Heterogeneous Computing, Agent-model based Software Design, and Validating Multiagent AI Agent Processes

CYRUS. F. NOURANI[1,2]

[1]ARDAFW AI Labs
USA

[2]TU Berlin University of Technology
GERMANY

Abstract: AI agents and Mediators define the stages of conceptualization, design and implementation. Objects, message passing actions, and implementing agents are defined by syntactic constructs, with agents appearing as functions. By defining specified agent activators events and activity are computed for the models. There are two specific directions on modelling and with applications AI and the second direction is based on models and transformations. Past few years have brought forth an excellent composition on what might be the practical culmination on modelling, transformations and newer software engineering techniques, for example with SDF, or model refactoring with AGG. Newer techniques and applications to multiagent mission planning and AI system validation systems are presented while bridging the AI-Software design engineering with new multiagent computing techniques. Design validation is based on morphisms on multiagent structures with agent computing processes for object lever programs. The process is applicable to the existing AI systems with structural lifts with objects and ontology preserving functions. Future directions with double-vision orchestration, collaborative AI and spcecrafr mission planning examples, are preseted in brief.

Keywords: Agetic AI, Mission Planning, Agent/ Module Ontology Preservation Principle, Heterogeneous Data and Knowledge Engineering, Validating AI Systems, AIVV, Generative Transformation, Model-based Software Engineering Agents, Model Refactoring.

Received: June 24, 2025. Revised: October 15, 2025. Accepted: November 12, 2025. Published: February 10, 2026.

1 Introduction

A software design paradigm incorporating novel implementation techniques and a definition of abstract intelligent implementations (AII) are presented. Innovative techniques for design of knowledge bases and heterogeneous software systems with techniques for automated reasoning are put forth. The application areas include support for highly responsive planning. Intelligent implementation of software, i.e., the design and implementation by AII techniques is due to be an area of crucial importance. The AII techniques are being applied [8] gradually to the real problems encountered in fields such as intelligent systems, aerospace, robot design, and knowledge bases, abbreviated by KB A basis for what we have called Artificial Algebras [2] has been written since the first version of All was written in 1993. It is a preliminary theory for an algebra for intelligent trees and artificial intelligence. The mathematical foundations for software agents might call for algebras with varying carriers and functionality. AI systems might be defined by the stages of Conceptualization, Design, and Implementation.

Each of the stages is to be approached in ways that minimize human error and enable the designed system to automatically recover from faults. The fault recovery issues are not the topic of this paper and are treated by this author in [12]. We design software with agents[10] via a methodology which commences with a knowledge acquisition phase, followed by a specification phase, and concluded by a system implementation phase. The present approach defines functional nondeterministic knowledge learning (Design Agents), fault free system specification, and multiagent abstract implementations. Design Agents is Flagrant Agent Computing by active agent learning, and includes exception knowledge as an essential component, as does system specification. The techniques are defined for designing heterogeneous software. System implementation is by independent concurrent computing agents. A pair of systems, each consisting, defines AI and software systems in the present paper of many computing agents. The two parts are mutually synchronized to enable fault and exception handling and recovery in an automatic manner [1,18].

Software agents are specific agents designed by a language that carry out specified tasks and define a software functionality. Most agents defined by our examples are software agents. In the space examples there, of course, implied hardware functionality specified. Objects are in the well-known sense of the word in object programming, abbreviated by OOP. However, our designs are with *intelligent objects* a concept we had invented since 1992. Its foundations has been developed and applied this authors publications. Ordinary objects consist of abstract data, perhaps encapsulation, and operations. Most recent programming techniques apply OOP in some form. Software engineering techniques with abstract data types have had OOP on their mind. IOOP is is based on technique developed by the author combining AI and software agents with OOP. For our project the modular programming concepts are combined with software agent computing, new IOOP constructs object-cobject pairs and kernels. Modules are aggregate objects with a specific functionality defined. Aggregate objects and their specified functions are defined by <module-co module> pairs called *kernels*. .

A kernel consists of the minimal set of processes and objects that can be used as a basis for defining a computing activity. The term kernel is analogous to the terminology familiar in operating systems concepts, but is at a high level of abstraction with objects and functions encapsulated. A system is defined by a set of kernels, which can be programmed to function synchronous applying software agents. The analogy is a distributed computing environment with many computers on a net. Each kernel corresponds to a computer on the net. The multiagent AI concepts are the standard terms[7]. For the Intelligent Systems, the nomenclature, e.g. Facilitator, Mediator is

from standardization defined and agreed on at a conference in Colorado for the purpose in Heterogeneous design with software agents dates back to the [17] papers and addressed on a formal funding proposals the same years.The applied terminology is defined in our paper. The same conventions define *heterogeneity* to be the mismatch found in autonomously developed resources and services, ranges from platforms, operating systems, database systems and models, data representations, ontology, semantics, and processing paradigms. Level is a conceptual categorization, where objects at a lower level depend on their ancestors at a higher level. Ancestor is an object at a higher level, source of inheritable attributes. The root object is the ultimate ancestor. Ontology is a set of terms and relationships used in a domain, denoting concepts and objects, often ambiguous among domains. The techniques in [1,18] have been applied to the design of knowledge-based systems by the present project and [13].

The basis for a sound theoretical and practical methodology for designing AI software systems is emerging. The paper's structure is as follows. Section one defines the way multiagent systems might be specified by software agents. Additional new concepts applied are <object-co objects> and intelligent objects. There are illustrating examples. Section 2 combines the designs with abstract mediators and applies the current Intelligent Systems terms to define formal agent-based designs. Section three defines event-prompted agents. Section 4 defines multiagent systems designed with the techniques, instantiating facilitators and mediators by an example. Section 5 defines formal algebras for multiagent systems and defines formal implementation maps for the algebras. It further defines ontology algebras incorporating the Ontology Preservation Principle. Section 6 is an overview to the AIS synthesizer for multiagent software design. The paper is concluded by section 7. *The techniques are further applied* to *model-based software engineering and model refactoring*. The paper outlines methods for validating multiagent AI systems by combining software engineering with agent-based computing. It emphasizes fault-tolerant, ontology-preserving, and modular design principles for mission-critical applications, including space missions. The key concepts are as follows:

Agents and Mediators: Agents are functional units (software or implied hardware) carrying out tasks, while mediators synchronize actions, handle exceptions, and enable recovery. Design_Agents & CoAgents: Design_Agents = normal activity modules. CoAgents = fault-handling modules. These run concurrently, ensuring fail-safe operation.Ontology Preservation: A system is valid only if its ontology (domain concepts, relations, rules) is preserved during implementation and refactoring. This is called the AII Ontology preservationPrinciple (AIIOPP). Model-based Engineering & Refactoring: Uses algebraic models and graph transformations (e.g.,TU Berlin's AGG) to refactor and validate agent models while ensuring structural consistency. Validation Techniques (AIVV): Object-level validation: actions and their duals (normal vs. fault response).

