The Arrow System Philosophy

Brian T. Rice

January 6, 2000

Abstract

This proposal introduces a unified system for computations based on a cybernetic theory introduced here as model-level reflection. It provides a specification of what the Arrow system is and what it does for software systems. It outlines the theoretical basis for the system in terms of analysis of modern programming systems and their reflective capabilities. The system transcends the limitations of state-of-the-art reflection systems due to the current restricted notion of meta-systems. The system can represent any intuitive concept, and can manage the interactions among arbitrary domains of knowledge. Because of these properties, the user may delegate arbitrary aspects of its own operation and management to itself, with reliable and reusable results. The nature of the system is ideal for computations in a unified field of knowledge, and as such provides an ideal basis for the migration of knowledge to new contexts and uses.

Contents

1 Introduction

The Arrow system is a way for computing devices to model and manipulate collections of information gained from the world, up to and including the higher-order reasoning structures of which humans are capable. The goal for the system is to promote complete freedom in the sharing of useful information among people and machines. The technical proposal to achieve this goal is a homo-iconic fine-structure intensional information-processing system intended to support model-level reflection in a clean and useful manner. The means to this goal is a central logical construct with as little semantics implicitly specified as possible, in order to allow the system the greatest possible freedom in reflection on the state of information obtained. This single primitive allows a great freedom in modeling capabilities for the systems knowledge base. Because the system is homo-iconic, this central constructs semantic capabilities extend to yield more freedom in reflection upon the systems state and dynamics. This core of a model-level reflective system has ideal properties for a unified computation system with vast reflective abilities. To place this proposal within the space of software types, one should accurately identify it as offering more than an information database, but less than full artificial intelligence.

The Arrow paradigm is actually a simple analytical process involving the identification of binary ordered relationships at the level of individual logical atoms from any domain conceivable. The essential concept is the analysis of a domain in terms of these primitive information atoms. This paradigm allows for an uncountable number of objects to exist within its domain, to allow for the effective discussion of, say, the real number system and its properties. From a mathematical perspective, the arrow world in terms of elementary model theory is a system for managing the Robinson diagrams and positive diagrams of all models that agents deem useful for a knowledge system. The subject of the Arrow system is an extension of this concept of a Robinson diagram abstracted over the symbols that form the logical basis for these diagrams. The extension involves the integration of the logical theory provided by a model and the rules and symbols of the model of the system of logic that supports that theory. This allows a semantics of declaration and inference that vary from category theory to state-machine algebras.

Section defines the basic vocabulary and concepts that the Arrow system is intended to support, most importantly outlining a theory of reflection at a model level and characterizing the types of systems that support this theory. Section characterizes modern programming systems in terms of their abilities to meet the goals of general-purpose model level reflection. Section specifies the characteristics of the Arrow system and derives its properties constructively.

1.1 MetaText

This paper assumes a familiarity with the philosophical domains of both cybernetics, as proposed by Norbert Wiener, and liberal utilitarianism as proposed by John Stuart Mills essays On Liberty and Utilitarianism. Though throughout the progression of the arguments the reader could consider the words man and machine as interchangeable, this viewpoint does not mean to infer that such concepts are identical from the perspective of any domain other than cybernetics. In addition, the word system denotes a cybernetic entity, consisting of a mechanism for managing information. Such entities usually involve an incomplete union between man and machine. A utilitarian goal in this light is to maximize freedom for people involved in such systems by removing the unnecessary information overload inherent in present-day systems. Another goal is to promote information freedom at all levels. In this way, not only does the argument seek to enable information access for humans; it also seeks to expand the amount and type of relevant information available to the reasoning powers of computing systems. Such availability increases the capacity of the system to be useful.

This paper also includes some concepts only recently elaborated upon by computer science researchers from around the world. Since the intent here is to introduce new concepts, a significant part of the argument will clarify those ideas to support understanding of the system's design. Certain concepts are typeset in boldface whose definitions are useful in building a picture of the Arrow system. For further information, the paper directs the reader to the references mentioned below.

2 Reflection

2.1 Basic Reflection

2.1.1 Definition

This argument takes reflection to be the capability of a system to contain references to itself within the contexts it supports. Since all atoms available for reference are thereby available for re-use, reflection yields the ability for a system to modify its purpose and scope; a reflective system may use itself in ways that only the user could otherwise.

This definition depends upon the expressiveness of the systems algebras, since the availability of the system for use as a first-order object provides for its interactions via those algebras only. These algebras determine the nature of reflective capabilities in terms of introspection and intercession.

Given that we restrict our domain to those systems that are homo-iconic, we may infer that reflective statements or actions are only as expressible as the statements or actions within the first-order model for a system. In the same way, such reflective abilities will be quite as permissive and powerful as the first-order model allows.

2.1.2 Related Terms

The argument considers the kernel of a system given to be non-reflective as first-order or first-class. This is the object of discussion for any reflective system or meta-system.

When the system reifies models of itself for analysis, it performs introspection. When an agent effectively identifies that model with the current systems state, so that modifications to the model within the language produce changes in the systems state to maintain consistency with the model, the system is then performing intercession. Generally, modern systems find intercession more difficult than introspection in providing semantic cleanness, primarily due to inadequate models of the semantic capabilities of current systems.

The meta-system, in traditional systems with logical models, is a cleanly separated system whose domain is the first-order system. Its results are available to the manipulator of the first-order system for evaluating and modifying the first-order system. From a programming perspective, meta-programming involves the ability to perform actions usually reserved for the compile-time environment within the run-time system. In this way, programs can themselves choose other code to compile and run within their procedure or function contexts during compilation or execution.

A homo-iconic system consists of structures built from a single type of construct. All atoms within the system have identical implicit semantics. A system that reflects upon a homo-iconic system therefore reasons about structures of this single construct. All reflective discussions exist only in terms of collections of these fundamental objects in a homo-iconic system.

A virtual machine for a given system is actually a formally defined interface between that system and another one. Alternately, an algebra for an identifiable interface constitutes an operational description of a virtual machine. Algebras involve operators, combinators, functions, and constants. A more neutral term is the mathematical concept of a state-machine. This consists of featureless objects called states and transitions defined between the states which comprise the operations allowed on that state. Any modification to a state-machine by its current state constitutes reflection. A language in this context is a state-machine model coupled with a syntactical formalism, so that it constitutes an interface to that model.

A reification of a given concept is a reflective action that alters the virtual machines model to create a new context. This action yields a new first-order type for the system within that context. Previously, that type was an implicit part of the contexts semantics, usually perceivable by a thinking observer only. The stance taken by this paper is that an algebra of context best handles the implicitness of a given concept. In such a system, all actions exist in terms of the context-shifts that they engender. In this way, context-space becomes part of a language for expressing the semantics of an action.

The loop of reflection for a given system consists of a closed path of information flow regarding that system. This flow extends outward from the system into some collection of meta-systems. A human-concerned philosophy expresses interest in that part of the flow that a person receives, digests, and responds to by returning information to the system in question. Cybernetics considers many loops of reflection to apply to a given domain, because a domain may have many interpretations.

2.1.3 Purpose

A philosopher would usually consider reflections utility in terms of the benefits derived from the structure of the meta-system(s) used to encode reasoning about a situation within the domain. Overall, the effects of that reasoning on the capabilities of the domain therefore should serve the largest possible domain. The significant loop of reflection to this argument is that which models and solves problems in the systems domain. The argument seeks to improve the utility of this loop by making those problem solutions applicable to the widest set of domains possible.

Reflection as an action should enable the system to encompass the parts of the manipulator of the first-order system that it can. As a transition for system contexts, it should allow complete freedom in such manipulations. It follows that the most useful type of system would encompass all the transitions that it can rationalize into its domain for reflection. It should also allow the greatest possible capacity for modeling in order to maximize its ability to understand these actions.

2.1.4 Example Domains

Human Thought A person, via reflection, can think about a thing that he or she does, given a suitable means of expressing an ontology for the situation in which the given action occurs. This is the applicable meaning of the term within the English language. It is generally accepted, for instance, that thought itself falls under the domain of actions, and that it is reasonable for a person to reflect on such actions as well.

Computer Programming Traditionally, to program a computer is to specify a structured collection of posited formal statements in some statically specified domain. This is the traditional model of formal specification for a human-computer relationship. This general concept models almost every human-computer interaction specification. Cybernetics characterizes the vast majority of present computer systems species as having a collection of such domains created for them. The meta-structure for such domains specifies their interaction. This meta-structure is the subject of programming reflection. The reflective part of current programming systems consists in procedures that manipulate the current state of running programs.

2.2 Model-Level Reflection

2.2.1 Definition

Model-level reflection is reflection on the state and structure of a system at a higher level of semantics. Model-level reflection considers ontologies and their effects for basic reflection and circumscription upon the domain, which includes its own information, knowledge, and processes. Therefore, this scheme constitutes an algebra for basic reflection and circumscription via ontologies. Within this system, there are several terms by which to discuss the action in question.

The definition of model-level reflection relates to the concept of the knowledge-level in current artificial intelligence research. The structure of a knowledge-level description per Aurelien Slodzians ideas is as follows. Agents elaborate on an information state over which they have complete access and control. Models describe the knowledge used or produced by the agent while performing its tasks. Models exist as first-order entities to describe domains. These models form the agents current state of knowledge. These models would contain the concepts whose interactions would form the structure of a language for describing various situations. The agent may have multiple models per domain and multiple domains per model. Methods are the means that the agent has available for modifying its knowledge state. Tasks describe the agents goals and their structure.

The intent of this structure is to provide a useful categorization for the functions of a system and their relations to other systems. Particularly, an agent represents a locus of action for a system to analyze. Models are obviously the gained state of information due to the actions of agents. Methods describe the building blocks and combinators available with which the agent may act. Tasks describe the set of constraints and axioms expected of the system by the various systems with which it interacts, notably the hardware of its implementation, other software systems, and humans and their organizations. This ontology provides the reflective system with a high-level model of itself for manipulation that respects the integrity of information gained.

Since the original intended context for this concept is that of knowledge engineering, or its extension, artificial intelligence, some translation is necessary to relate these terms to the concepts of programming. Knowledge-level description should be inherently concerned with neither the representation of the agents knowledge nor the concrete mechanism used to perform the tasks. However, both subjects should certainly be part of its domain for a system addressing overall utility for its users. Such is the subject of execution-level reflection, which is naturally not in the users best interest upon which to focus.

The mathematical notion of a model involves the identification of basic symbols, such as functions and relations, from which an agent develops a logical theory to account for the domain in question. Models as such are arbitrary to construct, since several sets of symbols may be equivalent in their expressive power. Intuitively, a Robinson diagram is a function of a model that represents all true closed statements within that model. A positive diagram is that similar function concerning true atomic sentences in general. The present argument takes an expanded interpretation of the model notion, including not only the symbols subject to axioms, but also the model for the inference system that provides the Robinson or positive diagram itself. The combination of these conceptual levels yields a deeper and more meaningful structure to the usual notion of Robinson diagrams and positive diagrams. For instance, functions mapping expressions to truth-values subsume relation symbols. In addition, the current stance is to abstract the diagram notion over the symbols used, so that the result is a powerful algebra of context shifts in terms of inference systems and functions. This discussion shall take this new notion of a diagram for a model to be an arrow diagram.

Within the usual reflective systems, the system holds a model of itself, and the user and system manipulates the system's state according to that model. In contrast, model-level reflective systems may dynamically instantiate arbitrary models for arbitrary structures within the system or without of it, and provide first-order identification of those models with their intended subjects. To elaborate, a model-level reflective system may be seen as a system of abstractions for reasoning about the world, wherein the system has an effective means for applying models to its own structure in arbitrary ways. It should also be able to enforce the constraints desired for itself based on reasoning enabled by those models.

