·计算机工程与应用·

下一代汽车软件系统运行时环境通信算法的研究

李 允，陈 昊，晏 华

(电子科技大学计算机科学与工程学院 成都 610054)

【摘要】以软件复用、支持分布式功能开发为主要目标的汽车电子下一代架构规定了ECU软件架构和新的开发过程，其中运行时环境是实现上述目标的关键。在研究下一代架构虚拟功能总线通信语义的基础上，结合应用部署的需求，提炼出下一代架构运行时环境的通信问题，即如何统一管理和支持构件在ECU内和ECU间的通信，设计出通信算法。实验结果表明，设计的通信算法能实现虚拟功能总线的全部通信语义。

关 键 词 汽车软件; 通信; 下一代架构; 运行时环境; 虚拟功能总线

中图分类号 TP319 文献标识码 A doi:10.3969/j.issn.1001-0548.2012.03.021

Research on Communication Algorithm of Runtime Environment of Next Generation Automotive Software System

LI Yun, CHEN Hao, and YAN Hua

(School of Computer Science and Engineering, University of Electronic Science and Technology of China Chengdu 610054)

Abstract The next generation automotive electronics architecture defines ECU software architecture and new development process. Runtime environment is critical to implement the above purposes. Based on the research of AUTOSAR virtual functional bus communication semantics, combining the requirements of application deployment, the communication problem of runtime environment in the next generation architecture is proposed and its corresponding algorithms are designed. The experimental results show that the designed communication algorithms implement all the communication semantics of virtual function bus.

Key words automotive software; communication; next generation architecture; runtime environment; virtual function bus

现代汽车电子是一个分布式、异构的环境，电控单元 ECU(electronic control unit)超过 70 个，总线多达 10 条，传统的嵌入式软件开发方法和过程已无法满足分布式汽车嵌入式系统的开发需求。针对新的需求，一些主流的汽车制造商、配件供应商、半导体生产商以及软件工具开发商于 2003 年联合成立了 AUTOSAR(automotive open system ARChitecture)组织，制定了一个便于扩展、定制和移植的开放式系统，即汽车电子下一代架构[1]。该架构的主要目标是定义支持分布式功能开发过程的方法学以及制定 ECU 软件架构标准[1-6]。

运行时环境(runtime environment, RTE)[4]是AUTOSAR架构的核心，是实现AUTOSAR架构下应用软件复用、软件开发与硬件分离的关键，起着承上启下的作用。有了RTE的支持，应用软件在设计时不考虑在哪个ECU上运行及与其他软件通信的具体细节等情况。当应用软件映射到具体的ECU后，RTE负责调度软件，负责软件通信的具体实现方式，负责对基础软件的访问。

由于 AUTOSAR 标准对 RTE 只给出了框架一级的定义，鼓励各软件工具开发商在标准上竞争，对于 RTE 如何实现软件的通信、如何实现调度以及如何访问基础软件并未给出实现算法。文献[7]从 RTE 功能的角度提出 RTE 的模板结构，将 RTE 层分为 5 大模块，即：RTE 主模块、RTE 任务管理模块、RTE 通信管理模块、虚拟功能总线跟踪模块、临界区域管理模块，但对各模块没有进行更深入、具体的设计说明。文献[8]从生成 RTE 层代码的工具角度讨论

收稿日期：2011 – 12 – 05； 修回日期：2012 – 03 – 19
基金项目：国家“核高基”重大专项(2009ZX01038-002-003)；四川省应用基础研究项目(2011JY0118)
作者简介：李允(1971 –)，男，博士，副教授，主要从事嵌入式实时软件方面的研究.

RTE 生成工具的设计和实现；文献[6,9]讨论了AUTOSAR 虚拟功能总线的通信机制；文献[10]提出了 AUTOSAR 软件在集成时产生的时序、性能和互操作等公开问题；文献[11]给出了构件运行体映射时如何优化 ECU 内的数据通信方案；文献[12]分析了现有汽车软件架构与 AUTOSAR 软件架构的不同，并提出迁移方法。少有文献对 RTE 各功能模块给出具体的算法设计。

因此，本文以 RTE 如何实现软件通信的问题作为研究重点，探讨了 RTE 通信管理模块的工作机制和实现算法。实验结果表明，该算法能够支持 AUTOSAR 标准规定的全部通信语义。

