UML specification of distributed system environments

D.H.Akehurst, A.G.Waters

Technical Report : 18-99
Computing Laboratory, University of Kent at Canterbury,
Canterbury, Kent CT2 7NF, UK
{D.H.Akehurst, A.G.Waters}@ukc.ac.uk

The specification of distributed systems is a complex task, which is made easier by the use of object-oriented design methodologies. With the advent of UML as a standard notation for object-oriented software design, the application of this notation to the design of distributed systems is appropriate. The design of distributed systems involves both software and hardware specifications, however, the UML notation is primarily directed at the design of the software within a system and the facilities directed towards the specification of the physical environment are limited. Specification of the physical environment using UML can be achieved, but by using alternative parts of the notation to the proposed 'implementation diagrams'. Using the alternatives presented in this paper enables a satisfactory specification, which is for example, detailed enough for the automatic generation of performance models.

1 Introduction

Many different processes, methodologies and notations cover the design and development of the software in a system. However, very little consideration is given to the context in which the system operates and the physical environment that empowers it to execute.

From a performance engineering perspective both of these two areas are of vital importance. The relationship between the system and the context in which it operates determines the load on the system, and drives or determines its behaviour. The other relationship, the mapping between the physical structure of the system and its logical behaviour is equally important; this effectively determines the capability, of both the system and its behaviour, to respond to the stimulus from the operating context.

The UML language has many facilities for the description of the software of a system, and these facilities are being extensively exercised and developed by the community. The facilities for definition of the hardware and system context are however, minimal and in the case of the hardware description notation, appear to be rarely used.

This paper discusses how to use the UML for the specification of the physical environment of a distributed system, and the mapping of the application components onto the physical environment. The method described in this paper has resulted from the Performance Modelling of ATM Based Applications and Services

(PERMABASE) project carried out by British Telecom and the University of Kent at Canterbury, and funded by British Telecom.

The paper starts by placing the research within the context of the PERMABASE project. This is followed by some background information including a discussion regarding the UML version 1.1 implementation diagrams, and a look at the PERMABASE concepts with regard to the ISO Reference Model for Open Distributed Processing (RM-ODP, [14]) to see how its viewpoint specifications address the problem area.

The main body of the paper provides a critique of UML version 1.1 deployment diagrams as a means to illustrate physical environment specifications. The critique comes from a distributed systems perspective, by reference to conventions developed under the PERMABASE project.

The PERMABASE adaptation of UML version 1.1 was developed in parallel and independently of UML version 1.3[1], the paper continues by providing a comparison with features introduced in UML version 1.3, and commends version 1.3 as a significant improvement. The PERMABASE adaptations are shown to be inline with the version 1.3 improvements, but in fact go further. The extra solutions proposed could be usefully adopted to extend version 1.3.

The paper concludes with a look at other work related to the area of distributed system modelling, in particular discussing other proposed solutions to the specification of the physical environment and comparing them with our approach, and a summary of the research documented in this paper.

2 Overview of PERMABASE

The PERMABASE project ([2], [3], [4]) is concerned with bringing the advantages of performance modelling into the realm of the distributed system designer. Although the performance model generation is of primary concern within the project, it should be secondary to the system designer. The system, hardware, and software designers should be primarily concerned with the process of *designing* the system, and not spending their time and energy on the generation of a performance model.

The PERMABASE philosophy is therefore, to automatically generate performance models directly from the system design model. The rise of the UML within the design community as a standard notation for object-oriented software design, suggested that this would be a suitable notation, on which to base the project's prototype toolkit.

The project identified three distinct domains involved in the specification of the entire system design, and also the distinction between declaring the types (or classes) of components that exist in the system, and the instances of those components that form the deployed system.

Four model viewpoints were defined, one for specifying the declaration of components from each separate domain, and the fourth to specify the instances of the components and their connectivity. They are described as follows:

[1] In draft form at the time of writing.

1. *Workload* – Specification of classes of component considered as external to the system, but that drive or load the system.
2. *Application* – Specification of classes of component that form the software or logical behaviour of the system.
3. *Execution Environment* – Specification of classes of component that are physical components, providing the resources used by the application during system operation.
4. *System Scenario* – Defines the instances of declared components, and the connections between them, which form a specific system architecture or configuration.

Combining the specifications from each of the four domains, a Composite Model Data Structure (CMDS) of the entire system specification is produced. The CMDS is checked for consistency, and via a number of transformations, is finally translated into a discrete event simulation, performance model, of the system.