The model level of a domain, in this light, recognizes the universality of model theory, and therefore applies the model metaphor to all the elements of its domain. For instance, the agents knowledge proper is subject to modeling (as it also is at the knowledge level) within a system that reflects at this level. However, the tasks, the methods, the models themselves and representations of the agent are also available for analysis and modification according to semantics provided by the system. In order to maximize utility, the system should include within its domain the actual implementation of the system on a given set of hardware devices and the management of interface with system users. It should also provide reliable tools for verification of all implementation and communication decisions made by the system and the user. In this way, the user may prolong the effective lifetime of the system and the information that it maintains in a simple manner. In addition, the interface to users of a system whose design pays close attention to overall utility could handle issues of migration among implementations and interfaces transparently.

Informally, this discussion takes knowledge to be the closure of an information structure with an intelligent context provided by an underlying system, as symbolically provided by the presence of agents. We attribute knowledge to the agents by observing their actions; an agent knows something if it acts as if it had the information and is acting rationally to achieve its goals. The intelligence of the context is such that it can understand the effects of information updates on the knowledge that the system manages. Such structures as inference systems, type systems, and ontologies provide these effects. A model-level system therefore maintains such a database of knowledge derived from received information. A desirable property of this maintenance is for the system to manage consistency reliably within that knowledge database. There are two basic approaches to this question of consistency. The system may maintain a single consistent state wherein the new information and the current task determine the effect on the knowledge structure. Otherwise, the system maintains multiple states of information under an algebra of states wherein the new information always provides an information transition from the old set of states to a new one in a decidable manner. The latter approach results in a more complete system of information regarding the interactions of ontologies. A complete system in this regard is the right step towards model-level reflection.

This approach to reflection is much more general in scope than programming reflection in that programming reflection usually consists of a library of functionality that accesses and manipulates the executing programs. This takes a restricted view of the scope of reflection, since the structure of the virtual machine model and the process it facilitates, taken as a whole, is not subject to analysis or significant modification. Model-level reflection takes knowledge descriptions of a system and its substrate, the virtual machine, and integrates them into a coherent whole. While reflection may modify the state-machine that defines the virtual machine, model-level reflection observes and modifies the reasoning behind the choice of model. The model is not only the subject of model-level reflective analysis; it is also the medium for such operations.

Model-level reflection in a homo-iconic system must derive from the same primitive as the building block for the first-order system in order to preserve the accessibility of information and knowledge.

2.3 Related Terms

A context is a function of a model that provides an environment for agents that fully supports that model. A context therefore provides all accessible information in terms of structures specified by that model. In the same way, information actions that the agent takes may produce outgoing information that the system casts into remote domains via inverse translations.

An ontology is an explicit formal specification of a conceptualization, or a conceptual model for a domain. Specifically, ontologies are concerned with the objects, concepts, and relationships among them within some domain. In formal terms, an ontology is the statement of a logical theory. In terms of contexts, an ontology is associated with the space in which actors model domains in terms of the ontology, and so respect the restrictions posited by the ontology concerning its objects. Informally, this argument will consider the ontology for a context to be a description of the objects that are available for discussion.

An ontological frame denotes the formal model specified by a structure of ontologies, as well as the universe of discussion that it generates. Frames are structures of contexts, perhaps uncountable in size. The ontologies specified for the frame may crosscut each other in arbitrary ways, to allow the frame user to have the structure due to crosscutting available for study and first-order use. A model-level system develops its knowledge in terms of ontologies and their frames. Often, agents may find models within such a system mutually contradictory, and this is both permissible and desirable. Contradictions between models in the system allow for agents in the system to make assumptions contrary to the current state of information and study the consequences of the assumptions upon the relationships among various models. The means for encapsulating contradictions is merely to reify the instantiation of a new context that distinguishes the new set of inconsistent information from the old.

Ontologies are not absolute, as is evident from the need for translation schemes among the various human and computer languages. This suggests a concept of ontological relativism, which denotes the idea that the preference for an ontological frame is entirely relative to some base frame or structure of frames. Equivalently, no ontology can be inherently optimal for working with a specific domain. The overriding motto would be that No knowledge is an island. All agents approach ontology constructions with some frame in mind. A naturally useful paradigm foresees the consequences of this method and adjusts it for the general relativity of ontologies and their frames. Considering the relativity of ontologies, then, the system should not consider a particular ontology or frame as an absolute reference, but instead a relative reference where the results achieved by the modeling by default apply locally unless proven otherwise.

One may consider an ontology analogous to a set of definitions that apply to a specific domain within a dictionary for a human language. The agent draws vocabulary used in the definitions from some known source of meaning. The grammatical structure of the sentences used interacts with the users knowledge to produce a refinement of the users knowledge of the domains in which the word is used. In common dictionaries, the source domain is often the vulgar language. Specialized dictionaries, however, may specify a vocabulary from some standard, but specialized, reference vocabulary, as is the case with medical or scientific articles. This situation involves the understood use of partial relativism of ontologies, wherein a common (root or general-purpose) vocabulary is seen as the starting point for the specification of sub-vocabularies useful in restricted domains.

This definition of an ontology suggests modeling specific ontologies as directed links between two domains. A source domain provides the information from which the user (or the system) builds the ontology within the description language of a target domain. Since agents operate within frames, or structures of contexts, it follows that there may exist multiple ontologies for a given domain within the current context, and that these ontologies may easily crosscut each other.

Commonly available ontologies enable ontological commitments for a set of agents so that they can operate on a domain without necessarily using a globally shared theory. We say that an agent commits to an ontology if its observable actions are consistent with the definitions in the ontology. The idea of ontological commitments is based on the Knowledge-level perspective (Newell, 1982).

Pragmatically, a common ontology defines the vocabulary by which agents exchange queries and assertions among themselves. As such, an ontological specification is a specification of definitions of terms. Ontological commitments are agreements to use the shared vocabulary in a coherent and consistent manner. The agents sharing a vocabulary need not share a knowledge base. Each knows things the other does not, and an agent that commits to an ontology is not required to answer all queries that can be formulated in the shared vocabulary. In short, a commitment to a common ontology is a guarantee of consistency, but not completeness, with respect to actions and combinators using the vocabulary from the ontology.

Ontological frames identify the agent from the knowledge perspective. In this way, a model-reflective system may represent the user as an agent, making logical guesses about the user ontology frame by the interactions provided. If the user exhibits contradictory ontologies, then the system will update the frame accordingly. In this way, an agent may study the interaction of these ontologies. This allows for two or more users to interact with the system via the same device with no secure identification made, while the interactions between their beliefs and desires are preserved and managed. It similarly provides for the use and analysis of a user history in tracking the beliefs and preferences of the user as system knowledge develops. The system provides all of these benefits through casting users, external software agents, and all other incoming information in terms of agents and their ontologies.

Crosscutting of ontologies is the process of taking a given domain and interpreting it in two or more mutually independent ontologies. One may recall the examples from typical mathematical education wherein constructions and equations express geometry and simple algebra in terms of the each other. Similarly, the ontologies that describe the various aspects of a domain often crosscut each other. We should say that contexts that support first-order references to the elements and structure of multiple ontologies and their interactions support crosscutting of a domain ontology. An ideal model-reflective system provides the arbitrary crosscutting of ontologies to maximize information accessibility while allowing arbitrary ontologies for use in the desired domain. This essentially provides the system with definitions for the elements of a domain from various perspectives, so that the reasoning structures with access to the domain knowledge can analyze the system from the currently optimal perspective.

A fine-structure system (a.k.a. fine-grained) is one wherein agents may create models for domains from the basic building blocks provided by the system and their combinations, where the semantic complexity of these building blocks is minimal. The label fine-structure refers to the fact that such systems that can compute the results of a complex interaction among objects may, primarily due to the nature of the modeling construct, encapsulate effects of the computation that are ancillary. Such effects are usually discarded by the computation process, but result in additional complexity for the system to manage because such effects are therefore unavailable for logical analysis. Fine-structure systems are typically also homo-iconic, since a homo-iconic design reduces complexity by reducing the number of fundamental types, thereby reducing logical assumptions necessary to manage the building blocks. However, current homo-iconic systems hide the bulk of their complexity within primitive building blocks, which are those objects whose semantic definition is unattainable by the systems domain. These primitive objects constitute the main obstacle to general-purpose useful reflection. This is primarily since the intent of such systems is to provide the user with a means to rely on those primitives and disregard the availability of information to all parts of the system. While such a system allows a simple-minded control-based approach to computers, it applies a penalty in terms of utility, since the system cannot easily reflect upon primitives, so that their semantic content is unavailable though quite substantial.

2.3.1 Purpose

The intent of model-level reflection is essentially to preserve the information structure governing the relations between the first-order system and other domains in a semantically well-defined way. The function of the model level is to provide support for mappings from arbitrary ontologies to the domain for analysis. It follows that model-level reflection should provide general-purpose modeling capabilities for any domain, and that all models produced should be available for use by all interested parts of a system. Therefore, model-level reflection is naturally useful for reifying the properties of a domain that provide the most information for the user in terms of various aspect issues.

2.3.2 Example Domains

Philosophy A natural generalization of the domain of human thought concerns the motives and ontologies that oneself and others use. This domain, once properly extended in its scope, constitutes a homo-iconic system of concepts. In this system, the person and language employed become transparent. That is, the system reduces them to the role of cybernetic medium while retaining their availability as objects of discussion.

Information Systems The natural generalization of the domain of computer programming concerns the postulation and processing of information in general. The computer and user both exit the model as primitive objects, instead becoming specific objects about which the system may postulate information. In the process, the computer as context becomes effectively transparent. In the present, the only systems qualifying the above criteria consist of a social structure dedicated to preserving the underlying computing system for a relatively small group of users. For example, the primary group of users of the information systems within a modern corporation would be top-level executives, although the boundaries in this system are quite vague in that even such executives are often necessarily involved in the process. The essential property of an information system is the fact that its interface with all other objects is a clean one that makes transparent all but the interactions of information, and that this interface can interact with arbitrary domain types. The kernel of such a system is the system that maintains such an interface. The cybernetic optimization problem for such a kernel consists of removing as much noise from human society as possible while preserving the function of the kernel. From the outside of such a system, the objects under discussion are atoms of information. Within this system, the loops of reflection and their elements become first-order objects for discussion. Orthogonal referential integrity and persistence of information become necessary properties in the management of the system in that they reduce the amount of redundant information necessarily processed by any element of the kernel. As the information-base grows, the need for such a management will become critical if the system is to be useful. Partial views within a large information system have historically been acceptable due to the complexity of system relationships. If the number of relationships between objects increases geometrically with object population, for instance, then the categorization of these relationships to filter them from views of a large system is necessary if these relationships are to be comprehensible. Traditional systems address this with the concept of information hiding, whereby the system casts the information as either internal or external within a specification. Systems that are more elaborate often categorize external information in terms of interfaces, which a small population of objects shares among itself as a property. However, these traditional means often exist only in human-accessible (static) ways. Unity of the information system is essential for and identical to information accessibility. If there is a cost inherent in maintaining this unity, then that cost detracts from the overall utility of the system. More specifically, system unity does not infer a disregard for the relativity of ontologies, which would suggest the use of a root ontology from which to base specialized ontologies. Instead, it employs the understanding of this concept to allow for a far greater range of information translation ability, by providing paths of information translation that a hierarchy of ontologies would prohibit. By losing a fixed reference point, the system gains the ability to reflect upon its own information structures and domain models from various perspectives. The result is a far greater potential yield for the system in terms of its inference capabilities.

