1989

A MODEL OF SYSTEMS DECOMPOSITION

Yair Wand
University of British Columbia

Ron Weber
University of Queensland

A MODEL OF SYSTEMS DECOMPOSITION

Yair Wand
University of British Columbia

Ron Weber
University of Queensland

ABSTRACT

The way in which systems should be decomposed so they can be better understood and better designed remains a fundamental problem in the information systems discipline. A number of different decomposition methodologies have been proposed. However, no methodology has emerged as dominant, presumably because the relative strengths and limitations of each methodology are still unclear. Case study research that has compared the methodologies, for example, has produced only equivocal results.

In the absence of a theory of decomposition, it is difficult to make insightful predictions about the merits and failings of a particular methodology. Consequently, it is difficult to undertake empirical research that produces compelling results. Accordingly, in this paper we develop a rudimentary model of decomposition that we hope might form the basis of a subsequent, more complete theory of decomposition.

1. INTRODUCTION

Much of the computer science and information systems literature has been concerned with the problem of decomposition: the way in which an object should be broken up into smaller objects so it can be better understood or better designed. Researchers' preoccupation with the problem reflects their attitudes about its widespread importance for systems analysis, design, and implementation. For example, Bergland (1981, p. 14) argues: "I believe the quality of the program structure resulting from a design methodology is the single most important determinant of the life-cycle costs for the resulting software system."

In spite of the attention that the decomposition problem has received, no single approach to decomposition has emerged as dominant (see, e.g., Pressman 1987). The issue of which decomposition method to use is still sorely debated (see, e.g., Zave 1984), and researchers still seek new approaches to decomposition as a manifestation of their dissatisfaction with current approaches (see, e.g., Dromey 1988).

We believe that research on decomposition has been undermined because a rigorous theory of decomposition has yet to be developed. Many of the central concepts used in decomposition research -- for example, concepts like coupling, module, and hierarchy -- are poorly defined. Consequently, decomposition approaches are often fuzzy and difficult to evaluate or apply. Indeed, as Bergland (1981, p. 35) comments: "All of the methodologies rely on some magic." Moreover, in the absence of an underlying theory of decomposition, it is difficult, if not impossible, to make good predictions about the strengths and weaknesses of different decomposition methodologies. Thus, empirical research that seeks to evaluate competing decomposition methodologies is problematical because researchers are unable to identify the strategic hypotheses to test and the relevant data to gather to allow comparative analyses to be undertaken.

Accordingly, in this paper we seek to develop a rudimentary theory of decomposition. Our ultimate goal has a threefold purpose. First, we are striving to identify a parsimonious set of precise, core concepts that can be used as a common framework for describing and comparing decomposition methodologies. Second, we then wish to use the model to better understand and predict the relative strengths and weaknesses of competing decomposition methodologies. Third, in light of our theoretical predictions about the relative strengths and weaknesses of different decomposition methodologies, we seek eventually to conduct better-directed empirical research to evaluate the methodologies.

The remainder of this paper proceeds as follows. The first section below provides a brief review of some important prior research on decomposition theories and methodologies. In the second section we develop the basic concepts used in our own model of decomposition. The third section is the crux of the paper: it presents our formal model of decomposition. In the fourth section we illustrate how the decomposition formalism can be applied via an example. Finally, the fifth section discusses future research directions and presents our conclusions.

2. PRIOR RESEARCH ON DECOMPOSITION

Prior research on decomposition has been undertaken for two reasons. First, researchers have sought better ways of **understanding** complex systems. During the definition phase of the system development life cycle, for example, systems analysts must come to an understanding of the existing system as a basis for eliciting the requirements for the new system. Second, researchers have sought better ways of **designing** systems. During the program development phase of the system development life cycle, for example, programmers must design and implement programs that satisfy the requirements definition. In short, decomposition has a role in both analysis and design.

While a large number of decomposition methodologies have been proposed, six have been prominent. The first approach, **functional decomposition**, requires analysts/designers to identify the primary function performed by the system. This function is then broken up into a co-ordinated set of subfunctions using some type of divide-and-conquer technique (see, e.g., Parnas 1972; Wirth 1973; Dijkstra 1976; Linger, Mills and Witt 1979). Functional decomposition has proven useful across a large number of diverse problems. However, as Bergland (1981) points out, it often leads to decomposition inconsistencies. Unfortunately, it does not specify whether decomposition should be undertaken with respect to time, control flow, data flow, etc.

The second approach, **data flow decomposition**, requires analysts/designers to construct a data flow diagram to describe the existing or proposed system. The data flow diagram then forms the basis for two types of decomposition: either (a) transform analysis, if the number of input data flows to a process roughly equals the number of output data flows from the process, or (b) transaction data analysis, if the number of input data flows differs substantially from the number of output data flows (see, e.g., Myers 1975; Yourdon and Constantine 1979). Pressman (1987, p. 263) argues that data flow decomposition has proved useful when "information is processed sequentially and no formal hierarchical data structure exists." In some systems, however, data flow is a subsidiary issue, and the approach is less useful.

The third approach, **data structure decomposition**, requires analysts/designers to first build the data structures for a system's input and output data streams. The structure of the system (and its decomposition) is then derived as a natural consequence of the transformations needed to map the input data structures to the output data structures (see, e.g., Jackson 1975, 1983; Warnier 1981; Orr 1981). While data structure design appears to lead to greater consistency among the decompositions obtained by different analysts/designers, it may not work well for all types of systems. For example, Yourdon and Constantine (1979) argue that the approach is useful only when systems are relatively simple. With large systems, "structure clashes" arise that may lead to complex design solutions.

The fourth approach, **higher-order software (HOS)**, requires analysts/designers to first represent a system as a single mathematical function. Decomposition is then undertaken by hierarchically breaking up the function into subfunctions. Successive levels of functions are decomposed iteratively using either mathematical partitioning of functions or composition of functions (Hamilton and Zeldin 1976; Martin 1985). Like data structure decomposition, HOS seems to lead to greater consistency among analysts/designers in their decomposition work. However, like functional decomposition, the basis for decomposition (time, resources, data flow, etc.) is unclear. Moreover, practical experience with the methodology is still limited.

