The Notion of the Interaction Space of an Information System

Srinath Srinivasa[1]
Brandenburgische Technische Universität
Postfach 101344,
D-03013 Cottbus
Germany
srinath@informatik.tu-cottbus.de

Abstract

Information systems' (IS) design concerns modeling systems that are dynamic in nature. A dynamic system essentially has two dimensions of concern – static structure and dynamic behavior. The existence of dynamics – or interactions among parts of the system distinguish a dynamic system from a heap or collection of parts. Specification and management of the static aspects of an information system like the data and metadata have been fairly well addressed by existing paradigms. However, an understanding of the dynamic nature of information systems is still low. Currently most paradigms model behavioral properties above an existing structural model, resulting in what may be called "entity centric" modeling. Such a kind of modeling would neglect properties that can be attributed to behavioral processes themselves, and relationships that might exist among such processes.

This thesis argues that the dynamics of an information system are best managed by explicitly characterizing an "interaction space" of the information system. An interaction space is defined as an abstract domain that represents the set of all dynamics of the information system. This is contrasted with an "entity space" that represents elements of the static structure of the information system. Recent results on the nature of interactive behavior and of open systems indicate that interaction spaces are characteristically different from the hierarchical nature of algorithmic problem solving. Interaction spaces consist of multiple interactive processes which affect the behavior of one another. Paradigms for the characterization of these spaces are hence explored as part of the thesis.

1 Introduction

Design processes for information systems (ISs) need to address issues of modeling dynamic systems. A dynamic system is characterized fundamentally by two dimensions of concern – the static structure and dynamic behavior. The structural elements of a dynamic system are those elements which may be identified from static snapshots of

[1]This research was supported by the German Research Society, Berlin-Brandenburg Graduate School in Distributed Information Systems (DFG grant no. GRK 316).

the problem domain; while the dynamic aspects involve those semantic elements of the system that exist over the time domain.

While modeling the static aspects of an information system like the data and meta-data, have been fairly well addressed; an understanding of the dynamic nature of information systems is still low. Currently, with paradigms like ER modeling and object oriented modeling, models can be built in terms of intuitive domain objects or entities and interactions defined among them. But a characteristic feature of most modeling techniques is that the modeling is "entity centric" in nature. In this, domain entities or objects, form the building blocks of the model; and dynamics are represented as interaction processes on top of the domain entities.

A shortcoming with such an approach is that behavioral aspects of the information system gets inadequate treatment. Behavioral issues of any fairly large information system are usually complex, consisting of many interactive sessions with the outside environment, tasks like coordination and collaboration among different actors of the information system, etc. Dynamic systems usually have emergent properties that result from the dynamics, and which cannot be attributed to the static structural aspects. In case of algorithmic computation, the emergent property depicts a computable function and can be understood easily. However, given any real world information system consisting of many multi-stream interactive processes, emergent properties are usually complex, without a common characteristic structure [6]. Such emergent properties are hence required to be addressed separately.

Currently, most approaches towards IS design proceed in an entity centric fashion, and neglect the complex nature of emergent behavioral properties of information systems. Behavioral modeling is usually carried out by defining a set of algorithmic processes depicting information system behavior. However this would be grossly inadequate for characterizing the complex nature of tasks like interactive problem solving, and its consequent activities like collaboration and coordination [10]. For instance, some aspects involving multiple behavioral processes are best specified in a negative fashion that depicts disallowed behavior, rather than allowed behavior [3]. Some examples of this are paradigms like mutual exclusion, locking and critical regions used in process management. Similarly modeling interactive behavior with open environments – like reactive processes – are not specifiable using algorithms [11], [18]. Modeling behavioral aspects of information systems thus involve more complex characterizations than just algorithmic specifications of behavioral processes over an entity centric system model.

In this work, the endeavor is to design a modeling paradigm that better addresses the dynamic aspects of information systems. This is done by dividing a dynamic system into two abstract "spaces" or domains of concern – the static or *entity space*, and the dynamic or *interaction space*. The entity space is the domain that contains the classes of all entities that can be part of the static system structure, and the interaction space is the domain that contains classes of all interaction processes that could be part of the information system dynamics.

