Information Systems = Databases + Interaction: Towards Principles of Information System Design

Dina Goldin[1]
University of Massachusetts, Boston
100 Morrissey Blvd., Boston, MA 02125
U.S.A
dqg@cs.umb.edu

Bernhard Thalheim Srinath Srinivasa[2]
Brandenburgische Technische Universität
Postfach 101344, D-03013 Cottbus
Germany
{thalheim,srinath}@informatik.tu-cottbus.de

April 11, 2000

Abstract

Information system (IS) design concerns different activities like conceptual modeling, database design, business process modeling and user modeling. However, despite the presence of an active research community that studies conceptual models for information systems, this term still lacks precise formal underpinnings. Unlike for databases, there is no agreement on what constitutes "principles of information systems." It would hence be desirable to provide formal underpinnings for information system design and to position it with respect to database design.

At first glance, it is not clear that the principles underlining the study of information systems should be different than those for databases. Any significantly advanced IS incorporates some type of a database. As a consequence, information system design addresses a number of database issues like conceptual modeling of data elements, metadata management, etc. On the other hand, any useful database system is actually an information system, providing additional services beyond simply maintaining data and running queries and updates. As a result, the distinction between issues related to databases and to information systems tend to get blurred.

In this paper we argue that the interactive aspect of information systems necessitates a fundamentally different set of design principles for an IS, as compared to conventional database design. We provide some promising directions for a formal study of IS models, based on the notion that interactive behaviors cannot be reduced to algorithmic behaviors.

Keywords: Information System, Interaction, Database design, Algorithmic behavior, Interactive behavior, Interaction Machines.

[1]Supported by NSF Grant #IRI-9733678.

[2]Supported by the German Research Society, Berlin-Brandenburg Graduate School in Distributed Information Systems (DFG grant no. GRK 316).

1 Introduction

Information Systems (IS) is a generic term referring to software systems that provide information based services. The notion of an information system has been addressed from various perspectives like managerial, technical, organizational, etc [3, 25].

However, despite the presence of an active research community that studies conceptual models for information systems, this term still lacks precise formal underpinnings. Unlike for databases, there is no agreement on what constitutes "*principles of information systems.*" It would hence be desirable to search for precise underpinnings to what constitutes an information system and how can it be positioned with respect to database design.

Any significantly advanced IS contains some type of a database. As a consequence, information system design addresses a number of database issues like conceptual modeling of data elements, metadata management, etc. On the other hand, any useful database system is actually an information system, providing additional services beyond simply storing data and running queries and updates. As a result, the distinction between a database and an information system tends to be blurred, and it is not clear that the principles underlining the study of information systems should be different than those for databases.

The distinction between a database and an information system is best appreciated when we consider their function, or "job." The "*job of a database*" is to store data and answer queries. This entails addressing issues like data models, schema design, handling distributed data, maintaining data consistency, answering queries, etc.

Given a query and a particular state of the system, the behavior of a database system, i.e. answering queries, is *algorithmic* in nature. *Algorithms* are Turing computable transformations from a predefined finite input (*question* state) to a finite output (*answer*). In particular, a database management system in principle carries out algorithmic mappings that manipulate a data store and provide individual query answers. This can be denoted by a partial function of the form:

> **DB computation**: an algorithmic mapping
> *(DB contents, user query)* $\rightarrow$ *(DB contents', query answer).*

On the other hand, the "*job of an information system*" is to provide a *service*, which entails considerations that span the life cycle of the larger system. Services are *interactive* in nature, involving user sessions with one or more users. Services over time which are specified by models of interaction, are not reducible to or expressible by algorithms or Turing machines [9, 32, 34]. Since models of interaction are a new area of research, it is not surprising that there have been no "principles of information systems" until now.

The distinction between a database that manages data and answers queries, versus an information system that provides services, is schematically depicted in Figure 1. Figure 1(a) shows information systems as wrappers above database systems; while Figure 1(b) shows issues of concern from a database and from an information system perspective.

In this paper, we argue that the interactive aspect of information systems necessitates a fundamentally different set of design principles for information systems, as compared to conventional database design. We provide some promising directions for a formal study of IS models, in particular based on concepts like Interaction Machines [32], Codesign [29] and Interaction Spaces [28]. The rest of the paper is organized as follows. Section 2 introduces some of the issues that contrast between the concerns of information systems and databases. Section 3 addresses these differences at

