An Ontological Model of an Information System

Yair Wand and Ron Weber

***Abstract*—Theoretical developments in the CS and IS disciplines have been inhibited by inadequate formalization of basic constructs. In this paper we propose an ontological model of an information system that provides precise definitions of fundamental concepts like system, subsystem, and coupling. We use this model to analyze some static and dynamic properties of an information system and to examine the question of what constitutes a "good" decomposition of an information system.**

***Index* Terms-Cohesion, coupling, decomposition, hierarchy, level structure, subsystem, system.**

I. Introduction

THE computer science (CS) and information systems (IS) fields are replete with fundamental concepts that are poorly defined. For example, concepts like system, subsystem, module, object, interface, coupling, cohesion, hierarchy, input, output, environment, and decomposition are central to many theories in both fields, but inevitably they have not been articulated rigorously. In the absence of carefully formulated foundations, fields are unlikely to progress quickly. We concur with Parnas [30, p. 19] who observes: "The use of . . . fuzzy terms (in computer science) is not merely sloppy wording; it prevents . . . the systematic analyses made possible by precise definitions." Moreover, we believe the current debate on the status of both the CS and IS fields reflects that progress has been undermined substantially by inadequate formalization of basic constructs [9], [25], [43].

In this paper we propose a formal model of an information system. We have three primary objectives. First, we seek to define a set of core concepts that can be used to describe the structure and behavior of an information system. Ultimately, we hope these concepts will allow the similarities and differences between many constructs used in the CS and IS fields to be elucidated. Second, we seek to better understand the static and dynamic properties of information systems. In this respect we are viewing information systems as objects (artifacts) to be studied in their own right, independently of the characteristics of their users, the organizations in which they are employed, or the technologies used to implement them [38], [43]. Third, we seek to make predictions about information systems based upon their static and dynamic properties. For example, we have sought to use our model to predict the behavior of information systems based upon the characteristics of alternative decompositions of the system [41] and to predict the consequences of changes to the information system for the reliability of controls that have been implemented in the information system [40]. In short, we have the usual goals for any model: understanding and prediction.

Manuscript received June 1, 1989; revised June 6, 1990. Recommended by M. Deutsch. This work was supported in part by an NSERC operating grant and by a grant from GWA Ltd.

Y. Wand is with the Faculty of Commerce and Business Administration, University of British Columbia, Vancouver, B.C. V6T 1Y8, Canada.

R. Weber is with the Department of Commerce, University of Queensland, St. Lucia, Queensland 4067, Australia.

IEEE Log Number 9038333.

Our approach is to model information systems within the context of a theory of ontology that is a modification and extension of one developed by Bunge [5], [6]. Because ontology is concerned with the structure of the real world, its relevance to the CS and IS fields is twofold. First, since information systems themselves are models of the real world, ontology identifies the basic things in the real world that information systems ought to be able to model. Second, since information systems are also things in the real world, ontology provides a basis for modeling information systems themselves [3], [4]. Elsewhere we have used ontology in the former way [42]. In this paper, we use ontology in the latter way.

Our approach differs from a number of other attempts to provide formal bases for modeling CS and IS constructs. For example, specification languages such as Z and VDM seek to provide a formal notation that describes the properties an information system must have without being constrained by implementation considerations [15], [18], [34]. Unlike our approach, however, they do not seek to define a set of core constructs that underlie the CS and IS fields. Users of formal specification languages are assumed (implicitly) to know these constructs at the outset. The languages can then be employed to express these constructs precisely. While rigorous specification of information systems is clearly an important goal, we see our objectives in modeling core constructs as being more fundamental.

The remainder of the paper proceeds as follows. Section II provides a brief review of some major types of information systems formalisms that bear on our goals and their respective strengths and weaknesses relative to our model. Section 111 articulates some of the fundamental notions that underlie our model. Section IV uses these basic notions to examine the nature of and some dynamics of system decompositions. Section V provides several insights into the model's predictive power via an analysis of some properties of good decompositions. In particular, we show how our model formalizes the intuitive notion that good decompositions have subsystems that behave relatively independently of one another. Finally, Section VI presents suggestions for further research and our conclusions.

II. Prior Research

The motivation to formalize system concepts is not just confined to the CS and IS domains. It is common to fields like engineering [11], cybernetics [44], biology [23], physiology [36], architecture [1], and general systems theory [21]. Bunge [6] provides a brief review of the major types of system models that have been proposed.

In the CS and IS fields, however, there now seems to be widespread acceptance of formal models that view systems (programs) in terms of their specifications: specifically, as a pair of input and output assertions; or as a function mapping input states to output states; or as a relation between input and output states [12], [20], [22], [24], [35]. In turn, *states* are conceived as mappings between system (program) identifiers and values [13].

0098-5589/90/1100-1282$01.00

While these types of models have provided valuable insights into such problems as design decomposition and proofs of program correctness, they are essentially black-box models. Accordingly, they suffer the limitations of all black-box models in terms of our goals of understanding and predicting system behavior [6]. For example, since they do not model the internal structure of a system, the relationship between the overall behavior of the system and the behavior of its internal components is difficult, if not impossible, to analyze.

To illustrate the problems of using these black-box models, consider the objective of attaining a good design decomposition during system implementation. If, say, the system to be designed is conceived as a mapping between input and output states, various design decomposition rules can be derived based upon well-known mathematical results for decomposition of functions [14], [22]. The subfunctions that arise from the decomposition can be conceived as subsystems of the system. A control hierarchy can then be defined among the function (system) and subfunctions (subsystems) [14]. Under this conceptualization, the focus is on what the system is supposed to achieve rather than bow it achieves it.

On the other hand, if the system is conceived as a hierarchy of modules, our focus shifts to internal structural matters. A programmer, for example, spatially arranges the various components of the system (e.g., program instructions and variables) and connects them via a control structure so they execute in particular sequences. Programmers are admonished to achieve objectives like loose coupling between modules and tight internal cohesion Within a module [26], [46].

With current systems formalisms that we have available, however, the mapping between the external view (what is to be accomplished) and the internal view (how it is to be accomplished) is not clear-cut. How do we know, for example, that we have chosen and implemented subfunctions in such a way that the resulting modules are internally cohesive and loosely coupled? Indeed, as Bergland [2, p. 35] points out in his review of decomposition methodologies, "all of the methodologies rely on some magic."

We propose that the quality of the mapping between external and internal views of the system can only be examined and evaluated when we have an integrated formalism for both views. Currently, we have well-developed formalisms for only the external view. However, formalisms that address the internal view (e.g., concepts like coupling) are poorly developed [2].

In summary, we argue that current CS/IS systems formalisms for viewing systems are deficient in that they primarily use black-box rather than white-box models. Until rigorous white-box models of information systems have been developed, progress in developing mappings between requirements specifications and implementation structures will be impeded. In the model we develop below, we seek to provide the rudiments of a white-box model that can a) accommodate current formalisms which view systems as mappings between inputs and outputs, and b) rigorously describe the internal structure of systems.

III. Basic Notions

In this section we develop some basic ontological notions required to analyze certain static and dynamic properties of information systems. To achieve generality, we formalize the concepts using standard mathematical notation. Our experience is that the concepts, once understood, can easily be translated into the syntax of a formal specification language like Z.

TABLE I
STATE DESCRIPTION OF FIVE THINGS

Thing	Property	State Variable
Inventory Item	Item Number	Item-No
	Quantity-on-Hand	QOH
	Unit Price	Unit-Price
	Item Discount (%)	Discount
Customer Order	Order Number	Order-No
	Customer Number	Cust-No
	Item Number	Item-No
	Quantity Ordered	Qty-Ord
	Quantity Supplied	Qty-Supl
	Sales Price	Sale-Pr
	Sales Amount	Sale-Amt
	Date	Date
	Processed Flag	Proc
Customer Account	Customer Number	Cust-No
	Customer Address	Cust-Add
	Balance Due	Bal-Due
	Credit Limit	Cr-Limit
Customer Repayment	Customer Number	Cust-No
	Repayment Amount	Amount
	Date	Date
	Processed Flag	Proc
Inventory Replenishment	Shipment Number	Ship-No
	Item Number	Item-No
	Date	Date
	Replenishment Amount	Replen
	Processed Flag	Proc

A. Things, Properties of Things, and States of Things

The elementary notion in our formalism is a thing.' We define first the state space of a thing (definitions taken from or adapted from Bunge [5], [6] are starred "*").