The characterization of a space is called as a *schema*. Hence, a characterization of the interaction space is in the form of an interaction schema which consists of interaction processes as first class objects that build up the schema, and relationships among interaction processes.

In current modeling techniques, we occasionally encounter non-intuitive classes as part of the system model. (For example, a class called "Command" in GUI design; a class called "Event" in designing discrete event simulation, etc.) Such non-intuitive classes are often touted as "great" classes which form the key to reducing the complexity in modeling. However, with a notion of entity and interaction spaces; we can recognize that almost always such non-intuitive classes represent abstractions of interaction processes among domain objects. That is, they are classes that are part of the interaction space. The endeavor in this thesis is to explicitly bring out this notion of an interaction object and reason about how interactive processes affect one another inside an information system.

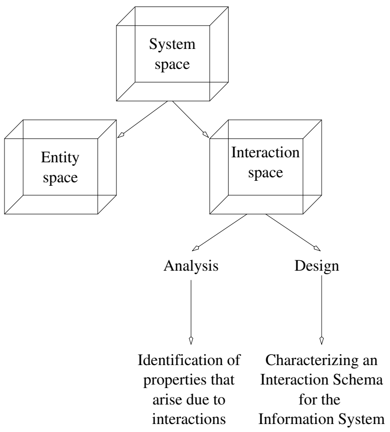

Figure 1: Overview of the research framework

2 Research Framework

An overall framework of the research is depicted in Figure 1. The research work is directed towards a better understanding and characterization of the interaction space of information systems. Broadly, the research efforts are divided into two parts – analysis and design.

In analysis, the task that is addressed is to discern properties of a given dynamic system that manifest due to the interactions that take place in the system (emergent properties of dynamic systems). In order to do this, a general framework of a dynamic system called an "ad hoc" system, is considered. This is a system framework consisting of actors interacting among one another with each actor pursuing its own goals. There is no known common vision or goal binding the actors' actions. The task is to discern global patterns of behavior emerging from these interactions.

In [14] an approach is presented that looks for consistent interaction patterns by mining transaction data. In [16] an approach is presented that seeks to discern task dependencies and possible semantic states existing in any such interaction pattern discovered. As an example, a situation involving transactions between autonomous, physically disparate departments of a hospital are considered. The task is to determine properties of interaction patterns by analyzing the transactions between these departments, in order to help design better logistics between the physically disparate departments.