Knowledge Systems The following statements outline the type of design intended for the Arrow system to support, considered as a naturally useful extension to the previous definition of an information system. The formalization of such systems has not occurred among the research community, so that these specifications are vague. Knowledge systems are those information systems whose purpose is to yield real-world modeling capabilities via free interaction with users. No system today preserves these properties in a manageable way. These systems are those intended to outlast their creators and the conventions they use. Present day knowledge systems require a vast array of human participation to propagate. Utility compels society to improve on this situation. General-purpose circumscription and pattern-recognition are necessary to manage such a system. These services are traditional human mainstays. Circumscription, the vague primitive notion of reflection upon knowledge context, constitutes model-level reflection in a simple way. It introduces contexts as first-order objects and reifies them as classes of formulas. The significance of this concept is that predicates and objects from separate domains may be studied together, allowing the construction of systems that cross the boundaries between contexts. This action enables non-monotonic reasoning in a simple way. Pattern-recognition maps syntactical representations of object-structures in a source domain and identifies those structures with atomic objects of a target domain. Significantly, real-time instantaneous pattern-recognition is a task primarily limited to humans. However, conventional systems do not yet encapsulate the result of such a task within their reasoning systems. Knowledge systems allow such an encapsulation to exist across the boundaries of existing ontologies, allowing knowledge derived by the user to be available to the system in a useful way. In this way, a knowledge system that functions non-monotonically in complete isolation from human attention constitutes artificial intelligence. Regarded in the terms of cybernetics, the question of artificial intelligence shifts from the issue of construction to that of mapping existing knowledge systems onto available hardware.

3 Review of Existing Systems

3.1 Stacks

The basic popular context algebra consists of application of a transition over a hierarchy of contexts. These contexts represent the collection of model, method, and task into one form, resulting in a loss of model-level structure which is difficult to reverse. The simplest logical model is that of intuitionist logic, wherein assumptions as updates and downdates develop a lattice of contexts within which an agent bases its logical reasoning. The pertinent limitation of this system of logic is that downdating of information is not subject to constraints of high-level semantics. Primitive forms of this system include stack-based reasoning systems.

3.1.1 Procedural

The Pascal programming language exemplifies the modification of the simple von Neumann state-update machine from a maximally permissive model to one with a simple, rigid hierarchy of declarations of state-update permissions.

This simple language paradigm is so widely used that it merits analysis. The stack models context in allowing the program to push sub-contexts onto the stack or pop them from the stack. Sub-context types include procedures, which are sequences of updates to the machines state, and records, which are nested name-scopes. The primitive construct is a static sequential hierarchy of these contexts. The state-machine enters sub-contexts in sequence from the base context. The purpose of the context-inclusion relation that forms the hierarchy is to guarantee that variable access and update privileges between each pair of contexts are easy to understand by the programmer and simple to enforce by the compiler. Expressed as a context-algebra, the system eventually transforms all contexts to a single (super-) context, which manages all derived information. At all times, the current context on the top of the stack has available to it exactly the current states of all of its super-contexts and no other information. The solution obtained brings the inevitable result of a tremendous loss of information and a relatively low utility for information re-use.

This design arose from a paradigm where state-updates were maximally permissive. This permissivity constitutes an obvious inability to easily deal with meta-system management. In order to compensate, a user must consistently crosscut the model-level design ontology of the language with the desired one.

Even such a simple language as Pascal has provisions for control-flow modification based on conditional logic structures and simple looping constructs. This evidences the incredibly small semantic capabilities of a language that allows only two operations per knowledge context. It also supports run-time call crosscutting via exceptions for extending this basic system of context restriction. Simula constitutes the first generalization of the paradigm by merging the procedure and record types capabilities into that of objects (to be precise, state machines). Beta, a derivative language, generalizes procedures, objects, and exceptions as patterns in order to enhance further the structures that crosscut this hierarchy.

A possible solution to this dilemma involves the method that file-system designers have used for quite some time. The availability of path references constitutes a tremendous extension to the ability of the system to re-use gained information. The path concept is a simple reification of the geometrical structure of the hierarchy, and, as such, constitutes a first-order expression of the systems context algebra. In this simple case, block-structure equals context-structure, so that the familiar operators from Unix may be re-used. In this way, the foreslash operator (/) acts as a selector coupled with an identifier for the particular relative sub-context. The ellipses . and .. select the current context (self) and the parent context (parent), respectively. Concatenation of these operators and elements of the domain specification produces programs of context-selection, manipulation, and inspection via standard first-order procedures. Given the current apparent success of interfaces to storage based on this meta-system scheme, it seems amazing that no current large-scale procedural programming language totally integrates this scheme into its definition as a way of reifying context network topology.

3.1.2 Functional

The original homo-iconic language paradigm concerns the function application and the concatenation (currying) operators as fundamental and identical. Context-inclusion relations in the various flavors of Lisp have employed either run-time call or lexical inclusion.

Functional programming is that which encapsulates all the effects of a context shift on a system as a first-order logical atom. The essential word in this case is all. In traditional languages, the concepts of closures, continuations, and currying operators encapsulate all the effects that would traditionally concern the language. We may characterize these effects as introducing new first-order atoms, specifically the results of computations in an intensional way, that is, by reference to function application, not to the state of a virtual machine in the traditional sense.

What results is a paradigm of computation where all context-updates equal a function application, so that the functions available to a system and the restrictions on application invocations entirely determine the range of model-level reflective methods available to an agent. In addition, the clean semantics of function application allow the managing system to both apply and construct functions dynamically, which yields a system of far greater utility than the basic intuitionist paradigm. The properties of functional semantics relate to formal proof theory via the Curry-Howard-deBruyn isomorphism, allowing for the simple formal determination of program semantics.

Here, the functional metaphor extends the path metaphor from the simple hierarchy where block-inclusion equaled context-inclusion to a system wherein many other function application webs crosscut the unique block-structure hierarchy. The selection of a particular set of function applications constitutes the usual idea of a directed graph of data-flow where nodes model function applications and arrows model transitions in context due to those applications. Obviously, our modeling strategy shows that the current context-algebra is far richer than the procedural method, since identifier selection is no longer sufficient to identify a context-shift. Instead, context shifts also require agents to specify an atom of state as the functions argument, so that the selector is binary vice unary.

3.2 Objects

The object-orientation paradigm consists of a basic translation of the state-machine model into the model that localizes all permissions for context updates to the states of (usually) simpler machines. It consists of the definition of a clear virtual-machine creation vocabulary in terms of the substrate state-machines operations. It is, therefore, a translation from the substrate machine into an environment of declared state-machines. Taking the paradigm purely without dilution yields a clean model where interfaces are entirely controlled local to the information concerned. The object-based paradigm is a specification of permitted information update methods per logical atom. The result is a useful model for the production of cleanly defined information systems.

Current programming methods, including text-based enumeration of code, limit this paradigm by forcing the overloading of the meaning of various operators. A better solution would involve the denotation of metaphors between objects, formally known as morphisms. Methods currently must be itemized and limited by text identifiers, preventing the system from accessing higher-order patterns in a general way. For instance, current systems for handling objects in software cannot reify the concept of a binary symmetrical relation into a first-order entity whose model-level understanding agents may encode within the language.

The object-orientation scheme provides context shifts via instantiation of objects by various methods, as well as by the semantics of the objects operations, whether procedural or functional. The aspect of this paradigm most lacking in semantical integrity concerns meta-instantiation, wherein new objects are instantiated concerning lower objects. This notion of meta-systems existing at various levels with respect to a base context forbids model-level reflection in meaningful ways. For example, if a modern digital computer is considered the base context, then the current model of meta-instantiation does not cleanly allow structures to be built dynamically from objects for first-order identification with the hardware as a state-machine. It follows that the loops of programming reflection are not available for reflection via modeling within the universe of objects and their interactions. Such a restriction on reflective capabilities limits the system domain from exploring various aspects of its operation and use.

3.3 Declarations

Just about every information system contains some aspect of declarational specification. Some information in a system always will be interpretable by the user alone, particularly during the development of a system application. Traditional programming methodology casts such information in the terms of declarations for which the system must provide static guarantees. This concept can extend to support the model-level for a domain via an entire information environment that provides those guarantees in a transformable way. The extension essentially allows information transformations in such a way that they do not fall off the edge of the earth. In this way, all declarations may extend to apply at the knowledge level. At such a level, the declarations are relative to ontologies and their frames, so that the reliable translation of meaning for various ontologies is imperative in order to maximize the potential use of this information. Declaration-based programming is the information expressed in terms of relationships between logical atoms. Such information exists within a larger structure with access to the information that the declaration provides. This larger structure is traditionally the standard first-order logic of predicates, often restricted further to Horn clauses.

Because the traditional declarational paradigm does not vary from the standard first-order logic, and because the base model for reasoning it provides is central for the systems ability to analyze declarations, the misnomer logic programming is often applied. In truth, the capabilities of the underlying system of logic have no inherent need for Boolean operators and basic inference relations by themselves. Furthermore, the specialized languages that advertise constraint-based programming simply augment the systems domain in a fashion that leaves semantics integrity by the wayside. A more reasonable solution would most likely cast many simple constraint problems into the mathematical space of general linear problem solving and its special cases, such as convex linear problem spaces and NP-complete and NP-hard problems. Such a solution would provide a common ontology in which to express and convert problems from various domains in way that is computation-friendly.

The pure declarational specification model transforms all specifications of logical atoms into declarational form, so that for informal purposes, declarational programming completely reifies the implicit semantics of the system. The virtual machine for such a model consists of a system that only preserves relationships explicitly stated by the specification (the explication) and those required by the system of logic supported by the declarational system (the implicit meaning). This model only realizes its full potential once the underlying model of the logic system is available for modification. For instance, a new set of logical paradigms introduced in the last two decades provides a structure linking standard predicate logic to the minimal modal logic. Such a structure includes domains like dynamic predicate logic, multi-modal logic, arrow logic, and information-transition logic that include subvariants exhibiting decidability while possessing a quite valuable power to express concepts. Using these logics as a basis, model-level reflection at all levels of the system results in a powerful system for managing information. [1]

Current declarational programming implementations have achieved limited expressiveness and power, so that they are hardly suitable for general-purpose computations, due primarily to the limitations placed on their application to the appropriate domains. For instance, the state-machine for an unbounded stack requires an infinite number of first-order declarations, a condition for which current declarational systems can not account. Although a finite description exists for such a state-machine, such a scheme relies on close interactions between the meta-system and the first-order system, a strategy that is unpredictable in general among current systems.

A unified system for specifying declarations would combine the concept of specifying procedural logic and the idea of specifying static predicate logic concerning inert concepts.

3.4 Aspects

A current advance in the theory of meta-programming is the notion of aspect separation. This approach formally recognizes that the above paradigms fail to singly capture many design issues with their provided ontologies for building knowledge systems. Consequently, if the system is to address these design issues, the system must provide a transformed specification of the ontology, so that elements that address these issues by crosscutting the original ontology become part of the specification orthogonally. The transformed system of information exists within several spaces, each of which shares a name-space and uses a unique virtual machine to specify relationships obtaining in the domain. Preferably, agents should express each of these virtual machines at a high level of abstraction from the execution domain, since the intent is to supply information about the system that should remain invariant among various implementations. The integration of these specifications into a single unit specification tends to result in a loss of the meta-information that includes the domain ontology and the aspect ontologies. In traditional systems, the aspect methodology only occurs within the human mind, and so the computer has no access to this type of meta-information; the loss is therefore implicit and hence irreversible without considerable human assistance. Therefore, the more useful means to handling aspects is to cleanly separate the ontologies and define the means by which the system combines ontologies in order to produce results addressing all the issues without loss of system information. In current systems, the provision via human-directed development of orthogonally independent semantic models for the various aspect virtual machines addresses this issue of ontological separation.

