problem of consistency in ODP is one of relating partial specifications of a system. These specifications will typically be written in different notations, with some of the partial specifications existing at the same level of abstraction, but others existing at different levels of abstraction. Although the context is slightly different, this is the same issue, essentially, as that facing UML. The UML situation does have one advantage, and thats that the specification languages used are known in advance (i.e., they are the notations that make up the UML). Thus, we know the set of notations that will potentially arise in a UML specification, whereas in general in ODP any specification language at all could be instantiated as one of the viewpoint languages.

In order to be able to check the consistency of multiple viewpoint specifications (whether in ODP or UML) we first need to define what is meant by consistency. This is not obvious, for example, at one time the ODP reference model alluded to three different definitions. However, this can be resolved by adopting a formal framework (e.g., see [9]) and this provides a definition of consistency between viewpoints general enough to encompass all three ODP definitions.

The traditional way to check for consistency is by translating all the viewpoints to a common underlying semantic framework (e.g., first order predicate logic [36]) – if all of these translations have a common model (their conjunction is satisfiable) then the viewpoints were consistent. However, this approach suffers from traceability problems, see [3]: a reported inconsistency will be in terms of the semantic framework, and can normally only be partially translated back into the original languages. In any case, it will be hard to recover all of the original syntactic structure in such a back-translation. Additionally, the use of an underlying common model, also due to loss of syntax, makes incremental consistency checks (on a specification in development) more difficult.

An alternative approach to consistency checking is described in [9, 10, 6]. Here consistency of viewpoint specifications is defined as the existence of a common "implementation".

Although intuitively pleasing, this definition is not practical for actual consistency checking. It would require the viewpoint specifiers to attempt to reconcile their viewpoints and prototype an implementation each time a consistency check was desired, which, if at all possible, is completely unrealistic. Hence, for consistency checking to be feasible, it has to be defined at the abstract level of specifications. This is realised by using two important concepts: *refinement relations* and *correspondences*.

The latter of these is a concept for describing the common aspects of different viewpoints. The issue of consistency only arises because viewpoints may overlap on certain aspects of the envisaged system. In simple examples, such overlapping parts will be linked implicitly by having the same name and type in both viewpoints. In general however, we may need more complicated descriptions for relating these common aspects. *Correspondences* provide this linkage. It is also worth pointing out that although correspondences between viewpoints may be quite complex and intricate, many of them are generic. In the RM-ODP, for example, many such correspondences can be derived directly from the ODP architectural semantics, in UML some correspondences will arise from the different roles the notations play.

Refinement is the usual notion of development, and its use is well documented elsewhere. Its use is that the consistency of several partial specifications can be checked by finding a common *refinement*. If both specifications use the same refinement relation, and the refinement relation is complete (in the sense that all implementations can be obtained by refinement), existence of a common refinement is even *necessary*. Furthermore, under certain conditions common refinement of viewpoint specifications can be used to check consistency with additional viewpoints, ensuring that all consistency checks can be just between two specifications.

We can thus define consistency as:

A set of viewpoint specifications are consistent if there exists a specification that is a refinement of each of the viewpoint specifications with respect to the identified refinement relations and the correspondences between viewpoints. This common development is called a unification.

As an example, consider the two simple pieces of behaviour, given as labelled transition systems in Figure 4.

Are these consistent? Well, it all depends on the notion of refinement used. If we use the LOTOS reduction relation or the CSP failures-divergence refinement relation, then they are *not* consistent since the first behaviour says that after an a one must be able to do a b, but this behaviour isn't allowed by the second behaviour.

However, if one used a different refinement relation, say one that allowed the extension of traces (as in the LOTOS *ext* relation), then these two behaviours *are* consistent, their unification is given in Figure 4. Thus what this very simple example illustrates is that consistency *really* is tied to the notion of development of refinement used. This issue is as relevant to UML as it is to ODP, without

Fig. 4. Two simple behaviours defined as labelled transition systems and their unification according to extension

knowing what valid refinements (or implementations) of state charts, sequence diagrams etc are we cannot hope to tackle consistency in UML with any meaning.