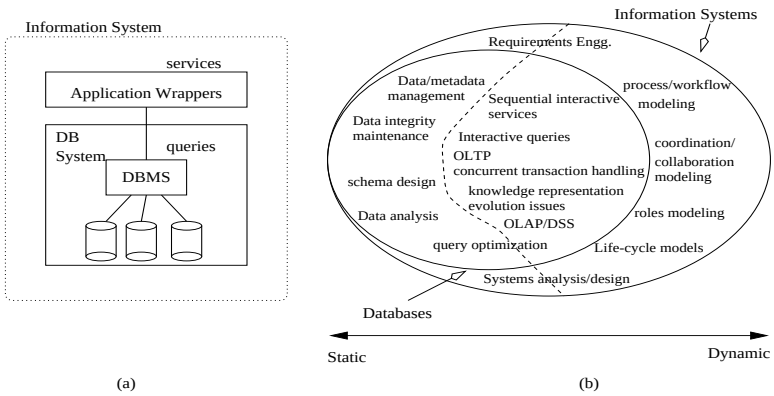

Figure 1: Information Systems vs Database Systems

the formal level – contrasting interactive behaviors with algorithmic behaviors. Section 4 surveys a few of the approaches which could hold promise in formalizing the notion of an information system; and Section 5 concludes the paper.

2 Databases, Information Systems, and their Users

Information Systems design historically began from a managerial and organizational perspective, and addressed issues such as systems analysis and design, requirements engineering, data flow design, systems development life cycles, etc. Over time, many of these ideas became more formalized. Concepts like life cycle models of ISs obtained support in the form of various representational paradigms and tools. In addition, issues like organizational structure, process structure, coordination and collaboration came to be treated along mathematical terms [18, 15, 14].

In a parallel vein, Database Systems (DBs) which began from issues like managing flat file systems of data, have progressed greatly to include a number of related issues. Contemporary database systems are no longer just sophisticated file systems with ASCII interfaces accepting one ASCII-based command (query or update) at a time. Currently, useful database systems are actually information systems, whose computation requires a richer model than just a mapping from one database state to another, and whose issues of concern extend beyond questions of data modeling and query evaluation.

The expansion of concerns in both ISs and DBs have resulted in the blurring of the domains. For example, Loucopoulos [17] defines an information system as "systems which are data-intensive, transaction-oriented, with a substantial element of human-computer interaction." It could well be argued that contemporary database research also adopt a similar model for defining a database system. But, at the level of foundations, the outlook is somewhat different. Research on *principles of database systems* is well-defined, at least since the middle '80s [30]. It includes issues such as schema design, data modeling, query equivalence and expressiveness, and dependency preservation. On the other hand, there is no consensus on what constitutes the *principles of information systems*, and no accepted formalisms for studying these principles.

This section contrasts between concerns that have motivated IS design versus those which have been the motivating factors for databases.

2.1 Statics vs. Dynamics

As discussed in section 1, database research has focused on executing single queries (or sets of queries grouped into *transactions*) against a given *database instance*. Though distributed database research has considered queries distributed among many users who are accessing the system simultaneously, the issues addressed have mainly concentrated on maintaining database consistency.

On the other hand, an information system provides database-backed services to its user community. Its computation is *interactive*, with the dynamically generated streams of user requests serving as input, and the corresponding feedback to the users serving as output [9].

> **IS computation**: an interactive transduction of one or more autonomous *input streams* of user requests into *output streams* of system feedback, accompanied by an *evolving system state*

Note that the input consists of all the user requests, throughout the lifetime of the system, and *not* of single commands that constitute the low-level tokens of the input. The system *evolves* over time, during its computation over its input, allowing its *behavior* to be *history dependent* and its services to be *individualized*.

On the other hand, a database is fundamentally concerned with maintaining the *static* aspects of the system. Its computational episodes are carried out over individual strings representing single user requests (queries or updates). That request is executed over the current database *instance*, which is an instantaneous static snapshot of the database contents. Both the instance and the request can be represented as finite strings over a finite alphabet. Together, they constitute the input for the database computation, which is a mapping to a new database instance and/or to a query answer.

Whereas database computations can be modeled by algorithms or Turing machines, capturing IS dynamics requires interactive models of computation that are more expressive. As a result, the formalization of information systems demands interactive principles that are quite different from that of databases. Section 3 contrasts DB and IS dynamics in a formal manner.

2.2 Individualization of Interactive Services

When we consider the interactive nature of an information service we realize that the user plays much more of a role than just providing queries. Users of information systems require services that are catered to their individualized requirements. Information systems may employ a common schema to maintain all data elements and provide central control for all system processes. However, as information services grow in complexity, a single logical structure cannot satisfy all functional requirements.

Individualization is the answer to this problem. It refers both to enabling users to formulate customized requests, as well as to allowing them to customize their view of the feedback. Individualization can be viewed as a projection of the information system onto the user's space, tailoring the available services to the user preferences and characteristics. Examples of customized user requests are:

1. `list the current valuation for all houses on my street`
2. `purchase recommendations for users based on their purchase history`
3. `individualized forms of existing services for users with disabilities`

Individualization requires awareness of *user characteristics* and *user preferences*, as well as of the *history* of user interaction with the system. In order to individualize information services, an IS must be able to create and maintain individual *user profiles* with the relevant information. Some of the user characteristics are fixed (such as the user's gender) while others may be reset by the user (such as their nickname). There are also characteristics that are never set by the user explicitly; some are simple facts (such as their software version number), while others, probably the most interesting category of user preferences, are derived and updated from the interaction history itself (such as their interests and patterns of behavior).