In this model of programming, the appropriate paradigm captures the natural ontology for the domain in question within a specification. This specification intentionally leaves unspecified issues that do not directly concern the knowledge

[1] Elaborate here on the properties of dynamic logics, considering the basic concepts of modes from declarations to modalities and projections in the opposite direction.

level of the intended domain, such as storage management, execution speed, and communications constraints. Such issues should affect the translation of the domain ontology into working code. One or more aspect ontologies that address the process of translation between domains from orthogonally independent perspectives explicitly handle those issues.

3.5 Intensionality

Intensionality is a term relative to extensionality. The duality expresses the difference between systems that implicitly maintain referential integrity in knowledge specifications and systems that rely on human intervention to maintain the meaning of references, respectively. Intensionality guarantees referential integrity by replacing references to text identifiers with direct links (called hyperlinks). This referencing mechanism effectively raises efficiency of maintaining referential integrity within an orthogonally persistent system of relationships between code fragments. In this system, the user does not invoke first-order entities by simply specifying the name of the objects in the codes text, but instead establishes a link from the object invoked to the invocation site. Critical to maintaining such links is a system that can maintain these links upon transformations of the persistent storage system to preserve consistency. This opposes the relative inefficiency of a system that must parse text references, specify their name scopes, and invoke a search-and-retrieval procedure to bring the referenced code in hand for analysis and integration with the argument code. An ideal example for this line of thought is that, within knowledge systems, the wish is to distinguish what I mean to use from what I explicitly state to use. The eventual natural extension of this process is to distinguish objects by semantic specification, rather than denoting the product of an arbitrary action. This suggests that to establish the link, the system must search for an object with identical or similar semantics content. However, similarities (considered as morphisms) may be numerous, as they should be within a system that models many domains. Strategies for addressing this characteristic should require a careful attention to the management of pattern recognition. A preliminary suggestion is for the system to posit a new independent object within the specified domain and to establish morphisms later, factoring out redundant information as encouraged by an optimizing activity. Morphisms also could be multiplexed through objects, yielding a simple generalization of the inheritance metaphor in traditional object-oriented programming.

The work in progress on the Napier88 and TemplateNapier persistent hyperprogramming language systems best exemplifies research along these lines among programming languages. Additional research in the field of visual programming, and, more specifically, direct manipulation concepts therein provides ample evidence of the conceptual simplifications for a computing system gained due to intensional interfaces. The operational behavior of such systems is often far simpler to understand and manipulate consequently.

4.6Incremental development

Incremental development is the application of fine-structure analysis to the logic of the evolution of a knowledge system. The progress of an agent toward defined (perhaps evolving) goals tends to create a lot of information. Traditional systems discard most, if not all, of this information, since the usability of the derived information by other agents is severely dependent upon human effort. Such a notion runs counter to the production of a long-term information system or knowledge system, in that the information gained by analysis of a difficult problem often yields beneficial results for other domains. One can imagine an inventor working for years on some difficult problem, and then, upon discovering the solution, throws out all of the unsuccessful attempts as though they were in vain. The utilitarian viewpoint suggests that all worthwhile effort be available in some form to others to avoid having to duplicate something that may turn out to be vital in another field.

The current programming method to retain this information involves partial evaluation. This scheme consists of dividing the translation of a specification from one context to another into small incremental parts, as seen by external agents. The result is a structure of specifications arranged in a network flowing from the source context to the target context, with all the specifications (and hence, contexts) available for reference, and hence for re-use. However, the current computing tradition of specifying contexts manually in text-based interfaces tends to discourage this re-use, since unique identifiers must be posited for each new information structure to be re-used.

Incremental development as a useful methodology applies to every domain of semantics. For instance, syntactical structures contain information in their sub-structures. Traditional text-based programming environments often redundantly encodes this information within many structures, as evidenced by contrast with the relatively high compression ratios of adaptive sub-tree compression methods applied to source code in existing implementations of the Oberon system. High compression ratios in themselves are not a worthwhile goal; enhancing overall system utility by maximizing information availability, however, is such a goal. For the block-structured language Oberon, the information in syntax tree structure is directly useful for performing very late optimizations, wherein run-time information is available to the program specification structure for efficient, high-level code transformations.

3.6 Abstraction

Abstraction is a concept relating both to basic and model-level reflection. Its domain includes the transitions from first-order systems to their various types of meta-systems. More generally, abstractions are those classes of information transitions that result in factorization of the semantics implicit in a system. They factor heavily in the construction of useful loops of reflection, in that the semantics of the output of the abstraction translation approach the semantics of the knowledge level. Ideally, decidable abstractions should link the domain to various ontologies in a direct manner.

The classic algebra of abstractions, the lambda calculus as introduced by Alonzo Church, is a general system of functional specifications. While the notations of the lambda calculus involve the traditional variable-as-state model for computations, the calculus itself is an abstraction over the syntactic formalism that yields an intensional model. Therefore, the true form of the lambda calculus identifies lambda terms with equivalence classes of formulae modulo alpha-conversions. The intuitive idea of an alpha conversion is that the actual symbols used in denoting the bound variables in calculations (atomic actions) are irrelevant to the meaning of a lambda term, allowing arbitrary re-assignments that obey basic laws. This is in contrast to beta reduction, which considers the free variables for re-assignment; such a model limits the lambda form from applications to typed logical atoms. For example, the natural (informal) lambda formalism for expressing the sum of two integers is essentially lxy.(x+y), where the variables x and y are bound for the term and lose their symbolic character according to the lambda calculus. Logicians term these bound variables as alpha variables. They constitute the first-order level for the context introduced by the lambda term. In this way, what remains of the variables in the term is merely the concept of preserving the flow of information concerning bookkeeping. The interpreter merely must refer to the first input variable for the first term of the sum and respectively for the second. The free variables of a lambda term are those defined by the context into which the agent posits the lambda term. Logicians term these free variables as beta variables; they constitute the state of the information of the parent context. In this way, actual lambda calculus consists of the study of lambda terms as described, modulo beta-reduction, so that the relationship of a functional specification to arbitrary contexts is the essential subject. In the example given, the lambda formalism merely maps the addition operator into the space of lambda terms.

The result of this abstraction process is a functional metaphor for computations that is intensional in meaning. As such, the lambda calculus is a highly attractive formalism for expressing most abstractions. Few implementations, however, completely achieve intensionality from the users perspective. Furthermore, many undesirable assumptions linger in present systems due to the constriction of the domain by the system developers. Church and others who initially considered the properties of this functional theory were only concerned with formalizing the theoretical capabilities of the kinds of systems that extant computing systems emulate. The lambda calculus is a formalism that identifies in its domain all finite recursively-definable functional structures. The practical significance of a formalism that can distinguish those algorithms of which a finite-state machine is capable cannot be over-estimated. However, this formalism simultaneously fails to express those algorithms of which modern machines are incapable. The assumptions used by Church and others to formalize the systems model express this limitation well. These assumptions limit the domain of application strictly from describing non-calculable structures, which a modern information system encounters quite often in its work. Because of these limitations, the lambda calculus alone is insufficient for knowledge systems as described above, in that human-reachable structures will be quite unmanageable. (Notice that the goal is for the machine to assist the human-reachable tasks by managing the information involved, not actually attempting to perform such tasks as specified.) Even among less capable systems, it seems useful for the system to express to the user when it cannot handle some type of action directly. However, the computer system should make itself available for discussing various aspects of actions that it cannot complete, and to manage the information gained as knowledge.

Note that the lambda calculus employs the intuitionist logical model for context structures. This is no mere coincidence. Lambda terms are famously identified by Church as the effectively calculable functions with the intuitively computable actions of a machine with atomic (non-recursively divisible) actions. Since the generally recursive functions are equivalent to the set of lambda-definable terms, it then follows that lambda terms model exactly those computations (actions) of a finite atomic nature. Church's famous thesis (Sieg, 1997) identifies those computations with the recursively definable computations, which the intuitionist logic models precisely. However, a model-reflective system's intended domain easily exceeds the actions that it may take, since the system must model at least its tasks, which are human-directed. For humans, the concept of an action that is recursively dividable is as intuitive as the conceptualization of continuous motion in space, since logically, transitions in spatial position are divisible into infinitesimal units. It follows that they naturally overlook the difference between intuitively conceptualizable and intuitively definable for calculations.

As an algebra, it has difficulty expressing important human-level activities, such as instantaneous pattern recognition and conditioned responses. Moreover, it seems natural that a model-reflective computing system should understand the operations of what constitutes user-interface in modern graphical interfaces: the abstraction from a bitmapped display of information presented by the software system. If a system can prove that its graphical interface processor communicates information in a way intuitive to the users ontology, then it can modify that interface in a way that preserves those properties.

General-purpose equivalence class notation (i.e. at every abstraction level)

Current systems lack the ability to express ontologies as first-order vocabularies for domains coherently and to allow for ontological crosscutting in a clean and safe manner.

3.7 Primitives

An overriding characteristic of current-generation software systems of any kind is the implementation of standard mathematical and logical structures as primitive objects. The advantages of having the standard numerical calculation system available as a set of first-order constructs are obvious, since they allow the environment designer to provide relatively direct access to the capabilities of the underlying hardware. However, the distinct inability of these systems to immerse the context in other mathematical or algebraic systems makes obvious their limitations for dealing with information structures in general. Furthermore, such a system historically leads users to cling to direct control of the machine, and for developers of the systems to respond with fewer improvements in the area of abstractions. The overall result expresses the limitations in utility of common software systems and their immediate derivatives.

To that end, software systems should enable agents to encapsulate arbitrary models of various formal objects. For instance, a system that understands the mathematical concepts of 3-dimensional rotations and translations can use this knowledge to build algorithms that manage this process for the user in an abstract way. However, if the system derives this ability from primitives, not structures thereof, then the generalization of 3-dimensional geometry to space-time geometry or Riemannian geometry will be unattainable unless other constructs compensate in their abilities. This limitation coerces the system user into managing the entire conceptual and resulting procedural framework due to any new domains the user would introduce. The utility of the abstract form of the models is that such models translate to various hardware and software architectures in ways that the software can manage automatically.

This lies in direct contrast to modern systems where the most abstract of mathematical systems, the symbolic algebra systems, encapsulate as their highest-level symbols of abstraction the countably-complex vectors with combinations of elementary mathematical operations over them. Furthermore, those systems fail to encapsulate those symbols at a first-order level, and thereby limit the changes available for the fundamental model to those allowed by intuitionistic logic. Even those systems are insufficient to reify or abstract any notions built from equivalence-class creations or morphisms.

3.8 Finite size

The intuitionist virtual machines internal state can be characterized by a finite but unbounded stack and a single register for manipulating that stack with respect to a pool of random-access data known as the heap. The unbounded assumption is just included for simplicity of the axiomatic definition for the machine. The uniqueness of the stack is irrelevant as well, since modern systems using multiple stacks only achieve a faster implementation instead of higher-order modeling capabilities.

There are two essential assumptions that exemplify the limitations of the model. First, only a finite number of objects may be on the stack at any time. Also, any atomic action may add or remove only a finite number of objects from the stack. If the model for the system uses the second assumption, the addition of a simple assumption allows the derivation of the first. Namely, if actions are not recursively divisible without limit, then it we may easily derive that an agent may ever perform only a finite number of actions, and hence that the size of the stack will always be denumerable.

From these two, we can infer that the system would not consider programs those sequences of atomic actions that are infinite in number, since this would allow violation of the finite stack-size principle.

