1988

AN ONTOLOGICAL ANALYSIS OF SOME FUNDAMENTAL INFORMATION SYSTEMS CONCEPTS

Yair Wand
The University of British Columbia

Ron Weber
University of Queensland

AN ONTOLOGICAL ANALYSIS OF SOME FUNDAMENTAL INFORMATION SYSTEMS CONCEPTS

Yair Wand
The University of British Columbia

Ron Weber
University of Queensland

ABSTRACT

This paper describes how ontological concepts can be used to model information systems. We view an information system as an object that is independent of its use or its technology of implementation. The main premise of the model is that an information system is a representation of a real-world system, and as such it should possess certain characteristics. We show how the model can be used to define various concepts such as real-time, batch, data processing, management reporting, decision support, controls, and decomposition. Furthermore, we show how the model may serve as the foundation of a theory of systems analysis and design. In particular, it provides a formal definition of information systems specifications and a normative model of decomposition.

1. INTRODUCTION

Methodologies for systems analysis and design deal with modelling of information systems. Nonetheless, despite the abundance of such methodologies, no theoretical foundation for systems analysis and design exists (Bubenko 1986; Floyd 1986). Indeed, when examining different methodologies one must wonder why such varied concepts as activities (Kung and Solvberg 1986; Lundberg, Goldkuhl, and Nillson 1981), processes (Jackson 1983), data flows (De Marco 1979; Gane and Sarson 1979) and objects (Bubenko 1980; Essink 1986) are used for describing information systems.

This paper reports research results obtained from our attempt to model an information system as an object itself, regardless of its use, physical implementation, or management. The motivation for the research has been to provide a foundation for a theory of information systems structure and design. We seek a theory of information systems artifacts (Weber 1987) that will not change as new technology appears, nor as new ways of using the technology evolve. Thus, the formalization is undertaken without reference to either the purpose and use of the information system, or to the available technology. The proposed model comprises a set of constructs, assumptions, and propositions about the nature of an information system.

We will show that the modelling constructs can be used to formalize various aspects of information systems. Specifically, the model will be used to derive necessary requirements for good information systems; to formalize the roles of software and data; to define the differences between data processing, management reporting, and decision support; to define real time and batch; to formalize the notion of information systems controls; and to suggest a theoretical foundation for decomposition.

The paper proceeds as follows. Section 2 discusses the motivation for selecting the proposed modelling approach. Section 3 presents the concepts of the model. Section 4 defines an information system in the terms of the model and presents some necessary conditions for an information system to be "good." Section 5 explains various information systems concepts according to the model. Section 6 presents our conclusions and some ideas regarding systems analysis and design.

2. FOUNDATIONS OF THE MODEL

We begin with the following proposition:

> An information system is an artificial representation of a real-world system as perceived by humans.

This notion accords with many discussions in the systems analysis and design literature (Borgida, Greenspan, and Mylopoulos 1985; Bubenko 1986; Jackson 1983; Myers 1978). Note, the representation is that of *perceptions*, rather than the "real" system, because the only way for us to know about reality is via the perceptions of human beings (Borgida, Greenspan, and Mylopoulos 1985). Also, humans may perceive systems that exist only in their minds, namely, conceptual systems. These are included in the definition as well. To make a distinction between the representation and the perceived system, we call the latter "the real-world system" (although we have no way knowing if it really exists). These perceptions may depend on the individual or the situation, but since issues of purpose are excluded from the model, the perceptions are viewed as given. Also, the technology used to implement the system is not included in the model. Thus, the definition is independent of both the purpose of the system and its implementation technology. Consequently, the model does not deal with questions of value, benefits, cost, and efficiency.

The specific part of the real world to be captured by the information system is called "the universe of discourse" (Bubenko 1986). In the definition we confine ourselves to human-created representations. Hence, we exclude representations that are created in the mind only. In this light, we define:

> Information systems development is a transformation from some perceptions of the real world into an implementation of a representation of these perceptions.

We assume that the development transformation proceeds through the following three successive transformations:

1. **Analysis**: from perceptions of reality into a formal model of this perception. The model generated in this process is called a conceptual model (Kung and Solvberg 1986), a model of the world (Borgida, Greenspan, and Mylopoulos 1985), or a WHAT-oriented model (Bubenko 1980). In the following it will be called "The Model of Reality," although it actually is a model of perceptions of reality, rather than reality itself.

2. **Design**: From the model of reality into a model of a representation. Note the outcome of the design transformation is a model of the information system and not the information system itself.

3. **Implementation**: From a model of the information system into a realization of the information system.

The first transformation operates on human perceptions that are not well defined. This imprecision is manifested in the difficulties associated with requirements identification and analysis (Sibley 1986). Nonetheless, the transformation should provide structured and unambiguous output to enable the second transformation to be carried out. Thus, a key element in information systems design methodologies is a modelling tool to describe reality.

The second transformation deals with models rather than concrete systems. It operates on a formal model of the real system to generate a formal model of an information system that can be mapped into implementation primitives. The real-system model and the implementation system model conform to the "conceptual information system model" and the "data system model," respectively (Essink 1986).

The basic assumption underlying our formalization of information systems is:

> For an information system to be a good representation, some essential characteristics of perceived reality must be captured in the information system.

The characteristics that should be preserved in this transformation must exist in any implementation of the information system, regardless of the technology or the interfaces between the system and its environment. Therefore, they will be termed the *invariants* of the information system development transformation.

To model an information system as a representation, a formal scheme is needed to represent the real world. This scheme must be able to capture both the *statics* (structure) and the *dynamics* (behaviour) of the real world (Kung and Solvberg 1986). Also, apart from its role as a representation, the information system can be viewed as a real thing, hence, the same scheme will also be used to describe it.

