SOC 软硬件协同设计中多任务性能评估算法*

李栋娜[1]，曹 阳[1,2]，张 奇[1]，郑 刚[1]
(1. 武汉大学 电子信息学院，湖北 武汉 430072；2. 武汉大学 软件工程国家重点实验室，湖北 武汉 430072)

摘 要：在分析现有的性能评估方法之上，提出了用 MTLS 算法对软硬件划分结果进行性能评估，验证系统软硬件划分的优劣。并且针对单任务图描述多 CPU 系统结构的不足，提出采用多任务图来描述的方法。首先搭建了软硬件协同设计的平台并描述了软硬件协同设计的流程，其次对目标系统进行形式化的描述，最后重点阐述了多任务图的 MTLS 性能评估算法，并与 MD, HNF, HLHET 三种算法进行了比较。实验结果表明，提出的 MTLS 算法比其他三种算法优越。

关键词：调度；分配；性能评估；软硬件划分；多任务图
中图法分类号：TP301.6 **文献标识码**：A **文章编号**：1001-3695(2005)06-0052-04

Performance Evaluation Method of Multi-Task in SOC Hardware/Software Codesign

LI Dong-na[1], CAO Yang[1,2], ZHANG Qi[1], ZHENG Gang[1]
(1. *School of Electronic Information, Wuhan University, Wuhan Hubei* 430072, *China*; 2. *State Key Laboratory of Software Engineering, Wuhan University, Wuhan Hubei* 430072, *China*)

Abstract: MTLS algorithm is proposed to evaluate the result of Software/Hardware partition and verify the validity of the partition. Multi-Task graph is proposed since the single task graph is not enough to describe the multi-processors system. Firstly, a Software/Hardware codesign system is set up and the flow of Software/Hardware codesign is described. Secondly, the object system is formalized by DAGs. Lastly, MTLS algorithm is discussed in detail. In order to verify the validity of the method, three previous algorithms are suggested. The performance comparison study shows that MTLS algorithm is better than others.
Key words: Schedule; Allocation; Performance Evaluation; Hardware/Software Partition; Multi-Task Graph

随着微电子技术的发展，芯片的集成度越来越高，片上系统(SOC)的发展已经是必然趋势。SOC 系统是将原来由许多芯片完成的功能，集中到一块芯片中完成。但 SOC 不是各个芯片功能的简单叠加，而是从整个系统的功能和性能出发，用软硬协同设计和验证方法，利用 IP 复用及深亚微米技术，在一个芯片上实现复杂的功能。软硬件协同设计是一个从高层抽象建模到低层硬件建模的过程。在从无时间概念的功能模型转换为总线周期精确的功能模型过程中，软硬件划分和性能评估是软硬件协同设计的难点和关键。

传统的芯片设计中工程师一般都是凭经验进行软硬件划分，随着系统结构复杂度的增加，凭经验已经难以给出适当的划分结果，而且系统的软硬件划分结果直接影响到后续的软硬件协同设计和最终的硬件实现，因此软硬件自动划分技术成为当前研究的热点。为了实现软硬件自动划分，并且评估软硬件划分的优劣，评估因子的选择以及性能评估算法的实现相当重要，调度和分配一直是贯穿在软硬件划分性能评估的始末。目前，软硬件划分技术可以分为两大类：①首先对系统初始化，然后在调度中分析系统特征并优化分配[1]。如 Yuan Xie 和 Wany Wolf 提出的方法是将系统模块初始化为硬件，然后根据时间约束条件改变分配方式将部分硬件由软件替换减小系统耗费[2]。②在固定分配方式下进行调度，采用遗传算法、模拟退火算法等进行寻优[3]。如 Dick 首先初始化多种分配方案构成种群，每个个体采用 MD 算法进行调度，然后采用遗传算法对种群进行演化，直到搜索到最优解[4]。

