1991

A UNIFIED MODEL OF SOMWARE AND DATA DECOMPOSITION

Yair Wand
The University of British Columbia

Ron Weber
The University of Queensland

A UNIFIED MODEL OF SOFTWARE AND DATA DECOMPOSITION

Yair Wand
Faculty of Commerce and Business Administration
The University of British Columbia

and

Ron Weber
Department of Commerce
The University of Queensland

ABSTRACT

Decomposition is an important part of information systems analysis and design and is manifested as the breakdown of the system to elements such as subsystems, modules, activities, processes, entities, and objects. Good decomposition is considered a major requirement for a good system design. However, there is no comprehensive theory of information systems decomposition and no single dominant decomposition approach exists. Consequently, software decomposition relies on "guidelines" and designer's experience.

In this article, we propose a foundation for a theory of good decomposition based on two principles: 1) the decomposition of an information system should reflect the nature of the real world system represented by it, and 2) static and dynamic aspects of systems cannot be separated and hence good decomposition should be based on both.

The model enables the analysis of concepts such as good software modules, normalized relations, objects, and entities as special cases of one generalized construct.

1. INTRODUCTION

Decomposition is a major activity of systems analysis and design and is manifested as the breakdown of the system description to elements such as subsystems, modules, activities, processes, entities, and objects. Good decomposition is considered a major requirement, and frequently the essence of a good design.[1] Decomposition materializes in design practices such as modularization of systems and software (Parnas 1972, Myers 1978, Yourdon and Constantine 1979, DeMarco 1979; Gane and Sarson 1979), data normalization (Date 1981), and, more recently, object oriented design and implementation (Korson and McGregor 1990; Coad and Yourdon 1990).

However, there is no comprehensive theory of information systems decomposition and no single dominant decomposition approach exists (see, e.g., Pressman 1987). Consequently, software decomposition relies on "guidelines" (Gane and Sarson 1979, p. 189; Myers 1978, p. 151) and designer's experience. Moreover, data and software decompositions are frequently done independently. Thus, data decomposition does not reflect the dynamics of the system that is manifested in the software decomposition. The object-oriented approach tries to remedy this situation by encapsulating state information (data) with behavior. Nevertheless, no theory exists on how to identify "good" objects and one depends on "practice and experience" (Coad and Yourdon 1990, p. 60).

In previous articles we proposed a rudimentary formal model of system decomposition and presented a necessary condition for a decomposition to be "good" (Wand and Weber 1989b, 1990a). Briefly, the condition implies that each component (subsystem) in a good decomposition should have a well-defined behavior in the sense that its state information is sufficient to determine its state changes for every input. Our model was incomplete in several respects. First, the condition provided was only necessary, so while it could be used to rule out possible decompositions, it did not prescribe how to generate *good* decompositions. Second, the implications of the formal model to systems analysis and design were not elaborated. Finally, the model was not used for describing and explaining existing decomposition practices and methods.

Here we expand our model in two directions. First, we anchor the notion of good decomposition to a set of postulates on how to model reality. Second, we show that a common foundation for software decomposition, data decomposition, and objects-oriented concepts can be derived from this view. Our model is based on two fundamental premises: 1) decomposition should reflect the nature of the real world system represented by the information system, and 2) static and dynamic aspects of systems cannot be separated and hence good decomposition should be based on both.

In the following, Section 2 presents the principles of our approach, Section 3 describes the formal model of reality we use, Section 3 presents the notion of a *proper thing*, which is the key concept in our model of good decomposition, Section 5 defines good decomposition, Section 6 discusses software, data, and object related concepts. Finally, Section 7 summarizes the model briefly and describes current research directions.

2. PRINCIPLES

Decomposition is intuitively understood as the breakdown of a whole into small parts. However, in reality, a decomposed structure is attained by *composing* the whole from smaller units, in a hierarchical way. Indeed, it is claimed that this is the way complex natural objects evolve (Simon 1981). Decomposition is applied to complex objects for two purposes. First, it can be used for the *analysis* of existing natural or human-created objects.[2] Second, it can be employed in the *design* of new human-created objects (artifacts). We therefore begin by defining what we mean by analysis and design:

Definition 1: *Analysis* is the understanding of a system's behavior. *Design* is the creation of a plan for a system that must have a certain *specified* behavior.

