Software Agents and Intelligent Object Fusion

Cyrus F. Nourani

Revised October 1997

Abstract Techniques for modular software design are presented applying *software agents*. The conceptual designs are domain independent and make use of specific domain aspects applying *Multiagent AI*. The stages of conceptualization, design and implementation are defined by new techniques coordinated by objects. Software systems are designed by knowledge acquisition, specification, and multiagent implementations. Multiagent implementations are defined for the modular designs, applying our recent projects which have lead to fault tolerant AI systems. A new high level concurrent syntax language is applied to the designs. A novel multi-kernel design technique is presented. Communicating pairs of kernels, each defining a part of the system, are specified by object-coobject super-modules. New linguistics constructs are defined for object level programming with String and Splurge functions treating object visibility and messages. Treating objects as abstract data types and a two level programming approach to OOP allows us to define Pull-up abstractions to treat incompatible objects.

Keywords Abstract Objects, Intelligent Syntax, MJOOP, Multi Agent Object Level Programming, Multi Kernel Design With OOP, Software Agent Diffusion, Intelligent Object Fusion

Project METAAI@CompuSrv.Com

Copyright © Photo reproduction for noncommercial use and conferences is permitted without payment of royalty provided that the reference and copyright notice are included on the first page.

Academic Address USA UCSB when at University

1. Introduction

An AI and software system design paradigm is presented which incorporates a novel implementation method based on what we have called Abstract Intelligent Implementations ,A.I.I.[16] , put forth by this author[1,3] for an applied version). Intelligent implementation of software, that is design and implementation of software by AI techniques, is due to be an area of crucial importance[11]. The new design techniques combined with AI are applied to the real software engineering problems encountered in fields such as intelligent software systems, aerospace, and robot system design. Furthermore, there is demand for computing models and languages sharing a few basic concepts. There are simple countenances for describing location of data and a small number of features for coordinating the work of agents in a distributed/parallel setting. Coordination languages are being called into being by the computing community as language support for composing and controlling software architectures. The present paper project since 1991 has defined design with software agents and intelligent objects as a three-phased methodology. There is a knowledge acquisition phase, followed by a specification phase, and concluded by a system implementation phase. The present approach applies functional nondeterministic knowledge acquisition(FNKA), fault free system specification, and multiagent abstract implementations.

It is design *functional* in the sense that it defines knowledge acquisition with objects and functions defined on objects. It is *nondeterministic* in the sense that the formulation is with multiple concurrent kernels that are implemented by agents. System implementation is by independent concurrent computing agents. AI and software systems are defined in the present paper by a pair of systems, corresponding to two views of the functionality, each consisting of many computing agents. The two views are mutually synchronized to enable fault and exception handling and recovery in an automatic manner. The proposed methods have been presented in the context of problems that concern human error and expert judgment in AI a brief by this author [1,3], and for fault free AI system design in [3]. We are developing a basis for a sound theoretical and practical methodology for designing dependable AI and software systems. Some practical application areas in aerospace system design are presented in [1,3] and forthcoming papers.We present new techniques and languages for object level programming with intelligent trees implemented by agent functions.

2. Intelligent Objects and Multiagent OOP

The term "agent" has been recently applied to refer to AI constructs that enable computation on behalf of an AI activity[1,2,3]. 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/hardware, thus implementation, independent [1]. The present project develops new techniques, and linguistics constructs for programming with objects implemented by agents, based on a theory of computing with trees on signatures carrying agent functions on trees [5]. The agents are designated functions with specified functionality and message syntax. Thus context can be carried at syntax. We present new techniques and languages for object level programming with intelligent trees implemented by agent functions. We show in [6], and brief in the present paper, how a two-level language paradigm and intelligent object level programming can handle what otherwise is a complicated computing phenomena. There are objects as situated automata, for which abstract syntax trees and a computing theory merging with the current practice of programming theories is quite impossible. Objects are in the well-known sense of the word in object programming, abbreviated by OOP, for example. Objects consists 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 [6] is a recent technique developed by the author combining AI and software agents with OOP. The techniques for software design and implementation we call Design by Software Agent Diffusion and Intelligent Object Fusion are defined by our papers [3,6] and reviewed in brief here . The