Individualization is not restricted to modeling the user. The "user" may as well denote different kinds of target platforms over which the IS services are provided. Information systems need to adapt to constraints posed by these target platforms. For example, a service providing email access over the web may need to have different interaction structures depending on whether the user accesses it over a web browser, a cell phone or a public terminal; even though the fundamental service provided is the same. Besides disambiguating user requests, individualization is also used to determine the structure for the whole interaction process [16].

Individualization of interactive services can be considered to be an extension of the concept of views in database design. Database views are static individualized structures, while IS individualization involves individualization of the dynamics as well. Database views contain the same data that are stored in the central database, but with possibly a different schematic structure. Individualized IS views may contain information about user preferences and interaction history that are not part of the larger information system, and also constraints which affect how IS services are used by the user. A database view is a subset of the larger database; but an individualized service may have completely different process structures which are not part of the larger IS.

2.3 Modeling the User

Since users generate the input streams for the IS, they constitute the external *environment* within which the IS performs its computation [9]. Modeling the environment of an interactive system is necessary in order to have a complete model that can predict the system's behavior [33]. It is not surprizing then that information systems design extensively involves the in its modeling.

Modeling a system for designing an IS usually involves conceptual modeling of the system's actors, its users, and ontological descriptions of the system's structure and behavior. Users interact with the system that is organized according to its ontological descriptions, to make use of the services that it provides. The modeling paradigm here is usually at a much higher level of granularity than for DBs and does not concern itself with how data is organized and how they can be efficiently queried.

On the other hand, when modeling the computation of a database, the *users* (or their characteristics) are not modeled explicitly. It is an axiom of database theory that all queries are expressions over a non-ambiguous query language whose semantics depend only on the database, and not on the user that generated the request [30]. Though the user is implicitly present, it is assumed that his wishes are completely expressed by the query itself.

Since user profiles are a part of the information system but not a part of the database that underlies the IS, we must conclude that individualization (section 2.2) is a strictly IS issue; The next section considers differences between DBs and ISs in a more formal manner, and provides motivations regarding why principles of information systems may need to be determined separately.

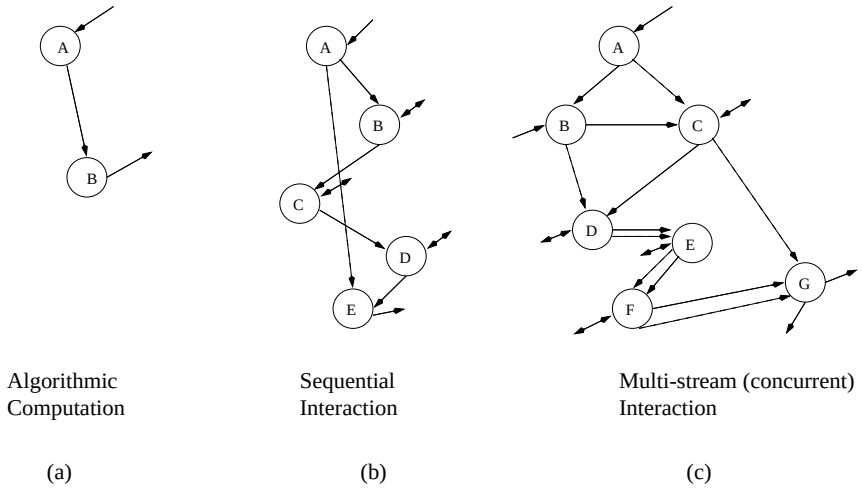

Figure 2: Algorithmic and Interactive Processes

3 Formal models of DBs and ISs

3.1 Algorithmic vs. interactive dynamics

With the onset of paradigms like object orientation, it is believed that the distinction between high level conceptual modeling and the lower level data organization and management gets bridged. This is because data and their associated processing may now be represented in terms of problem domain objects and interactions between them. Many approaches towards database design hence proceed from an application perspective – that is, progressively coming down to low level data modeling from higher level conceptual modeling [7]. However paradigms for modeling the dynamics of object oriented databases are unclear regarding whether services provided by objects, are part of the object database or are explicitly outside the database [20].

In a similar vein, the principles which concern information systems design, (as contended by Alter [3]) is said to boil down to management of the following three issues:

- The information content
- The organization of the information
- Mechanisms to enable users obtain whatever information they need.

