

## *On the ontological expressiveness of information systems analysis and design grammars*

Y. Wand and R. Weber\*

*Faculty of Commerce and Business Administration, The University of British Columbia,  
2053 Main Mall, Vancouver, B.C. Canada V6T 1Z2, and \*Department of Commerce, The  
University of Queensland, Australia 4072*

**Abstract.** *Information systems analysis and design (ISAD) methodologies provide facilities for describing existing or conceived real-world systems. These facilities are ontologically expressive if they are capable of describing all real-world phenomena completely and clearly. In this paper we formally examine the notion of the ontological expressiveness of a grammar and discuss some of its implications for the design and use of ISAD methodologies. We identify some generic ways in which ontological expressiveness may be undermined in a grammar and some potential consequences of these violations. We also examine ontological expressiveness within the context of some other desirable features that might be considered in the design of ISAD methodologies.*

**Keywords:** clarity, completeness, expressiveness, grammar, IS development, ontology.

### INTRODUCTION

A large number of methodologies have now been developed to assist with the analysis, design and implementation of information systems (Olle *et al.*, 1988). Various attempts have been made to better understand their nature and to evaluate their relative strengths and weaknesses. They have been studied, for example, via feature analyses, case studies, experiments and field studies (e.g. Olle *et al.*, 1982; Floyd, 1986; Shoval & Even-Chaime, 1987; Batra *et al.*, 1990). Researchers appear generally disillusioned, however, with the progress made so far. Many now call for better theory to guide the empirical work (Bubenko, 1986; Iivari, 1986). The need is twofold. First, fundamental information systems concepts that underlie the methodologies need to be clarified and defined precisely (Lindgreen, 1990). Second, theoretical bases need to be developed that allow different features of methodologies to be better understood and evaluated.

In prior work, we have attempted to address both concerns. In Wand & Weber (1990a), we formally define and articulate a number of fundamental information systems concepts, e.g. system, subsystem, input, output, environment, coupling, level structure and decomposition. In Wand & Weber (1990b), we provide a summary description of three theoretical models we have developed to evaluate how well information systems analysis and design (ISAD) methodologies facilitate the accomplishment of three major tasks that must be undertaken during the analysis, design and implementation of information systems:

1 *Representation of the real world.* A methodology must possess features that enable users to construct a representation of their view of the real world.<sup>1</sup> A user's view may reflect existing real-world phenomena (e.g. the way employees are currently compensated in an organization) or imagined real-world phenomena (e.g. consumer behaviour if prices were to change). We have proposed a model that specifies a set of ontological constructs that ISAD methodologies must be able to describe. Otherwise, they may not be able to capture the meaning that users ascribe to the real-world phenomena because some phenomena cannot be represented.

2 *State tracking of the real world.* An information system must faithfully track changes in the existing or imagined real-world phenomena it is intended to model.<sup>2</sup> In some cases these changes may be transformations of matter or energy – for example, the production of a finished good from raw materials. In other cases they may be transformations of information – for example, transmission of messages in a data communications system. In this latter type of case, the information system models (simulates) another information system. Such a simulation can be used to study, say, the impact of changes in the data communications system on response time.<sup>3</sup> For both types of cases, however, we have proposed a model that articulates the conditions under which an information system will be a faithful state-tracking mechanism. An ISAD methodology must contain constructs that allow information systems to be developed that ensure that these conditions are fulfilled.

3 *Decomposition of the real-world model.* We argue a 'good' information system must be structured so that it is well decomposed. A good decomposition, in turn, must reflect the structure and dynamics of the represented real system. Otherwise, the script that describes the information system will convey the meaning of the real-world system it represents less effectively.

<sup>1</sup>Whether a user's view of the real world reflects objective reality of socially constructed reality (Klein & Hirschheim, 1987; Lyytinen, 1987a; Iivari, 1991) does not affect our model. We take the view as given, however it is formulated, and model it accordingly.

<sup>2</sup>The real-world system is the *represented* system, and the information system that models the real-world system is the *representing* system. Of course, the information system (the representing system) is itself part of the real world, but not that part of the real world being represented by the information system (i.e. it cannot represent itself).

<sup>3</sup>We may develop an information system to simulate another system, to replace an existing information system, to enhance an existing information system or to implement an information system where none previously existed. In all cases, however, the information system *tracks* or *represents* real-world phenomena, some of which may be information systems phenomena themselves. In our view, the essence of information is that it is a *representation* of a real-world phenomena that we use to avoid the costs of having to observe the real-world phenomena directly (see also Kary, 1999). In the absence of an order-entry system, for example, we would have to regularly ask customers directly whether they wished to purchase merchandise or services. Similarly, in the absence of a simulation of information processing in a data communications system, we might have to incur costly changes to the system to evaluate how these changes affect response time.

In this light, we have proposed a set of necessary (but not sufficient) requirements that an information system must fulfil if it is to be well decomposed. An ISAD methodology must contain constructs which allow information systems to be developed to ensure these requirements are met.

In this paper, we develop certain aspects of our first model, namely the representation model. In particular, we focus on the *grammars* that ISAD methodologies provide to describe various features of the real world.<sup>4</sup> We address the issue of how these grammars might be evaluated to determine whether they are *ontologically expressive*. By *ontological expressiveness*, we mean that an ISAD grammar can be used to describe all ontological constructs completely and clearly.<sup>5</sup> We develop this notion in the analysis below.

The remainder of the paper proceeds as follows. The second section briefly describes how we view ISAD methodologies within the representational model. The third, fourth and fifth sections outline some basic foundations that we need to be able to define precisely ontological expressiveness. The sixth and seventh sections develop the two major aspects of ontological expressiveness we examine: ontological completeness and ontological clarity. The eighth section discusses the relationship between the extensibility of an ISAD grammar and ontological completeness. Finally, the ninth section presents future research directions and our conclusions.

## A REPRESENTATIONAL VIEW OF ISAD METHODOLOGIES

The process of building an information system can be conceived as constructing a mapping of certain real-world phenomena into a script (generated by some programming language) that can be read, interpreted and executed by a machine (Computer Science and Technology Board, 1990). Since the mapping from the real world to the program script can be complex, it is usually done in stages (Fig. 1). Initially, scripts are generated that focus on capturing important real-world characteristics. For example, an entity-relationship diagram may be prepared to model real-world entities and their relationships. However, as the real-world characteristics become better understood, new scripts are generated that reflect the constraints of the machine world in which the information system must operate. For example, entity-relationship diagrams are mapped onto relations, which in turn are mapped onto record structures, which in turn are mapped onto access paths.

<sup>4</sup>A grammar generates a language, which is a set of strings over some alphabet. Syntactically correct strings are sentences of the language (see Sudkamp, 1988). ISAD grammars often generate languages that comprise a set of graphical strings, e.g. concatenations of data flow diagram symbols. In these grammars, sentences provide a graphical representation of some real-world phenomena.