本文研究的软硬件划分技术是基于第二类方法，即是在固定的分配方式下进行调度。当系统结构中存在多个处理器时，系统的工作流程由多个并发的进程构成，单任务图有时无法描述多个任务的并发性，本文采用多任务图对系统抽象。针对 MD(Mobility Directed)[5], HNF(Heavy Node First)[6], HLFET(High Level First with Estimated Time)[7] 算法的缺陷，提出了 MTLS(Multi-Task List Scheduling) 性能评估方法。MTLS 算法考虑了系统的关键路径、任务的权重大小以及系统的时间约束，从全局把握。实验证明，MTLS 算法比上述三种算法能够更好地评估划分性能。

1 SOC 软硬件协同设计

1.1 SOC 软硬件划分平台

软硬件划分是软硬件协同设计首要问题，只有在系统各个模块确定了实现方式之后才能保证软件和硬件的设计可以同时进行。系统实现的软硬件划分平台如图 1 所示，由六个模块组成，分别为图形用户编辑接口、系统模型、分配模块、IP 特征

收稿日期：2004-06-06；修返日期：2004-07-21
基金项目：国家“863”计划资助项目(2002AA1Z1490)

库、性能评估模块、结果分析。

图形用户编辑接口是用户输入/输出的接口。系统模型主要是对系统进行形式化的定义,对系统进行建模。IP 特征数据库中收集了很多现有 IP 核的性能参数,如 IP 核的尺寸、最大工作频率、工作时的功耗、空闲时的功耗等。它是 IP 核复用技术的基础。分配模块主要是分析输入的系统模型以及该系统模型与特征数据库之间的关系,从 IP 库中提取特征参数,完成对数据流图的分配。性能评估模块是核心,主要是对分配完成后的数据流图进行调度,评估出系统的运行时间和功耗。结果分析模块主要是分析可行的划分解,比较得出较优的划分结果。

1.2　SOC 软硬件划分流程

本文采用速度、功耗和价格作为衡量的标准,这几个参数也是生产者和消费者共同关注的问题。要使划分的结果最优,实际上是一个多目标优化的过程,遗传算法可以很好地进行多目标寻优。图 2 是软硬件划分的流程图,整个流程在外框架是一个多目标寻优的演化过程,而在每次的演化过程中采用 MTLS 算法对分配后的模型进行调度和评估。

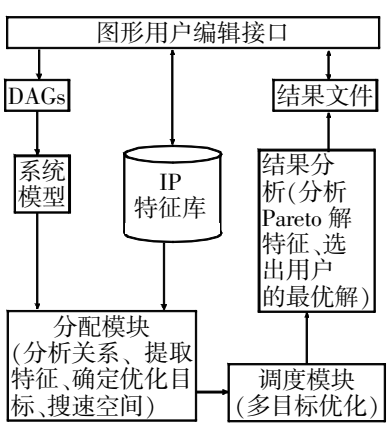

图 1　软硬件划分平台

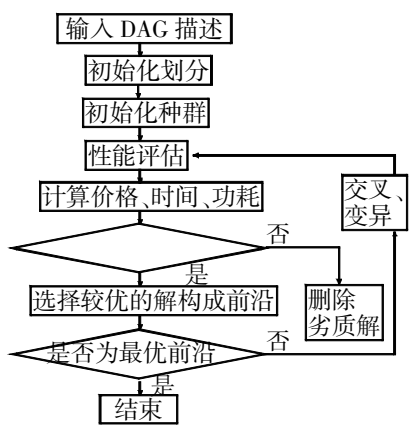

图 2　软硬件划分流程

首先,在图形接口输入系统形式化定义后的数据流图,从 IP 特征库选择合适的 IP 进行初始化的分配。同时初始化种群,界定优化算法的搜索空间,为多目标优化模块指定相应的优化目标。其次,调度种群中的个体,给出优化目标的评估结果,并将不满足时间约束条件的解直接删除。采用小生境技术选择较优的解构成前沿,经过演化,该前沿会逐渐靠近空间的最优前沿。最后,判断逐步演化的前沿是否为最优前沿,如果是最优前沿,演化过程结束,并输出一组非劣的划分结果和相应的目标值。