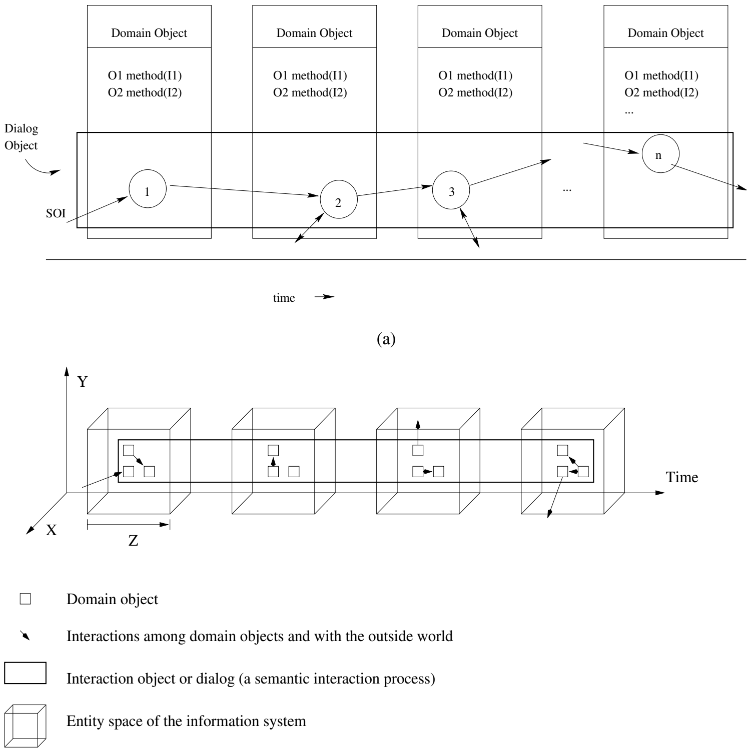

Figure 2: Domain objects and dialogs

The design aspect in this work seeks to characterize an interaction space. A characterization of an abstract space is called a *schema*. An interaction space is hence characterized by an "interaction schema." For our purposes, we consider the structure of a schema to be made up of *entities* and *relationships*. A schema $S = \langle E, R \rangle$, where E is the set of entity types that exist in the domain, and R is the set of relationship types that exist among the entity types in E. When S is an interaction schema, we call the entity types as "dialogs" in order to distinguish them from entities of the static entity space. A *dialog* represents a semantic interaction process that may involve one or more entities from the entity space, and may be related to other dialogs by *dialog association* relationships.

3 Dialogs and Dialog Associations

Figure 2 contrasts a dialog to a domain object. A dialog may be associated with one or more domain objects and is said to *represent* the domain object or subsystem. The domain object or subsystem is said to have *adopt*ed the dialog. Two or more dialogs may share a domain object and a domain object may have more than one dialog adopted by it at any time.

The system model consists of two schemata – an entity schema consisting of domain objects and relationships among them; and an interaction schema consisting of dialogs and relationships among them. A *compilation* of the system model results in implementable code that may be in one of two paradigms – (a). *actor* or *proactive* *domain object* model, where domain objects have independent threads of execution, and act as schedulers of dialogs which they have adopted; or (b). *process* or *proactive dialog* model, where dialogs have independent threads of execution, and use domain objects as shared data stores. Compiled into a language like Java, the system model can be considered to consist of two schemata, where, in exactly one of the schemata, all classes contain the `main()` method in them (i.e. can have their own independent thread of execution).

Some examples of domain objects would be objects like Account, Transaction Log, User, etc. A dialog, on the other hand represents a semantic process that is part of the information system. They are semantic objects that are identified in a dynamic four-dimensional picture of the problem domain. Some examples of dialogs are objects that represent processes or workflows like “OpenNewCreditAccount”, “OpenNewDebitAccount”, etc., in a banking situation; or objects like “LandingProtocol”, “TakeoffProtocol”, “CruiseControlProtocol”, etc., in an air traffic control scenario. Each of the above dialogs involve one or more domain objects like User, Teller, Account or Airplane, ControlTower, etc., and interacts with a larger environment.

The interaction space represents the complete set of dynamics of the information system. Hence any message passing between any two domain objects is always qualified within the context of some dialog object. In addition, dialog objects may pass messages among themselves; however, a dialog object is not allowed to interact with any domain object other than those which it represents. An interaction between a dialog object D and an arbitrary domain object P should take place only through another dialog object D' which represents P. This is a consequent of the assertion that the interaction space represents the complete set of IS dynamics.

The complexity in characterizing an interaction space comes from the interactive nature of dialogs. Interactive processes which carry out interactive sessions with their environments cannot be reduced to algorithms [18], [19], [20].

