High Level Design Reuse through Fuzzy Learning

Pao-Ann Hsiung
Institute of Information Science, Academia Sinica Taipei 115, Taiwan,
R.O.C.

Abstract

Object-oriented design has made possible the reuse of individual components during computer system synthesis. We demonstrate how the degree of reuse can be increased to a larger extent through a high level reuse of complete subsystems by the application of machine learning techniques and fuzzy logic in an object-oriented system-level synthesis environment. The fuzzy learning techniques were implemented in the ICOS methodology. It is shown by experiments that efficiency of synthesis improved by a factor of two to three after fuzzy learning was applied.

Keywords: *design reuse, fuzzy logic, machine learning, object-oriented system-level synthesis*

1. Introduction

Object-oriented (OO) synthesis allows the reuse of individual design component [4], but this is not sufficient. Often a system designer may change a few design specifications and must resynthesize the whole system. Unnecessary repetitions of identical synthesis steps are inevitable if previously designed subsystems are not stored or learnt. Motivated by this fact, *machine learning* techniques are directly incorporated into the synthesis scheme, thus improving synthesis efficiency by *reusing complete subsystems* learnt from previous experiences.

In object-oriented synthesis, design components are modeled as object classes and stored in a *Class Hierarchy* as illustrated in Fig. 1. The classes are classified into aggregate node (*A-node*), generalized node (*G-node*), and physical node (*P-node*) [3] depending on their respective relationships with their child classes, if any. *Object-oriented synthesis* is defined as a traversal of a Class Hierarchy starting from a root node (representing a desired computer system) downwards until the leaf-nodes (representing physical components). During traversal, synthesis related actions are taken corresponding to different object-oriented relationships. For example, iterator is used at an A-node, generator is used at a G-node, and instantiator is used at a P-node. Interested readers are advised to refer to *Performance Synthesis Methodology* (PSM) [4] and *Intelligent Concurrent Object-Oriented Synthesis* (ICOS) methodology [3] for further details.

Machine learning is basically classified into *Similarity Based Learning* (SBL) and *Explanation Based Learning* (EBL) [7]. There are two kinds of SBLs: *Empirical SBL* and *Rational SBL*; and EBL includes *Inductive Learning* and *Deductive Learning*. *Deductive Learning* is further classified into *Specification-Guided Learning* (SGL) and *Example-Guided Learning* (EGL). An example of machine learning used in the synthesis of VLSI systems is the *Learning Apprentice for VLSI Design* (LEAP) [8]. Besides, machine learning has been rarely used in synthesis. Fuzzy logic [9] has been used in the VLSI design such as in VLSI placements [6], but not in system-level synthesis.

Here, we mainly use SGL and EGL. In SGL, the synthesis system compares previous design specifications with the current user specification to derive an acceptable, previously learnt, partial system for current use. For EGL, we select from a set of example components the design that best meets the current user specification. Due to numerous specifications, comparisons among versions of component classes are fuzzified. This process is called *Fuzzy Specification-Guided Learning*, whereas EGL deals with physical parts, thus the comparison is more crisp; a “*most-often-used*” scheme is used.

Section 2 describes how fuzzy learning is applied in object-oriented synthesis. Section 3

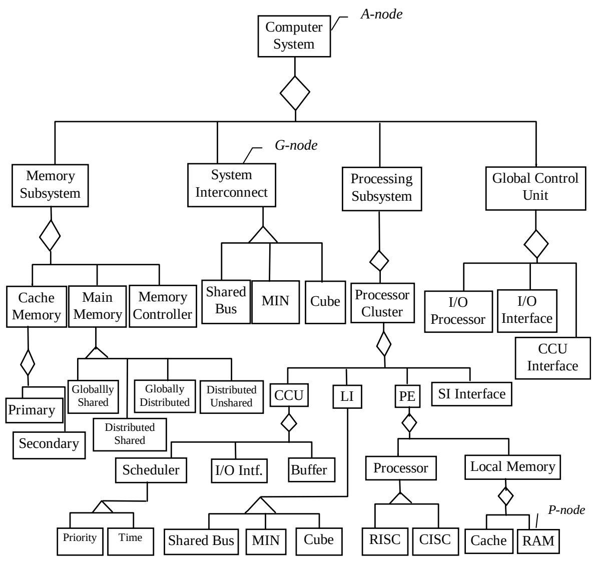

Fig. 1 Class Hierarchy

illustrates benefits of fuzzy learning through an example. Section 4 concludes with future work.