It seems to suggest that database design and information systems design converge using the object paradigm. However, as contended by Wegner and Goldin [32], the paradigm shift from the earlier procedural (algorithmic) to object oriented (interactive) paradigms involves a *qualitative* shift in design principles, rather than a simple shift in the granularity of programming. Information system dynamics are interactive in nature, while database dynamics are algorithmic in nature.

Figure 2 depicts the qualitative difference between database and information system dynamics. Figure 2(a) depicts an algorithmic mapping that maps a "problem" state to a "solution" state in a closed functional mapping. Such a behavior is characteristic of read-only databases that answer queries by mapping $(query + database)$ to $(answer)$. Figure 2(b) depicts a sequential interactive process. This is characteristic of a single-user read-write database. Here, response to a query is

dependent on the present database state, which is a result of past interactions of the database with the user. In the earlier read-only case, the database state would be the same regardless of when the query is provided. In a read-write single user database, the database state is a function of the interaction history. Figure 2(c) depicts multi-stream interaction that is characteristic of the dynamics of most information systems. Here, interactive processes occur concurrently over multiple interaction streams. The response to a query on any stream would depend on the current system state, which in turn depends on past interactions as well as interactions taking place on other streams. In the figure, while the system state changes according to the sequence $ABCDEFG$, the interaction stream on the right perceives the state change sequence as ACG and the stream on the left perceives the state change sequence as $BDEFG$. Multi-stream interactions also model true concurrency and activities like coordination and collaboration between two or more interacting actors.

In a formal sense, algorithmic computation of Figure 2(a) is represented by a Turing Machine (TM) which is a mathematical model for computable functions. Sequential interaction, as depicted in Figure 2(b), represents composition of computable functions sequentially, such that the state is persistent over the computations. This is represented by Sequential (single-stream) Interaction Machines (SIMs); Persistent Turing Machines (PTMs) are one example [9]. PTMs provide a minimal extension to Turing Machines by having a persistent worktape whose contents are maintained intact over multiple PTM computations. Sequential interaction is also expressed by other mathematical models like Labeled Transition Systems [19], Coalgebras [13] and Transducers [30].

Multi-stream interaction, on the other hand, is still largely unexplored. Many areas of computation have dealt with multi-stream interaction. Some examples are: collaboration models, different kinds of coordination models in distributed computing [4, 5, 21], process management and resource sharing having shared contents between multiple processes, etc. However, a generic mathematical model of the notion of multi-stream interaction is still to be agreed upon. Wegner and Goldin [34] propose a model called Multi-stream Interaction Machine (MIM) as an extension to the SIM model, in order to represent multi-stream interactions.

A MIM can hence be considered to be a mathematical model for information systems, in the same way as a TM and SIM can represent read-only and read-write database management systems. However, the main hurdle that remains is that there is still no agreed upon mathematical structure that depicts a MIM. The formal model that comes the closest to modeling a MIM is Oracle Turing Machines, which are Turing Machines that interact with an "Oracle" as part of its computation. The Oracle may be a non computable function whose behavior cannot be modeled. The observed behavior on the interacting stream would hence record non deterministic behavior of the machine influenced by hidden adversaries. This is similar to the observed behavior of a MIM on any given interaction stream.

Despite the above, it would still be desirable to obtain a mathematical model that depicts MIM functionality which can be used to provide underpinnings for activities like collaboration and coordination among the interacting streams.

3.2 Algorithmic vs interactive solution spaces

A problem solving process can be considered to represent transitions in an abstract state space that depicts different states of the system.

Definition: The *solution space* of a problem solving process is defined as the set of all state transition sequences that characterize the problem solving process.

For example, if the problem solving process is an algorithm that computes the square of a given number, and the problem domain is the set of all integers, then the solution space of the algorithm is the set of all positive integers which are perfect squares. An algorithm is hence a functional mapping from the problem domain (set of integers) to the solution domain (set of perfect squares). The behavior of the process can be represented as a mapping $i \rightarrow o$, where i is the input from the problem domain and o is the output in the solution domain. For an algorithmic mapping, the solution space is also called the *language* of the algorithm. It is the set of all strings that can be produced by the algorithm.

For an interactive problem solving process as depicted in Figure 2(b), that involves more than one interactions before the final state is reached, it is not possible to determine the end state that is reached (state E in the figure) at the start of the computation (when in state A). The notion of a "language" consisting of a set of strings is no longer pertinent. Interactive problem solving is described in terms of streams which could be an infinite sequence of computational mappings.

Consider an *answering machine* which stores and plays back messages from the user. It supports three operations: `record()`, `playback()`, and `erase()`. Consider an interactive user session that involves the following operations: `record(Y), playback()`. The response from this session (the resulting state) would not only be dependent on the record message in this session, but also on any messages previously recorded. Such a process can be represented as a *partial function* that consists of mappings of the form $(s, i) \rightarrow (s', o)$, where s is the current state of the system at the start of the process, s' is the modified state of the system at the end of the process, i is the input from the problem domain and o is the output from the solution domain. A sequential interactive problem solving process that consists of n interactions can be considered to be the *transitive closure* of n partial function mappings like the above. In Figure 2(b), the interaction session from A to E is the transitive closure of four partial function mappings: $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow D$, and $D \rightarrow E$.