<sup>5</sup>The notion of ontological *completeness* parallels the notion of expressive power in programming languages (Halpern & Wimmers, 1987). Oei *et al.* (1992, p. 5) also address the notion of the expressive power of a modelling technique in the context of *completeness*: 'The expressive power of a modelling technique is a measurement of *what* can be described, i.e. the possible universes of discourse that can be described.' Besides expressive completeness, however, we are also concerned with expressive *clarity*: the notion that an ISAD grammar and the scripts (strings) it produces can be interpreted unambiguously.

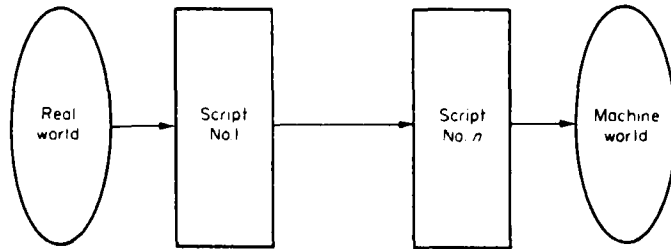


Figure 1. Mapping the real world via scripts to the machine world.

The scripts used in the process of building an information system are generated via *grammars*.<sup>6</sup> All ISAD methodologies provide at least one grammar that permits scripts considered useful in the process of building an information system to be generated. For example, the structured analysis and design methodology provides a grammar for generating data flow diagram scripts to describe certain features of the real-world system. In light of substantial research and experience, however, it is now well known that the quality of these scripts as a representation of real-world characteristics can vary considerably (Lyytinen, 1985, 1987b).

Our representation model provides a basis for addressing the question of why information system scripts might differ in terms of how well they represent real-world phenomena. In short, we argue that a grammar will not be able to generate scripts describing all real-world characteristics that might be of interest when information systems are built unless it contains *constructs* that enable it to model any real-world phenomenon in which a user is interested. This issue addresses the *completeness* of a grammar. Moreover, the grammar will not produce *clear* scripts unless the mapping from the grammatical constructs to the real-world constructs is straightforward. We need a theory or model, therefore, that tells us what real-world constructs a grammar must be able to describe. Moreover, we need a theory or model that allows us to evaluate the clarity of the mappings between the grammatical constructs and the real-world constructs. The following two sections address these issues.

## ONTOLOGICAL FOUNDATIONS

To determine the set of real-world constructs that an ISAD grammar must be able to describe, we have turned to the discipline of ontology. Ontology is a branch of philosophy concerned with articulating the nature and structure of the world.

In choosing the set of real-world constructs that an ISAD grammar must be able to describe, we have relied upon a particular ontological model developed by Bunge (1977, 1979) (see also Wand & Weber, 1990a). We chose to employ, adapt and extend Bunge's ontology for

<sup>6</sup>Some ISAD grammars are formal because they are specified precisely (e.g. Z and Lotus); others are semiformal because they are specified partially (e.g. NIAM's grammar for conceptual schema design and grammars that generate data flow diagrams and entity-relationship models); still others are informal because they are relatively unspecified (e.g. 'unconstrained' English).

several reasons. First, in our view his ontology is better developed and better formalized than any we have encountered so far. In this regard, Agassi (1990, p. 120) recently evaluated the importance of Bunge's ontology and concluded: '... philosophy and science will never be the same again'. Second, Bunge models the world as a world of systems. Thus, he is concerned with concepts that are fundamental to the computer science and information system domains. In this regard, Bochenski (1990, p. 99) comments, somewhat acidly: 'One great merit of Mario Bunge is surely that he made the study of the system leave the computer science ghetto, to which it was so often confined, and introduced it into the science to which it belongs, namely to philosophy.' Third, we believe we have been able to use Bunge's ontology to obtain useful theoretical and practical results. Rather than engaging in woolly philosophical polemics, we seek instead to produce concrete outcomes using Bunge's ontology and to judge these outcomes by their usefulness.

Table 1 provides an overview of the ontological (real-world) constructs derived from Bunge's ontology that we have used in our work (see, e.g., Wand & Weber, 1989b). In the context of this paper, however, our choice of this specific ontology and its constructs can be viewed as exemplars, chosen for their high level of rigour and completeness. We believe that the notion of ontological expressiveness can be applied to evaluate any ISAD grammar, irrespective of the ontology adopted.<sup>7</sup> The results produced using different ontologies to account for the strengths and weaknesses of ISAD grammars can then be evaluated. The remainder of the paper proceeds with this view in mind.

## MAPPING FOUNDATIONS

To evaluate whether a grammar provides a *clear* representation of a real-world construct, we rely on basic notions from the mathematics of mappings. In the analysis below, we focus on two sets: the set of real-world constructs we obtain from the ontological model; and the set of grammatical constructs we obtain from a description of the grammar. We are concerned with two mappings between these two sets: first, a *representation mapping*, which describes whether and how a real-world construct is represented via a grammatical construct; and, second, an *interpretation mapping*, which describes whether and how a grammatical construct stands for a real-world construct (Fig. 2). The former mapping addresses the issue of whether real-world constructs can somehow be represented by a grammar (the issue of ontological completeness). The latter mapping, on the other hand, addresses the issue of what each grammatical construct means in terms of the real-world constructs (the issue of ontological clarity).<sup>8</sup> In short, we will argue that completeness is achieved when the representation mapping is *total*; on the other

<sup>7</sup>Indeed, we would be pleased to see our colleagues use our approach with other ontologies they might deem useful.

<sup>8</sup>Ultimately, users must also be able to map back from the operational information system to reality. The extent to which they can accomplish this mapping will depend upon how clearly the relevant real-world constructs can be represented by the grammar that generates the final, machine-executable script.