The formal scheme we use is based on the ontological formalism developed by Bunge (1977, 1979). Ontology was chosen because its objective is to describe the structure of the real world. The adaptation of ontology to information systems is elaborated elsewhere (Wand 1988; Wand and Weber 1988b). The next section outlines the basic concepts of the model.

3. CONCEPTS OF THE MODEL

The world is viewed as made of things, or objects, that have properties by which they are known. A thing is modelled by a functional schema -- a set of functions that assign values to its properties. A possible combination of property values comprises a *state* of the thing. The set of states that a thing may assume is termed the *possible state space* $S = \{s\}$. For a given purpose, we are only interested in a certain *observable* set of properties of the thing, and we describe their values by a *state vector* $<x_1,...,x_n>$. We will assume that the state vector contains all the necessary structural information about the thing, as required by the purpose of the analysis. Therefore, we will equate the state of a thing with its state vector.

The dynamics of a thing are modelled by state changes that are called *events*. An event, e, is defined by the states before (s_1) and after (s_2) the change: $e = <s_1,s_2>$. The set of states a thing assumes over time is called: the *history* of the thing. Two things will be said to *interact* if their histories are not independent, namely, the history of at least one of them differs from what it would be without the presence of the other. In other words, one of the things will not assume certain states that it might have otherwise assumed.

A *system* is a thing that comprises a set of interacting things. These things are the *composition* of the system. The feature that makes a system different from an arbitrary aggregate of things is the requirement that for every partition of its composition, there must be interactions between things from the two subsets. Because of these interactions, the possible states of the things in the composition are constrained. Also, a system can be viewed as a thing that has its own state space. Therefore, a mapping must exist between the state of the system and the possible states of its components, namely, $S \rightarrow S_1 * ... * S_n$ (S_i being the state space of thing i). In particular, some of the system's state variables might be state variables of its component things.

To demonstrate these concepts, consider a manufacturing firm as an example. The various things that comprise the system include products, workers, machines, and raw materials. The state of a machine at a given time includes information on the product the machine is processing at that time. The state of the product at a given time includes information on the workers and machines involved in manufacturing it at that time. Clearly, the sequences of states the machines and products in process are traversing in time are related.

Things that interact with the system but are not included in the composition of the system are called the *environment* of the system. Since the environment interacts with the things in the system, it might cause a change in the state of a thing in the system; and, due to the interactions inside the system, other things may change their state. This view of system dynamics is formalized via the notion of a stable state:

Definition: A *stable state* of a system is a state in which the system will remain unless forced to change its state by the environment.

Two assumptions will be made regarding stable states:

The Stability Assumption: A change of state will happen if and only if the system assumes a state that is not a stable state.

The Unique Response Assumption: A system in an unstable state will change to a stable state that is *uniquely* defined by the unstable state.

Consider again our manufacturing example. Suppose that the firm is producing only to fill customer orders. If there are unfilled orders, the firm must produce to fill them; if there are none, the plant will be idle. In this case, there is one stable state: "no production." The environment of the firm includes customers who submit orders. As soon as an order is received, the state includes unfilled orders and becomes unstable.

The two assumptions above underlie the concept of a *law*. A law is a mapping that might change the state of a system. The law will only change the state if it is unstable, and the new state will be stable.

Definition: A law is a function on the set of possible states: L: $S \rightarrow S$ such that only unstable states are changed and they are mapped into stable states.[1]

A law is a function that captures two types of information: (a) Whether a state is stable; and (b) how an unstable state will change. Laws are compact descriptions of the interactions inside a system. In many cases, they are the practical way to describe the behaviour of the system. Also, while laws are a manifestation of interactions among system components, they are expressed in terms of the state variables of the system rather than the state variables of the components.

The dynamics of a system describe the way it may change its state. Assume that the system is in a stable state s_1 and that the environment forces it to change its state. Such a change will be termed an *external event*. Note that according to this view, the external event is not what really happens in the environment but the change of state of the system. In the manufacturing example the external events are orders as placed with the firm, and whatever triggered these orders is unknown to the company.

As a result of an external event, the system will be in a new state s'. If this state is unstable, the system laws will force it to change to a stable state s_2. The transition from s_1 to s_2 will be termed the system's *response*. The system's response is a change of state, namely an event, hence it will be called an *internal event*. The dynamics of the system are described, therefore, in terms of a sequence external event-internal event (system response):[2]

stable state $\rightarrow$ *external event* $\rightarrow$ *unstable state* $\rightarrow$ *internal event* $\rightarrow$ *stable state*[3]

In the example, the first stable state (s_1) is a state without unfilled orders, The external event is a customer order, the unstable state (s') is a state with unfilled orders, and the second stable state (s_2) is the state after all orders have been filled. This last state is different from the first stable state as now the financial status of the firm is different.

As an additional illustration of the concepts, consider an accounting system where the state of an account is defined, at a given point in time, by the balance, B, and the accruing transactions up to that time $\{T_1,...,T_n\}$. Denote the value for transaction T_i by A_i. The system law is described in terms of two conditions for stability:

(L_1) $B = \text{Sum}\ \{A_i \mid i=1,...,n\}$

(L_2) A_i is fixed, $i=1,...,n$

Consider, now, two external events:

1. A new transaction is added to the state. Then, due to L_1, the balance, B, will change to reflect this transaction.

2. The balance is forced to change (say, it was found to be incorrect). L_1 now does not hold. However, L_2 does not allow a change to any of the existing transactions. Therefore, a new *adjustment* transaction will have to be added to the state vector to maintain the balance.

Based on the above concepts, a full description of the statics and dynamics of a system is given by the pair <S,L> where S is the possible state space and L is the system law. In practice, when analyzing a system, the set of possible external events may be limited to a desired subset of *relevant events*, thereby limiting the set of unstable states for which the law has to be known. Accordingly, the formal scheme used to describe a system is a triplet <S,L,E>, where S is the set of possible states, L is the system law, and E is the set of relevant events. If E is unspecified, L must be defined for all possible states of the system.

