Multi-Level Modeling of Concurrent and Distributed Systems

Peter Tabeling
Hasso-Plattner-Institute for Software Systems Engineering
P.O. Box 90 04 60, 14440 Potsdam, Germany
tabeling@hpi.uni-potsdam.de

Abstract

This paper presents an approach to the description of distributed and concurrent systems in which a system's compositional structure and behavior are closely related. Central ideas behind this approach are the concept of virtual locations which is derived from our intuitive understanding of physical locations as well as the separation of multiple modeling levels.

In the academic community, the terms "concurrency" and "distribution" are not used in a consistent manner, thus resulting in additional difficulties in the description of distributed and concurrent systems. The modeling approach presented here leads to a better understanding of these terms because it allows for both a clear differentiation of concurrency and nondeterminism and different interpretations of distribution. Furthermore, the interdependence of concurrency and distribution is illuminated.

Keywords: distribution, concurrency, interleaving model, snapshot, transaction, system modeling

1. Introduction

When we consider the large number of textbooks and lectures which deal with concurrent or distributed systems there is no doubt that these systems are common and important topics in academic teaching and publications. Furthermore, concurrency and distribution are typical characteristics of current commercial systems.

In this context, it seems unacceptable that the terms "concurrency" and "distribution" continue to have different definitions for different people. When studying related textbooks or papers, one can easily find contradicting interpretations of these terms as well as authors who criticize that inconsistency.

1.1 The Problem of Differentiating Concurrency and Nondeterminism

In publications about concurrent systems, the interpretation of "concurrency" often depends on the author or on the theoretical background. Nevertheless, two prevalent types of interpretations can be identified.

The first interpretation is closely related to net theory [1][2] where concurrency means causal independency of events. Two concurrent events can occur temporally-ordered or simultaneously. Lamport's definition, based on the "happened before relation" [3], belongs to the same category because the temporal order of concurrent events is also left open.

In the case of the second interpretation, concurrent events can take place in any order but not simultaneously. This point of view is often referred to as the "interleaving model" of concurrency. Typical examples are process algebras [4][5] and transition systems [6][7].

The relationship between these two "concurrency models" is illustrated by the example in Figure 1. A Petri Net with concurrency (I) can always be transformed - with the reachability graph (II) as an intermediate step - to a net which describes indeterminate behavior without any concurrency (III). In the case of the interleaving model, simultaneous occurrence of concurrent events would be omitted (in this case, the gray shaded transition in (III) was removed).

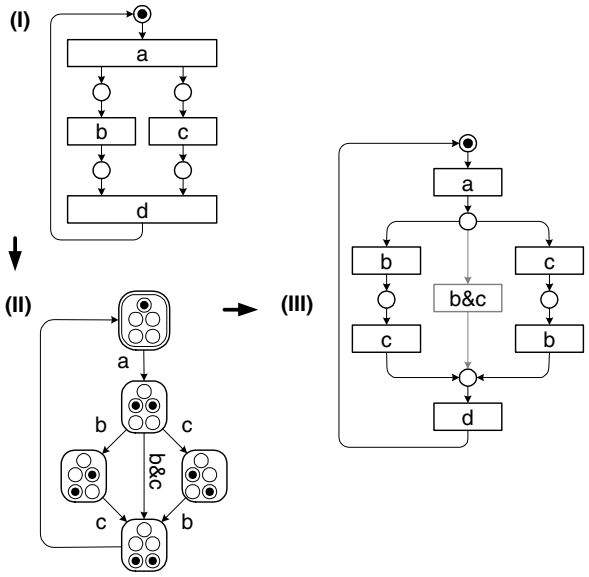

Figure 1: Transformation of Concurrency into Nondeterminism

As a reason for excluding simultaneous occurrence of events some authors refer to "typical hardware structures"[1] or assume that simultaneous occurrence simply cannot be observed.[2] These assumptions certainly do not apply in general. Therefore, it must be scrutinized if the interleaving model represents a sound understanding of concurrency. Manna and Pnueli ([7], p. 20) view this problem as an important one:

"The question whether the representation of concurrency by interleaving is adequate and acceptable or whether we should treat concurrency as a separate and unique phenomenon that cannot be reduced to nondeterminism, is one of the most debatable issues in the theory of concurrency."