If the context includes an arbitrarily infinite size stack, then the stack can no longer be indexed by an integer register. The register for such a device would therefore be a state machine of arbitrarily infinite cardinality in order to maintain the indexability of the stack. The semantic system that results is necessarily one that can index its own complexity in terms of itself. This also allows effectively random access to the internal state of the machine. It follows that the semantic model of the stack system does not permit the use of infinitary extensions.

A simple axiom of infinity also allows us via standard logic to provide the system a framework for discussing the decidability of properties of objects. This is useful in allowing the information system to recognize and reason about the limits inherent in its various substrate devices (particularly digital logic microprocessors). It follows that if the system can recognize those limitations as first-order entities, then it possesses a means for avoiding traditional logic faults in software systems, commonly known as deadlocks.

3.9 Model-Level Properties

Tasks for the system are states that agents modeled but cannot or have not constructed. Examples of this category of knowledge for a model-level reflective system include input from the user, the management of an infinite-state device, and all data from unreasonable sources. Traditional systems cannot manage the information provided by such domains, and therefore fail to overcome complete reliance on the user for much of the information necessary to construct and maintain the system. Hence, systems that cannot support general-purpose model-level reflection will not succeed in terms of overall utility for society.

The scheme for implementing the notion of a model-level description of the system is as follows. The agent is the information system as a whole or taken as a consistent part. The agents models consist of the interaction of collections of information atoms. The agents methods include the functional semantics that the construct can provide. The agents tasks are actually models of the desires of systems with which the reflective system interacts, but does not fully understand. The constructs used must be broad enough in scope to achieve this range of interpretability.

4 The Proposal

This paper intends to present a cybernetic system that fulfills all the stated requirements for an information system that provides system-wide model-level reflection, as well as the properties necessary to manage a complete knowledge system. As such, it should provide an excellent substrate for the development of artificial intelligence prototype systems, including expert systems and knowledge bases with a far greater utility than conventional systems. It also trivially supplies the means for a unified system for computing systems with high utility and an appropriate and adaptable conceptualization system. The goal is a unified system capable of modeling any concept, including those that reflect on the system domain. Pragmatically, the system may also serve as an inter-system translator for the various types of information stored on modern software systems, using a notion of arrow definability of information. In this way, much of current informations systems and their content would be available for re-use by any Arrow system that models them. It should be clear that the system should support the advancement of the sharing of information and knowledge in terms of human-interested utility in as much as the necessary implementation allows.

4.1 Basic Metaphor

4.1.1 A Binary Relation

The system is a specification of atomic relationships (called arrows) between objects that are themselves arrows as well. The translation of the notion of a logical binary relation is therefore a set of arrows. This yields an expressive capability that easily rivals the relational algebras of mathematics. The system enables a function type by specifying a left-deterministic relation, allowing arrows to be composed into functions as well.

This yields a constraint-based or axiomatic programming construct. The total set of constraints by other arrows that applies to a given arrow constitutes its definition. However, the constraint metaphor provides an arbitrary abstraction level, since any object that the system can postulate is available as an arrow for declaration of constraints.

Although the system provides the metaphor of relational declaration, we can see that the world of arrows that an Arrow system will implement has a far greater capability than those standard systems of specification. For instance, within a complete information system built from arrows, each arrow is available for statements constructed by other collections of arrows. This property enables arrow models to build the definitions of objects constructively and incrementally.

The system that does these things functions quite differently from ordinary computing systems. Its base semantic paradigm constitutes a reasonable theory of information atomicity in terms of cybernetics, although it is trivially recognizable that these information atoms are far from indivisible in the ordinary sense. As shall be shown, the ability to identify these atoms of information with arbitrarily posited patterns of other atoms results in the ability to crosscut ontologies in first-order ways that are often calculable. In this way, the system easily reifies, studies, and manipulates the mental process by which users ordinarily build information and knowledge systems.

This lies in direct contrast with the de facto standard for information atomicity, the bit. The bit embodies the von Neumann state-space architecture rather succinctly, and as such is limited from many of the higher-order abstractions of which arrows are capable in a meaningful way. More specifically, the bit concept generalizes to spaces of necessarily enumerated states, so that the concept assumes a linear ordering and denumerability among its first-order atoms.

4.1.2 A Homo-iconic System

Any application of arrows to other arrows constitutes model-level reflection. The arrow construct itself forms the basis for reflection via object-level / context-level distinction. This common construct has such little semantics implicit to its definition. Therefore, though much information that agents would considered unreachable because it is sub-structural or crosscutting to the model specified by the semantics of ordinary languages; it is reified easily in this new system for first-order reuse. This allows the Arrow system to act as an effective meta-system language providing inter-language translation up to the limits of computability via a uniform modeling mechanism.

Although the arrow construct may be intuitively seen to model transitions of information between languages, it also provides an additional metaphor, since the nodes representing the pair of languages (actually, state-machines) in the translation are actually arrows themselves. These node-like arrows again model another information translation, since a language represents an interface between systems.

4.1.3 Not a Language

The Arrow system blurs the distinction between context type and other types. Essentially, the system renders all types in terms of arrows. Only a context can yield meaning for a given arrow. Such a context is due to a type of abstract virtual machine built dynamically by the environment from the specifications of ontologies.

A natural view for the Arrow system involves generalizing the notion of a hyperlink to include all invocations (and references) both above and below the first-order level of a given language. Therefore, all code consists of structures of hyperlinks, individually reified as arrows. The result is an abstract diagram that mirrors data-flow in its interpretation, since it links the first-order objects of an ontology (for, say, a programming language) to their invocation points. The structure relating the invocation points is an abstract version of a syntax graph, which again agents may easily construct from arrows.

The semantics of an Arrow system specification will have a functional character itself (mirroring the intuitive interpretation of an arrow as an ordered pair specifying a function or a lambda term), to provide a clean way to produce correctness verification. If presented with a sequential representation of an arrow specification, the system should separate the information inherent in the sequential form, unless the agent states that the form contains information used by some part of the system.

4.2 A Simple Construct

The simple visual concept of an arrow is a directed link between two points. This simple visual metaphor should aid in representing the state of the system and its parts in an easily understandable form.

A formal metaphor for specifying arrows should consist of viewing arrows as data structures with exactly two slots that are ordered. For the sake of convention, this formalism denotes the first slot by 0 and the terminal slot by 1. This suggests a C-language struct syntax resembling the following: A={x, y}, so that A.0=x and A.1=y. However, the preferred method for representing and specifying arrow incidence relations is by intension; by factoring out the necessary text-identifiers from a textual specification, agents may achieve such an abstract level in their models of arrow structures. In an interactive system, however, the natural state of system interaction would be solely intensional, with text-laced interfaces overlaid for ease of use only.

These slots can only contain references to other arrows, but a slot cannot contain a reference to another slot directly. In this way, all arrows lead from arrows to arrows, not to or from an arrows head or tail. To implement the case where an arrows head points to the location of anothers tail, one must instead have both slots refer to a third arrow that acts as a node with respect to them. A reference may also be empty, that is, containing a reference to nothing. Note that the semantics of the system would be identical if empty references were disallowed, given that the environment simulate the concept by replacing empty references by references to a unique arrow with no intrinsic meaning. Such a simple reification as this suggests the immediate utility of context shift for a system of arrows.

4.3 Graphs

The natural ontology for viewing the Arrow system consists of collecting arrows into various groups. Some kinds of these groups are similar to directed multigraphs of arrows and nodes as in mathematical graph theory. Graphs are not intrinsically special types in the Arrow system, instead merely collections of arrows. As such general abstractions, they constitute a natural construct for reflection among arrows. In this way, the user can apply various abstraction schemes to collections of arrows as basic derivatives of set theory. Since arrows model declarations in traditional specification languages, this action can work at the model-level.

In mathematics, a graph is a way to specify the relationships between selected objects viewed as nodes. In the Arrow system, the nodes are really other arrows within the system. Taking general relational algebra or category theory as the logical system, each graph corresponds to a relation. In this way, the natural theoretical view of a graph is as a set of arrows, per modern set theory in formal logic and mathematics. (To be accurate, the most natural view takes Fraenkel-Zermelo set theory to be the fundamental model for the system from the viewpoint of first-order logic.) It should be clear that the nodes of the graph, that is, the arrows referenced by the relational atoms, do not necessarily belong to the graph, although there is no inherent restriction on the references an arrow may contain. Therefore, an arrow of a graph may reference another arrow within that graph. We shall consider graphs of this nature abnormal, in that they do not respect the relational metaphor cleanly for first-order systems. Graphs constitute reflective-order relations, since each arrow represents the application of a relation symbol to an information atom, and interactions between these relation invocations constitute a higher abstraction level for the relation. In this way, an arrow that references itself as an information atom results in an unlimited recursive application of a relation. Curiously, if the arrow refers to itself as a relational atom, then it generates a recursive application of an anonymous, meaningless function. Such graphs may generate contradictory specifications unless the creative agent pays close attention to the relationship between the first-order and meta-systems specified by such a graph. In addition, a graph may actually contain arrows with empty references, as well, so that the arrows of the graph may be dangling or free-floating, so to speak. These degenerate graphs are certainly useful although agents find the relational metaphor contradictory via standard first-order logics.

Graphs may also represent featureless sets, viewed as a graph of nodes in category theory. They correspond to meaningless relations, if agents impose the first-order relational metaphor, since they consist of arrows with empty references only. Note that the usual notion of a graph still falls under this interpretation, since all are merely collections of arrows.

The system may group arrows into graphs in completely arbitrary ways. Many graphs may contain a given arrow. Arrows may contain references not only to nodes, but also to other arrows within the graph. Only an interface to the system may provide restrictions on the methods of grouping. These interfaces algebras are constructible from within the system. These interface algebras constitute a system for building ontologies from existing information. Since our grouping mechanism is completely unrestricted, the ontologies that are possible in the system may crosscut each other, as well they should! In fact, the design of the Arrow system should encourage the use of crosscutting in order to maximize information availability for the system, such as reifying the most attributes possible over a collection of objects.

Taking the relational metaphor as a basis for construction, the grouping mechanism can be modeled itself by a set of arrows. These arrows model atoms of information regarding the set-membership relation Î. In this way, the representation of a set in the Arrow system treats both the set and the elements as the nodes of a graph. The arrows of the graph all lead either from the elements to the set in question or vice versa; the issue of arrow direction per relational model is arbitrary as long as consistent. In this model, the statement xÎy identifies with a (possibly unique) arrow leading from x to y, stated as {x, y} or x®y. Notice that this notion primitively reflects on the model of the simple abstraction mechanism, the graph. The construct thereby enables arrows to contain references to graphs themselves. This concept generalizes to a scheme that can reify any concept as the node of a graph. To begin, imagine that for every graph there exists at least one meta-graph: a graph that describes the original, in terms of set theory in the present case. This graph maps the relation symbol to its invocations in the original graph; for each arrow in the graph, there corresponds exactly one arrow in the meta-graph, termed a meta-arrow. The meta-arrow for the original arrow in the graph would be stated as {Î, {x, y}} or Î®{x, y} or Î® (x®y) equivalently. Note that for each meta-graph, there exists a further meta-graph for that graph; notice that the resulting extensional structure is an infinite recursion process, which is often termed meta-regress in traditional systems. This process evidences the need for an axiom of infinity to conceptually distinguish those constructions that the system should implement lazily. Note also that this abstraction is by no means the only type of meta-instantiation possible; it is merely the simplest in terms of conceptual separation. In fact, we may describe any graph within a coherent Arrow system as the meta-graph of all the graphs containing arrows referenced by the arrows in the original graph. In this way, the degenerate graphs described are reflective graphs, being graphs that crosscut the usually traditionally clean separation between the meta- and first-order levels.