The fifth approach, **object-oriented design**, requires analysts/designers to first identify the objects in the domain of interest and operations upon the objects (see, e.g., Cox 1986; Pressman 1987; Shaler and Mellor 1988). The software realization of an object and the operations on the object are made up of a private part and an interface. The private part is hidden from other objects. It contains the data structure that describes the attributes of the object that are of interest and the operations on the object. The interface is the shared part of the object through which messages are received and transmitted to other objects. Decomposition is attained as a natural result of the information hiding that occurs via the private parts of objects. Furthermore, operations that require two or more objects must access the private part of only one of the objects; otherwise, the objects are considered to be too tightly coupled. While this approach to decomposition seems promising, experience with object-oriented design is still limited.

The sixth approach, **formal derivation**, requires analysts/designers to specify the system as a predicate transformation between a set of input states (the precondition) and a set of output states (the postcondition) (see, e.g., Mills et al. 1987). The predicate transformation is then transformed into a set of simpler transformations. Several strategies can be used to simplify the predicate transformation (see, e.g., Hoare 1987). A common one involves making a sequence of refinements on the predicate transformation, each of which establishes the postcondition for progressively weaker preconditions (Dromey 1988). Eventually, the preconditions are sufficiently weak for the problem to be solved. While the approach provides rigor to the decomposition process, its applicability to large problems is still limited. Moreover, it can lead to multiple decomposition solutions (Bergland 1981).

Which decomposition approach is "best" remains a fundamental, unresolved problem. Often, comparative evaluations of the approaches are undertaken in light of experience with their use on some case study (see, e.g., Bergland 1981). Unfortunately, the generality of the conclusions reached on the basis of case studies is always a moot issue. Accordingly, we argue that a **theory** of decomposition needs to be developed to provide a framework for systematically investigating the strengths and weaknesses of the different decomposition approaches. Each can then be examined in terms of how well it instantiates the theory. Based upon the theory, predictions about the likely strengths and weaknesses of the approach can be generated. These predictions can then be systematically evaluated via empirical research.

3. BASIC CONCEPTS UNDERLYING THE DECOMPOSITION MODEL

In this section, we develop the basic concepts that underlie our model. Since we have formally articulated these concepts at length elsewhere (Wand and Weber 1988, 1989), we proceed below in a concise and intuitive fashion. The concepts are based upon and an extension of a theory of ontology developed by Bunge (1977, 1979).

We view the world as being made up of ***things*** or objects that have known properties. Things are modeled via a ***functional schema***, which is a set of functions that assigns values to the properties of the thing. Each function in the set is called a state function or ***state variable***. A combination of values that the state variables might assume is called a ***state***. The set of all states that the thing might assume is called the ***possible state space*** of the thing, $S = \{s\}$. We represent a state via a ***state vector*** of properties, $s = \langle x_1, ..., x_n \rangle$, and we assume that the state vector contains all the information needed to analyze the thing of interest.

To illustrate these notions, consider Figure 1. The vertices in the graph represent things. For example, raw materials is a thing. It can be described by a state vector of properties such as inventory item number, quantity on hand, and warehouse location. The values that these properties assume at some point constitute a state of the raw materials thing. The set of states that the raw materials thing might assume constitutes the possible state space of the raw materials thing.

The states that a thing might assume are often constrained by ***laws***. Laws reflect restrictions imposed upon things either by nature or humans. For example, if the quantity-on-hand property of the raw materials thing shown in Figure 1 assumes a negative value, the state may be deemed unlawful. The set of states of a thing that are deemed lawful constitutes the ***lawful state space*** of the thing.

The dynamics of a thing are modeled via events and histories. An event arises because an existing state is mapped into a new state via a ***transformation***. Thus, ***events*** can be represented as pairs of states, $e = \langle s, s' \rangle$, where s represents the "before" state, s' represents the "after" state, and $s' = g(s)$. If the before and after states are lawful and the transformation, g, that gives rise to the event is lawful, the event is a ***lawful event***. The set of lawful events that may arise constitutes the ***lawful event space*** of the thing.

To illustrate these notions, consider, again, the raw materials thing shown in Figure 1. If sufficient inventory is on hand, a lawful transformation on the state space of the raw materials thing may be a withdrawal of inventory.

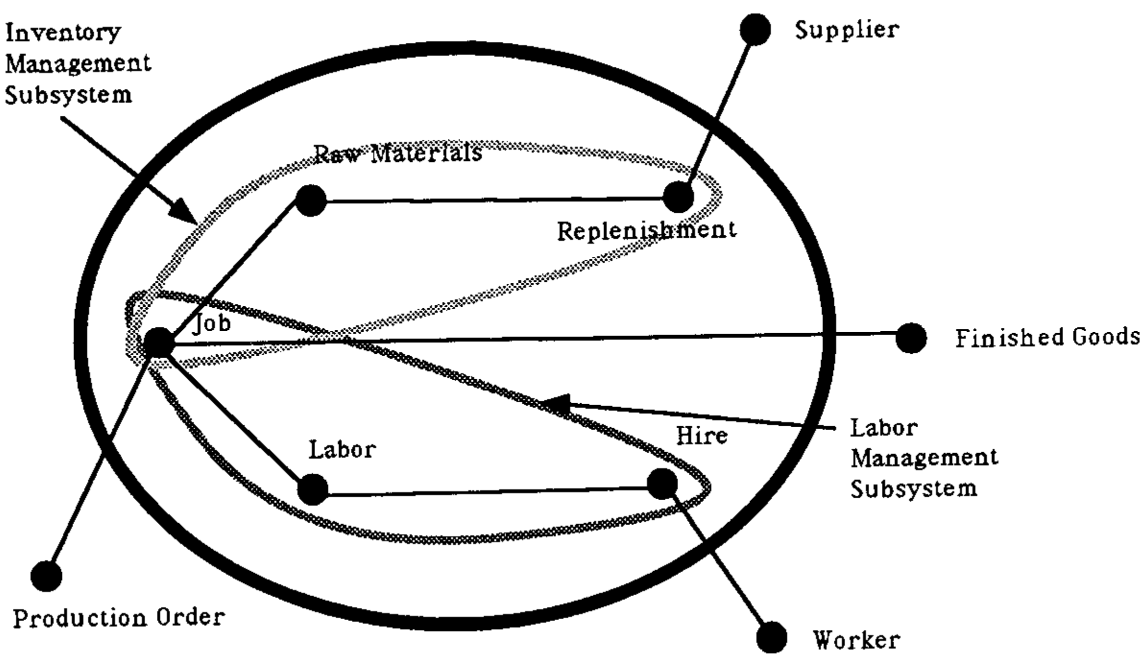

Figure 1. First Decomposition of Production Management System

A new state arises when the value of the quantity-on-hand property changes because inventory has been withdrawn. The change of state represents an event.