Definition I *: Let X be a thing modeled by a *functional schema* $X_m = (M, \tilde{F})$, and let each component of the function

$$\tilde{F} = \langle F_1, \cdots, F_n \rangle : M \rightarrow V_1 \otimes \cdots \otimes V_n$$

represent a *property* of X. Then F_i, $1 \leq i \leq n$, is called the ith *state function* (variable) of X, $\tilde{F}$ is called the *total state function* of X, and $S(X) = \{\langle x_1, \cdots, x_n \rangle \in V_1 \otimes \cdots \otimes V_n | x_i = F_i(M)\}$ is called the *possible state space* of X.[2]

To illustrate our formalism, Table I shows a state description for five things: an inventory item, a customer order, a customer account, a customer repayment, and an inventory replenishment. Note that Table I shows only one possible state description of the things. For example, we have not used state variables to describe the physical dimensions of the inventory item thing. In this respect, we have chosen a specific *functional schema* to reflect our particular purposes (simplicity) in modeling a subset of the real world. In short, our formalism recognizes that absolute state variables do not exist. Instead, state variables are chosen to reflect our knowledge, goals, views, etc., at some time. Moreover, different observers of the same thing may choose to model it differently.

[1] Note, we do not equate things with objects. Elsewhere we have argued that objects are special types of things [37]. All objects are things, but only some types of things are objects.

[2] The nature of the domain M is sometimes complex. In systems theory it is often considered to be a set of time instants, or the Cartesian product of a set of reference frames and the real line [5, p. 120].

The possible ***state*** *space* of the inventory item thing would be represented by the set of all combinations of values that the state variables might assume-the Cartesian product (represented by $\otimes$) of the ranges of the state variables. Furthermore, the state space of a thing must be modeled so it includes all possible states that the thing might assume from the beginning to the end of its life. For the inventory item thing, for example, we must choose a total state function $\tilde{F}$ so that some subsets of the state space reflect that the values of state variables describing the inventory item have been updated and others reflect new state variables have been chosen to describe the inventory item. This view of the state space of a thing reflects what Bunge (5, p. 221) calls the *principle* ***of*** *nominal invariance:* "a thing, if named, shall keep its name throughout its history as long as the latter does not include changes in natural kind--changes which call for changes of name."

B. Laws and Lawful States

We proceed, now, to recognize that not all states of a thing are lawful. The values of the attributes of a thing may be restricted by various rules, and these rules define the lawful state space of a thing. We begin with the notion *of a law statement.*

Definition 2:* Let $X_m = \langle M, F\rangle$ be a functional schema for a thing X. Any restrictions on the possible values of the components of $\tilde{F}$ and any relation among two or more such components is called a *law statement,* $l(X) \in L(X)$. *Thus,* $l(X): V_1 \otimes \cdots \otimes V_n \rightarrow$ {unlawful, lawful}.

Laws reflect either natural or artificial constraints imposed upon things.[3] For example: 1) the *QOH* state variable of the inventory item thing might be restricted to values that are zero or positive (a natural law); 2) the *Discount* state variable may be related to the *Unit-Price* state variable (an artificial law). Thus, laws provide information about things. As such, they are also properties of things. Accordingly, if we choose, laws can be modeled via the total state function used to describe a thing.

Since we postulate the existence of laws, we recognize that only a subset of the possible state space of a thing may be deemed lawful.

Definition 3:* Let $X_m = (M, \tilde{F}\rangle$ be a functional schema for a thing X, where $\tilde{F} = \langle F_1, \cdots, F_n\rangle : M \rightarrow V_1 \otimes \cdots \otimes V_n$ is the total state function, and let $L(X)$ be the set of all law statements on X. Then the subset of the codomain V restricted under $L(X)$ is called the *lawful* ***state*** space of X in the representation X_m. That is, $S_L(X) = \{\langle x_1, \cdots, x_n\rangle \in V_1 \otimes \cdots \otimes V_n \;\; \tilde{F}$ satisfies every $l(X) \in L(X)\}$.

Thus, the lawful state space of the inventory item thing would include all combinations of state variable values that we deem allowed.

C. Events, History, and Coupling

When a thing undergoes change, the value of at least one of its properties must alter. A change of state constitutes an ***event.***

Definition 4:* An ordered pair (s, $s'\rangle$, where $s, s' \in$ S(X), will be called an *event.*

However, not all changes of state are lawful, and so not all events are lawful. We define, first, the notion of a lawful transformation.

Definition 5:* Let S_L (X) be the lawful state space of a thing X. We denote by $G_L(X)$ the set of *transformations* from the lawful state space into itself that are given as ***lawful*** *in* the system. That is, $G_L(X) \subseteq S_L(X) \otimes S_L(X)$.

Hence we have the following.

Definition 6:* Let S,(X) be the lawful state space of a thing X, and let $G_L(X)$ be the set of lawful transformations on the state space into itself. Then a ***lawful*** *event* in X is represented by the ordered pair $\langle s, s'\rangle$, where $s, s' \in S_L(X)$ and $s' = g(s), g \in G_L(X)$.

Corollary 6:* The *lawful event space* of a thing X is the set of ordered pairs: $E_L = G_L(X) \cap [S_L(X)]^2$.

To illustrate these concepts, consider again our inventory item thing. Assume at some time the state functions for the inventory item thing map it into the following values: *Item-No* = *P1*; *QOH* = 10; *Unit-Price* = 15.00; *Discount* = 5. *Thus,* $s = (P1$, 10, 15.00, 5).

Assume, also, that one lawful transformation on this state updates the value of *QOH* in light of an inventory replenishment. For example, if 5 more units of inventory are received, the following lawful event occurs: $\langle s, s'\rangle = \langle (P1, 10, 15.00, 5), (P1, 15, 15.00, 5)\rangle$. Thus, a subset of the lawful event space for the inventory item thing includes all events that arise as a result of an inventory replenishment and the "firing" of the lawful inventory replenishment transformation.

Changes of state manifest a *history* of a thing. Thus we have the following.

Definition 7:* Let X be a thing modeled by a functional schema $X_m = (M, \tilde{F}\rangle$, let $t \in M, t > 0$ be a time instant. Then a ***history*** of X is the set of ordered pairs, h(X) = $\{\langle t, \tilde{F}(t)\rangle\}$.

In turn, the notion of a history allows us to determine when two things are bonded or coupled to each other. Intuitively, if two things are independent of each other, they will have independent histories. If they are coupled in some way, however, at least one of the things' history will depend upon the other thing's history. Thus we have Definitions 8 and 9.

Definition 8*: A thing X *acts on* a thing Y, denoted X $\sqsupset$ Y if $h(Y \mid X) \neq$ h(Y).

Definition 9:* Two things X and Y are ***coupled,*** denoted $B(X, Y)$, iff (X $\sqsupset$ Y) $\vee$ (Y $\sqsupset$ X).

To illustrate these notions, consider the inventory item and the customer order shown in Table I. If the customer order represents a sale of inventory to a customer, the states of the inventory item will depend upon the states of the customer order. Thus, the two things are coupled.

D. Systems and Subsystems

The notions of things and couplings enable us to define precisely the concept of a *system.* Intuitively, a system comprises a set of things where each thing in the set is coupled to at least one other thing in the set and where, in addition, it is impossible to partition the set of things such that the histories of the two partitions are independent of each other. Thus we have the following.

Definition 10: Let C be a set of things, and let $B_C = \{(X, Y) | X, Y \in C \wedge B(X, Y)\}$. Let $\sigma(C, B_C)$ be a graph, where C is the set of vertices (things) and B_C is the set of edges (couplings). Then $\sigma(C, B_C)$ is a *system* iff it is a connected graph. Henceforth, $\sigma(C, B_C)$ will be denoted by σ.

Fig. 1 is a graph of a system. Consider the various couplings that exist. First, the order thing is coupled to the inventory item thing and the customer account thing. When the order occurs, the value of ***Qty-Supl*** depends upon the value of *QOH.* The quantity ordered can only be supplied if there is sufficient inventory on hand. Furthermore, the maximum discount that the salesperson

[3] Laws correspond to the notion of invariants in Z.

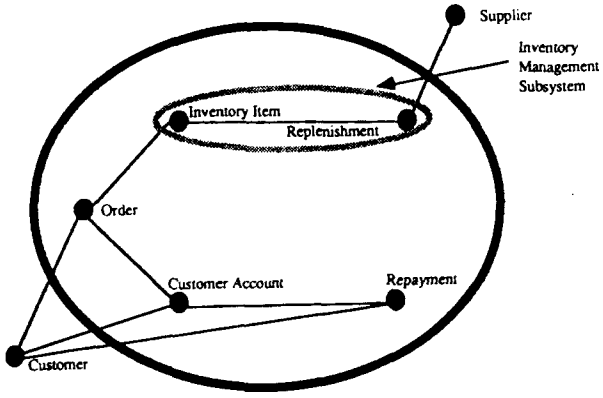

Fig. 1. Graph of a system.