Let us assume that we abandon the interleaving model and its questionable assumption that concurrent events do not occur simultaneously. Even in this case, the transformation of concurrency into nondeterminism (as shown in Figure 1) can still be done. The only difference is that the gray shaded transition in (III) would not be removed and the simultaneous occurrence of b and c had to be seen as an additional type of event (e.g. "e").

In general, this example shows that every concurrent behavior can be modeled as an indeterminate behavior - if we choose a suitable set of event types. It remains unclear which choice of event types is the "most appropriate". Manna's and Pnueli's question (see above) of concurrency as a separate and unique phenomenon or just a special case of nondeterminism, remains to be answered.

1.2 The Fuzzy Meaning of the Term "Distributed System"

Another frequently used but also ambiguous term is the term "distributed system". This ambiguity has also been criticized by authors such as Tanenbaum (see [8], p. 2):

"Various definitions of distributed systems have been given in the literature, none of them satisfactory and none of them in agreement with any of the others."

Indeed, some questions arise from these inconsistencies. To some, the existence of multiple computers being connected by a network is the characteristic feature of a distributed system.[3] This definition does not allow for a clear differentiation of distributed and nondistributed systems, because the term "computer" is too fuzzy, which, for example, makes it difficult to classify certain multi-processor systems.

Other authors view "spatial distribution of the system components" as the typical feature of distributed systems.[4] This raises the question about what distances must be given between system components for the system to be called distributed. Any length specification would be purely arbitrary.

Not all authors define distribution as a physical feature of a system. According to Broy, a system is distributed if it is "built from at least *conceptually* distributed components" ([11], p. 3) and Tel considers even "a collection of communicating processes"[5] distributed system.

Obviously, there is no consensus on whether distribution is a physical feature of a system or not. Furthermore, it remains unclear what "spatially" or "conceptually distributed" means, etc.

2. A Multi-Level Approach to System Modeling

An approach to the description of information processing systems is presented below, in which a system's compositional structure and behavior are brought into close relationship. An important element of this approach is the idea of virtual locations which is derived from our intuitive understanding of the term "location".

2.1 Physical Locations and Agents

An information processing system performs activities which are otherwise done by human beings. Therefore, it is always possible to imagine such a system as several cooperating persons (if we ignore complexity, speed and reliability issues [12], p. 246). In this case, for each peace of information existing in the system, there would be some physical medium (such as a note or document) that is kept at a certain place to which only some of the persons have access. Figure 2 shows an example of an information processing system in the broadest sense. Each of the two persons on the left continuously place playing cards on a stack. (We are only interested in the uppermost card of each stack.) The cards are placed at two different locations L_1 and L_2 which allows both persons to act independently. Each currently visible

1. see [6], bottom of page 372: "it is usually the case that two accesses to the same memory location do not overlap in time because of a hardware arbitration device." A similar argumentation is given in [7].
2. see [4], page 4: "The reason is that we assume of our external observer that he can make only one observation at a time; this implies that he is blind to the possibility that the system can support two observations simultaneously, so this possibility is irrelevant [...]."
3. [9], p. 2: "A distributed system consists of a collection of autonomous computers linked by a computer network [...]"
4. [10], p. 767: "A system is called distributed if its components are placed or could be placed at spatially separated locations [...]." (translated by author)
5. [13], p. 2: "[...] the definition [of a distributed system] includes software systems built as a collection of communicating processes, even when running on a single hardware installation."

