Proceedings of the 7th WSEAS International Conference on Automation & Information, Cavtat, Croatia, June 13-15, 2006 (pp185-189)

EOOA: An Extensible Object Oriented Data Model For Automata Applications

Mina Zolfy, Saeed Nikmehr
Electrical and Computer Engineering Department
Tabriz University
Tabriz, IRAN

Abstract: - A navel data model for automata applications is presented. The EOOA model relies on the hierarchical possibility of object oriented concept. EOOA has an extensible object oriented structure with a powerful hierarchy and flexibility which could be used to represent automata in automata related applications. It also allows designers and developers to integrate different applications together in the same environment with a lesser effort.
In this work, we have illustrated the data structure model and also the technique of extending the structure for a sample application.

Key-Words: - automata, implementation, object oriented, inheritance, hierarchy

1 Introduction

The role of the computer science in technological advancement is inevitable. Therefore consolidating the relation between computer science and its practical applications is an effective enhancement step.

The computational theory with its automata branch as a field in computer science could help us in this enrichment approach [1]. Automata are used in many applications in software as well as hardware engineering [2].

As a design example in software engineering, a compiler design procedure could be mentioned. A compiler design starts with a complete definition of the corresponding language. Then an automaton is designed which accepts the language, and finally the compiler is produced. In hardware engineering, when the objective is the design of a sequential or any other controller digital circuit, the automata are used. The process involves an automaton design which after some algorithmic steps yields the intended circuit layout.

These examples present some of automata applications. Later, we will discuss the role of our innovative EOOA method in accelerating the technological advancement.

As the technologies grow in a faster rate, the decreasing of a project completion time becomes an important evaluation criterion. The automating of a number of process steps in the projects could considerably shorten the manufacturing time. The time reduction by means of automation, using the data model is presented in this paper.

Although, the developed tools for a specific field are not exactly alike, but they might share some common backgrounds. The ability of linking together such tools could have some benefits for users as well as developers. This could be done with using the proposed data structure in common practice. The proposed data model has enough potential to acquire all of the requirements of related applications, and also has enough extensibility for development of each application without affecting others.

One of the widely used data structures for automata related application is *ASTL*. However, it has not enough extensibility to fit several applications concurrently [3]. Also AutoML is a representation of automata but it is based on XML and must have a runtime data structure to be loaded into memory during the related tool execution [4].

The description of automata and some of its categorization is given in section 2. The proposed data model, which is to be used in common for automata related applications, is explained in section 3. Sections 4 and 5 contain conclusion and references in order.

2 What is an Automaton?

An automaton is a general term for any formal model of computation. In other words, an automaton is an

abstract model of a digital computer, which includes some essential features. It has a mechanism for reading inputs. It will be assumed that the input is a string over a given alphabet, written on an input file, which the automaton can read but not change. The input file is divided into cells, each of which can hold one symbol at a time.

The automaton can produce an output of some desired form. It may have a temporary storage device, consisting of an unlimited number of cells, each capable of holding a single alphabetical symbol. The automaton can read and change the contents of the storage cells. Finally the automaton has a control unit, which can change state in some defined manner.

2.1 Finite Automata

Typically, a finite automaton is represented as a *state machine*. That is, it consists of a finite set of states with some outputs, a set of transitions from state to state, a start state, a set of final states, and an input string.

A state transition usually has some rules associated with it that govern when the transition may occur, and are able to remove symbols from the input string. The main specification of Finite automata is its limited storage. The storage is restricted to the states and no temporary storage is included.

2.1.1 Acceptors (DFA, NFA)

An automaton whose output response is limited to simple “yes” or “no” is called an acceptor. Presented with an input string, an acceptor either accepts or rejects the string.

DFA (Deterministic finite acceptor) is one in which each state of an acceptor of this kind has a transition for every symbol in the alphabet. On the other hand is NFA (nondeterministic finite acceptor) in which states may or may not have a transition for each symbol in the alphabet, or can even have multiple transitions for a symbol.