One natural consequence of the definition of consistency is that ("global") consistency between a set of viewpoints is *not* implied by consistency between all pairs of viewpoints in the set. Each pair of viewpoints may have a nonempty set of common refinements, but the intersection of those sets may still be empty. However, if we use the least unification (with respect to the refinement ordering) then it ensures that using unifications as the intermediate results, *global consistency* of a set of viewpoints can be established by a series of binary consistency checks, assuming a few reasonable restrictions on the refinement relations involved [10]. This is not the case if an arbitrary common refinement is selected as the unification.

As just indicated, different viewpoint specifications may be related to the unification by different refinement relations. For example, in ODP the LOTOS engineering viewpoint might be related by a conformance relation, while the Z computational viewpoint might be related by the Z refinement relation. Note also that the definition makes no reference to ODP or its particular viewpoints. We thus view this as a general definition which could be adopted as the basis for consistency within UML.

This perspective on consistency, that is, one that uses refinement as the key notion in development makes for a very behavioural focus for questions of consistency. This focus on behavioural consistency has consequences for UML. Such an approach is allied to a behavioural semantics where the semantics of a specification records only the pertinent aspects of behaviour rather than structured information or syntactic detail. All meaningful notions of semantics take such a view.

Consider, for example, the two UML specifications given in Figures 5. They describe the same behaviour, and therefore are consistent, a behavioural approach (to semantics and consistency) should record this rather than stressing the syntactic or structural differences.

Thus to adopt these ideas in a UML context we have to take the view that all that matters in a specification is its behaviour. Indeed most specification

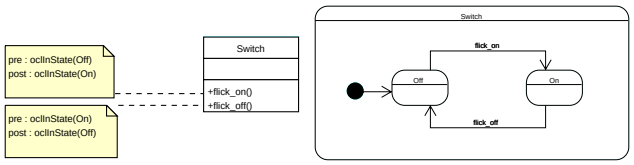

Fig. 5. Class diagram (with OCL annotations) and Statechart for a switch

notations take a behavioural view on semantics so, e.g., in a process algebra its semantics is precisely the aspects of behaviour that are observed, so get events in traces and not process structure. Although UML hasn't got a settled formal semantics we believe that the behavioural approach to semantics and consistency is one of merit.

4 The behavioural approach

A central feature of our work is the use of formal methods and one motivation for employing a formal approach is that they enable consistency checking through analysis of the formal semantics of specifications. However, using formal methods also allows one to take a behavioural approach to consistency.

Consistency of viewpoint specifications is often only considered at the level of *structural* and *syntactic* consistency. However, for consistency to truly correspond to implementability, *behavioural* aspects must also be checked. For example, if we have a pair of viewpoints describing a particular object it is not sufficient just to know that the signatures of the pair of specifications are non-contradictory, but it is also necessary to know that both the order in which operations can be performed and the effects of performing those operations are non-contradictory.

Formal semantics give a behavioural interpretation of notations in a mathematical model, e.g., Z [32] is interpreted in ZF set theory and LOTOS [5] is interpreted as labelled transition systems or trace/refusal models. Consequently, through formal semantics the behavioural consistency of specifications written in formal notations can also be checked.

Indeed, besides a definition of consistency, we have also investigated methods for constructively establishing consistency [6]. This involves defining algorithms which build unifications from pairs of viewpoint specifications. An important notion in this context is that of a *least developed unification*. This is a unification such that all other unifications are refinements of it. Thus, it is the *least developed* of the set of possible unifications according to the refinements relations of the different viewpoints.

Unfortunately, it is not the case that least developed unifications can always be derived. In [6] properties that refinement relations must possess for such unifications to exist are investigated. In most cases refinement relations possess the

required properties (in particular, for Z refinement, least developed unifications can always be constructed).

One immediate consequence of this is that for consistency checking to be defined in this many then the languages involved need a notion of development or refinement. Most languages have this, e.g., there are well-defined refinement relations for Z, LOTOS, SDL etc, however, there are obvious consequences for UML in this respect. In particular, well-defined notions of refinement for the languages used within UML are needed, since this is true for general software engineering reasons, work in this direction will produce as a side-effect beneficial consequences for the consistency checking work.