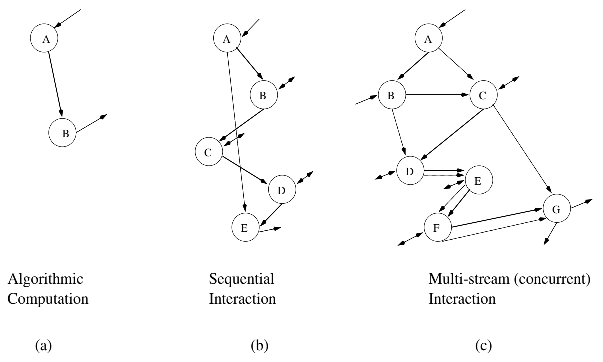

Figure 3: Algorithmic and interactive computations

Figure 3 contrasts between algorithmic and interactive processes. Figure 3(a) depicts an algorithmic mapping that maps a “problem” state to a “solution” state in a closed functional mapping. Such a behavior, characteristic of (say) read-only databases are represented as a mapping $i \rightarrow o$, that maps an input from the problem domain to an output from the solution domain. Figure 3(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. The mapping between any two interactions can be represented by a partial function of the form $(s, i) \rightarrow (s', o)$, that maps an input to an output based on the current state of the system, and changes the system state in the process. The system state is hence a function of interaction history. A semantic sequential interactive process consisting of n interactions is the *transitive closure* of n partial function mappings like the above. Figure 3(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 (non serializable) concurrency and activities like coordination and collaboration between two or more interacting actors.

In a formal sense, algorithmic computation of Figure 3(a) is represented by a Turing Machine (TM) which is a mathematical model for computable functions. Sequential interaction, as depicted in Figure 3(b), represents composition of computable functions sequentially, such that the state is persistent over computations. This is represented by Persistent Turing Machines (PTMs) [5]. PTMs provide a minimal extension to TMs 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, Coalgebras and Transducers.

Multi-stream interaction, on the other hand, is still largely unexplored. Many areas of computation have dealt with multi-stream interaction. Some examples are: different kinds of collaboration and coordination models in distributed computing, process management, resource sharing, etc. However, a generic mathematical model of the notion of multi-stream interaction is still to be agreed upon. Wegner and Goldin [18] propose a model called Multi-stream Interaction Machine (MIM) as an extension to the PTM model, in order to represent multi-stream interactions. However, the MIM model still lacks a formal mathematical structure.

An algorithm maintains a *subroutine* relationship with its environment while an interactive process maintains a *coroutine* relationship with its environment. The abstract space characterized by an algorithm can be broken down into well formed hierarchies, with processes in higher levels of granularities invoking processes at lower granularities as subroutines. Such a hierarchy can be maintained for partial function structures of sequential interaction. However, for multi-stream interaction, such hierarchical structures breakdown. There is no single characteristic structure of the space characterized by a multi-stream interaction. This aspect is elaborated in [6].

An information system can be considered to be a huge MIM; or consisting of many MIMs. The main challenge here is to be able to build a mathematical model that characterizes a MIM behavior. In this work, we address this question; although no claim is made that the resulting model is *the* complete mathematical model for a MIM.

Dialog Structure: A dialog represents a *single-stream* interactive process. That is, it does not distinguish between two or more environments that interact with it. As mentioned earlier, a dialog *represents* a subsystem and carries out an interactive process with the larger environment. The behavior of a dialog is dependent on the states of the domain objects that it represents. The dialog interfaces with its environment through a set of method interfaces M. The method interfaces can be considered to map the behavior of the dialog into a set of observational equivalence classes; $M : \mathcal{B} \rightarrow C$, where $\mathcal{B}$ is the behavior of the dialog, and C is a set of observational equivalence classes called "operational contexts." An operational context is determined by the values of the states of the domain object which the dialog represents. The formal structure of a dialog is defined below:

Definition: A *dialog* is represented as $D = \langle A, M, C, s_0, \delta \rangle$, where:

- A is the set of domain objects that the dialog represents, and is called the *attribute* set of the dialog. Elements of A are of the form of either T – representing any domain object of type T, or $x : T$ – representing a particular instance x of type T. (Two dialogs which have attributes of the form $x : T$ hence share the same instance x of the domain object of type T).
- M is a set of *method interfaces*, also called *observer functions*, which are of the form $m(T_i) : T_p \vee T_{p+1} \vee \ldots T_q$, which indicates that the function can provide an input of type T_i to the dialog (which could be a combination of multiple inputs of different types), and can observe an output of one of the types in $T_p \ldots T_q$ from the dialog.
- C is a set of *operational contexts* of the dialog. Each operational context depicts an observably closed context of dialog behavior, represented by a finite Labeled Transition System (LTS).
- s_0 is the *Start* state of the current context in which the dialog is presently operational. Each operational context begins from its *Start* state, and in most cases the start state s_0 represents the same state for all operational contexts.
- δ is a set of *context morphisms*, of the form $e : c_i \rightarrow c_j(Comp_e)$, where e is an expression involving elements of A, and $c_i, c_j \in C$. This denotes that when e holds, and current operational context is c_i, then the operational context changes to c_j, after executing the compensatory operation $Comp_e$.