2　性能评估

2.1　形式化定义

系统结构是异步多处理器结构,由多个 RISC 和多个 ASIC 组成,多个 CPU 共享一个存储器。同一个 CPU 上可以执行不同任务,而同一个 ASIC 只能执行一种任务。为了较好地描述这样的系统结构,采用 DAG[8] (Directed Acyclic Graph) 来对系统进行抽象。DAG 称为有向无环图,如图 4 所示,DAG 由节点和弧线构成,$N_1 \rightarrow N_2$ 表示 N_2 的执行必须在 N_1 完成之后。首先要把系统分解为若干个并行的流程。对于每个流程,按照一定的粒度对硬件描述代码和软件代码进行分析,分解为若干个任务。然后确定每个流程中任务间(包括硬件和软件) 的依赖关系。任务映射为 DAG 中的节点,任务之间的数据依赖映射为 DAG 中的弧线,因此系统的调度问题也就转换成为对多任务图的调度。

对系统做如下定义: $DAG = \{ D_1, D_2, \ldots, D_i \}$, D_i 表示任务图, $D_i = \{ N, E \}$, $N = \{ N_1, N_2, \ldots, N_i \}$, N_i 代表任务图中的节点, NC_i 表示 N_i 的子节点, NP_i 表示 N_i 的父节点; $E = \{ E_{N_1 \rightarrow N_2}, E_{N_1 \rightarrow N_3}, \cdots, E_{N_i \rightarrow N_j} \}$, $E_{N_i \rightarrow N_j}$ 表示 N_i 到 N_j 的弧线。如果 N_i 和 N_j 都由硬件或者软件实现, $E_{N_i \rightarrow N_j}$ 代价为零,否则映射为总线。

定义 1　*indegree* 表示流入任务节点弧线数, *outdegree* 表示流出任务节点弧线的个数。*src* 代表入度为零的任务节点, *sink* 代表出度为零的任务节点。每个 *sink* 任务节点都会赋有截止时间 *deadline*, 表示对 *sink* 任务的时间约束。*EXT* 表示任务的执行时间。*data* 定义为总线上传输的数据。*ST* 表示系统调度所需的时间。

定义 2　EFT(Earliest Finish Time) 表示任务节点最早完成任务的时间,可以通过从 *src* 节点向下遍历获得。假设 N_i 有 p 个父节点,如果 $E_{NP_{ik}} \rightarrow N_i$ 的代价为零, $w = 0$, 如果 $E_{NP_{ik}} \rightarrow_{N_t}$ 由总线实现, $w = 1$(以后的 w 表示相同的含义)。EFT 由如下的公式确定:

$$\max_{1 \le k \le p} \{ NP_{i_k}^{EPT} + N_i^{EXT} + wE_{NP_{i_k} \rightarrow N_t}^{EXT} \}$$

定义 3　LFT(Latest Finish Time) 表示任务节点最晚完成任务的时间,可以通过从树的根节点向上遍历获得。假设 N_i 有 p 个子节点, LFT 由如下公式确定:

$$\min_{1 \le k \le p} \{ NC_{i_k}^{LFT} - NC_{i_k}^{EFT} - wE_{N_i \rightarrow NC_{i_k}}^{EFT} \}$$

$$N_i \mid sink$$

如果 N_i 为 *sink* 节点, $N_i^{LFT} = deadline$。

定义 4　*mobility* 表示任务节点 N 和总线 E 执行时间的灵活度。任务节点的 *mobility* 值定义为 LFT 和 EFT 的差值:

$$mobility = \text{LFT} - \text{EFT}$$

任务节点 N 和总线 E 的 *mobility* 的属性有如下关系:

$$E_{ij}^{mobility} = \min\{ N_i^{mobility}, N_j^{mobility} \}$$

定义 5　CP(Critical Path) 定义为关键路径,表示从 *src* 节点到 *sink* 节点执行时间最长的路径。Critical 是一个布尔量,表示任务节点 N 和总线 E 是否位于关键路径上。