Based on the formal definitions of consistency and unification above, we have developed several consistency checking techniques for and between specifications written in different languages. Doubtless some of this work could be ported to the UML context.

Consistency in LOTOS What makes viewpoint consistency checking in LOTOS a particularly challenging task is the existence of a large collection of refinement relations. These characterise different ways in which LOTOS specifications can be viewed as partial specifications, with refinement being e.g., conformance, functionality extension, or reduction of non-determinism. We have defined unification and consistency checking strategies for the spectrum of these relations [33, 34].

Furthermore, using this characterisation we can check the consistency of LOTOS specifications. We do this by generating the unification of source specifications using the above principle and if this unification is well formed (a set of well defined criteria exist to check this) then the original descriptions were consistent.

However, although this gives us an algorithm for consistency checking, it is performed completely at the semantic level. In particular, the resulting unification will at worst be a semantic model and will at best be a LOTOS specification with no high level structure (in LOTOS terminology, conforming to the monolithic specification style).

Consistency in Z and Object-Z For Z as a viewpoint specification language we have so far assumed the established states-with-operations specification style, with an eye towards encapsulation of these in object-oriented variants of Z [16]. The Z unification techniques described in [4] operate on two viewpoints at the same level of decomposition.

A unification of two viewpoints is constructed in two phases. In the first phase ("state unification"), a unified state space (i.e., a state schema) for the viewpoints has to be constructed. The essential components of this unified state space are the correspondences between the types in the viewpoint state spaces. At this stage we have to check that a condition called *state consistency* is satisfied. The viewpoint operations are then adapted to operate on this unified state.

In the second phase, called *operation unification*, pairs of adapted operations from the viewpoints which are linked by a correspondence have to be combined into single operations on the unified state. This also involves a consistency condition (*operation consistency*) which ensures that the unified operation is a refinement of the viewpoint operations. A similar procedure also needs to be executed for the initialisations of the viewpoints, and the adapted initialisations together need to be satisfiable. Automation of the checking of the consistency condition is possible to a large extent.

Relating different notations We have also considered comparing viewpoints written in LOTOS and Object-Z. This was interesting since it requires a bridge to be built between completely different specification paradigms. Although both languages can be viewed as dealing with states and behaviour, the emphasis differs between them. Our solution for consistency checking between these two languages was to exploit a behavioural interpretation of Object-Z.

Object-based languages have a natural behavioural interpretation, and there is a strong correlation between classes in object-oriented languages and processes in concurrent systems (see for example [35, 21, 30]). We used this correlation as the basis of a translation between the two languages, which was verified by defining a common semantics for LOTOS and Object-Z.

The ADT component of a LOTOS specification is translated directly into the Object-Z type system. To translate the behavioural aspect of a LOTOS specification we map each LOTOS process to an Object-Z class. Adopting this approach allows a natural mapping to be identified between many of the behavioural constructs in the two languages, for example, we find that process instantiation in LOTOS corresponds naturally to object instantiation in Object-Z.

To map a LOTOS process to an Object-Z class their observable atomic actions are related, i.e., events in LOTOS and operations in Object-Z. Therefore the translation will map each LOTOS action into an equivalent Object-Z operation schema. The Object-Z operation schemas have appropriate inputs and outputs to perform the value passing defined in the LOTOS specification. In addition, each operation schema includes a predicate to ensure that it is applicable in accordance with the temporal behaviour of the LOTOS specification.

The translation is given in [15], where it is verified against a common semantic model of the two languages. This model is based upon the semantics for Object-Z described in [30], which effectively defines a state transition system for each Object-Z specification. This model is used as a common semantic basis by embedding the standard labelled transition system semantics for LOTOS into it in an obvious manner. This provides a basis by which we can verify that the translation is correct, i.e., that the meaning of a term in one language is (bisimulation) equivalent to the meaning of that term after translation. [15] verifies the translation in detail.

Specifications, now written in a single language can be checked for consistency by using mechanisms to unify two or more specifications as sketched above.

