文章编号:1671-4598(2014)07-2324-03　　中图分类号:TP311　　文献标识码:A

控制密集型嵌入式系统软件构件模型研究

李海文，南建国，顾文灿，黄　雷，魏　斌
（空军工程大学 航空航天工程学院，西安　710038）

摘要：针对控制密集型嵌入式系统对软件实时性、安全性等要求高的特点，吸收现有构件模型优点，提出了控制密集型嵌入式系统软件构件模型—CIES模型；该模型引入了输入、输出控制接口，将数据传输的控制流与数据流分离；采用先进先出锁机制对临界区接口数据传输进行保护，保证了接口数据传输原子性及构件间交互的有序性，提高了系统可靠性与安全性；最后，给出实例的代码实现，并以汽车电子稳定控制系统为例，给出其CIES模型实现。

关键词：控制密集型；实时性；先进先出锁机制；原子性

Research of Software Component Model for Control－intensive Embedded Systems

Li Haiwen, Nan Jianguo, Gu Wencan, Huang Lei, Wei Bin
(Institute of Aerospace Engineering, Air Force Engineering University, Xi'an　710038, China)

Abstract: For the features that control－intensive embedded systems software has high requirement on real－time and safety properties, based on absorbing the advantages of the existing component models, a component model for control－intensive embedded systems—CIES is presented. This model separates data flow and control flow by the introduction of input and output trigger interface. The first－in/first－out locking mechanism which used to protect the data transfer for the conflicting set interfaces preserves the atomicity of data transfer between interfaces and the component interactions order, which improve the reliability and safety of the system. Finally, we give implementation code of the example and use CIES model to compositing the electronic stability control system of a car.

Keywords: control－intensive; real－time; first－in/first－out locking mechanism; atomicity

0　引言

控制密集型嵌入式系统（Control－ Intensive Embedded Systems）在航空、飞行控制、工业自动化、汽车电子、过程控制等领域应用广泛，它对系统实时性、可靠性、安全性、可预测性等要求较高。然而，随着系统复杂性不断增加，应用传统软件开发方法生产满足系统需求的软件越来越困难。为提高控制密集型系统软件开发效率，降低开发成本，结合基于构件的软件开发（CBSD）思想，提出了适合控制密集型嵌入式系统的构件模型—CIES模型（Control－Intensive Embedded System Software Component Mod－ el）。

目前，具有代表性的嵌入式构件模型有：PBO（Port Based Object）模型，PECO（Pervasive Component System）模型，SEES－ COA模型、Koala模型等[1-2]。上述嵌入式模型大都依赖于特定操作系统及应用平台，在实时性约束、安全性、预测性、控制性能方面存在不足，很难满足控制密集型系统要求。CIES模型在关注构件功能性、非功能性特征的基础上，分离控制流与数据流，提高系统执行效率。对数据交互进行严格控制，构件必须严格按照：输入触发—读入数据—处理数据—输出触发—输出数据的顺序执行；接口处数据传输为原子操作，保证数据传输正确性，提高了系统的安全性。

收稿日期:2014-02-27；　**修回日期**:2014-04-03。

作者简介:李海文(1989-),男,山东淄博人,硕士生,主要从事机载计算机技术方向的研究。

1　CIES构件模型

CIES构件模型是一种面向控制密集型嵌入式系统的构件模型，由C构件、连接子、服务协议构成。C构件用于实现系统具体功能，是CIES模型的主体，C构件间由连接子连接，采用pipes－ and－filters模式[3]进行数据交互，通过定义多样的构件接口与连接子，分离控制数据与执行数据。

1.1　C构件模型

C构件模型如图1所示，构件由外部接口与内部实现构成，接口是构件与外界交互的唯一通道。设计5种外部接口（IIG接口、OIG接口、CI接口、SI接口、SVI接口），满足用户功能性需求与非功能性需求（功能性需求指构件所提供的功能服务；非功能需求指构件在环境依赖、资源控制、时间性、调度性、聚合性及监控模拟等方面的非功能特性[4]）。C构件从构件粒度来看可分为复合构件与原子构件，复合构件由子构件集成。