1 下一代汽车电子架构基本概念

在下一代汽车电子架构下，软件系统的各个功能部分能够被拆分、重组、迁移和重新部署。RTE是保证在该过程中不需要修改构件的内部行为及其描述的关键。图1给出了下一代架构的基本组成和软件开发过程，即软件设计阶段和软件部署阶段。应用软件的设计是面向虚拟功能总线(VFB)进行的，最小部署单位是构件(swc)，通过端口和通信连接器进行交互。构件中以运行体(runnable entity, RE)为单位组织代码，可设置事件(event)和等待点(wait point)使得运行体被激活。VFB是一种抽象的总线，在软件部署阶段，VFB上的swc被分配到各个ECU上。每个ECU上软件分为应用层、RTE和基础软件(basic software, BSW)3层，RTE是VFB在ECU上的代码实现，它使用ECU上所提供的基础软件为应用软件的通信和运行提供服务。

鉴于下一代架构涉及术语多、关系复杂，下面给出准确的定义。

定义 1 端口是构件与外界通信的接口，以p表示，定义为一个二元关系：

$$p=(\mathrm{DE},\mathrm{Dir})$$

其中DE是端口可通信数据元素集合；Dir是通信方向，从集合DIRECTION={P, R}中取值。P表示发送，R表示接收，如果p[Dir]=p，则称p为一个PPort；否则称p为RPort。

定义 2 访问点是运行体在隐式调用方式下对外通信接口，以ap表示，定义为一个二元关系：

$$\mathrm{ap}=(\mathrm{de}_p,p)$$

其中，de_p表示p[DE]中的一个元素。访问点用于表示描述隐式通信，若p为PPort，则称ap是一个DataWriteAccess，否则称ap为DataReadAccess。运行体RE上访问点的集合表示为$\mathrm{ACCESS}_{\mathrm{re}}$。

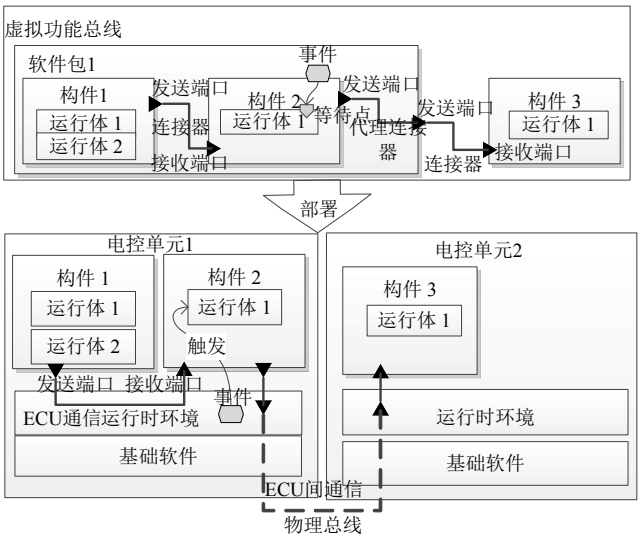

图1 下一代汽车电子架构

定义 3 等待点是激活运行体执行的统一接口，以w表示，定义为一个二元关系：

$$w=(\mathrm{trig},t_{\mathrm{out}})$$

其中，$\mathrm{trig}\in\mathrm{Trigger}_{\mathrm{swc}}$；$\mathrm{Trigger}_{\mathrm{swc}}$是swc上所有触发点的集合。$t_{\mathrm{tout}}$是等待超时时间。运行体RE上等待点集合表示为$\mathrm{WAIT}_{\mathrm{re}}$。

定义 4 运行体是应用算法的最小单位，RTE保证运行体并发执行，以re表示，定义为一个三元关系：

$$\mathrm{re}=(\mathrm{symbol},\mathrm{ACCESS}_{\mathrm{re}},\mathrm{WAIT}_{\mathrm{re}})$$

其中，symbol是用户定义运行体代码的入口符号。

定义 5 事件是触发运行体执行的基本对象，分类为不同事件，以event表示，定义为一个映射：

$$\mathrm{event}:(\mathrm{type},\mathrm{Trigger}_{\mathrm{swc}})\to\mathrm{re}$$

其中，type是事件类型，从集合TYPE={DataSend Complete, DataReceived, DataReceiveError,Timing}中取值。

定义 6 构件是应用的最小构成单元，以swc表示，定义为一个三元关系：

$$\mathrm{swc}=(P,R,E)$$

其中，P、R和E是集合，分别定义为swc上端口集合、运行体集合和事件集合。

swc上的事件集合表示为：

$$\mathrm{EVENT}=\{(\mathrm{type},\mathrm{trigger},r)\,|\,\mathrm{type}\in\mathrm{TYPE},\ \mathrm{trigger}\in\bigcup_{\mathrm{re}\in\mathrm{swc}[R]}\mathrm{re}[\mathrm{WAIT}],\mathrm{re}\in\mathrm{swc}[R]\}$$

