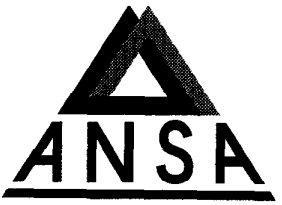
A SURVEY OF ORDERING ALGORITHMS FOR MONITORING

TR.040.00

ANSA

A Survey of ordering algorithms for Monitoring

TECHNICAL REPORT



A Survey of ordering algorithms for Monitoring

Yigal Hoffner

TR.40.00

May 1993

The material in this Report has been developed as part of the ANSA Architecture for Open Distributed Systems. ANSA is a collaborative initiative, managed by Architecture Projects Management Limited on behalf of the companies sponsoring the ANSA Work-programme.

The ANSA initiative is open to all companies and organisations. Further information on the ANSA Work-programme, the material in this report, and on other reports can be obtained from the address below.

The authors acknowledge the help and assistance of their colleagues, in sponsoring companies and the ANSA team in Cambridge in the preparation of this report.

Architecture Projects Management Limited

Poseidon House
Castle Park
CAMBRIDGE
CB3 0RD
United Kingdom

TELEPHONE UK	(0223) 323010
INTERNATIONAL	+44 223 323010
FAX	+44 223 359779
ARPA Internet	apm@ansa.co.uk

Contents

1 Monitoring and ordering

1.1 Abstract

This paper provides a survey of the ordering methods which can be used to order the monnitoring messages produced in a distributed system.

A taxonomy of the characteristics of re-ordering methods is presented as the starting point for the discussion of their relative advantages and disadvantages. Finally a recommendation on which minimal ordering methods should be implemented is given.

1.2 Introduction

When monitoring a distributed system it is necessary to collect monitoring messages from different parts of the system. Given the nature of communication in such systems, it is not usually possible to assume that the order in which the monitoring messages are collected will reflect the order in which they were sent. Furthermore, the relationship between the monitored event and the respective monitoring message cannot be assumed to be straight forward, unless certain precautions are taken. For example, making the monitored event and the generation of the resulting monitoring message indivisible.

In order to present the observer with a coherent and consistent picture of the monitored events it is necessary to re-order the monitoring message so that the new order will reflect the sequence of events which took place in the system. In the absence of a global clock or of sufficiently accurate set of synchronized clocks, it remains possible to order the message according to their causal relationship. The *re-ordering* of the messages is aimed at *reconstructing* the sequence of system events in a coherent and consistent way.

The reconstruction of the sequence of system events can be done in parallel to, or at the end of, the monitoring session. The two cases are referred to as *run-time* and *post run-time re-ordering*. In the latter case a sufficient delay can be assumed so that all the monitoring messages which have not arrived can be treated as lost; this simplifies the task of re-ordering. In the former case, if re-ordering is to be of use in real-time, there must be a limit on the period of waiting for the arrival of monitoring messages. Any messages which arrive after this period must either be ignored or be used to roll back the ordering to an earlier point. Unpaired send/receive system messages must be treated specially.

Two general approaches to reconstruction are possible. The first approach requires the modification of the communication infrastructure to guarantee that the order in which the monitoring messages arrive at the point of collation, reflects the order in which they were sent. However, this approach imposes a high level of interference with the behaviour of the monitored system. It thus contradicts a general requirement of the monitoring process,

that the level of interference with the monitored system is kept to a minimum. This solution will therefore not be further discussed in this paper.

An alternative approach is based on providing sufficient information (about the event) in the monitoring message, to enable subsequent re-construction of the sequence of system events. Such re-ordering methods consists of two main parts or processes:

- the process within the observed system which generates the information necessary for subsequent re-ordering
- the re-ordering process itself whereby monitoring messages are ordered according to the causal relationships among the events they represent.

A taxonomy of the characteristics of re-ordering methods, based on the distinction between these processes, is presented as a starting point for the discussion of their relative advantages and disadvantages.

Each method fits a different (set of assumptions about the) environment, has different advantages and disadvantages both in terms of the effect on the behaviour of the system, the ease of ordering, and the information which can be deduced from the monitoring messages.

Four methods are discussed:

- Simple method
- Lamport's Logical clock method
- Vector time method
- Causal history method.