This is not to say that arrows must always be explicitly instantiated in order to construct a relation. It merely suggests that relations in the Arrow system consist of indivisible information atoms, which are arrows. If an inference system exists for dealing with arrows, then the extant relations applying to a given 'node' could define a new relation in target domains. The existence of an arrow as a member of this axiomatically defined relation could be determined dynamically. In this way, relations that apply to an uncountable number of objects could be implemented effectively and retain arrow definability for a finite store. This technique is hardly novel. Many existing systems employ this idea of evaluating the structure of objects only on demand, known as a lazy evaluation strategy.

Finally, it should be clear that graphs considered as isolated entities would mean very little to a knowledge system. The modeling capabilities of graphs encompass their entire purpose in an information system, so that the system would naturally maintain links between models and their intended subjects. Furthermore, a graph within the system has very little meaning if it is bare; that is, if its arrows are not annotated by some structure that develops the meaning of those arrows. As an example, consider figures one and two below, which depict the diagram for a state-machine and its syntax-level abstraction. The extension of the graph of the automaton to its syntax graph (reifying basic arrow-level relationships in the first diagram) represents a shift of information content from the text identifiers to arrow structure. The system extends this shifting process to encompass the information that is textual even in the syntax graph, until all the information necessary for the system to encompass semantics of the diagram exists in terms of arrows alone.

4.4 Logic

The logic of arrows as expressed here has only been analyzed in the last decade as part of the larger field of logical research that intends to explore the various types of logics between full first-order predicate logic and the less expressive, though decidable, minimal modal logic (van Benthem, 1996). Arrow logic explores the notion of the definability of processes and information transitions in terms of these logical atoms. The logic of arrows is actually a family of logics whose atoms may or may not be associated with ordered pairs or tuples of points. Note that the Arrow systems as presented here fundamentally differs from the current subject in logic in that the underlying nodes in Arrow graphs are also arrows that are subject to inclusion by any graph in the system. This yields both homo-iconism and a clean mechanism for reflection by the system at all levels.

The fundamental notions of arrow logic involve some simple predicates over featureless logical atoms called arrows. Just as in the Arrow system, these objects fundamentally represent specifications of binary relations. The three canonical predicates specify identity, inverse, and composition relations over those atoms. Arrows themselves may be thought of as corresponding to ordered pairs over an underlying space consisting of states (viewed as nodes) with the arrows representing the allowed transitions in the system. However, the idea of pointless arrows is just as valid a model. The fundamental notion of a collection of arrows, known as an arrow frame, corresponds precisely with the introduced notion of a graph in the Arrow system. The reason for the shift in terminology is to effect a different point of view on the utility of arrows and the modeling capabilities they engender, and to clarify the difference between ontological frames and arrow frames. Current research practices consider arrow frames in isolation, whereas the graph is intended to be an atomic module of information for a new class of high utility computing systems. As such, the primary focus in the design of these information systems involves the interactions between the graphs, since they constitute the refinement of information within the system until it approaches the level of knowledge.

<< Decidability, axioms, modalities, and stuff (i.e. get material from the books on logic!) >>

This range of semantic flexibility achieved <<>>

As for arrows in particular, the fundamental relations considered of interest for the Arrow system to extract and describe information from arrow collections are the three canonical relations, the incidence relations, and the reflective relation, specified as follows.

4.4.1 Canon

These relations describe the essential features of arrows only from the perspective of the usual relational metaphor, without regard to the internal structure of arrows.

The identity predicate applies to all arrows whose endpoints are identical, represented in this way:

The inverse predicate applies to pairs of arrows whose lists of endpoints are identical when reversed.

The composition predicate applies to arrow triples where the vector addition metaphor applies to the first two to yield the third.

4.4.2 Incidence

Incidence relations depend upon the structured view of arrows, which is the model of arrows implying association with ordered pairs over an underlying state-space. These relations express the possible combinations of relationships between arrows and the objects to which they refer. Again, these assume a clean relational metaphor.

The incidence relations

The coincidence relations

4.5 Categories

The mathematical notion of a category relies directly on the directed multigraph construct, just as the non-degenerate graph does in the Arrow system. It naturally follows to point out the similarities and differences between the two ideas, particularly considering the logical and modeling characteristics.

Categories assume a fundamental difference between the notion of arrow (as morphism) and node (as object), so that the theory of categories cleanly separarates the notion of system and meta-system by restricting categories from crossing that boundary. In category theory, the arrows and nodes are fundamentally different types.

Within categories, the notions of identity, inversion, and compositions of the arrows are central to the model, as they are to the Arrow systems central logical model. However, categories operate within the context that all arrows in the correct configuration compose, whereas the fundamental issue for composition within the Arrow system is whether (or where) arrows compose to form logically valid relational atoms. Furthermore, the Arrow system does not assume associativity of the composition relation, except with respect to identity arrows. However, both systems regard inversion as a subject for logical analysis. The main reason for the differences lies in the fundamental metaphor for the node concept between the two systems. Within categories, nodes describe strongly (or trivially) typed logical atoms, that is, that internal form of a node defines its meaning. The Arrow system, on the other hand, considers nodes to simply be the subjects of informational atoms, and to have no inherent meaning. The Arrow system constructs of meaning on its own.

Now, the modeling benefits of categories are very extensive, in that categories easily encapsulate many functional paradigms as well as the majority of intuitive structured processes, including the typed lambda calculus. It therefore seems rational to implement categories using arrow constructs and include the relations it defines among its structures as graphs with implicit meaning. In logical research, this implementation already exists in terms of categorial grammars for arrow frames, allowing the analysis of the models that categories produce.

4.6 Multi-Arrows

Another concept for the Arrow system is the basic notion of chaining together arrows, providing arrows with arbitrary numbers of references (even infinite numbers). The basic metaphor for an N-reference arrow (or a multi-arrow or N-arrow) is a relation over N-place tuples of objects. It reifies the basic computational result of currying. This allows a simple system of arrow types for expressing logical structures of greater complexity over the base system of (2-place) arrows by a relatively simple reification operation. This mechanism even allows for the simple reification of the graph notion using multi-arrows only. Since a graph corresponds to the usual notion of a set, the notion of a collection facilitates the remaining discussion. A collection in the Arrow system is a graph of references to the members of a source graph, so that we have an unordered group of (possibly repeated) references to some unique objects. Similarly, sets are collections modulo repeated references. An agent in this context may consider the graph for a collection of arrows to be the equivalence class of all multi-arrows with references to exactly the members of the graph, modulo the full linear ordering of the references. (The visual modeling of the meaning of previous sentence within the Arrow system is left as an exercise to the reader.) Paradoxically, the definition for multi-arrows may derive from the notion of a collection of arrows, over which a relation specifies a linear ordering. The duality of these viewpoints is desirable and exhibits the natural ability of the Arrow system to support ontological relativism in a simple manner. It also exhibits the suggestion that the cardinality of multi-arrows be as unrestricted as is the usual notion of a set.

As for implementation, any abstraction mechanism that exhibits the same characteristics as a multi-arrow will do. For instance, if the domain in question consists of graphs of Lisp-like linear lists of references, then it behaves as a multi-arrow environment. These lists are simple to construct given the isomorphism between the arrow primitive as an ordered pair of references to arrows and the Lisp construct of an ordered pair of pointers. Since a set is actually a graph of the set-membership relation, each set is available for reference as the source node of this graph, allowing multi-arrows may contain references to other multi-arrows. Upon this structure, an agent may posit the desired relational structure to elaborate the issues for discussion. In this way, an entirely new primitive context arises from a simple scheme of graphs. This new context behaves as an isolated environment with respect to its own relationships and constraints. However, it also can model the environment used to create it, using information-theoretic proofs about the context of two-reference arrows, and thereby reflect upon it.

Note that this new context includes multi-arrows with both zero and one references. This allows for increased expressiveness concerning the original context of arrows with two references only. A zero-reference arrow represents an effectively empty statement, true in all cases, while the single reference arrow reifies the references that arrows contain as a first-order atom.

$<<$ Elaborate on the generalization effects on the logical arrow relations. $>>$

4.7 Contexts

Arrows rely on variation in context for meaning. However, variation in context is not under the state-update von Neumann permissive model or its derivatives. The system may and shall provide the simple dynamic semantics of the functional paradigm as well as the clean semantics of declarational specification. In this way, an ontology specification introduced to the system results in the incremental modification of the current knowledge base until the state of the system realizes perceived ontology. Because of this, agents can manage the interaction of various ontologies in a manner heretofore unattainable.

The general paradigm for using arrows as a tool for modeling information and knowledge systems involves a relational metaphor. In the Arrow system, we would identify a binary ordered relation as a set of arrows. One arrow of the set represents the affirmative result that the relation applies to an ordered pair of objects. The nodes connected by the arrow are identical to the objects considered. A simple system of relational algebras allows the description of arbitrary kinds of relations between objects. The notions of relational inversion, composition, and identity enhance the expressive power of the arrow metaphor. Since the Arrow system allows infinite numbers of objects, the system may express the characteristics of very large domains. Examples include number theory where the natural, integer, rational, real, and complex number systems form an intricate and useful structure for a knowledge system. Morphisms, acting as metaphors, may be constructed, allowing the system to represent any large systems similar or measurable by these numbering systems. In this way, the system may manipulate and understand any formal symbolic theory.

Since we may build any conceivable single relation, we may assume that the interaction of these relations would produce a declaration-based system of description for first-order objects. In this way, an external definition of an object develops incrementally and constructively, using the same methodology as a theoretician would in formalizing the notion of a system.

However, this metaphor has more power than the previous description suggests. Since the Arrow system possesses a very simple, clean underlying layer of semantics, agents need assume very little in the logical structure of the systems substrate. This allows the Arrow system to delve into areas of reflection previously unattainable by current programming and knowledge-modeling systems. For instance, within current systems, the systems designers enumerate the types of reflective actions that agents may take. The system therefore does not achieve the necessary level of reflection in order to overcome these inherent limitations; one can imagine a reflection barrier beyond which the capabilities of software systems could grow quickly with minimal human involvement. The Arrow system design intends to transcend this limitation in current reflective capabilities by fully supporting the principles of ontological relativism at a first-order level.

Arrows create contexts constructively in our system. Since the primary metaphor for arrows is the declarative specification of a binary relation, it seems natural to use binary relations to specify the rules and axioms for a logical system. The remaining necessary construction consists of an evaluator that obeys those rules of inference and axioms of specification to yield results to a querying agent. However, though an evaluator must exist for every new context, by no means should agents consider evaluator-ontology pairs unique for a given ontology. In this way, the common bulk of the system will guarantee and avoid unnecessary restrictions to information accessibility to a particular idiom.

A lack of restriction on the types of evaluators permitted to access ontologies should provide a wealth of semantics. For instance, one could consider a graph over ontologies, considering ontologies as nodes, and the arrows between the nodes as the evaluation methods between them. In this way, a system actor can sequentially combine evaluators to produce new evaluators for any ontology in the system. This notion is conceptual generalization of the mathematical notion of a category.

<< Insert ontology transition graph here. >>

<< Nodes: hardware machine-state language (internal and peripherals in both state-space and time), ASCII text, assembly language, high-level languages, operating system or virtual machine state language, interface specification languages, graphics display interface languages, drawing primitive languages, user interaction ontology language, and more. >>

<< Arrows: interpretation, compilation, assembly, reverse engineering, and various representations. >>

It is very interesting to note that the construction used in the diagram consists of arrows, providing the first of many examples of the utility of the construct. In addition, the Arrow system may obviously use this diagram to model and manipulate itself and its processes, since it provides a set of first-order handles for identifying and invoking these information translation services. This ability forms the practical beginning of model-level introspection and intercession. Not only is the graph and its implicit structure available for first-order analysis, but also the system may augment this graph by positing new ontologies or even splitting and joining ontologies in arbitrary ways.