Table 1. Ontological constructs in the Bunge–Wand–Weber representational model

Ontological construct	Explanation
Thing	A thing is the elementary unit in our ontological model. The real world is made up of things. A composite thing may be made up of other things (composite or primitive)
Properties	Things possess properties. A property is modelled via an <i>attribute</i> function that maps the thing into some value. A property of a composite thing that belongs to a component thing is called a <i>hereditary</i> property. Otherwise it is called an <i>emergent</i> property. A property that is inherently a property of an individual thing is called an <i>intrinsic</i> property. A property that is meaningful only in the context of two or more things is called a <i>mutual</i> or <i>relational</i> property
State	The vector of values for all attribute functions of a thing is the state of the thing
Conceivable state space	The set of all states that the thing might ever assume is the conceivable state space of the thing
State law	A state law restricts the values of the properties of a thing to a subset that is deemed lawful because of natural laws or human laws. A law is considered a property
Lawful state space	The lawful state space is the set of states of a thing that comply with the state laws of the thing. The lawful state space is usually a proper subset of the conceivable state-space
Event	An event is a change of state of a thing. It is effected via a transformation (see below)
Event space	The event space of a thing is the set of all possible events that can occur in the thing
Transformation	A transformation is a mapping from a domain comprising states to a co-domain comprising states
Lawful transformation	A lawful transformation defines which events in a thing are lawful
Lawful event space	The lawful event space is the set of all events in a thing that are lawful
History	The chronologically ordered states that a thing traverses are the history of the thing. The two things are said to be copied or interact
Coupling	A thing acts on another thing if its existence affects the history of the other thing. The two things are said to be coupled or interact
System	A set of things is a system if, for any bipartitioning of the set, couplings exist among things in the two subsets
System composition	The things in the system are its composition
System environment	Things that are not in the system but interact with things in the system are called the environment of the system
System structure	The set of couplings that exist among things in the system and among things in the system and things in the environment of the system is called the structure of the system
Subsystem	A subsystem is a system whose composition and structure are subsets of the composition and structure of another system
System decomposition	A decomposition of a system is a set of subsystems such that every component in the system is either one of the subsystems in the decomposition or is included in the composition of one of the subsystems in the decomposition
Level structure	A level structure defines a partial order over the subsystems in a decomposition to show which subsystems are components of other subsystems or the system itself
Stable state	A stable state is a state in which a thing, subsystem, or system will remain unless forced to change by virtue of the action of a thing in the environment (an external event)
Unstable state	An unstable state is a state that will be changed into another state by virtue of the action of transformations in the system
External event	An external event is an event that arises in a thing, subsystem or system by virtue of the action of some thing in the environment on the thing, subsystem or system. The before-state of an external event is always stable. The after-state may be stable or unstable

Continued overleaf

Table 1. Continued

Ontological construct	Explanation
Internal event	An internal event is an event that arises in a thing, subsystem or system by virtue of lawful transformations in the thing, subsystem or system. The before-state of an internal event is always unstable. The after-state may be stable or unstable
Well-defined event	A well-defined event is an event in which the subsequent state can always be predicted given that the prior state is known
Poorly defined event	A poorly defined event is an event in which the subsequent state cannot be predicted given that the prior state is known
Class	A class is a set of things that possess a common property
Kind	A kind is a set of things that possess two or more common properties

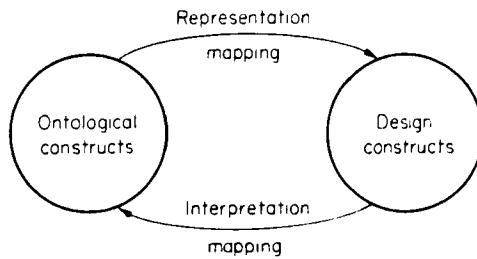


Figure 2. Modelling of the real world and interpreting the model.

hand, clarity is achieved when the interpretation mapping is *total* and *one-to-one*. We develop these ideas below.

### SOME BASIC, FORMAL NOTIONS OF GRAMMARS

If we are to rigorously compare constructs in an ISAD grammar against constructs in an ontological model, we need some common formalism in which both can be expressed. The approach we adopt is to use the concepts of formal languages.










We begin with the notion of a grammar. Formally, a grammar can be defined as follows:

#### Definition 1

A grammar,  $G$ , is a four-tuple,  $G = (V, T, S, P)$ , where  $V$  is a finite set of elements called *variables* or *non-terminals*,  $T$  is a finite set of elements called *terminals*,  $V \cap T = \emptyset$ ,  $S$  is an element in  $V$  called the *start symbol* and  $P$  is a finite set of *productions*, where a production is an ordered pair  $(\alpha, \beta)$ ,  $\alpha \rightarrow \beta$ , and  $\alpha, \beta$  are elements of  $V \cup T$ . At least one  $\alpha$  must be an element of  $V$ .

Table 2. Backus–Naur form (BNF) representation of a grammar for generating data flow diagrams

---

<data flow diagram>::=	<transition list>	
<transition list>::=	( <transition>   <transition> <transition list> )	
<transition>::=	( <terminator> <data flow> <process>	
	<datastore> <data flow> <process>	
	<process> <data flow> <process>	
	<process> <data flow> <terminator>	
	<process> <data flow> <data store> )	
<terminator>::=		
<data flow>::=		
<process>::=		
<data store>::=		

---

To illustrate these different concepts on the context of an ISAD grammar, Table 2 shows the Backus–Naur form representation of a grammar designed to generate data flow diagrams (DFDs).<sup>9</sup> The *terminals* in the grammar comprise the fundamental elements from which sentences can be built. In the DFD grammar, the terminals are the terminator symbol, data flow symbol, process symbol, and data store symbol. The *variables* in the grammar comprise meaningful strings of terminals. One of the variables in the DFD grammar, <transition list>, represents particular concatenations of three of the four terminals in the grammar – for example, an instance of <transition list> might be (<terminator><data flow><process>). This instance stands for a particular (sub)graph showing a data flow emerging from a terminator to enter a process symbol. The *start symbol* in the grammar is the other variable, namely, data flow diagram. It represents the entire data flow diagram, which in turn represents some subset of the real world to be modelled. *Productions* show how variables in the grammar are replaced by strings of a specified form. In the DFD grammar, <data flow diagram> → <transition list> is a production.

Note that the grammar shown in Table 2 is not the only grammar that might be defined to generate data flow diagrams. A different grammar might have different variables. For example, rather than have a variable named <transition list> that represents particular concatenations of three terminals, another grammar might define two-terminal concatenations instead:

<input>::=( <data flow> <data store> | <data flow> <process> | <terminator> <data flow> )  
 <output>::=( <data store> <data flow> | <process> <data flow> | <data flow> <terminator> )

This grammar would also need productions defining how a data flow diagram could be composed from these variables.

<sup>9</sup>Little work seems to have been undertaken on formalizing ISAD grammars. Nevertheless, see Adler (1988) and Parent & Spaccapietra (1985). See also Sudkamp (1988) for a description of Backus–Naur form.



Note, also, that the grammar shown in Table 2 does not capture all the rules needed to define a lawful data flow diagram.<sup>10</sup> For example, the user of the grammar has to know that two transitions can be concatenated only when the ending terminal of the first transition equals the beginning terminal of the second transition (e.g. if the first triple ends with a process, the adjacent triple must begin with a process). Similarly, the user has to know that a lawful data flow diagram begins and ends with a terminator.<sup>11</sup>

From our viewpoint of analysing the ontological expressiveness of a grammar, the variables and terminals in grammars have special significance (see, for example, Meyer, 1990, p. 47).

### Definition 2

A *construct* is an element of the union of the sets  $V$  and  $T$ . A construct describes a set of objects or things that have common properties. These objects or things are the *specimens* of the construct.