Due to the dual role of decomposition, a theory of decomposition should be able to be employed in two ways. First, it should be able to *explain* why a certain breakdown is considered a "proper" or *good* decomposition. Second, it should be able to *prescribe* how a complex object can be constructed from simpler objects.

We begin our discussion of the formal theory of good decomposition by posing three requirements for such a theory:

1. It should show how a system can be analyzed or designed by concentrating on parts of the system.
2. It should afford a view of a system as a whole, and also as made of components that are independent in some way.
3. It should prescribe when the process of breakdown is to stop, or, alternatively, when an object can be viewed as a fundamental, non-decomposable, unit.

We propose to base a theory of decomposition on two fundamental principles.

Principle 1: A decomposed thing is viewed sometimes as a whole, and sometimes as a composite where the focus shifts to the parts.

Principle 2: A good decomposition is behavior-related.

The first principle implies that for an object to be "interesting" there should be some information that cannot be obtained by considering it as an aggregate of independent smaller objects. However, "quite a bit" of the information about the object can be known in this way. Hence, a good decomposition theory should provide for alternating the focus between a "holistic" point of view and a component point of view.

The second principle establishes the importance of behavior in the model. Behavior will be defined as the way a thing responds to certain possible stimuli applied to it. The definition of these stimuli amounts to defining the purpose of analyzing or designing a certain object and is critical to the nature of the decomposition obtained.

Finally, we note that rather than considering an object as made of smaller components, it can be considered as part of a larger "whole" and hence its behavior might be viewed as reflecting part of the behavior of the larger object.

We summarize the above as three observations about the nature of good decomposition:

1. It is related to the ability to describe or achieve the behavior of a complex object via the behavior of its components.
2. It is an approximation in the sense that not all the behavior of the complex object can be represented by the behavior of its components.
3. The same principles that guide the "correct" breakdown of an object are those that apply when determining if an object can be analyzed or constructed independently from other objects.

3. FORMAL FOUNDATION

We base our theory of decomposition on the notion that an information system is a *representation* of a real system[3] (Wand and Weber 1988; 1990b; Jackson 1983). Hence, we propose:

Principle 3: The decomposition of an information system should reflect a decomposed view of the represented system.

Based on this principle, we turn to the question of how to describe a decomposition of a real system. To model reality we use an ontological model proposed by Bunge (1977; 1979).[4]

Ontological Postulate 1*: The world is made of *things* that possess *properties*.

Ontological Postulate 2*: A thing may be *simple* or *composite*, that is, composed of other things. Any two things can be combined to a thing.

Definition 2*: A thing is a *composite* thing if there exist (at least) two things from which it is combined. Otherwise, it is a ***simple*** thing.

Some of the properties of a composite thing belong to the composing things, but others belong only to the composite thing as a whole, hence we define:

Definition 3*: A property of a composite thing will be termed *hereditary* or *resultant* if it is a property of some composing things. Otherwise, it will be termed an *emergent* property.

Furthermore, every composite thing must have properties that do not belong to its components:

Ontological Postulate 3*: Every composite thing must possess emergent properties.

Only some of the properties of a thing are of interest in a given context. This is modelled via the concept of a *functional schema*:

Definition 4*: A thing is modelled in terms of a *functional schema* $F=<F_1....F_n>$ where each function, F_i assigns a value to an observed property, at time t.

Based on the concept of a functional schema, the notions of *state* and *state space* are defined:

Definition 5*: The functions F_i are called the *state variables* of the thing, the values of the functions $F=<F_1....F_n>$ at a given time comprise the *state* of the thing. The set $S(X)=\{<x_1,...x_n> \mid x_i=F_i(t)\}$ is termed the *possible state space* of the thing X.

The dynamics of a thing are described via the notion of *events*:

Definition 6*: An *event* is an ordered pair of states $e=<s,s'>$ $s,s' \in S(X)$.

Not all possible combinations of values of the state variables (i.e., states) can occur:

Definition 7: Any constraint on the allowed values of state variables or their combinations will be termed a *state law*. The set of states conforming to the state laws will be termed the *lawful state space* of the thing X and denoted by $S_L(X)$.

Not all possible pairs of states designate events that can happen:

Definition 8*: The set of lawful events is the set $E_L \subseteq S_L \otimes S_L$ which conforms to the laws of the thing.