2.1.2 Transducers

A more general automaton capable of producing strings of symbols as output.

2.2 Push Down Automata (PDA)

Such machines are identical to finite automata, except that they additionally carry memory in the stack form. The transition function δ will now also depend on the symbol(s) on top of the stack, and will specify how the stack is to be changed at each transition.

2.3 Turing Machine

A famous automaton is the Turing machine, invented by Alan Turing in 1935. It consists of a (usually infinitely long) tape, capable of holding symbols from some alphabet, and a pointer to the current location in the tape.

There is also a finite set of states, and transitions between these states, that govern how the tape pointer is moved and how the tape is modified. Each state transition is labeled by a symbol in the tape's alphabet, and also has associated with it a replacement symbol and a direction to move the tape pointer.

3 EOOA

3.1 Why common representation?

There are many applications that embed the same or at least similar structures of computational model in their infrastructure. Such systems use some scientific aspects in common, so their connectivity will be one of their significant advantages. This is the purpose of our proposed common structure for all automata applications in this work.

3.2 EOOA

All automata applications require a data structure for automata representation which could be our presented EOOA. EOOA is an object oriented data structure in which individual aspects of each automaton is laid on different objects of its classes.

Each successful data structure in all applications must have some features. Some of the features that are included in the EOOA are presented here:

3.2.1 Extensibility

Functionality requirement of useful tools, typically, increases over time. Especially in a research environment, a successful data model must provide a means by which new information or functionality can be associated with an existing description or entirely new kinds of informational structures can be represented on the spur-of-the-moment.

In order to maximize the performance and capacity, the ability to allocate space for extension

information, as the core information, within the same storage unit is important.

3.2.2 Efficiency

A data model must be much more efficient (in both space and time) to read and write than reading or writing the other equivalent representation. Both space and time efficiency dictates a binary (rather than strictly textual) data model. The binary representation must be compact, storing the required information with minimal redundancy.

Objects within the EOOA must be rapidly mapped into and out of the corresponding file representation. A direct map between EOOA and the corresponding file, such as a memory-mapped file, provides higher performance. However, such a technique does not satisfy our portability or integration requirements.

3.2.3 Integration

A practical data structure must facilitate integration of separately implemented units and extraction of design fragments for reintegration with other designs.

3.2.4 Security

Designs often embody proprietary information and intellectual property. Typically some subset of this information must be exported in order to make the design useful. This exported design information must be usable for exactly what the information supplier was intended; nothing more and nothing less.

3.4 EOOA Classes

In EOOA structure the pure data of automata are stored in the classes with names that are prefixed with ADMBase_ and ADM_ (for example ADMBase_State and ADM_State). Each class such as ADM_State inherits the relevant class with ADMBase_ prefix (in this example ADMBase_State). The classes are explained in detail in the following subsections.

3.4.1 Base Classes

Fig.1 illustrates EOOA Base classes, and parent-child relationships in various levels. Base classes of EOOA have three levels in their inheritance hierarchy. The root class in the shown tree of Fig.1 is named "ADMBase" which is the parent of all classes. It means that other classes are derived from it in several hierarchical levels.

The classes shown in Fig.1 have the "ADMBase" phrase as prefix in their names which illustrates the base class of "**A**utomata **D**ata **M**odel" that have the essential data members of automata. For example, Fig.2 shows data members of