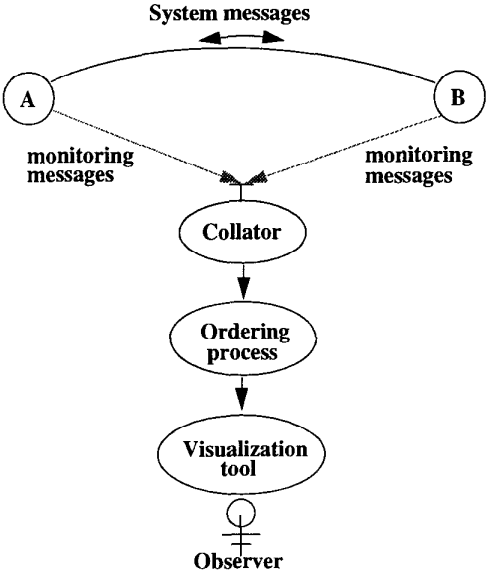
A discussion of their merits and the problems they pose is presented. Some conclusions regarding the circumstances in which a specific method is appropriate will be presented. Finally, a decision on which minimal ordering methods should be implemented is given.

1.3 System events, monitoring events and ordering

The observer is interested in **system events** which in this case are system messages exchanged between objects A and B (Figure 1.1.1)

The respective **monitoring events** result in *monitoring messages* sent to the collator. The collected messages are ordered and subsequently fed to the visualization tool [TR.41 93].

Figure 1.1.1: System messages, monitoring messages and ordering



1.4 Background material

The main sources for this paper are the following references: [BIRMAN 87], [BIRMAN 91], [COTROT 91], [FIDGE 91], [FORTIN 92], [FOWLER 90], [LAMPORT 78], [MATTERN 89], [PETERSON 89], [RAYNAL 90], [SCHWARZ 92].

2 Problems, assumptions and the scope of the work

2.1 Problems

A distributed system may introduce several complications in relation to monitoring and the process of reconstruction. For example, problems such as partial failure, incomplete observability, hidden communication channels, parallelism and pre-emption, unreliable communication, may cause:

Loss of information:

- lost monitoring messages
- delayed monitoring messages

Unexpected information:

- duplication of monitoring messages
- unexpected messages from parts of the system which are hidden from the observer in normal circumstances

2.2 Assumptions

Where possible, a communication platform which guarantees no loss or duplication of messages and a bounded delay of message arrival, should be used. Where such a communication platform is not available, or where the overheads associated with such guarantees are too high, it is necessary to modify the re-ordering method to deal with such cases.

Every monitoring message must have sufficient information to enable to identify the source of the message [LINDEN 91]. In order to ensure that a monitoring message is interpreted in the context in which it was generated, the message must be unique in this context. If the message leaves the context in which it was generated, context relative information must be added to it. Thus, we have identified another requirement from the collator.

2.2.1 Lost and delayed messages

If run-time re-ordering is performed, it is difficult to distinguish between lost and delayed messages. In such a case, delayed messages can be treated as lost messages after a specified length of time. Where necessary, late messages can be used to facilitate a roll-back option. In some run-time cases it will be possible to detect the delay of a message.

Similarly, in case of post run-time re-ordering it is possible to detect lost messages. In either case the observer can be notified.

When detecting a lost message or deciding not to wait for a delayed message, the un-paired message can be marked and treated specially.

- unexpected messages can be marked and treated specially

2.2.2 Duplicated and unexpected messages

Duplicated messages can be detected and discarded while the re-ordering takes place.

Unexpected messages will usually arise from:

- errors in system behaviour or program trying to perform illegal actions
- dynamic management of monitoring which results in extensions to the scope of monitoring

The re-ordering methods should attempt to place such monitoring messages in the context of the sequence of preceding events or alternatively, mark these messages as special cases. They can then be treated accordingly by the visualization process, which should be programmed to deal with special cases.

The communication in the system is based on a message passing paradigm in which communication channels are made explicit. It is assumed that no shared memory or hidden channels of communication are used.

No pre-emption of active processes between the occurrence of the system event being monitored and the generation of the respective monitoring message is allowed. Alternatively, if pre-emption is possible, the means to make the two events non-interruptible is available.

2.3 Complete observability

This paper assumes that all the parts which the observer wishes to monitor are observable, that is, it is possible to activate the relevant monitoring facilities. The case of *incomplete observability*, where it may not be possible to monitor certain parts of a system [HOFFNER 92], is not dealt with in this paper also the problem of *partial failures* is outside the scope of this paper.