2. Learning in OO Synthesis

Figure 2 illustrates fuzzy SGL and EGL, which are explained in the following two subsections.

2.1. Fuzzy Specification-Guided Learning

Consider a component class *cls* in a *Class Hierarchy*, having a set of k specifications, $\{s_1, s_2, ..., s_k\}$. Suppose that n design results of *cls*, $V = \{cls_1, cls_2, ..., cls_n\}$, obtained from previous design experiences have the following sets of specification values: $X_i = \{x_{ij} \mid x_{ij}$ is the value of s_j corresponding to cls_i, $j = 1, 2, ..., k\}$, $i = 1, 2, ..., n$. Assume that the component class, *cls*, is currently to be synthesized again for the (n+1)th time, with the specification values, $X_{n+1} = \{x_{(n+1)j} \mid x_{(n+1)j}$ is the value of s_j corresponding to cls_i, $j = 1, 2, ..., k\}$. A fuzzy comparison between the values of a current user specification and those of each previous design is made using a fuzzy set (P), which represents the functional *proximity* of previous design results to the current one under design. The fuzzy membership function of P is defined as follows: (1)

$$\mu_P = V \rightarrow \begin{cases} [0,1] & \text{if design satisfies specs} \\ \{-1\} & \text{otherwise} \end{cases}$$

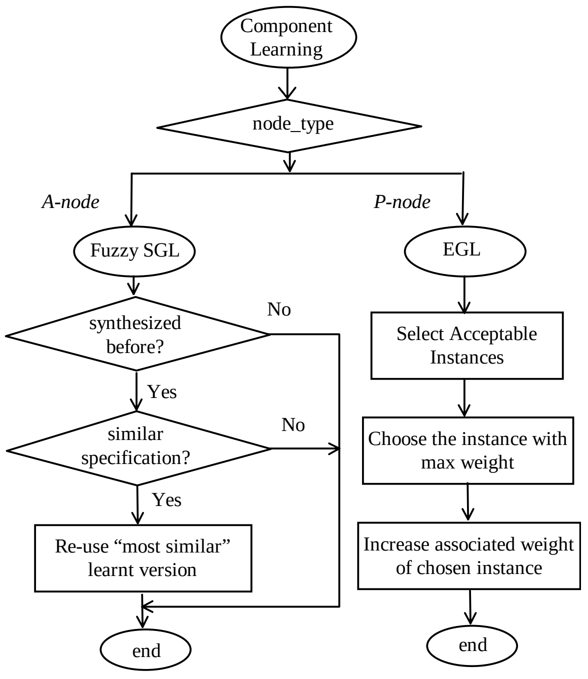
Component Learning
node_type
A-node
P-node
Fuzzy SGL
EGL
synthesized before?
No
Yes
similar specification?
No
Yes
Re-use "most similar" learnt version
end
Select Acceptable Instances
Choose the instance with max weight
Increase associated weight of chosen instance
end

Fig. 2 Component Learning in OO Synthesis

μ_P takes a negative value of –1 when the design result under consideration does not satisfy the specification of the component under design. Depending on the type of specification, the *proximity* of cls_i, $\mu_P(cls_i)$, is calculated as a sum over all the specification values, (2)

$$\mu_P(cls_i) = \sum_{j=1}^{k} \mu_P(x_{ij})$$

where $\mu_P(x_{ij})$, the partial *proximity* of cls_i corresponding to specification s_j, is defined in Table 1 for each type of specification, and w_j is the weight associated with s_j such that $\sum_{j=1}^{k} w_j = 1$.

The set of design results considered to be "similar" to the current one under design is called the *similarity set*, $S = \{cls_i \mid \mu_P(cls_i) \geq \delta\}$, where δ is a threshold value known as *the degree of similarity*. The higher the value of δ, the greater is the *degree of similarity* required between design results. If S is not empty, the design result having maximum $\mu_P(cls_i)$ is selected as the partial-design for reuse.