1.1. Specifying Multiagent Systems

The hypotheses for the realization of systems in our project might appear "linear" steps of software engineering, however its linearity is no more stringent than the concept of modular design. It is the least we can demand from a design. In reality the design concept is highly nonlinear. The agents can be applied in ways which, compared to an ordinary software engineering design, appear highly nonfunctional and non-modular. From the software agent designer's viewpoint, however, there is molecularity with artificial structures. Artificial structures [2] are implemented by agent morphisms. The process is thus includes loops amongst the phases in the software life cycle. The intelligent objects and modules, agents, facilitator and mediators leave many degrees of freedom to design. There are artificial loops in the design resembling aerobatics by high-speed airborne agents. The AI and software designer specifies the actions

and operations from the informal specifications supplied by a "user." . Some past contributors are [8], [13]. The initial phase of the design of the proposed AII techniques is to present the design with Mediator [10,17,19], Abstract Specifications, where specifications are in the sense of [1,4]. Ontology algebras are defined at meta-data and meta-knowledge level. Intelligent tree computing theories [22] and artificial algebras [2] can be applied to the theoretical development. Knowledge acquisition requires either interviewing an expert, brainstorming with a group of experts, or structuring one's thoughts if the specifier is the expert. For multiagent designs there are active learning agents and automatic learning. We present the notion of Functional Nondeterministic Knowledge Learning (Design_Agents) in [11].

Design_Agents is formulated to deal with the conceptualization stage and is being applied by the present project to define active learning by agents. Design_Agents requires the user to inform the specifier as to the domains that are to be expected, i.e. what objects there are and what the intended actions (operations) on the objects are, while fully defining such actions and operations. The actions could be in form of processes in a system. The relations amongst the objects and the operations (actions) can be expressed by algebras and clauses, which the specifier has to present. The usual view of a multi-agent systems might convey to an innocent AI designer that an agent has a local view of the environment, interacts with others and has generally partial beliefs (perhaps erroneous) about other agents.

On the surface the Design_Agents specification techniques might appear as being rigid as to what the agents expect form other agents. The Design_Agents specification does not ask the agents be specified up to their learning and interaction potential. Design_Agents only defines what objects might be involved and what might start off an agent. It might further define what agents are functioning together. Thus specifications are triples <O,A,R> consisting of objects, actions and relations. Actions are operations or processes. The views of abstraction [1,4], object-level programming, and agent views of AI computation [12], are the important components of inter-play in the present paper. Design_Agents have some additional requirements to be put forth. The requirement is that each object to be defined has to have a dual definition in terms of the actions to be taken for flagrant agents, faults, exception and recovery.

At the knowledge learning phase the expert is to state all exceptions to actions and what recovery and corrective actions are to be carried out. For each action on an object a dual action is to be supplied through Desig_Agents, such that a specifier can fully define the effect of the dual actions. The design techniques do not imply asking the expert to state all the exceptions to actions. The exceptions naturally present themselves by the object-coobject concept. A coobject is an object defined with the same carriers as the object, but with a set of operations complementary to the object's operations carrying on an alternate symmetric Exception operations. In the figure let OPS denote operations, EXP denote exceptions. The last equations define the exception action. In the example there is a process (action) that is always checking the supply of Angelika coffee implementing the exception function. APs:= <A trivial example>, many robots appear at a critical entrance at once, necessitating FA activity. APs are computing events which activates an agent (see section 3). 1.1 Multigent Planning

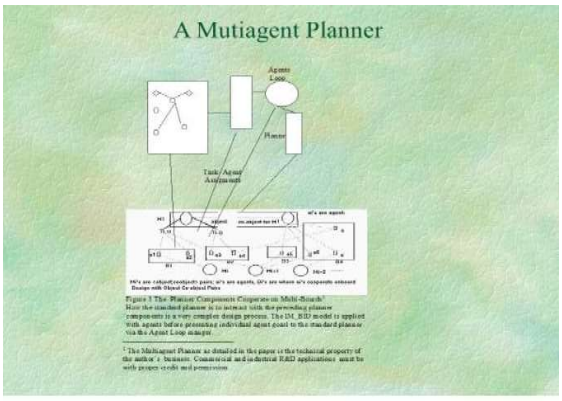

Figure 1

Components Cooperate on Multi-Boards How the standard planner is to interact with the preceding planner components In the figures above Mi's are objects; ai's are agents. The dotted lines are agent message paths. A specific super module appears as an <object-coobject> pair.. The following is a Python processing depiction for the above planner. Think of the simulation like a mini world where agents act, communicate, and replan.

A depicted Python design follows: Roles Planner (Global or distributed) Decides who does what and when Agents (A1, A2, A3...) Execute tasks, observe environment, report status Environment Changes over time (resources, obstacles, goals). Visual Metaphor Agents = moving nodes or characters Tasks = icons/cards Planner = control dashboard or "brain"

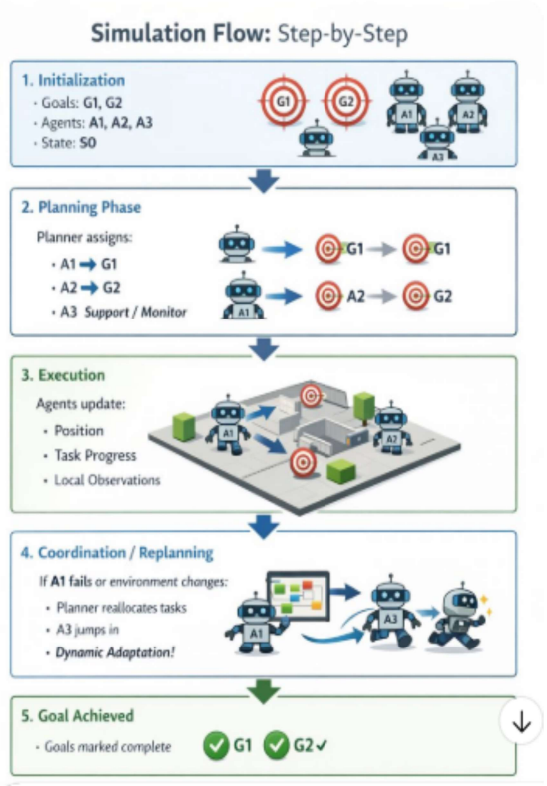

Figure 2

Color-code: Agent executing Agent waiting ●Agent replanning

1.2 Visual mission planning

An example IM mission planning is as follows. Hybrid picture 1- Spacecraft "Spacecraft" \i A Navigation Window "Navigation Window" \i

Agents: A1 Computes available docking times based on the visual field on the window.

A2 carrysout docking sequence based on messages to Spacecraft B

Hybrid picture 2 Spacecraft B Navigation Window

Agents: B1 ccarries on course based on its visual field window

B2 Accepts and carries out docking maneuvers from external hovering craft agents Plan Goal "Plan Goal"

Engage docking between A and B at appropriate A and B field windows.