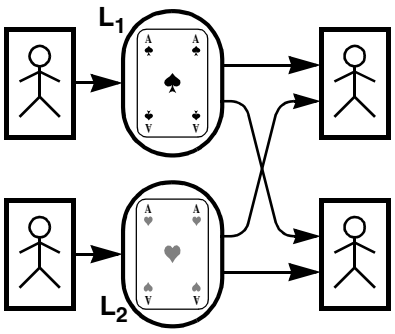

Figure 2: An Example of Physical Locations and Agents

card carries the information "card value" (7,8,9,...,ace) as well as the information "card color" (diamonds, heart, spade, clubs).

The process taking place at L_1 and L_2 is observed by two other persons (on the right). These persons observe events of type "value changes at location L_i" (V_i) and "color changes at location L_i" (C_i) ($i \in \{1,2\}$).

Two card events taking place at different locations can occur temporally-ordered or simultaneously. The same applies to card events taking place at the same location. (In the latter case only value changes and color changes can occur simultaneously.) This behavior can be described by the Petri Net (a) in Figure 3. However, most people will intuitively prefer Petri Net (b), which describes the process as two event sequences (one per location). Here, the simultaneous occurrence of a value change and a color change at the same place is considered an additional type of event (V_i&C_i).

In addition to intuition, there are additional reasons to choose Petri Net (b) instead of Petri Net (a). The event sequence at each location is the result of a sequence of elementary actions. Because each of these activities requires a certain period of time there must be a lower limit to the time interval between two events occurring at the same location. Therefore, we may assume that each observer is always able

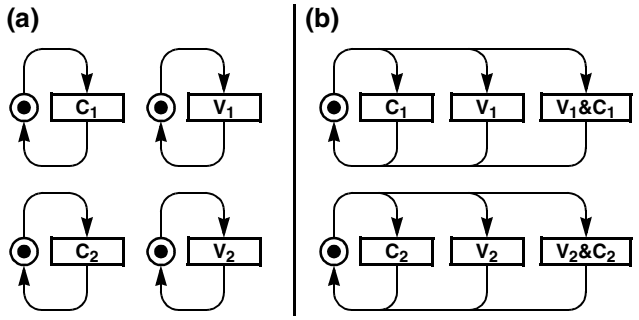

Figure 3: Alternative Behavior Models for Figure 2

to distinguish between two observed events. Simultaneous events at the same location (color change combined with value change) are the result of a single elementary action, thus these events are observed by all as "parts" of one single event (the next card is put on a stack).

In contrast, concurrent events at different locations are the result of independent activities. Therefore, the appearance of new cards at different locations can occur within an arbitrary short time period. In this case it depends on the observer's individual time resolution, whether two concurrent events are perceived as simultaneous or temporally-ordered.

Causal dependency between two events can only be achieved by an activity which also takes a certain amount of time. (For example, one of the card-placing persons could wait for the other person's next card before putting his own card on the stack.) Hence, there must be a lower limit to the time between causally related events. Again it is a sound assumption that these events can always be observed in the corresponding temporal order.

The following conclusion can be drawn from these observations. Concurrency at one location is a modeling artifact which can be eliminated by combining simultaneous events to a single event, i.e. this kind of "concurrency" is simply a special type of nondeterminism and thus should be modeled as indeterminate sequential behavior. In contrast, concurrent events at different locations should always be seen as separate events, because in general "simultaneity" of such events will depend on the observer. This is the nature of "true" concurrency which cannot be represented as an indeterminate sequential behavior.

2.2 Virtual Locations and Agents

The view of information processing systems as given in section 2.1 is the basis for a modeling approach [14][15] that brings system behavior and a system's compositional structure into a close relationship. The central ideas behind that modeling approach are outlined below.

In order to illustrate a system's compositional structure, each piece of information existing in the system is assigned a virtual location. These locations are virtual because they are imaginary locations where information can be placed or observed, but no assumption about the physical location of information is linked to them ([14], pp. 75). The information at a certain location can be a simple value such as an integer or an arbitrary structured value such as a complex data structure.