A sequence of events gives rise to a history. A ***history*** of a thing is represented via the set of states that the thing assumes across time. For example, a history of the raw materials thing in Figure 1 might contain a series of state changes that have occurred to the thing because replenishments and withdrawals have altered the value of the quantity-on-hand state variable.

Relationships among things are modeled via ***bondings*** or ***couplings***. Two things are coupled when the history of at least one of the things depends upon the history of the other thing -- in other words, the states assumed by one of the things are different because the other thing exists. The latter thing is said to ***act on*** the former thing. For example, in Figure 1 the raw materials thing is coupled to the replenishment thing because the history of the raw materials thing depends upon the history of the replenishment thing.

A set of things that are coupled to each other constitutes a ***system***, providing the set cannot be partitioned into two or more independent subsets of interacting things. The things that make up the system are called its ***composition***. The ***environment*** of the system comprises the set of things which are not in the composition of the system but which are coupled to things in the composition of the system. The ***structure*** of the system comprises the set of couplings among things in the composition of the system (the internal bondings) and among things in the composition of the system and the environment of the system (the external bondings).

For example, Figure 1 shows the graph of a production management system. The composition of the system comprises the replenishment thing, the raw materials thing, the job thing, the labor thing, and the hire thing. Note how all these things are coupled together and how they cannot be partitioned into disjoint subsets. The environment of the system comprises the supplier thing, the production order thing, the worker thing, and the finished goods thing. The structure of the system includes the internal bondings between replenishment and raw materials, raw materials and job, job and labor, and labor and hire. In addition, the structure includes the external bondings between supplier and replenishment, production order and job, worker and hire, and job and finished goods.

A ***subsystem*** is a system which has a composition and structure that are subsets of another system. In addition, its environment is a subset of those things that are in the environment of the system plus those things that are in the composition of the system but not in the composition of the subsystem. In Figure 1, for example, it is easy to show that the system designated as the inventory management subsystem is a subsystem of the production management system because it possesses these characteristics.

Things in the composition of the system which are acted upon by things in the environment of the system are called *input components*. Conversely, things in the composition of the system that act upon things in the environment of the system are called *output components*. For example, in Figure 1 the replenishment thing is an input component because the supplier thing acts upon it to change its state. Similarly, the job thing is an output component because it acts upon the finished goods thing to change its state.

Things may undergo different types of events. An *input event* is a state change that a component of a system experiences by virtue of the action of a thing in the environment of the system on the component. An *output event* is a state change that occurs to a thing in the environment of a system by virtue of the action of a component of the system on the environmental thing. A *processing event* is a state change that occurs to a component of a system by virtue of the action of another thing in the composition of the system on the component.

4. THE FORMAL MODEL

Given our basic concepts, we now develop a formal model that enables us to analyze the nature of a good decomposition. Again, some of the fundamental elements of our formal model of decomposition have been developed elsewhere (Wand and Weber 1989), but we repeat them here as the basis for our analysis.

We begin with the notion of a **decomposition** of a system, not necessarily a good decomposition. Intuitively, a decomposition of a system is a set of subsystems that has three properties: (a) every element in the composition of the system is included in the composition of at least one of the subsystems in the set; (b) the (set) difference between the union of the environments of the subsystems and the composition of the system equals the environment of the system; and (c) every element in the structure of the system is included in the structure of at least one of the subsystems in the set. Formally we have:

Definition 1: Let I be an index set, let σ be a system, and let x_i be a subsystem of σ, denoted $x_i < \sigma$. Then $D(\sigma) = \{x_i\}_{i \in I}$ is a *decomposition* over σ iff:

(a) $\tilde{C}(\sigma) = \cup_{i \in I} \tilde{C}(x_i)$, where $\tilde{C}(\sigma)$ is the composition of the system and $\tilde{C}(x_i)$ is the composition of the i-th subsystem;

(b) $\tilde{E}(\sigma) = \cup_{i \in I}(x_i)\tilde{E} - \tilde{C}(\sigma)$, where $\tilde{E}(\sigma)$ is the environment of the system and $\tilde{E}(x_i)$ is the environment of the i-th subsystem;

(c) $\tilde{S}(\sigma) = \cup_{i \in I} \tilde{S}(x_i)$, where $\tilde{S}(\sigma)$ is the structure of the system and $\tilde{S}(x_i)$ is the structure of the i-th subsystem.

Figure 1 shows that the production management system has been factored into two subsystems: an inventory management subsystem and a labor management subsystem. It is a straightforward exercise to show that these two subsystems constitute a decomposition of the system.

Since subsystems can be nested within other subsystems, the concept of a decomposition leads naturally to the concept of a **level structure** over a system. Formally we have:

Definition 2: Let $\tilde{D}(\sigma)$ be a decomposition of a system, σ, and let $L = \{L^i \mid i = 1, ..., n\}$ be a partition of $D(\sigma)$. Then a ***level structure*** $\tilde{L} = \langle L, < \rangle$ over $D(\sigma)$ is defined **recursively** as follows:

(a) (Basis): $L^0 = \{\sigma\}$

(b) (Induction Step): $L^{i+1}(i = 0, ..., n - 1)$ is a (finite) nonempty set of subsystems such that

$$(\forall x)[x \in L^{i+1} \Rightarrow \exists y \in L^i \wedge x < y].$$

The top panel of Figure 2 shows the level structure for the production management system of Figure 1. The bottom panel has been added to show the composition of the lowest-level subsystems and, ultimately, the composition of each higher-level (sub)system.