The results of executing the performance model are 'fed back' to the system designer(s), and can be used to adjust the system design specification such that it meets any performance requirements that have been imposed upon it.

It was the intention that each of the model viewpoint specifications be described using a standard notation and tool. However, the application model is the only specification area that already receives significant notation and tool support. Consequently, we determined to use, and adapt if necessary, the UML notation for the representation of all four domain specification areas. The adaptation of the UML to the representation of the Execution Environment and System Scenario specifications is discussed in this paper.

3 Background

Within the UML, the two implementation diagrams are the intended aspects of the notation for the portrayal of the type of information specified by the PERMABASE Execution Environment and System Scenario specification viewpoints. However, as we will see, the UML version 1.1 standard's definition of these diagrams is inadequate for the detailed specification of a distributed systems physical environment, as required by the PERMABASE project.

In this section, we discuss the UML implementation diagrams, indicating their unsuitability for the specification of distributed system physical environment issues. This is followed by an RM-ODP perspective on such a specification.

The discussion is based on early literature definitions of the UML version 1.1 that were available during the PERMABASE project. This sets the background for the ideas proposed in the paper. Since the project termination, and the subsequent release of UML version 1.3, the definition of the implementation diagrams has been significantly improved. The impact of the changes on the PERMABASE solutions is discussed later in the paper.

3.1 UML Implementation diagrams

There are two types of UML Implementation diagram, the Component Diagram and the Deployment Diagram. They both appear to be evolutions of diagrams originally part of the Booch Notation ([9]) – Module and Process Diagrams. Neither diagram is given much discussion in most of the literature on UML, and the impression of the author (although not founded on any quantified measures), is that the diagrams are not frequently used, or commonly understood how to be used.

The component diagram, as defined in the UML version 1.1 standard ([1]), "… shows the dependencies among software components, including source code components, binary code components, and executable code components." Fowler ([10]) indicates that components correspond to packages, and hence does not include component diagrams in his book.

If we look back to the component diagram's ancestor, the module diagram, Booch ([9]) states that "A module diagram is used to show the allocation of classes and objects to modules in the physical design of the system." with a heavy indication that the components tie up with (implementation) code files.

The Deployment diagrams, from the version 1.1 standard, "… show the configuration of run-time processing elements and the software components, processes, and objects that live on them. Software component instances represent run-time manifestations of code units." Fowler says much the same thing, that they "… show the physical relationships among software and hardware components in the delivered system." He also states that he has not "In practice, …seen this kind of diagram used much. Most people do draw diagrams to show this kind of information, but they use informal cartoons."

It can be seen from these descriptions that the diagrams are, as their collective name suggests, diagrams to aid implementation. The physical environment is assumed to be in existence, and the diagrams show the "deployment" of the software design over it. From the perspective of distributed system design, there are two major issues or difficulties with the use of these diagrams:

1. The physical environment is not necessarily a 'given', it may also be in the process of being designed.
2. The assumption of 'distribution transparency', and the fact that we are working with high level designs, eliminates the necessity to think about modules and code files; instead we wish to specify the location of objects and the connections between them.

The concept of components and deployment of components is still valid, but with a different emphasis. The components of the system are not organised into modules and are not only the objects, or classes, but are also the platforms, networks, users and any other entity that is part of the system or it's environment. The deployment of these components includes their instantiation (as the components are all definitions of types) and connection both physically and logically. Physical connections show which hardware objects are connected together and which platforms the software objects 'run' on. Logical connections show how the software objects are configured.

3.2 An RM-ODP perspective on physical environment specifications

The Reference Model for Open Distributed Processing (RM-ODP, [14]) is a well recognised standard related to distributed system design. It is not prescriptive in defining a specific design process or notation, but it identifies areas and concepts that, if addressed, will aid the success of the design, and that must be addressed in order that the design can be considered an 'Open Distributed System'.

The area of design specification relating to physical parts of the system within the RM-ODP, are covered within its Engineering and Technology Viewpoint models and definitions, and the mapping of Computational (Viewpoint model) objects to Engineering (Viewpoint model) objects. Also relevant are the distribution transparency mechanisms identified by the RM-ODP standard.

From the perspective of the application design (the logical or behavioural domain) we take a philosophy of distribution transparency, in particular, location transparency. Any object may communicate with any other object (provided it has a reference to it) and no attention is given to the mechanism used to enable that communication.