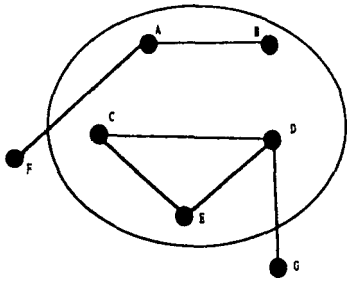

Fig. 2. Graph of an aggregate.

can give to a customer is given by *Discount*. Thus, *Sale-Pr* is a function of *Unit-Pr* and *Discount. The* order will be filled only if the sales amount is less than or equal to the customer's credit limit less their balance due. Thus, *Qty-Supl* is also a function of *Cr-Limit* and *Bal-Due*. Second, the inventory item thing is coupled to the inventory replenishment thing. When inventory replenishment occurs, the quantity-on-hand is updated with the value of the replenishment amount. Thus, *QOH* is a function of *Replen*. Third, the customer account thing is coupled to the customer repayment thing. When a repayment occurs, the value of balance due is decreased. *Thus, Bal-Due* is a function of *Amount*.

Thus, all things in Fig. 1 are coupled to at least one other thing. Moreover, it is possible to "walk" from one thing to another thing via the arcs. In Fig. 2, however, each thing is coupled to at least one other thing, but the set of things can be partitioned so the histories of *A* and *B* are independent of the histories of C, *D*, and *E*. Thus, Fig. 2 shows two disjoint systems. It is an *aggregate* but not a system.

Unfortunately, the transition to defining the concept of a subsystem is not straightforward. We must begin by defining the notions of the *composition, environment,* and *structure* of a system.

Definition 11:* Let σ be a system. Then:

a) the *composition* of σ at time t is the set of things in σ at t:

$$\tilde{C}(\sigma, t) = \{x \mid x \in \sigma\}$$

b) *the environment* of σ at time t is the set of things that are not components of σ but which act on or are acted upon by Components of σ at t:

$$\tilde{E}(\sigma, t) = \left\{ x \mid x \notin \tilde{C}(\sigma, t) \wedge (\exists y)\left(y \in \tilde{C}(\sigma, t) \wedge B(x, y)\right) \right\}$$

c) the *structure* of σ at time t is the set of couplings among the components of σ and among them and the set of components in the environment of σ at t:

$$\tilde{S}(\sigma, t) = \{R, \in \bar{B}(\sigma, t) \cup \hat{B}(\sigma, t)\}$$

where

$$\bar{B}(\sigma, t) = \left\{ B(x, y) \mid x, y \in \tilde{C}(\sigma, t) \right\}$$

$$\hat{B}(\sigma, t) = \left\{ B(x, y) \mid x \in \tilde{C}(\sigma, t) \wedge y \in \tilde{E}(\sigma, t) \right\}$$

To illustrate these notions, consider the Fig. 1 system. Its composition comprises five things: inventory item, customer order, customer account, customer repayment, and inventory replenishment. The environment of the system comprises two things: customer and supplier. Customer is coupled to order, customer account, and customer repayment because customers place orders, change addresses, and make repayments. Supplier is coupled to inventory replenishment because suppliers supply inventory. Finally, the structure of the system comprises a) the internal bondings $\bar{B}$, between order and inventory item, order and customer account, inventory item and replenishment, and customer account and repayment, and b) the external bondings $\hat{B}$, between supplier and replenishment, customer and order, customer and customer account, and customer and repayment.

Given the concepts of composition, environment, and structure, we now define a *subsystem* as a system whose composition and structure are subsets of another system and whose 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.

Definition 12:* Let σ be a system with composition $\tilde{C}(\sigma, t)$, environment $\tilde{E}(\sigma, t)$, and structure $\tilde{S}(\sigma, t)$ at time t. Then x is a *subsystem* of σ, denoted $x \prec \sigma$, iff:

a) x is a system at time t, and

b)

$$[\tilde{C}(x, t) \subseteq \tilde{C}(\sigma, t)]$$
$$\wedge \left[\tilde{E}(x, t) \subseteq \left\{\tilde{E}(\sigma, t) \cup \left\{\tilde{C}(\sigma, t) - \tilde{C}(x, t)\right\}\right\}\right]$$
$$\wedge \left[\tilde{S}(x, t) \subseteq \tilde{S}(\sigma, t)\right].$$

Notation 12: If x is a subsystem of σ, then σ is a *supersystem* of x.

To illustrate these notions, consider a system that comprises inventory item, replenishment, and their bonding. This system, which we might call the inventory management subsystem, is a subsystem of the Fig. 1 system. Its composition $\tilde{C}(x)$ = {inventory item, replenishment}, is a subset of the composition of the Fig. 1 system. Its environment, $\tilde{E}(x)$ = {supplier, order}, is a subset of the things that are in the environment of the Fig. 1 system plus those things that are in the composition of the Fig. 1 system but not in the composition of the inventory management subsystem. Its structure, $\tilde{S}(x)$ = {(inventory item, replenishment), (replenishment, supplier), (inventory item, order)}, is a subset of the structure of the Fig. 1 system.

E. Equilibrium[4]

Systems can be in stable or unstable states. Since both types of state are important from the viewpoint of examining system

[4] This section may be skipped by those readers who are primarily interested in our decomposition results in Section V.

dynamics, we develop the notion of a system equilibrium. We propose two types of equilibrium: a static equilibrium and a dynamic equilibrium. Each is simply a specific instance of a more general concept.

Definition 13: Let σ be a system, and let G' $\subseteq G(\sigma)$ be a subset of the set of transformations on the possible state space S(a). Furthermore, let S'(n) $\subseteq$ S(a) be a subset of the possible state space of the system. Then σ is in *equilibrium* in the region S'(a) with respect to the set of transformations G'(a) iff (Vs $\in$ S'(a)), (Vg $\in G'(\sigma)$), s' = g(s) and $s' \in S'(\sigma)$. Note, for some transformations, s' may equal s.

Corollary 13a: A state $s \in$ S(a) is *stable*[5] with respect to the set of transformations $G'(\sigma)$ iff (Vg $\in G'(\sigma)$), g(s) = s.

Corollary 13b: A state $s \in S(\sigma)$ is unstable with respect to the set of transformations G'(u) iff ($\exists g \in$ G'(u)), g(s) $\neq$ s.

Corollary 13c: A system σ is in a *static equilibrium* in the region S'(a) with respect to the set of transformations G'(a) iff (Vs $\in$ **S'(a)),** ($\forall g \in$ **G'(a)),** g(s) = s.

Corollary 13d: A system σ is in a *dynamic equilibrium* in the region S'(a) with respect to the set of transformations G'(a) iff it is in equilibrium and (3s $\in$ S'(a)), ($\exists g \in$ G'(a)), such that $g(s) \neq s$.

To illustrate the notion of a *static equilibrium,* consider again the Fig. 1 system. Assume we focus on the transformation that updates *QOH* for the inventory item in light of a new replenishment from a supplier. (Below we call this "the replenishment transformation.") When a new inventory replenishment is received, the state of inventory replenishment will change. In particular, the value of *Proc* (Processed Flag) will indicate the replenishment has not been processed. This new state is unstable because a transformation exists that will change it. The transformation will update the value of *QOH* and set the value of *Proc* to indicate the replenishment has been processed. Once the replenishment has been processed, the system will once again be in a static equilibrium. Indeed, with respect to the replenishment transformation, the system is in a *stable state.*

To illustrate the notion of a *dynamic equilibrium,* consider a situation where several orders are received from customers for the inventory item. As the orders deplete the quantity-on-hand, the inventory item moves through a set of states. Providing the set of states are included within some *a priori* defined set of states that for some reason are of interest, the system is in a dynamic equilibrium. However, once a state arises that is not within the *a priori* defined set, the system is no longer in equilibrium. For example, an order may deplete the quantity-on-hand to zero so that customer orders can no longer be filled. If the inventory item state where the quantity-on-hand is zero is not a member of the *a priori* defined set of states, the system is in disequilibrium.

There are two reasons why a system may move from either a static equilibrium or a dynamic equilibrium. First, an environmental component may act upon a system component to change its state. Second, a transformation exists that is not within the subset of transformations being considered.

Historically, the tendency of systems to move toward a static equilibrium because of the action of transformations on unstable states has been called homeostasis [23]. In this light, note that unstable states are not necessarily unlawful. Lawfulness and stability are two different properties of a system state (Fig. 3).

[5] Stable states correspond to the notion of fixed pints described in [7, p. 10].

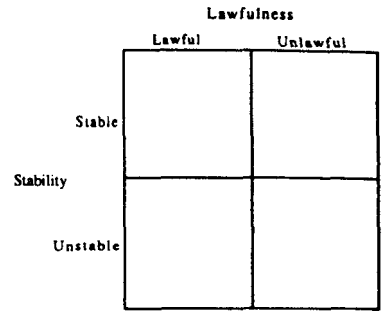

Fig. 3. Lawfulness and stability as disjoint properties of states.

F. Inputs, External Events, and Internal Events

Systems are usually of interest because they accept inputs and transform them into outputs. To formalize these notions, we begin with the notion of an *input component.*

Definition 14: Let $x \in \tilde{C}(\sigma)$. Then x is an *input component* of σ iff $\exists y$, y $\in \tilde{E}(\sigma)$ such that y D x.

In other words, a thing is an input component of a system if it is acted upon by an environmental thing. For example, in Fig. 1, order is an input component because customer can act upon order to change its state.