定义 7 连接器是构件间端口连接关系，以aconn表示，定义为一个二元关系：

$$\mathrm{aconn}=\{(p_1,p_2)\,|\,p_1\in\mathrm{swc}_1[\mathrm{pport}],\ p_2\in\mathrm{swc}_2[\mathrm{RPort}],\mathrm{swc}_1\neq\mathrm{swc}_2\}$$

其中，swc[PPort]和swc[RPort]分别定义为：

$$\begin{cases}\text{swc[PPort]}=\{p\mid\forall p\in\text{swc}[P]\wedge p[\text{Dir}]=P\}\\ \text{swc[RPort]}=\{p\mid\forall p\in\text{swc}[P]\wedge p[\text{Dir}]=R\}\end{cases}$$

软件包cp上的连接器集合表示为ACONN_{cp}。

定义 8　代理连接器是构件端口与软件包端口连接关系，以dconn表示，定义为一个二元关系：

$$\text{dconn}=\{(p_1,p_2)\,|\,p_1\in\bigcup\text{swc}[p],p_2\in\text{cp}[p],p_1[\text{Dir}]=p_2[\text{Dir}]\}$$

其中cp是一个软件包，其定义见定义 9。软件包cp上的代理连接器集合表示为DCONN_{cp}。

定义 9　软件包是构件的集合体，以cp表示，采用嵌套组合的结构，定义为一个四元关系：

$$\text{cp}=(\text{CP}_{\text{cp}},p_{\text{cp}},\text{ACONN}_{\text{cp}},\text{DCONN}_{\text{cp}})$$

其中，CP_{cp}是软件包内子构件和子包实例所构成的集合：

$$\text{CP}=\{\forall\text{cp}\in\text{SWC}\bigcup\forall\text{cp}\in\text{composition}\}$$

P_{cp}是cp上对外通信端口集合，其定义与swc[P]定义相同。

AUTOSAR标准定义两种特殊的软件包：swcp是全局唯一软件包，包含了系统中所有swc，swcp没有对外端口和代理连接器，因为它独立包含了软件系统的全部功能[10]；top是ECU局部唯一的软件包，它包含被分配到特定ECU上swc。TOP表示系统中所有top_{ECU}的集合。

2　运行时环境的通信问题

如前所述，构件设计时无法预知被部署在什么样的ECU上，因此，使用什么样的通信接口与其他构件进行通信都是未知的。与此同时，系统添加或删除一些应用构件后，余下构件的运行时位置可能发生变化。因此，RTE要解决的一个重要问题是支持构件部署时产生的ECU内通信(intra-ECU communication)和ECU间通信(inter-ECU communication)两种通信类型。

此外，VFB提供的应用软件通信语义包括同步/异步、LIB/Q、显式调用/隐式调用[10]。运行时环境的通信问题还包含对这些语义的支持。

同步/异步：同步通信的双方阻塞等待数据的到达或失败，RTE使用事件通知通信双方。异步通信则不阻塞而使用接口Feedback和Receive检查数据发送接收状况。

LIB/Q：LIB语义表达接收方只存储最后一次到达的数据，Q(Queued)使用队列缓存收到的所有数据。

显式调用/隐式调用：表达通信双方使用函数调用还是句柄(DataHandler)读写方式进行通信。

总之，RTE要无差别地对两种通信类型提供6种语义的支持，即不论通信类型怎样变化，只要设定相同的语义，通信过程和结果就都是一致的。

使用函数g将软件包swcp映射为有向图G：

$$g:\text{swcp}\to G(V,E):V=\bigcup_{\forall\text{swc}\in\text{CP}}\text{swc}[P],E=\text{ACONN}$$

使用函数f将软件包top映射为一幅图G'：

$$f:\text{top}_{\text{ECU}}\to G'(V,E):V=\bigcup_{\forall\text{swc}\in C}\text{swc}[P],E=\text{ACONN}$$

构造加权图$G_{\text{impl}}=(V,E)$，使其顶点集合V为所有$f(\text{top})[V]$的并集，即$V=\bigcup_{t\in\text{TOP}}f(t)[V]$。

由于在ECU提取过程中，没有添加和删除任何swc及其上端口p，由此可证$G_{\text{impl}}[V]$与$g(\text{swcp})[V]$中的元素一一对应。