From a performance engineering perspective, it is vitally necessary to know the location of each object. There are two important relationships, dependent upon the object location, which significantly affect the performance of the system:

- Between the platform providing the execution resources and the objects using those resources.
- Between the location of the platform supporting an object and the location of the platforms supporting other objects it communicates with, and hence the network specifications providing the communications path.

The PERMABASE performance engine and the translation process between the design model and the performance model provide the location transparency mechanism.

In fact, the performance model engine, and the translation process from CMDS to the performance model, automatically provide all of the engineering transparencies and functionality required to support the execution of the designed software application (or computational objects).

There is only one Engineering Viewpoint piece of information that the system design is required to provide: The initial configuration of objects on platforms, or in RM-ODP terms, the allocation of Computational objects to Engineering nodes. All other engineering details are hidden by the automation.

The Technology Viewpoint requires the definition of the specific technological components used within the distributed system. From the perspective of the PERMABASE system model, this requires the specification of types of hardware component. The characteristics of the hardware components must be specified such that the quantity of resource provided, can be determined and used for performance model generation. The specification technique, should however, be directed towards *system design* not performance model design.

4 Physical environment specification

The specification of the physical environment of a distributed system is divided in PERMABASE between two specification domains: the Execution Environment and System Scenario specifications. The use of UML for the representation of these two specification domains is discussed in the following subsections.

4.1 Execution Environment

The information specified within this PERMABASE viewpoint is covered completely by the RM-ODP Technology Viewpoint requirements[2]. It is used to define the types of hardware components that are used within the distributed system, and the characteristics of those components.

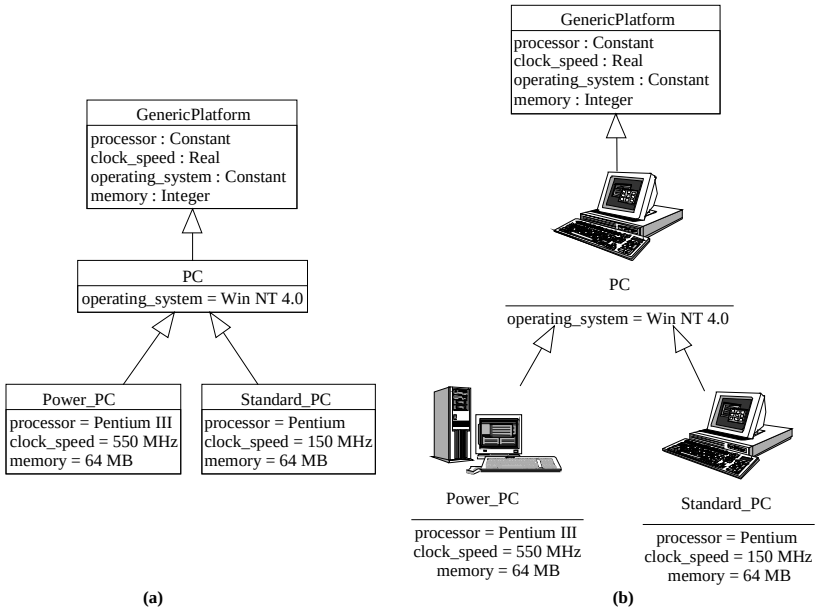

Fig. 1. Static Structure diagram showing a physical component type hierarchy

Although we are not specifying software or logical classes of component, we are still specifying *classes*, though in this case they are classes of hardware component, and hence a Class or Static Structure diagram is an appropriate means to represent the information specified.

Fig. 1a shows an example Static Structure diagram, specifying some Execution Environment (or physical) components. As can be seen in the figure, the

[2] Although conversely, it is recognised that there may be other information specified within an RM-ODP Technology Viewpoint specification, which is not specified here.

generalisation relationship can be used to group types of component, and/or aid with the re-use of components and (predefined) component libraries.

By using the UML facility to represent a stereotyped component by an alternative icon, the diagram can be made more visually recognisable, as in Fig. 1b. This technique, however, does not strictly conform to the UML standard. Stereotypes are intended for the definition of new semantic variations of the original UML meta-model element, not for defining alternative icons, on a per-project basis, for particular components.

4.2 System Scenario

The System Scenario PERMABASE viewpoint is for specifying the particular configuration of component instances and connectivity that represents the instance of the overall system that is being designed (or of which we wish to predict the performance). The specification is covered by both the Computational and Engineering Viewpoint models from the RM-ODP, relevant aspects of which, define the connectivity and instantiation of application (computational) objects, the allocation of those objects to hardware platforms, and the connectivity of the platforms.