IOOP project and Intelligent Software Agent and Intelligent Object Diffusion are what had been on our papers form 1991 [3,6]. There is a term popularized by an HP UK group called FUSION which might be relevant. The design techniques we had put forth with specifications, abstract intelligent object models, and abstract implementations are what fusion can be applied with. The specific areas are defined by our [3,6].

There has been the emergence of a class of languages and models named *coordination languages* and *software architecture description languages*. The languages are intended to provide a clean separation between individual software components and their interaction in the overall software organization. The separation makes large applications more tractable, global analysis feasible, and software reusable. Modular languages deal with languages, techniques, and tools for the design and implementation of large-scale software systems in a modular and extensible way. Since the Modula project many languages and environments support the goal. Modula-3, Oberon, Eiffel, Java, and SAP are a few new such languages. 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-comodule> 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.

3. Software Agents, IOOP and Design

The techniques, concepts, and paradigms defined by the Project are as follows. Intelligent Objects, MJOOP - The Multiagent Junction to OOP; Intelligent OOP Languages; String and Splurge Functions; The Object Coobject Design Paradigm; The Pullup Abstraction Function; and Multi Kernel AI Designs With Intelligent Objects. The project fills the gap between object programming as languages, design with software agents and artificial intelligence practice. Another interest for the above project is to design knowledge based systems, abbreviated as KBS, from formal specifications. The motivation for moldering KBS at the knowledge level in the above project is to obtain formal designs and techniques that are implementation independent.

We have defined intelligent syntax and put forth the basis for automatic tree implementation techniques for object level programming. We define *OOPI*, OOP Intelligent, functions and Pull-up Abstractions in brief from our [5,6] as a structural analogy to operation overloading. New linguistics constructs are defined for object level programming treating objects as abstract data types. The OOP defined by the present programming techniques have tree rewrite automatic implementations. This project is towards programming techniques that could provide a foundation for the method of computing that is inevitable with the current and forthcoming AI programming techniques, supported by our theoretical development in [5,6]. There is a gap between the OOP object programming trends and objects as applied to AI programming [13,14,16]. We had started on a project to bridge the abstract data type AI KR in our 1985 paper [15]. For OOP the AI application areas are outlined in [10,11,13]. The present paper offers a syntactic OOP techniques applicable to AI and ordinary OOP. An important technical point is that the agents are represented by function names that appear on the free syntax trees of implementing trees. The trees defined by the present approach have function names corresponding to computing agents. The computing agent functions have a specified module defining their functionality.

3.1 Intelligent Syntax Agents

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 [5,6] and by [4,17]. A set of function symbols in the language, referred to by Agent Function Set, are function symbols that are modeled in the computing world by AI and software agents. The objects, message passing actions, and implementing agents are defined by

syntactic constructs. Agents appear as functions, expressed by an abstract language that is capable of specifying modules, agents, and their communications. We have put this together with syntactic constructs that could run on the tree computing theories that are presented in [7].

Definition 3.1 We say that a syntax is **intelligent** iff it has intelligent function symbols. []

Definition 3.2 A language L is said to be an **intelligent language** iff L is defined from an intelligent syntax. []

The most recent language support for AI programming, languages are pursued where objects are at play without much handle on the syntax trees. Objects appear on various semantic networks and are a sort of situated automata, that implement a computation by asynchronous methods. Our goal in the present project is to have language constructs that allow us to handle objects on abstract syntax trees and implement the mystical behavior of situated automata by agents.