As can be seen, the generated system of abstraction does not promote the metaphor that it provides abstractions in arbitrary layers above some substrate state-machine. Instead, abstraction is now an action taken in arbitrary directions. In the new system, the substrate state-machine is merely a new specification of an ontology, and evaluating some specification in terms of the state-machine ontology provides execution. The system may provide evaluations in the direction opposite of the execution directions, with various semantic meanings. For instance, an evaluation from one hardware context to another fits into this model, and is often decidable; witness the utility of emulation software for prototyping hardware and porting software between platforms. However, evaluations may also be proposed wherein high-level pattern recognition is required, such as the traditional idea of reverse interpretation between programming languages, which is undecidable in general. Note that the system can easily encapsulate such an action, though modern hardware may be unable to perform the action or some of its parts as specified. Note, however, that currently the state of hardware research is rapidly exploring the possibilities of devices that perform calculations far beyond the conceptualizations of Church and Gödel. The data introduced by such devices are obviously untenable for analysis by current computing systems in a useful way, evidenced by the specialized nature of these devices interfaces and the lack of a coherent theory for abstracting upon that data. It is for these devices as well as for humans that the Arrow system design intends to serve.

This graph may also apply to traditional systems. Modern systems deal with a restricted subset of the diagram used above, whereas the Arrow system shall model arbitrarily specified new arrows and nodes. To represent this restriction, it must make obvious the parts of the graph which traditional systems represent, and those that they ignore. Traditional information and knowledge systems focus on a finite set of high-level nodes representing program code representations and standard data protocol representations (encoded within the code representations). The arrows concerning traditional systems are at most those extending through that central set of nodes via all other available ontologies, so that they constitute the direction of implementation. The vast majority of modern systems do not even appraoch the limits described here. Such systems also express those arrows that implement ontologies that are more abstract than the base set. The most general-purpose of these systems are the homo-iconic systems wherein those arrows extending from the central sources of abstraction constitute all possible information transformations. Computer scientists refer to such systems as exploratory in nature, in that they make available all ontologies to the user in meta-order form, via first-order specifications of implementation. The parts of the system labeled reflective constitute the system implementation. As such, the user necessarily performs part of the representation of the periphery ontology as he or she reads the code. Notice that the notion of such systems traditionally assumes the star pattern which this discussion elucidates, and that this pattern is isomorphic to the notion of a hierarchy. It therefore constitutes an explicitly limited domain of information management. The other possible information transitions are not available in this scheme, even if they represent computable translations of information. Furthermore, the system only makes available the static set of central abstraction ontologies for first-order use, that is, implicitly requiring that the system multiplex implementation through one or more of the central abstraction ontologies.

$<<$ Insert the modified graph, highlighting the star configuration. $>>$

$<<$ Also, note the multiplexed implementation routes from specification to implementation. $>>$

This formalism for expressing the contextual expressiveness of modern information systems suggests a proof (both cybernetic and topological) of their limitations in terms of utility due to their inability to modify the central static set of ontologies. Such a modification would thereby express ontological relativism at a system-wide level, and enable model-level reflection.

Using the concept of a graph of information transitions over ontologies, it seems natural to propose the following model for context management. Constructive specification from within an existing context helps create new ontologies, and therefore contexts. Arrows represent the generated ontologies within the diagram, as suggested earlier by the definition of an ontology. Other arrows link the arrow in the ontology graph to the ontologys specification, contained within the target domain. Since ontologies represent processes of translation (or interpretation) of information in the source domain, multiple arrows may exist in this diagram per pair of contexts. In addition, an intuitive idea is the use of sequential composition of ontologies, which would provide a combinator for generating new ontologies via the analogy of vector composition. Such an operation represents the gluing together of ontologies to extend the accessibility of information among domains in an automatic way. An agent may inspect this operation for its properties in preserving information as well as its characteristics for maintaining consistency among domains. Potentially, an agent may even construct inverse ontologies that provide reversal of information translations; this is easily possible if the original translation results in no loss of information. If the inverse of a translation requires no extensive pattern recognition of the underlying hardware, then such an inverse is calculable as well.

What results is a natural system for expressing domains and ontologies within the frame of graphs.

4.8 Meaning

Meaning describes the dynamic effect of information received relative to the current frame of knowledge. Similarly, truth within a system of logic is local to a knowledge frame.

The means for creating an ontology within the Arrow system entails an understanding of the means for state machine construction within the Arrow system, to enable the understanding of the state machine concept within ontology descriptions. State in traditional systems implies that a certain valuation exists for the variables of the machine. Well-defined machines have variables whose allowed values satisfy certain constraints. For instance, standard computational models include as their basic variable types the register types of the underlying hardware architecture. Note that the Arrow system design intends to model all of the conceptual levels of a system. By including the entire model of a type as a first-order object in the system, variable assignment consists of arrows that select the appropriate element from the structured set of possible values for the type. That set constitutes the extension of the type in the usual terminology. The means for predicating the type of an object is the use of an arrow-style mapping of the kind just introduced from the object to the intensional (axiomatic) description of the type. The creation of a type system then enables the construction of arbitrary state machines, which formally specify a language identical to the state-space of the machine.

From this beginning, the means for ontology construction consists of specifying a set of nodes that form the constructs for a contextual vocabulary. The relations that directly refer to a node of this set are those that help define it. The closure of specification due to relations that refer only to nodes in the argument set is the ontology of the set. From this method, an agent may create all kinds of specifications whose entities are available for use by any object in the system.

Meaning is relative to context as well. The system handles this concept by means of ontological crosscutting. The Arrow system provides the crosscutting of ontologies trivially, since ontologies consist of fine-structure constructs, which inherently observe no arbitrary laws with respect to composition. It follows that structures built from these constructs are available at a fine-structure level. Therefore, crosscutting is trivially impossible, since it implies an identification of information between ontologies that violates the atomicity of the structures involved. Even infinitary or recursive structures are similarly available for identifications, allowing crosscutting to occur for domains whose internal structure is quite complex. Using this ability, the user can model a domain in any way he or she chooses and retain the ability to translate to any other model of any domain, including those of the same domain.

Some definable systems have no inherent capability for a truth concept; such properties of those systems are as local as the meaning of the terms of those systems.

Agents may compare the ontologies determined by frames. A simple metaphor for the inspection by the system of an ontology considers the state of a persons mind as it peruses a dictionary entry. As an analogy, the ontological frame would describe the collection of subjects in which the person is currently interested and the persons current state of knowledge concerning them, both of which are dynamic in nature. The references made by arrows in the graphs of the ontologies would mirror the words used in the definition statement for an entry. In addition, one knows from general-purpose dictionaries for common use that often multiple entries exist for the various uses of a word. The Arrow system mirrors this concept in a far cleaner manner, since such rules of term usage apply based on the context used by the speaker, with term meaning distributed among ontologies. In the Arrow system, agents make the choice of context constructively (whether explicit or implicit), and the terms of the ontologies that a context supports are available for use implicitly. In this way, our definitions for terms are those necessary for logical discussion of an argument, and gather into reusable units automatically.

The only problem foreseeable in this metaphor is for those dictionary aspects that contain contradictory concepts, which in an ordinary language would be troublesome when agents combine the ontologies. However, this is really the old monster of name conflicts, which really only relates to extension-derived identity found in traditional systems. The Arrow system overcomes this by positing the existence of all resulting information spaces that result from the various types of combinations of the clashing ontologies.

Now the Arrow system proposal relates to a dynamic, growing, general-purpose dictionary of concepts. The system naturally distributes a definition of an arrow in our dictionary, even across frames of knowledge. The definition extends to a far finer structure of information than does the traditional language, in that this dictionary via arrows describes issues that constitute language and linguistic context definition as well. A frame of knowledge of a domain in this light relates to a collection of ontologies, where the various aspects (as ontologies) of the dictionary specify the definitions of the subject matter. The aspects relate to form a whole structure in which agents may reason about the relations between these definitions and analyze the composite definitions formed. Ontological crosscutting is easily provided, since simple identification of terms between ontologies is sub-structural, that is, that one need not be limited to mappings between atoms, but instead between arbitrary structures (perhaps infinitary or self-similar) of atoms.

In order to present a specification to the user, the system collects the constraints specified for an arrow within the desired frame of ontologies identified with the vocabulary specified by the user. The natural system development spreads constraints across borders between explicit and implicit worlds. The system sees all constraints as equal, yielding reflection on context trivially. The interface filters out the implicit parts of the definition and produces a structure of specification that extends to the base vocabulary that the user wants.

However, those parts of the definition, whether explicit or implicit, are available for the user to view and modify, since the system allows the user to migrate around ontological frames freely. If calculable ontologies define the transitions between frames, then all the information with which the user was originally in contact is similarly accessible within the destination frame.

4.9 Dynamics

The Arrow system as described above is static (or kinematic at best). It remains to define the dynamics of the Arrow system. The Arrow system is a general-purpose information system that expresses contexts, ontologies, and language models as first-order concepts. It may range over many logics as context modification systems. These properties enable a novel means of expressing system dynamics.

As an information system, agents interpret all atoms within its domains primarily as atoms of information. The means for introducing information to the system include automated deduction and undecidable selection. The former method consists of all information posited by the system due to invoking reasoning services that are available to the system. Technically, such actions do not introduce information to the Arrow system, since the arguments, as constructions, are purely epistemic. The latter consists of all actions that reduce to a kernel of information provided solely by a human user. For instance, we may consider for practical purposes that all automated deduction systems are tactical problem-solvers, in that they decidably develop a given domain in search of results applicable to a target domain. However, these deduction systems must always be applied by an algorithmic approach that is often less semantically clean in its structure. This strategy a user often provides who has in mind some goal for the results of the inference engine that the system is often unable to encapsulate. This is often either due to restriction of domain or to absence of the applicable information that characterizes the users reasoning. In current programming systems, the concept of a command (or message-selection for objects or application for functions) usually represents the kind of information that only the user can decide. Some kinds of advanced reflective systems produce code themselves at a fine-grain level and in doing so transfer the user choice type of information to the meta-level. No current systems, however, integrate the information involved into a single space for reasoning.

Of course, any programmer can tell you that, within a given programming system, the means to improving system functionality is to feed the system more information that determines the ability of the system to manage other information that it receives. Obviously for the programmer, user choice embodies the program concept as well as the data that it manages. Within systems that cannot reflect, this information remains at the first-order level unless a programmer explicitly and manually gives the system the meta-information required to handle the case and again manually removes the redundant information. This definition constitutes a solid cybernetic definition of the usual notion of programmer as distinguished from user. This discussion considers the natural evolution of information systems toward utility to necessarily include the migration of user choice information to system deduction capabilities, since such a tendency naturally obviates redundant human involvement in the management of the information system. What distinguishes those aforementioned systems from the arrow proposal is that the Arrow system should optimize the transition of information from user choice to system deduction. From the knowledge-level perspective, the system transports information from the space of tasks to that of models and methods. At the model level, the view is somewhat different, since the domains of models and methods blur due to extensive reflection. The overall result is to blur the distinction between user and programmer, to allow for a more equitable society concerning computations in general.

Taking the graph over ontologies offered as an example for a basis, the Arrow system may develop its role in managing information from the paths of decidable translation outlined among those contexts available for modeling. A logic that manages the arrows and nodes in the graph system of ontologies and contexts can therefore manage the overall process of transporting this information in a reliable way. In addition, the logics employed as well as the process for translating this information is subject to modeling, allowing the system direct control over its operation via reflection.