3 A taxonomy of the characteristics of re-ordering methods

Causality based re-ordering methods consist of two major processes:

- the processes which takes place within the observed system and which generate monitoring messages carrying the information necessary for subsequent re-ordering
- the process in which the monitoring messages are re-ordered according to the causal relationships among the events they represent.

This distinction between the two processes outlined above allows us to introduce a taxonomy of the characteristics of re-ordering methods along the following points:

- the level of **interference** with the observed system:

(i) the processing required in the system to generate the information for re-ordering

(ii) the frequency of monitoring messages

(iii) the amount of additional information each system message has to carry

(iv) the amount of information which each monitoring message has to carry

- the **complexity** of the process of re-ordering of messages and the feasibility and ease of determining:

(i) complete or incomplete partial order. Incomplete partial order is where certain arbitrary decisions about the order of some events have been made

(ii) total order.

Another way of summarizing the above points is to ask whether the method **scales** as well as the number of observed processes increases.

Note that the taxonomy ties in with the fact that if less work is to be done in the actual re-ordering of monitoring messages, the more information about non-local events has to be in each process and its monitoring messages, and the more work is carried out in the observed system during the monitoring session. In other words, more local information has to be distributed to other processes. This inevitably means:

(i) that system messages have to carry more information thereby creating an overhead

(ii) to compute the relations between events at run-time also creates overheads.

There are a number of other characteristics:

1) how sensitive is a method to the loss of messages. In order words, whether causal dependence and independence can be determined in spite of lost messages

2) how flexible is the method in dealing with dynamic monitoring, that is, with extending and contracting the scope of monitoring.

4 Re-ordering methods

In general, the methods presented in this chapter are based on the idea of potential causality introduced by Lamport [LAMPORT 78]. The term potential causality derives from the principle that each cause has to precede its effect. The principle of such cause and effect can be applied to events in distributed systems, where the events are the sending of a message and the receipt of a message. **Partial ordering** specifies which events occurred before or after other causally related events. In cases where events are not causally related, however, all partial ordering can provide is an indication as to which events may have occurred concurrently with other events, that is either before, after or simultaneously. The causal relationship or the causal precedence order is characterized by two constraints:

(i) events (receipt, transmission or an internal event) which take place within the same process are totally ordered

(ii) the sending of a message must always precede its arrival at the recipient.

A **Total ordering** imposed on a partial ordering arbitrarily determines the order of events (which are not causally related). In other words, which may have occurred before, after or simultaneously, but there is insufficient information to determine what actually happened.

Some re-ordering methods provide a partial ordering of events without imposing any arbitrary decisions concerning ordering of unrelated events. We refer to such a case as a *complete partial order*. Where a re-ordering method imposes some arbitrary decisions on the order, it is referred to as an *incomplete partial order*. An example of an incomplete partial order is the information from Lamport clocks.

Four re-ordering methods are presented:

- The Simple monitoring [COUTROT 92]
- Lamport clocks [LAMPORT 78]
- Vector time [MATTERN 89]
- Causal history [PETERSON 89].

Each method is explained in terms of the taxonomy of the characteristics of re-ordering methods presented in Chapter 3.

4.1 The Simple re-ordering method

This method is based on the transmission of a monitoring message for each event of interest in the observed process, so that each monitoring message carries sufficient information to allow subsequent reconstruction of the events in the system.

The method requires the following:

1) a monitoring message is required for each event of interest in the system. However, any loss of messages may result in the inability to derive causal chains of events.

2) each monitoring message has to include the following information:
   (i) source identity
   (ii) event type: send/receive/internal
   (iii) SEND: destination identity
   RECEIVE: source identity
   (iv) message identity
   (v) local event count
3) each process has to keep a local event count
4) each system message has to include at least:
   (i) source identity unique within the monitoring session
   (ii) message identity unique within the message source.

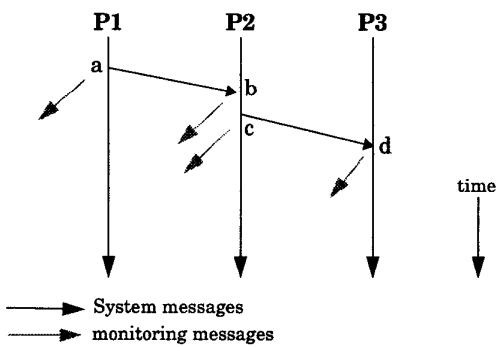
Note, that the identity of a process, source, destination and message have to be unique within a given context [AR.002 93].