In addition to locations, there are active system components - called *agents* - which can place information at locations as well as observe locations. (As in the case of locations, an agent can be totally unrelated to the physical system structure.)

Figure 4 gives an example of a compositional structure which consists of two agents and eight locations. Each location can be used by a single agent as a private storage (L_1 for example) or as an interface to other agents (L_6 for example). Interfaces can be used by agents as output locations (L_7 as used by A_2, for example) or as input locations (L_7 as used by A_1, for example). For each location, there is at least one agent which generates the observable values; also, a location can be observed by multiple agents. Each agent generates values for at least one location and observes at least one location.

A given compositional structure in terms of virtual locations and agents does not imply any physical system structure.

Virtual locations can nevertheless be called locations because we require them to be equivalent to physical locations (see section 2.1) with respect to observation:

Virtual locations are imaginary physical locations to which the following constraints apply:

- *temporal consistency*
  The observed temporal order of events occurring at the same location does not contradict the real temporal order (i.e. the order in which the events have been generated). Any two observers of the same location always observe the same temporal order of events.
- *causal consistency*
  The observed temporal order of events does not contradict their causal order. If concurrent events do not occur at the same location, different observers can have different views of the temporal order of these events.

The compositional structure is the basis for describing a system's behavior. A value change taking place at a certain location and at a certain point in time is called an *event*. This

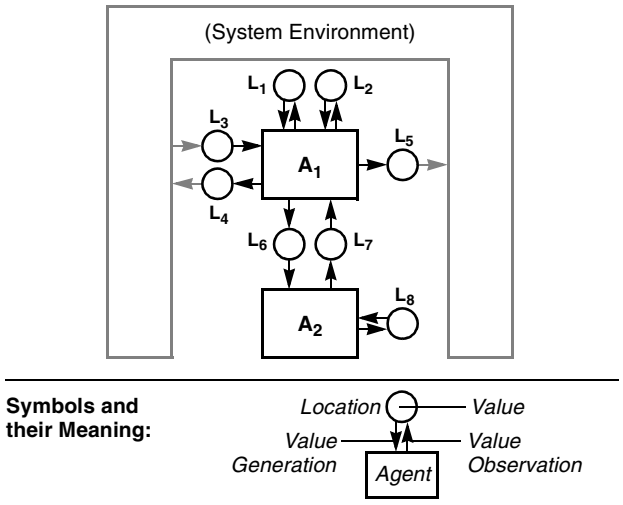

Figure 4: Example of a Compositional Structure

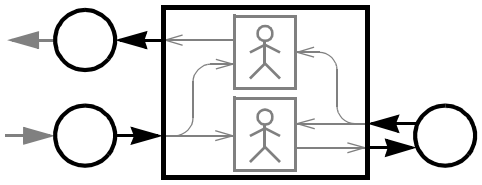

Figure 5: Illustration of a Virtual Agent

definition reflects our intuitive understanding of the term event, because an event is something happening at a certain time and place. It implies that at a single location there will always be an event *sequence*. Of course it is still possible that separate values within a data structure change simultaneously at a single location. Nevertheless, these changes are not to be seen as independent events but as "parts" of one single event. Simultaneous events and "true" concurrency can only be given if more than one location exists.

The sequence of values appearing at a certain location is the outcome of a sequence of elementary activities, called *operations*. When performing an operation an agent assigns a new value (the operation result) to a certain location. This value is derived from values which are observed by the agent at one or more locations.

In this way, every agent performs a sequence of operations for each location it uses as output location or state storage. An operation result can be an output value which is computed from a state value observed at a storage location and placed at an output location. It could also be a new state value that is derived from the previous state value and an input value, etc.

For each operation sequence an agent performs, i.e. for each storage location or output location, we can imagine a person "inside" the agent (see Figure 5). The agent shown has one output location and one state storage, thus two imaginary persons are presented. When performing an operation an agent (i.e. the corresponding imaginary person) reads values from the observed locations in order to derive the operation result:

The values being read during one operation represent simultaneously observable values from the point of view of a virtual agent.