定义 6　BD(Bottom Distance) 表士任务节点到 *sink* 节点的最长的执行路径,由如下公式确定:

$$\max_{1 \le k \le p} \{ N_{i_k}^{EFT} + NC_{i_k}^{BD} + wE_{N_i \rightarrow NC_{i_k}}^{EFT} \}$$

$$N_i \mid sink$$

如果 N_i 为 *sink* 节点, $N_i^{BD} = N_i^{EXT}$。

2.2　MTLS 算法的思想与实现

2.2.1　优先级确定

优先级的比较分为同一任务图中节点之间的比较和不同任务图中节点之间的比较。对于同一任务图,关键路径上的任务执行时间最长,要使任务尽早完成,即要尽量使关键路径上的任务尽早执行,因此关键路径上的任务优先级最大。关键路径上任务的父辈任务越早执行使得关键路径上的任务越早就绪,因此它们的优先级次高。不属于前两类的任务节点通过 *BD* 来比较它们的优先级。对于不同的任务图,首先根据任务是否在关键路径上来比较它们的优先级,在关键路径上的任务优先调度。如果两者都在关键路径上或者都不在关键路径上,就比较它们的 *mobility* 大小。*mobility* 体现了任务图的时间约束,约束条件紧迫的流程应当优先考虑。

2.2.2 速度和功耗的评估

算法实现的流程图如图 3 所示。输入为配置好的多任务图, 根据上述定义计算出 CP, 以及每个任务节点和弧线的 *mobility* 和 *BD* 参数。根据 2.2.1 节中优先级确定计算出每个任务节点和弧线的优先级大小, 搜索就绪的任务节点和弧线, 选择并行执行的节点, 包括优先级最高的硬件、优先级最高的总线和优先级最高的软件。如果有多个 CPU, 则要根据 CPU 的个数确定多个软件并行执行的情况, 并且在多 CPU 上进行调度, 保证 CPU 资源的充分利用。选择每次调度中最小的执行时间, 判断调度是否完成, 如果没有则继续循环调度, 如果结束就计算调度的累加时间。

假设 N_i 的实现 *IP* 为 *PE*, E_{ij} 实现 *IP* 为 *Bus*, $IP_i = \{ PE_1, PE_2, \ldots, PE_n, Bus\}$, $PE = PE_h \cup PE_s$, PE_h 代表软件 *IP*, PE_s 代表硬件 *IP*。由 IP_i 建立链表 IPList, 每个 IP_i 需要执行 j 个任务, 用 T_{ij} 表示, j 个任务 T_{ij} 建立链表 TaskList, 每个 T_{ij} 对应一个任务节点 N_i。

$T_{ij} = \{ N_{i1}, N_{i2}, \ldots, N_{ij}\}$, $IP_i \in PE$

$T_{ij} = \{ E_{i1}, E_{i2}, \ldots, E_{ij}\}$, IP_i 为 *Bus*

调度前为每个任务图建立链表 TaskReadyList 和 TaskCompareList, TaskReadyList 用来比较同一任务图中任务的优先级, TaskCompareList 用来比较不同任务图中任务之间的优先级。

实现 MTLS 性能评估算法的伪码如下:

```
输入: DAG = { D1, D2, . . . , Di}
输出: ST
schedule( DAG) :
建立 IPList;
for( each IPii ∈ IPList)
建立 TaskList;
while( true)
for( each IPi ∈ IPList)
Tij e = find_execute_node ( IPi) ;
Tij e = Ni ∪ E( 0 < i < s + 1) ;
mintime = min( N1ext, N2ext, . . . , Nsext, Eext) ;
假设 T 为执行时间最小的任务
TCij ready 增 1;
ST + = mintime;
If( IPList = φ)
return ST;
find_execute_node ( IPi) :
for( each Tij ∈ TaskList)
setcritical( ) ;
setmobility( ) ;
setbotdis( ) ;
就绪任务 Tijr 插入 TaskReadyList;
for( each Tijr ∈ TaskReadyList)
建立 TaskCompareList;
for( Dn ∈ DAG)
if ( Tijr ∈ Dn)
比较同一任务图中优先级大小;
优先级最大的任务 Tijmax 插入
TaskCompareList;
for( each Tijmax ∈ TaskCompareList)
比较不同任务图中优先级大小;
return 优先级最大的任务 Tij e;
```