Input components must be distinguished from input *states.* First we have Definition 15.

Definition 15:* Let x and y be two things such that x acts on y. Then the *total action* of x on y is $A(x,y) = h(y|x) - h(y)$.

The total action of one thing on another thing is the set of states indexed by time that arise with the latter by virtue of the existence of the former. This concept leads to the notions of the *totality of input* for an input component and a system and an *input state* of a system.

Definition 16: Let x be an input component of σ. Then the *totality of input* of x is the set of environmental actions on x: $U(x) = \bigcup_{y \in \tilde{E}(\sigma)} A(y,x)$.

Corollary 16: The totality of input of σ is the set of all environmental actions on all input components of σ: U(b) = $\bigcup_{x \in \tilde{C}(\sigma)} U(x)$ where x is an input component of σ.

Definition 17: $S_I(\sigma) = \{\tilde{F}(t) \mid <t, \tilde{F}(t)> \in U(a)\}$ is the *set of input states* of a system σ. $s \in S_I(\sigma)$ is an *input state.*

The concept of an input state leads to the notion of an *external event* and an *external transformation.*

Definition 18: An event $<s,s'>$ is an *external event* iff $s' \in$ S,(u).

Corollary 18: A transformation g is an *external transformation* iff s' = g(s) and s' $\in S_I(\sigma)$.

To illustrate these concepts, the totality of input of the order thing in Fig. 1 is the set of states of the order thing indexed by time that arise because the customer places an order. The totality of input of the system is the (set) union of the totality of input of the order thing, the totality of input of the repayment thing, and the totality of input of the replenishment thing. An input state of the order thing arises by virtue of the action of a customer. When an input component changes state because of the action of an environmental component on it, the resulting event is called an external (input) event. For example, when the state of the *Sales-Amt* state variable of the order component changes from, say, zero dollars to $200 to reflect a customer has placed an order, this event is an external event. The transformation that evokes an external event is an external transformation.

Finally we have the notions of an internal event and an internal transformation.

Definition 19: An event $<s, s'>$ is an *internal event* iff $s' \notin S_I(\sigma)$.

Corollary 19: A transformation g is an *internal transformation* iff $s' = g(s)$ and $s' \notin$ S,(a).

Internal events arise as a result of external events. For example, in Fig. 1 a customer may place an order. The state of the system changes to reflect the external event that has occurred. If the resulting state is unstable, an internal event (or sequence of internal events) then occurs to restore the system to stability. For example, as a result of the order, inventory is depleted and the customer's account is updated. Internal transformations effect these events.

G. Outputs and Transfer Functions'

Output concepts are similar to input concepts. We begin with the notions of an *output component* and the *totality of output.*

Definition 20: Let $x \in \tilde{C}(\sigma)$. Then x is an *output component* of σ iff $\exists y$, y $\in \tilde{E}(\sigma)$ such that x D y.

In Fig. 1, order is both an input component (as discussed above) and an output component. It is an output component because the confirmation or rejection of an order affects the state of customer (e.g., by altering the value of a property of the customer such as amount of cash they have in the bank).

Definition 21: Let x be an output component of σ. Then *the totality of output* of x is the set of all actions of x on the environment of IT: $V(x) = \bigcup_{y \in \tilde{E}(\sigma)} A(x, y)$.

Corollary 21: The totality of output of σ is the set of all actions of output components of σ on the environment of σ: $V(\sigma) = \bigcup_{x \in \tilde{C}(\sigma)}$ V(x).

The totality of output of the order thing is the set of states of the customer thing that arise because orders are confirmed or rejected. Since we assume no other system output, the totality of output of order is also the totality of output for the system.

The concepts of input and output give rise to the notion of a system *transfer function.*

Definition 22:* Let σ be a system, and let $U(\sigma) \neq \emptyset$ and $V(\sigma) \neq 0$. Then the function f that maps the totality of inputs of σ to the totality of outputs of σ is called the *transfer* (or *transducer*) *function* of σ. That is, $f: U(\sigma) \rightarrow V(n)$.

In short, the transfer function of a system maps the totality of inputs of the system to the totality of outputs of the system.' It manifests the "processing" that the system undertakes.

IV. On the Nature of and Some Dynamics of System Decompositions

In this section we develop the notion of a system decomposition. In addition, we examine some dynamics of system decompositions. Our purpose is twofold. First, we seek to show

[6] As with Section III-E, this section may be skipped by those readers who are Primarily interested in our decomposition results in Section V.

[7] Whether the transfer function is a single-valued function or a multivalued function (relation) depends upon how well the system has been "carved out" from its environment. A system state may map into two or more subsequent states because the domain of the transformation includes values of slate variables for components in the environment of the system. In Section V we argue that internal transformations in a well-decomposed system will depend only on the values of state variables for components in the composition of the system. Indeed, drawing the boundary of the system can be viewed as a decomposition of a larger system (composed of the environment and the system), similar to breaking the system itself up into subsystems.

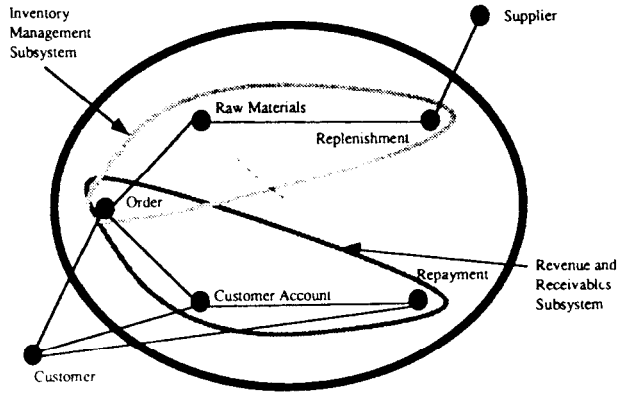

Fig. 4. Graph of a decomposition of a system

the power of our basic concepts for building more complex notions. Second, we lay the groundwork for examining the nature of a "good" decomposition.

We begin with the notion of a decomposition, which we define as a set of subsystems of a system where a) every element in the composition of the system is included in 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) each element in the structure of the system is included in at least one of the subsystems in the set. Thus, we have the following.

Definition 23: Let I be an index set, and let $D(\sigma) = \{x_i\}_{i \in I}$ where $x_i \prec \sigma$. Then *D(o)* is a *decomposition* over σ iff $\tilde{C}(\sigma) = \cup_{i \in I} \tilde{C}(x_i)$.

Corollary 23a: $\tilde{E}(\sigma) = \cup_{i \in I} \tilde{E}(x_i) - \tilde{C}(\sigma)$.

Corollary 236: $\tilde{S}(\sigma) = \cup_{i \in I} \tilde{S}(x_i)$.

To illustrate the notion of a system decomposition, consider Fig. 4. The Fig. 1 system has been decomposed into two subsystems: an inventory management subsystem and a revenue and receivables subsystem. It is a straightforward exercise to show that these two subsystems constitute a decomposition of the Fig. 1 system.

The concept of a decomposition leads naturally into the notion of a *level structure* over a system. A level structure formalizes the idea that sets of subsystems are "nested" within particular systems which in turn are nested within other systems [29]. Thus we have the following.

Definition 24: Let Σ be a set of systems. Let L be a partition of Σ: $L = \{L^i \mid i = 1, \cdots, n\}$ with $n > 1$. Then L will be called a *level structure* iff $(\forall i > 1)$, $(\forall x)[x \in L^i \Rightarrow \exists y \in L^{i-1}$ Ax $\prec y]$.

Definition 25: Let $D(\sigma)$ be a decomposition of a system σ. D(a) will be termed a level structure of σ iff a partition L of $D(\sigma)$ exists that is a level structure.

Corollary 25a: For every decomposition D(a) of a system σ, $\{\sigma\} \cup D(\sigma)$ is a level structure.

Corollary 256: If $D(\sigma)$ is a level structure of σ, then $\{\sigma\}$ U $D(\sigma)$ is also a level structure.

The top panel of Fig. 5 shows the level structure for the system decomposition shown in Fig. 4. (The bottom panel has been added to show the composition of each subsystem.) The decomposition comprises three (sub)systems: the accounting system itself, and the inventory management and revenue and receivables subsystems. The level structure comprises two partitions: one, L^1, contains the accounting system itself; and the other, L^2, contains the two subsystems. For every system in the second partition $(i > 1)$, note that each is a subsystem of some system in the higher-level partition.

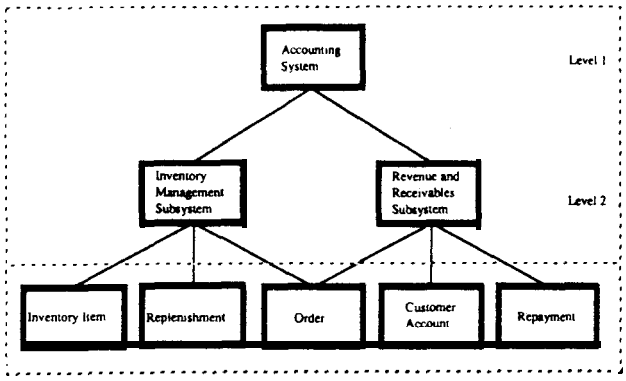

Fig. 5. Level structure for first decomposition.