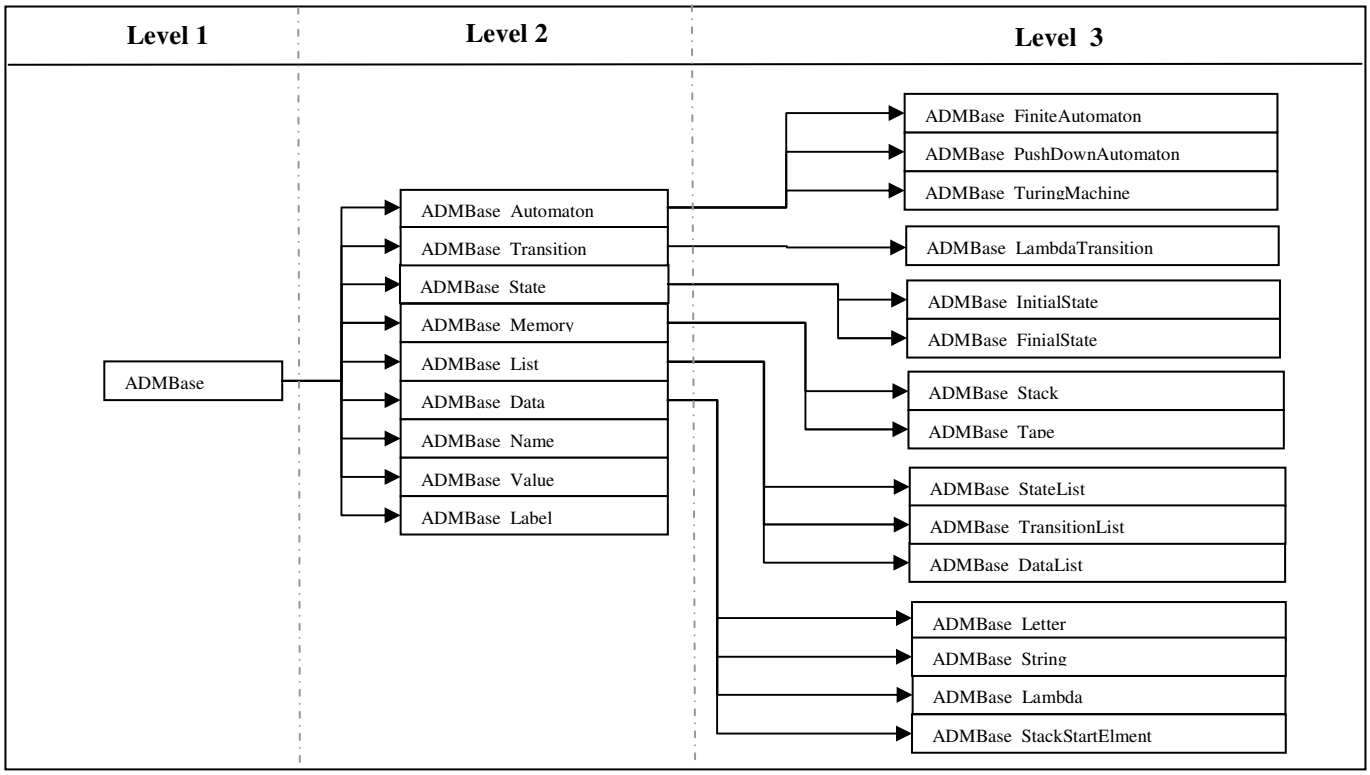

Fig.1: Three layer inheritance hierarchy of EOOA classes

"ADMBase_Automaton" class, its base parent, and its base children.

3.4.2 Main Classes

Main classes are named with ADM_ prefix and there is a one to one relation between the set of base classes and the set of main classes. Each class in the set of main classes inherits the related class in the set of base classes. This inheritance is shown more clearly for the levels 1 and 2 in Fig.3.