A dialog represents a subspace of the interaction space. This subspace is characterized by the set of *operational contexts*, that the dialog operates in. Each operational context represents a particular scenario and depicts an observationally closed world of dialog behavior. Operational contexts of a dialog are represented in the form of a finite LTS that interacts with its environment, until it reaches one of the end states.

Dialog trace: Any instance of a dialog is said to leave a *trace* in the interaction space. The *trace* of a dialog is a string of the form $s_0(T_I \rightarrow T_O)^*p$, where s_0 is the start state, $(T_I \rightarrow T_O)$ is the invocation of an observer function on the dialog, and p is an end state in any of the operational contexts. The trace of a dialog represents a particular interaction stream that maps from the initial state of the dialog to any end state in any operational context of the dialog. The trace of a dialog is treated as a semantic entity that represents a dialog instance that once existed in the interaction space. Queries on interaction spaces return entire dialog traces and not any substructure of them.

Context morphisms: A dialog currently in context c_i is said to switch to operational context c_j, after performing operation $Comp_e$, if a condition e holds such that $e : c_i \rightarrow c_j(Comp_e) \in \delta$. Context morphism may occur at any time during the life time of a dialog. Let t denote the dialog trace prefix that has charted the behavior of the dialog in the current context c_i. When the dialog changes context to c_j, it changes to a state s in c_j determined by the longest prefix $p \in pref(t)$, such that p is recognizable in c_j. The trace prefix that is recognizable by all contexts is s_0. Hence in the worst case, the dialog rolls back to the start state in the new context and begins its process anew.

Context Subsumption: In the definition of a dialog it was mentioned that the start state s_0 is common to all contexts "in most cases." The exception to the above rule occurs when contexts subsume one another. Context subsumption is used to denote special areas of particular operational contexts, for example, critical regions. Since each operational context contains a separate state machine that defines the dialog, every context hence shares a common "start" state. However, a context that is subsumed by another context does not depict a new state machine, but instead depicts a subset of the state machine of the larger context. Hence such contexts need not have a common "start" state.

Example 1: A dialog that represents a student application process, would represent domain objects like: Admission_office, University, Student_database, etc. The environment (applicant) interacts with the dialog to submit an application for admission into the university. The dialog may have many operational contexts like: (a). *admission process for local students*, (b). *admission process for foreign students*, (c). *admission process for working students*, etc. The current operational context can be determined only after the student has submitted the initial request. In this case contexts do not have to change in the midst of the process, and no compensatory operations are required.