When a system changes its state, we know that at least one subsystem in the decomposition of the system must have changed its state. Indeed, we observe changes at the system level because changes have occurred at the subsystem level. Thus, state changes can be viewed as propagating upwards in the level structure. Alternatively, we can take the view that changes at the system level are reflected in some subsystems-in short, system-level changes induce changes in subsystems. It is this propagation, or induction, of changes in the system decomposition that manifest the dynamics of the system. The dynamics of a decomposition can be studied, therefore, in terms of how state changes in the system decompose into state changes in the subsystems of the system. Ultimately we evaluate whether a decomposition is "good" via the characteristics of its dynamics.

We begin with the notion of the *state space of a decomposition* of a system.

Definition 26: Let I be an index set and D(a) be a decomposition over 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 26a: Henceforth, $S(D(\sigma))$ will be abbreviated to $S(D)$.

Notation 266: s_i^j denotes the jth state of the ith system; s_0^k denotes the *k*th state of the system.

To illustrate this concept, consider, again, the accounting system illustrated in Figs. 4 and 5. To simplify matters, assume we use only one state variable to describe the state of a subsystem: *Inv* (dollar value of inventory) for the inventory management subsystem: and *Rec* (dollar value of receivables) for the revenue and receivables subsystem. Furthermore, assume the inventory management subsystem can take on only three dollar value states, $S(x_1) = \{s_1^1, s_1^2, s_1^3\} = \{0, 100, 200\}$, and the revenue and receivables subsystem can take on only four dollar value states, $S(x_2) = \{s_2^1, s_2^2, s_2^3, s_2^4\} = \{0, 100, 200, 300)$. Under these assumptions, the possible state space of the decomposition of the accounting system is:

$$\begin{aligned} S(D) = \{&(s_1^1, s_2^1), (s_1^1, s_2^2), (s_1^1, s_2^3), (s_1^1, s_2^4), (s_1^2, s_2^1), (s_1^2, s_2^2), \\ &(s_1^2, s_2^3), (s_1^2, s_2^4), (s_1^3, s_2^1), (s_1^3, s_2^2), (s_1^3, s_2^3), (s_1^3, s_2^4)\} \\ = \{&(0,0), (0,100), (0,200), (0,300), (100,0), \\ &(100,100), (100,200), (100,300), (200,0), \\ &(200,100), (200,200), (200,300)\}. \end{aligned}$$

A fundamental relationship can now be articulated between the possible state space of a system and the possible state space of a decomposition of the system. Specifically, we have the following straightforward but important result.

Lemma 1: Let S(a) be the possible state space of a system σ, and let *S(D)* be the possible state space of a decomposition over σ. Then for every state $s \in S(\sigma)$, there exists at least one state $\delta \in S(D)$.

Corollary: There exists a mapping from S(a) into the power set of $S(D)$, $P(S(D))$.

Notation 26c: Let $s \in S(\sigma)$, $d(s) \equiv \{\delta \mid \delta \in S(D)$ corresponds to $s \in$ S(o)}. d(s) $\in P(S(D))$.

To illustrate Lemma 1, assume we represent the state of the accounting system x_0 using a single state variable: *Asset* (total dollar value of inventory plus receivable assets). Since the state of the accounting system, s_0^j, is a simple function of the state of its two subsystems, we have

$$\begin{aligned}
d(s_0^1) &= \{(s_1^1, s_2^1)\}, && \text{i.e., } d(0) = \{(0.\ 0)\} \\
d(s_0^2) &= \{(s_1^1, s_2^2), (s_1^2, s_2^1)\}, && \text{i.e., } d(100) = \{(0,100), (100,0)\} \\
d(s_0^3) &= \{(s_1^1, s_2^3), (s_1^2, s_2^2), && \text{i.e., } d(200) = \{(0,200), (100,100), \\
&\quad (s_1^3, s_2^1)\}, && \qquad (200,0)\} \\
d(s_0^4) &= \{(s_1^1, s_2^4), (s_1^2, s_2^3), && \text{i.e., } d(300) = \{(0,300), (100,200), \\
&\quad (s_1^3, s_2^2)\}, && \qquad (200.100)\} \\
d(s_0^5) &= \{(s_1^2, s_2^4), (s_1^3, s_2^3)\}, && \text{i.e., } d(400) = \{(100,300), (200,200)\} \\
d(s_0^6) &= \{(s_1^3, s_2^4)\}, && \text{i.e., } d(500) = \{(200,300)\}.
\end{aligned}$$

In general, because we may choose different variables to represent the state of a system and the state of a subsystem, we cannot say the possible state space of the system is *equal to* the possible state space of a decomposition of the system. Nonetheless, we postulate that the possible state space of a system can always be *mapped* into the possible state space of a decomposition of the system in a meaningful way.

In this light, we can now show how events in a system reflect events in its subsystems. To assist our exposition, we introduce, first, some additional notation and a lemma.

Notation 266: Let $\delta \in$ d(s), δ is a state of the decomposition. Hence it has a component in each of the subsystems. The ith component of δ, δ_i will be termed a *projection* of state s in subsystem x_i. Alternatively, let $\delta = d^j(s)$ be the jth element of the set d(s). The projection 6, will be denoted by d_i^j. *The* set of projections $\{\delta_i\}$ of a state s on subsystem x_i will be denoted by $d_i(s)$.

Lemma 2: Let $s' \neq$ s" be two different states of σ. Then $d(s') \cap d(s'') = 0$. That is, for every 6' $\in d(s')$ and $\delta'' \in d(s'')$, at least one subsystem x_i exists such that $\delta_i' \neq \delta_i''$.

To illustrate the notation in the above example, let $s' = s_0^2$ and $s'' = s_0^4$. Thus, $d(s') = \{(s_1^1, s_2^2), (s_1^2, s_2^1)\}$ and $d(s'') = \{(s_1^1, s_2^4), (s_1^2, s_2^3), (s_1^3, s_2^2)\}$. It can be seen that none of the combinations for s_0^2 equals any of the combinations for s_0^4.

Using this foundation, we now propose the notion of an *induced event.* Intuitively, an induced event is an event that occurs in a subsystem which reflects that an event has occurred in the system. The notion of an induced event reflects that we can view a change in the state of the system at the level of the system or at some level of subsystems when the subsystems are organized in a level structure. Nonetheless, a change of state in the system must still be manifested as a change of state in at least one of its subsystems. Thus we have the following.

Definition 27: Let $\langle s, s' \rangle \in E(n)$ be an event in the possible event space of the system, and let $d^j(s)$ and $d^k(s')$ be corresponding states in S(D) (j and k are not necessarily distinct). Let $d_i^j(s)$ and $d_i^k(s')$ be the state of the ith subsystem when the system is in states s and s', respectively. Then the pair $\langle d_i^j(s), d_i^k(s') \rangle$ will be called an ***induced event*** on subsystem $x_i \prec \sigma$ 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, consider our accounting system. Assume the event $\langle s_0^3, s_0^5 \rangle$ occurs; that is, total assets change from \$200 to \$400. Assume $d^3(s_0^3)$ and $d^2(s_0^5)$ are the corresponding states in S(D). In our example above, $d^3(s_0^3) = \{(s_1^3, s_2^1)\} = \{(200, 0)\}$ and $d^2(s_0^5) = \{(s_1^3, s_2^3)\} = ((200,200))$. Thus: $d_1^3(s_0^3) = s_1^3 = 200$; $d_2^3(s_0^3) = s_2^1 = 0$; $d_1^2(s_0^5) = s_1^3 = 200$; $d_2^2(s_0^5) = s_2^3 = 200$. Since $d_1^3(s_0^3) = d_1^2(s_0^5)$, no induced event has occurred in the inventory management subsystem. However, note that $d_2^3(s_0^3) \neq d_2^2(s_0^5)$. Thus, in the revenue and receivables subsystem, an induced event has occurred in light of the change of system state. Consequently, we conclude the dollar value of assets has risen by \$200, not because the value of inventory has risen, but because the value of receivables has increased.

Not every change of state in a subsystem will be manifested as a change of state in the system. Since a system state may map into two or more subsystem states-that is, the cardinality of $d(s)$ may be greater than one-a change of subsystem states may not be manifested as a change of system states. For example, assume the following event occurs in the decomposition: $\langle (s_1^1, s_2^3), (s_1^2, s_2^2) \rangle$. Note that an induced event has occurred in both subsystems. However, since $s_0^3 = 200$ maps into both states of the decomposition, the change of subsystem states will not be manifested as a change of system state. If a change of state does occur in the system, however, we have the following lemma.

Lemma 3: For every event in the system, at least one subsystem must exist in which an induced event occurs.