We now define the notions of *behavior* and *interaction*:

Definition 9*: The *behavior* of a thing in a given time interval is the ordered set of states it traverses in this interval.

Definition 10*: Thing X acts-on thing Y iff the "trajectory" of states Y traverses in time when X is present, is different than what it would have been without X. Things X and Y *interact* iff X acts-on Y or Y acts-on X.

Finally, we define a *system* as a special case of a composite thing:

Definition 11: A *system* is a composite thing that cannot be broken down to non-interacting components.

Using the above postulates and definitions, we now discuss what a good decomposition is.

4. ON PROPER THINGS

Our model of good decomposition is based on the notion of a *well-defined thing*. We begin by making a distinction between two types of change of states that a thing may undergo:

Definition 12: A change of state due to a stimulus from the environment will be termed an *external event*. Any other change of state will be termed an *internal event*.

The following premise reflects the belief that things do not behave arbitrarily, but according to some *law of behavior*:

Working Premise 1 (regularity):[5] Without external stimuli, a thing will change its state if, and only if, there exists a lawful transition to another state.

The regularity premise leads to the following categorization of states:

Definition 13: A state will be termed *unstable* iff there exists a transition to another state. Otherwise, the state is *stable*.

We further assume the following:

Working Premise 2 (stability): A thing in an unstable state will change its state to a stable state.[6]

The following lemma ties the notion of stability to the two types of event:

Lemma 1: An event e = <s,s'> where s is a stable state is an *external event*. An event e = <s,s'> where s is an unstable state is an *internal event*.

Behavior was defined in Section 3 as the *evolution of the states of a thing in time*. According to our premises, once a thing reaches a stable state it will stay in that state unless receiving a stimulus from the environment. Hence, the actual behavior of a thing depends on the external events it might undergo. We therefore define:

Definition 14: The *relevant behavior* of a thing is its possible state transitions in response to a given set of external events that might occur.

Hence-on we will assume that all analysis of the behavior of a thing is done *with respect to a given set of external events* which will be called the *relevant event set*.

A thing undergoing an external event might reach an unstable state from which there is more than one possible internal transition. In such cases, the behavior of the thing will appear to be unpredictable. It is desirable that a thing will be *well-behaved* in the following sense:

Definition 15: A thing whose internal transitions occur as a result of a given set of external events that are well defined[7] will be termed *well-behaved*.

The notion of a well-behaved thing is fundamental to our model because we are interested in the *behavior* of things. This is established in the following lemma:

Lemma 2: A thing is well-behaved with respect to a given set of external events if, and only if, the evolution of the state of the thing for these events can be fully determined.

While all things must be subject to some external events in order to change, it is not necessary that a thing will possess internal transitions for states that can be reached as a result of the relevant event set. We therefore make a distinction:

Definition 16: A thing that undergoes internal transitions as a result of a given set of external events will be said to have *internal dynamics* and will be called a *dynamic thing*; otherwise, it will be termed a *static thing*.[8]

We now analyze the above concepts in terms of the state variables of the thing. When an external event occurs, at least one state variable of the thing is modified as a result of a change in the environment of the thing. It follows that for external events to occur, some state variables must jointly belong to the state definitions of the thing and the environment. On the other hand, a thing might have state variables that are only affected by internal transitions. Hence, we define:

Definition 17: A state variable of a thing will be called an *input state variable* if it can be modified only by external events, it will be called a *derived state variable* if it is modified only in internal events, it will be called a *shared state variable* if it can be modified by both types of events.

Note, according to this definition input and shared state variables "belong" to the thing and to its environment (or to a thing in the environment). Furthermore, a dynamic thing must possess state variables that are either derived or shared. We make two further distinctions:

Definition 18: A dynamic thing that has at least one derived state variable will be termed a *self-controlled* thing. A dynamic thing that does not have any shared state variable will be termed a *completely self-controlled* thing.

Lemma 3: In a completely self-controlled thing, no external event can be reversed directly by an internal event.

Consider now the following problem. Given a "universe" described in terms of state variables, can a given subset

of the state variables be viewed as representing a thing that is well "separated" from the universe?