The re-ordering process consists of two parts; a sort of local events related to each process followed by a cross correlation of related send and receive messages in different processes. Such an algorithm is given in "An Algorithm for Ordering Messages" [COUTROT 91].

The simple re-ordering method provides sufficient information to do a complete simulation of the events and to define a partial and total order, to detect any causal dependencies as well as independencies.

In order to detect any causal relations or independence it is necessary to have all the messages which form the dependency path [Figure 4.1].

Figure 4.1: Causal dependency detection in the simple re-ordering method requires a monitoring message from each related event



Any loss of a monitoring message which represents the transmission or receipt of a system message will prevent any causal dependencies in which the last event participated to be derived. The method is thus highly sensitive to loss of monitoring messages.

The necessity to send a monitoring message per event, which may not directly be of interest except for the fact that it is part of a causal chain of events, may create significant overheads.

The method allows a complete simulation of the events of interest in the system and thus facilitates the detection of causal dependence and independence relations.

The information which has to be included in the system message does not create a considerable overhead and in most systems is available anyway. (In ANSAware the Session Identity together with a unique message identity within the session provides the necessary information [AIM 93]).

4.1.1 Implementation issues

From the point of view of the donor system this is probably the simplest method to implement. All that is required is to generate a monitoring message per event of interest with the specified information. The monitoring management facilities should allow the dynamic activation of monitoring message generation in each process as the need arises.

It is easy to extend the method dynamically at run-time to incorporate additional objects, by activating the relevant monitoring message event generations in the objects. This requires the existence of appropriate management mechanisms discussed in [HOFFNER 93].

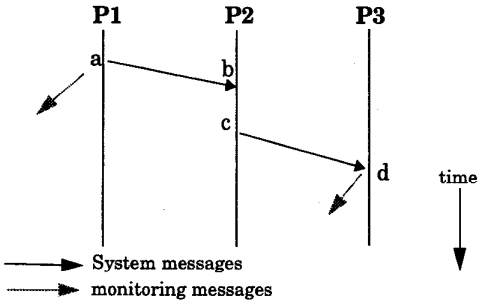
4.2 Lamport logical clock

This re-ordering method is based on each process having a logical clock which is updated both as a result of internal events and also when external events take place. The value of the clock is carried with the message sent between the processes.

The method requires the following:

1.a monitoring message is required for each event of interest. However, in order to obtain a partial order of events in a sequence which takes place in more than two processes, it is not necessary to have all the messages from all the events. It is sufficient to have a logical clock implemented in all the processes of interest (fig 4.2), without any messages from P2, provided the logical clock algorithm is implemented in P2. This will ensure that the logical clock of P3 at event d > logical clock of P3 at event c, thus maintaining the relation a -> b and b-> c and c ->d implies a -> d.

Figure 4.2 Partial ordering can be obtained with Lamport's clock without requiring a monitoring message from each related event



Note that in the absence of intermediate messages it is not possible to derive causal dependence or independence relation.

1) a monitoring message is required for each event of interest. The method guarantees that causally related events will have the appropriate clock values regardless of lost monitoring messages.

2) each monitoring message has to include the logical clock value in addition to all the other information specified in the simple method.

3) each process has to do the following:

Upon the receipt of a system message with the clock value h of the sender, the receiver does *hi = max(hi, h) + d*, where *hi* is the local clock value and d is the clock increment, *d>0*, and usually *d=1*.

4) each process message has to contain the senders logical clock value in addition to the information specified in the simple re-ordering method.

Lamport's Logical clock provides an incomplete partial ordering of events which can easily be turned into a total ordering consistent with the given partial ordering. This can be obtained from the resulting partial order by arbitrarily ordering any two events which have the same logical clock value. One way of doing this is to arbitrarily join the Logical-clock value and the process-ID: *<Logical-clock.process-ID>*.

The problem with Lamport's logical clock is that it does not maintain sufficient information to determine whether two events are dependent or independent of each other. Also, unless the method is extended to include a local event count, it is not easy to detect loss or delay of message when doing run-time re-ordering. Where causal dependency and independence are required this method is insufficient.