The key component of the consistency checking strategy presented here is to be able to identify common refinements of multiple viewpoints with respect to the correspondences between the viewpoints. Such refinements can also be viewed as common *models* for the collection of viewpoints. These common models will typically be expressed in terms of the most primitive entities in the viewpoints. However, finding a suitable set of primitives is not always possible. In particular, different ODP viewpoints occur at different levels of abstraction, thus identifying one-to-one correspondences is almost certain to be impossible in general.

This difficulty raises many questions about how the viewpoints are specified, how to document the correspondences and how to deal with changes in the level of abstraction between viewpoints. General viewpoint models have great difficulty dealing with correspondences, having to use similarity checking [19] or low level common models [36]. However, because ODP has a fixed set of viewpoints with predetermined roles, more specific guidance can be provided for establishing correspondences. Similar leverage should be possible for UML given the known set of notations which are likely to be used. But it is also necessary to determine how best to describe the correspondences between the different aspects of a complete UML specification.

5 The challenge for UML

As we have discussed above there are a number of possible existing techniques for consistency checking partial specifications akin to those listed above. For example, consistency checking might involve a transformational approach or use a common semantics or construct unifications which are common refinements of related viewpoints.

We comment on each of these in turn.

5.1 Transformation

A transformational approach is concerned with checking consistency by translating one specification into another, e.g., the LOTOS to Z translation mentioned above. In order to do this one really needs a combined semantics in order to verify that the translation is correct.

Such a transformational approach is not necessarily applicable to all aspects of UML. This is because it is not clear that there is a single notation within UML that can encompass all concerns (indeed if there was UML wouldn't be composed of multiple notations). For example, it is relatively easy to translate between collaboration and sequence diagrams, both providing the designer with alternative ways to represent essentially the same behaviour.

Given a situation like that, appropriate consistency checks, defined for the particular notation, could then be applied. Thus this gives rise to a further issue for UML: namely defining consistency checks for the individual diagrammatic notations. As described above, this should be defined via appropriate refinement relations in the language concerned, and we comment on this below.

However, it is not always the case that a transformation is feasible. For example, it is not clear that information in all the UML diagrams could be transformed into a single diagram that is part of UML. Thus transformations are likely to be useful for defining consistency checks between particular diagrams, but they will not provide a complete solution to the problem.

5.2 Common semantics

One might wonder if a symmetric strategy, translating both viewpoint specifications to some common model might not be better. However, such models are necessarily at a semantic level, bringing in the problems of consistency checking at the semantic level as described above. Our experience suggests that in effect using the more expressive abstract notation as a common model is more effective.

However, what a common semantics *can* achieve is that it provides a basis by which to define refinement in UML, and this can then provide a practical consistency checking strategy. In terms of UML there is much work in this direction, as a single example we consider the work of Davies and Crichton [13].

In their work Davies and Crichton are concerned with providing a formal behavioural semantics to parts of UML. [13] shows how a semantics can be given to class, object and state diagrams by using the semantic model of CSP. This then induces refinement relations in UML based upon those in CSP.

For example, consider the class, object and sequence diagram given in Figure 6 (this example is taken from [13]).

The sequence diagram specifies that one particular trace is a desirable behaviour of the system, and thus they derive a trace in the CSP semantics that represents this information:

$$in.m, message.m, out.m, ack, ready$$

In a similar, but slightly more complicated, way [13] derives the set of allowable traces from the objet diagram.

A common semantics has thus been derived based, here, upon the traces model of CSP. More discriminating models in CSP can also be used, in particular, the failures-divergences model can capture information not present in a trace model, and which semantic model is used will depend on which and how the UML diagrams are put together.

With a common semantic model in CSP, the final piece in the consistency jigsaw is to use CSP refinement to define refinement for UML diagrams. This is defined in the obvious fashion: one UML specification is a refinement of another, if they are CSP refinements when mapped into the common semantics. Using different CSP refinement relations (trace, failures-divergences etc) allows one to define different refinement relations in UML which can be used as necessary.

It is then possible to show that the sequence diagram given above is indeed consistent with the object diagram (using the trace refinement relation).