2.2. Example-Guided Learning

EGL uses a simple "*most-often-used*" scheme. Whenever a *P-node* is to be instantiated, the *most*

Table 1 Types of Specification and Proximity Calculations

#	Type of specification	$\mu_P(x_{ij})$
1	Exact value or set enumeration	-1 if $x_{(n+1)j} \notin \text{ENUM}\{x_{ij}\}$, w_j if $x_{(n+1)j} \in \text{ENUM}\{x_{ij}\}$
2	Minimum value	-1 if $x_{ij} < x_{(n+1)j}$; $w_j \dfrac{x_{ij} - x_{(n+1)j}}{\underset{1 \le i \le n}{\text{Max}} \left\| x_{ij} - x_{(n+1)j} \right\|}$ if $x_{ij} \ge x_{(n+1)j}$ and $M > 0$; 0 if $M = 0$, where $M = \underset{1 \le i \le n}{\text{Max}} \left\| x_{ij} - x_{(n+1)j} \right\|$
3	Maximum value	-1 if $x_{ij} > x_{(n+1)j}$; $w_j \dfrac{x_{ij} - x_{(n+1)j}}{\underset{1 \le i \le n}{\text{Max}} \left\| x_{ij} - x_{(n+1)j} \right\|}$ if $x_{ij} \ge x_{(n+1)j}$ and $M > 0$; 0 if $M = 0$, where $M = \underset{1 \le i \le n}{\text{Max}} \left\| x_{(n+1)j} - x_{ij} \right\|$
4	Approximate value	$w_j \dfrac{\left(\left\| x_{(n+1)j} - x_{ij} \right\| \right)^{-1}}{\underset{1 \le i \le n}{\text{Max}} \left\| x_{(n+1)j} - x_{ij} \right\|}$ if $x_{(n+1)j} \ne x_{ij}$; w_j if $x_{(n+1)j} = x_{ij}$

often used acceptable instance will be given a more favorable consideration. This scheme is implemented by associating a weight with each instance, which is increased whenever it is used for a system design. Instances with higher weights are the *most often used* and hence are considered more favorably during instance selection at a *P-node*.

Consider all instances I_j that meet the specifications of the *P-node*, P_0. The set of such instances is called the *acceptable set* (A_{P_0}) of that *P-node*. An instance with the maximum associated weight w_{I_j} is chosen from A_{P_0}. If I_k is used in the current design, then its weight w_{I_k} is incremented.

3. Learning Example

The target system is an asynchronous MIMD hybrid (shared-memory and message-passing) architecture with globally shared memory, shared bus as System Interconnection, 1024 RISC processors distributed in 64 clusters interconnected with a multi-stage interconnection network, and the performance specifications are $700,000 maximum cost and 500 MFlops throughput. This example was synthesized using ICOS [3]. Only partial synthesis is shown in Fig. 3 and described below to illustrate how learning saves design time and cost.

3.1. Fuzzy SGL at Processing Subsystem

Processing Subsystem (PSS) specifications are as follows: 16 RISC CPUs interconnected by a multistage interconnection network, local memory of size at least 1 MB and access time at most 8 ns, cache size at least 0.5 MB, and buffer size approximately 1 MB. Performance specifications include $10,000 maximum cost and 8MFlops minimum throughput. Other assumptions are: 8ns RAM cost=$30/MB, 7ns RAM cost=$35/MB, and 6ns RAM cost=$38/MB.

As shown in Table 2, using Equation (2) and Table 1, the proximity values of the designs are calculated as 0.3889, -0.3889, 0.6667, 0.6556, -0.5556, and -1.3333 with equal weights ($w_j = w_i$ for all $i \ne j$), respectively. The similarity set, S, is {A, C, D} with $\delta = 0.38$. The best choice is design C.

As shown in Table 3, learning helped save considerable design time and cost. Only one-third of the nodes need to be synthesized when learning is used and the design time decreased to nearly one-third of that required without learning.