We start by looking at how to use the current (UML version 1.1) deployment diagrams, for the representation of a System Scenario specification. We illustrate a number of problems with this approach, followed by our alternative representation using Object diagrams. The new approach demonstrates the solutions to the problems identified with existing deployment diagrams.

Finally we look at the future UML version 1.3 deployment diagrams, indicating how these change the significance of the problems.

4.2.1 Using version 1.1 deployment diagrams

Within the UML, the obvious place to look for notation to specify System Scenario information is the UML Deployment diagram, as it is intended for illustrating the deployment, or configuration, of a system. However, if we look at the example in fig. 2, we see a number of shortcomings.

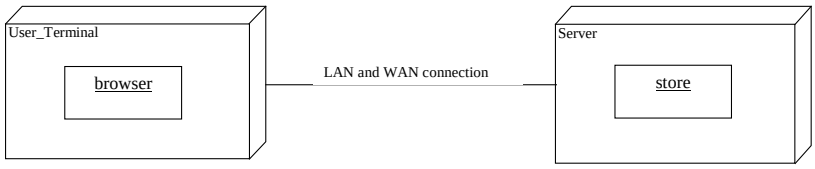

Fig. 2. A UML deployment diagram

We would like the diagram to represent a system of ten user terminals connected via a local area network and wide area network link to an off-site server. The users access a store of information, located on the offsite server, which they browse randomly at will.

The main drawback of deployment diagrams, is the lack of documentation on how to use them (as discussed above), but for this particular example there are specific difficulties in representing the required information:

1. The nodes are intended to represent the physical platforms that support the behavioural aspects of the system, however there is no mechanism for connecting the node *instance* with a specification of the class of type node, i.e. one of the Execution Environment components specified.
2. A simple line and label represent the communication path between the two nodes. Distributed systems are often deployed across complex network topologies, which cannot be adequately represented without a more sophisticated notation. Fig. 2 is a good (simple) example of this problem.
3. The objects supported by the various nodes should indicate their type[3] or class.
4. There is no mechanism for indicating multiple instances of nodes, other than drawing each individually. The above example requires the specification of ten user terminals; this has not been indicated in the deployment diagram of fig. 2.
5. There needs to be a mechanism for defining the connectivity of the application objects. In this example, there could be more than one store of information, with some browsers connected to one and some to another.
6. This type of diagram is likely to be used to show non-experts the configuration of the system being designed. Hence, the possibility of using icons that give a visual representation of the system components would be useful.
7. Although not demonstrated by this example, there is no mechanism for implementing hierarchy in the diagrams. Many (particularly large) systems could not sensibly be represented on a single diagram.

4.2.2 Using object diagrams as an alternative

An object diagram, defined in version 1.1 of the UML standard as a Static Structure diagram that shows only objects and not classes, is a more flexible and appropriate mechanism for showing the specification of a system scenario, or configuration.

The deployment diagram does not add anything, other than a three dimensional icon for node objects, above the notation provided by an object diagram, and in fact it is more limited in its notational capabilities. Using an object diagram, each of the problems with the deployment diagram representation can be addressed and satisfactorily solved.

Fig. 3 shows an object diagram illustrating the same system we attempted to show previously in the deployment diagram. This use of an object diagram to illustrate the system configuration, solves the problems, outlined above, of using a deployment diagram as follows:

1. Nodes, or physical platforms, are represented by objects, whose class can be given, which enables specification of the 'technology' characteristics of the node, via a 'node type'.
2. Communication paths are represented by a topology of 'communication' or network objects, each of which requires a class to be specified. The class defines the communication characteristics of the network object.

---

[3] It is quite possible (textually) to indicate the class type of objects on the diagram, but the version 1.1 standard does not include enough detail to show whether and if this is how the information should be specified.