This is but one example of relevant work in this area. There is much more to be said about a common semantic model for UML, but this is not the forum

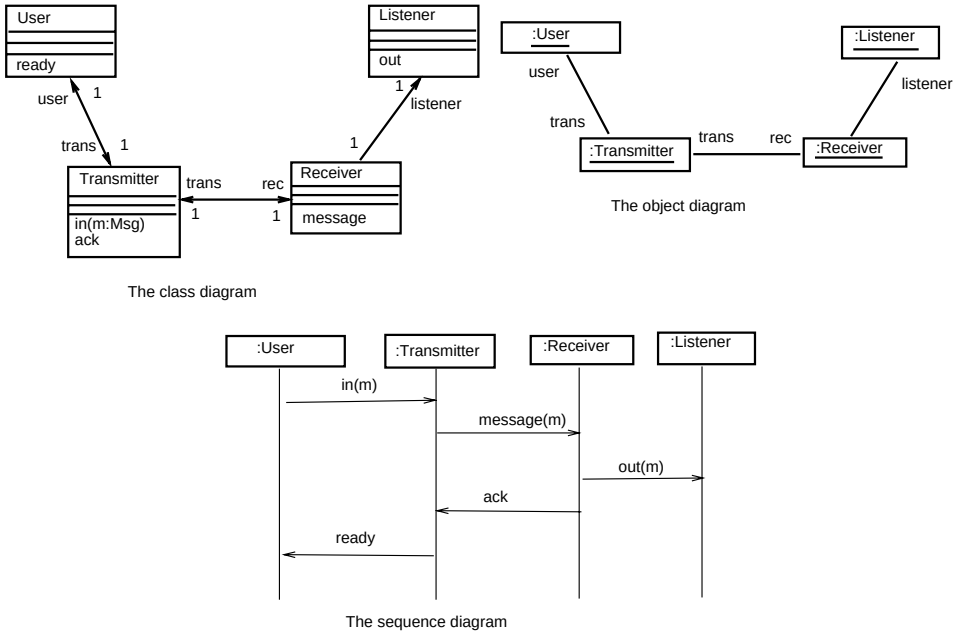

Fig. 6. A small fragment of UML specification

to do so. The point we are trying to make, however, is that common semantic models provide a means by which to define consistency and possible even develop tool support. Other common semantic models would provide similar frameworks, for example, Jurjens [26] defines a common semantic model using Abstract State Machines. He then uses this to define a notion of refinement for diagrams written in UML, again opening up the way to achieve consistency checking via finding common refinements.

However, the challenge for UML is to accept a common semantic framework. This will clearly not be easy since there may be competing forces, and it might even be the case that different semantic models are used.

5.3 Defining consistency

In fact, one can approach the definition of consistency from a number of angles. Starting with either a definition of conformance, a definition of refinement or a semantic model, consistency can be defined in a number of equivalent ways.

For example, given a notion of conformance between a UML specification S and a program P, denoted $conf(P, S)$, then a definition of consistency can be given as: $consistent(S1, S2) = \exists P \cdot conf(P, S1) \wedge conf(P, S2)$

Alternatively, as we have seen a notion of refinement between specifications generates a definition of consistency. If the refinement of one specification into

another is denoted $ref(S1, S2)$, then a definition of consistency can be given as: $consistent(S1, S2) = \exists S3 \cdot ref(S3, S1) \wedge ref(S3, S2)$

In fact, a notion of conformance can be used to generate the definition of refinement (which shows the definitions of consistency are equivalent) as: $ref(S1, S2) = \forall P \cdot conf(P, S2) \Rightarrow conf(P, S1)$

Finally, a common semantic basis for UML would provide a set of models of a UML specification, $mod(S)$. Consistency between specifications can then be defined as finding a common model: $consistent(S1, S2) = \exists S3 \cdot S3 \in mod(S1) \cap mod(S2)$

These definitions show, via a simple use of formalism, that there are a number of approaches to consistency, all of which turn out to be equivalent. The definition we have used in our work on ODP is that based upon a common refinement. Whichever approach UML took, much of the above existing work could be re-used.

6 Conclusions

In this paper we have tried to sketch how some of the existing work on consistency checking could be applied to a UML context. In reviewing the existing work there are a number of points around which the techniques are based.

Firstly, a specification in one viewpoint may need to be translated into a specification in another viewpoint. Clearly with UML such translation techniques may need to transform specifications from one notation into another.