To illustrate these notions, consider again the grammar shown in Table 2 to generate data flow diagrams and a specific instance of a data flow diagram generated via this grammar to describe, say, a library system. The process symbol (terminal) and the transition variable are constructs within the grammar. A specimen of the former construct would be a process symbol showing a particular activity that updates the status of a library book from on-the-shelf to on-loan. A specimen of the latter construct would be a request (data flow) from a borrower (terminator) entering a process that updates the borrower's record of book borrowings.

Our representation model can also be expressed as a formal grammar. Indeed, Meyer (1990, p. 297) argues that any theory or model can be conceived in this way: 'A theory may be viewed as a formal language, or more properly a metalanguage, defined by syntactic and semantic rules.' In light of our arguments above, the constructs in an ontological grammar are the terminals and variables that generate descriptions of the real world in ontological terms. *Primitive* or atomic constructs within the representation model are terminals within the grammar. *Derived* constructs are the variables within the grammar.

At this time we have not specified our adaptation of Bunge's ontological model in our representation model as a formal grammar. Nevertheless, it will be sufficient for our analysis below if we illustrate the form such a grammar might take. Recall that Table 1 shows the ontological constructs we use in our representation model. Two terminals in the representation grammar would be defined via the *primitive* ontological constructs of 'things' and 'transformation'. A variable would be used to stand for the *derived* ontological construct of an 'event'. It might be defined as  $\langle \text{event} \rangle ::= \langle \text{state} \rangle \langle \text{transformation} \rangle \langle \text{state} \rangle$ . The start symbol in the grammar might be named  $\langle \text{ontological script} \rangle$ .

<sup>10</sup>In part this limitation reflects the fact that we have used a context-free grammar. More-powerful grammars, such as context-sensitive grammars, allow more rules to be specified (Sudkamp, 1988).

<sup>11</sup>We are assuming that the initial data flow always arises from a terminator and that the final data flow always enters a terminator.

In summary, formal grammars provide a *common* way of describing both our representation model and the facilities included in an ISAD methodology for describing existing or conceived real-world systems. They formally specify the constructs that exist in both the representation model and an ISAD grammar such that a more precise comparison can be undertaken between the two sets of constructs. In our analysis below, we focus on comparing the constructs in a grammar that might be used to describe our representation model (henceforth the ontological constructs) and one that might be used to define the representation facilities in an ISAD methodology (henceforth the design constructs). We show how these comparisons address the notion of ontological completeness and ontological clarity.

### ONTOLOGICAL COMPLETENESS

Recall that we defined ontological completeness informally in terms of whether an ISAD grammar can represent the same information about the real world that can be represented in our representation model. In the context of the concepts we described above, evaluating ontological completeness means that we search for a correspondence between each terminal and variable in a grammar describing our representation model (the ontological grammar) and each terminal and variable in an ISAD grammar. We focus, therefore, on the mapping from real-world constructs to grammatical constructs – the representation mapping (Fig. 2).

In this light, let  $O_c$  be the set of constructs in our ontological grammar, and let  $M_c$  be the set of constructs in an ISAD grammar  $M$ . We define *ontological completeness* and *ontological incompleteness* (or *construct deficit*) as follows:

#### Definition 3

Let  $f$  be the mapping from the set of ontological constructs  $O_c$  to the set of constructs  $M_c$  of an ISAD grammar  $M$ .<sup>12</sup> Then  $M$  is ontologically *complete* if the mapping  $f$  is *total*. Otherwise, *ontological incompleteness* or *construct deficit* exists.

Figures 3 and 4 illustrate the notions of ontological completeness and ontological incompleteness or construct deficit. Note the condition that differentiates the two situations. Providing the mapping exhausts the domain, the grammar is ontologically complete; otherwise, it is incomplete. In other words, a grammar is ontologically complete if and only if it provides at least one grammatical construct for every ontological construct.

Unfortunately, we are unable to give any examples of ISAD grammars that are ontologically complete according to the list of constructs shown in Table 1. All ISAD grammars we have examined so far exhibit ontological incompleteness or construct deficit. For instance, in the grammar used to generate entity-relationship models, there is no grammatical construct to

<sup>12</sup>We assume  $f$  is given implicitly as part of the definition of the design grammar. In other words, the semantics of each construct in the grammar can be determined. The specification can be undertaken either via intension (by rule) or by extension (by an explicit list).

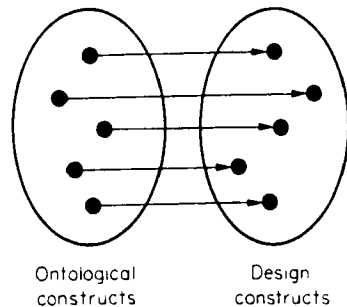


Figure 3. Ontological completeness.

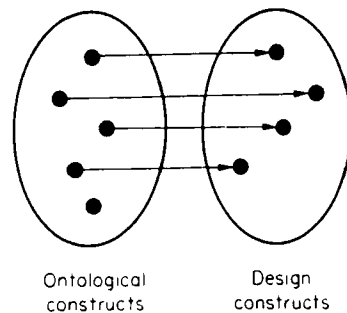


Figure 4. Ontological incompleteness or construct deficit.

represent the ontological construct of a transformation; in the grammar used to generate NIAM (Nijssen's information analysis method) conceptual schema diagrams (Nijssen & Halpin, 1989), there is no construct to represent the ontological construct of an event; in the grammar used to generate data flow diagrams, there is no construct to represent a state law.

We argue that, *prima facie*, ontological incompleteness or construct deficit is undesirable. Users of a grammar with ontological incompleteness or construct deficit are unable to represent all real-world phenomena that might interest them. As a result, we have observed two strategies they employ to try to overcome the problems that arise. First, they invoke another design construct to stand for the unrepresented ontological construct. For example, the entity-relationship model provides no construct for representing the ontological construct of an event. Nevertheless, we have observed users employing the entity construct to stand for both things and events. We argue below that using one design construct to represent multiple ontological constructs produces a new set of problems because the design construct becomes semantically overloaded.<sup>13</sup>

<sup>13</sup>By using an entity construct to represent an event and marking it as such, designers effectively create a new construct (an entity symbol with the word 'event'). The meaning of the relationships attached to it will change ('event modifies entity' or 'event generates event'). Someone reading the diagram still must bring to bear knowledge outside the diagram to interpret it.

Second, users will employ other grammars to represent the ontological constructs where deficit exists. For example, data flow diagrams might be used in conjunction with NIAM to compensate for NIAM's inability to represent events. Similarly, it is well known that data flow diagrams and entity-relationship diagrams complement each other as a manifestation of their relative strengths and weaknesses in representing real-world phenomena. Unfortunately, using multiple grammars to describe the real world may produce incompatibilities in the representations generated by the grammars. As a result, users of the grammars may become confused and produce erroneous information systems designs (see especially Coad & Yourdon, 1991, p. 25).