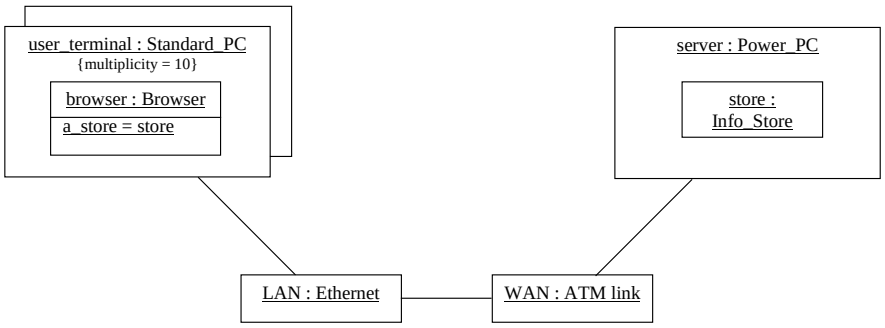

Fig. 3. Object diagram of system deployment

3. The UML standard defines quite clearly the semantics and notation of Static Structure Object diagrams, and how to specify objects (or class instances) within those diagrams. The representation of an object allows for an object-type to be specified, hence the class of the application objects can be shown.
4. Multiple instances can be shown, graphically by using the Collaboration diagram notation for multiple objects, and a tagged value to indicate (textually) the number of instances represented. (Strictly speaking this is not standard UML, though the graphical notation would be recognisable from the Collaboration diagram notation, and could be omitted if strict conformance is required.)

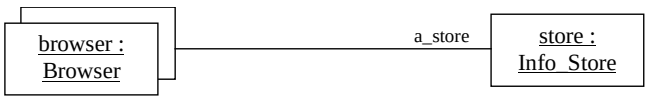

Fig. 4. An object diagram showing the logical configuration

5. The application object connectivity, or logical configuration could be shown in a number of ways. The information that is required to be illustrated, is that the value of an attribute in one object that references another object is set to the value (name) of some other object of the correct type. One way would be to illustrate it textually, within the object's attribute value compartment as shown in fig. 3. An alternative is to show the application configuration as a separate 'logical configuration' object diagram, using association (instance) links to connect the objects. This method is illustrated in fig. 4.
6. The diagram can be given more visually recognisable graphics (fig. 5), by the use of stereotype icons (as with the Execution Environment specification defined above). Again, this is not strictly a correct use of the stereotype mechanism, but is similarly used by Bourdeau et al. in [8]. When using alternative icons for representation of the nodes, one cannot show containment graphically by enclosing the object inside the node box, hence an association link between an object and its supporting node is used to convey this information.

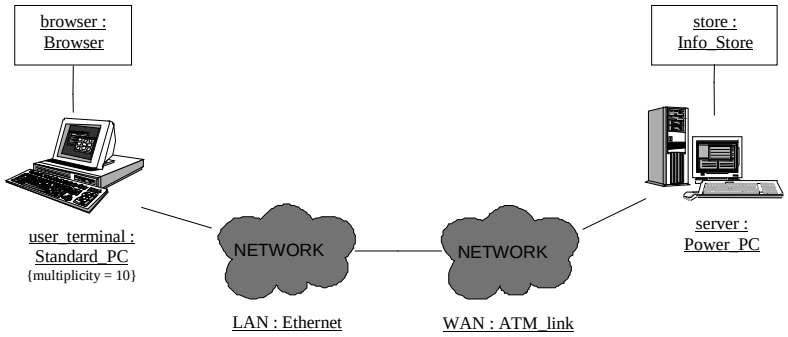

Fig. 5. System configuration using alternative icons

7. The issue of hierarchical decomposition is covered in the following section.

4.2.3 Hierarchical Decomposition / Subsystems

To handle the specification of large distributed system configurations it is necessary to break the system up into a number of subsystems. The whole system is composed of components, some of which may be subsystems, and the subsystems are likewise composed of components and possibly sub-subsystems etc.

Although the UML (version 1.1) semantics support a subsystem concept, they are not given much attention in the early literature, nor are they mentioned in the notation guide. A subsystem, according to the semantics, is both a classifier and a package, providing the capability of hierarchical containment and instantiability. The subsystem itself does not have any behaviour, and the instance of a subsystem equates to the instance of its component parts. A subsystem may have interfaces, through which the behaviour of its contents can be accessed.

We wish to use the subsystem concept within the specification of the system's physical configuration, and therefore, connections to the subsystem represent physical connectivity to components of the system. This requires, similarly to behaviour decomposition, an interface-like concept through which the connections can be made; a more appropriate name is an 'access point'.

Externally to the subsystem an access point works like an interface, showing connectivity to the subsystem and its components. Inside the subsystem, there must be a representation of every distinct access point referenced externally. The internal representation should be used to show connectivity of the access points to internal components of the subsystem.