V. On Good Decompositions

In Section IV we defined the concept of a decomposition of a system. In the CS and IS fields, this concept serves two purposes. First, it is used as a basis for understanding complex systems [10]. For example, a number of systems analysis methodologies "factor" processes and data into lower-level processes and data so the processes and data can be better understood [45]. Second, decomposition is used as a basis for *design* [16], [17], [33], [47]. For example, the so-called structured design methodologies rely on the concept of decomposition for deriving program modules, Which in turn form the basis for system implementation in some programming language [46]. Similarly, relational da&abase management theory uses a form of decomposition to identify relations that are relatively independent of one another [8].

Irrespective of whether the purpose of decomposition is to understand or to design systems, two fundamental questions must be addressed. First, what is the nature of a good decomposition? Second, how do we achieve good decompositions? While many heuristics have been proposed to address both questions [2], [12], [16]–[20], [28], [32], [35], [46], [47], no general theory of decomposition has been developed that enables us to evaluate whether these heuristics do, in fact, produce good decompositions.

Accordingly, to show the power of our ontological model, we now use it to define precisely the characteristics of a particular type of decomposition that we hypothesize will allow analysts, designers, and programmers to better understand systems. In using our model for this purpose, we point out four important issues that underlie our analysis. First, whether the concept of decomposition we propose leads to a better understanding of systems is an untested empirical issue. The psychological theory needed to support our hypotheses is still poorly developed, and the empirical research needed to test our predictions has not yet been undertaken. Second, different types of decomposition may be required to accomplish other purposes when designing and implementing systems.' For example, the type of decomposition needed to achieve efficient execution of systems on parallel machine architectures may be at odds with the type of decomposition needed to facilitate understanding of systems [7]. Third, while our model allows existing decompositions to be evaluated to determine whether they are good, at this time it provides only limited insights on the matter of how good decompositions should be generated in the first place. Indeed, it is a moot point whether a complete set of prescriptive rules for generating good decompositions can ever be generated or whether the development of good decompositions, like the development of good theoretical models, will inevitably rely on intuition and experience. Finally, the primary contribution of our model is that it allows a ***precise definition*** of our notion of a good decomposition to be articulated. At first glance this contribution may seem modest. However, we are unaware of any other work in the CS and IS fields where the notion of a good decomposition has been defined precisely.

A. Some Characteristics of a Good Decomposition

It is generally believed that system decompositions which have "loosely-coupled" subsystems are easier to understand than system decompositions which have "tightly-coupled" subsystems (see, e.g., [46]). The rationale is that human problem solving is constrained by cognitive limitations; thus, things are easier to understand if they can be considered relatively independently of other things. Unfortunately, the meaning of "loose coupling" has remained somewhat obscure.

In this light, we use our model to make the notion of relatively-independent (loosely-coupled) subsystems precise. In summary, our model shows that subsystems are loosely coupled if the events in the subsystem that arise as a result of inputs to the subsystem can be defined independently of the states of other subsystems in the decomposition. In addition, the model shows that ***decompositions are good only with respect to a certain set of transformations on the system.*** The designer's task is to identify the relevant set of transformations to consider during the decomposition process.

We begin, then, with a definition that will aid our analysis.

Definition 28: Let a be a state of subsystem x_i. Then the set of system states that map into a will be denoted by $S_i(a) = \{s \mid a \in d_i(s)\}$. $S_i(\alpha)$ will be called the ***subsystem equivalence states*** with respect to a.

Notation 28: Let $\alpha = s_i^k$, the kth state of subsystem x_i. $S_i^k = S_i(s_i^k)$.

To illustrate this concept, consider our previous accounting system example. Recall the system is decomposed into only two subsystems: the inventory management subsystem x_1 and the revenue and receivables subsystem x_2. Given the six states the system can assume, the sets of subsystem equivalence states are:

$$S_1^1 = \{s_0^1, s_0^2, s_0^3, s_0^4\};$$

$$d_1^1(s_0^1) = d_1^1(s_0^2) = d_1^1(s_0^3) = d_1^1(s_0^4) = \$0$$

$$S_1^2 = \{s_0^2, s_0^3, s_0^4, s_0^5\};$$

$$d_1^2(s_0^2) = d_1^2(s_0^3) = d_1^2(s_0^4) = d_1^1(s_0^5) = \$100$$

$$S_1^3 = \{s_0^3, s_0^4, s_0^5, s_0^6\};$$

$$d_1^3(s_0^3) = d_1^3(s_0^4) = d_1^2(s_0^5) = d_1^1(s_0^6) = \$200$$

$$S_2^1 = \{s_0^1, s_0^2, s_0^3\};\quad d_2^1(s_0^1) = d_2^2(s_0^2) = d_2^3(s_0^3) = \$\ 0$$

$$S_2^2 = \{s_0^2, s_0^3, s_0^4\};\quad d_2^1(s_0^2) = d_2^2(s_0^3) = d_2^3(s_0^4) = \$100$$

$$S_2^3 = \{s_0^3, s_0^4, s_0^5\};\quad d_2^1(s_0^3) = d_2^2(s_0^4) = d_2^2(s_0^5) = \$200$$

$$S_2^4 = \{s_0^4, s_0^5, s_0^6\};\quad d_2^1(s_0^4) = d_2^1(s_0^5) = d_2^1(s_0^6) = \$300.$$

In light of the concept of the set of subsystem equivalence states, *we now* define the ***decomposition condition.***

Definition 29: Let g be a transformation on a system, and let S_i^k be a set of subsystem equivalence states. The ***decomposition condition*** holds for transformation g and subsystem x_i if, for every s_i^k of x_i a state s_i^h of x_i exists such that: $s \in S_i^k \Rightarrow g(s) \in S_i^h$, k and h are not necessarily distinct (i.e., h is a function of k and g and i but not of s).

Corollary 29: If the decomposition condition holds, then d!(a) $= d_i^m(s') \Rightarrow d_i(g(s)) \cap d_i(g(s')) \neq 0.$

In other words, for the set of all system states that map into the same ith subsystem state, the decomposition condition requires that the set of new system states which arise from an event will also map into the same ith subsystem state. In short, for the decomposition condition to hold, subsystems must "behave" independently in the following sense. If we know that the ith subsystem is in state $y = d_i^k(s)$ when the system is in state s, we have sufficient knowledge to predict what the new state $z = d_i^l(g(s))$ will be.

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

Definition 30: Let g be a transformation on a system σ. Then the transformation g_i induced on the subsystem $x_i \prec \sigma$ is ***well-defined*** iff g_i is a function, i.e., g,(s) = s' $\wedge\ g_i(s) = s'' \Rightarrow s' =$ s" for every $s, s', s'' \in S(x_i)$.

Definition 31: 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 ***internal*** transformation g_i in every subsystem x_i of the decomposition.

Recall from Definitions 18 and 19 that we have identified two types of events that can occur in a system: an external (input) event and an internal event. Since external events in a system arise from the actions of the environment, they may not be described by a well-defined transformation. For example, given a certain level of inventory, it may be impossible to predict the new level of inventory that arises after an order has been satisfied because the amount ordered cannot be predicted. Once the external event has occurred, however, we require that the subsequent internal events that occur in each subsystem be well defined if the decomposition is to be good. In summary, although external events in a system may not be well defined, all internal events in a well-decomposed system must be well-defined.

B. An Example

To illustrate the notion of a good decomposition, consider once more our accounting system example. Assume again that we represent the state of the inventory management subsystem via the state variable *Inv* and the state of the revenue and receivables system via the state variable ***Rec.*** However, assume now that we represent the state of the system via the state vector (*Inv*, *Rec*). Clearly, a mapping ***H*** from the state of the system to the state of the decomposition is trivial:

$$H(s_0^1) = H(\langle s_1^1, s_2^1\rangle) = \{\langle s_1^1, s_2^1\rangle\} = \{\langle 0,0\rangle\}$$
$$H(s_0^2) = H(\langle s_1^1, s_2^2\rangle) = \{\langle s_1^1, s_2^2\rangle\} = ((0,100))$$
$$H(s_0^3) = H(\langle s_1^1, s_2^3\rangle) = \{\langle s_1^1, s_2^3\rangle\} = ((0,200))$$
$$H(s_0^4) = H(\langle s_1^1, s_2^4\rangle) = \{\langle s_1^1, s_2^4\rangle\} = \{\langle 0,300\rangle\}$$
$$H(s_0^5) = H(\langle s_1^2, s_2^1\rangle) = \{\langle s_1^2, s_2^1\rangle\} = \{\langle 100,0\rangle\}$$
$$H(s_0^6) = H(\langle s_1^2, s_2^2\rangle) = \{\langle s_1^2, s_2^2\rangle\} = \{\langle 100,100\rangle\}$$
$$H(s_0^7) = H(\langle s_1^2, s_2^3\rangle) = \{\langle s_1^2, s_2^3\rangle\} = \{\langle 100,200\rangle\}$$
$$H(s_0^8) = H(\langle s_1^2, s_2^4\rangle) = \{\langle s_1^2, s_2^4\rangle\} = \{\langle 100,300\rangle\}$$
$$H(s_0^9) = H(\langle s_1^3, s_2^1\rangle) = \{\langle s_1^3, s_2^1\rangle\} = \{\langle 200,0\rangle\}$$
$$H(s_0^{10}) = H(\langle s_1^3, s_2^2\rangle) = \{\langle s_1^3, s_2^2\rangle\} = \{\langle 200,100\rangle\}$$
$$H(s_0^{11}) = H(\langle s_1^3, s_2^3\rangle) = \{\langle s_1^3, s_2^3\rangle\} = \{\langle 200, 200\rangle\}$$
$$H(s_0^{12}) = H(\langle s_1^3, s_2^4\rangle) = \{\langle s_1^3, s_2^4\rangle\} = ((200,300)).$$