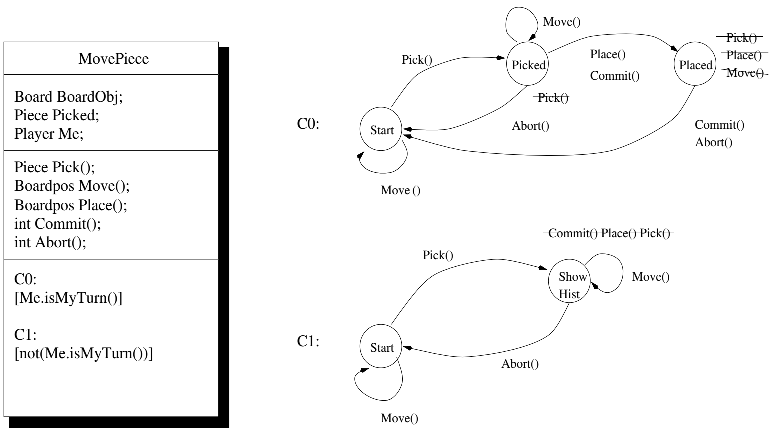

Figure 4: Movepiece: A dialog object

Example 2: Figure 4 depicts a dialog that has two operational contexts. The dialog represents a process by which a user moves his/her piece in a board game like chess. The context C0 represents the case when it is the user's turn to move; and context C1 represents the case when it is the other player's turn to move. In C0, the user can "Pick()" a *Piece* object which is a domain object and "Move()" it to another location on the *Board* object and "Place()" it there. In C1, the user cannot "Pick()" the piece since it is not the user's turn to move. Instead, the user can query as to how a piece came to its present location.

Dialog Associations: Earlier, a schema was defined as a tuple of entity and relationship types. In an interaction schema, dialogs form the entity types of the schema.

Relationships among dialogs hence need to be represented in order to obtain a complete characterization of the interaction schema.

Relationships may be of different types, and different relationship types may be identified based on the problem domain. However, we introduce a specific type of dialog relationship that is common across all application domains. This is called *constrained association*.

Definition: A *constrained association* between n dialogs is an n-ary association denoted by $R(D_1..D_n) = \langle\psi, D_1, D_2, \ldots, D_n\rangle$, where $D_1..D_n$ are n dialogs, and ψ is a constraint relationship on the *operational contexts* of the n dialogs.

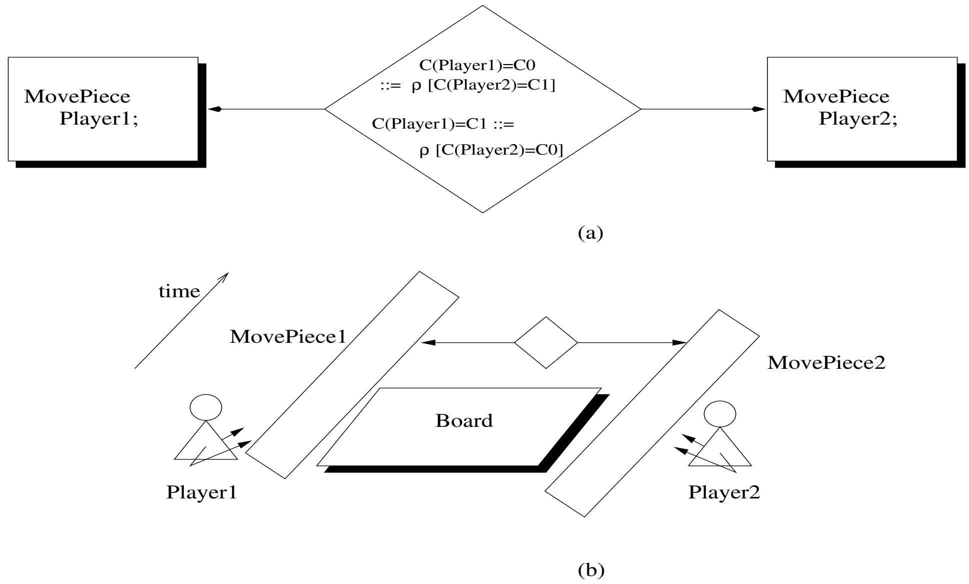

Figure 5: Constrained association between dialogs

Figure 5 depicts a constrained association relationship. This relationship represents the association that exists between two "MovePiece" processes of opposing players. The constraint that needs to hold in this association is that while it is the first player's turn to play – that is, when the first player is in context C0, the second player would be in context C1 and vice versa.