These values represent a "snapshot" [16] because causal consistency is required for the observation of multiple locations.[1] To assure temporal consistency, the sequence of values read from one location during an operation sequence must not contradict the real value sequence.

1. Actually the causal consistency required here is a stronger type than the causal consistency required in [16]; see [14], pp. 77 for details.

2.3 Multiple Models at Different Levels

An essential feature of a complex information processing system is that the system must always be described on more than one level of abstraction [18]. Therefore, multiple models have to be defined which are more or less close to the physical level. A model is preordered to another model if the second model can be derived from the first by a design decision.[1] Depending on the amount of design decisions, a relatively complex hierarchy of partially ordered models is presented. This is another important concept of the modeling approach presented here.

Furthermore, it should be underlined that the close relationship between compositional structure and behavior is present at each modeling level because the modeling elements for compositional structure and behavior are strongly linked. For example, certain events and operations always affect certain locations and are always observed or performed by certain agents, etc. It is not possible to define a pure behavioral model from this point of view.

When executing a transition from a higher level model to a lower level model, certain higher level elements such as events, agents, locations, etc. must be mapped to respective elements of the lower level. Such transitions mean changes in behavior and/or compositional structure from level to level, but the terms and concepts at each modeling level are always the same and the number of levels is not limited by principle. A detailed discussion of basic mappings of model elements can be found in [14], pp. 81.

3. An Alternative View of Distributed and Concurrent Systems

3.1 Functional Distribution versus Spatial Distribution

The modeling approach presented above strongly differentiates between active components (agents) and passive components (locations) being affected by activities inside the system. Only then can two interpretations of the term "distribution" be clearly and appropriately separated. The first interpretation is related to the division of system functionality:

Functional distribution is given if a system consists of multiple agents.

Functional distribution means a separation of responsibilities in which different tasks in the system are assigned to certain agents. Each task can be of arbitrary complexity because an agent may access multiple locations and thus can perform numerous operation sequences.

For a detailed understanding of the system behavior locations are as important as agents because these are the places where activities can be observed. This leads to the second interpretation of "distribution":

Spatial distribution is given if a system contains multiple locations.[2]

3.2 Distribution as a Nonphysical Model Feature

Neither functional distribution nor spatial distribution refer to the physical structure of the system. Therefore, a system can be distributed even on higher levels of abstraction, but this need not be seen as a hint concerning the physical implementation. Functional distribution is a division of functionality which can be totally unrelated to physical agents. Even spatial distribution, as defined above, is not related to the physical system structure because the term "location" is defined without reference to (real) physical locations.

Distribution is to be considered primarily as a nonphysical system characteristic.

However, this does not exclude the fact that agents and locations of a lower level model can easily be mapped to physical system parts. Therefore, the modeling approach as presented here can also be used for the description of hardware systems.

It was discussed in section 2.3 that a complex system requires multiple models at different levels of abstraction. Because the compositional structure depends on the level of abstraction, functional distribution and spatial distribution are dependent on the level of abstraction as well:

Distribution is not a system feature but a model feature.

3.3 Distribution as a Feature of Model Mapping

If we look at the major topics in the context of distributed systems, many terms are given which can be best explained as model mappings where agents or locations of a higher level model are implemented as multiple agents or locations of a lower level model.

One example is replication, in which a single system component is composed of multiple components of the same type to achieve fault tolerance [19]. This can be seen as a special type of model mapping where an agent of a higher level model is mapped to several agents of a lower level model.

1. this does not require that the system is developed in a "top-down" style.

2. Interfaces between the system and its environment are not to be taken into account.

Caching is another concept which can also be explained as a model mapping, since a state storage location of a higher level model is implemented as a set of storage locations at the lower level.

These examples illustrate that important topics in the context of distributed systems refer to model mappings in which high level agents or locations are distributed during the transition to a lower level model. This leads to another interpretation of the term "distribution":

Distribution can be seen as a way to map a higher level model to a lower level model.