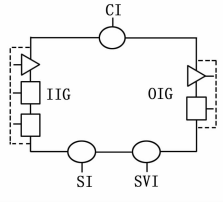

图1　C构件模型

输入接口集IIG（Input Interface Group），由一个输入触发接口（Input Trigger Interface，图中左侧三角形所示，简称

ITI）和一组数据输入接口（Input Date Interface，图中左侧矩形所示，简称 IDI）组成，输入触发接口用于控制数据输入接口状态。

输出接口集 OIG（Output Interface Group），由一个输出触发接口（Output Trigger Interface，图中右侧三角形所示，简称 OTI）和一组数据输入接口（Output Date Interface，图中右侧矩形所示，简称 ODI）组成。当数据在 C 构件内部处理完成后，触发 OTI 接口，数据由 ODI 接口输出。

配置接口 CI（Configuration Interface），属于非功能性接口，用于构件的动态及静态配置。主要包括：环境配置接口、资源配置接口、性能配置接口、聚合关系配置接口。完成调度性能配置，时间周期性、截止时间约束，同步异步关系定义等诸多方面功能。

状态接口 SI（State Interface），该接口用于构件性能测试及状态监控。可用于配置完整性及正确性验证、服务线程运行状态监控等。

服务接口 SVI（Service Interface），定义了构件基本信息（制作信息、功能信息、构件类型、运行信息、交互类型、交互协议。）

1.2　输入、输出接口数据交互

C 构件模型与其它构件模型相比最大特点是对构件行为的严格控制，构件间不能通过输入、输出接口自由交互。

服务是构件功能性描述的基本单元，一个服务过程是指从输入触发、数据读入、处理、输出触发、到输出的过程，每个构件可以包含多个服务，一个服务（S_i）可以表示为与之对应的输入接口集（IIG_i）与输出接口集（OIG_i）的二元组合，即 $S_i=(IIG_i, OIG_i)$。

将 C 构件模型简化后如图 2 所示（只保留输入、输出接口集，分析构件数据输入、输出及构件间交互），构件执行可以描述为以下 3 步。

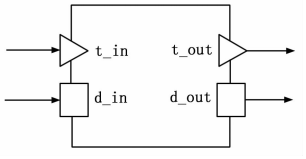

图 2　C 构件简化模型

（1）控制数据到达 t _ in 接口后，触发并激活该接口（如果 t _ in 已经被激活，则忽略其它控制数据的触发）。

（2）d _ in 接口将待处理数据读入构件内部，对应服务 S_i 被激活，进行数据处理。对于原子构件，服务对应相应的执行函数；对于复合构件，服务由子构件及连接子组成。

（3）数据处理完毕后，控制流触发 t _ out 接口（若此时有数据输出，则 t _ out 接口不会被再次出发），数据从 d _ out 接口输出。

由上述分析可以看出，从输入触发接口被触发到数据读入构件内部的过程，以及输出接口被触发到数据输出完毕的过程属于原子操作，不允许被其它任务中断。正是由于控制接口的引入，保证了接口处数据传输的原子性及构件间的交互顺序。

图 2 所示构件，其输入接口集输入过程实现代码如下。

```
int Component_t_in(Component_svc * svc)
{   //如果该服务被占用,则忽略触发
    if(svc->active)
      return 0;
    else
    {
      Component_transfer_d_in(sve);
      return 1
    //调度构件执行
    }
}
void Component_transfer_d_in(Component_svc * svc)
{
  //将数据复制到接口内部:cnx_d_in->d_in
  svc->d_in+svc_h->cnx_d_in;
}
```

构件输出接口集,输出过程代码如下：

```
START_Component Component_t_out(Component_svc * svc)
{
  return START_Component_t_out;
}
void Component_transfer_d_out(Component_svc * svc,
            int * * dest)
{
  //检测数据是否更新
  if(svc->d_out_updated)
  {
    dest=&(svc_h->d_out);
    svc->d_out_updated=0;
    //重设数据更新标识符
  }
}
```