These concepts can be applied to both the real system and its information system representation. The next section analyzes the role of an information system as a representation.

4. THE INFORMATION SYSTEM MODEL

In this section we use the formalism developed in the previous section to model the information system and to suggest necessary requirements for an information system to be a good representation. We begin by making the following assumption:

The Interaction Assumption: The real world system is part of the environment of the information system that represents it.

This assumption implies that the real system can effect state changes in the information system. Such state changes can be viewed as transactions in the information system and may happen whenever the real system changes its state.

We examine, now, the information system as a *representation*. Our starting point is the premise that information systems are needed to save the effort required to constantly monitor or predict the state of the real system. Therefore, we define the following *necessary condition*:

An information system is a *good representation* only if its sequence of states in time is a mapping of the sequence of states the real system traverses or may traverse.

In light of this condition, we conceive information systems as **STATE TRACKING MECHANISMS**. For the information system as a state tracking mechanism to be a good representation (in the above sense), there are four necessary requirements that together are also sufficient. To formulate the requirements we recall the representation of a system as a triplet <S,L,E>. Accordingly, the information system will be described in terms of a triplet <M,P,T>, where M is the set of possible states, P is the information system law, and T is the set of relevant external events. Using this representation, the four requirements are:

1. **The Mapping Requirement**: For every state of the real system there exists at least one matching state of the information system. Every state of the information system matches a state of the real system. The information system states that match a real system state s will be denoted by rep(s).[4]

The mapping requirement relates the states of the two systems and, therefore, deals only with statics. The next requirement links the dynamics of the two systems, that is, the way the states may change:

2. **The Tracking Requirement**: The laws of the information system replicate the laws of the real system.[5]

Explanation: Let s be a state of the real system and m = rep(s) its representation in the information system. Assume that the real system law, L, maps s into s', and that the representation of s' in the information system is m'. Then the information system law, P, maps m into m'. Note, this requirement implies, in particular, that stable states in the real system map into stable states in the information system. The tracking requirement guarantees that the information system "knows" how to replicate the real system behaviour.

Consider the accounting example above. The mapping requirement implies that for every possible combination of transactions and balances there will be a possible matching state in the database of the system. The tracking requirement implies that the way the database states may change reflects the two "laws" of this example.

However, even when both requirements are satisfied, we still can not guarantee that the information system reflects the behaviour of the real system, because we have no requirement that ensures the information system will "know" about changes in the real system. Note that according to our view, the information system "depends" in its behaviour on "proper reporting" by the real system. The following two requirements specify how reporting must happen to ensure tracking:

3. **The Reporting Requirement**: For every event that happens in the real system, an event that reflects it must happen in the information system.[6]

If the information system is in a state that reflects the real system state (not necessarily in the same time), this requirement ensures that the unstable states traversed by the information system reflect the unstable states traversed by the real system. However, the reporting requirement still does not guarantee that the information system will track the real system properly because the information system has to reflect sequences of events that might happen rather than just one event. The following requirement specifies when this condition will hold:

4. **The Sequencing Requirement:** Let $e_1,...,e_n$ be a sequence of events in the real system. Let $t_1,...,t_n$ be the corresponding events in the information system. Then the order of reporting $\{t_i\}$ must be the same as the order in which $\{e_i\}$ happen.

Assume that all four requirements hold and that the information system state has been "reset" to match the state of the real system before the latter began effecting events in the information system. As events happen in the real system, it will generate events for the information system (the reporting requirement) in the same order that the original events happen in the real system (the sequencing requirement). According to the tracking requirement the information system will proceed through a set of states that reflects the set of states traversed by the real system. Formally, we define:

Definition: An information system will be said to be a *representation* of a real system if the information system traverses the sequence of states $m_1,m_2,...,m_n$, such that m_i = rep(s_i), when the real system traverses the sequence of states $s_1,s_2,...,s_n$.

5. DEFINITION OF INFORMATION SYSTEMS CONCEPTS

In this section, we demonstrate the explanatory power of the model by using it as a framework to formalize some fundamental concepts of information systems. First, time of events is introduced to define batch and real time. Second, the model is used to distinguish between data processing, management reporting, and decision support. Third, information systems controls are defined. Fourth, the role of data and processes is examined. Finally, formalization of system decomposition is proposed.

5.1 Batch and Real Time

In the model presented above, time did not appear explicitly, except for defining a definitions for real-time and batch systems. We begin by introducing some notation.

In the following analysis, the time of an event is denoted by time(event) where "event" could be a real event, e, an internal event (system response) in the real system, r, an information system event, t, or its response (processing), p. An event happening when the system is in a state s_i (m_i) will be denoted by e_i (t_i), the information system event resulting from the real event e_j will be denoted by t_j, and the information system response to t_k by p_k. Note, this notation does not necessarily imply that the information system states $m_1,m_2,...$ are the representations of $s_1,s_2,...$ because nothing has been assumed regarding the information system behaviour.

Consider an information system that is a good representation. The information system tracks the real system faithfully, but tracking might occur with delay. In short, at a given time the information system may be in a state that represents an earlier state of the real system, and therefore it will not reflect the current real state. The following definition provides a condition for the information system to reflect the present state of the real system:

Definition: An information system is said to be a *real-time representation* of a real system if (1) it is a representation, and (2) for every pair of adjacent events, time (p_i) < time (e_{i+1}).

The *Real-Time Condition* implies that any real event must be reported to the information system and processed by it before the next real event happens. Therefore, the information system will be in a state that is a representation of the actual state of the real system just before the event happens. This requirement is important when the next real event depends on knowledge of the state of the real system. To illustrate, consider an accounting system. If, at all times, the next transaction depends on the state of the account, the real time condition must hold. In particular, consider a query, namely, a request for the state of the system, as a real event. When the real-time condition holds, the observed state of the information system will always be a representation of the state of the real system just prior to the observation.