The set of subsystem equivalence states are

$$S_1^1 = \{s_0^1, s_0^2, s_0^3, s_0^4\};$$
$$d_1^1(s_0^1) = d_1^1(s_0^2) = d_1^1(s_0^3) = d_1^1(s_0^4) = \$\ 0$$
$$S_1^2 = \{s_0^5, s_0^6, s_0^7, s_0^8\};$$
$$d_1^1(s_0^5) = d_1^1(s_0^6) = d_1^1(s_0^7) = d_1^1(s_0^8) = \$100$$
$$S_1^3 = \{s_0^9, s_0^{10}, s_0^{11}, s_0^{12}\};$$
$$d_1^1(s_0^9) = d_1^1(s_0^{10}) = d_1^1(s_0^{11}) = d_1^1(s_0^{12}) = \$200$$
$$S_2^1 = \{s_0^1, s_0^5, s_0^9\};\quad d_2^1(s_0^1) = d_2^1(s_0^5) = d_2^1(s_0^9) = \$\ 0$$
$$S_2^2 = \{s_0^2, s_0^6, s_0^{10}\};\quad d_2^1(s_0^2) = d_2^1(s_0^6) = d_2^1(s_0^{10}) = \$100$$
$$s; = \{s_0^3, s_0^7, s_0^{11}\};\quad d_2^1(s_0^3) = d_2^1(s_0^7) = d_2^1(s_0^{11}) = \$200$$
$$S_2^4 = \{s_0^4, s_0^8, s_0^{12}\};\quad d_2^1(s_0^4) = d_2^1(s_0^8) = d_2^1(s_0^{12}) = \$300.$$

Note, the set of decomposition states into which each system state maps comprises only one element, i.e., $|d(s)| = 1$. Thus, there is a one-one mapping (injection) between a system state and a state of the decomposition.

Assume, now, that the organization which operates the accounting system has only one customer and that the customer always orders inventory in \$100 lots as either a cash order or a credit order. Replenishment of inventory occurs on a regular time schedule as the organization produces inventory at its manufacturing facility. From time to time, stockout situations may occur if the customer orders more inventory than normal during a time period.

Assume, first, that the organization has a policy of generating a rush production order if a stockout situation occurs because it is concerned about losing goodwill if it cannot meet a future order from the customer. Rush production orders occur in \$200 lots. A transformation g^1 which effects an inventory replenishment in light of a rush production order when the inventory level reaches zero can be represented as follows:

$$g^1(\langle 0,.\rangle) = \langle 200,.\rangle; g^1(\langle 100,.\rangle) = \langle 100,.\rangle;$$
$$g^1(\langle 200,.\rangle) = (200,\ .\rangle.$$

The transformation g^1 induces the following internal transformations on x_1, the inventory management subsystem, and x_2, the revenue and receivables subsystem?

$$g_1^1(0) = 200 \qquad g_2^1(0) = 0$$
$$g_1^1(100) = 100 \qquad g_2^1(100) = 100$$
$$g_1^1(200) = 200 \qquad g_2^1(200) = 200$$
$$g_2^1(300) = 300.$$

Note that a well-defined internal transformation has been induced in each subsystem. Thus, the decomposition is a good decomposition under the inventory replenishment transformation.

Assume now that the organization changes its policy with respect to stockout situations. It will generate a production order for stock replenishment purposes only if the credit limit of $200 has not been reached. It is not willing to bear the extra costs associated with a rush production order if its customer is in debt at or beyond the credit limit amount. Consider a transformation g^2 which effects the inventory replenishment under this policy:

$$g^2(\langle 0, 0)) = (200, 0), \text{ i.e., } g^2(s_0^1) = s_0^9$$
$$g^2(\langle 0.100)) = (200, 100\rangle, \text{ i.e., } g^2(s_0^2) = s_0^{10}$$
$$g^2(\langle 0,200)) = \langle 200,200\rangle, \text{ i.e., } g^2(s_0^3) = s_0^1$$
$$g^2(\langle 0,300)) = (0,300\rangle, \text{ i.e., } g^2(s_0^4) = s_0^4$$
$$g^2(\langle 100, 0)) = (100, 0), \text{ i.e., } g^2(s_0^5) = s_0^5$$
$$g^2(\langle 100,100\rangle) = (100, 100\rangle, \text{ i.e., } g^2(s_0^6) = s_0^6$$
$$g^2(\langle 100, 200)) = \langle 100,200\rangle, \text{ i.e., } g^2(s_0^7) = s_0^7$$
$$g^2(\langle 100,300\rangle) = \langle 100,300\rangle, \text{ i.e., } g^2(s_0^8) = s_0^8$$
$$g^2(\langle 200, 0)) = (200, 0), \text{ i.e., } g^2(s_0^9) = s_0^9$$
$$g^2(\langle 200,100\rangle) = (200, 100\rangle, \text{ i.e., } g^2(s_0^{10}) = s_0^{10}$$
$$g^2(\langle 200, 200)) = \langle 200,200\rangle, \text{ i.e., } g^2(s_0^{11}) = s_0^{11}$$
$$g^2(\langle 200,300\rangle) = \langle 200,300\rangle, \text{ i.e., } g^2(s_0^{12}) = s_0^{12}.$$

The transformation g^2 induces the following *internal* transformations on the two subsystems:

$$g_1^2(0) = \{0,200\} \qquad g_2^2(0) = 0$$
$$g_1^2(100) = 100 \qquad g_2^2(100) = 100$$
$$g_1^2(200) = 200 \qquad g_2^2(200) = 200$$
$$g_2^2(300) = 300.$$

Note, the induced internal transformation on the inventory management subsystem is not well defined. Moreover, it is easy to show that the decomposition condition has been violated; specifically, $s_0^1 \in S_1^1$ and $s_0^4 \in S_1^1$ but $g^2(s_0^1) = s_0^9 \in S_1^3$ and $g^2(s_0^4) = s_0^4 \in S_1^1$. Thus, the inventory management subsystem does not behave independently because it is not always possible to predict its new state given its current state. In short, the decomposition is not good with respect to the replenishment transformation under the revised policy.

To conclude, our model shows the "goodness" of a decomposition must be evaluated with respect to a set of transformations. The designer's task is to choose the "relevant" set of transformations and to devise a decomposition that is good with respect to each transformation in the set. We have not shown how this task can be undertaken in this paper, but the answer lies in identifying which couplings are important under each transformation that is of interest. This issue has been addressed elsewhere [31], [41].

[8] In the interests of simplicity, we have not considered the external events that might have given rise to the internal events. Assume, however, that they are customer orders that deplete the inventory when they are satisfied.

VI. Research Directions and Conclusions

As we indicated in the introduction to this paper, two major tests of the quality of a model are its ability to facilitate understanding and to aid prediction. In terms of understanding, we believe our model clarifies some previously fuzzy but fundamental constructs in the CS and IS fields. It also shows rigorously how these constructs are related to one another. In addition, elsewhere we believe we have used the model successfully to analyze the meaning of concepts like batch *systems*, real-time systems, system controls, data, programs, abstract data types, and objects [37]–[40]. An important attribute of these analyses is that they have been independent of implementation or technology considerations. Thus, we believe they will prove to be robust in the long run.

In terms of prediction, we have used the model to forecast the locus of control and audit procedure change when an information system is modified [40] and to identify some necessary attributes of a good requirements specification [39], [42]. In light of the analysis of the characteristics of a good decomposition that we have undertaken in this paper, we believe the model can now be used to make predictions about the efficacy of the various decomposition rules and heuristics that have been proposed in the literature. For example, we can address the question of how well functional decomposition, data flow decomposition, and data structure decomposition meet the requirements of a good decomposition that have been articulated in this paper. Our current research is directed toward these issues.

Acknowledgment

We are indebted to P. Bailes and A. Street for helpful comments on this paper. We are especially grateful to G. Dromey and three reviewers for their detailed comments and to K. Raymond for helpful discussions on the Z language.

References

[1] C. J. W. Alexander, *Notes on the Synthesis of Form.* Cambridge, MA: Harvard University Press, 1964.