The syntax of a constrained association is of the form $body ::= head$, where $body$ is a conjunction of predicates of the form $b_1 \wedge b_2 \wedge \ldots \wedge b_n$, that depict conditions that need to hold as part of the constraint. The $head$ consists of a disjunction of one or more predicates, of the form $h_1 \vee h_2 \vee \ldots h_n$, that need to hold as a consequence, in order for the constraint to be valid. Different constructs are provided for depicting conditions over operational contexts of dialogs. An important construct is the "require" construct denoted by ϱ. A constraint equation of the form $body ::= \varrho(head)$ is read as "when $body$ holds, then it is *required* to have corresponding dialogs such that $head$ also holds." This means that, if the $head$ refers to other dialogs which are not presently running, instances of such dialogs have to be created and made to conform to the conditions in $head$. In the absence of the "require" construct, a constraint equation is read as "when $body$ holds, if there are corresponding dialogs that are affected by this association, then they need to conform such that $head$ holds."

Constrained association is a generalization of paradigms like mutual exclusion, locking, monitors and critical sections used in resource sharing and process management domains. While locks and critical regions seek to exclude processes from regions, constrained association may also have other constraints as well. For example, a constrained association may necessitate a dialog to *be* in a particular context when an other dialog is in an other context. This not only represents an exclusion constraint, but also an inclusion constraint that specifies where a particular dialog ought to be in. Constrained associations may have multiple arity and may hold over n dialogs simultaneously.

Multi-stream interactive processes: A constrained association actually represents an interactive process that interacts over multiple streams. The entire set of dialogs and the constrained association in Figure 5 may be considered to be a multi-stream interactive process that interacts with two environments. To an observer on any of the streams, the behavior of the dialog seems to change due to hidden adversaries.

Formal Underpinnings: The formal underpinnings for the notion of a dialog and an interaction schema is provided based on category theory and the theory of coalgebras. Coalgebras have been shown to be a convenient mechanism for representing dynamic systems and their composition [13]. Using category theory, a dialog can be represented as a category, with the operational contexts being represented as coalgebras. The combined behavior of the interaction space at any point in time is now represented as the coproduct of the operational contexts of all dialogs currently functional. Constrained association can also be defined in terms of coproducts and set differences of operational contexts. The formal underpinnings are addressed more comprehensively in [15].

4 Related Work

The notion of Interaction Machines (IMs) by Wegner and Goldin [18], [19], [20], show that interactive problem solving cannot be reduced to algorithmic problem solving. Although an object in the conventional sense would depict an interaction machine, the notion of IMs led us to ask how the "solution space" of an interactive problem solving is different from an algorithmic solution space. The solution space is the set of all states that are visited by a Turing Machine (TM) or an Interaction Machine (IM) in mapping between the problem and solution states. In [15] we argue that the solution space of an interactive problem solving can be reducible to that of an algorithmic problem solving only if the behavior of the environment that interacts with the IM is representable by an algorithm. The above led to the insight that perhaps the dynamic nature of information systems are very less understood, leading to the proposal of a paradigm called "interaction space."

Depicting an interactive process as a first class object, and relating interaction processes are not particularly new notions. There have been some approaches which have sought to represent processes as first class objects and also to relate process structures. Some examples are as follows – use cases (now part of UML [17]) consider process structures as coherent units of functionality. Workflow design for example, considers process structures as semantic entities in their own right. Recently, there have also been some approaches to relate workflow structures using aggregation and inheritance, to characterize a workflow space of sorts [1]. Similarly Liu and Meersman [9] build an "activity schema", where each object in the schema represents an activity instead of a domain object. However, there is no precise distinction between a structural schema and activity schema as proposed in our approach. At more theoretical levels, there have been paradigms that relate dynamic processes thus creating a schematic structure. Some examples are Hierarchical Petri Nets [12], and Abstract State Machines [7]. However, as noted earlier, they do not address the complex nature of multi-stream interactive processes.

Other approaches towards combining interactive components include Broy's compositional refinement of interactive components [4]. Here interactive components are combined based on a notion called "streams" that indicate the history of the component's interactive behavior. Hence the behavior of a component is history sensitive; however this approach lacks the addressal of constraint relationships that could exist among components.

In a larger sense, interactive paradigms of programming like object oriented, multi-agent and reactive systems, have been generally acknowledged to represent a different computing paradigm than algorithms. For example, in the context of reactive systems, Manna and Pnueli [11] conjecture that reactive systems are irreducible to algorithms. Abadi and Cardelli [2] develop a theory of objects using the term objects as fundamental concepts, rather than trying to explain objects as functions.