4. String and Splurge Functions

Formal techniques with intelligent syntax, String and Splurge programming linguistics allow us to treat visibility in a precise way and to get theoretical results [5,6,7] for MJOOP and OOP. These techniques and our formalization for AI computations [9] lead to an exciting new programming theory and practice for MJOOP. The new programming technique present the basis for programming with nondeterministic syntax[11].

Definition 4.3 We say that a function f is a *string* function iff there is no message passing or information exchange except onto the object that is at the range set for f, reading parameters visible at each object. Otherwise, f is said to be a *splurge* function. We refer to them by *string* and *splurge* functions when there is no ambiguity. Remark: Nullary functions are string functions. Amongst the functions in AFS only some interact by message passing. By defining String and Splurge functions we can have a formal treatment for programming with objects to handle object visibility and other related OOP computation problems.

5. The Pull-up Abstraction Function

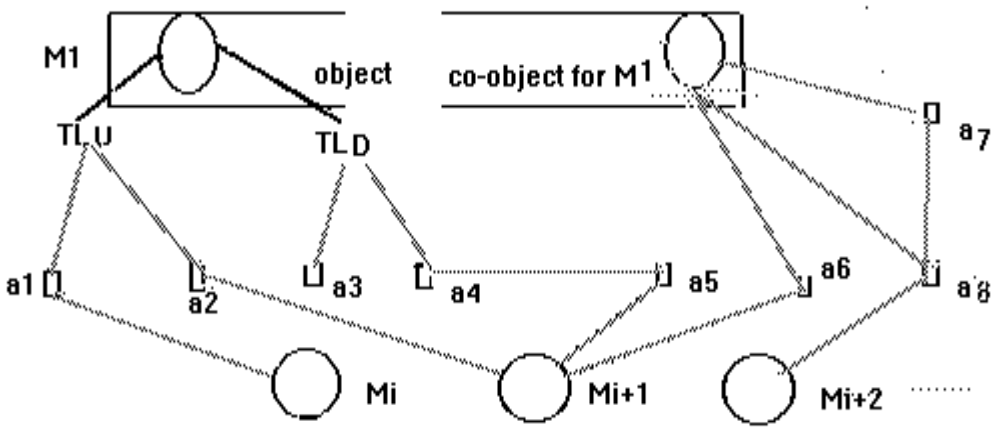
5.1 The Two Level Language Paradigm

The abstract syntax put forth for to be implemented are expected at two levels. Level 1 is a language that only expresses the functionality of modules by names of objects and their message passing Actions, and abstract data types , for example by SLPX[8]. Level 2 defines the functions themselves. The implementing agents, their corresponding objects and their message passing actions can also be presented by the two-level abstract syntax. From the practical stand point, the models as individual programs can be specified with well known specifications languages.

5.2 The Abstraction Function

To handle the compatibility problem, arising often in OOP, we have defined the **Pull-up** abstraction function in [7]. The idea is when defining operations across two incompatible objects, in the sense of types and the set of operations defined for each, we pull-up either or both objects to abstract objects for which we could define compatible operations across the resulting abstract objects. The analogy to Pull-up at the operation level is overloading, except Pull-up is a structural paradigm. A trivial example might be the objects defining a Chair with the operations of sitting on or getting up defined, and the object Table with operations having to do with putting things on, or taking things off the Table. We might Pull-up the objects Chair and Table to the object Dinning_Table.

6. The Object Coobject Design Paradigm



Design with Object Co-object Pairs

Mi's are objects, ai's agents, TLU and TLD specfic agent modules.

It would be nice to view the problem form the stand point of an example. The example of intelligent languages we could present have <O,A,R> triples as control structures, where O's are objects, A's the activities and operations, and R the defining relations. 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. <O,A,R> is made up from many <o,a,r>'s running in parallel.