We now want to analyze how events propagate through a system -- through the various subsystems and up and down the level structure of a system. Our eventual goal is to design decompositions and level structures that force events to propagate throughout systems in well-defined ways. Accordingly, we begin with the notion of the **state space of a decomposition** of a system:

Definition 3: Let I be an index set, and let $D(\sigma)$ be a decomposition of a system σ. Then the ***possible state space of the decomposition*** is the Cartesian product of the possible state spaces of the subsystems that constitute the decomposition. That is, $S(D(\sigma)) = \otimes_{i \in I} S(x_i)$.

Notation 3(a): Henceforth, $S((D(\sigma))$ will be abbreviated to $S(D)$.

Notation 3(b): s_i^j denotes the j-th state of the i-th subsystem.

To illustrate this concept, consider the two subsystems shown in Figure 1. Assume, for simplicity, that we use only one state variable to describe each of the two subsystems: *RM-Avail* (raw materials availability in units) for the inventory management subsystem; and *Lab-Avail* (labor availability in units) for the labor management subsystem. Furthermore, assume the inventory management subsystem can take on only three states representing differing levels of raw material availability: $S(x_1) = \{s_1^1, s_1^2, s_1^3\} = \{0, 10, 20\}$. Likewise, assume the labor management subsystem can take on only four states representing

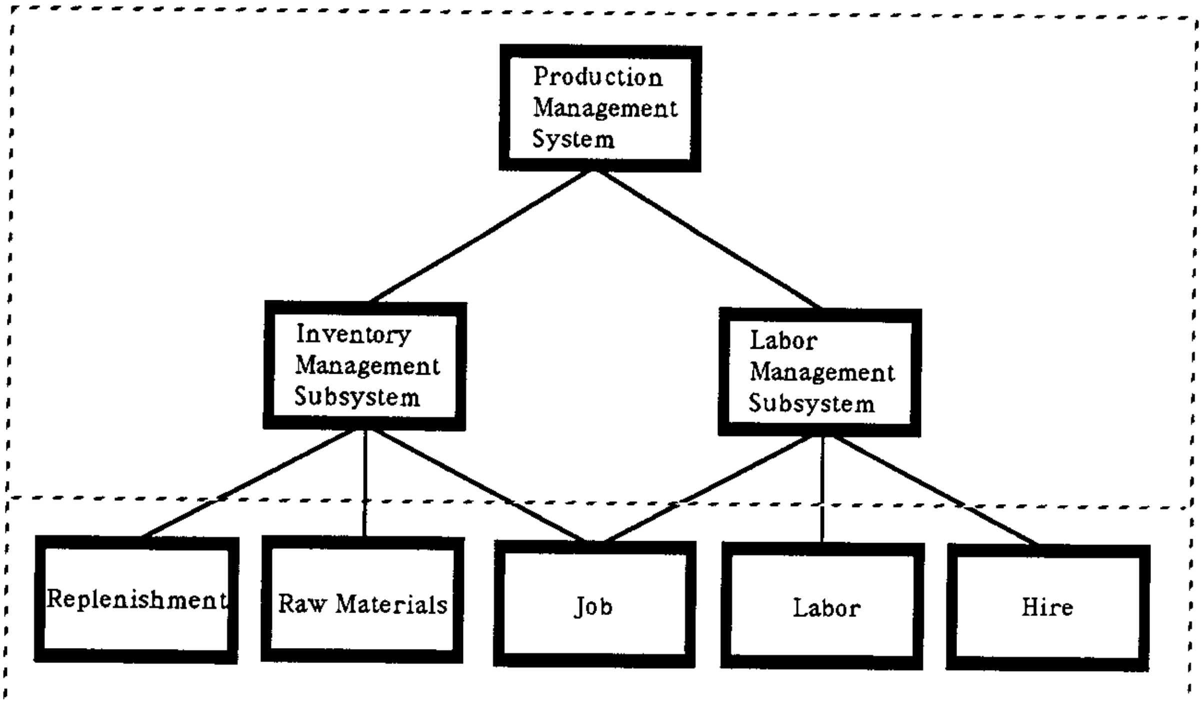

Figure 2. Level Structure for First Decomposition

differing levels of labor availability: $S(x_2) = \{s_2^1, s_2^2, s_2^3, s_2^4\} = \{0, 10, 20, 30\}$. In this light, the state space of the decomposition is:

$$S(D) = \{ (s_1^1, s_2^1), (s_1^1, s_2^2), (s_1^1, s_2^3), (s_1^1, s_2^4), (s_1^2, s_2^1), (s_1^2, s_2^2), (s_1^2, s_2^3), (s_1^2, s_2^4), (s_1^3, s_2^1), (s_1^3, s_2^2), (s_1^3, s_2^3), (s_1^3, s_2^4)\}.$$

$$= \{ (0,0), (0,10), (0,20), (0,30), (10,0), (10,10), (10,20), (10,30), (20,0), (20,10), (20,20), (20,30)\}.$$

Since system states are a manifestation of subsystem states (and *vice versa*), it must always be possible to map a state of the system into one or more states of a decomposition of the system. Thus we have:

Lemma 1: Let $S(\sigma)$ be the possible state space of a system, σ, let $S(D)$ be the possible state space of a decomposition of σ, and let $P(S(D))$ be the power set of $S(D)$. Then a function, H, exists that maps the possible state space of the system into the power set of the possible state space of a decomposition of the system. That is, $H\colon S(\sigma) \rightarrow P(S(D))$.

For example, assume only one state variable is used to describe the state of the production management system, x_0, shown in Figure 1: *Prod-Cap* (available production capacity in units). Furthermore, assume that production must occur in lot sizes of 5 units and that each unit of finished goods requires 2 units of raw materials and 4 units of labor to produce. Given the state space of the decomposition of the system described above, the following mapping shows the circumstances under which 0 and 5 units of finished goods can be produced:

$$H(s_0^1) = H(0) = \{ (s_1^1, s_2^1), (s_1^1, s_2^2), (s_1^1, s_2^3), (s_1^1, s_2^4), (s_1^2, s_2^1), (s_1^2, s_2^2), (s_1^3, s_2^1)\ (s_1^3, s_2^2)\}$$

$$= \{ (0,0), (0,10), (0,20), (0,30), (10,0), (10,10), (20,0), (20,10)\}$$

$$H(s_0^2) = H(5) = \{ (s_1^2, s_2^3), (s_1^2, s_2^4), (s_1^3, s_2^3), (s_1^3, s_2^4)\}$$

$$= \{ (10,20), (10,30), (20,20), (20,30)\}$$

It is important to note that a state of the system may map into more than one state of a decomposition of the system. In other words, if we "observe" the system only at the system level, we may not know the states of the subsystems because a one-one mapping (an injection function) may not exist between the state space of the system and the state space of a decomposition of the system.