1.3 Morph Gentzen

Morph Gentzen computing can be applied to the hybrid pictures to satisfy a plan goal. Thus morphing is applied with precise fluidity o plan computation. A well-known agent computing design paradigm is BID. At BID the functional or logical relations between motivational attitudes and between motivational attitudes and informational

attitudes are expressed as meta-knowledge, which may be used to perform meta-reasoning resulting in further conclusions about motivational attitudes. If we were to plan with BID with intelligent multimedia the logical relations might have to be amongst worlds forming the attitudes and event combinations. For example, in a simple instantiation of the BID model, beliefs can be inferred from meta-knowledge that any observed fact is a believed fact and that any fact communicated by a trustworthy agent is a believed fact. With IM_BID, the observed facts are believed facts only when a conjunction of certain worlds views and evens are in effect and physically logically visible to the windows in effect. Since planning with IM_BID is at times with the window visible agent groups, communicating, as two androids might, with facial gestures, for example.

2. The Formal Basis

The present approaches have a theoretical basis abbreviated in the following sections. We start with agents, define modules and algebras, and agent and module morphisms.

2.1 Agents

Starting with what are called hysterectic agents (Genesereth&Nilsson 1987). A hysterectic agent has an internal state set I, which the agent can distinguish its membership. The agent can transit from each internal state to another in a single step. Actions by agents are based on I and board observations. There is an external state set S, modulated to a set T of distinguishable subsets from the observation viewpoint. An agent cannot distinguish states in the same partition defined by a congruence relation. A sensory function s :S → T maps each state to the partition it belongs. Let A be a set of actions which can be performed by agents. A function action can be defined to characterize an agent activity action:T →A. There is also a memory update function mem: I x T → I. To define agent at arbitrary level of activity knowledge level agents are defined. All excess level detail is eliminated. In this abstraction an agent's internal state consists entirely of a database of sentences and the agent's actions are viewed as inferences based on its database. The action function for a knowledge level agent maps a database and a state partition t into the action to be performed by an agent in a state with database and observed state partition t. action: D x T→ A The update function database maps a state and a state partition t into a new internal database. database: D x T→ D A knowledge-level agent is an environment is an 8-tuple shown below. The set D in the tuple is an arbitrary set of predicate calculus databases, S is a set of external states, T is the set of partitions of S, A is a set of actions, see is a function from S into T, do is a function from A S into S, database is a function from D x T into D, and action is a function from D x T into A. <D,S,T,A,see,do,database,action> Knowledge level agents are hysterectic agents.

2.3 Agent Morphisms and Module Preservation

Starting with what we called hysterectic agents (Genesereth&Nilsson 1987). A hysterectic agent has an internal state set I, which the agent can distinguish its membership. The agent can transit from each internal state to another in a single step.

Actions by hysterectic agents are based on I and observations. The observations are from problem solving boards, messages to the agent, and a database. There is an external state set S, modulated to a set T of distinguishable subsets from the observation view point. An agent cannot distinguish states in the same partition defined by a problem congruence relation. A sensory function s :S → T maps each state to the partition it belongs. Let A be a set of actions which can be performed by agents. A function action can be defined to characterize an agent activity action:T → A. There is also a memory update function. A *hysterectic* agent HA defined by a sextuple <I,S,T,A,s,d,internal,action> where d is a function form A x S→ S and internal I x T

→ I. Let HA be a set of sextuples defining a hysterectic agents. Define HA morphism by a family of functions defined component-wise on the sextuple above.

Definition 2.1 A HA morphism is a function F: HA→ HA' defined component-wise by

F[i]: I→ I'; F[S]: S → S', F[T]: T →T', F[A]: A →A'; F[s]: S→ T'; F[d]: A' x S' → S' and F[internal]: I' x T'→I'.

Definition 2.1 implies F defines a new hysterectic agents from HA by a morphism.

The definition might become further transparent in view of definitions is section 2.4. Component-wise definitions for a morphism might be viewed as functions on a multi- sorted signature carrying the sextuple. Similar morphisms can be defined for knowledge level agents defined in section 2.1 which we can refer to by KL-morphisms.

2.4 Agents, Modules, and Algebras

The computing enterprise requires more general techniques of model construction and extension, since it has to accommodate dynamically changing world descriptions and theories. The models to be defined are for complex computing phenomena, for which we define generalized diagrams. The authors a decade ago, e.g. [3], techniques for model building as applied to the problem of AI reasoning allows us to build and extend models through diagrams. It required us to define the notion of generalized diagram.or generic model diagrams were devied by this author, e.g, [3] to build models with prespecified functionns, for exam,ple baed on Skolem functions. The specific minimal set of function symbols is the set with which a model for a knowledge base can be defined. The G-diagram techniques allowed us to formulate AI worlds, KB's in a minimal computable manner to be applied to agent computation.

The model buildin tehnoques are applied to the problem of AI reasoning allows us to build and extend models through diagrams. A technical example of algebraic models defined from syntax had appeared in defining initial algebras for equational theories of data types (ADJ 1973) and our research in (Nilsson 1969). In such direction for computing models of equational theories of computing problems are presented by a pair (,E), where is a signature (of many sorts, for a sort set S) (ADJ 1973,Nourani 1995a) and E a set of -equations. Signatures are in the same sense as key signatures in music.

Definition 2.2 An s-sorted signature or operator domain is a family <w,s> of sets, for s S and w S* (where S* is the set of all finite strings from S, including the empty string). call f <w,s> and operation symbol of rank w,s; of arity w, and of sort s. We apply multi-sorted algebras via definition 2.3 to multiagent systems.

Definition 2.3 Let be an S-sorted signatures. A -algebra A consists of a set As for each s in S (called the carrier if A of sorts) and a function <A>: As1 x As2 x....xAsn As for each <w,s>, with w=s1s2...sn (called the operation named by). For <,s>, A As, i,e the (set of names) of constants of sort s.

Definition 2.4 If A and B are algebras, a -homomorphism h: A B is a family of functions <hs: As Bs> s in S that preserve the operations, i.e. that satisfy (h0) For <s>, the hs(A) = B; (h) If, For <w,s>, with w=s1s2...sn and <al,...,an> As1 x As2 x....xAsn, then hs[A(a1,...,an)] = B(hs(a1),...,hs(an)).

2. The Formal Basis

The present approaches have a theoretical basis abbreviated in the following sections. We start

with agents, define modules and algebras, and agent and module morphisms.

2.5 Agents

Starting with what are called hysterectic agents Genesereth&Nilsson [7]. A hysterectic agent has n internal state set I, which the agent can distinguish its membership. The agent can transit from each internal state to another in a single step. Actions by agents are based on I and board observations. There is an external state set S, modulated to a set T of distinguishable subsets from the observation viewpoint. An agent cannot distinguish states in the same partition defined by a congruence relation. A sensory function s :S →T maps each state to the partition it belongs. Let A be a set of actions which can be performed by agents. A function action can be defined to characterize an agent activity action:T →A. There is also a memory update function mem: I x T →I. To define agent at arbitrary level of activity knowledge level agents are defined. All excess level detail is eliminated. In this abstraction an agent's internal state consists entirely of a database of sentences and the agent's actions are viewed as inferences based on its database. The action function for a knowledge level agent maps a database and a state partition t into the action to be performed by an agent in a state with database and observed state partition t. action: D x T→A

The update function database maps a state and a state partition t into a new internal database.

database: D x T → D

A knowledge-level agent is an environment is an 8-tuple shown below. The set D in the tuple is an arbitrary set of predicate calculus databases, S is a set of external states, T is the set of partitions of S, A is a set of actions, see is a function from S into T, do is a function from A S into S, database is a function from D x T into D, and action is a function from D x T into A. <D,S,T,A,see,do,database,action> Knowledge level agents are hysterectic agents.