5 Conclusions and Expected Outcome

The notion of an interaction space can be further developed to suit different application contexts and provide general underpinnings for managing the dynamics of information systems. However, as part of the research work, we restrict the endeavor to developing the concept of dialogs as explained in this paper, and to design specific implementations for dialogs. Along this line, a specification language is planned that can be used to specify a dialog, its operational contexts and dialog associations. A CASE tool would then translate the specifications into some well known OO language like Java or C++.

In addition to the above, a paradigm called Interaction Schema Management System (ISMS) is also planned which manages dialog traces. Queries may be designed based on temporal logic declarations that return individual or clusters of dialog traces.

Acknowledgments: The author would like to thank his supervisors Prof. Bernhard Thalheim and Dr. Myra Spiliopoulou for all their support, and Prof. Dina Goldin for her guidance regarding Interaction Machines.

References

[1] W.M.P. van der Aalst. Generic Workflow Models: How to Handle Dynamic Change and Capture Management Information? *Proc. of CoopIS '99*, Edinburgh, Sept. 1999.

[2] M. Abadi, L. Cardelli. *A Theory of Objects*. Springer-Verlag, New York, 1996.

[3] R. Bol and J. F. Groote. The Meaning of Negative Premises in Transition System Specifications. *Journal of the ACM*, 43(5):863-914, September 1996.

[4] M. Broy. Compositional Refinement of Interactive Systems Modelled by Relations. *Proc. of Int'l Symposium on Compositionality*, 1997.

[5] D. Goldin. Persistent Turing Machines as a Model of Interactive Computation. *Proc. of FoIKS 2000*, Burg, Germany, Feb 2000.

[6] D. Goldin, B. Thalheim, S. Srinivasa. Information Systems = Databases + Interaction: On Principles of Information System Design. *submitted to ER 2000*.

[7] Y. Gurevich. May 1997 Draft of the ASM Guide. *Tech. Rep.*, Univ. of Michigan, EECS Department, CSE-TR-336-97.

[8] B. Jacobs, J.J.M.M. Rutten. A Tutorial on (Co)Algebras and (Co)Induction. *Bulletin of EATCS*, 62:222-259, 1997.

[9] L. Liu, R. Meersman. The Building Blocks for Specifying Communication Behavior of Complex Objects: An Activity-Driven Approach. *ACM Transactions on Database Systems*, 21(2):157-207, 1996.

[10] T. W. Malone, K. Crowston. The Interdisciplinary Study of Coordination. *ACM Computing Surveys*, 26(1):87-119, 1994.

[11] Z. Manna, A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*, Springer-Verlag 1992.

[12] A. Oberweis, P. Sander. Information System Behavior Specification by High-Level Petri Nets. *ACM Transactions on Information Systems*, 1(4):380-420, 1996.

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

[14] S. Srinivasa, M. Spiliopoulou. Modeling Interactions Based on Consistent Patterns. *Proc. of CoopIS'99*, Edinburgh, September 1999.

[15] S. Srinivasa, B. Thalheim. Dialogs and Interaction Schema: Characterizing the Interaction Space of Information Systems. *Technical Report, 13/99*, BTU-Cottbus, Germany.

[16] S. Srinivasa, M. Spiliopoulou. Discerning Behavioral Properties by Analyzing Transaction Logs. *Proc. of SAC'00*, Como, Italy, 2000.

[17] UML Resource Center. *http://www.rational.com/uml/index.jtmpl*

[18] P. Wegner. Why Interaction is More Powerful than Algorithms? *CACM*, May 1997.

[19] P. Wegner, D. Goldin. Interaction as a Framework for Modeling. *in LNCS #1565*.

[20] P. Wegner, D. Goldin. Mathematical Models of Interactive Computing. *Tech. Rep.*, Brown University, Jan 1999.