1.3　连接子

连接子概念引入已久,传统连接子的作用包括通信、转换、辅助交互和协调控制[5]。在 CIES 模型中连接子应用更加广泛,定义 6 种连接子,如图 3 所示。

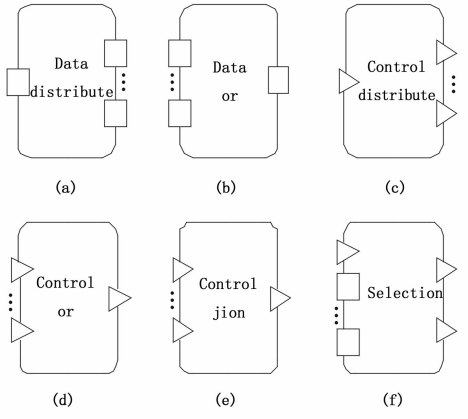

图 3　连接子模型

(a)为 Date distribute connector,由一个数据输入接口,两个以上的数据输出接口组成,从左端输入的数据流在输出端被分流。

(b)为 Date or connector,由一个数据输出接口,两个以上的数据输入接口组成,输入端输入的数据在输出端被合并。

(c)为 Control distribute connector,由一个输入触发接口,

两个以上输出触发接口组成，当输入端被触发时，所有输出触发接口被激活。

(d)为 Control or connector 由两个以上输入触发接口，一个输出触发接口组成，任意输入端触发都会激活输出端。

(e)为 Control join connector，接口组成与 d 相同，它与 d 的区别在于：当所有输入端被触发后，输出端才被激活。

(f)为 Control selection connector，由一个输入触发接口，至少一个数据输入接口，多个输出触发接口组成。该连接子有不同的工作模式，数据输入接口输入不同数据会触发不同的工作模式，从而将输入触发接口的控制流从不同输出接口输出。

定义多样的构件连接子，保证了数据流与控制流的可靠传输，保证系统安全性。

2 临界区接口交互顺序

如上文所述，接口集数据传输为原子操作，一般而言，不同接口集间的数据传输没有语义上的相关性，因此不会对其原子性产生影响。但在多任务并发执行中，接口可能被不同任务的进程同时访问，从而产生冲突，因此，对于临界区内接口，仅用输入、输出控制并不能保证数据传输的原子性及构件间的交互顺序。这里我们引入锁机制对临界区资源进行保护。

我们假设实时系统采用抢占式任务调度方式中的先进先出(FIFO)调度，该方式是实时嵌入式系统最常用的调度方法之一。同时不同的任务可以向同一组接口写入数据。

2.1 临界区接口

我们首先给出临界区接口的判别方法，以数据从输出接口集传输至输入接口集为例。

定义 1：设两组输出接口集(以下用 o_1 与 o_2 表示)与同一组输入接口集(用 I_1 表示)连接，在多任务并发执行时，o_1、o_2 有同时访问 I_1 可能性，用二元关系 $PC(o_1, o_2)$ 表示。

定义 2：用 OG 表示输出接口集，PC * 表示 PC 的传递闭包，PCS(可能发生冲突的区域)用公式(1)表示为：

$$PCS = \{S \mid S \subseteq OG \wedge \forall_{o_1 \in S} \forall_{o_2 \in S} PC^*(o_1, o_2) \wedge \forall_{o_1 \in S} \forall_{o_2 \in OG-S} \neg PC(o_1, o_2)\} \quad (1)$$

定义 3：临界区接口的判定，就是看存在冲突风险的接口是否会被同时访问。用 $T(o)$ 表示任务集，这些任务都可以被以 o 为输出接口集的构件所执行，临界区接口 CS 可定义为：

$$CS = \{S \mid S \in PCS \wedge \mid U_{o \in S} T(o) \mid > 1\} \quad (2)$$

如图 4 所示，T_1、T_2 为两个独立的任务，与 T_1、T_2 关联的输出接口有 $o_1 \cdots o_6$，输入接口有 $i_1 \cdots i_4$，它们之间的交互用箭头标出，则有

$PC = \{(o_2, o_3), (o_3, o_4), (o_5, o_6)\}$；

$PCS = \{(o_1), (o_2, o_3, o_4), (o_5, o_6)\}$；$CS = \{(o_2, o_3, o_4)\}$；

PCS 中其他组合都是在单任务中，因此可以排除并发的可能性。

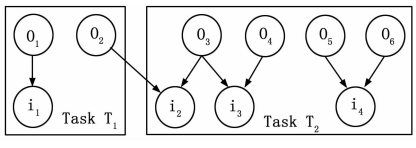

图 4 临界区接口实例

2.2 先进先出锁机制

临界区内的接口数据传输的原子性及交互顺序，我们通过先进先出锁机制(FIFO locking mech— anism)进行保证。先进先出锁机制借鉴了 ICPP[6](immediate ceiling priority protocol)的思想。ICPP 在实时系统中应用广泛，它可以使任务的阻塞时间最小化，还可以有效防止死锁。当任务通过 ICPP 对资源上锁后，该任务会拥有使用该资源的最高优先级(优先级置顶)，从而阻止其它任务获得该资源。