We now show how events observed at the system level are manifested as events in the subsystems of the decomposition. To aid our analysis, we first introduce some additional notation:

Notation 3(c): Let $s \in S(\sigma)$. Denote by $d(s) \in P(S(D))$ the set of all corresponding states in $S(D)$, $H(s) = d(s)$. Denote by $d^j(s)$ the j-th element of (ds). Denote by $d_i^j(s)$ the i-th component of the j-th element of $d(s)$, namely, the state of the i-th subsystem when the system is in state $s \in S(\sigma)$. Note, $d_i^j(s)$ will be termed the **projection** of state s in subsystem x_i.

To illustrate the notation using the production management system example described above:

$$H(s_0^2) = d(s_0^2) = \{(s_1^2, s_2^3), (s_1^2, s_2^4), (s_1^3, s_2^3), (s_1^3, s_2^4)\}$$

Thus:

$$d^1(s_0^2) = (s_1^2, s_2^3) \text{ and } d_1^1(s_0^2) = s_1^2,\ d_2^1(s_0^2) = s_2^3$$

$$d^2(s_0^2) = (s_1^2, s_2^4) \text{ and } d_1^2(s_0^2) = s_1^2,\ d_2^2(s_0^2) = s_2^4$$

$$d^3(s_0^2) = (s_1^3, s_2^3) \text{ and } d_1^3(s_0^2) = s_1^3,\ d_2^3(s_0^2) = s_2^3$$

$$d^4(s_0^2) = (s_1^3, s_2^4) \text{ and } d_1^4(s_0^2) = s_1^3,\ d_2^4(s_0^2) = s_2^4$$

Using this notation, we are now able to define the notion of an **induced event**. Intuitively, an induced event is an event that occurs in a subsystem of a system as a manifestation of an event that occurs in the system. If we observe a change of state in the system, at least one of its subsystems must undergo a change of state. This change of state in the subsystem is the induced event. Formally we have:

Definition 4: Let $\langle s, s' \rangle \in E(\sigma)$ be an event in the possible event space of the system, and let $d^j(s)$ and $d^k(s')$ be the corresponding states in $P(S(D))$ (j and k are not necessarily distinct). Let $d_i^j(s)$ and $d_i^k(s')$ be the state of the i-th subsystem when the system is in states s and s' respectively. Then the event $\langle d_i^j(s),\ d_i^k(s')\rangle$ will be called an ***induced event*** on subsystem $x_i < \sigma$ iff $d_i^j(s) \neq d_i^k(s')$. The induced event on the subsystem x_i will be designated e_i.

To illustrate the notion of an induced event, assume the event $\langle s_0^1, s_0^2\rangle$ occurs in the production management system. Furthermore, assume that $d^5(s_0^1)$ is the state of the decomposition when the system is in state s_0^1 and $d^1(s_0^2)$ is the state of the decomposition when the system is in state s_0^2. Thus we have:

$$d^5(s_0^1) = (s_1^2, s_2^1) = (10,0)$$

$$d^1(s_0^2) = (s_1^2, s_2^3) = (10,20)$$

$$d_1^5(s_0^1) = s_1^2 = 10$$

$$d_2^5(s_0^1) = s_2^1 = 0$$

$$d_1^1(s_0^2) = s_1^2 = 10$$

$$d_2^1(s_0^2) = s_2^3 = 20$$

Since $d_1^5(s_0^1) = d_1^1(s_0^2)$, the system event has **not** produced an induced event in the inventory management subsystem. However, an induced event has occurred in the labor management subsystem because $d_2^5(s_0^1) \neq d_2^1(s_0^2)$. In short, the increase of productive capacity of 5 units has occurred because 20 more units of labor have become available, presumably because more labor has been hired.

Note that whenever a change of state occurs in a system, it will always be manifested as an induced event in at least one of its subsystems. Formally, we have:

Lemma 2: $\forall(\langle s, s'\rangle) \in E(\sigma)$, $\exists x_i$, $x_i < \sigma$, such that $\exists d_i^j(s)$, $\exists d_i^k(s')$ and $d_i^j(s) \neq d_i^k(s')$. Note, j and k are not necessarily distinct.

However, the converse is not true: changes of state in a subsystem will not always be manifested as changes of state in the system. For example, in the production management system, assume the following event occurs in the event space of its decomposition: $\langle(s_1^1, s_2^1),(s_1^2, s_2^2)\rangle$. A change of state will not be produced in the system; the system will remain in s_0^1.

We turn, now, to address the concept of a good decomposition more directly. We begin with the notion of **subsystem equivalence states:**

Definition 5: Let s_0^j be the j-th state of the system, let $s_i^k(i > 0)$ be the state of the i-th subsystem, and let $d_i(s_0^j)$ be the set of possible states of the i-th subsystem when the system is in state j. Then the set of **system** states, S_i^k, that map into the same state k of the i-th subsystem will be called the ***subsystem equivalence states.*** That is, $S_i^k = \{s \mid d_i^l(s) = s_i^k\}$. Note, $d_i^l(s)$ is the l-th element of the set of possible states of the i-th subsystem when the system is in state s.