Object:= IOOP_BREW
OPS:= Serve_Coffee (Type,Table_no) | .....
Serve_Coffee (Spectacular_Brew,n) => Signal an available robot to fetch and serve (Spectacular_Brew,table n)
Exp:= Serve_Coffee (Spectacular_Brew,Table_no) |...
Serve_coffee(Spectacular_Brew,Table_no) => if out_of_it notify Table_no;
offer today's-brew <and make use of
intelligent decision procedures to
offer alternatives>

In the above example OPS denotes operations, EXP denotes exceptions, and the last equation defines the exception action. In this example there is a process(action) that is always checking the supply of a specific coffee implementing the exception function. As another example, while planning for space exploration, an agent might be assigned by the onboard computer system to compute the next docking time and location, with a known orbiting space craft. The objects and message passing by agents are programmed on SLPX [8] and the actual module definitions, for example the Coffee-Shop is defined by the parameterized algebraic specification language Compose. The object-coobject paradigm has been applied by this author to define new computing paradigms for artificial intelligence with multiagents, called **Double Vision Computing**[10].

7. CONCLUDING COMMENTS

Many envisions software agents to be important area. Intelligent syntax and computing with intelligent objects are promising techniques for design and computing software agents and in some sense inevitable programming paradigms over the next decade. Thus the MJOOP paradigm is a new practical trend as well as a theoretical development for OOP.The techniques are applicable to distributed OOP,e.g. [12]. New programming techniques are put forth for object level programming with abstract data types.

REFERENCES

[1] Nourani, C F."Abstract Implementations By Computing Agents:A Conceptual Overview," March 3, 1993 , Proc. SERF-93, Orlanda, FL, November 1993.

[2] Genesereth, M.R and N.J. Nilsson, **Logical Foundations of Artificial Intelligence**, Morgan-Kaufmann, 1987.

[3] Nourani, C.F.,"A Multi_Agent Approach To Fault-Free and Fault Tolerant AI," Proc. FLAIRS,93, Sixth Florida AI Research Symposium, April 1993.

[4] Genesereth, M. R. An Agent-Based Approach to Software Interoperability, In Proceedings of the DARPA Software Technology Conference, 1992.

[5Nourani,C.F.," Abstract Intelligent Tree Computing And A Tree Rewrite Theory For OOP", January 19, 1995, Abstract Data Types 95, Holmenkollen, Norway.

[6] _____,"The IOOP Project, 1994, SIGPLAN Notices 30:2, 56-54, February 1995.

[7] _____,"Slalom Trees Computing," April 1993, AI Communications, December 1996, IOS Press.

[8] _____"Parallel Module Specification on SLPX," November 1990,SIGPLAN, January 1992.

[9] _____,"Planning and Plausible Reasoning in AI,"Proc. **Scandinavian AI Conference** , May 1991, Denmark, 150-157, IOS Press.

[10] Nourani, C.F., "Double Vision Computing," December 1993, IAS-4,Karlsruhe, Germany.

[11] Wiederhold, G. The Architecture of Future Information Systems, Stanford University Computer Science Department, 1989.

[12] Liskov ,B, et.al., The Language-Independent Interface of the Thor Persistent Object System, MIT Programming Methodology Group Memo 80, March 1994.

[13] R. Fikes and T. Kehler, 'The Role of frame-based representation in reasoning," CACM 28, No.9, pages 904-920, 1985.

[14] R.J.Brachman, R.E.Fikes, and H.J.Levesque, KRYPTON: A Functional Approach to Knowledge Representation," in Readings in Knowledge Representation, Brachman and Levesque editors,Morgan-Kaufmann.

[15]Nourani, C.F. and K.J. Lieberherr, "Data Types, Direct Implementations, and Knowledge Representation," Proc. 19th HICSS, Honolulu, Hawaii, January 1986, Vol II, pp. 233-243.

[16] Nourani,C.F.,"AII and Heterogenous Design," 1995, MAAMAW, April 1997. Ronneby, Sweden.

[17] Genesereth, M.R. and S. P. Ketchpel, Software Agents, Computer Science Department Stanford University, 1995.