Parallel Object-Oriented Synthesis Methodology

Pao-Ann Hsiung
Institute of Information Science, Academia Sinica, Taipei 115, Taiwan, ROC.

Abstract

POSM, a new object-oriented synthesis methodology for multiprocessor systems, parallelizes the recently proposed Performance Synthesis Methodology (PSM). *POSM increases the efficiency of PSM and provides a mechanism for specifying the inter-dependence of system parts. POSM introduces a new object-oriented relationship for enhanced modeling of components and a new operator for administering design precedence among system parts. The concepts of POSM were implemented in the recently proposed ICOS methodology. POSM has been validated using a high-level Petri net model called MOBnets.*

Keywords: *multiprocessor systems, object-oriented design, parallel synthesis*

1. Introduction

Object-oriented technology had been successfully applied to the system-level synthesis of multiprocessor systems [6], [8], [2]. The efficiency of the methodologies can be considerably enhanced if the synthesis of components can be parallelized. Parallel computer-aided design had been used only in logic and high-level syntheses [1]. We now present a method for *parallelizing* object-oriented system-level synthesis.

Object-oriented synthesis use a *Class Hierarchy* (CH) consisting of classes representing system parts, as shown in Fig. 1. Each leaf class may be instantiated into a physical component. The synthesis process follows a top-down design scheme by traversing CH starting from the root class (representing a desired computer system) downwards until the leaf classes (representing physical components). Object-oriented relationships guide the synthesis of each component, which is accomplished by executing object-oriented operators.

Parallel design just like other parallel mechanisms have problems related to concurrency. While ensuring that parallel design gives an enhanced efficiency we must also ensure that newly created problems are tackled. In this paper we will not go into the details of problems created as a result of parallelism. Some problems related to parallel synthesis such as *deadlock avoidance* and *emptiness detection* were solved recently by the authors in [4].

The article is organized as follows. Section 2 describes related work. Section 3 describes POSM in detail. Section 4 gives a design example. Section 5 concludes with future work.

2. Related Work

Performance Synthesis Methodology (PSM) [6] introduced two object-oriented relationships, *aggregation* and *generalization*, and two operators, *iterator* and *generator*, to guide and accomplish the syntheses of component classes. *Aggregation* is the “whole/part” relationship and *generalization* is the relationship between a class and its one or more refined versions. The *iterator* iterates through each child node of an aggregate class, deciding whether to use it or not, based on the satisfaction of its specifications. The *generator* generates a number of acceptable specialized subclasses for a generalized class.

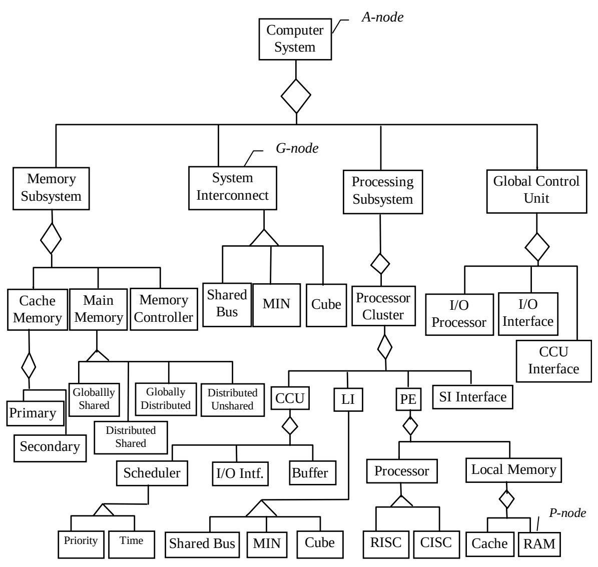


Fig. 1 Class Hierarchy

The order of component syntheses was sequential and fixed in PSM. Here, it is shown how it can be enhanced by parallelizing the tasks of component synthesis. Components may depend on each other as far as specifications are concerned. For example, the designs of a CPU class and of a RAM class may be quite different if the order, in which the two objects are synthesized, is changed. This inter-dependence of components may be physically perceived as hardware links between them. Parallel synthesis allows more than one component to undergo synthesis at the same time, hence this kind of dependence must be modeled into the synthesis process itself, otherwise the synthesized design parts when integrated, may not be a feasible system.