4. Conclusion

Machine learning techniques have been successfully applied to object-oriented

Table 2 Fuzzy Specification-Guided Learning at the PSS Component Class

Designs	CPU-Model	LI	CP	PSS cost ($)	PSS Power (MFlops)	LM RAM size (MB)	LM Cache size (MB)	LM access time (ns)	CCU Buffer (MB)	Proximity μ_P
A	SPARC	MIN	16	10,000	9	1	0.5	8	2	0.3889
B	MIPS-R4400SC	MIN	18	13,500	10	1	0.5	7	2	-0.3889
C	Alpha-21064	MIN	16	9,500	9	1	0.6	7	2	0.6667
D	PowerPC-601	MIN	16	9,800	9	1.2	0.5	6	2	0.6556
E	Intel Pentium	MIN	18	10,000	8	1	0.5	6	2	-0.5556
F	PA-7100	Mesh	18	13,000	10	1.2	0.5	6	2	-1.3333
Current	{RISC}	MIN	16+	10,000-	8+	1+	0.5+	8-	1±	1

LI = Local Interconnection, CP = Cluster Processors, LM = Local Memory, CCU= Cluster Control Unit

system-level synthesis saving design time and cost and improving its efficiency by a factor of two to three as shown by experiments. Fuzzy logic helped solve the complexity of comparing design specifications. Future work will include application of our concepts to hardware-software codesign as in CMAPS [1]. Another future work is the incorporation of learning techniques into the formal model of synthesis called MOBnets [5], [2].

References

[1] P.-A. Hsiung, "CMAPS: A Cosynthesis Methodology for Application-Oriented Parallel Systems," To appear in *ACM Transactions on Design Automation of Electronic Systems*, Vol. 5, No. 2, April 2000.

[2] P.-A. Hsiung, "Parallel Design Automation of Computer Systems," *Proc. International Conference on Parallel and Distributed Processing Techniques and Applications* (PDPTA'98), Vol. 1, pp. 183-190, July 1998.

[3] P.-A. Hsiung, C.-H. Chen, T.-Y. Lee, and S.-J. Chen, "ICOS: An Intelligent Concurrent Object-Oriented Synthesis Methodology for Multiprocessor Systems," *ACM Transactions on Design Automation of Electronic Systems*, Vol. 3, No. 2, pp. 109-135, April 1998.

[4] P.-A. Hsiung, S.-J. Chen, T.-C. Hu, and S.-C. Wang, "PSM: An object-oriented synthesis approach to multiprocessor system design," *IEEE Transactions on VLSI Systems*, Vol. 4, No. 1, pp. 83-97, March 1996.

[5] P.-A. Hsiung, T.-Y. Lee, and S.-J. Chen, "MOBnet: An Extended Petri Net Model for the Distributed Object-oriented System-level Synthesis of Multiprocessor Systems," *IEICE Transactions on Information and Systems*, Vol. E80-D, No. 2, pp. 232-242, February 1997.

[6] E. Q. Kang, R.-B. Lin, and E. Shragowitz, "Fuzzy logic approach to VLSI placement," *IEEE Trans. on VLSI Systems*, Vol. 2, No. 4, pp. 489-501, December 1994.

[7] Y. Kodratoff, *Introduction to Machine Learning*, Morgan-Kauffmann, 1988.

[8] T. M. Mitchell, S. Mahadevan, and L. I. Steinberg, "LEAP: A learning apprentice for VLSI design," *Proc. 9th IJCAI*, Los Angeles, 1985, pp. 573-580.

[9] L. A. Zadeh, "Similarity relations and fuzzy orderings," *Information Sciences*, Vol. 3, pp. 177-200, 1971.

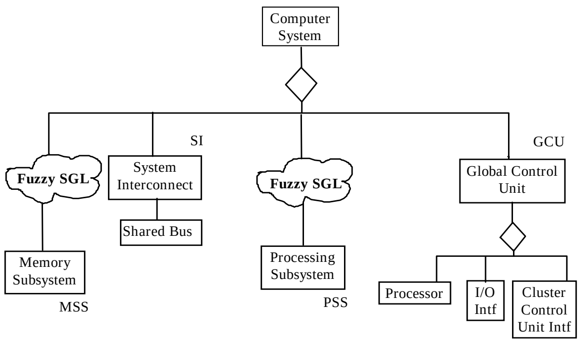

Fig. 3 A Fuzzy Learning Example

Table 3 Saving of Design Time and Cost in Example 2

	# of A-nodes	# of G-nodes	# of P-nodes	Total # of Nodes	Synthesis Time (s)
learning	4	1	4	9	392
No learning	9	4	15	28	1150