To answer this, a given set of *relevant* external events that might affect these state variables has to be given. Then the following criteria can be applied:

Things should be well-behaved:

1. For each state there is either no transition, or a well-defined transition which is not an external event.

2. Every transition which is not an external event ends on a stable state.

For the thing to be dynamic:

3. There must exist at least one internal event.

For dynamic things to be "proper":

4. To have at least some of its "own" behavior, the candidate thing must be self-controlled.

5. To view the environment as behaving independently of the thing, no shared variable should exist.

This is summarized in the following definition:

Definition 19: A *proper* thing is either a static or a dynamic, well-behaved, and completely self-controlled thing.

Intuitively, a proper dynamic thing is a thing whose behavior is predictable and depends only on its own internal transition laws. Furthermore, the thing cannot directly "reverse" the effects of the environment. Such a thing can be viewed as "well-carved-out" of its environment.

Finally, we note that in light of the discussion of dynamics, a state variable that never changes carries no useful information because it does not affect the result of any internal transition. Hence we define:

Definition 20: A state variable will be said to be *redundant* iff it never changes under any of the given external events or resulting internal events. A state representation will be termed *minimal* if it contains no redundant state variable.

Note that the above definitions are all with respect to a given set of external events.

5. WHAT IS A GOOD DECOMPOSITION?

We now apply the concept of a proper thing to the notion of good decomposition. We begin by considering the possibility that the state space of a composite thing can be reconstructed from the state spaces of the composing things:

Definition 21: A thing, X, will be said to be *state-divisible* iff there exists a set of things $\{X_k, k=1,...,m\}$ such that there exists a representation of the lawful state space of X, $S_L(X) = S_L(X_1) \otimes ... \otimes S_L(X_m)$. Otherwise it will be termed *indivisible*. For a divisible thing, the set $\{X_k, k=1,...,m\}$ will be termed a *state-preserving decomposition*.

For a state divisible thing, X, every combination of lawful states of each of the components X_k matches a lawful state of X and vice versa.

We turn now to dynamics. An external event with respect to the composite thing is reflected as a change of input state variables of the thing. If one of these state variables is also a state variable of a component, then this component also undergoes an external event. Another possibility is that as a component undergoes an internal event, a change in a derived state variable of the component will be recognized as an external event in another component. These possibilities are reflected in the following definition:

Definition 22: An event that occurs in a component as a direct result of an external event will be termed a *direct external event*. An event that occurs as a result of an internal transition in another component will be termed a *derived external event*.

A *well-decomposed* thing can now be defined:

Definition 23: A composite thing will be said to be *well-decomposed* under a given set of external events iff there exists a state-preserving decomposition where each of the components is a proper thing and has a minimal set of state variables under this set of events.

6. SOME RESULTS

In this section we use the decomposition model to analyze software and data decomposition. Our analysis is based on the following premise:

Working Premise 3 (representation): The components of the information system should reflect components that were identified in the process of analysis.

Consider a well-decomposed system according to our definitions in sections 4 and 5. For a given set of externally-imposed changes in the system, the following holds:

1. The complete state information of the system can be re-constructed from the states of the components.

2. Each of the components is well-behaved under each external event occurring in the component.

3. Components can be static or dynamic.

4. Dynamic components should be completely self-controlled.

We now apply these concepts to software and data.

6.1 General Concepts

A software module: A software module is a dynamic information system component that represents a dynamic component of the real system. It contains state information (as data variables) and dynamic information on how the state information might change as a result of possible external events (as software instructions).

A data aggregate: A data aggregate is a static component of the information system that represents a static component of the real system. It has no internal transitions and can change only as a result of external events that are manifested as updating transactions.

6.2 Software Decomposition

Module independence and module coupling: Various types of module coupling are defined based on various situations occurring in the software in terms of data and control structures (Myers 1978, Chapter 5).[9] Thus, evaluation of coupling among modules requires that the software be at least designed in detail. Moreover, the results might depend on the specific language constructs. In our model, a module is coupled to another if and only if information from the other module is needed to determine the outcome of an internal transition. That is, certain derived state variables cannot be fully determined by the state of the module. Whether modules are coupled or not depends on the set of external events. Accordingly, *loose coupling* can be viewed as no-coupling in the above sense under a certain *subset* of external events (selected based on some notion of importance or relative frequency). Note that the above interpretation of module coupling does not depend on any implementation details.

Strong cohesion: In our model, a component can be broken down if it is state-divisible to well-behaved subcomponents. Strong cohesion means that the given component cannot be broken in this way. In software terms this would mean that, for every possible bi-partitioning, one of the following occurs: 1) there are constraints connecting data values in the two submodules, or 2) data from both submodules are needed to determine the processing in each module.