Secondly, a collection of partial specifications may sometimes be integrated into one specification by a technique called unification. Each partial specification presents a partial description of the implementation. However, the ultimate aim is to develop an implementation that satisfies all partial specifications. At some stages during the development process it will therefore be necessary to compose different specifications in order to obtain an integrated view of the system to be developed.

Refinement plays an important part in consistency checking, but since a well-defined notion of refinement is necessary from a software engineering perspective anyway, this does not pose any additional burden on UML.

Finally, specifications are consistent with each other whenever it is possible to find at least one implementation that satisfies them simultaneously. Finding a common refinement is a sufficient consistency check. Consistency is also closely related to unification. Specifications can only be unified successfully if, and only if, they are consistent.

Existing work is tackling some of the issues necessary to realise these approaches to consistency, but if we have one conclusion, it is that we hope the UML community will re-use much of the foundational and practical work on viewpoints since some useful and relevant techniques have been worked upon for some years.

References

1. C. Bernardeschi, J. Dustzadeh, A. Fantechi, E. Najm, A. Nimour, and F. Olsen. Transformations and consistent semantics for ODP viewpoints. In H. Bowman and J. Derrick, editors, *FMOODS'97, 2nd IFIP Conference on Formal Methods for Open Object Based Distributed Systems*. Chapman and Hall, July 1997.
2. G.S. Blair and Jean-Bernard Stefani. *Open Distributed Processing and Multimedia*. Addison-Wesley, 1997.
3. E. Boiten, H. Bowman, J. Derrick, and M. Steen. Managing inconsistency and promoting consistency. In revision, available from `http://www.cs.ukc.ac.uk/research/tcs/consistency/tse.html`, September 1997.
4. E.A. Boiten, J. Derrick, H. Bowman, and M. Steen. Constructive consistency checking for partial specification in Z. *Science of Computer Programming*, 35(1):29–75, 1999.
5. T. Bolognesi and E. Brinksma. Introduction to the ISO Specification Language LOTOS. *Computer Networks and ISDN Systems*, 14(1):25–59, 1988.
6. H. Bowman, E. Boiten, J. Derrick, and M. Steen. Strategies for consistency checking based on unification. *Science of Computer Programming*, 1998. to appear.
7. H. Bowman, J. Derrick, P. Linington, and M. Steen. FDTs for ODP. *Computer Standards and Interfaces*, 17:457–479, September 1995.
8. H. Bowman, J. Derrick, P. Linington, and M. Steen. Cross viewpoint consistency in Open Distributed Processing. *IEE Software Engineering Journal*, 11(1):44–57, January 1996.
9. H. Bowman, E.A.Boiten, J. Derrick, and M. Steen. Viewpoint consistency in ODP, a general interpretation. In E. Najm and J.-B. Stefani, editors, *First IFIP International workshop on Formal Methods for Open Object-based Distributed Systems*, pages 189–204, Paris, March 1996. Chapman & Hall.
10. H. Bowman, M.W.A. Steen, E.A. Boiten, and J. Derrick. A formal framework for viewpoint consistency. *Formal Methods in System Design*, 21:111–166, September 2002.
11. CCITT Z.100. *Specification and Description Language SDL*, 1988.
12. G. Cowen, J. Derrick, M. Gill, G. Girling (editor), A. Herbert, P. F. Linington, D. Rayner, F. Schulz, and R. Soley. *Prost Report of the Study on Testing for Open Distributed Processing*. APM Ltd, 1993.
13. Jim Davies and Charles Crichton. Concurrency and refinement in the unified modeling language. *Electronic Notes in Theoretical Computer Science*, 7(3), 2002.
14. J. Derrick, E.A. Boiten, H. Bowman, and M. Steen. Supporting ODP - translating LOTOS to Z. In E. Najm and J.-B. Stefani, editors, *First IFIP International workshop on Formal Methods for Open Object-based Distributed Systems*, pages 399–406, Paris, March 1996. Chapman & Hall.
15. J. Derrick, E.A. Boiten, H. Bowman, and M.W.A. Steen. Viewpoints and Consistency - translating LOTOS to Object-Z. *Computer Standards and Interfaces*, pages 251–272, December 1999.
16. J. Derrick, H. Bowman, and M. Steen. Viewpoints and Objects. In J. P. Bowen and M. G. Hinchey, editors, *Ninth Annual Z User Workshop*, LNCS 967, pages 449–468, Limerick, September 1995. Springer-Verlag.
17. K. Farooqui and L. Logrippo. Viewpoint transformations. In J. de Meer, B. Mahr, and O. Spaniol, editors, *2nd International IFIP TC6 Conference on Open Distributed Processing*, pages 352–362, Berlin, Germany, September 1993.