In a multi-stream interaction, the system state may be altered by interactions taking place on any stream. This could further alter the behavior of interactive processes taking place on other streams. In such a case, the partial-function mappings appear non-deterministic in nature on any given interaction stream. However, the behavior of the process as a whole need not be non-deterministic.

The *problem solving process* of algorithmic, sequential interactive and concurrent interactive computation may be represented in terms of their solution spaces, as follows.

Definition: An algorithmic problem solving process (APSP) is represented as a tuple $\langle I, O, \delta \rangle$, where I and O are input and output domains respectively, and δ is a (computable) function $\delta : I \rightarrow O$ that maps an element from an input "problem" domain to an element from an output "solution" domain.

Definition: A Sequential Interactive Process (SIP) is represented as a tuple $\langle S, I, O, \delta, E \rangle$ where S is called the "system state," I and O are input and output domains, and δ is a partial function of the form $\delta : S \times I \rightarrow S \times O$ that maps between input and output domains based on the system state, and possibly changes the system state as a result. E is the environment that the process interacts with whose behavior may or may not be modeled.

In an interactive problem solving process, inputs given by the environment may be coupled to previous

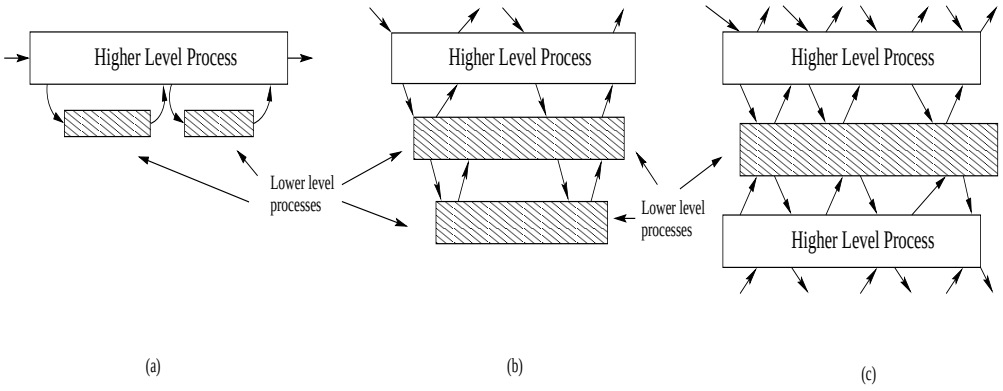

Figure 3: Algorithmic and Interactive Spaces

outputs. This is represented as follows: $(o_k, s_k) = \delta(i_k, s_{k-1})$; $i_k = E(o_{k-1})$

Definition: A Multi-stream Interactive Process (MIP) is represented as a tuple $\langle S, I, O, \delta, E_1, E_2, \ldots, E_n \rangle$, where S is the system state, I and O are input and output domains, δ is a partial function of the form $\delta : S \times I \rightarrow S \times O$; and $E_1 \ldots E_n$ are the n environments interacting with the MIP.

The interactive process is represented as: $(o_k^q, s_k) = \delta(i_k^p, s_{k-1})$; where the output to the q^{th} environment at k^{th} interaction is a function of the input received from the p^{th} environment. The input and output environments p and q may or may not be the same. The input from an environment at any instance may be coupled to the output received by it from a previous interaction: $i_k^p = E_p(o_j^p), j < k$.

The dynamics of a read-only database is given by APSP, and the dynamics of a read-write single user database is given by SIP. Multi-user databases are characterized by a MIP; however a MIP represents many other functionalities like coordination, collaboration, actors and roles, etc.

Multi-stream interactive spaces are more complicated than algorithmic or sequential interactive spaces. The solution space of an algorithm or of a sequential interactive process, can be broken down into well formed hierarchies, where each level in the hierarchy can be considered to be at a strictly lower level than those which invoke it. Hence, database transactions that read or write elements of data can be broken down in a hierarchical fashion to deal with multiple data sources. However multi-stream interactive spaces do not render themselves into well formed hierarchies. An interactive process maintains a *coroutine* relationship with its environment, rather than a *subroutine* relationship maintained by algorithmic processes. And when an interactive process interacts over multiple streams, the coroutine nature of an interactive process breaks down hierarchical structures defined by process granularities.