调度完成后, *ST* 就是系统执行的时间, 也是系统速度的衡量标准。价格的评估比较简单, 只需要简单的累加。在已知调度时间 *ST*, 每个任务执行时间 N^{EXT}、单位时间的功耗 N^{pow}, 总线的执行时间 E_{ij}^{EXT}、总线工作功耗 Bus^{pow}、总线空闲功耗 Bus^{idle} 下, 系统功耗的评估由如下公式确定:

$$power = \sum_{D_k \in DAG} \{ \sum_{N_i \in D_k} \{ N^{EXT_i} \cdot N_i^{pow} + N_i^{idle} \cdot (ST - N_i^{EXT}) \} + \sum_{E_{ij} \in D_k} (E_{ij}^{EXT} \cdot Bus^{pow} \cdot Bus^{idle} \cdot (ST - \sum_{E_{ij} \in D_k} E_{ij}^{EXT})) \}$$

2.2.3 实验结果与分析

MD 算法的基本思想是根据任务执行的灵活度来确定优先级; HNF 算法的基本思想是根据任务的权重大小来确定优先级; HLFET 算法的基本思想是根据任务执行的路径大小来确定优先级。这三种算法都存在它们的不足, 它们没有从全局上考虑系统的整体信息, 而是通过系统的局部信息来调度。MTLS 算法考虑了系统的关键路径、任务的权重大小以及系统的时间约束, 是从全局上进行把握的。

为了验证算法的有效性, 采用包含两个 DAG 的系统作为例子, 如图 4 所示, 第一个 DAG 由七个任务节点组成, 第二个 DAG 由 10 个任务节点组成。任务的执行时间已经在图中注明, 黑色节点代表任务由软件实现, 白色节点代表任务由硬件实现, 每个 *sink* 节点的时间约束都为 1 400。采用 MTLS 算法, 计算出了 *CP*(*CP* 为 1 表示节点在关键路径上) , 以及每个任务节点的参数 *mobility*, BD, 并得到了每个任务在相应 *IP* 完成时间, 如表 1 所示(*mb* 表示 *mobility*) 。

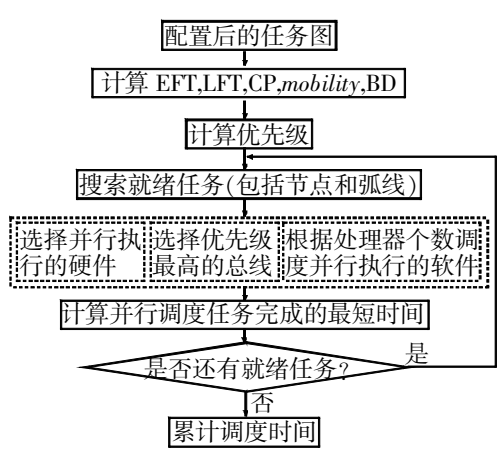

图 3 MTLS 性能评估算法流程

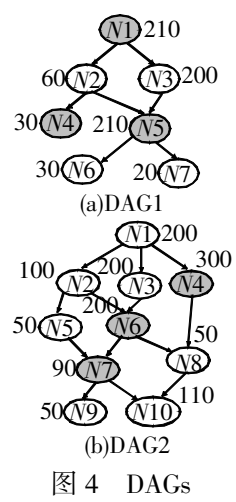

图 4 DAGs

可见最后执行的任务是 DAG_1 中的 N_7, 在 1 310 时刻完成任务, 因此采用 MTLS 算法需要 1 310 个时间单位。同样用 MD, HNF, HLFET 算法调度后, 得到结果如下: MD 算法需要 1 440 个时间单位, HNF 算法需要 1 390 个时间单位, 用 HLFET 算法需要 1 470 个时间单位。采用 MTLS 算法执行的时间最短。实验数据如表 1 所示。

表 1 实验数据

G	*N*	*CP*	*BD*	*mb*	PE_1	PE_2	PE_3	PE_4
1	N_1	1	1020	390		210		
1	N_2	0	540	780			270	
1	N_3	1	690	970				780
1	N_4	0	180	390	300			
1	N_5	1	360	500		1130		
1	N_6	0	30	390			1160	
1	N_7	1	20	770				1310
2	N_1	1	1160	240			200	
2	N_2	0	740	490				300
2	N_3	1	840	240		520		
2	N_4	0	580	620	500			
2	N_5	0	490	490				570
2	N_6	1	520	240	880			
2	N_7	1	320	240		970		
2	N_8	0	160	280			930	
2	N_9	0	50	420				1020
2	N_{10}	1	110	240			1280	