Given most, if not all, ISAD grammars seem to possess ontological incompleteness or construct deficit, as researchers we must try to account for why this situation has arisen. Perhaps it reflects the fact that the designers of ISAD grammars have traded expressive power for simplicity. As additional constructs and production rules are added to a grammar to enhance expressive power, unfortunately the grammar becomes more complex to use. Perhaps, also, designers of grammars believe that information systems development complexity will be reduced if individual grammars facilitate modelling only a subset of real-world phenomena. Whatever the reason, an analysis of the ontological completeness of a grammar still provides a way of structuring discourse and research to address these issues.

## ONTOLOGICAL CLARITY

The expressive power of an ISAD grammar is also a function of how 'clearly' each construct in the grammar represents an ontological construct. To evaluate the level of ontological clarity that exists in an ISAD grammar, we focus on the interpretation mapping (Fig. 2). We are concerned with the question of how to interpret the *meaning* of each design construct.

The subsections below examine three situations that can undermine design construct clarity. We first define and discuss the nature of each problematical situation. Then we provide some examples of each from existing ISAD grammars and outline some research issues.

### Construct overload

Construct overload occurs when one design construct maps into two or more ontological constructs (Fig. 5).<sup>14</sup> More formally:

#### Definition 5

Let  $g$  be the mapping from the set of design constructs  $M_c$  of an ISAD grammar  $M$  to the set of ontological constructs  $O_c$ .<sup>15</sup> Then  $M$  has *construct overload* if  $g$  is a one-to-many mapping.

<sup>14</sup>Note, Figs 5–7 show three situations that we argue undermine ontological clarity in the context of a grammar that is ontologically complete. Clearly, these situations can also exist with grammars that are ontologically incomplete.

<sup>15</sup>Again, we assume  $g$  is given as part of the specification of the grammar.

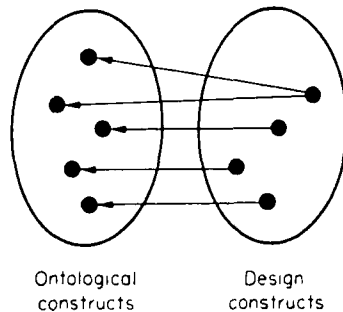


Figure 5. Ontological completeness with construct overload.

As an example of construct overload, Weber & Zhang (1991) found that the grammatical construct called 'entity' in NIAM represents two ontological constructs in our representation model – namely, a thing and a property of a thing. Thus, the NIAM grammar entity design construct is 'overloaded' because it stands for more than one ontological construct.

Similarly, in the relational model a relation can stand for the ontological constructs of a thing and a mutual property (a property defined over two or more things) in our representation model. For example, a book in a library may be borrowed by a borrower. In our ontological model the book and the borrower are things. The fact that a book may be borrowed by a borrower is a mutual property because it is an assertion made about two things. It is both a property of the book (the book may be borrowed) and a property of the borrower (the borrower may borrow the book). If relations in the relational model are in third normal form, the book and borrower will be represented by relations. Also, the mutual property (relationship) reflecting that books may be borrowed by borrowers will be represented as a relation. In short, the relational grammar design construct of a relation is overloaded because it stands for more than one ontological construct.

We argue that construct overload is undesirable, *prima facie*, because users of an ISAD grammar must bring to bear other knowledge, which they might not always possess, to determine which ontological construct is being represented by the design construct. For example, when analysts or designers first sight a relation in a relational model, it is not immediately clear whether the relation is representing an entity or a mutual property (relationship). Analysts or designers must employ knowledge not manifested in the relational model to make this determination. For example, they first use commonsense reasoning to identify the keys in the relation. They then check how many keys exist and the nature of the domains in the relation. In short, construct overload means that an ISAD grammar loses expressive power because design constructs no longer clearly manifest the real-world semantics.

When construct overload occurs, we claim that in due course a new design construct will be introduced into the ISAD grammar to eliminate the overload. For example, early versions of the entity-relationship model used the relationship symbol (diamond) to stand for both a

relationship and the IS-A (subset hierarchy) constructs. In our representation model, a relationship is a mutual property, and the IS-A construct reflects the ontological notion of a kind (see Table 1). Thus, the relationship symbol is overloaded. In light of our model, therefore, we can understand why later versions of the entity-relationship model have introduced a new symbol to stand for the IS-A notion (Teorey *et al.*, 1986).

Given the apparent prevalence of construct overload among ISAD grammars, again, as researchers, we would like to account for why this situation exists. Once more, designers of ISAD grammars may be trading off expressiveness against simplicity. Introducing construct overload into a grammar reduces the number of constructs in the grammar and makes it simpler. However, expressive power is lost, because the meaning of the constructs is not always clear. Using our model to identify the nature and extent of construct overload in a grammar should enable the trade-offs to be evaluated more carefully.

Construct overload may also reflect efforts by designers of grammars to undertake *abstraction* on the ontological constructs. Perhaps they see common properties among several ontological constructs. In this light they provide a single design construct to represent each ontological construct in the class or kind. For example, the designers of a grammar may conceive of a relationship as any type of association between two things. Accordingly, they may view both composition (part-of) and class membership (IS-A) as subtypes of the general notion of a relationship. However, even if construct overload reflects abstraction, we argue that our model still provides a way of evaluating whether the abstraction undertaken is likely to be problematical.

### Construct redundancy

The second situation that might arise we call construct *redundancy*. Here two or more design constructs are used to represent a single ontological construct (Fig. 6). More formally:

#### Definition 6

Let  $g$  be the mapping from the set of design constructs  $M_c$  of an ISAD grammar  $M$  to the set of ontological constructs  $O_c$ . Then  $M$  has *construct redundancy* if  $g$  is a many-to-one mapping.

As with construct overload, construct redundancy establishes a *prima-facie* case of lack of ontological clarity with respect to the design construct. As a result, it may reflect that the expressiveness of an ISAD grammar has been undermined. Potentially several deleterious effects arise: users have to remember a larger number of design constructs; they may be uncertain about whether two or more design constructs truly stand for the same ontological construct; and a larger number of production rules must be remembered to accommodate the redundant design constructs.<sup>16</sup>

<sup>16</sup>Note, we distinguish construct redundancy from syntactic redundancy in a grammar. The data flow diagram construct of a process can be shown as both a circle and a rounded rectangle. This is an example of *syntactic* redundancy.