2.6 Agent Morphis Module Preservation

Starting with what we called hysterectic agents (Genesereth&Nilsson 1987). A hysterectic agent has an internal state set I, which the agent can distinguish its membership. The agent can transit from each internal state to another in a single step. Actions by hysterectic agents are based on I and observations. The observations are from problem solving boards messages to the agent, and a database. There is an external state set S, modulated to a set T of distinguishable subsets from the observation view point. An agent cannot distinguish states in the same partition defined by a problem congruence relation. A sensory function s :S ◊ T maps each state to the partition it belongs. Let A be a set of actions which can be performed by agents. A function action can be defined to characterize an agent activity action:T →A. There is also a memory update function. A hysterectic agent HA defined by a sextuple <I,S,T,A,s,d,internal,action> where d is a function form A x S →S and internal I x T → I. Let HA be a set of sextuples defining a hysterectic agents. Define HA morphims by a family of functions defined component-wise on the sextuple above.

Definition 2.1 A HA morphism is a function F : HA →HA' defined component-wise by F[i]: I→ I'; F[S]: S →S', F[T]: T →T', F[A]: A →A'; F[s]: S→T'; F[d]: A' x S' →S' and F[internal]: I' x T'→I'.

Definition 2.1 implies F defines a new hysterectic agents from HA by a morphism. The definition might become further transparent in view of definitions is section 2.4. Component-wise definitions for a morphism might be viewed as functions on a multi-sorted signature carrying the sextuple. Similar morphisms can be defined for knowledge level agents defined in section 2.1 which we can refer to by K-morphisms.

2.7 Agents, Modules, and Algebras

The computing enterprise requires more general techniques of model construction and extension, since it has to accommodate dynamically changing world descriptions and theories. The models to be defined are for

complex computing phenomena, for which we define generalized diagrams. The techniques in (Nourani 1983,87,91,94a) for model building as applied to the problem of AI reasoning allows us to build and extend models through diagrams. It required us to define the notion of generalized diagram. We had invented G-diagrams(Nourani 1987,91,93b,94a) to build models with prespecified generalized Skolem functions. The specific minimal set of function symbols is the set with which a model fro a knowledge base can be defined. The G-diagram techniques allowed us to formulate AI worlds, KB's in a minimal computable manner to be applied to agent computation. The techniques in (Nourani 1991,94a) for model building as applied to the problem of AI reasoning allows us to build and extend models through diagrams. A technical example of algebraic models defined from syntax had appeared in defining initial algebras for equational theories of data types ADJ [4]and this author in (Nilsson 1969). In such direction for computing models of equational theories of computing problems are presented by a pair (Σ,E), where is a signature (of many sorts, for a sort set S) and E a set of -equations. Signatures are in the same sense as key signatures in music.

Definition 2.2 An s-sorted signature or operator domain is a family <w,s> of sets, f Є or s S and w S*(where S* is the set of all finite strings from S , including the empty string). call f <w,s> and operation symbol of rank w,s; of arity w, and of sort s.

The figure depicts an S-sorted signature from ADJ[6]. We apply multi-sorted algebras via definition 2.3 to multiagent systems.

Definition 2.3 Let be an S-sorted signatures. A -algebra A consists of a set As for each s S (called thecarrier if A of sort s) and a function <A>: As1 x As2 x....xAsn As for each <w,s>, with w=s1s2...sn (called the operation named by).For <,s>, A As, i,e the (set of names) of constants of sort s.

Definition 2.4 If A and B are algebras, a -homomorphism h:A B is a family of functions <hs:As Bs> s in S that preserve the operations, i.e. that satisfy (h0) For <,s>, the hs(A) = B; (h1) If , For <w,s>, with w=s1s2...sn and <a1,...,an> As1 x As2 x....xAsn, then hs[A(a1,...,an)] = B(hs(a1),...,hs(an)).

For an intelligent signature I, let T<I> be the free tree word algebra of signature I. The quotient of T<I>, the word algebra of signature , with respect to the I-congruence relation generated by a set of equations E, will be denoted by T<I,E>, or T<P> for presentation P.

The computing and reasoning enterprise require more general techniques of model construction and extension, since it has to accommodate dynamically changing world descriptions and theories. The techniques in the author's projects for model building as applied to the problem of AI reasoning allows us to build and extend models through diagrams. A technical example of algebraic models defined from syntax had appeared in defining initial algebras (ADJ 1977) for equational theories of data types, this author since 1990`s at least. In such direction for computing models of equational theories of computing problems are presented by a pair (Σ,E), where Σis a signature (of many sorts, for a sort set S) and E a set of Σ-equations.

Let T<Σ> be the free tree word algebra of signature Σ. The quotient of T<Σ>, the word algebra of signature Σ, with respect to the Σ-congruence relation generated by E, will be denoted by T<Σ,E>, or T<P> for presentation P. T<P> is the "initial" model of the presentation P.

The Σ-congruence relation will be denoted by ≡P. One representation of T(P) which is nice in practice consists of an algebra of the canonical representations of the congruence classes, abbreviated by Σ-CTA. It is a special case of generalized standard models the author had defined (Nourani 1996 for newer examples). Some definitions are applied from our papers that allow us to define standard models of theories that are Σ-CTA's. The standard models are significant for tree computational theories

that the author had presented. Generic diagrams are applied to define canonical standard models in the same sense as set theory. This definitions are basic to sets and in defining induction for abstract recursion and inductive definitions. The canonical models are applied to multiagent computing during the last several years by the author.

Definition 2.5 We say that a signature Σis intelligent iff it has intelligent function symbols. We say that a language has intelligent syntax if the syntax is defined on an intelligent signature. To define a specific mathematical linguistics basis for agent augmented languages intelligent languages were defined (Nourani 1995d) as follows.

Definition 2.6 A language L is said to be intelligent language iff L is defined from an intelligent syntax.

Agent augmented languages and signatures allow us to present computational theories with formulas on terms with intelligent function symbols.

2.7.1. Abstract Intelligent Syntax

It is essential to the formulation of computations on intelligent trees and the notion of congruence that we define tree intelligence content. A reason is that there could be loss of tree intelligence content when tree rewriting because not all intelligent functions are required to be on mutual message exchanges. Theories are presented by axioms that define them and it is difficult to keep track of what equations not to apply when proving properties. What we have to define, however, is some computational formulation of intelligence content such that it applies to the present method of computability on trees. Once that formulation is presented, we could start decorating the trees with it and define computation on intelligent trees. It would be nice to view the problem from the stand point of an example.