3. Parallel Object-Oriented Synthesis Methodology

As shown in Fig. 2, besides *aggregation* and *generalization*, one more object-oriented relationship is introduced to model dependence, it is called the *association* relationship. Whenever the design of adjacently connected components of a multiprocessor system are inter-related such that the synthesis of one affects the other, the two components are said to have an *association* relationship. *Association* is further classified into absolute and relative. *Absolute association* occurs

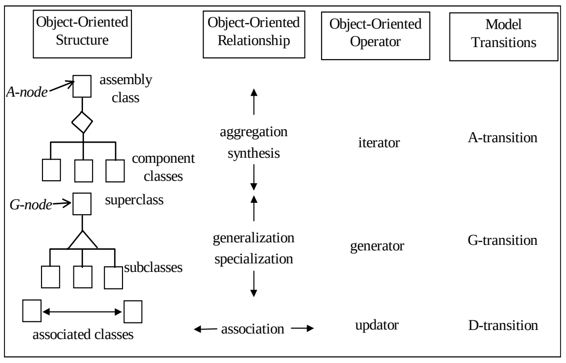
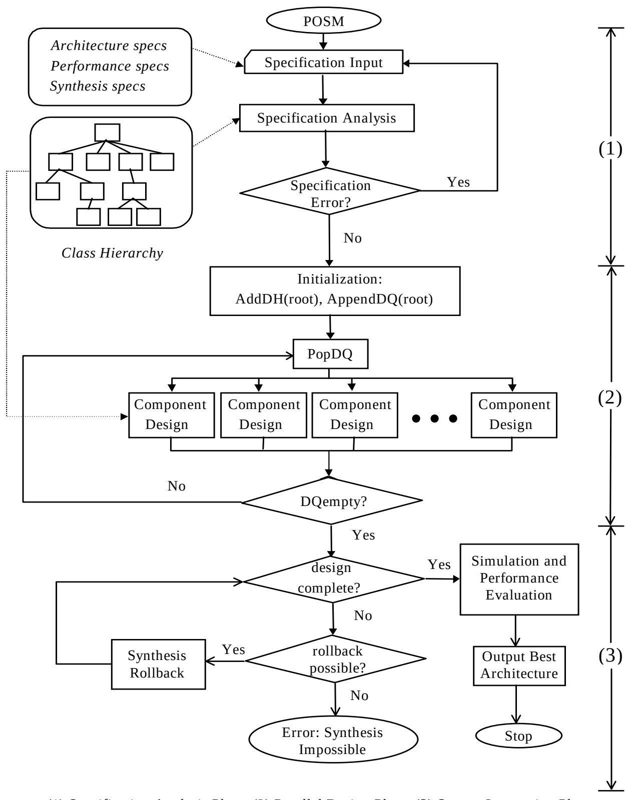


Fig. 2 Object-Oriented Relationships and Operators

between two classes when the design of a class depends on a post-design characteristic of the other class. Two classes having *absolute association* cannot be synthesized simultaneously, thus the order of synthesis is a total-order. Partial order of synthesis is observed between two classes having a relative association. *Relative association* is used to model the case in which the design of a class depends on a pre-design characteristic of another class. Two relatively associated classes can be simultaneously synthesized, once the dependent class has acquired the information it needs from its relatively associated class. This acquiring of information is accomplished by a third object-oriented operator known as *updator*. The dependent class uses *updator* to query its relatively associated class for the required specification.

For example, processor class and memory class are said to be absolutely associated because the *memory access time* (*m*) can be expressed in terms of the *processor cycle time* (*p*): $m = k \times p$, where k is some constant and it is assumed that m is a specification and p a post-design characteristic. Further, Cluster Control Unit (CCU) and System Interconnect (SI) are relatively associated because the CCU *data transfer rate* (d_{CCU}) and the SI *data transfer rate* (d_{SI}) are related as follows: $d_{\mathrm{SI}} = c \times d_{\mathrm{CCU}}$, where c is a constant and it is assumed that both d_{SI} and d_{CCU} are specifications.

The overall design flow of POSM is described in Fig. 3, where parallel design is obvious from the concurrent syntheses of components. There are three phases in POSM, namely, *specification analysis*, *parallel design*, and *system integration*.



(1) Specification Analysis Phase (2) Parallel Design Phase (3) System Integration Phase

Fig. 3 Overall Design Flow of POSM

A design hierarchy (DH) is used to record the design status and a design queue (DQ) is used to control the order of component design. Synthesis rollback becomes necessary in the third phase of system integration if there exist unsatisfiable component specifications. The main concept of POSM is presented here and the implementation details are described in [5].

In the specification analysis phase, a user inputs a desired system description using a specification language provided by ICOS. The specification language was presented in [5]. It is then checked if the specification does not contain any constraint contradictions or errors such as the cost bound is too low for a given system throughput or the minimum throughput is too high for a given cost. Architectural specifications may also contain errors such as specifying both a message-passing architecture with unshared memory and a distributed shared memory organization.

In the parallel design phase, starting from the root node, classes are appended to DQ as soon as it is *ready*. DQ sequentializes the beginning of each component synthesis but there is no blocking of synthesis, that is, the next component in the queue can begin synthesis without waiting for the previous one to finish. By a component being *ready*, it means that all association relationships have been attended to through the updator operator.