先进先出锁机制采用分层设计的观念，定义了两层优先级：活动优先级与功能优先级(即普通意义上的优先级)[7-8]。当被触发的接口在临界区时，我们定义活动优先级高于功能优先级，确保临界区接口数据传输的原子性与构件间的交互顺序；当被触发的接口不在临界区域时，我们定义活动优先级与功能优先级相同，构件根据功能优先级执行构件。

以数据由输出接口集传入输入接口集(位于临界区)为例，接口数据传输顺序如下：

(1)由于输入接口集位于临界区，任务申请对临界区资源上锁(假设该临界区未被其它任务上锁)；

(2)而后数据传送至相应数据输入接口，等待数据输入；

(3)控制流触发该输入接口集的输入触发接口，数据输入接口将数据读入构件内部；

(4)数据读入完成后，对该区域解锁。

图 5 给出了先进先出锁机制运行过程的描述。最初先进先出锁机制处于空闲状态，当收到任务上锁需求(lock?)后，将该任务标识符存入循环缓冲区(buf)，任务继续执行(continue [locker_ID]：=true 表示)，临界区资源被上锁。如果有其它任务到达(lock?)将会被阻塞(continue[locker_ID]：=false)，任务标识符存入缓冲区。当该任务解锁后(unlock?)，后到任务继续执行(continue [locker_ID]：=true 代替)，若缓存区没有任务(current＝＝storage)，则释放锁，重设内部参数。

接下来我们通过一个实例，给出优先级置顶方法的实现代码。如图 6 所示，复合构件由 Producer、Consumer1、Consumer2 3 个子构件和一个 Data distribute 连接子组成。子构件的触发

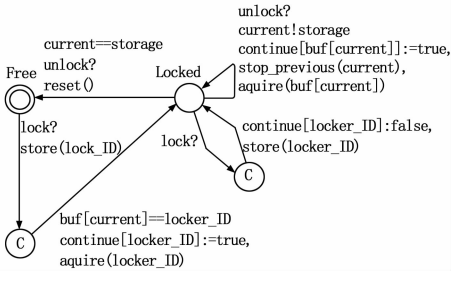

图 5 先进先出锁机制运行过程

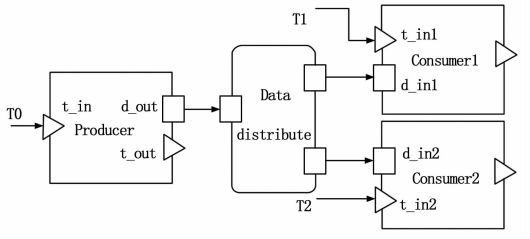

图 6 构件连接实例

由 3 个独立外部事件完成，分别对应 3 个任务 T_0、T_1、T_2。分析可知，图中 Producer 的输出接口集、T_1、T_2 形成了临界区接口。当任务 T1 触发 t_in1 后，该临界区被上锁，代码如下所示：

```
void T1_schedule(Task * tsk)
{
  lock(tsk->port_group);
  if(Consumer1_t_in1(tsk->Consumer))
    T1_todo_list_push(tsk,STATE_Consumer1_t_in1);
  unlock(tsk->port_group1);
  set_functional_priority(tsk);
  while(T1_todo_list_not_empty(tsk))
  {
    STATE_T1current_state_T1_todo_list_pop(tsk);
      switch(current_state)
      {
      case STATE_Consumer1_t_in1:
        subservice_Consumer1(task);
      break
      }
  }
reset_priority(tsk);
}
```

3　实例验证

在这里我们首先给出图 6 所示构件间交互的代码实例，而后以汽车 SCS 子系统为例，给出其 CIES 组装实例。

3.1　构件连接代码实现

对于图 6 所示的构件连接，当任务 T_0 到来时，Producer 构件被触发并开始执行，其输出数据进入临界区，若此时 t_in1、t_in2 接口被触发，则构件 consumer1、consumer2 读入数据并执行。对于给出的代码，我们假设 t_in1、t_in2 接口没有被触发（代码中注释所示），若 consumer1、consumer2 被触发，则代码段中注释位置应变为相应实现代码。