Given this basis, the Arrow system proves an ideal platform for the automation of software production in arbitrary ways, since it can model information transitions that include hardware-related issues. For example, various types of formal specifications easily describe the format and content of processor instructions and data formats. The Arrow system casts this type of information as an ontology. In addition, agents may reliably specify protocols for interacting with various hardware devices and external software programs, with possibly less precision, in the same way. In modern systems, such information is specialized so that access to that information is itemized. By providing a specification for an interface to an information device to the Arrow system, the user may then manage that device in arbitrary ways, since the description is available for sequential composition into new ontologies and structuring into frames.

As for the semantic character of these information transitions, the system will provide quite a different paradigm than do traditional systems for its modification. As discussed, the primary purpose of the system is to translate information between contexts in order to promote overall system utility. To maximize its capabilities to do so, it provides maximum visibility for all information in the system relative to the context of the agent requesting the information. Simply removing any scope restriction stipulations for the systems arrows provides maximum visibility: from a context-neutral perspective, any arrow may reference any other arrow without compromising system integrity. However, the nature of the target contexts logical model directly affects the kind of information that the context may support.

For example, a context of a purely functional paradigm must reify all concepts as functions that agree ontologically with current functional atoms in that context. Therefore, in order to realize the interpretation of a certain concept in a functional domain, the target context must understand the types of information that the necessary function manages. This could require that agents transmit more information from the source domain to assist in matching ontologies between the domains. <<>>

The progression of user choice to system deduction strictly involves context shifts. In order to introduce new information to the system, the system must be capable of instantiating new arrows from specifications within the system. Assume, for instance, that the user makes a visual choice using a pointing device that corresponds to a standard graphical user interface (GUI). Within the Arrow system, the essential information update is the notification of the update of the mouses hardware state-machine, which exists in time, space, and state-space. This information is immediately available for translation into the user-interface ontology (say, as a single-click with the mouse button at such and such coordinates and about this time). Traditional GUI programming models then translate the meaning of this event into an abstract piece of information within a context that is abstract with respect to the underlying hardware. At that point (at the application level of abstraction), the information atom is decidedly within the arrow space and available for reasoning structures; it still embodies user choice, but is available in the context of logical operations for modeling. Therefore, if an agent posits a pattern about the users inputs, then other agents may factor out the appropriate information from the user-input data, such as statistically insignificant discrepancies between the model invocation and the measured input parameters. The remaining information is strictly redundant, and as such is subject to optimization heuristics by the system software; it is garbage for collection in traditional systems. The difference between the two types of systems is that the traditional type ignores information redundancy as an issue altogether and instead focuses on simplicity of the implementation from a restricted perspective.

The total systems semantics are of a functional paradigm, but at an information level, not simply at an execution level. In this way, information transformations are the fundamental abstraction type at the user level for the system. The constraint for such actions in this sense is to conserve information as bounded by user choice: the user may specify a heuristic that discards certain information types as noise. Typical examples of noise in current software systems include the timings and details of mouse input sequences of the previous example. Therefore, these information transformations map an existing arrow information state to a new one.

Side effects in the system are impossible in this information model. Any side effects resulting from such transformations could only generate information that is by definition unusable by the system; such information is implicitly discarded, and only the user can help the system recover from this loss. Of course, this sort of information loss has always existed in current systems, in terms of undecidable pattern recognition. The Arrow system improves on the current plight by offering a better base of information system from which to construct the pattern identifications.

However, the effects of the function for total system semantics would not necessarily be finite at a first-order level

The system can reference itself as a whole using self-similarity factorization of infinitary structures that result.

Specification of programs: data-flow and control-flow graphs have axiomatic definitions, with parts being static or dynamic with respect to contexts

Specification of verification: inference-flow graphs

Show that the method of integration of these graphs, traditionally static in traditional languages, is now dynamic and first-order

Context construction, thereby, and its character

Hence, model-level reflection is trivial, since this achieves a highly utilitarian method of dynamic agency

4.10 User Interface

To consider the application of the Arrow system to the issue of user-interface, one must consider the nature of the state of information of the system. The models within the system should transcend the usual context of a snapshot in time for the machine, so that it encapsulates the time predicates that the user wishes to enforce. In this way, the user's desires directly affect the usual notion of implementation as it relates to interactivity of the software system and efficiency.

The system models the ontological frame that the user employs, and interacts with the user based on that group of ontologies. In model-level terms, the system speaks directly to the ontology, and merely passes the conversation along to the user for review. The user, in turn, acts for the ontology in communicating with the system. The user may manipulate the model of the ontology directly, or the system may make guesses about shifts in the users knowledge frame based on input. It could produce an information feedback loop in the attempt to maintain effective communication with the user of the system. These guesses may generate several sub-frames inconsistent with each other, but the system can manage the interactions between these frames reliably and may even discard a sub-world based on interactions with the user.

Note that the information atoms presented to the user by the Arrow system need not be arrows within the system directly. The system should regard the user as (capable of) recognizing arbitrary kinds of ontologies, and, since it is often the more capable nonmonotonic reasoner, is the one for which the system should arrange its ontologies. In order to facilitate this process, the systems agents must understand this situation and its rationale.

Information density of various ontologies for user-interface, particularly for user-input

Text identifiers proper names, vocabulary, terminology, user dictionaries, etc

GUIs vs. CLIs windowing arrangement information (dialog boxes vs. command-line parameters) (views vs. pipes) (user choice vs. derived information and the transition process for the system) (how this improves system utility)

4.11 Garbage Collection

A naturally difficult notion for such a system is its representation on a finite-state device, such as the modern computer. The solution seems to be to maintain the required amount of kernel information to a finite, hopefully compact, size. This kernel would consist of the information required to build all other parts of the system, particularly the infinitary structures. The system software must manage a reference to an object with an infinitary internal structure in a different way.

The system reduces the representation of a domain to the axioms of decidable parts, including the necessary annotations for describing the rest of the domain.

Input from the user that has not been rationalized by the system must be part of this kernel, since the system deals with it within its own separate knowledge-frame that possesses no calculable ontologies until given them by the system.

The Arrow system directly supports a partial-evaluation model. The appropriate context considers states of evaluation as nodes within a graph. Arrows between the states representing function invocations are available for first-order study. The patterns that emerge in the graph, if easily re-constructible, justify garbage-collection. If the ontology for steps of evaluation is inappropriate, then an agent may apply a different ontology to the domain of the graph to provide a different level of structure.

The garbage collectors axiom is to conserve system information and discard all noise. It therefore operates by definition over the space of redundantly stored information as well as the undesired information (i.e. that labeled as noise). The system discards undesirable information based on a heuristic that addresses performance as regards speed and persistent storage requirements. The system software retains redundant information, however, based on performance criteria, since necessary calculations (for, say, data format conversion) are often expensive in resources to reproduce. If a non-calculable process develops the redundancy, then the system must retain resulting information unless explicitly countermanded by the user.

Potentially constructed information is substitutable for the actual information derived, just as the results of function applications are often denoted by their persistent invocation points in modern languages, vice identifiers bound to the results storage location. This referential shift to intension of function application, vice its extension (the stored result), allows the garbage collector to operate based on explicit references alone to data. A possible implementation of this concept within arrows is to explicitly encode the finite parts of both the intension and the extension of a structure, allowing the garbage collector to collect redundant extensional data in a lazy way. The garbage collector should tend to preserve the intension (the generating function) of the structure over extension, since the former tends to human-level intelligence to derive, whereas the latter is usually calculable.

4.12 Constructive Implementation

The plan for the implementation of the Arrow system involves the model of reflection often introduced in the usual texts involving reflection. By subsuming the functionality of the virtual machines upon which the implementators base the prototype system, the Arrow system environment may extend to a level that allows a reconstruction of the virtual machine implementation. However, this method simply allows the implementors to re-model the virtual machine, at a heavy cost in human effort. The criteria for the completion of the system include that the semantic capabilities should encompass the description of a minimum amount of functionality of the underlying hardware, as well as some operational algebras that enable code generation. This suggests a model where an initial virtual machine provides a simple basis for development of a knowledge base for the Arrow system, ensuring to include an adequate formal model of the virtual machine. The remaining task would be to completely re-implement the virtual machines semantics of interface from within the system. To that end, the virtual machine should provide a direct means to access the underlying hardware as well as a simple model of that hardware state-machine. Once the Arrow system completely re-implements the virtual machine successfully, so that the system can bootstrap itself, then it will have accomplished elementary reflection. The final task would then be to place the implementation code within an environment with an adequate number of modelings of the specification, to place the specification in reach of all of the systems main ontologies, thereby enabling model-level reflection.

Text identifiers constitute information that is only interpretable by the user. As such, the information belongs to the domains associated with user interface within the appropriate part of the system. This information subsumes what usually comprises the documentation for a programming system, traditionally maintained solely by the user. In the Arrow system, a more enlightened view exists, placing this domain under the systems model-level reflective capabilities. Formal theories of the understanding of natural human language may assist in translating ontologies into human-readable documentation, and this process itself is of course available for improvement under the system design. The naturally useful goal is of course to render all documentation as a dynamically calculable interface with the ontology frame that the user prefers. Of course, in any Arrow system wherein information is being added, the system receives all new information as non-calculable, and then proceeds to define it in terms of existing information as specified by meta-information introduced to it. It follows that the system will not be able to completely describe the meaning of its information.

The information interface between the system and the user is not to be trusted implicitly, since human mental processes are often incalculable for modern machines. In fact, often it is more useful to forgo system understanding in order to facilitate communication between people, given that the lifetime of the information is temporary.

5 Conclusions

While the claims made for the Arrow systems potential seem grandiose, the basis for systems information model lies in modern research systems as well as in the examples of computing systems of the past. The central concept of the system design is to increase the capabilities of the system to analyze and act to improve its performance and utility by generalizing the systems modeling capabilities in a meaningful way. The anticipated result is the greater overall freedom in the use and re-use of information for arbitrary purposes, and to minimize the negative impacts on society associated with maintaining a large body of knowledge. In this way, the design intends to fully deliver the promise that computing devices exhibit, in spite of the restrictions in modern systems due to hardware and information protocol heterogeny. The full implementation of the Arrow system should live up to its promises.

References

[Barenregt, 1997] Henk Barendregt. The Impact of the Lambda Calculus in Logic and Computer Science. In The Bulletin of Symbolic Logic Vol. 3, No. 2, June 1997.

[Brandt & Schimidt, 1995] Søren Brandt and René W. Schmidt. The Design of a Meta-level Architecture for the BETA Language. Technical Report, Department of Computer Science, University of Aarhus, 1995.

[Erwig, 1997] Martin Erwig. Abstract Visual Syntax. In Workshop on the Theory of Visual Languages, 1997.

[Gruber, 1993] Thomas R. Gruber (Knowledge Systems Laboratory). A Translatable Approach to Portable Ontology Specifications (Technical Report KSL 92-71).

Stanford University, 1993. C. Barry Jay. An Introduction to Categories in Computing. University of Technology, Sydney. School of Computing Sciences, 1993.

[McCarthy, 1997] John McCarthy and Sasa Buvac. Formalizing Context (Expanded Notes). Stanford University, 1997.

[Pierce, 1995] Benjamin C. Pierce. Foundational Calculi for Programming Languages. In the CRC Handbook of Computer Science and Engineering, 1995.

[Sieg, 1997] Wilfried Sieg. STEP BY RECURSIVE STEP: CHURCH'S ANALYSIS OF EFFECTIVE CALCULABILITY. In The Bulletin of Symbolic Logic Vol. 3, No. 2, June 1997.

[van Benthem, 1996] Johan van Benthem. Exploring Logical Dynamics. CSLI Publications, 1996.

[Wiener, 1957] Norbert Wiener. CYBERNETICS AND SOCIETY. Houghton Mifflin, 1957.