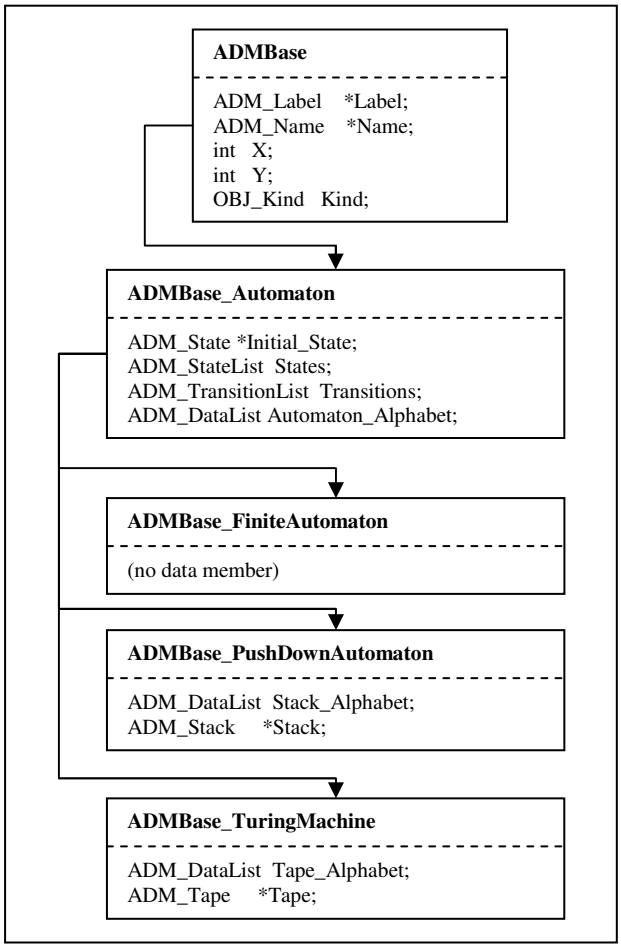

Fig 2 : Data Members of Automaton class and its children

The data members in the base classes are instantiated from the main classes, as shown in Fig.2 by means of an example.

3.4.3 Extension classes

Between each predefined level in the EOOA class hierarchy, one or more application-specific extension classes may be inserted. Application-specific methods may be inserted into the EOOA class hierarchy as part of these extension classes. Furthermore, application-specific data elements may be inserted into any of the extension classes associated with insatiable classes. For example, extension classes may add methods and data elements representing "temporary" information about application related features.

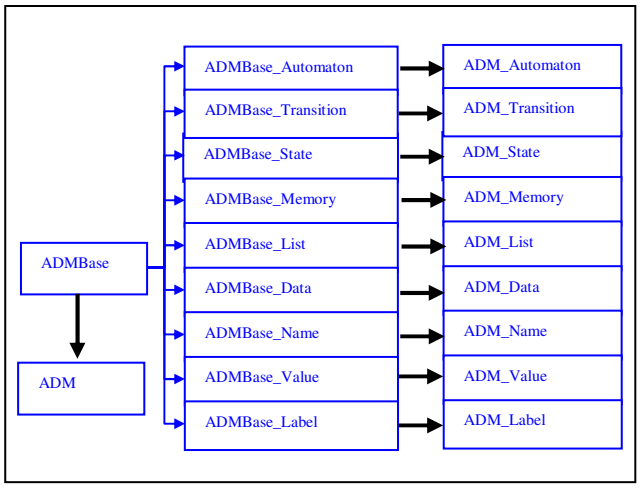

Fig 3 : Inheritance for main and base classes

For documentation clarity, all such extension classes and declarators within these extensions are distinctly identified. The names of any intervening extension classes should assume the form ADM<Extension Designator>_<Specific class name>. For example, a VHDL code extractor application might interpose an extension class ADMVHDL_Automaton between ADM_Automaton and ADMBase_Automaton**.**

Fig.4 shows an example of an extended class for ADM_State. As it is seen, the extension class is

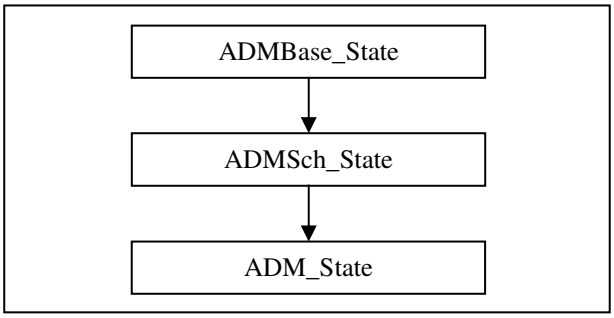

Fig.4 : Extended class example

inserted between the main and base classes in the inheritance tree. In a specific application, some classes

may need no new member (neither data nor function) for some classes.