The examples of agent augmented languages we could present have <O,A,R> triples as control structures. The A's have operations that also consist of agent message passing. The functions in AFS are the agent functions capable of message passing. The O refers to the set of objects and R the relations defining the effect of A's on objects. Amongst the functions in AFS only some interact by message passing. What is worse the functions could affect objects in ways that affect the intelligence content of a tree. There you are: the tree congruence definition thus is more complex for agent augmented languages than those of ordinary syntax trees. Let us define tree intelligence content for the present formulation. For an intelligent signature I, let T<I> be the free tree word algebra of signature I. The quotient of T<I>, the word algebra of signature, with respect to the I-congruence relation generated by a set of equations E, will be denoted by T<I,E>, or T<P> for presentation P.

2.8 Agents, Languages, and Models

By an intelligent language we intend a language with syntactic constructs that allow function symbols and corresponding objects, such that the function symbols are implemented by computing agents in the sense defined by this author in . Sentential logic is the standard formal language applied when defining basic models. The language is a set of sentence symbol closed by finite application of negation and conjunction to sentence symbols. Once quantifier logical symbols are added to the language, the language of first order logic can be defined. A Model for is a structure with a set A. There are structures defined for such that for each constant symbol in the language there corresponds a constant in A. For each function symbol in the language there is a function defined on A; and for each relation symbol in the language there is a relation defined on A. For the algebraic theories we are defining for intelligent tree computing in the forthcoming sections the language is defined from signatures as in the logical language is the language of many- sorted equational logic.

The signature defines the language by specifying the function symbols' arities. The model is a structure defined on a many-sorted algebra consisting of S-indexed sets for S a set of sorts. By an intelligent language we intend a language with syntactic constructs that allow function symbols and corresponding objects, such that the function symbols are implemented by computing agents. A set of function symbols in the language, referred to by AF, is the set modeled in the computing world by AI Agents with across and/or over board capability. Thus the language defined by the signature has designated function symbols called AF. The AF function symbols define signatures which have specific message paths defined for carrying context around an otherwise context free abstract syntax. A set of function symbols in the language, referred to by AF, are agents with nontrivial capability. The boards, message passing actions, and implementing agents are defined by syntactic constructs, with agents appearing as functions. The computation is expressed by an abstract language that is capable of specifying modules, agents, and their communications. We have put together the AI concepts with syntactic constructs that could run on the tree computing theories we are presenting in brief. We have to define how the syntactic trees involving functions from the AF are to be represented by algebraic tree rewriting on trees. This is the subject of the next section, where free intelligent trees are defined. An important technical point is that the for agents there are function names on trees.

Definition 2.5 We say that a signature is intelligent iff it has intelligent function symbols. We say that a language has intelligent syntax if the syntax is defined on an intelligent signature.

Definition 2.6 A language L is said to be an intelligent language iff L is defined from an intelligent syntax.

The example of intelligent languages we could present are composed from <O,A,R> triples as control structures. The A's have operations that also consist of agent message passing. The functions in AF are the agent functions capable of message passing. The O refers to the set of objects and R the relations defining the effect of A's on objects. Amongst the functions in AF only some interact by message passing. The functions could affect objects in ways that affect the information content of a tree. There you are: the tree congruence definition thus is more complex for intelligent languages than those of ordinary syntax trees. Let us define tree information content for the present formulation. Hence there is a new frontier for a theoretical development of the <O,A,R> algebras and that of the AII theory. <O,A,R> is a pair of algebras, <Alg[A],Alg[F]>(see section 3), connected by message passing and AII defines techniques for implementing such systems. To define AII we define homomorphisms on intelligent signature algebras.

Definition 2.7 An I-homomorphism is a homoprphism defined on algebras with intelligent signature I.

To define agent specific designs we apply HA-morphisms via the following definition.

Definition 2.8 Let A and B be I-algebras with signatures containing an agent signature HA. A HA-homoprphism from A to B is an I-homorphism with defined HA-morphism properties.

3. Multiagents and Mediators

The term "agent" has been recently applied to refer to AI constructs that enable computation on behalf of an AI activity. It also refers to computations that take place in an autonomous and continuous fashion, while considered a high-level activity, in the sense that its definition is software and hardware, implementation, independent [1] For example, in a planning problem for space exploration, an agent might be assigned by a

designed flight system [8,19] to compute the next docking time and location, with a known orbiting space craft. Agents are in most cases informable[1], thus allowing message passing actions. We can define AII software systems designed by AI methods as intelligent agent architectures, with external behavior that is a function of the degree of message passing actions and parallelism conceptualized.

Since our specifications consist of objects, actions, and relations defining the effect of actions on objects, we can define formal IF systems from the specifications and prove the specifications can be implemented by a set of agents. A *mediator* is a software module that exploits encoded knowledge about certain sets or subsets of data to create information for a higher layer of applications. and the definition goes on to state `It should be small and simple, so that it can be maintained by one expert or, at most, a small and coherent group of experts' Mediator instantiation is to populate a domain-independent service or tool with domain-specific knowledge.

Our*Mediator Specifications* consisti of a tuple of functions and relations of the form <O,(A,F),(RNA,RFA)>, where A is actions and F computes Flagrant Agents from APs to faults. (RNA,RFA) are their respective relations, NA for normal action and FA for flagrant or fault actions. In the example of the last section O is Coffee_shop, and serve-coffee an example of an action, a member of A. EXP defines the set F. The third line defines an example of a relation in RNA, and the last function is an example of a relation in RFA. This author invented a twin-engine agent-based computing system [12]. <A,F> := <Design_Agents, CoAgents>, consisting of Design_Agents := <O,A,RNA> and CoAgents;=<O,F,RFA>.The pairs <Ai,Fi> are modules composed to define <A,F>.

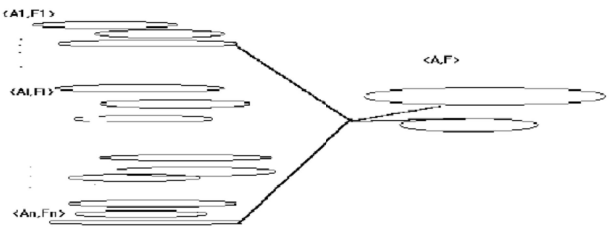

Figure 3

The modules are defined from multiple objects. The Design_Agents corresponds to an algebra Alg[A] of Normal Activities and CoAgents to an algebra Alg[F] for Flagrant Agent Computing, faults,recovery, and revision of actions. It consist of a pair of complex algebras, connected only by agent message passing-

Definition 3.9 A system is Intelligent Fail-safe, abbreviated by IF, when defined by a pair <Alg[A],Alg[F]> where A and F are I -algebras where I is an intelligent signature bearing agent functions.