The need for *snapshot algorithms* [16] can also be derived from a model mapping. The primary purpose of a snapshot algorithm is the observation of the global state of a spatially distributed system. From an abstract point of view this can be modeled as a system observer which has direct read access to all state storages (S_1,S_2,S_3) of the observed agents (see Figure 6). When a snapshot command is sent to the system observer, he constructs a snapshot from the local states. The approach presented in this paper allows this to be modeled as one simple operation with the snapshot as the operation's result.

In practice, the compositional structure shown in Figure 6 must often be implemented using a communication subsystem which requires agents to communicate indirectly via message passing. At this level, the system observer cannot have direct access to another agent's state storages, thus a snapshot can no longer be generated by a simple operation. This problem can be solved by using a snapshot algorithm, i.e. these algorithms are implementations of snapshot operations.

In addition to snapshot algorithms, the need for *transactions* [17] can also be derived from a model mapping: if an agent accesses a location during an operation (i.e. reads, writes or modifies the corresponding value at the location), this single access can be implemented as a set of accesses on a lower level of abstraction. In order to be a correct, fault tolerant access implementation, this set of accesses must satisfy the ACID properties. Therefore, transactions can be interpreted as the outcome of mapping one operation or location to multiple operations or locations of a lower level model ([14], pp. 118). At a higher level, a transaction can be viewed as one simple access to a virtual location which yields a model with reduced complexity.

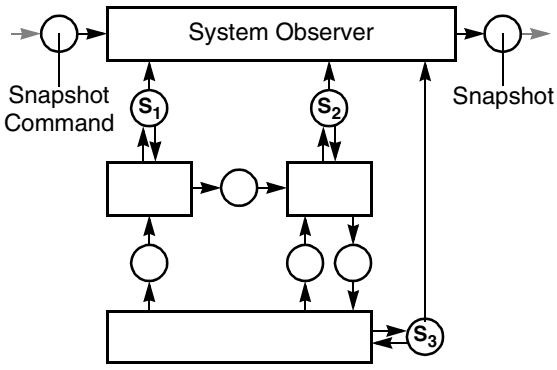

Figure 6: An Abstract View of Snapshot Algorithms

3.4 The Relationship between Concurrency and Distribution

Because an agent can have access to multiple locations (private storages) it is possible to have a spatially distributed system without functional distribution. If, on the other hand, functional distribution is present, there will (usually) be several locations within the system (interfaces between agents and state storages of agents), i.e. the system is spatially distributed. Furthermore, spatial distribution is a prerequisite of concurrency. In summary, we obtain the following interdependence:

Concurrency and functional distribution are independent model features. Each implies spatial distribution but not vice versa.

3.5 Concurrency as a Nonphysical Model Feature

It has been stated that spatial distribution is not a system feature but a model feature (see section 3.2). Because spatial distribution is a prerequisite of concurrency, the following statement applies:

Concurrency is not a system feature but a model feature.

Bearing this in mind, the interleaving model of concurrency can be viewed as an unfortunate mixture of two levels of abstraction. On the one hand, interleaving models are presented as descriptions of concurrent behavior. On the other hand, there is the assumption that concurrent events do not occur simultaneously which clearly contradicts the idea of causal independence. This assumption is simply a reference to a lower level model where concurrency is no longer existent due to certain design decisions.

4. Conclusion

The modeling approach presented above provides a solution to the conceptual problems and inconsistencies described in the introduction (cp. section 1.). In addition to the separation of modeling levels, the close relationship between compositional structure and behavior is particularly important. In this context, virtual locations represent a key concept.

Compositional structure and behavior of a system should be viewed as equally important and complementary modeling aspects on each level of abstraction.

The insight that concurrency and distribution are non-physical model features allows a differentiation of application-related and implementation-related concurrency and distribution. However, another important consequence is that the classification of a system (as distributed or concurrent) depends on the chosen level of abstraction.