Figure 3 explains this distinction better. Figure 3(a) represents a database engine that executes a transaction over a database. The transaction can be broken down into sub transactions which are within the scope of the larger transaction. Even though the sub processes might interact with one another, the larger mapping is still algorithmic in nature. Figure 3(b) represents a read-write database executing transactions. The behavioral space can still be broken down into a hierarchical fashion, as long as the transactions being executed are serializable – that is, as long as the interactive process is sequential. The concept of a hierarchy here is established by the fact that any operation executed by a process at a lower level is within the scope of the higher level process. This criteria

breaks down when processes interact over multiple streams.

Figure 3(c) represents multi-stream interaction where two interactive processes are executing simultaneously on two streams. The processes interact with the IS "core" during their lifetime. However, because of the existence of the other interactive process, the behavior of any single process cannot be reduced to a hierarchical functional mapping. This situation is analogous to non serializability of concurrent database transactions. They also represent situations involving coordination, collaboration, workflows, etc. which are common in information system design. Note that, without the existence of multiple processes, Figure 3(c) reduces to a hierarchical space of Figure 3(b).

Referring back to Figure 1(b), we can see that the left "database" part of the figure concerns spaces that are algorithmic in nature. The intersection of the "database" and "information system" concerns topics which are predominantly sequential-interactive in nature. However, information systems, as defined from managerial or organizational perspectives have mostly concerned concurrent interactions or MIM spaces, which are depicted in the right "information systems" part of Figure 1(b).

The next section looks at promising directions for formalizing principles of information systems. The motivating factors are the existence of open interactive spaces, and individualization of interactive services.

4 Towards Principles of Information Systems

4.1 Complex Systems

Information systems are a *complex systems* [26] whose *raison d'etre* is the interactions that it carries out within itself and with its environment. Interactive services that a system provides to its users, results in evolution of the system behavior in time.

A *snapshot* of the system captures its *static* aspects, such as the database schema, its contents, the current interaction history, and user account information. Some of these aspects are fixed for a given system, while others change (*evolve*) over time; we refer to the latter as *system state*. A system snapshot at any point in time hence provides information about the system state.

A snapshot model of the system, which captures only its static aspects, is inadequate for expressing its *dynamics*. Dynamics are processes that manipulate the system state. For software systems such as ISs, dynamics can either be algorithmic, sequential interactive, or concurrent interactive. These dynamics are represented by a TM, SIM and a MIM respectively. (Note the stress on software systems; although other systems like social networks or communities also contain multi-stream interactions, the behaviors carried out by the actors may not be effectively computable. In contrast, TMs, SIMs and MIMs all assume the effective computability of their behavior).

In order to model and address the complexities of emergent properties from multi-stream interactions, usually, the first step that is taken is to look for behavioral archetypes or "patterns." Some well known patterns of behavior in information systems have been documented like the MVC framework, the Publisher-subscriber pattern, the Observer pattern, etc. [8, 22]. However, the notion of behavioral patterns still lack precise definitions and notational semantics which makes it hard to formalize and automate their usage.

Similarly, the complexity of interactive modeling have also been addressed by paradigms like actors and roles [1]. Actors are autonomous objects of the information system, whose behavior at

any given instance, is determined by the role that they have adopted. This paradigm has also been used by the Artificial Intelligence (AI) community [2]. The evolution in AI from logic and search to agent-oriented models is not a tactical change, but a strategic paradigm shift to more expressive interactive models [31]. IS modeling based on actors have extended to different related paradigms like mobile agents and collaborative agents [37, 12].

While the actor and agent paradigms provide an intuitive way of modeling a dynamic system; they are what might be termed "entity centric" in nature. That is, domain entities or actors form the building blocks of the model, and the system dynamics are represented on top of the actors. This would be inadequate to address the complexities of interactive solution spaces, where interactions between entities need to be modeled and managed explicitly.

4.2 Towards a theory of Information Systems

As noted in the previous section, the complexity of modeling an information system comes from the complex nature of its solution space. While there have been many paradigms that represent dynamic processes, the question of interactive services operating concurrently on multiple streams, have not received adequate attention. In this subsection, we look at some approaches towards this end which could provide promising directions towards principles of information system design.

Interaction Machines: Interaction Machines [32, 33, 34] address interactive solution spaces by proposing two models of interaction – single stream and multi stream interaction. Single stream interaction is represented by a Single-stream Interaction Machine (SIM) such as a Persistent Turing Machine (PTM) [9]. PTM computation can be modeled by maintaining a persistent state across one or more TM computations. A model for multi-stream interaction has also been proposed, called a MIM; but it still lacks formal mathematical underpinnings.