Having defined the intelligent algebras, HA morphisms, and IF designs, we can define formal multiagent implementations for IF systems applying HA-homomorphism and formal implementation techniques Nourani[1,18], EKP[5] EKW[14]. It is obvious how to define AII implementations direct from HA-homeomorphism applied to our 1980's papers. The details are outside the scope of the present paper. Each of the Design_Agents and CoAgents consists of agents that are mutually, often pair-wise, informable. The systems <Ai,Fi>, each consist of objects, actions and relations. Actions could be in form of operations or message communication from one object to another. A set of computing agents forms Design_Agents and a dual set forms CoAgents. Thus a pair of systems is defined that can be implemented by agents that logically or physically can be thought of as running on several microprocessors. The algebras Alg[A] and Alg[F] define wrappers for the mediators as functions for interacting with resources. A wrapper is a tool to access known resources and translate their objects. The spontaneity and fault tolerance degree is a function of the intelligence of the agents implementing the <Design_Agents, CoAgents> pair. The agents have incomplete

information about the immediate needs of activating other agents or exceptions. Thus the efficiency and strength of functionality of our software systems are a function of the degree of intelligence we build in the implementing agents. The agents must have some reasoning ability to at least make transition or message passing decisions. This approach allows us to design systems that can deal with unplanned or erroneous behavior in an AI system.

The next step is defining the <Design_Agents,CoAgents> from the Flagrant Agent knowledge learning (Design_Agents) inputs. Its implementation consists of an autonomous pair of communicating systems to be defined in the following section. We have thus defined a formal computing model, the IF definition consisting of an algebra of processes and objects, with possible use of new parallel languages and intelligent object programming put forth in preliminary reports by this author in [7,9]. Theories for intelligent syntax tree computing are being put forth by this author in [3]. Starting form our techniques, programs capable of generating mediators, routers, and translators from formal specifications can be designed. In some cases these generators may work automatically, in some cases interactively with humans.

4. AI Model-based Software Engineering

The above multi-agent implementation of the mediator specifications implies model design with a pair of concurrent systems. Each of the two systems is to be designed with a collection of modules, such that there corresponds a module for each specification. A module consists of the minimal set of processes and objects that can be used as a basis for defining a computing activity. The objects and the operations of one set of modules once defined specifies the basis for Design_Agents, while those of the CoAgents' basis is defined by the dual module. The set of modules defining Design_Agents and CoAgents are synchronized by cross operations and interact by some operations that are implemented by message communications between Design_Agents and CoAgents.These operations are defined to either inform the various processes that are mutually dependent or to take the system from an active state in Design_Agents to an active state in CoAgents. Note that when exceptional conditions occur the active state is CoAgents. However, both sets of modules are considered concurrently "running." CoAgents' major task is that of handling unexpected events, recovery from faults, and revision of actions. Thus CoAgents has to know what agents can become active to compute for APs and be designed to activate remedies for ontology revision. If exception recovery takes place, in each module, the active module (a collection of agents) for a particular function, will be the Design_Agents' component, while the CoAgents component does concurrent checks for further exceptions should they be encountered. In each of the modules there are objects, processes defining the operations, and objects to which there is a corresponding function in the other module.

Thus Design_Agents and CoAgents imply a set of objects and processes defined by many-sorted OAR algebras. The objects 01 are many-sorted structures with the <pi>,<qi> and <ei> as the operations. RNA and RFA define the algebras via relations.

Design_Agents := <{01,<p1,...,pn>}, {O2,q1,q2,..},... {On,...},RNA> RNA is the set of relations on each object and cross objects.
CoAgents:=<{01,<e1,..., en}>, {02,<e11,e12,...,elm>},.. .,On,<...>},RFA>

RFA is the set of relations on each objects and cross objects. Each of the processes can have a corresponding agent in the dual family. The <Design_Agents, CoAgents> pair in a computing system "run" as a concurrent family of processes. Various functions in Design_Agents and CoAgents are represented

by agents that are mutually informable across the <Design_Agents, CoAgents> pair. For the fault model there is a predefined AP set and a corresponding functionality. The overall functionality of the system depends on the messages passed across from one agent to another. To each specification defined by Design_Agents there corresponds two modules running concurrent. The vision underlying mediators is one where domain experts, equipped with inter-operating software modules, provide value-added services for data access and processing over networks. The vision underlying facilitators is `one in which any system (software or hardware) can inter-operate with any other system, without the intervention of human users or their programmers' Interoperability is the capability to interoperate, often used at the transport layer.

5. Abstract Implementation and Model-Refactoring with Agents

Model driven SE deploys models and transformations as primary artifacts. The techniques presented are to use graph model representations and apply graph transformations at the model refactoring arena. Refactoring (Opdyke) is changes to the internal program structure to improve without changing the external functioning. We can lift refactoring to models, introduce model refraction as a new transformation, and apply the theoretical basis here and based on model graph transformations on TU Berlin's AGG is applied on graph grammars to specify model refactoring. Model consistency can be carried on UML with description logic. Our intelligent signature languages and morphisms allow us to carry on agent model computing, whereby agents facilitate model refactoring. Abstract implementations, i.e. to the process of transforming an abstract characterization of an AI or software system to concrete representations and executable code is accomplished with the new techniques. Thus implementations express the relationship between two forms of representations. The notion of abstract implementation defined by this author in [1,4] are either algebraic or model-theoretic (algebraic logic) definitions.

Let us refer to specifications of the form <O,A,R> as presentations. We also expect a presentation of the form <I[O],I[A],I[R]> for the implementing abstract or concrete machine. The former could be the designer's conceptualization, and the latter the specification of the syntax and semantics of a programming language. This is similar to how the problem was viewed by this author over a decade ago, and there were many research papers that were developed by us and EKP-EKW [5] for the most part. Informally the process of implementation was defined by this author to be that of encoding the algebraic structure of the conceptualization of a problem onto the algebra that specified an implementing machine (a programming abstract machine).

Thus the problem was that of defining such implementations by morphisms of algebras. The problems we are proposing are to address are much more complex. It is because the implementations proposed for AI systems are by multiagent designs. Each of the functions defined by <O,A,R> are implemented by agents, that characterize the implementation function I:<O,A,R> →<I[O],I[A],I[R]> is to be defining a mapping I: <Alg[A],Alg[F]>→ <Alg[I(A)],Alg[I(F)]> We refer to Alg[A] and Alg [F] are what we call *ontology algebras*. The implementation apping I defines wrappers to resources in a manner preserving the ontology algebra. Ontology algebras are multi-sorted algebras defining multiagent systems defined by formal agents, e.g., hysterectic or knowledge level agents and agent morphisms. A formal definition is provided in section 7.1. *The Ontology Preservation Principle* The AII is correct only if it preserves the ontology algebras. It will be abbreviated by AIIOPP. Widerhold's domain knowledge base algebra DKB consists of matching rules linking domain ontology. There are three operations defined for DKB.