4.2.1 Implementation issues

The system overheads associated with this method are relatively small making this method attractive, particularly in view of the ease of implementation in the system and the total ordering algorithm.

Also in order to be used effectively for run-time ordering it is necessary to include a local event count to help with the ordering of local events.

It is easy to extend the method dynamically to accommodate extending the scope of monitoring in a monitoring session.

4.3 Vector time

The concept of vector time [MATTERN 89] provides a generalization of Lamport's Logical clocks. Instead of a single clock value, each process is assigned its own clock which only it can update. Together, the clock values form a vector, and each process which participates in the observed interactions has the entire vector. Each process updates its own clock in the vector as internal events occur. The updating of the other clocks takes place only when an incoming message arrives in the respective process and has a specific clock value which is higher than the value in the local vector.

The method requires the following:

1) a monitoring message is necessary for each event of interest. However, in order to detect causal dependency and independency of events it is not necessary to have a monitoring message generated for any of the intermediate events, provided that the intermediate processes involved participate at run-time by updating their vector and include it in their sent messages.

2) each monitoring message has to include the current value of the vector time.

3) each process Pi has to:

   (i) execute $vh_i[i] := vh_i[i] + 1$ whenever an internal event takes place

   (i) upon receipt of a message (m, hk) the receiving process Pi updates its vector by doing: $j \in 1..n : vh_i[j] := \max(vh_i[j], vk[j])$.

4) each process message has to carry the current value of the Vector time.

The Vector time method provides an easy way to detect causal dependence and independence and hence allows easy derivation of the partial order and total orders. In order to determine whether two events are related:

Definition. vh and vk are two n dimension clock vectors.

$$vh \leq vk \Leftrightarrow \forall j : vh[j] \leq vk[j]$$

$$vh \geq vk \Leftrightarrow \forall j : vh[j] \geq vk[j]$$

$$vh \,||\, vk \Leftrightarrow \text{not } (vh \leq vk) \text{ and not } (vk \leq vh)$$

let a and b be two events time stamped respectively with the vectors vh and vk; we have:

$$a \rightarrow b \Leftrightarrow vh < vk$$

$$a, b : \text{independents} \Leftrightarrow vh \,||\, vk$$

4.3.1 Implementation issues

The major advantages of this method lies in the ease at which causal dependence and independence of events can be determined, both at run-time and post run-time reconstruction. Also the resilience of the method to lost messages and the possible reduction in monitoring messages is appealing.

Different observers of the same monitoring session will get the same partial orders of events regardless of whether they receive all or only some of the monitoring messages. This is not the case with the Simple method.

The major drawback of this method is that when trying to extend the scope of monitoring to incorporate new object, it is necessary to extend the vector and notify all participants of the change.

In addition, the size of the vector will depend on the number of objects observed and this could be extremely large in some monitoring sessions. The overheads can thus be prohibitive.

In order to deal with the problem of a large number of processes participating in interactions and the overheads associated with the inclusion of the vector in system messages, it is possible to use vector compression techniques [SCHWARZ 92].

To deal with a changing number of processes it is possible to provide additional information relating the process (PId) with its time value in the vector time. Instead of a simple array, the vector time consists of a sorted linked list of records or SEQ of RECORDS { PId : integer; h : integer }; thus, simple array indexing is substituted by a list search. The search can be optimized given the list is sorted. Extending the scope of the monitoring session can now be dealt with by the insertion of a new record with the PId of the new process and a zero value for its clock. Contracting the scope of the monitoring session can now be dealt with by the deletion of the relevant record. To update the list when a message is received should take no longer than the equivalent array operations.

4.4 Causal history methods

Editorial: this method has not been investigated sufficiently due to lack of reference material which discusses its actual implementation. Any references would be welcome!

These methods entail encoding the partial ordering of previously received message into each system message [SCHWARZ 92]. The origin of such methods is in protocols which guarantee the order in which messages are processed at several destinations, are consistent [BIRMAN 87] and [BIRMAN 91]. An example of such a protocol is Psync [PETERSON 89].

This method is more applicable for cases where the ordering in which difference destinations process a sequence of messages should be the same.This however, is of no concern to monitoring unless there are several observers and there is a requirement to provide each observer with exactly the same view of the events in the system.

5 Discussion and conclusions of implementation issues

The Simple method can be regarded as a default method, since events of interest should be made observable regardless of whether ordering is an issue or not [AIM 93]. The respective monitoring messages for these events are expected to be implemented.