To illustrate this concept, consider the production management system example. Given the two states the system can assume -- $S(x_0) = \{s_0^1, s_0^2\} = \{0,5\}$ -- the sets of subsystem equivalence states are:

$$S_1^1 = \{s_0^1\} \quad ; \quad d_1^1(s_0^1) = d_1^2(s_0^1) = d_1^3(s_0^1) = d_1^4(s_0^1) = 0$$

$$S_1^2 = \{s_0^1, s_0^2\} \quad ; \quad d_1^5(s_0^1) = d_1^6(s_0^1) = d_1^1(s_0^2) = d_1^2(s_0^2) = 10$$

$$S_1^3 = \{s_0^1, s_0^2\} \quad ; \quad d_1^7(s_0^1) = d_1^8(s_0^1) = d_1^3(s_0^2) = d_1^4(s_0^2) = 20$$

$$S_2^1 = \{s_0^1\} \quad ; \quad d_2^1(s_0^1) = d_2^5(s_0^1) = d_2^7(s_0^2) = 0$$

$$S_2^2 = \{s_0^2\} \quad ; \quad d_2^2(s_0^1) = d_2^6(s_0^1) = d_2^8(s_0^2) = 10$$

$$S_2^3 = \{s_0^1, s_0^2\} \quad ; \quad d_2^3(s_0^1) = d_2^1(s_0^1) = d_2^3(s_0^2) = 20$$

$$S_2^4 = \{s_0^1, s_0^2\} \quad ; \quad d_2^4(s_0^1) = d_2^2(s_0^2) = d_2^4(s_0^2) = 30$$

In light of the notion of subsystem equivalence states, we can now define the **decomposition condition:**

Definition 6: Let g be a transformation on a system, σ, and let S_i^k be the set of subsystem equivalence states for the i-th subsystem in the k-th state. The ***decomposition condition*** holds when:

$s \in S_i^k \Rightarrow g(s) \in S_i^h$, k and h are not necessarily distinct.

Corollary 6: $d_i^l(s) = d_i^m(s') \Rightarrow d_i(g(s)) \cap d_i(g(s')) \neq \emptyset$

For the decomposition condition to hold, subsystems must behave independently. Given we know that the i-th subsystem is in state $y = d_i^k(s)$ when the system is in state s, we have sufficient knowledge to predict the new state $z = d_i^l(g(s))$ when an event occurs. Thus, the new system states that arise by virtue of a transformation on the states in a set of subsystem equivalence states must also map into the same new subsystem state. Otherwise, it is impossible to predict the new state of the subsystem when a transformation occurs on the system states.

We formalize these notions in terms of the concepts of a **well-defined, induced transformation** and a **good decomposition:**

Definition 7: Let g be a transformation on a system σ. Then the transformation, g_i, induced on the subsystem $x_i < \sigma$ is *well-defined* iff g_i is a function, i.e., $g_i(s) = s'$ and $g_i(s) = s'' \Rightarrow s' = s''$ for every s, s' $s'' \in S(x_i)$.

Definition 8: A decomposition, $D(\sigma)$, of a system is a *good decomposition* with respect to a transformation, g, on the system iff the transformation g induces a well-defined transformation, g_i, in every subsystem, x_i, of the decomposition.

5. APPLYING THE DECOMPOSITION FORMALISM: AN EXAMPLE

To illustrate the notion of a good decomposition in the context of the model, consider, again, the production management system. Assume that we have a transformation, g^1, which effects a change of production capacity in light of a production order for 5 units of finished goods. Thus, the mapping for g^1 is:

$$g^1(s_0^1) = s_0^1, \text{ i.e., } g^1(0) = 0$$

$$g^1(s_0^2) = s_0^1, \text{ i.e., } g^1(5) = 0$$

In short, if a production order for 5 units is received and the production capacity is 5 units, the production capacity is reduced by 5 units to reflect that production of these units has started.

Consider, now, the transformation g_1^1 induced on the inventory management subsystem, x_1. Table 1 shows the changes of state that occur as manifested in the values of the *RM-Avail* state variable. The mapping for g_1^1 is:

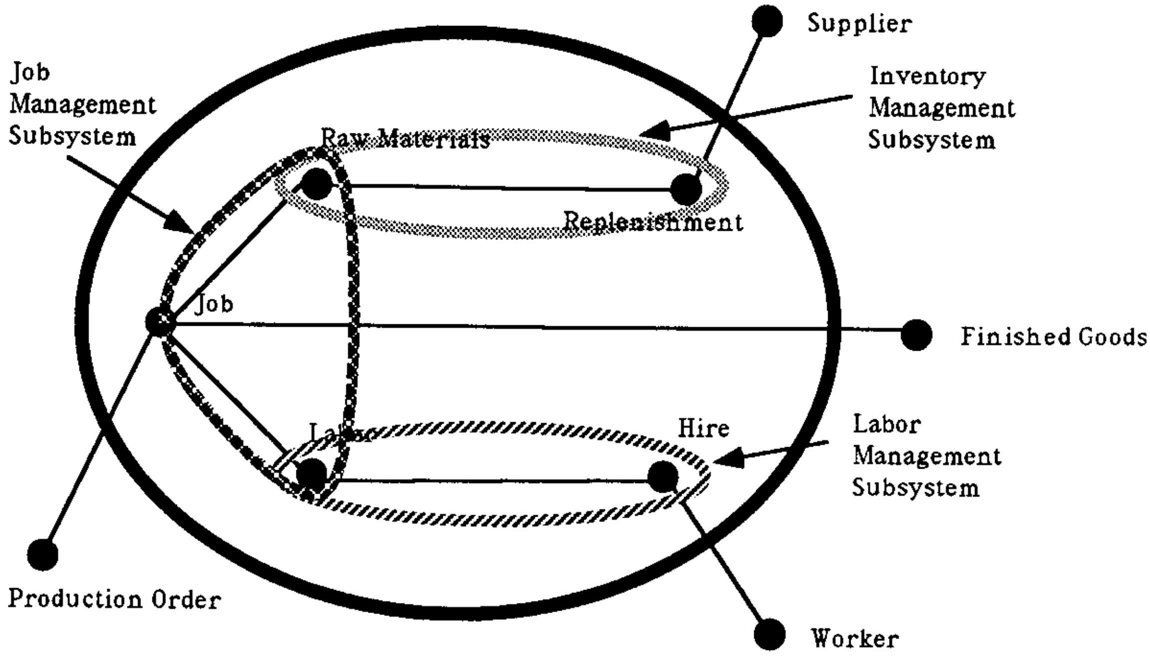

Figure 3. Second Decomposition of Production Management System

$g_1^1(s_1^1) = s_1^1$, i.e., $g_1^1(0) = 0$

$g_1^1(s_1^2) = \{s_1^1, s_1^2\}$, i.e., $g_1^1(10) = \{0,10\}$

$g_1^1(s_1^3) = \{s_1^2, s_1^3\}$, i.e., $g_1^1(20) = \{10,20\}$

Clearly the transformation induced on the inventory management subsystem is not well defined.