In this light, a *batch* system is an information system where the real time condition does not hold. More specifically, when a real event happens, the previous real event may not have been processed (or reported) yet; hence, the information system state may not yet reflect the change in the real system.

In addition to providing untimely information, a disadvantage of batch systems is the possibility of inconsistencies arising from "updating cycles." In particular, events may be reported but not in their order of occurrence. The model provides a condition that must hold to avoid such inconsistencies:

Definition: An information system is said to be a *consistent representation* of a real system if (1) it is a representation of the real system, and (2) for every pair of events, time(p_i) < time(t_j) if and only if time(e_i) < time(e_j).

The consistency condition implies that an event will be reported (and processed) only after all preceding events have been processed, and, therefore, the information system will assume a state that represents the state of the real system immediately after the event has happened. Note, since the inequalities contain real or information system events only they impose constraints on the order of events in the two systems rather than on their timing. Hence, the information system does not have to be in a state that represents the real state at that time. To illustrate, consider, again, the accounting system. If balances must be correct after each transaction -- say, to calculate interest on the running balance -- then consistency is required.

In some cases, consistency may not be required for all events. Thus, we introduce a weaker requirement:

Definition: An information system is said to be a *complete representation with respect to an event* e_0 if (1) it is a representation, and (2) $\text{time}(p_j) < \text{time}(t_0)$ for all j such that $\text{time}(e_j) < \text{time}(e_0)$.

The completeness condition implies that all previous events have been processed by the time event e_0 is reported (as transaction t_0) to the information system. Note this requirement relates to a specific event (e_0) and not to all events. Under this requirement, the order of processing for some sequences of events may not reflect the order in which they happened. Therefore, at certain points in time the information system may be in a state that is not a representation of any real-system state. In our accounting system example, if the only requirement is that the end-of-month balances are correct after processing, then completeness for the end-of-month event is sufficient.

The three conditions are linked by the following theorem:

Theorem: Real Time $\Rightarrow$ Consistency $\Rightarrow$ Completeness for all events.

5.2 Transaction Processing, Management Reporting, and Decision Support

The definitions of transaction processing, management reporting, and decision support are usually based upon the way systems are used. We now propose alternative definitions, based on the concepts of states, events, and laws, to distinguish between such systems. Such definitions reveal the differences between the different kinds of systems from a *design* point of view.

Data processing systems are systems in which business *activities* are captured and used for (1) posting *status* information, and (2) generating other transactions. For example, a customer file may be updated by a change-of-address transaction, or an invoice may be generated based on name and address information in the customer master file and on customer purchase transactions. To formalize data processing systems, we begin by defining master and transaction records.

Definition: A *master record* is a representation of the state of some aspects of the organization at a given point in time.

A *transaction record* is a representation of a change that happened to the status of some aspects of the business.

Recall, an information system is represented in the model by the triplet {M,P,T}, where M is the set of system states, P is the set of laws governing the information system responses to events, and T is the set of possible (external) events. For a data processing system, the states include information about the real-system status, and the events are the *transactions* that are the inputs to the information system. Since the information system and its environment (namely, the real system) are linked, the environment must be able to modify some components of the state vector. We term these components the *input components* of the information system. Thus, a given information system state ($m \in M$) comprises both status and input information and can be viewed schematically as a pair: m = (status,input). In this view, transactions (t) are information system external events and their effect is to modify the input components: $t \rightarrow \text{input}$.

As a transaction affects the value of the input components, the new state becomes unstable and the information system responds by changing its state to restore stability. When this change occurs, the input component remains fixed according to the definition of an external event. Thus, the sequence of states traversed by the information system (IS) as a result of a transaction is:

(old status,old input) -- transaction t -- (old status,input=t)
(old status,input=t) --- IS Response --- (new status,input=t)

Thus, the information system laws effect a transformation within the components of the state vector, and updating is viewed as the outcome of the information system responding to external events according to its laws. Note that a transaction, t, applied to a master record, m, must represent a change happening while the real system is in a state that is represented by m.

The model can also accommodate two types of management information systems: *management reporting* and *decision support*. *Management reporting* is defined as the generation of information that potentially may be used to decide on a course of action. In our model, however, purpose can not be accommodated, and, in particular, there is no explicit way to accommodate decision making.

Nonetheless, the concepts of laws and states can be interpreted in the context of decision making. In order to make a decision, the decision makers must have a *model of the decision situation* -- namely, a view of the world and its behaviour -- and they need to know or be able to anticipate states of this world. This view may be quite different from the operational-level view represented in the data processing system. For example, consider a marketing and sales information system that processes orders and provides sales summaries. From an operational point of view, the real-system states include information about products and customer orders, and the information system tracks events like creation of orders, order delivery, and changes in inventory. For decision making purposes, the real-system states include information such as sales of a product in a given unit of time (e.g., month) by customer, by area, etc.

The information system must represent both the operational and the decision making views of the world. In the real world, the two views are connected by laws governing their states. For example, the monthly sales that are of interest to decision makers, are summaries of the sales transactions occurring in a month, that are in the operational realm. Via the Mapping Requirement: $S \leftrightarrow M$, a state m should be a representation of both the operational and the decision aspects of reality in an information system that provides for data processing and management reporting. Since, in practice, these two aspects can be considered separately, we make the following assumption:

> The real state vector, s, can be decomposed into two components: s = (op,dm)

Where op is a state description of the *operational* part of the real system (state space: OP), and dm is a state description of the *decision making* part of the real system (state space: DM).

If the information system is to be a good representation of reality, it should reflect this separation. We assume therefore:

> The management reporting state components of an information system are separable from the data processing state components, namely, the information system state can be viewed as m = (dp,mr).