18. A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke. Viewpoints: a framework for integrating multiple perspectives in system development. *International Journal on Software Engineering and Knowledge Engineering, Special issue on Trends and Research Directions in Software Engineering Environments*, 2(1):31–58, March 1992.
19. A. Finkelstein, G. Spanoudakis, and D. Till. Managing interference. In A. Finkelstein and G. Spanoudakis, editors, *SIGSOFT '96 International Workshop on Multiple Perspectives in Software Development (Viewpoints '96)*, pages 172–174, 1996.
20. A.C.W. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. Inconsistency handling in multiperspective specifications. *IEEE Transactions on Software Engineering*, 20(8):569–578, August 1994.
21. C. Fischer. CSP-OZ - a combination of CSP and Object-Z. In H. Bowman and J. Derrick, editors, *Second IFIP International conference on Formal Methods for Open Object-based Distributed Systems*, pages 423–438. Chapman & Hall, July 1997.
22. The RAISE Language Group. *The RAISE Specification Language*. Prentice Hall, 1992.
23. ISO 8807. *LOTOS: A Formal Description Technique based on the Temporal Ordering of Observational Behaviour*, July 1987.
24. ISO/IEC JTC1/SC21/WG7. Basic Reference Model of Open Distributed Processing. ISO 10746, 1993. Part 1 to 4.
25. ITU Recommendation X.901-904 — ISO/IEC 10746 1-4. *Open Distributed Processing - Reference Model - Parts 1-4*, July 1995.
26. Jan Jurgens. Formal semantics for interacting UML subsystems. In Bart Jacobs and Arend Rensink, editors, *Formal Methods for Open Object-based Distributed Systems V*, pages 29–44. Kluwer Academic Publishers, 2002.
27. P. F. Linington. Introduction to the Open Distributed Processing Basic Reference Model. In J. de Meer, V. Heymer, and R. Roth, editors, *IFIP TC6 International Workshop on Open Distributed Processing*, pages 3–13, Berlin, Germany, September 1991. North-Holland.
28. P. F. Linington. RM-ODP The Architecture. In K. Raymond and L. Armstrong, editors, *IFIP TC6 International Conference on Open Distributed Processing*, pages 15–33, Brisbane, Australia, February 1995. Chapman and Hall.
29. K. Raymond. Reference model of open distributed processing (RM-ODP): Introduction. In K. Raymond and L. Armstrong, editors, *IFIP TC6 International Conference on Open Distributed Processing*, pages 3–14, Brisbane, Australia, February 1995. Chapman and Hall.
30. G. Smith. A fully abstract semantics of classes for Object-Z. *Formal Aspects of Computing*, 7(3):289–313, 1995.
31. I. Sommerville. *Software Engineering*. Addison-Wesley, 1989.
32. J. M. Spivey. *The Z notation: A reference manual*. Prentice Hall, 1989.
33. M. W. A. Steen, H. Bowman, and J. Derrick. Composition of LOTOS specifications. In P. Dembinski and M. Sredniawa, editors, *Protocol Specification, Testing and Verification, XV*, pages 73–88, Warsaw, Poland, 1995. Chapman & Hall.
34. M.W.A. Steen. *Consistency and Composition of Process Specifications*. PhD thesis, University of Kent at Canterbury, United Kingdom, 1998.
35. A. Yonezawa and M. Tokoro. *Object-Oriented Concurrent Programming*. MIT Press, 1987.
36. P. Zave and M. Jackson. Conjunction as composition. *ACM Transactions on Software Engineering and Methodology*, 2(4):379–411, October 1993.