Coalgebras: Jacobs and Rutten [13, 23] propose coalgebras as a general paradigm for modeling dynamic systems. A *dynamic system* is represented as a tuple $\langle S, \alpha_S \rangle$, where S represents the state space of the system and α_S is called the system "dynamics." α_S is of the form $S \rightarrow F(S)$ which maps each state in S to a set of dynamics determined by the functor $F(S)$. Such a system is also called an *F-coalgebra*, since its dynamics is determined by the functor F. Coalgebras present a powerful mechanism for representing many different kinds of dynamic systems using a common formal framework. However, the issue of multiple streams of interaction cannot be directly modeled using the above paradigm.

Categories and Abstract State Machines: A category K is represented as a collection of "objects" $obj(K)$ and "morphisms" $mor(K)$, such that:

1. Each morphism f has a "typing" on a pair of objects A, B written $f : A \rightarrow B$. This is read as 'f is a morphism from A to B'. A is the "source" or "domain" of f and B is its "target" or "co-domain".

2. There is a partial function on morphisms called composition and denoted by an infix ring symbol, $\circ$. The "composite" $g \circ f : A \rightarrow C$ may be formed if $g : B \rightarrow C$ and $f : A \rightarrow B \in mor(K)$.

3. This composition is associative: $h \circ (g \circ f) = (h \circ g) \circ f$.

If "objects" of a category are taken to be the system state and morphisms as dynamics between the system states, a category would hence represent a dynamic entity. Each object can further

represent a subsystem within itself. Such a model would represent an Abstract State Machine (ASM) [10].

But, as with coalgebras, categories do not distinguish between two or more environments with which they interact with.

Stocks and Flows: Stocks and flows is a formal model resulting from managerial and organizational perspectives of IS design [11]. In this model, a system consists of static and dynamic elements. The static elements are called *stocks* and are referred to as the "nouns" of the system; the dynamic elements are called *flows* and are referred to as the "verbs" of the system. An entire configuration of stocks and flows forms the system which interacts with one or more "environments." Stocks are represented by variables and flows are represented by difference equations over stocks and environments.

Stocks and flows provide intuitive underpinnings for IS design. However, some of the features that it lacks are mechanisms for abstraction, reasoning, and integrity checks on the system.

Codesign and Interaction Spaces: Codesign [6] is a paradigm proposed for IS design that addresses the design of both statics and dynamics simultaneously. Global and local (individualized) views are also defined for the dynamics, analogous to corresponding paradigms regarding the statics. Codesign provides a more comprehensive design paradigm for information system by taking the whole picture of the information system into consideration: IS = DB + Interaction.

Codesign divides IS design into four different aspects of concern. These are: individual-static, individual-dynamic, global-static and global-dynamic. The *individual-static* aspects concern database views and their computation; while *global-static* concerns design of global static issues like conceptual modeling, logical design, etc. The *global-dynamic* aspect forms the design issues for the application software that represents the IS; and *local-dynamic* aspects are individualized dynamic elements of the IS.

Interaction Spaces [28], which forms a part of the Codesign approach proposes a formalism for representing the local-dynamic and global-dynamic aspects. In this approach, the IS concerns are divided into two abstract "spaces" of concern: the static "entity space" and the dynamic "interaction space." The system model hence consists of two schemata – the static entity schema and the dynamic interaction schema. The interaction schema denotes the global-dynamic aspect of the IS; while entities that make up the interaction schema (called "dialogs") form the local-dynamic aspects of the IS.

The endeavor in the above is to be able to characterize MIM computations. The concept of an interaction space addresses the complex nature of multi-stream interactive behaviors. However a mathematical model that could represent every multi-stream interaction space is yet to be found.

5 Conclusions

This paper presented some promising directions for establishing *information systems priciples* as a well-defined area of research, unified with the *database principles* community by a common framework, but whose concerns and techniques are *interactive* and therefore distinct from those of databases. The remaining challenge seems to be the predominantly uncharted territory of modeling multi-stream interactive behaviors. Bridging this gap would provide strong foundations for areas of systems design and empirical computer science.

References

[1] Gul Agha, Ian A. Mason, Scott Smith, Carolyn Talcott. Towards a Theory of Actor Computation. Proc of CONCUR '92, R. Cleaveland (Ed.), LNCS 630, pp. 565-579, Springer-Verlag, 1992.

[2] Philip E. Agre, Stanley J. Rosenschein. Computational Theories of Interaction and Agency. *MIT Press*, 1996.

[3] Steven Alter. *Information systems: A Management Perspective*, Benjamin/Cummins, 1996.

[4] N. Carriero, D. Gelernter. Linda in Context. *Communications of the ACM*, Vol 32, No. 4, pages 444-458, 1989.

[5] P. Ciancarini, A. Omicini, F. Zambonelli. Coordination Technologies for Internet Agents. *Nordic Journal of Computing*, Vol 6, No. 3, pages 215-240, 1999.

[6] Wolfram Clauß, Bernhard Thalheim. Abstraction Layered Structure Process Codesign. *Proceedings of the 8th International Conference on Management of Data (COMAD '97)*, Narosa Publishers, Chennai, India, 1997.