[2] G. D. Bergland, "A guided tour of program design methodologies," *Computer*, vol. 14, no. 10, pp. 13-37, Oct. 1981.

[3] A. Borgida, S. Greenspan, and J. Mylopulos, "Knowledge representation as the basis for requirements specifications," *Computer*, vol. 18, no. 4, pp. 82-91, Apr. 1985.

[4] F. P. Brooks, Jr., "No silver bullet, essence and accidents of software engineering," *Computer*, vol. 20, no. 4, pp. 10-19, Apr. 1987.

[5] M. Bunge, *Treatise on Basic Philosophy: Vol. 3: Ontology I: The Furniture of the World.* Boston, MA: Reidel, 1977.

[6] M. Bunge, *Treatise on Basic Philosophy: Vol. 4: Ontology II: A World of Systems.* Boston, MA: Reidel, 1979.

[7] K. M. Chandy and J. Misra, *Parallel Program Design: A Foundation.* Reading, MA: Addison-Wesley, 1988.

[8] E. F. Codd, "A relational model of data for large shared databanks," *Commun. ACM*, vol. 13, no. 6, pp. 377-387, June 1970.

[9] M. J. Culnan, "The intellectual development of management information systems, 1972-1982: A co-citation analysis," *Management Sci.*, vol. 32, no. 2, pp. 156-172, Feb. 1986.

[10] P. J. Curtois, "On time and space decomposition of complex structures," *Commun. ACM*, vol. 28, no. 6, pp. 590-603, June 1985.

[11] J. D. DiStefano, III, A. R. Stubberud, and 1. J. Williams, *Feedback and Control Systems.* New York: McGraw-Hill. 1976.

[12] R. G. Dromey. "Systematic program development," *IEEE Trans. Software Eng.*, vol. SE-14, no. 1, pp. 12–29, Jan. 1988.
[13] J. D. Gannon. R. G. Hamlet. and H. D. Mills. "Theory of modules," *IEEE Trans. 'Software Eng, vol.* SE-13, no. 7, pp. 820-829, July 1987.
[14] M. Hamilton and S. Zeldin, "Higher order software-A methodology for defining software," *IEEE Trans. Software Eng., vol.* SE-2, no. 1, pp. 9-32, Mar. 1978.
[15] 1. J. Hayes. Ed., *Specification Case Studies.* Englewood Cliffs, NJ: Prentice-Hall, 1987.
[16] C. A. R. Hoare, "An overview of some formal methods for program design," *Computer*, vol. 20, no. 9, pp. 85-91, Sept. 1987.
[17] M. A. Jackson. *Principles of Program Design. New* York: Academic, 1975.
[18] *C.* B. Jones, *Systematic Software Development Using VDM.* Englewood Cliffs, NJ: Prentice-Hall, 1986.
[19] J. Karimi and B.R. Konsynski, "An automated software design assistant," *IEEE Trans. Software Eng., vol.* 14, no. 2, pp. 194-210, Feb. 1988.
[20] R. C. Linger, H. D. Mills, and B. I. Witt, *Structured Programming: Theory and Practice.* Reading, MA: Addison-Wesley, 1979.
[21] M. D. Mesarovic and Y. Takahara, *General Systems Theory. New* York: Academic, 1975.
[22] A. Mili, J. Desharnais, and J. R. Gagne, "Formal models of stepwise refinement of programs." *ACM Comput. Surveys. vol. 18.* no. 3. pp. 231-276, Sept. 1986.
[231 J. G. Miller. *Living Systems.* New York: McGraw-Hill. 1978.
[241 H.G. Mills, V. R. Basili, J.D. Gannon, and R.G. Hamlet, *Principles of Computer Programming: A Muthematicul Approach.* Boston, MA: Allyn and Bacon, 1987.
[25] J. Moses, "Computer science as the science of discrete man-made systems," in *The* Study *of Information:* Interdisciplinary *Messages,* F. Machlup and U. Mansfield, Eds. New York: Wiley, 1983.
[26] G. J. Myers, *Reliable Software Through Composite Design. New* York: Petrocelli/Charter, 1975.
[27] T. W. Olle, J. Hagelstein, LG. Macdonald, C. Rolland, H.G. Sol, F. J. M. Van Assche, and A. A. Verrijn-Stuart, *Information System Methodologies: A Framework for Understanding.* Reading, MA: Addison-Wesley, 1988.
[28] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Commun.* ACM, vol. 15, no. 12, pp. 1053-1058, Dec. 1972.
[29] ——, "On a 'buzzword': Hierarchical structure," in *Inform. Processing* 74. Amsterdam, The Netherlands: North-Holland, 1974, pp. 336-339.
[30] ——, "Education for computing professionals," *Computer, vol. 23,* no. 1, pp. 17-22, Jan. 1990.
[31] J.D. Paulson, "Reasoning tools to support system analysis and design," Ph.D. dissertation, Univ. British Columbia, 1988.
1321 S. B. Rogers. "A simple architecture for consistent application design," *IBM Syst. J.*, vol. *22,* no. 3, pp. 199–213, 1983.
[33] H. A. Simon, *The Sciences of the Artificial,* 2nd ed. Cambridge, MA: MIT Press, 1981.
[34] J. M. Spivey, *The Z Notation: A Reference Manual.* Englewood Cliffs, NJ: Prentice-Hall, 1989.
[35] T. H. Tse, "The identification of program unstructuredness: A formal approach," *Comput. J.*, vol. 30, no. 6, pp. 507-511, 1987.
[36] P.A. M. Van Dongen and J.H. L. Van Den Bercken, "Structure and function in neurobiology: A conceptual framework and the localization of functions," *Int. J. Neurosci., vol.* 16, pp. 49-68, 1981.
1371 Y. Wand, "A proposal for a formal model of objects," in *Object-Oriented Concepts, Applications, and Databases,* W. Kim and F. Lochovsky, Eds. Reading, MA: Addison-Wesley, 1989, pp. 537-559.
1381 Y. Wand and R. Weber, "An ontological analysis of some fundamental information system concepts," in *Proc. Ninth Int. Conf. Information Systems,* Dec. 1988, pp. 213-226.
[39] ——, "A deep structure theory of information systems," Univ. British Columbia, unpublished working paper, Mar. 1988.
[40] ——, "A model of control and audit procedure change in evolving data processing systems," *Accounting Rev., vol.* LXIV, no 1, pp. 87-107, Jan. 1989.
[41] ——, "A model of systems decomposition," in *Proc. Tenth Int. Conf. Information Systems,* Dec. 1989, pp. 41-51.
[42] ——, "An ontological evaluation of systems analysis and design methods," in *Information Systems Concepts-& In-Depth Analysis,* E. Falkenberg and P. Lindgreen, Eds. Amsterdam, The Netherlands: North-Holland, 1989, pp. 79–107.
[43] R. Weber, "Toward a theory of artifacts: A paradigmatic base for information systems research," *J. Inform. Syst.*, vol. 1, no. 2, pp. 3-19, Spring 1987.
[44] N. Wiener, *Cybernetics: Or Control and Communication in the Animal and the Machine,* 2nd ed. Cambridge, MA: MIT Press, 1961.
[45] J. L. Whitten, L. D. Bentley, and T. I. M. Ho, *Systems Analysis and Design Methods.* St. Louis, MO: Times Mirror/Mosby, 1986.
[46] E. Yourdon and L. L. Constantine, *Structured Design: Fundamentals of a Discipline of Computer Program and System Design.* Englewood Cliffs, NJ: Prentice-Hall, 1979.
[47] P. Zave, "The operational versus the conventional approach to software development," *Commun. ACM,* vol. 30, no. 2, pp. 104–118, Feb. 1984.

Yair Wand received the B.Sc. degree in mathematics and physics from the Hebrew University, Jerusalem, Israel, in 1966, the M.Sc. degree in physics from the Weizmann Institute of Science, Rehovot, Israel, in 1970, and the D.Sc. degree in operations research from the Technion, Haifa, Israel, in 1977.

Since 1977 he held positions at the Faculty of Management, the University of Calgary; the Faculty of Commerce, the University of British Columbia; and the Faculty of Industrial Engineering and Management, the Technion. His industry experience includes working with IBM (Israel) Ltd. and as an independent consultant. Presently he is on faculty at the MIS Division, Faculty of Commerce and Business Administration, the Universitv of British Columbia. His research interests are modeling of information systems and formal foundations of systems analysis-and design.

Dr. Wand is a member of the Association for Computing Machinery and the IEEE Computer Society.

Ron Weber received the B. Com.(Hons.) degree and University Medal from the 'University of Oueensland. Oueensland. Australia. in 1972 and the M.B.A. degree and Ph.D. degree in management information systems from the University of Minnesota in 1976 and 1977, respectively.

He is currently GWA Professor of Commerce at the University of Queensland. His current research interests are in the areas of formal modeling of information systems, computer control and audit, and systems analysis and design.