Where dp is a state description of the *data processing subsystem* (state space: DP), and mr is a state description of the *management reporting subsystem* (state space: MR). Furthermore, we assume:

> The mapping between the real system and the information system can be separated: $OP \leftrightarrow DP$; $DM \leftrightarrow MR$.

However, it is important that the two state components must be connected by laws in their respective systems since they represent aspects of the same state.

In this light, what does the view of an information system as a state tracking mechanism mean for management reporting? Typically, no events are reported directly to the management reporting component. Rather, the states of the information system are decided by the operational (transaction processing) events. The tracking requirement implies that the information system laws should be defined so that the management reporting components will behave as a (good) representation of the decision making system as operational events are reported. Note, according to this description, no external events have to occur in the management reporting component of the information system.

Consider, now, external events in the decision making system. To be able to respond to such events, the information system must contain a representation of the *laws* governing the behaviour of the "real" decision making system, that is, the information system must be a representation of the conceptual *model of the world* of the decision maker. In such a system, the Tracking Requirement implies that changes of state variables of the decision making component will cause the information system to assume a state that represents a new state of the world according to the decision making model. Information systems that can track the behaviour of a decision making model are useful for decision support. Thus, our model of an information system as a tracking mechanism shows that the difference between a management reporting system and a decision support system lies in the latter containing laws that represent the decision making model and in the types of events reported to it.

No assumption has been made about the nature of the laws, namely, the *functional relationships* among the state variables. Thus, the model allows for both mathematical and logical (rule based) laws. Also, *outputs* and *inputs* are not distinguished because they are determined by which state variables are changed and which are observed. Thus, the model allows for straightforward and goal-seeking calculations for numerical laws, as well as for forward- and backward-chaining for rule-type laws.

We conclude with a note on the relationships between the times of the real-world events and the corresponding information system events. In data processing and management reporting systems, past events are reported as transactions in the information system. Thus, the state of the information system lags in time behind the state of the real system. In decision support systems, an assumed change may be reported to the information system in order to predict the behaviour of the real world should this event happen. Thus, such systems respond to *possible or imaginary future events* rather than to *actual past events*.

5.3 Controls in Information Systems

Information systems controls are usually defined by their purpose and objectives (Weber 1988, pp. 38-39). Our model provides a definition of controls that is independent of the purpose (but reflects a designer's viewpoint) using the concepts of states, events, and laws. Since the model does not deal with any technological or implementation issues, the discussion of controls relates only to the logical aspects of the information system. Thus, we do not deal with controls that relate to physical resources and physical integrity of the system.

Controls are required because information systems may contain or provide information or do some processing that is considered *unlawful* in some sense. This outcome can occur even if an information system is a good representation. Recall that the idea of an information system as a state tracking mechanism is based on the notion of laws that define the behaviour of the system. If some states or events are viewed as unlawful, then additional knowledge about the world exists that is not captured by the system laws. This additional knowledge refers to the specification of some possible states or changes of states as being undesirable or *unlawful*. The notion of unlawfulness captures semantic knowledge because it conveys *meaning* assigned by humans to certain situations. There are two reasons why such knowledge may be required. First, the real-world model may not capture lawfulness. Second, errors may be made in the implementation. Both cases are modelled in the same way.

We begin by formally introducing the notion of lawfulness.

Definition: Let S be the state space of the system. The *lawful state space* is the subset of states $S_L \subseteq S$ that are considered valid in some sense.

Definition: Let S_L be the lawful state space. The *lawful set of events* is a subset E_L of all possible transitions in S_L ($E_L \subseteq S_L{*}S_L$) that are considered valid in some sense.

The latter definition pertains to any state transition, recognizing that both external events and internal events according to system laws may allow for unlawful transitions.

Corollary: If a system is in a lawful state and only lawful events are allowed, then it will always be in a lawful state.

A *control* can now be defined as a function that identifies each event as "lawful" (1) or "unlawful" (0), formally:

Definition: A control is a function: $C: E \rightarrow \{0,1\}$.

This definition also provides the basis for defining whether a control is *in place* and *working*:

Definition: A control is said to be *in place* if it is defined for all possible events.

Definition: A control is said to be *working* with respect to a lawful event set if and only if: for every event, the control assigns "lawful" if, and only if, the event is lawful.[7]

The notion of a control can be compared to the definition of a system law. A law has two components: (1) a function that maps a state into {stable,unstable}, and (2) for unstable states, a change function that specifies a stable state. While a law maps states into states, a control maps events into the set {lawful, unlawful}. However, for all pairs <s,L(s)> the transition is uniquely defined by the state s. Hence, for these pairs the control can be viewed as a function of the state, s.

To demonstrate, consider again the accounting example. The system law defines conditions such that the balances will be correct, it can not "prevent" reporting of a non-valid transaction (e.g., one that does not relate to the account). For this, we need a control that will prevent any events that might change the balance when a non-valid transaction is reported. Such a control checks the state of the system after a transaction has been reported, but prior to the change in the account. Therefore, it can be viewed as a function on the states of the system. More specifically, this is a function that is applied to a state that reflects an external event, hence, this is an input control.

5.4 Data and Programs in Systems Design

As defined in (Section 2), information system design is a mapping from a model of the real world to a representation that is viewed as an abstract system. The representation has to meet certain requirements regarding its state space and laws in order to function as a good representation. No reference was made to the actual implementation in terms of data and software. This issue will be addressed now. We begin with an implementation-oriented definition of an information system:

An information system is a representation that is implemented using data and transformations on data (processes).

An information system is modelled by the triplet <M,P,T>; hence, we must establish the link between the abstract constructs of states, laws, and events, and the concrete concepts of data and processes. Accordingly, we propose:

> In an information system implementation, data represent states and events; processes are the implementation of laws.

External events are known to a system via state changes. In the implementation, external events imply changes in some data. Changes in data arising via external events will be termed *transactions*, in accordance with the common meaning of a transaction. Since transactions are only known to the information system via changed states, their values must be part of the state vector. Recall, elements of the state vector of an information system that can be changed by the real system are called input components.