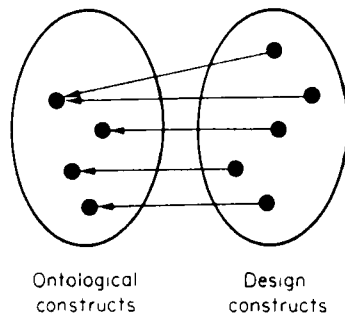


Figure 6. Ontological completeness with construct redundancy.

The notion of construct redundancy teases out some thorny issues that arise, however, when mapping from design constructs to ontological constructs. Consider, for example, four constructs in a grammar that generates physical data flow diagrams: a data flow, a data store, a process and a terminator. In prior work (Wand & Weber, 1989a), we argued that each of these four design constructs represents the ontological construct of a thing in an implemented information system: a data flow may represent a piece of paper that stands for a customer order; a data store may represent a filing cabinet in which customer orders are kept; a process may represent a clerk who handles customer orders; and a terminator may represent a customer who sends orders to an organization. Thus, four design constructs have been used to represent a single ontological construct, and at first glance we would conclude that construct redundancy exists.

Note, however, that one might conceive data flows, data stores, processes and terminators as *subtypes* of a more general construct that could be called 'thing'. For example, things that are terminators are distinguished from other things because they exist in the environment of the information system. This more general construct could be named in a grammar for data flow diagrams, and a choice production rule could then be specified to define it through a set of alternatives – namely, data flow, data store, process and terminator ( $\langle \text{thing} \rangle ::= \langle \text{data flow} \rangle \mid \langle \text{data store} \rangle \mid \langle \text{process} \rangle \mid \langle \text{terminator} \rangle$ ). This new grammatical variable 'thing' can then be mapped directly to the ontological construct of a thing.

Two issues now arise. First, we must ensure that the design construct called 'thing' *means* the same as the ontological construct called 'thing'. In the context of the above example, we must ensure that the four types of things represented by data flows, data stores, processes and terminators are exhaustive – that is, all things in an information system that we might want to model via a physical data flow diagram are an instance of one of these four types.<sup>17</sup> Otherwise, the *meaning* of the design construct and the *meaning* of the ontological construct are not the

<sup>17</sup>More formally, we seek to determine whether the union of the extensions of the subtypes equals the extension within the domain of the ontological construct. In other words, an instance of the ontological construct is also an instance of at least one of the subtypes. In this regard, it is important that the meaning of each subtype be defined clearly. Otherwise, ontological clarity will suffer.

same, and the former should not be mapped to the latter. Second, it is evident that some arbitrariness surrounds how we select the variables in a grammar<sup>18</sup> (we alluded to this problem above). The problem is that different grammars can be specified to generate the same language. Whereas one grammar might have construct redundancy because of the way variables are chosen, another might not.

To summarize, we believe construct redundancy will always show use of subtyping in a grammar (relative to the ontological constructs). Once construct redundancy has been identified, it is then important to check that all instances in the domain of the ontological construct are covered by one of the grammatical subtype constructs. If some instance of an ontological construct is not covered, we then have a problem of ontological incompleteness or construct deficit. In this regard, Weber & Zhang (1991) found several design constructs in NIAM that represented the ontological construct of a state law. At first glance, therefore, construct redundancy existed. It became clear, however, that the NIAM design constructs represented different *types* of state law (in other words, the NIAM grammar was employing subtypes). Nevertheless, they found that some types of state law could not be represented using the NIAM grammar. Thus, the NIAM grammar exhibits ontological incompleteness or construct deficit rather than construct redundancy.

### Construct excess

The third situation that may arise we call *construct excess*. Here a grammatical construct does not map into any ontological construct (Fig. 7):

#### Definition 7

Let  $g$  be the mapping from the set of design constructs  $M_c$  of an ISAD grammar  $M$  to the set of ontological constructs  $O_c$ . Then  $M$  has *construct excess* if  $g$  is a partial mapping.

We see three ways in which construct excess can be interpreted. First, the construct may reflect deficiency of the ontological model. In other words, the construct represents a real-world phenomenon that cannot be captured via the ontological constructs used in the evaluation for ontological completeness.

Second, construct excess may mean that a grammar contains a design construct for some type of real-world phenomenon that is supposed to be outside the domain of discourse it models. For example, logical data flow diagrams are supposed to model only the *logical* aspects of an information system. We interpret this objective to mean that logical data flow diagrams model the *representations* that carry meaning in an information system and not the physical components of the information system. Yet an external terminator in a logical data flow diagram stands for a *physical* thing – for example, a customer. In short, a logical data flow diagram is

<sup>18</sup>Nevertheless, we predict that the designers of a grammar choose their variables so that each variable represents some type of real-world construct. In other words, designers ascribe real-world meaning to the variables they specify.



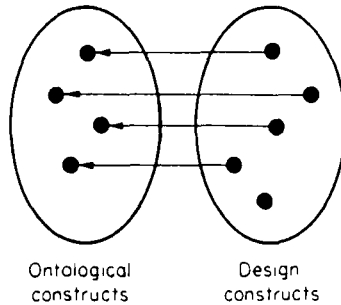


Figure 7. Ontological completeness with construct excess.

providing a design construct to model something that is supposedly outside its domain of discourse.

Earlier we argued that having multiple grammars to model different parts of the real world can lead to inconsistencies and incompleteness in the scripts produced. Nevertheless, if the designers of an ISAD grammar intend that it should model only some part of the real world, they should at least ensure that the constructs they provide in the grammar represent only those phenomena that fall within the domain of discourse covered by the grammar. Otherwise, ontological clarity again suffers, and we predict that users will become confused about the real-world meaning conveyed in the scripts generated via the grammar.<sup>19</sup>

Third, construct excess may reflect confusion between constructs in the modelled domain and constructs in the implemented information system. For example, consider the design construct of message passing in grammars that generate object-oriented scripts (Coad & Yourdon, 1991). Message passing may be meaningful in object-oriented programming (implementing the information system), but it is not meaningful when modelling the real-world domain (Parsons & Wand, 1991). For example, it is unlikely that we would conceive of a customer placing an order by sending a message to a product! Clearly a message might be sent to a salesperson, but the salesperson may not even be included in the application domain to be modelled. In the context of Table 1, we should be concerned with representing the ontological construct of coupling rather than the implementation construct of message passing. Again, we predict that ontological clarity suffers when modelling (design) constructs and implementation constructs are included in the one ISAD grammar.

#### SOME ISSUES OF EXPRESSIVE POWER AND EXTENSIBILITY

The extent to which an ISAD grammar achieves ontological completeness and ontological clarity is a measure of its *expressive power*. Accordingly, if two grammars each achieve the same measure of ontological completeness and clarity, we deem them to be *equally expressive*.

<sup>19</sup>Indeed, our experience is that new users of logical data flow diagrams are often confused about the difference between a terminator that has both input and output data flows and a process.

In the context of our representation model, both are capable of capturing the same set of semantics about the real-world system being represented with the same level of clarity.