Table 1. State Changes Produced by Production Order Transformation

Before			After		
RM-Avail	Lab-Avail	Prod-Cap	RM-Avail	Lab-Avail	Prod-Cap
0	0	0	0	0	0
0	10	0	0	10	0
0	20	0	0	20	0
0	30	0	0	30	0
10	0	0	10	0	0
10	10	0	10	10	0
10	20	5	0	0	0
10	30	5	0	10	0
20	0	0	20	0	0
20	10	0	20	10	0
20	20	5	10	0	0
20	30	5	10	10	0

Similarly, consider the transformation, g_2^1, induced on the labor management subsystem, x_2. Table 1 shows the changes of state that occur as manifested in the values of the *Lab-Avail* state variable. The mapping for g_2^1 is:

$g_2^1(s_2^1) = s_2^1$, i.e., $g_2^1(0) = 0$

$g_2^1(s_2^2) = s_2^2$, i.e., $g_2^2(10) = 10$

$g_2^1(s_2^3) = \{s_2^1, s_2^3\}$, i.e., $g_2^1(20) = 20$

$g_2^1(s_2^4) = \{s_2^2, s_2^4\}$, i.e., $g_2^1(30) = 30$

Again, the transformation induced on the labor management subsystem is not well defined.

We conclude, therefore, that the decomposition shown in Figure 1 is not a good decomposition with respect to the production order transformation. Indeed, the formal results probably reflect our intuition about the decomposition -- namely, given the important bonding between the *RM-Avail* and *Lab-Avail* state variables, the components whose states these values manifest ought to be together in the same subsystem.

Consider, now, an alternative decomposition of the production management system -- the decomposition shown in Figure 3. The system has been factored into three subsystems: an inventory management subsystem, x_1; a labor management subsystem, x_2; and a job management subsystem, x_3. Assume that the states of the inventory management and labor management subsystems are represented by the *RM-Avail* and *Lab-Avail* state variables respectively. Furthermore, assume that the states of the job management subsystem are represented by the state *vector* (*RM-Avail*, *Lab-Avail*).

Under this decomposition, the transformation, g_3^1, is induced on the job management system. From Table 1, the mapping can be derived as follows:

$$g_3^1(\langle s_1^1, s_2^1\rangle) = \langle s_1^1, s_2^1\rangle \text{, i.e., } g_3^1(\langle 0{,}0\rangle) = \langle 0{,}0\rangle$$

$$g_3^1(\langle s_1^1, s_2^2\rangle) = \langle s_1^1, s_2^2\rangle \text{, i.e., } g_3^1(\langle 0{,}10\rangle) = \langle 0{,}10\rangle$$

$$g_3^1(\langle s_1^1, s_2^3\rangle) = \langle s_1^1, s_2^3\rangle \text{, i.e., } g_3^1(\langle 0{,}20\rangle) = \langle 0{,}20\rangle$$

$$g_3^1(\langle s_1^2, s_2^4\rangle) = \langle s_1^1, s_2^4\rangle \text{, i.e., } g_3^1(\langle 0{,}30\rangle) = \langle 0{,}30\rangle$$

$$g_3^1(\langle s_1^2, s_2^1\rangle) = \langle s_1^2, s_2^1\rangle \text{, i.e., } g_3^1(\langle 10{,}0\rangle) = \langle 10{,}0\rangle$$

$$g_3^1(\langle s_1^2, s_2^2\rangle) = \langle s_1^2, s_2^2\rangle \text{, i.e., } g_3^1(\langle 10{,}10\rangle) = \langle 10{,}10\rangle$$

$$g_3^1(\langle s_1^2, s_2^3\rangle) = \langle s_1^1, s_2^1\rangle \text{, i.e., } g_3^1(\langle 10{,}20\rangle) = \langle 0{,}0\rangle$$

$$g_3^1(\langle s_1^3, s_2^4\rangle) = \langle s_1^1, s_2^2\rangle \text{, i.e., } g_3^1(\langle 10{,}30\rangle) = \langle 0{,}10\rangle$$

$$g_3^1(\langle s_1^3, s_2^1\rangle) = \langle s_1^3, s_2^1\rangle \text{, i.e., } g_3^1(\langle 20{,}0\rangle) = \langle 20{,}0\rangle$$

$$g_3^1(\langle s_1^3, s_2^2\rangle) = \langle s_1^3, s_2^2\rangle \text{, i.e., } g_3^1(\langle 20{,}10\rangle) = \langle 20{,}10\rangle$$

$$g_3^1(\langle s_1^3, s_2^3\rangle) = \langle s_1^2, s_2^1\rangle \text{, i.e., } g_3^1(\langle 20{,}20\rangle) = \langle 10{,}0\rangle$$

$$g_3^1(\langle s_1^3, s_2^4\rangle) = \langle s_1^2, s_2^1\rangle \text{, i.e., } g_3^1(\langle 20{,}30\rangle) = \langle 10{,}10\rangle$$

Since the mapping shows that g_3^1 is a function, the transformation induced on the job management subsystem by the job order transformation is well defined.

Table 2. State Changes Produced by Price Increase Transformation

Before			After		
RM-Val	Lab-Val	Tot-Val	RM-Val	Lab-Val	Tot-Val
0	0	0	0	0	0
0	10	10	0	20	20
0	20	20	0	40	40
0	30	30	0	60	60
10	0	10	20	0	20
10	10	20	20	20	40
10	20	30	20	40	60
10	30	40	20	60	80
20	0	20	40	0	40
20	10	30	40	20	60
20	20	40	40	40	80
20	30	50	40	60	100

Note that g^1 does not induce a transformation on either the inventory management subsystem nor the labor management subsystem under the new decomposition. While both subsystems may undergo a change of state because they each share a state variable with the job management subsystem, the change of state does not manifest that an induced transformation has occurred on the subsystem. Rather, it manifests an **input** to each subsystem.