Section 4 outlined requirements for an information system to be a good representation. In particular, the mapping of an unstable state of the real system must be unstable in the representation and the laws in the representation must change it to a stable state that corresponds with a stable state in the real system. Since processes in an information system must meet these requirements, we observe that:

> Processes act as *law preserving mechanisms.*

This view of processes relates to the role of an information system as a representation. It is different from the usual definition of processes as transformations on data (implemented as sequences of manipulations).
In this light, the system design transformation can be defined as a mapping:

$$\{\text{States,Laws,Events}\} \rightarrow \{\text{IS States, Processes, Transactions}\}$$

The left-hand side is a model of the real system and conforms to the *system specifications*. The right-hand side is a model of the information system, using implementation-oriented constructs that correspond to the notion of the *system design*. In the implemented system, "IS States" and "Transactions" become data, and "Processes" materialize as processing mechanisms. No specific assumptions were made as to the exact mechanism by which data and processes are implemented. In computerized systems, data will be implemented in some machine-readable form, and processes will be implemented as software. We observe, therefore, that:

> Software is a machine executable *representation of laws* that operates on the data implementation.

5.5 On Decomposition

The complexity of information systems is the main obstacle in their implementation. Complexity is viewed as an "essential property" of software, that reflects the application complexity (Brooks 1987). Systems analysis and design methodologies deal with complexity by using decomposition strategies. Hence, most methods for analysis and design of information systems incorporate hierarchical decomposition of the system. For example Structured Analysis (De Marco 1979; Gane and Sarson 1979), SADT (Ross and Schoman 1977), Warnier-Orr Diagrams (Warnier 1974; Orr 1977), HOS (Hamilton and Zeldin 1976), and NIAM (Verheijen and Van Bekkum 1982). Indeed, obtaining a "good" decomposition of the system is a fundamental objective of systems design. Yet there is no generally accepted theory of decomposition. Rules for decomposition are heuristic and are viewed as "guidelines" (see, for example, Gane and Sarson 1979, p. 189).

Decomposition can be viewed as either a method of building complex objects or as a way of analyzing their behaviour. Simon (1981, p. 229) claims: "On theoretical grounds we could expect complex systems to be hierarchies in a world in which complexity had to evolve from simplicity." His conclusion is that success in building complex artifacts depends on the possibility of constructing them from lower-level aggregates. The alternative view is expressed by Curtois (1985, p. 590): "Decomposition has long been recognized as a powerful tool for the analysis of large and complex systems."

Our approach to decomposition accommodates both views. First, it should be used for the *analysis* of the *real system* to understand its behaviour. Then the knowledge obtained should be employed in the *design* to construct the model of the information system. The translation between the two systems can be undertaken directly because both are modelled using the same constructs. We now explain intuitively how decomposition can be formalized (an analytical treatment can be found in Wand and Weber 1988a).

Definition: A *subsystem* is a system whose composition is a subset of the system's composition. In addition, every thing in its composition interacts with other things in the same way it does in the system.

Definition: A *decomposition* of a system is a set of subsystems such that every thing in the system is either one of the subsystems or it is included in the composition of one of the subsystems.

We begin our analysis of decomposition with the *good decomposition* premise:

> A decomposition is good if the behaviour of the system can be represented by the behaviour of the subsystems in the decomposition.

For a decomposition to represent the behaviour of the system, two conditions must hold. First, each subsystem should have a *well-defined* behaviour by satisfying the stability and unique response assumptions (Section 3); namely, a law can be defined with respect to the subsystem. Second, each subsystem's behaviour must conform to the system's behaviour, namely, it must change its states in accordance with the way the system changes its state.

In the following, we will demonstrate the concepts using a simplified business example. Consider a mail order firm where the main operations include order processing, inventory management, and accounting. The decomposition we examine is into three subsystems: order processing and invoicing, inventory management and accounting. The state of the order processing/invoicing subsystem is defined in terms of customer master records and order details inputs; the state of the inventory subsystem is defined in terms of item master records and inventory transactions inputs; and the state of the accounting subsystem is defined in terms of ledger accounts and accounting transactions inputs.

Consider now the following types of events:

1. Inventory management events: these will affect changes of state in the inventory subsystem only. For example, an arrival of a shipment from a supplier is an external event that causes the "on-hand" state variable to change.

2. Orders are prepared by sales persons who check for product availability and price and write the prices into the orders. The invoicing subsystem checks the status of the customer and may "refuse" an order if the customer is in default. If the order is approved, an invoice is generated using the order details and the customer data, and the customer record is updated. For approved orders, the outcome of this processing includes transactions for the accounting subsystem and the inventory subsystem.

3. Customer payments are processed by the accounting subsystem, and transactions are generated for updating the customer balances in the order processing subsystem.

Consider now what happens in a subsystem when an external event occurs in the system. Examination of the sample events shows that the following possibilities exist:

1. No change occurs in the state of the subsystem. This is the case for the accounting and invoicing subsystems when an inventory addition occurs.

2. The external event directly changes the state of the subsystem. This is the case for the inventory subsystem when an inventory event occurs.

3. The external event does not change the state of a subsystem directly, but the resulting internal event (system's response) will change it. In the example, after an order has been approved by the order processing system, transactions are generated for the inventory and the accounting subsystems.

For an observer who "watches" a subsystem only, cases 2 and 3 appear as an external event in the subsystem. Whether this event will cause a response in the subsystem depends on how the state of the whole system will change as a result of the original event. If the system state is unstable, the state of the subsystem may change. This is the case for all the events in the example above. Our observer will conclude that the subsystem was in an unstable state in each of these events. If no further change happens in the state of the subsystem, the observer will conclude that the subsystem reached a stable state. For example, after an invoice issued, the invoicing subsystem state will not change further although events might still happen in the other two subsystems.