Assuming that the ontological model used in an evaluation for completeness and clarity is 'valid', nothing can be done to improve a grammar's ontological clarity if it is deficient (short of changing the grammar). If construct overload, redundancy or excess exists, the grammar must be modified to add or delete design constructs and production rules. In effect, a new grammar must be produced. For example, recall that a new construct was added to entity-relationship grammars to represent the ontological notion of a class (the IS-A construct), thereby eliminating one form of overload on the relationship construct. This new construct could not be generated via existing production rules from existing constructs. Instead, it had to be added to the grammar.

However, a grammar that appears to be ontologically incomplete, may achieve completeness through being *extensible*. If a grammar is extensible, constructs in the grammar can be combined via production rules to form other constructs. For example, even if a grammar that generates data flow diagrams has no variable to represent the ontological construct of an event, it is extensible if it permits an input data flow, a process and an output data flow to be combined to represent the ontological construct of an internal event and a terminator and an input data flow to be combined to represent the ontological construct of an external event (see Table 1). Having generated these new constructs, we might then name them so that we are mindful of their ontological significance when we subsequently use the data flow diagram grammar to model the real world – for example:

```
<event>::=<internal event>|<external event>
<internal event>::=<data flow><process><data flow>
<external event>::=<terminator><data flow>
```

Therefore, because the data flow diagram grammar is extensible, we can create new grammatical constructs that correspond to ontological constructs that are not covered by the initial set of constructs defined in the grammar. From the viewpoint of ontological completeness, we are indifferent as to whether the needed constructs are defined at the outset or defined through extension. Indeed, in the interests of simplicity and flexibility, designers of grammars have incentives to provide only the minimal set of defined constructs yet allow maximum flexibility in construct creation through a few powerful production rules.

Unfortunately, the goals of expressive adequacy and simplicity are often in conflict. Additional constructs and production rules enhance expressive power at the cost of increased complexity. Moreover, as Bailes *et al.* (1992) point out, grammars may be equally expressive but not equally extensible. Some grammars permit new constructs to be defined via declaration; others require schemes of encoding and interpretation. It is generally believed that declarative extension is easier to comprehend and use than interpretive extension. In this regard, the example of an extension shown above for the ontological construct of an event should be easy to comprehend and use because it has been defined declaratively. On the other hand, if an event had to be defined via some procedural grammar, such as structured English, it will supposedly be harder to comprehend and use.

How an ISAD grammar achieves ontological completeness is therefore an important issue in assessing how easily it can be used to represent real-world phenomena. If an ontological construct is represented by a grammatical construct that is exceedingly complex (both syntactically and semantically) because it has been defined through interpretive extension, the 'semantic power' of the grammatical construct may be undermined. In other words, some of the real-world meaning of the construct may be lost because it is cognitively difficult for users to understand. In short, designers of ISAD grammars not only must strive for ontological completeness, they also must carefully evaluate how they will achieve ontological completeness with the grammatical constructs and production rules they choose.<sup>20</sup>

#### FUTURE RESEARCH DIRECTIONS AND CONCLUSIONS

In light of our analysis above, we are currently pursuing five research directions. First, to facilitate the use of our representation model in evaluating the expressive power of an ISAD grammar, we are seeking to describe it via a formal grammar. Second, we are continuing our evaluation of existing ISAD methodologies to determine whether the facilities they provide to describe existing and conceived real-world systems are ontologically complete and exhibit construct clarity. Third, in light of our evaluations of the expressive power of existing ISAD grammars, we are empirically testing our predictions about the strengths and weaknesses of these grammars for representing real-world systems. Fourth, we are using our representation model to predict which ISAD grammars are likely to work well together and which are likely to produce conflicting and redundant descriptions of real-world phenomena. Fifth, we are seeking to better understand the trade-offs that might be made between expressive power and simplicity in an ISAD grammar.

In summary, we believe that our approach provides the rudiments of a theory that will allow systematic design and appraisal of some important features of ISAD methodologies. In particular, we believe that we can begin to formally articulate the facilities that should be provided by ISAD methodologies if they are to be capable of describing the real world in a clear, straightforward fashion. We are hopeful, therefore, that in due course these results will affect how research on ISAD methodologies proceeds and how the design of ISAD methodologies is undertaken.

Finally, we reiterate that we believe that the approach we advocate in this paper to the evaluation of ISAD methodologies is general. While we have described the approach in the context of Bunge's ontological model, clearly other ontologies might be used (e.g. Guha & Lenat, 1990; Stamper, 1992). Indeed, we see substantial merit in conducting the evaluation using different ontologies. Alternative views of the world may give us better insights into how it should be modelled and what modelling capabilities should be provided by ISAD grammars.

---

<sup>20</sup>When designing grammars, clearly other issues must be considered. For example, what syntactical representations the designer chooses for different constructs may affect how easily the grammar can be learned and used. Moreover, aesthetics must be considered when designing grammars that produce graphs (Ding & Mateti, 1999).

## ACKNOWLEDGEMENTS

With the usual caveat, we are indebted to participants in a workshop at the University of Alberta and to Paul Bailes, Maria Orlowska, Kerry Raymond, Roly Sussex, Janice Thomas and an anonymous referee for helpful comments on earlier versions of this paper. The research described in this paper was supported in part by Canadian NSERC and SSHRC operating grants and by grants from the Australian Research Council and GWA Pty. Ltd.