As it has already been mentioned, EOOA is proposed in order to integrate the applications of automata to the same environment. When more than one extension is required to appear in the data structure, all of extensions are inserted between the main and base classes. If one application requires a data element from another application, then must be a child for it.

3.5 EOOA Example

An example of using EOOA structure for an application with two separate parts is discussed here. Each part could work separately and also both could be extended in the same structure.

Part one is a schematic editor for automata and part two is a behavioral Verilog code extractor for sequential circuits.

These applications are implemented using the presented structure, and the inheritance hierarchy is shown for a sample class in Fig.5.

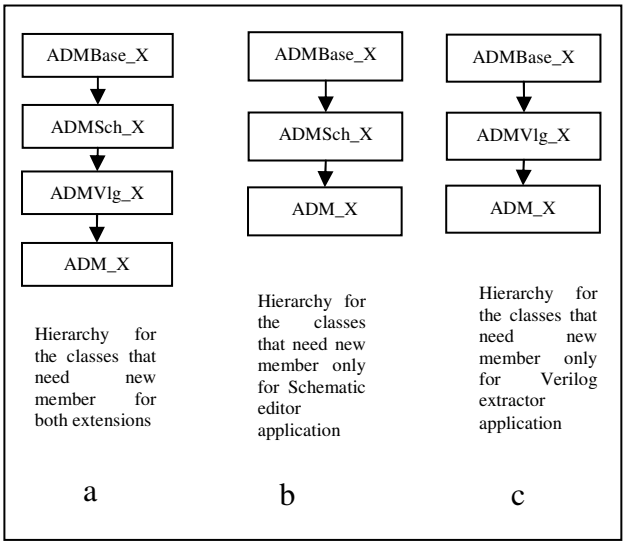

Fig.5: Two concurrent extensions

As shown above, two extension layers are inserted in the inheritance tree of some classes (fig 5.a). But all classes are not the same. Depends on the application some no extension for some classes will be added. First part is an application which provides a graphical user interface for drawing automata and second part is an application extracting Verilog code from an automata. If both parts gathered in an environment, the users would have the ability of both drawing and extracting code from an automata. If the second part works separately, then automata must be analyzed and loaded into the ADM structure using an separate analyzer then the code be extracted from analyzed automata.

4 Conclusions

A hierarchical object-oriented data model is presented. The model is implemented in C++ and it has been extended for the case of two applications.

The model could be implemented in any other object oriented language for any applications. The abilities of model are presented, and the extensibility is recognized as the major capability.

The proposed model is to be used as a standard for automata applications, and enables the interaction and integration of them.

5 References

[1] Peter Linz, “An introduction to formal languages and automata”, jones and Bartlett publishers,2001

[2] Shalyto A.A. “Logical control. Methods of hardware and software algorithms implementation.” SPb.: Nauka (Science), 2000

[3] Lincent le Maout, Dominique Revuz, “Tools to implement automata, a first step : ASTL”, Proceeding of the workshop on Implementation and Application of Automata ,1997,pp 104-108

[4] Yongbo An, Sheng Yu, “AutoML: Definition and Implementation”, 2004

[5] Yasmina Abdeddaim, Oded Maler, “Scheduling with Timed Automata”, PHD thesis, INPG Grenoble, 2003

[6] Pierre Wolper, Bernard Boigelot, “An Automata-Theoretic Approach to Presburget Arithmetic Constraints”, Proc of Analysis symposium, Glascow 1995, pp 21-32

[7] Richard Raimi, Ramin Hojati, Kedar S. Namjoshi, “Environment modeling and language universality”. ACM Transaction on. Design Automation of Electronic Systems ,2000, vol. 5, pp 705-725

[8] Michael Sipser, “Introduction to the theory of comptation”, PWS publishing company,1997.

[9] Hopcroft, J. and J. Ullman, “Introduction to Automata Theory, Languages, and Computation(3rd edition)”, Addison-Wesley.2006