Consider now a subsystem that is in an unstable state. Two possibilities exist:

a. It will change to a state that depends only on its unstable state.

b. It will change to a state that is not decided by the unstable state.

To demonstrate these two possibilities in our example, consider the calculation of an invoice. If the price appears on the order, then the amount due can be calculated and the invoice can be issued based on the state information of the invoicing subsystem. Imagine, however, that the company may have, at times, "special sales" and that the discount information appears (as a percent value) in the inventory master records. The order includes the regular price and an indication if the price is final or may be subject to discount. Then, when calculating an invoice, different outcomes may appear for the amount due. In this case, the invoicing subsystem state information, as defined above, is insufficient to calculate the outcome for customer orders for items that might be on discount, but is sufficient for items for which the price is final.

The above observations about the behaviour of subsystems lead to the following definition:

Definition: A subsystem behaves *independently* for a given set of events if, for every event in the set, it eventually reaches a stable state that depends only on the unstable state it attains due to the event.

For a subsystem that behaves independently, a subsystem law exists that maps the subsystem states in accordance with the system law for a given set of events. If, however, there is an event for which the outcome can not be "predicted" from the state information of the subsystem, then the subsystem does not behave independently. The additional information required "belongs" to other subsystems. This situation arises because the things in the subsystem interact with other things in the system.

Based on the notion of independently behaving subsystems, a good decomposition can now be defined:

Definition: A decomposition is good with respect to a given set of external events if every subsystem in the decomposition behaves independently for all events in the given set.

We return to the observer who watches a subsystem that behaves independently. When the subsystem undergoes an external event, its state after responding to the event can be fully predicted and it is stable with respect to the subsystem law. However, the whole system may still be in an unstable state, so that states of other subsystems may have to change. In our example, the accounting and inventory subsystems will change state after the invoicing subsystem has approved an order and generated an invoice. It follows that an internal event in the system may be viewed as a *sequence of changes* in subsystems.

More detailed analysis of the behaviour of an independently behaving subsystem reveals various possible cases. These cases differ depending upon whether the subsystem is affected directly by the external event or by the internal event following it, and whether the event changes the subsystem's state to a state that is part of a stable or an unstable state of the whole system. In some cases, one of several subsystems may be "activated"; these correspond to *transactional decomposition* ("transaction analysis" Yourdon and Constantine 1979). In other cases, subsystems operate in "sequence"; these conform to *source-transform-sink* (STS) decomposition (Myers 1978) and "transform analysis" in (Yourdon and Constantine 1979).

Some observations can be made about decomposition in the model. First, a "good" decomposition is explained in terms of the behaviour of a structural decomposition with respect to the system law - that is, both statics and dynamics of the system must be considered. Second, a "good" decomposition is defined with respect to a given subset of external events. Therefore, it is an "approximation." The subset of events defines an "application." Finally, good decomposition in the model is derived from knowledge of the real system. It is removed from implementation related considerations. This result is in contrast to common definitions based on information systems-dependent concepts such as modules, data flows, inputs, and outputs.

To conclude, we comment on the meaning of decomposition for information systems implementation. Recall that states are mapped into data and laws are implemented as processes. An independently behaving subsystem will therefore be implemented as a combination of data and processes that fully describe how the data change, with no need for other data. This requirement conforms to the intuitive notion of module independence (Gane and Sarson 1979). However, since subsystem independence holds for some events only, a module is independent only for a *given application*. Finally, a subsystem that behaves independently for *all* conceivable events can be implemented as a set of data and related processes that behave independently for all applications. This notion underlies the concept of an abstract data type or an object.

6. CONCLUSION

The model described in this paper is an attempt to define information systems as abstract objects, independent of their use and implementation technology. We have attempted to demonstrate how a model based on ontological concepts can be used to describe various fundamental information systems concepts. Thus, the notion of real time that is usually viewed as technology dependent was defined with no reference to technology. Similarly, a distinction between transaction processing systems, management reporting systems, and decision support systems could be made via formal definitions derived from the model rather than via use-oriented concepts. These two examples do not mean that technology and use are unimportant; rather they indicate that important concepts can be analyzed in terms of the characteristics of the information system itself.

The discussion of the above concepts, as well as the discussion of controls, can be viewed as a demonstration of the *descriptive* power of the model. However, we believe that the model also has *predictive* power -- specifically, it can be used to suggest new approaches to information systems analysis and design. We make three such predictions. First, since laws have a fundamental meaning, systems analysis practice should benefit from concentrating on eliciting system laws in explicit form in addition to, or instead of, the current practice of identifying processes or activities. A potential advantage of obtaining specifications in the form of system laws is that law definitions may be tested for consistency and completeness, which are not defined in most methodologies. Second, the model formalizes the concept of systems decomposition. Decomposition rules that may improve system design practices can be derived on the basis of the formalism. Third, since the model constructs are independent of information systems implementation, it can be used for comparing and evaluating systems analysis methodologies. Finally, the level of formalization attained in the model, might enable it to serve as the basis for developing automated systems analysis and design tools. Our current research is directed towards these issues.

7. ACKNOWLEDGEMENT

This research was supported in part by a Natural Sciences and Engineering Research Council of Canada operating grant N° OGP0004105, and by a grant from GWA Ltd.

8. REFERENCES

Borgida A., Greenspan, S., and Mylopoulos, J. "Knowledge Representation as the Basis for Requirements Specifications." *Computer*, April 1985, pp. 82-90.

Brooks, F. P., Jr. "No Silver Bullet, Essence and Accidents of Software Engineering." *Computer*, April 1987, pp. 10-19.

Bubenko, J. A., Jr. "Information Modeling in the Context of System Development." Information Processing 1980, *Proceedings of IFIP Congress*, Amsterdam: North Holland, pp. 395-411.