Although subsystems are tied very closely to this configuration (or system scenario) view of the system, they should still be included in the class and instance dichotomy. There are likely to be occasions where a subsystem is replicated within the system as a whole, hence implying the idea of instances of a particular subsystem design. It must also be possible to define 'multiple instances' of a subsystem, as we can with other components in this System Scenario specification. Fig. 6 shows an example of using the subsystem concept.

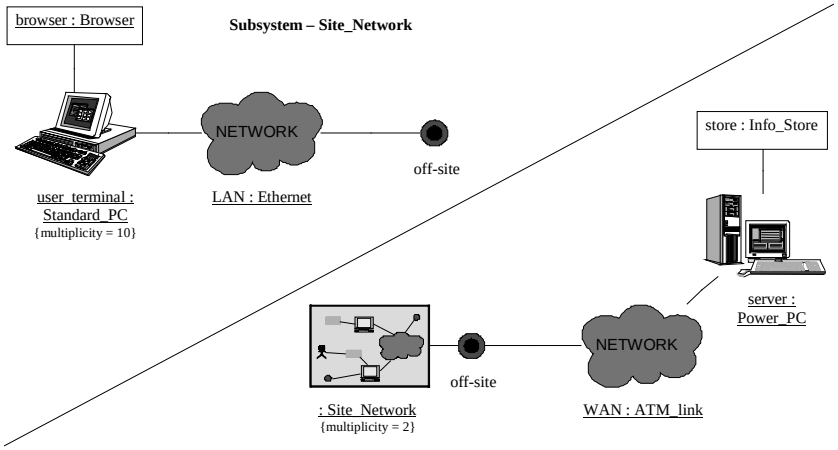

Fig. 6. System configuration using subsystems

The system shown is an extension to the example running through this paper, and assumes that the company (of ten users) has expanded, enabling a second set of ten user terminals connected by the same WAN connection to the information store, but on a separate Ethernet LAN.

4.2.4 The consequences of UML version 1.3

Recently two books have been released ([12] and [13]) which are based on version 1.3 of the UML[4]. These give a much fuller description of Deployment diagrams and how to use them. The new, refined, version 1.3 specification of the UML Deployment diagrams solves the majority of the issues and problems with the version 1.1 raised above, in a similar way to that outlined by our use of Object diagrams. The authors of [12] and [13] recognise the deployment diagram as being a variation on Static Structure (class or object diagrams), and illustrate their use as such.

If we use version 1.3 of the UML, the improved deployment diagrams enable a much more satisfactory representation of the system configuration. Some of the problems identified with the use of version 1.1 have been solved by the following changes to the deployment diagrams semantics and notation:

- A node is defined as a classifier, and it may have attributes, and tagged values.
- Nodes, like any other UML element, can be classes or instances.
- Objects can be correctly represented on a Deployment diagram, in the same way as on any other diagram.
- Networks can be represented as nodes.
- Deployment diagrams can exist as two variations, one showing the types of node that can exist, the other showing actual instances.
- The Subsystem concept can be used within any UML diagram, i.e. within Deployment diagrams.

[4] Although version 1.3 is still in draft form at the time of writing.

Using the new deployment diagram notation and semantics, our example's extended system configuration can be shown as in fig. 7, and as before we can use stereotype icons to give an alternative graphical representation identical to fig. 6.

Although this diagram appears to be UML version 1.3 compliant, it is unclear from the literature as to whether it is totally so. The use of «Subsystem» Package instances, and interfaces (as access points), both within deployment diagrams, are not explicitly defined as compliant uses of UML notation. However, the use of these components in this manner, does not directly violate any aspect of the current draft 1.3 standard.

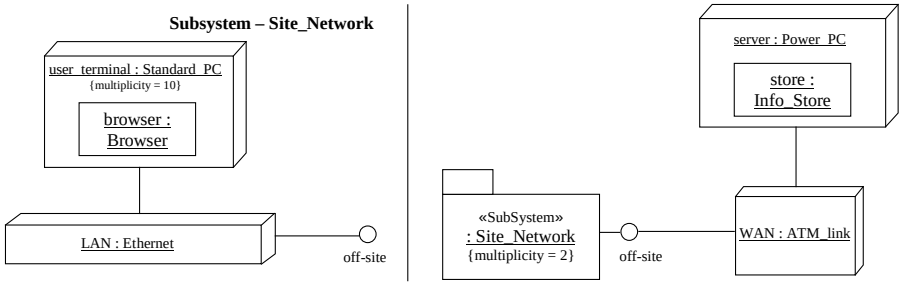

Fig. 7. System configuration, using version 1.3 Deployment diagram

The definition, within the new standard, of nodes as classifiers allows the Execution Environment declarations (Technology Viewpoint related definitions) to be illustrated using a deployment diagram showing node classes and the attributes of those nodes. However, the use of this mechanism to define 'node types' (as in fig. 1 above) would require the use of generalisation relationships within a deployment diagram, which is not compliant with the standard.