该过程实现代码如下：

```
void T0_schedule(Task * tsk)
{
  if(Producer_trigger_in(tsk->Producer))
    T0_todo_list_push(tsk,STATE_Producer_t_in);
  while(T0_todo_list_not_empty(tsk))
    STATE_TO current_state=T0_todo_list_pop(tsk);
      swithch(current_state)
      }
      case START_Producer_t_in:
      subservice_Producer(task);
    break;
    }
}
void subservice_Producer(Task * tsk)
{
  STATE_Producer state=entry=Producer(tsk->Producer);
  switch(state)
  {
  case STATE_Producer_t_out:
  lock(tsk->port_group1);
  Producer_transfer_d_out(tsk->Producer,
      &tsk->Consumer1->cnx_d_in1);
    Producer_transfer_d_out)tsk->Producer,
      &tsk->Consumer2->cnx_d_in2);
  //没有相关的输入触发接口
  unlock(tsk->port_group1);
  if(Producer_store(tsk->Producer,
          STATE_Producer_out))
  {
    T0_todo_list_push(tsk,STATE_Producer_t_in);
  }
 break;
  }
}
```

3.2　SCS 系统组装实例

我们以汽车电子稳定控制系统（ESC 系统）中的 SCS（stability control system）子系统[9]为例，给出了该子系统的 CIES 构件模型。

如图 7 所示，图中空心圆圈代表 Control join 连接子，实心圆点代表 Control distribute 连接子。SCS 系统有一个 50 Hz 的周期性时钟，首先测得的速度数据由 Wheels speed 接口读入，结合此刻的加速度，actual direction 子构件计算汽车实际方向；由 Steering Wheel angle sensing 构件测得车轮此时角度数据，经过计算，得出汽车潜在行驶方向。而后 Slide detection 构件结合上述数据，做出最终决策，确保汽车稳定性。

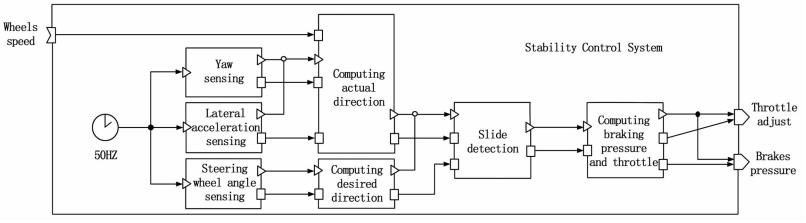

图 7　SCS 组装实例

4　结束语

本文主要论述了 CIES 构件模型的组成及数据交互保障机制。首先给出了 C 模型、连接子的构成，通过触发接口与多种连接子引入，保障接口数据传输原子性与构件间交互时序性。然而，在临界区内上述机制存在缺陷，因此通过先进先出锁机制确保临界区接口数据交互顺序。下一步研究重点工作是 CIES 模型底层服务协议的定义。

参考文献：

[1] Ivica Crnkovic, Severine Sentilles, Aneta Vulgarakis. A Classification Framework for Software Component Models [J]. IEEE Transactions on Software Engineering, 2011, 5 (37): 593 - 615.

[2] Akerholm M, Carlson J, Fredriksson J, et al. The SAVE Approach to Component-based Development of Vechicular Systems (24) [J]. Journal of Systems and Software, 2007, 80 [5]: 655 - 667.

[3] Vulgarakis A, Suryadevara J, Carlson J, et al. Formal semantics of the ProCom real-time component model [A]. In 35th Euromicro Conference on Software Eng- ineering and Advanced Applications (SEA A' 09). IEEE Computer Society [C], 2009: 478 - 485.

（下转第 2334 页）

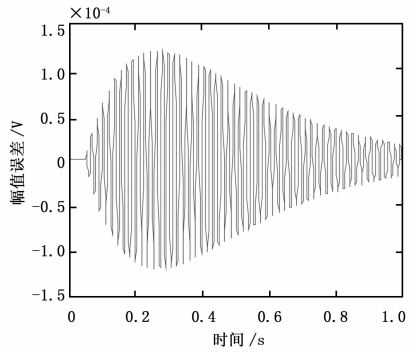

图 4 非线性动态辨识误差曲线图

表 1 两种不同方法对传感器辨识结果对比

采样长度	噪声方差	算法	时间(s)	MAE(V)
20	0.01	SVR	0.263 0	0.044 7
		LSSVM	0.050 0	0.036 5
20	0.02	SVR	0.193 3	0.063 8
		LSSVM	0.020 0	0.033 7
50	0.01	SVR	2.537 0	0.062 1
		LSSVM	0.211 0	0.017 8
50	0.02	SVR	2.421 0	0.411 0
		LSSVM	0.198 0	0.096 3
120	0.01	SVR	69.331 0	0.109 4
		LSSVM	1.245 0	0.031 0
120	0.02	SVR	65.312 0	0.121 5
		LSSVM	1.026 0	0.069 2