A strict separation of concurrent/distributed and non-concurrent/distributed systems is impossible as a matter of principle.

Moreover, the strong interdependence of compositional structure and behavior allows a clear differentiation between nondeterminism and "true" concurrency:

Concurrency can only be assumed in conjunction with spatial distribution and should not be viewed as a special kind of nondeterminism.

Our approach does not only simplify the modeling of concurrent and distributed systems. It also has the advantage of more "natural" models, because every system can be seen as a collection of cooperating agents. This has already proven helpful in the documentation of complex system architectures [15][20].

The ideas presented in this paper could also be the starting point for the development of advanced programming concepts which would not only allow a differentiation of modeling levels but would also contain elements for describing a system's compositional structure [14].

5. References

[1] Carl Adam Petri, *Kommunikation mit Automaten,* PhD Thesis, Technische Hochschule Darmstadt 1962

[2] Wolfgang Reisig, *Petrinetze,* 2nd Edition, Springer Verlag, Heidelberg 1986

[3] Leslie Lamport, *Time, Clocks, and the Ordering of Events in a Distributed System,* Communications of the ACM, Vol. 21, No. 7, July 1978

[4] Robin Milner, *A Calculus of Communicating Systems,* Lecture Notes in Computer Science, Springer Verlag, Heidelberg 1980

[5] Matthew Hennesy, *Algebraic Theory of Processes,* MIT Press series in the foundations of computing, Boston 1988

[6] Robert M. Keller, *Formal Verification of Parallel Programs,* Communications of the ACM, Vol. 19, No. 7, July 1976

[7] Zohar Manna u. Amir Pnueli, *The Temporal Logic of Reactive and Concurrent Systems,* Springer Verlag, New York 1992

[8] Andrew S. Tanenbaum, *Distributed Operating Systems,* Prentice Hall, New Jersey 1995

[9] George Coulouris, Jean Dollimore u. Tim Kindberg, *Distributed Systems - Concepts and Design,* 2nd Edition, Addison Wesley, 1994

[10] *DUDEN Informatik*, 2nd Edition, Dudenverlag, Mannheim 1993

[11] Manfred Broy, *Informatik - Eine grundlegende Einführung,* Vol. 2; 2nd Edition; Springer Verlag, Heidelberg 1998

[12] Siegfried Wendt, *Nichtphysikalische Grundlagen der Informationstechnik,* Springer Verlag, Heidelberg 1989

[13] Gerard Tel, *Introduction to Distributed Algorithms,* Cambridge University Press, 1994

[14] Peter Tabeling, *Der Modellhierarchieansatz zur Beschreibung nebenläufiger, verteilter und transaktionsverarbeitender Systeme,* Shaker Verlag, Aachen 2000 (PhD Thesis, University of Kaiserslautern).

[15] Frank Keller et. al. *Improving Knowledge Transfer at the Architectural Level: Concepts and Notations,* International Conference on Software Engineering Research and Practice, Las Vegas, June 2002

[16] K. Mani Chandy, Leslie Lamport, *Distributed Snapshots: Determining Global States of Distributed Systems,* ACM Transactions on Computer Systems, Vol. 3, No. 1, February 1985, pp. 63-75

[17] Theo Härder, Andreas Reuter, *Principles of Transaction Oriented Database Recovery - A Taxonomy,* University of Kaiserslautern, 1982

[18] Andreas Bungert, *Beschreibung programmierter Systeme mittels Hierarchien intuitiv verständlicher Modelle,* Shaker Verlag, Aachen 1998 (PhD Thesis, University of Kaiserslautern)

[19] Flaviu Christian, *Understanding Fault-Tolerant Distributed Systems,* Communications of the ACM, Vol. 34, Nr. 2, February 1991

[20] Bernhard Gröne et. al. *Design recovery of Apache 1.3 — A case study,* International Conference on Software Engineering Research and Practice, Las Vegas, June 2002