The operations are Intersection- creating

subset ontology and keeping sharable entries. Union- creates a joint ontology merging entries. Difference- creates a distinct ontology and removing shared entries. Mapping functions must be shown to preserve ontologies. Applying AIIOPP we can state specific preservation principles as follows. The DKB Preservation Principle- All implementations must preserve ontologies under Intersection, Union, and Difference operations. The preservations are important to check model refactoring is consistent and does not cause invalid implementations.

6. Module Validation

6.1 AIIV

Multi-agent Object Level AI Validation make use of each framework to do most of the programming. AI systems consist of software modules some of which are models of human knowledge and reasoning. In the case of systems that are already designed object level views can be constructed to characterize the processes for validation. The following steps (a-c) indicate the approach to the AIVV is that of lifting (viewing). This author presented the AIVV for system design or for the developed AI systems a decade ago. It is apparent that knowledge acquisition is highly correlated to the methodology developed for an object-level approach to programming and the system designs indicating the prototype systems that are to be built. The tech- II plans are to build actual prototype automated AIVV techniques put forth in 4,11 are applicable to the development tems.

The following steps are proposed in developing the of practical expert and AI systems that are AIVV. There are techniques of AIVV presented here. systems well-suited for developing an object level view of such *a. Knowledge Acquisition and Specification:* designs, where various types of reasoning methods and comunicating objects can be brought together. The initial phase of designing the AIVV system software paper lays the foundations for the development of automated (FTS) [8,19]is to present the design in form of a specification .techniques for AIVV. The approach here is to start with the knowledge acquisition and representation phase, where knowledge on the associate Bridging The AI Software Gap system is to be represented at the object level. The practiced software verification and validation techniques. Some particular methods are put forth by this author in (3). presuppose the well-known software life-cycle methods of soft It requires us to make note of the domains that are to be ware design and implementation. Most practical AI systems expected, i.e. what objects there are and what the intended are designed and implementedby

AI paradigms that consist actions (operations) on the objects are, while fully defining of various reasoning models that are implemented by various such actions and operations. The actions could be in form of types of heuristic and expert systems. The planning impleentations processes in a system. are often best viewed as several paradigms only connected by message passing. It is not a case of "specified" modules that The relations amongst the objects and the operations can be expressed by objects and clauses. Once the modules correspondence and relations knowledge is represented in form of objects and actions indicating their operations and communications, their functional inter/relations. The present proposed project is to make use of the current ity can be expressed by specification that are defined below. object level Al systems by taking an object-level view of the design process. Thus specification are triples <0,*A*,R> consisting of objects, designed Al systeris such that AIVV systems could be devel actions and relations. Actions are operations or processes and objects. The problem of errors inthe models of reasoning itself the newer views of abstraction (1) and object-level program is an important concept addressed by the approach in [4].

This transforms the usual approaches to knowledge acquisition discipline, while there is a well-defined discipline of fault tolerance

multiagent software fault tolerance was first presented by this author, e.g. on :EAAI journal, A coherent knowledge acquisition in such approaches merely consists of well-defined approach to A//software fault tolerance is lacking obtaining knowledge in clausal form relating the expert knowl and the start-up concept paper presented in [4] lays the foundations for AIVV for to the AI expert, without any structural requirements actions. For AIVV systems additional principles to the validation and verification with. requirements are put forth here. We view software design as a methodology that commences defined have a dual definition in terms of the actions that are with a knowledge acquisition phase, followed by a specification phase taken for exception and recovery, and concluded by a system realization phase.

Thus at the knowledge acquisition phase the recovery and approaches defines knowledge acquisition for software corrective actions are to be noted at the object level. For fault tolerance, system specification for fault tolerant software each action on an object a dual action is to be identified for (FTS), and system realization for fault tolerant software systems exception and recovery. Knowledge acquisition ncludes exception knowledge as an essential component, as does system specification. *Multi-Agent View of* AIVV *Systems* System realization is by independent concurrent computing The term "agent" has been recently, for example) agents. FTS is defined in [18] by a pair of systems, each con-applied to refer to Al constructs that enable computation consisting of many computing agents. The two systems are dually behalf of an AI activity. It also refers to computations that synchronized to enable fault and exception handling and re- take place in an autonomous and continuous fashion, while recovery in an automatic manner. AIVV in this approach can considered a high-level activity, in the sense that its definition be correlated to the approaches to software validation and verification is software/hardware, thus implementation, independent [1].

6.2 Morphic Validations

Let us apply the definition for HA agents and HA morphisms to state a preservation theorem. Let A and B be I-algebras with the signature I containing HA agents. Let Alg[B] be an I-algebra defined from B implementing[1,4,5,6] a specified functionality defined by A. An AII is an implementation for Alg[A] by Alg[B].

Definition 7.10 Let A and B be I-algebras with intelligent signature I containing agents. An I-ontology is an I-algebra with axioms for the agents and functions on the signature.

Theorem 7.1 Let A and B be I-algebras with the signature I containing HA agents. The All with HA morphisms defined from A to B preserve I-ontology algerbas iff defined by HA-homorphisms.

Proof The definition for ontologies, HA morphism, definition 2.7 and 2.8, I-algebras and I-homorphisms entail the I-ontology axioms are preserved iff agents are carried by HA-homorphisms from A to B.

Theorem 7.2 Let A and B be I-algebras with the signature I containing KL agents. The All with KL morphisms preserve I-ontology algerbas iff defined by KL-homorphisms. **Proof** Similar to 7.1.

There are precise statements for preservation principles and mappings in [32]. DKB mappings are specific All's were the ontology algebra operations are the same at source and target. We prove in [32] DKB mappings are AIIOPP consistent.

6.3 ASF-SDF

New specifications based on general purpose algebraic specification formalism based on conditional term rewriting that is an interactive develpment environment that generates

environments interactive. ASF-SDF can provide a basis to carry out agent-based designs to accomplish model-based design and refactoring with our new agent models and implementation morphisms. An important technical point is that the agents are represented by function names that appear on the free syntax trees of implementing trees. This formulation will prove to be an important technical progress. The software development techniques which make use of the recent advances consist of several stages, a few of which are possibly iterated through, before the process of program production is completed. The stage of mapping the specifications onto computing agents is in part carried out by naming the computing agents, their corresponding objects in <O,A,R>, and their message passing actions presented in [1,18].