There is an alternative variation on 'node type' which adds another level of complexity. It is quite feasible that it is required to define a 'node type' that shows a specific allocation of objects on a node. A user_terminal for example, could be thought of as a 'node type' that supports a Browser object. In this case we have two conflicting ideas of the 'type' of a node when specifying an instance – the type defining its attributes, and the type defining its supported objects. A node instance can only have one 'type'.

A possible solution to this problem is to use generalisation relationships within a 'declaration' style of deployment diagram. Our example user_terminal could be declared as a subtype of Standard_PC, supporting a Browser object, and instances of this node class can be specified.

5 Related Work

The majority of work related to the design or modelling of distributed systems concentrates either on the application of behavioural elements of the system, or on the detailed modelling of the networks and communication mechanisms of the distributed system. The PERMABASE work covers a much broader set of requirements, enabling the design and performance prediction of the whole system, software and hardware

components, and includes a specification of the – possibly complex – workloads or driving component behaviour.

Jonkers et. al. ([5]) describe a system with a similar purpose to the PERMABASE system. Their approach identifies two domains, the entity domain and the behaviour domain, and is based on a bespoke modelling and design language – Architectural Modelling Box (AMB), a language developed to meet design prerequisites as well as performance modelling formalisms. Their system identifies most of the same concepts as identified in PERMABASE, but uses bespoke notations and tools for design specification. Of particular interest to this paper, is their identification and representation of entity components (comparable to execution environment components in PERMABASE). Their entity components (similarly) provide a resource, which is used by behavioural elements. There does not appear to be a mechanism for specifying how the system is loaded, though this is not a problem for their approach as the behaviour specifications are simpler than those supported by PERMABASE, modelling only the behaviour of a single process at a time.

The AUTOFOCUS project ([6] and others), although not specifically aimed at performance engineering, is another similar project to PERMABASE, which provides simulation, consistency checking, and code generation for a distributed system specification. Provision for the system specification is split into four views or description techniques, System Structure Diagrams, Data Types Definitions, State Transition Diagrams, and Extended Event Traces. Their System Structure Diagram attempts to describe the same area of the system specification addressed by this paper, but it is directed towards embedded systems and describing hardware bespoke to a particular system. The PERMABASE approach is more flexible, allowing also for the specification of general hardware components used for supporting a variety of software applications.

Woodside in [15] specifies the resource abstraction (physical components) directly as a queuing model, and although this is suitable for performance modelling purposes, we require a method more oriented towards system design.

Kandé et al. in [7] describe a relationship between UML diagrams and the ODP viewpoints. However, this is rather a simple approach, particularly with reference to the Technology viewpoint (which they do not attempt to represent in UML), and the Engineering viewpoint, represented by Component and Deployment diagrams which, as described earlier in this paper, are not really adequate for the job. The example given is not complex enough to adequately show how they intend to use the diagrams to give full engineering type specifications.

Bourdeau et al. [8], describe an interesting approach in which UML Collaboration diagrams are used to represent hierarchical context diagrams. They start by treating the system as an object, and then show the external entities interacting with it, and then break down the system into subsystems, showing the (sub)contexts in which the subsystems operate. They also recognise the importance of using domain specific icons, instead of abstract boxes, to make their diagrams more visually recognisable by non-UML experts.

The Real-Time UML book ([11]), makes some use of Deployment diagrams, though in a manner more suited to embedded systems than distributed ones. They define only the need to specify the deployment of active objects, whereas we require the location of all objects whether they are active or passive.

6 Conclusion

We have demonstrated, within this paper, that it is possible to specify the physical details of a distributed system design using the UML notation. The research illustrated in this document has arisen from the requirement of the PERMABASE project to enable the specification of distributed systems in such a way that a performance model can be generated. It is clear from the results of the project ([16] and those referenced therein) that the specification contains sufficient detail to automatically generate a performance model of the system.