The Simple method requires the least changes to application or infrastructure code. Most of the necessary information is available in the communication protocols used in distributed systems. The problem is how to design proper interfaces which allow access to the required information.

Integration with the management of monitoring is also necessary and this brings up questions such as:

- when monitoring is switched on in a capsule, does the entire capsule participate in updating the clock
- what happens when an object participates in more than one monitoring session.

Further discussion should be carried out in light of the documents produced since this work was wrriten. In particular [HOFFNER 92], [HOFFNER 93], [TR.39 93], [AR.010 93] and [TR.041].

[AIM 93]
ANSAware 4.1 Implementation Manual, APM,Architecture Projects Management Ltd., Poseidon House, Castle Park, Cambridge CB3 0RD, UK (1993).

[AR.003 93]
van der Linden, R., AR.003: "The ANSA Naming Model", Architectural Report 003,APM,Architecture Projects Management Ltd., Poseidon House, Castle Park, Cambridge CB3 0RD, UK (1993).

[AR.010 93]
Hoffner, Y. "AR.010: Monitoring in Distributed Systems", Architectural Report 010, Architecture Projects Management Ltd, Posiedon House, Castle Park, Cambridge CB3 0RD, UK (January 1993).

[BIRMAN 87]
Birman, K. "Exploiting Virtual Synchrony in Distributed Systems", ACM Operating Systems Review, 21(5), 123-138 (November 1987).

[BIRMAN 91]
Birman, K.P., "Maintaining Consistency in Distributed Systems", Cornell University, Dept. of Computer Science (November 1991).

[COUTROT 92]
Coutrot, G. "An Algorithm for Ordering Messages", 5911/SYD08, Synergie Informatique et Development, (October 1991).

[FIDGE 91]
Fidge, C. "Logical Time in Distributed Computing Systems", IEEE Computer, 24(8), 28-33 (August 1991).

[FORTIN 92]
Fortin, D. & Valot, C. "Observability and Controllability of Distributed Systems and Applications", INRIA, B.P. 105, 78153 Le Chesnay Cedex, France, (1992).

[FOWLER 90]
Fowler, J. & Zwaennepoel, W. "Causal distributed breakpoints". In 10th Int. Conf. on Distributed Computing Systems [6], 134-141 (1990).

[HOFFNER 92]
Hoffner, Y. and Statham, A. "The Visualization of Distributed Systems", the Second International Conference on Computational Graphics and Visualization Techniques, Lisbon, Portugal, 14-17 Decmber 1992 (1992).

[HOFFNER 93]
Hoffner, Y. "The Management of Monitoring in Object-Based Distributed Systems", The Proceedings of the Third International

Symposium on Integrated Network Management, San Franscisco, April 1993 (1993).

[LAMPORT 78]
Lamport, J. L. "Time, Clocks, and the Ordering of Events in a Distributed System", Comm. ACM, 21(7), 558-565 (1978).

[MATTERN 89]
Mattern, F. "Virtual time and global states of distributed systems". In Parallel and Distributed Algorithms, 215-226. Elsevier Science Publishers B.V. North-Holland, (1989).

[PETERSON 89]
Peterson, L. L., Bucholz, N. C. & Schlichtig, R. D. "Preserving and Using Context Information in Interprocess Communications", ACM Transactions on Computer Systems, 7(3), 217-246 (August 89).

[RAYNAL 90]
Raynal, M. "Order notions and atomic multicast in distributed systems: A short survey", Technical Report 1197, Institut de Recherche en Informatique et Systemes Aleatoires. Rennes , France, (March 1990).

[SCHWARZ 92]
Schwarz, R. & Mattern, F. "Detecting Causal Relationship in Distributed Computations: In Search of the Holy Grail", Department of Computer Science, University of Kaiserslautern, D-6750 Kaiserslautern, Germany (1992).

[TR.39 93]
Hoffner, Y. "TR.39.01: Management in Object-Based Federated Distributed Systems", Architecture Projects Management Ltd, Posiedon House, Castle Park, Cambridge CB3 0RD, UK (January 1993).

[TR.41 93]
Hoffner, Y. "TR.41.00: The Visualization of Distributed Systems", Architecture Projects Management Ltd, Posiedon House, Castle Park, Cambridge CB3 0RD, UK (January 1993).