The further advanced methodology for synthesis of programs that this author has put forth [4,15] need a revisit in view of the present concepts. The characterization in the auhtor`s earlier publications expesses a paradigm which is further developed and lifted to AI applications in the present paper to be carried on with ASF-SDF

6.4 Fail-Safe Multiagent Processes

The above MAI implies design with a pair of concurrent systems, each consisting of a collection of kernels. Each of the components of a system's specification is a kernel. A kernel consist of those essential processes and objects that can be used as a basis for defining a computing activity. This term is analogous to the terminology familiar in 7 operating systems concepts. However, it is a level of abstraction higher and it is defined from intelligent computing agents, instead of ordinary processes. The objects and the operations of one set of kernels once defined specifies the FNA, while those of the FFA are defined by the dual kernel. The set of kernels defining FNA and FFA have a prespecified set of cross operations and message communications between FNA and FFA. Both collection of kernels are considered concurrently "running." FFA's major task is that of exception handling and recovery. If exception recovery takes place, in each kernel, the active kernel (a collection of agents) for a particular function, will be the FNA component, while the FFA component is concurrent checking for further exceptions should they be encountered. Note that this is not the methodology usually pursued in realizing systems. In each of the kernels there are objects, processes defining the operations, and objects to which correspond a dual function in the other kernel. Thus FNA and FFA are a collection of objects and processes. FNA := FFA = Each of the processes can have a corresponding agent in the dual family or at least to a pre-defined subset of the processes in the dual family. The pair in a computing system "run" as a concurrent family of processes. Various functions in FNA and FFA are represented by agents that are mutually informable across the pair. In a formal model the could be modelled by an infinite number of 's,running at highest computing speed, corresponding to all object and action instantiations. The overall functionality of the system depends on the messages passed across from one agent to another. To each specification defined by FFNRA there co rresponds two kernels running concurrent.

7 Future directions and Conclusions

The techniques proposed address the innovations claimed in the earlier section in realizing multi-agent, multi kernel AI software systems that are fault free. By fault tolerance is meant that the designed systems can recover from exceptions, in either anticipating them or spawning agent processes that can correct for the exceptional conditions and recover to a normal state. The designs are also ensured fault free since the design methodologies incorporate structural techniques that result in the design of AI systems that are verifiably accomplishing what they were purported to. The approach to fault tolerance proposed above has been actually applied in its basic form to challenging practical problems in system

design by the present author (Nourani 1992-1999). The techniques have been promising enough to prompt the author to conduct an R&D project to establish the methods and techniques for the AI and software community, presenting prototypes of various systems of interest. A technical paper by the author written towards the end of 1991 was the first in a series of concept papers crystallizing the field of Multi Agent Fault Free and Fault Tolerant Artificial Intelligence Systems. Agent Interoperability: Natural fit for multi-agent orchestration, collaborative AI. Future-Proof Alignment: Conceptual foundations already mirrored in today's agentic AI, neuro-symbolic systems, and multimodal transformers. Commercialization Path. Devising agentic stack specificatons for the application areas.

References

[1] Nourani,C.F., "Abstract Implementation Techniques For Artificial Intelligent Systems By Computing Agents- A Conceptual Overview," Proc. SERF-93, Orlanda, FL., November 1993.

[2] Wiederhold: "Interoperation, Mediation and Ontologies"; Proc. Int. Symp. on Fifth Generation Comp Systems, ICOT, Tokyo, Japan, Vol.W3, Dec.1994, pages 33-48. [4] Nourani, C.F., Abstract Implementations and Their Correctness Proofs," 1979, JACM April 1983.

[3] Nourani, C.F. 1994, Slalom Tree Computing," 1994, AI Communications, The European AI Journal,. December 1996, IOS Press, Amsterdam.

[4] ADJ-Thatcher, J.W., E.G. Wagner, and J.B. Wright,"Data Type Specifications: parameterization and the power of specification techniques," Proc. Tenth ACM SIGACT Symposi5um on Theory of Computing, San Diego, CA. May, 1978, 119-132.

[5] Ehrig, H., H.J. Kreowski, and P. Padawittz, "Algebraic Implementation of Abstract Data Types, Concepts, Syntax, Semantics, and Correctness," Proc. 7th International Colloquium on Automata, Languages, and Programming, Noordwijkerhout, July 1980,Springer-Verlag Lecture Notes in Computer Science, Vol. 85. New York. [6] ADJ-Goguen, J.A., J.W. Thatcher, E.G. Wagner and J.B. Wright, A Junction Between Computer Science and Category Theory (parts I and II), IBM T.J. Watson Research Center, Yorktown Heights, N.Y. Research Report, 1975.

[7] Genesereth, M.R. and Nilsson, N.J., Logical Foundations of Artificial Intelligence," Morgan-Kaufman, 1987.

[8] Nourani, C.F.,"A Multiagent Approach To Software Fault Tolerance," September 1991, (revised 1992) FLAIRS-93, Florida AI Conference, April 1993.

[8] Koopman, M.R.J., Spee J.W., Jonker W., Montero, L., O'Hara K., Mepham, M, Motta E., in't Veld L.J., VITAL Conceptual Modelling, VITAL Deliverable DD213, PTT Research, Groningen, December 1991.

[9]Gio Wiederhold: Mediation in the Architecture of Future Information Systems, 1989; published in IEEE Computer Magazine, Vol.25 No.3, March 1992, pp.38-49; republished in Hurson et al: Multi-database Systems: an Advanced Solution for Global Information Sharing; IEEE Press, 1993. [10] Gio Wiederhold et al: A Mediator Architecture for Abstract Data Access; Stanford University, Report Stan-CS-90-1303, August 1990.

[11]Nourani, C.F. 1999 Multiagent AI Design A New Emerging Software Engineering Trend, Engineering Applications AI- EAAI, Vol. 12:1, February 1999, pp. 37-42 Elsevier.

[12] Gio Wiederhold: "The Roles of Artificial Intelligence in Information Systems"; in ras Methodologies for Intelligent Systems, Lecture Notes in AI, Springer 1991, pp.38-51; republished in the Journal of Intelligent Information Systems, Vol.1 No.1, 1992, pp.35-56.

[13] Siewiorek, D.P., "Fault Tolerance in Commercial Com associate system can be viewed at the object level with object level knowledge, object specification, and message communiputers," Computer V23, no.7, July 1990.

[14] Ehrig, H. H-J Kreowoski, and H. Webber, "Algebraic Specification Scheme For Data Base Systems," Han-Meitner-Institut Berlin and Technical Universitat Berlin, HMI-B 266, February 1978.

[15]Nourani,C.F., "Double Vision Computing," December 6, 1993, Proc. IAS-4, March 1995, Karlsruhe, Germany.

[16]Kelly, J.P., McVittie, T.J., Yamamoto, Implementing Design Diversity To Achieve Fault Tolerance, IEEE Software V8, no.4, July 1991.

[17] Nourani,C.F. 1977, Design with Software Agents, Parallel Module Coordination and Object Languages, February 3, 1997. Technische Universit-at Berlin, Fachbereich 13 - Informatik,Berlin, Germany Proceedings, IFL98, September 1998, London, Conf. Proceedings Publisher Division of Computer Science, School of Mathematical, University of St. Andrews, Fife,

[18] Nourani,C,F; 1998, Multi Agent AI Implementation Techniques A New Software Engineering Trend IEA-98-AIE. In New Methodologies, Knowledge Modeling and Hybrid Modeling,

[19] Nourani, C.F. 2002. Multiagent Flight Control and Virtual Navigation, July 2002, HCI_Aero, MIT, Cambridge, October 2002.

[20] Intelligent Multimedia Computing Science: Business Interfaces, Wireless Computing, Databases, and Data Mining, Cyrus F. Nourani, American Scientific Publishers.