从表 1 可以看出，当采样长度较短时，两种算法的耗时相差不大，建模精度 LSSVM 略高于 SVR，但是当样本数量进一步增大时，差距明显增大，当采样长度增大到 12 时，SVR 耗时比 LSSVM 高出将近两个数量级，这是由其算法的机理所决定的，前文已经介绍了 LSSVM 算法相较于 SVR 算法的优势，通过仿真实验进一步验证了这一理论。

5 小结

本文提出了 Wiener 模型传感器非线性系统辨识新方法，将 Wiener 模型的动态线性环节和静态非线性环节分别利用 Laguerre 正交函数和 LSSVM 函数描述。利用 LSSVM 求解 Wiener 模型非线性环节的方法相比于 SVR，只需要求解线性方程组便可获得 LSSVM 参数，省去了 SVR 中较为繁琐的 QP 优化问题，当样本数量比较大时，这一方法的优势非常明显。最后，对实际传感测试系统进行了仿真实验，验证了该方法的可行性，同时，与支持向量回归机（SVR）辨识方法进行了对比，结果表明本文所采用算法辨识精度更高，速度更快，能够有效的辨识非线性动态传感器模型。

参考文献：

[1] 任 好. 汽车发动机相关传感器动态特性的研究 [D]. 合肥：合肥工业大学，2005.

[2] Yu D H，Liu F，Lai P Y. Nonlinear dynamic compensation of sensors using inverse — model — based neural network [J]. IEEE Trans. Instrum. Meas，2008，57 (10)：2364 - 2376.

[3] Wang J S，Chen Y P. A hammerstein recurrent neuro—fuzzy network with an online minimal realization learning algorithm [J]. IEEE Trans. Fuz. Sys，2008，16 (6)：1597 - 1612.

[4] Marconato A，Hu M Q，Boni A. Dynamic compensation of nonlinear sensors by a learning — from — examples approach [J]. IEEE Trans. Instrum. Meas，2008，57 (8)：1689 - 1694.

[5] Xu K J，Wang X F. Identification of sensor block model using volterra series and frequency response function [J]. Measurement，2008 (41)：1135 - 1143.

[6] Cervantes A L，Agamennoni O E，Figueroa J L. A nonlinear model predictive control system based on wiener piecewise linear models [J]. Journal of Process Control，2003，13 (7)：655 - 666.

[7] Norquay S J，Palazoglu A，Romagnoli J A. Application of Wiener model predictive control (WMPC) to a pH neutralization experiment [J]. IEEE Transactions on Control Systems Technology，1999，7 (4)：437 - 445.

[8] Suykens J A K，De Brababter J，Tony Van Gestel . Least Squares Support Vector Machines [M]. World Scientific，2002.

[9] Suykens J A K，De Brababter J，Lukas L et al. Weighted least squares support vector machines：robustness and sparse approximation [J]. Neuro computing，2002，48，1 - 4，85 - 105.

[10] Walberg B. System identification using Laguerre models [J]. IEEE Transactions on Automatic Control，1991，36 (5)：551 - 562.

[11] 朱 照，张志杰. 传感器实时动态补偿方法与误差分析 [J]. 传感器与微系统，2010，29 (6)：68 - 74.

[12] Tötterman S，Toivonen H T. Support vector method for identification of Wiener models [J]. Journal of Process Control，2009，19 (7)：1174 - 1181.

(上接第 2327 页)

[4] 宇天文，刘晓燕，沈嘉权. 实时嵌入式构件模型组装方法及时间性推理 [J]. 计算机工程与应用，2009，45 (25)：74 - 77.

[5] 陈 波. 基于软件体系结构的构件模型和语言研究 [D]. 长沙：国防科学技术大学，2007.

[6] Sha L，Rajkumar R，Lehoczky J. Priority inheritance protocols：An approach to real—time synchronization [J]. IEEE Transac— tions on Computers，1990，39 (9)：1175 - 1185.

[7] Etienne Borde，Jan Carlson. Toward Ve—rified Synthesis of ProCom，a Compone—nt Model for Real—Time Embedded Sy—stems [J]. IEEE Transactions on SoftwareEngineering，2011，20 (24)：129 - 138.

[8] Bonakdarpour B，Bozga M，Jaber M，et al. From high—level component—based modelsto distribute implementation [A]. In ACM International Conference on Embedded Software (EMS — OFT) [C]，2010：209 - 218.

[9] 任艳斐，亓传伟. 基于嵌入式技术的汽车防撞报警系统的设计与应用 [J]. 计算机测量与控制，2014，22 (2)：516 - 518.