Allowing each component class to self-synthesize as soon as all its specifications are available, either inherited from its parent class in the Class Hierarchy or updated using the updator operator from associated classes, *parallelizes* object-oriented synthesis. Though several components may be undergoing synthesis at the same time yet the design precedence and the inter-dependence among them can be consistently maintained through the association relationships and the use of updator [5].

A system is complete when all components have been designed. This can be difficult to detect in a parallel environment, a detection mechanism was proposed in [4]. Basically, a component once having completed self-synthesis sends a *synthesis-complete* message back to its parent. Thus, when the root receives a *synthesis complete* message, POSM can be sure that the design is complete.

In the system integration phase, if a design cannot be complete then a deadlock has occurred. Synthesis rollback was the mechanism proposed in [4] and implemented in [5]. On applying synthesis rollback we can deduce if a design can be completed or not. If a design is complete, performance is estimated through simulation. Simulation is mainly accomplished through the SES/Workbench simulation tool along with our own system interconnection simulators. Implementation details of POSM can be found in ICOS [5].

4. Design Example

As shown in Fig. 4, synthesis begins by considering the design of the root aggregate class, Computer System (CS). The next step is the synthesis of the subsystems. Processing Subsystem (PS) and Global Control Unit (GCU) are not associated, but Memory Subsystem (MS) and System Interconnection (SI) are relatively

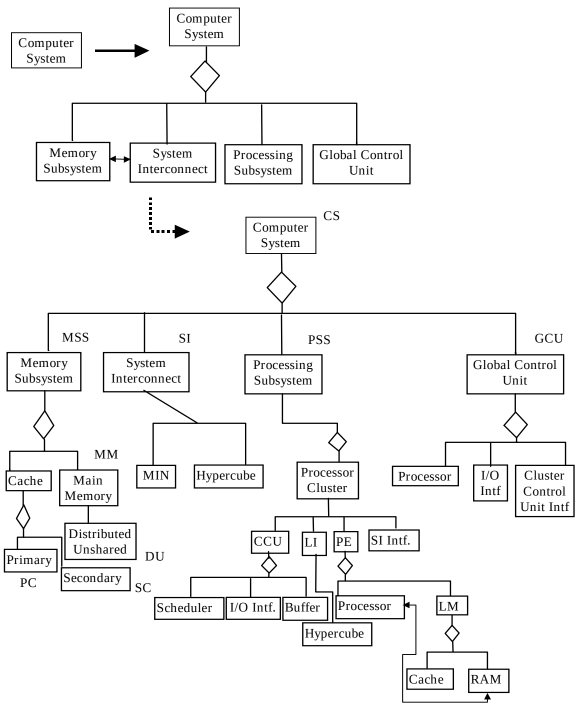


Fig. 4 Parallel Synthesis Example

associated. Hence, PS and GCU begin synthesis, while MS and SI use the *updator* operator to update their specifications (*data transfer rate* of SI or *memory access delay* of MS). Suppose the *data transfer rate* of SI is inherited from CS, thus MS can have its *memory access delay* updated from SI. Now, both the subsystems can begin synthesis.

Parallel synthesis proceeds and reaches a stage where RAM in Processor Cluster has to wait for the synthesis of processor to be complete before it can begin synthesis since RAM requires the *processor cycle time* specification for determining its *memory access delay*. Sequential synthesis would require a total of 28 synthesis steps whereas parallel synthesis required only 6 steps. It is observed from experiments that the total synthesis time differed by a factor of at least 4, thus significantly increasing the overall speed of synthesis.

5. Conclusion

POSM parallelizes PSM by introducing a new object-oriented relationship (absolute and relative association) and a new operator (updator). Both, the synthesis efficiency of PSM is increased and a new mechanism is provided for modeling the inter-dependence of system parts. POSM was briefly described and the main concept introduced. POSM has been validated using a high-level Petri net model called *Multi-token Object-oriented Bi-directional net* (MOBnet) [7] [4], which due to space limitations is not introduced here.

Future work will include designing more systems using POSM and verifying its capabilities and benefits. Another direction would be trying to applying the same principles to hardware-software codesign methodologies such as CMAPS (*Codesign Methodology for Application-Oriented Parallel Systems*) [3].

References

[1] R. Dutta, J. Roy, and R. Vemuri, “Distributed design-space exploration for high-level synthesis systems,” *Proc. 19th ACM/IEEE Design Automation Conference*, 1992, pp. 644-650.

[2] A. P. Gupta, W. P. Birmingham, and D. P. Siewiorek, “Automating the design of computer systems,” *IEEE Transactions on CAD*, Vol. 12, No. 4, pp. 473-487, April, 1993.

[3] 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.

[4] 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, CSREA Press, Las Vegas, Nevada, USA, July 1998.

[5] 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.

[6] 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.

[7] 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.

[8] S. Kumar, J. H. Aylor, B. W. Johnson, and W. A. Wulf, “Object-oriented techniques in hardware design,” *Computer*, Vol. 27, No. 6, pp. 64-70, June 1994.