## REFERENCES

- Adler, M. (1988) An algebra for data flow diagram process decomposition. *IEEE Transactions on Software Engineering*, **14**, 169–183.
- Agassi, J. (1990) Ontology and its discontent. In: *Studies on Mario Bunge's Treatise*, Weingartner, P. and Dorn, G.J.W. (eds), pp. 105–122. Rodopi, Amsterdam.
- Bailes, P.A., Ming, G. & Moran A. (1992) A dynamic subtypes extension of an expressively-complete functional language. Unpublished paper, Dept. of Computer Science, University of Queensland.
- Batra, D., Hoffer, J.A. & Bostrom, R.P. (1990) Comparing representations with relational and EER models. *Communications of the ACM*, **33**, 126–139.
- Bochenski, J.M. (1990) On the system. In: *Studies on Mario Bunge's Treatise*, Weingartner, P. and Dorn, G.J.W. (eds), pp. 99–104. Rodopi, Amsterdam.
- Bubenko Jr, J.A. (1986) Information system methodologies – a research review. In: *Information Systems Design Methodologies: Improving the Practice*, Olle, T.W., Sol, H.G. and Verrijn-Stuart, A.A. (eds), pp. 289–318. North-Holland, Amsterdam.
- Bunge, M. (1977) *Treatise on Basic Philosophy*. Vol. 3, *Ontology I: The Fumiture of the World*. Reidel, Boston, Massachusetts.
- Bunge, M. (1979) *Treatise on Basic Philosophy*. Vol. 4, *Ontology II: A World of Systems*. Reidel, Boston, Massachusetts.
- Coad, P. & Yourdon, E. (1991) *Object-Oriented Analysis*. Prentice-Hall, Englewood Cliffs, New Jersey.
- Computer Science and Technology Board (1990) Scaling up: a research agenda for software engineering. *Communications of the ACM*, **33**, 281–293.
- Ding, C. & Mateti, P. (1990) A framework for the automated drawing of data structure diagrams. *IEEE Transactions on Software Engineering*, **16**, 543–557.
- Floyd, C. (1986) A comparative evaluation of system development methods. In: *Information Systems Design Methodologies: Improving the Practice*, Olle, T.W., Sol, H.G. and Verrijn-Stuart, A.A. (eds), pp. 19–54. North-Holland, Amsterdam.
- Guya, R.V. & Lenat, D.B. (1990) Cyc: A midterm report. *AI Magazine*, **11**, 32–59.
- Halpern J.Y. & Wimmers, E.L. (1987) Full abstraction and expressive completeness for FP. *Proceedings of the Symposium on Logic in Computer Science*, pp. 257–271. IEEE Computer Society, Ithaca.
- Iivari, J. (1986) Dimensions of information systems research design: a framework for a long-range research program. *Information Systems*, **11**, 185–197.
- Iivari, J. (1991) A paradigmatic analysis of contemporary schools of IS development. *European Journal of Information Systems*, **1**, 249–272.
- Kary, M. (1990) Information theory and the *Treatise*. In: *Studies on Mario Bunge's Treatise*, Weingartner, P. and Dorn, G.J.W. (eds), pp. 263–280. Rodopi, Amsterdam.
- Klein, H.K. & Hirschheim, R.A. (1987) A comparative framework for data modelling paradigms and approaches. *The Computer Journal*, **30**, 8–15.
- Lindgreen, P. (ed.) (1990) *A Framework of Information Systems Concepts: Interim Report*. The IFIP WG 8.1 Task Group FRISCO, The Netherlands.
- Lyytinen, K.J. (1985) Implications of language for information systems. *MIS Quarterly*, **9**, 61–74.
- Lyytinen, K. (1987a) Two views of information modeling. *Information & Management*, **12**, 9–19.
- Lyytinen, K. (1987b) Different perspectives on information systems: problems and perspectives. *ACM Computing Surveys*, **19**, 5–46.
- Meyer, B. (1990) *Introduction to the Theory of Programming Languages*. Prentice-Hall, Hertfordshire, England.
- Nijssen, G.M. & Halpin, T.A. (1989) *Conceptual Schema and Relational Database Design: A Fact Oriented Approach*. Prentice-Hall, Sydney.

- Oei, J.L.H., van Hemmen, L.J.G.T., Falkenberg E. & Brinkemper, S. (1992) The meta model hierarchy: a framework for information systems concepts and techniques. Technical Report No. 92-17, Department of Information Systems, University of Nijmegen.
- Olle, T.W., Hagelstein, J., Macdonald, I.G. *et al.* (eds) (1988) *Information Systems Methodologies: A Framework for Understanding*. Addison-Wesley, Reading, Massachusetts.
- Olle, T.W., Sol, H.G. & Verrijn-Stuart, A.A. (eds) (1982) *Information Systems Design Methodologies: A Comparative Review. Proceedings of the CRIS 82 Conference*. North-Holland, Amsterdam.
- Parent, C. & Spaccapietra, S. (1985) An algebra for a general entity-relationship model. *IEEE Transactions on Software Engineering*, SE-11, 634–643.
- Parsons, J. & Wand, Y. (1991) The objects paradigm – two for the price of one. In: *Proceedings of the Workshop on Information Technologies and Systems*, pp. 308–319. Boston, Massachusetts.
- Shoval, P. & Even-Chaime, M. (1987) Data base schema design: an experimental comparison between normalization and information analysis. *Data Base*, 18, 30–39.
- Stamper, R.K. (1992) Signs, organizations, norms and information systems. In: *Proceedings of the Third Australian Conference on Information Systems*, MacGregor, R., Clarke, R., Little, S., Gould, T. and Ang, A. (eds), pp. 21–65. University of Wollongong.
- Sudkamp, T.A. (1988) *Languages and Machines: An Introduction to the Theory of Computer Science*. Addison-Wesley, Reading, England.
- Teorey, T.J., Yang, D. & Fry, J.P. (1986) A logical design methodology for relational databases using the extended entity-relationship model. *ACM Computing Surveys*, 18, 197–222.
- Wand, Y. & Weber, R. (1989a) An ontological analysis of systems analysis and design methods. In: *Information Systems Concepts – An In-Depth Analysis*. Falkenberg, E. and Lindgreen, P. (eds), pp. 79–107. North-Holland, Amsterdam.
- Wand, Y. & Weber, R. (1989b) A model of control and audit procedure change in evolving data processing systems. *The Accounting Review*, 64, 87–107.
- Wand, Y. & Weber, R. (1990a) An ontological model of an information system. *IEEE Transactions on Software Engineering*, 16, 1282–1292.
- Wand, Y. & Weber, R. (1990b) Towards a theory of the deep structure of information systems. In: *Proceedings of the Eleventh International Conference on Information Systems*, DeGross, J., Alavi, M. and Oppelland, H. (eds), pp. 61–71. Copenhagen.
- Wand, Y. & Weber, R. (1991) A unified model of software and data decomposition. In: *Proceedings of the Twelfth Annual International Conference on Information Systems*, DeGross, J., Benbasat, I., DeSanctis, G. and Beath, C.M. (eds), pp. 101–110. New York.
- Weber, R. & Zhang Y. (1991) An ontological evaluation of NIAM's grammar for conceptual schema design. In: *Proceedings of the Twelfth Annual International Conference on Information Systems*, DeGross, J., Benbasat, I., DeSanctis, G. and Beath, C.M. (eds), pp. 75–82. New York.

## Biography

**Yair Wand** has a DSc in Operations Research (Technion, Israel). He has held positions at the Faculty of Management, the University of Calgary, Canada, and the Faculty of Industrial Engineering, Technion, Israel. Presently, he is on faculty at the Management Information Systems Division, Faculty of Commerce, the University of British Columbia (Vancouver, Canada). His research interests are modelling of information systems and formal foundations of systems analysis and design.

**Ron Weber** is GWA Professor of Commerce at the University of Queensland, Brisbane, Australia. His primary research interests are in formal modelling of information systems, computer control and audit, and management of the information systems function. He is a Fellow of the Australian Computer Society, a Fellow of the Australian Society of Certified Practising Accountants and a Past President of the Accounting Association of Australia and New Zealand.