We have shown that the method developed within PERMABASE is in accordance with the RM-ODP guidelines for the design of a distributed system's physical environment and is therefore appropriate for the design of distributed systems in general.

Although some of the solutions in this paper will be superseded by the release of UML version 1.3, this simply demonstrates that our method of use is in general compliant with UML thinking. Given that version 1.3 solves some of the problems in a similar way to that illustrated in this paper, we believe that the additional solutions proposed here are suitable amendments to the use of UML. In particular we advocate the following additions and uses:

- The use of generalisation relationships within a deployment diagram.
- The use of «subsystem» package classes and instances to enable a hierarchical structure for the whole system deployment specification.
- The use of «interface» classes as physical 'access points' (or a similar construct) to specify connections to «subsystems».

References

1. Joint submission to OMG - Rational Software, Microsoft, Hewlett-Packard, Oracle, Texas Instuments, MCI Systemhouse, Unisys, ICON Computing, IntelliCorp; The Unified Modeling Language, version 1.1; *OMG Technology Adoptions, ad/97-08-02 to ad/97-08-09*; November 1997.
2. Peter Utton, Brian Hill; Performance Prediction: an Industry Perspective (Extended Abstract); *Computer Performance Evaluation, Proceedings of the 9th International Conference on Modelling Techniques and Tools (Lecture Notes in Computer Science 1245)*; June 1997; pp. 1-5.
3. Peter Utton, Gino Martin; Further Experiences with Software Performance Modelling; *Proceedings of the First International Workshop on Software and Performance, WOSP 98*; October 1998; pp. 14-15.
4. Gill Waters, Peter Linington, David Akehurst, Andrew Symes; Communications software performance prediction; *13th UK Workshop on Performance Engineering of Computer and Telecommunication Systems*; July 1997; pp. 38/1-38/9.
5. H. Jonkers, W. Janssen, A. Verschut and E. Wierstra; A unified framework for design and performance analysis of distributed systems; *Proc. of the 3rd Annual IEEE Int. Computer Performance and Dependability Symposium (IPDS'98)*; Sept. 1998; pp. 109-118.
6. Franz Huber, Sascha Molterer, Andreas Rausch, Bernhard Schätz, Marc Sihling, Oscar Slotosch; Tool supported Specification and Simulation of Distributed Systems;

Proceedings International Symposium on Software Engineering for Parallel and Distributed Systems; 1998; pp. 155-164.

7. Mohamed Mancona Kande, Shahrzade Mazaher, Ognjen Prnjat, Lionel Sacks, Marcus Wittig; Applying UML to Design an Inter-Domain Service Management Application: A Case Study Based on the ACTS Project TRUMPET; *«UML» '98 Beyond the Notation*; June 1998; pp. 173.
8. E. Bourdeau, P. Lugagne, P. Roques; Hierarchical Context Diagrams with UML: An experience report on Satellite Ground System Analysis; *«UML» '98 Beyond the Notation*; June 1998; pp. 215.
9. Grady Booch; Object-Orinted Analysis and Design with Applications, second edition; *The Benjamin/Cummings Publishing Company, Inc.*; 1994.
10. Martin Fowler with Kendall Scott; UML Distilled: Applying the Standard Object Modeling Language; *Addison Wesley Longman, Inc.*; 1997.
11. Bruce Powel Douglass; Real-Time UML: Developing Efficient Objects for Embedded Systems; *Addison Wesley Longman, Inc.*; 1998.
12. Grady Booch, James Rumbaugh, Ivar Jacobson; The Unified Modeling Language User Guide; *Addison Wesley Longman Inc.*; 1999.
13. James Rumbaugh, Ivar Jacobson, Grady Booch; The Unified Modeling Language Reference Manual; *Addison Wesley Longman Inc.*; 1999.
14. ISO/IEC 10746-1/2/3; Reference Model for Open Distributed Processing – Part1:Overview / Part 2: Foundations / Part 3:Architecture; *ISO/IEC*; 1995.
15. C.M. Woodside; A Three-View Model for Performance Engineering of Concurrent Software; *IEEE Trans. On Software Engineering, Vol. 21, No. 9*; Sept. 1995; pp. 754-767.
16. P. Utton; PERMABASE Document Set Overview; Technical Report 9334:PERMABASE: BT: 047; *Systems and Software Strategy Unit, British Telecom*; 1998.