为了更好地比较四种算法的优劣, 采用个数不同的任务图来验证。分别输入个数为 2, 4, 6, 8 的任务图, 如图 5 所示, 可以发现随着 DAG 个数的增多, 四种算法的差距越大, MTLS 算法

比其他三种算法性能优越。实验结果说明, MTLS 算法适于并行进程比较多的系统, 它能够很好地减少并行进程的调度时间。

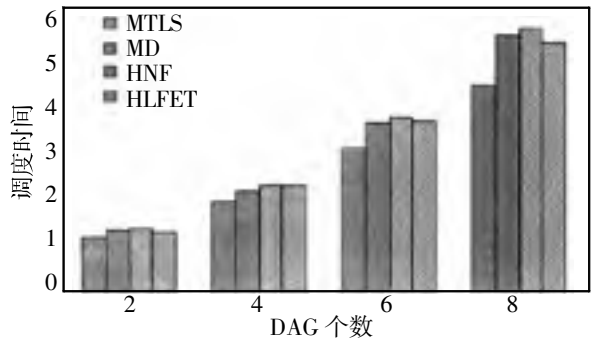

图 5　性能比较

MD 算法虽然考虑到了关键路径上任务, 但没有考虑到非关键路径上的任务对调度的影响, 而且会导致约束条件严格的任务一直保持高优先级, 使得其他任务无法抢占。HNF 算法没有考虑关键路径, 也没有考虑时间约束, 这种算法使用的任务特征太局部化。在两个任务图的关键路径的长度相差比较大的情况下, HLFET 算法容易延迟路径短的任务图, 虽然缩短了调度时间, 却无法满足约束条件。MTLS 算法既考虑了关键路径, 又考虑到了非关键路径, 同时也满足了时间约束条件, 从全局把握。同一任务图中的就绪任务和不同任务图中的就绪任务采用不同的策略调度, 使得对软硬件划分的性能评估最优。

3　结论

硬件集成度的飞速发展导致 SOC 产品的开发面临着巨大的挑战, SOC 的设计方法学的研究还是任重而道远的。多任务图比单任务图能够更好地描述包含多个处理器的系统结构, 因此提出用多任务图建模。本文的软硬件划分平台是在系统级进行建模, 提高了设计的层次, 在高层对系统的速度、功耗、价格进行评估, 避免了底层设计带来的不必要的损失。MTLS 算法是本文提出的适合评估软硬件划分性能的评估算法, 通过实验比较比其他三种算法优越。在下一步的改进方面, 可以在目标值的优化方面改进为优化更多的目标或是根据用户的需求动态选择优化目标, 以满足不同用户对产品的需求。

参考文献:

[1] upta R, Micheli G D. Hardware-Software Cosynthesis for Digital Systems[J]. IEEE Design & Test of Computers, 1993: 29-41.

[2] Yuan Xie, Wayne Wolf. Cosynthesis with Custom ASICs[C]. Proceedings of ASP-DAC2000, 2000. 129-133.

[3] Eles P, Peng Z, Kuchinski K, *et al*. System Level Hardware/Software Partitioning Based on Simulated Annealing and Tabu Search[J]. J. Design Automat. Embedded Syst., 1996, 2(5): 5-32.

[4] R P Dick, N K Jha. MOGAC: A Multiobjective Genetic Algorithm for Hardware-Software Cosynthesis of Distributed Embedded Systems[J]. IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems, 1998, 17(10): 920-935.

[5] M Y Wu, D D Gajski. Hypertool: Aprogramming Aid for Message-Passing Systems[J]. IEEE Trans. Parallel and Distributed Systems, 1990, 1(3): 330-343.

[6] B Shinazi, M Wang, G Pathak. Analysisand Evaluation of Heuristic Method for Static Task Scheduling[J]. Journal of Parallel and Distributed Computing, 1990, 10(3): 222-232.