Decomposition techniques: Various practical decomposition rules have been proposed. Here we discuss *transaction decomposition* and *source-transform-sink decomposition* (STS) (Yourdon and Constantine 1979; Myers 1978). Transaction decomposition is the breakdown of software to components that respond independently to different input transactions. STS decomposition is the breakdown to modules that deal with the original inputs, modules that deal with data transformations after the input data have "disappeared," and modules that produce outputs. These methods, in particular STS, require quite a bit of judgment by the analyst.

In terms of our model, transaction decomposition is a breakdown to components that are well behaved under the *direct effect* of the external events. STS is a breakdown to components when *derived events* are considered. Accordingly, an input module is directly subject to external events of the system and a processing module is subject only to derived events. This view predicts some inherent difficulties with STS. In particular, the role of a module might not be completely defined if it is subject to both direct and derived events.

6.3 On Data Decomposition

Normalization is introduced as a solution to problems such as update and deletion anomalies and is formalized in terms of dependencies that exist among data elements. Here, we take the approach of normalization as a special case of decomposition to static system components. Rather than using formal arguments, we analyze a simple example.

Consider a simple order processing system where three components have been identified: *customer, product,* and *order*:[10]

customer (Customer Name, Customer Address, Customer Type)
product (Product Name, Unit Price)
order (Order Date, Order Quantity, Total Price)

Where Customer type can be 'P' (preferred) or 'R' (regular) and Total Price is calculated according to the formula:

Total Price = Quantity * Unit Price *(1-Discount)

Where only preferred customers get a 10% discount.

In our model, a good decomposition is defined with respect to a *given set of external events*. Assume the following set:

1. Modify Customer data (change customer name, or address, or type).

2. Modify Product data (change product name or unit price).

3. Modify Order data (quantity).

Both Customer and Product modifications are external events in the respective components and are the only changes that can happen to these components. Hence, these components are static components. Consider now the Order component. As Unit Price, Customer Type, and Order Quantity change, Total Price changes as well. However, the order component does not contain the complete information about how price should be calculated and is, therefore, not a well-behaved component. Moreover, the combined state of the components might be an invalid system state (as total price in the order component is not guaranteed to conform to the formula). To remedy this, the system decomposition can be modified to include an additional *dynamic* component:

Price (Customer Type, Unit Price, Quantity,
Total Price; Price Law)

where the notation *;Price Law* indicates that the component's state always conforms to the price formula. This is a dynamic component that may be activated when customer type, unit price or order quantity are changed. As total price changes, the Order component can be modified by an external event. The system is now composed of four components, three are static and one is dynamic and well-behaved under the given set of external events. Moreover, every state combination of these modules is now valid. Note that the new (dynamic) Price component is affected by external events to the system, while the (static) order component is affected either by an external event (i.e., change of Quantity) or by a derived event (changing Total Price).

The three static components can be implemented as relations. The dynamic component can be implemented either as a routine, or as a table listing all possible combinations of Unit Price, Quantity, Customer Type and Total Price.[11] It turns out that relations may have two distinct roles:

a. Represent the state of static components.

b Act as a representation of a law, or *functional relationship* storing only "allowed" combinations of state variables.

6.4 On Objects in the Object-Oriented Approach

The object-oriented approach has recently attained wide acceptance. However, there is still no common agreement on the fundamentals of the approach (Nierstrasz 1986; Banerjee et al. 1987; Tsichritzis and Nierstrasz 1989). As well, there are no formal rules on what constitutes an appropriate object. In the following we briefly discuss the relationship of the decomposition model to the object paradigm.

The most fundamental concept of the object-oriented approach is the notion of *encapsulation* (also termed *data abstraction*) which is the "packaging" of data and operations.[12] Related to encapsulation is the notion of *independence* which means that only the operations defined in the object (usually termed "methods") can modify its state information.

The notion of a dynamic, well-behaved, completely self-controlled thing, defined in Section 4, encompasses encapsulation and independence in the object-oriented approach. The state of a well-behaved thing can only be changed by a given external event or via well-defined internal transitions. A completely self-controlled thing must have some state variables that are uniquely determined only by internal transitions. This can be interpreted as encapsulation and independence.