[7] David Embley. *Object Database Development: Concepts and Principles*, Addison-Wesley, 1998.

[8] E. Gamma, R. Helm, R. Johnson, J. Vlissides, G. Booch. *Design Patterns : Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995.

[9] Dina Goldin. Persistent Turing Machines as a Model of Interactive Computation. *Proceedings of the FoIKS 2000*, Burg, Germany, Feb 2000.

[10] Yuri Gurevich. May 1997 Draft of the ASM Guide. *Technical Report*, University of Michigan EECS Department, CSE-TR-336-97.

[11] B. Hannon, M. Ruth. Dynamic Modeling. *Springer-Verlag*, 1994.

[12] Michael N. Huhns, Munindar P. Singh, Les Gasser (Eds.). Readings in Agents. Morgan Kaufmann Publishers, 1998.

[13] B. Jacobs, J.J.M.M. Rutten. A Tutorial on (Co)Algebras and (Co)Induction. *Bulletin of EATCS*, Vol. 62, 1997, pp. 222–259.

[14] Yan Jin, Raymond E. Levitt. The Virtual Design Team: A Computational Model of Project Organizations. *Computational and Mathematical Organization Theory* 2(3), 1996, 171–196.

[15] Peter Lawrence (Ed.) Workflow Handbook. *Workflow Management Coalition*, 1997.

[16] J. Lewerenz, S. Srinivasa. Abstraction of the Interaction Properties of an Information System for Achieving Target Platform Independence. *Proceedings of Modellierung 2000,* St. Goar, Germany, April 2000.

[17] Pericles Loucopoulos, Roberto Zicari (Eds.). Conceptial Modeling, Databases and CASE. *John Wiley & Sons*, 1992.

[18] Thomas W. Malone, Kevin Crowston. The Interdisciplinary Study of Coordination. *ACM Computing Surveys*, Vol. 26, No. 1, March 1994, pages 87-119.

[19] Zohar Manna, Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*, Springer-Verlag 1992.

[20] Oscar Nierstrasz. A Survey of Object-Oriented Concepts. In W. Kim, F. H. Lochovsky (Eds.) *Object-Oriented Concepts, Databases, and Applications*, ACM Press 1989.

[21] G. Papadopoulos, F. Arbab. Coordination Models and Languages. *Advances in Computers*, Vol 46, pages 329-400, 1998.

[22] Wolfgang Pree. Design Patterns for Object-Oriented Software Development. *Addison-Wesley*, 1995.

[23] J.J.M.M. Rutten. Universal Coalgebra: a theory of systems. *Technical Report*, CS-R9652, Centrum voor Wiskunde en Informatica, Amsterdam, Netherlands, 1996.

[24] Klaus-Dieter Schewe, Bernhard Thalheim (Eds.). Readings in Fundamentals of Object Oriented Databases. *Informatik-Bericht I-10/1998*, BTU-Cottbus, September 1998.

[25] James A. Senn. Information Technology in Business: Principles, Practices, and Opportunities. *Prentice Hall*, 1997.

[26] Herbert Simon. *The Sciences of the Artificial.* MIT Press, 1996.

[27] Srinath Srinivasa, Bernhard Thalheim. Dialogs and Interaction Schema: Characterizing the Interaction Space of Information Systems. *Technical Report 13/99*, BTU-Cottbus, Germany, 1999.

[28] Srinath Srinivasa. The Notion of the Interaction Space of an Information system. *to appear in Proceedings of CAISE'00 Doctoral Symposium,* Stockholm, Sweden, June 2000.

[29] Bernhard Thalheim (Ed.). Readings in Fundamentals of Interaction in Information Systems. *BTU-Cottbus*, February 2000.

[30] Jeffrey D. Ullman. *Principles of Database Systems.*

[31] Peter Wegner. Interactive Software Technology. *CRC Handbook of Computer Science and Engineering*, May 1996.

[32] Peter Wegner. Why Interaction is More Powerful than Algorithms? *Communications of the ACM*, May 1997.

[33] Peter Wegner, Dina Goldin. Interaction as a Framework for Modeling. Conceptual Modeling: Current Issues and Future Directions, LNCS #1565, Peter Chen et al, eds.

[34] Peter Wegner, Dina Goldin. Mathematical Models of Interactive Computing. *Technical Report*, Brown University, Jan 1999.

[35] Peter Wegner, Dina Goldin. Coinductive Models of Finite Computing Agents. *Electronic Notes in Theoretical Computer Science*, Vol 19, Elsevier, 1999.

[36] Peter Wegner, Dina Goldin. Interaction, Computability, and Church's Thesis." to appear in *British Computer Journal.*

[37] Gerhard Weiss. Multiagent Systems : A Modern Approach to Distributed AI. *MIT Press*, 1999.