[7] T L Adam, K Chandy, J Dickson. A Comparison of List Schedules for Parallel Processing Systems[J]. Communications of the ACM, 1974, 17(12): 685-690.

[8] Markenscoff P, Joe D. Computation of Tasks Modeled by Directed Acyclic Graphs on Distributed Computer Systems[C]. Circuits and Systems, IEEE International Symposium, 1990. 2400-2404.

作者简介:

李栋娜(1980-), 女, 硕士研究生, 主要研究领域为 SOC 软硬件协同设计; 曹阳(1943-), 男, 教授, 博士生导师, 研究方向为 SOC 软硬件协同设计、高速网络协议、智能网管、网络安全与性能评价; 张奇(1980-), 男, 硕士研究生, 主要研究方向为系统 ASIC 设计与验证等; 郑刚(1979-), 男, 硕士研究生, 主要研究领域为 SOC 软硬件协同设计。

(上接第 22 页)记录了用户应用系统的多层体系结构信息和数据库组态信息; 网页文件(HTML, ASP 格式)作为目标应用系统的表示层, 记录了目标应用系统的人 - 机交互操作界面; 与网页文件一一对应的组态表文件记载了完成此页面业务逻辑过程相关的所有逻辑组件及其关联信息。

工程上传主要是完成目标应用系统的部署和实现。它首先将记录组态操作结果的网页文件、组态表文件以及组态过程中用到的大量的构件(组件)分别上传到客户端、应用服务器端; 通过"组件注册和管理"程序注册组件并根据服务(如队列服务、事务服务、安全服务、内存数据库等)需求将其添加到 COM+服务中。工程上传的另一个重要工作是根据数据库组态结果完成目标应用系统运行数据库在数据库服务器端的配置和实现, 即根据数据库结构组态信息在数据库服务器端转换生成实际的运行数据库。

3　结束语

基于 CBDSCM 模型的软件开发以构件为中心, 以组态工具软件为手段, 以系统逻辑组态描述为纽带, 可以直接面向最终用户快速定制实现个性化的应用系统。目前, 设计实现的信息系统组态平台已经通过国家教育部电化教育办公室组织的专家鉴定, 借助该平台定制实现的个性化、B/S 结构信息管理系统已在深圳、肇庆、大连等多所中小学校园办公自动化系统中得到实际应用, 反馈良好。构件化大大提高了软件的开发速度和效率, 逻辑和实现的分离明显改善了应用软件系统的灵活性和逻辑可扩充性, 而系统逻辑组态描述能始终保持软件系统应用和需求的一致性。

实践证明, 这种基于 CBDSCM 模型的软件组态定制开发在提高软件的复用性、灵活性、保持软件系统应用和需求的一致性等方面具有显而易见的效果和广泛的实用价值。

参考文献:

[1] W Brown, K C Wallau. The Current State of CBSE[J]. IEEE Software, 1998, 15(5): 37-46.

[2] 张杰, 关永, 孙继平. 工控组态软件的可视化[J]. 中国图像图形学报, 2002, 7(2): 201-204.

[3] Hofmeister C Nord, R Soin D. Applied Software Architecture[J]. Addison-Wesley, 2000, 3: 61-96.

[4] 仲萃豪, 曹东启, 郭荷清. 信息系统的开发方法及其体系模型[J]. 软件学报, 1995, 6: 219-225.

[5] Reuse Library Interoperability Group. Uniform Data Model (UDM) for Reuse Libraries[R]. RIG: Technical Report: 0002, 1994.

[6] 常继传, 李克勤, 郭立峰, 等. 青鸟系统中可复用软件构件的表示与查询[J]. 电子学报, 2000, 28(8): 20-23.

作者简介:

李朝辉(1974-), 男, 博士生, 主要研究方向为软件复用、系统集成、ERP; 邓贵仕(1945-), 男, 教授, 博士生导师, 主要研究方向为复杂系统分析与综合建模、广义信息理论研究与应用、经济管理策略; 逯宇铎, 男, 副教授, 硕士生导师。