According to our model, for a thing to change state it must be subject to external events, modelled as changes to input state variables. For a well-behaved thing, the response to an external event depends only on the dynamics of the thing. Hence, external events provide a model of communication among objects without compromising object independence.

Consider now a well-behaved static thing. Such a thing has no internal dynamics and matches the notion of an entity in the entity-relationship model. It follows that an entity can be viewed as a "limiting case" of an object where no internal dynamics exists. Practically, this is

attained by viewing all changes to state variables of the entity as external events.

7. SUMMARY AND RESEARCH DIRECTIONS

We presented a model for good decomposition as a view of a composite thing in terms of well-behaved components. The key premises were that a good decomposition of an information system is related to the dynamics of the things in the represented real system, and that dynamics have to be examined with respect to a given set of stimuli applied to the system. These stimuli, modelled as external events, reflect the conditions under which we are interested in the system; that is, they define the application.

The model enables the analysis of concepts such as good software modules, normalized relations, objects, and entities as special cases of one construct-proper thing.

Presently, we are working in several directions. First, we seek to improve the formalization of the concepts. So far we have formalized a necessary condition for good decomposition. Second, the definition of necessary and sufficient conditions for a good decomposition can be used as the basis for computer-supported decomposition. The model was already used in the development of a prototype set of automated tools for decomposition of systems based on specifications (Paulson 1989; Paulson and Wand 1990). Third, existing analysis and design methods can be examined in light of the model as to their support for obtaining good decompositions. This was already tried for the entity relationship model and data flow diagrams (Wand and Weber 1989a; 1990b). Fourth, the model can provide specific guidelines for systems analysis and design practices, in particular, it shows the importance of identifying the relevant external events and their consequences (i.e., derived events). Finally, we plan to study the relation between our notion of good decomposition and the effort required to maintain a system. This is important, as one of the main justifications for decomposition is the reduction of maintenance effort (Parnas 1972).

8. ACKNOWLEDGEMENTS

This research was supported in part by an operating grant from the Natural Sciences and Engineering Research Council of Canada and by a grant from GWA Ltd.

9. REFERENCES

Banerjee, J.; Chou, H.-T.; Garza, J.; Kim, W.; Woelk, D.; Ballou, N.; and Kim, H.-J. "Data Model Issues for Object-oriented Applications." *ACM Transactions on Office Information Systems*, Volume 5, Number 1, January 1987, pp. 3-26.

Bunge, M. *Treatise on Basic Philosophy (Volume 3): Ontology I, The Furniture of the World.* Boston: D. Reidel Publishing Company, 1977.

Bunge, M. *Treatise on Basic Philosophy (Volume 3): Ontology II, A World of Systems.* Boston: D. Reidel Publishing Company, 1979.

Coad, P., and Yourdon, E. *Object-oriented Analysis.* Englewood Cliffs, New Jersey: Prentice-Hall (Yourdon Press), 1990.

Courtois, P. J. "On Time and Space Decomposition of Complex Structures." *Communications of the ACM*, Volume 2, Number 6, June, 1985, pp. 590-603.

Date, C. J. *An Introduction to Database Systems*, Third Edition. Reading, Massachusetts: Addison-Wesley, 1981.

DeMarco, T. *Structured Analysis and System Specification.* Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1979. Copyright 1978 YOURDON, INC.

Gane, C. J., and Sarson, T. *Structured Systems Analysis: Tools and Techniques.* Englewood Cliffs, New Jersey: Prentice Hall, Inc., 1979.

Jackson, M. A. *System Development.* London: Prentice-Hall International, 1983.

Korson, T., and McGregor, J. "Understanding Object-oriented: A Unifying Paradigm." *Communications of the ACM*, Volume 33, Number 9, September 1990, pp. 40-60.

Myers, G. J. *Composite/Structured Design.* New York: Von Nostrand Reinhold Company, 1978.

Nierstrasz, O. M. "What is an 'Object' in Object Oriented Programming?" In D. Tsichritzis (Ed.), *Objects and Things*, Universite De Geneve, 1987, pp. 1-13. Reprinted from *The Proceedings of the CERN School of Computing*, Renesse, The Netherlands, August 31-September 13, 1986.

Parnas, D. L. "On the Criteria to be Used in Decomposing Systems Into Modules." *Communications of the ACM*, Volume 15, Number 12, December, 1972, pp. 1053-1058.