Bubenko, J. A., Jr. "Information System Methodologies - A Research Review." In T. W. Olle, H. G. Sol and A. A. Verrijn-Stuart (eds.) *Information Systems Design Methodologies: Improving the Practice*, Amsterdam: Elsevier Science Publishers B.V. (North-Holland), IFIP 1986, pp. 289-318.

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

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

Curtois, P. J. "On Time and Space Decomposition of Complex Structures." *Communications of the ACM*, Vol. 28, No. 6 (June 1985) pp. 590-603.

De Marco, T. Structured Analysis and System Specification. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1979.

Essink, L. J. B. "A Modelling Approach to Information System Development." In T. W. Olle, H. G. Sol and A. A. Verrijn-Stuart (eds.) *Information Systems Design Methodologies: Improving the Practice*. Amsterdam: Elsevier Science Publishers B.V. (North-Holland), IFIP 1986, pp. 55-86.
Floyd, C. "A Comparative Evaluation of System Development Methods." In T. W. Olle, H. G. Sol and A. A. Verrijn-Stuart (eds.) *Information Systems Design Methodologies: Improving the Practice*. Amsterdam: Elsevier Science Publishers B.V. (North-Holland), IFIP 1986, pp. 19-54.

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

Hamilton, M., and Zeldin, S. "Higher Order Software: A Methodology for Defining Software." *IEEE Transactions on Software Engineering*, Vol. SE-2 No. 1, (March 1976) pp. 9-32.

Jackson, M. A. *System Development*. Englewood Cliffs, NJ: Prentice-Hall, 1983.

Kung, C. H., and Solvberg, A. "Activity Modeling and Behavior Modeling." In T. W. Olle, H. G. Sol and A. A. Verrijn-Stuart (eds.) *Information Systems Design Methodologies: Improving the Practice*. Amsterdam: Elsevier Science Publishers B.V. (North-Holland), IFIP 1986, pp. 145-172.

Lundberg, M., Goldkuhl G., and Nillson, A. *Information Systems Development, A Systematic Approach*. Englewood Cliffs, NJ: Prentice Hall Inc., 1981.

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

Orr, K. T. *Structured Systems Development*. New York: Yourdon Press, 1977.

Ross, D. T., and Schoman, K. E. "Structured Analysis for Requirements Definition." *IEEE Transactions on Software Engineering*, Vol, No. 1 (January 1977) pp. 6-15.

Sibley, E. H. "The Evolution of Approaches to Information Systems Design Methodology." In T. W. Olle, H. G. Sol and A. A. Verrijn-Stuart (eds.) *Information Systems Design Methodologies: Improving the Practice*. Amsterdam: Elsevier Science Publishers B.V. (North-Holland), *IFIP 1986*, pp. 1-18.

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

Wand, Y. "An Ontological Foundation for Information Systems Design Theory." In B. Pernici and A. A. Verrijn-Stuart (eds.), *Proceedings of the IFIP WG 8.4 Working Conference on Office Information Systems: The Design Process*, Linz, Austria, August 1988.

Wand, Y. "A Proposal for a Formal Model of Objects." In F. Lochovsky and W. Kim (eds), *Object-Oriented Languages, Applications, and Databases*, Reading, MA: Addison-Wesley Publishing Co.

Wand, Y., and Weber, R. "A Deep Structure Theory of Information Systems." Working Paper 88-MIS-003, Faculty of Commerce and Business Administration, The University of British Columbia, March 1988a.

Wand, Y., and Weber, R. "A Model of Control and Audit Procedure Change in Evolving Data Processing Systems." Unpublished, March 1988b (revised).

Wand, Y., and Weber, R. "A Ontological Model of an Information System." Unpublished, July 1988c.

Weber, R. "Toward a Theory of Artifacts: A Paradigmatic Base for Information Systems Research." *Journal of Information Systems*, Spring 1987, pp. 3-19.

Weber, R. *EDP Auditing*, Second Edition. New York: McGraw-Hill Book Company, 1988.

Warnier, J. D. *Logical Construction of Programs*. New

York: Van Nostrand Reinhold, 1974.

Verheijen, G. M. A., and Van Bekkum, J. "NIAM: An Information Analysis Method." In T. W. Olle, H. G. Sol and A. A. Verrijn-Stuart (eds.), *Information Systems Design Methodologies: A Comparative Review*. Amsterdam: North Holland, IFIP, 1982.

Yourdon, E., and Constantine, L. L. *Structured Design*. Englewood Cliffs, NJ: Prentice-Hall, 1979.

9. ENDNOTES

1. In formal notation: if $s' = L(s)$ then $L(s') = s'$, or: $L(L(s)) = L(s)$. A state that satisfies $L(s) = s$ is a *stable state* for which $L(s) \neq s$ is an *unstable state.*

2. To avoid undefined states, we assume for every two events: If e_1 happens before e_2, the response to e_1 also precedes e_2.

3. In our formal notation, the external event is $e = <s_1,s'>$, $L(s') \neq s'$, and the internal event is $r = <s',s_2>$, $L(s') = s_2$.

4. More formally: A mapping exists between the state space of the real system and the state space of the information system: $S \leftrightarrow> M$. This mapping is exhaustive for both sets ("onto" mapping). We assume, henceforth, that $|rep(s)| = 1$ for every state.

5. Formally: for every state of the real system, $s \in S$, $rep(L(s)) = P(rep(s))$.

6. Formally: let $e \in E$ be an external event in the real system, $e = <s,s'>$, and assume that the information system state is $m = rep(s)$. The real system will generate an external event $t \in T$ for the information system such that the latter system will be in $m' = rep(s')$; namely, $t = <m,m'> = <rep(s),rep(s')>$.

7. In formal notation: $C(e) = 1 <=> e \in E_L$ (C is the characteristic function of the subset E_L).