Assume, now, that another state variable is used to describe the production management system: *Tot-Val,* representing the total value of raw material and labor assets in the system. Similarly, assume that the inventory management and labor management subsystems are now described via the state variables *RM-Val* (value of raw materials) and *Lab-Val* (value of labor) respectively. To simplify matters, assume, further, that a unit of raw materials and a unit of labor are each worth $1. In this light, the "before" columns of Table 2 show the possible states of each of these state variables. Note that the value of *Tot-Val* is simply the sum of *RM-Val* and *Lab-Val.*

Assume, now, that an across-the-board increase of 100 percent occurs in the value of all assets (e.g., a large inflationary increase in prices). Let g^2 be the transformation that effects the change on *Tot-Val.* Thus, the mapping will be:

$$g^2(0) = 0$$

$$g^2(10) = 20$$

$$g^2(20) = 40$$

$$g^2(30) = 60$$

$$g^2(40) = 80$$

$$g^2(50) = 100$$

Furthermore, under the two decompositions shown in Figures 1 and 3, the transformations induced on the inventory management subsystem and the labor management subsystem will be:

$$g_1^2(0) = 0 \qquad g_2^2(0) = 0$$

$$g_1^2(10) = 20 \qquad g_2^2(10) = 20$$

$$g_1^2(20) = 40 \qquad g_2^2(20) = 40$$

$$g_2^2(30) = 60$$

Clearly, both induced transformations are well formed, and both the Figure 1 and Figure 3 decompositions are good decompositions under the transformation g^2.

Our example illustrates, therefore, that the goodness of a decomposition must be evaluated **with respect to a particular transformation.** A decomposition may be good in the context of one transformation but poor in the context of another transformation. Thus the formalism reflects what intuitively we might expect -- namely, that the goodness of an existing decomposition might be undermined as new transformations (e.g., modifications to the system or different transaction types) must be taken into account.

6. FUTURE RESEARCH DIRECTIONS AND CONCLUSIONS

There are at least three major research directions that can be pursued in light of our model. First, an attempt can be made to express existing decomposition methodologies in terms of the constructs and relationships used in the model. For example, the object-oriented approach to decomposition can be formalized via things, states, laws, bondings, etc. (Wand 1989). The ability of the model to describe existing decomposition methodologies using a common language is an important test of the model's adequacy.

Second, if the decomposition methodologies can be expressed in terms of the model, an attempt can then be made to use the model to generate predictions about the strengths and weaknesses of the different methodologies. Specifically, the methodologies can be evaluated to determine whether they generate decompositions that always comply with the good decomposition condition. The circumstances under which a decomposition methodology does or does not comply with the good decomposition condition also might be identified.

Third, once the predicted strengths and weaknesses of the different decomposition methodologies have been determined via the model, empirical tests of the predictions can then be undertaken. Hopefully, the model will allow better-directed empirical tests than the isolated and somewhat random case study comparisons among the methodologies that have been undertaken in the past.

7. ACKNOWLEDGEMENTS

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

8. REFERENCES

Bergland, G. D. "A Guided Tour of Program Design Methodologies." *IEEE Computer*, Volume 14, October 1981, pp. 13-37.

Bunge, M. *Treatise on Basic Philosophy: Volume 3: Ontology I: The Furniture of the World*. Boston, Massachusetts: Reidel, 1977.

Bunge, M. *Treatise on Basic Philosophy: Volume 4: Ontology II: A World of Systems*. Boston, Massachusetts: Reidel, 1979.

Cox, B. *Object Oriented Programming*. Reading, Massachusetts: Addison-Wesley, 1986.

Dijkstra, E. W. *A Discipline of Programming*. Englewood Cliffs, New Jersey: Prentice-Hall, 1976.

Dromey, R. G. "Systematic Program Development." *IEEE Transactions on Software Engineering*, Volume SE-14, January 1988, pp. 12-29.

Hamilton, M., and Zeldin, S. "Higher-Order Software: A Methodology for Defining Software." *IEEE Transactions on Software Engineering*, Volume SE-2, March 1976, pp. 9-32.

Hoare, C. A. R. "An Overview of Some Formal Design Methods for Program Design." *IEEE Computer*, Volume 20, September 1987, pp. 85-91.

Jackson, M. *Principles of Program Design*. New York: Academic Press, 1975.

Jackson, M. *System Development*. Englewood Cliffs, New Jersey: Prentice-Hall, 1983.

Linger, R. C.; Mills, H. D.; and Witt, B. I. *Structured Programming--Theory and Practice*. Reading, Massachusetts: Addison-Wesley, 1979.

Martin, J. *System Design from Provably Correct Constructs*. Englewood Cliffs, New Jersey: Prentice-Hall, 1985.

Mills, H. G.; Basili, V. R.; Gannon, J. D.; and Hamlet, R. G. *Principles of Computer Programming: A Mathematical Approach*. Boston, Massachusetts: Allyn and Bacon, 1987.

Myers, G. J. *Reliable Software through Composite Design*. New York: Petrocelli/Charter, 1975.

Orr, K. *Structured Requirements Definition*. Topeka, Kansas: Ken Orr and Associates, 1981.

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

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

Shaler, S., and Mellor, S. J. *Object-Oriented Systems Analysis*. Englewood Cliffs, New Jersey: Yourdon Press, 1988.

Wand, Y. "A Proposal for a Formal Model of Objects." In *Object-Oriented Concepts, Applications, and Databases*. Reading, Massachusetts: Addison-Wesley, 1989, publication forthcoming.

Wand, Y., and Weber, R. "A Model of Control and Audit Procedure Change in Evolving Data Processing Systems." *The Accounting Review*, Volume LXIV, January 1989, pp. 87-107.

Wand, Y., and Weber, R. "An Ontological Analysis of Some Fundamental Information System Concepts." *Proceedings of the Ninth International Conference on Information Systems*, Minneapolis, MN, November 1988, pp. 213-226.

Warnier, J. D. *Logical Construction of Systems*. New York: Van Nostrand Reinhold, 1981.

Wirth, N. *Systematic Programming*. Englewood Cliffs, New Jersey: Prentice-Hall, 1973.

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

Zave, P. "The Operational versus the Conventional Approach to Software Development." *Communications of the ACM*, Volume 27, February 1984, pp. 104-118.