Paulson, D. "Reasoning Tools to Support Systems Analysis and Design." Unpublished Ph.D. Dissertation, University of British Columbia, 1989.

Paulson D., and Wand, Y. "An Automated Approach to Information Systems Decomposition." Working Paper 89-MIS-002, Faculty of Commerce and Business Administration, The University of British Columbia, May 1990 (revised).

Pressman, R. S. *Software Engineering: A Practitioner's Approach*, Second Edition. New York: McGraw Hill, 1987.

Robson, D. "Object-oriented Software Systems." *Byte*, August 1981, pp. 74-86.

Simon, H. A. *The Sciences of the Artificial*, Second Edition. Cambridge, Massachusetts: MIT Press, 1981.

Tsichritzis, D. C., and Nierstrasz, O. M. "Directions in Object-Oriented Research." In W. Kim and F. H. Lochovsky (Eds.), *Object-Oriented Concepts, Databases, and Applications*, Reading, Massachusetts: ACM Press, Addison-Wesley, 1989, Chapter 20, pp. 523-536.

Wand, Y., and Weber, R. "An Ontological Analysis of Some Fundamental Information Systems Concepts." In J. I. DeGross and M. H. Olson (Eds.), *Proceedings of the Ninth International Conference on Information Systems*, Minneapolis, Minnesota, November 1988, pp. 213-225.

Wand, Y., and Weber, R. "An Ontological Evaluation of Systems Analysis and Design Methods." *Proceedings of the IFIP WG 8.1 Working Conference on Information Systems Concepts: An In-Depth Analysis, Analysis of Some Fundamental Information Systems Concepts*, Namur, Belgium, October 1989a.

Wand, Y., and Weber, R. "A Model of Systems Decomposition." In J. I. DeGross, J. C. Henderson, and B. R. Konsynski (Eds.), *Proceedings of the Tenth International Conference on Information Systems*, Boston, Massachusetts, December 1989b, pp. 41-51.

Wand, Y., and Weber, R. "An Ontological Model of an Information System." *IEEE Transactions of Software Engineering*, Volume 16, Number 11, November, 1990a, pp. 1282-1292.

Wand, Y., and Weber, R. "Toward a Theory of the Deep Structure of Information Systems." In J. I. DeGross, M. Alavi, and H. Oppelland (Eds.), *Proceedings of the Eleventh International Conference on Information Systems*, Copenhagen, Denmark, December 1990b, pp. 61-71.

Wegner, P. "Dimensions of Object-Based Language Design." *Proceedings of the OOPSLA Conference*, Orlando, Florida, 1987, pp. 168-182.

Yourdon, E., and Constantine, L. L. *Structured Design: Fundamentals of Computer Program and Systems Design*. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1979.

10. ENDNOTES

1. Myers (1978) introduces the concepts of system partitioning, module hierarchy and module independence under the chapter title "Underpinnings of a Good Design."

2. Courtois (1985) claims: "Decomposition has long been recognized as a powerful tool for the analysis of large and complex systems."

3. The term "real system" should be understood as a "system as perceived by someone." In this respect, the term encompasses conceptual systems as well.

4. In the following we designate postulates and definitions adapted from Bunge's work with an asterisk.

5. In the following, we term modelling assumptions that go beyond the general ontological postulates *working premises*. These premises are assumed valid for the types of things and systems that are the subject of the domain of information systems.

6. This premise can be viewed as an adaption of the principle of homeostasis in systems theory. It is a simplification, as the thing may traverse several unstable states before reaching a stable state.

7. That is, the state assumed after the internal transition is uniquely determined by the state prior to the transition.

8. Note that these concepts are defined with respect to a given set of external events.

9. Myers (1978) uses concepts such as data coupling, stamp coupling, control coupling, external coupling, etc.

10. Here we refer to a decomposition of a model of the *real* system. Also, we consider only one generic instance of each component.

11. In our example, the latter possibility is unrealistic due to the number of possible combinations.

12. Nierstrasz (1986, p. 3) claims that "by far, the most important concept in the object-oriented approach is data abstraction." Wegner (1987, p. 168) states that an object has "a set of 'operations' and a 'state' that remembers the effect of operations." Robson (1981) defines an object as "A package of information and description of its manipulation."