G_{impl}的边权值从$attr=\{\text{INTER},\text{INTRA}\}$中取值，分别表示为ECU间和ECU内通信。

定义函数comm将$g(\text{swcp})[E]$映射到$G_{\text{impl}}[E]$，映射规则如下：

$\forall(u,v)\in g(\text{swcp})[E]$，在$G_{\text{impl}}[V]$找到与$u$和$v$对应的顶点$u'$和$v'$，若$u'$和$v'$属于相同的$f(\text{top})$，则：

$$G_{\text{impl}}[E]\leftarrow G_{\text{impl}}[E]\bigcup\{(u',v',\text{INTRA})\}$$

否则，有：

$$G_{\text{impl}}[E]\leftarrow G_{\text{impl}}[E]\bigcup\{(u',v',\text{INTER})\}$$

汽车电子下一代构架运行时环境通信问题可抽象为在满足实现VFB所要求的全部通信语义条件下，实现函数comm，完成图G_{impl}边集的构造。

3　通信算法设计

3.1　总体设计

为了实现通信所要求的6种通信语义，设计一组RTE接口。语义与接口对应关系如表1所示。

表1　语义/RTE接口对应关系表

接收	发送	同步/异步	LIB/Q	显/隐式
Rte_Write	Rte_Read	若设置w	LIB	显式
Rte_Send	Rte_Receive	若设置w	Q	显式
Rte_IWrite	Rte_IRead	异步	LIB	隐式

通信代码生成算法分为3个阶段：1) 应用接口的映射，该阶段不生成任何运行时代码，在c语言中使用宏定义完成；2) 静态申请所需要的存储空间，该阶段完成后可以计算所消耗的数据段内存大小；3) 生成实现代码，该阶段生成用于数据收发的代码。

第一阶段应用接口映射是为了让每一个运行体都有构件内的命名空间，通过这个特性，构件在ECU间迁移时不会因为调用了相同的接口而导致静态链接错误。接口映射分为显式调用映射和隐式调用映射两个部分。

对于所有的swc[*p*]，都需要生成其显式调用代码，发送和接收端口所对应的函数调用不同。显式接口映射使用函数指针的方式完成，将用户接口通过宏定义转接为构件结构体实例相应的成员函数指针的调用操作。在第三阶段完成后对结构体内函数指针进行填写。

隐式调用代码映射仅对ACCESS不为空的re进行，它通过宏定义将用户接口转接为构件结构体实例相应的成员变量上的读写操作。RTE会在每次调入运行体之前将结构体实例成员变量填写好。

第二阶段所申请的通信缓冲区是通信算法所必须用到的基本数据结构，对于队列化通信语义，申请线性循环队列缓存接收到的数据，对于非队列化语义，使用独立缓冲变量存放接收到的数据。

3.2 生成实现算法伪代码

第三阶段实现各种语义的发送和接收代码。在接收方，还要注册相应的回调函数，以支持调度器的触发。为了实现Inter-ECU通信的功能，RTE需要注册通信服务(communica tion,COM)回调函数[6]，完成把数据从通信层拷贝到缓冲区，并触发调度器的通信事件。

队列化发送生成算法为：

输入：conn　top_{ECU}中的连接器，attr
输出：conn的队列化发送代码
通过conn，获取发送端构件实例、端口和数据元素，并分别赋值到变量send_cp，send_p和de中。

```
function-name = "Rte_Send_<send_cp>_<send_p>_<de>"
function-argument = "<de.Type> value"
if attr == INTRA then
   find receive_queue for conn[RPort] in total swc
   if receive_queue is full then status = RTE_E_LIMIT
   else
        receive_queue.push_back(value)
        status = RTE_E_OK
   end if
   raise event: “DataReceivedEvent”
else // attr == INTER
      在信号映射中找到de所对应的com_signal_id
      call Com_SendSignal(com_signal_id, &value)
end if
raise event: “DataSendCompleteEvent”
return status
```

非队列化发送生成算法为：

输入：conn　top_{ECU}中的连接器，attr
输出：conn的非队列化发送代码
通过conn，获取发送端构件实例、端口和数据元素，并分别赋值到变量send_cp，send_p和de中。

```
function-name = "Rte_Write_<send_cp>_<send_p>_<de>"
function-argument = "<de.Type> value"
if attr == INTRA then
   find receive_buffer for conn[RPort] in total swc
   copy value to receive_buffer.value
   raise event: “DataReceivedEvent”
   else // attr == INTER
   在信号映射中找到de所对应的com_signal_id
   call Com_SendSignal(com_signal_id, &value)
end if
raise event: “DataSendCompleteEvent”
return status
```

队列化接收生成算法为：

输入：conn　top_{ECU}中的连接器，attr
输出：conn的队列化接收代码
通过conn，获取接收端构件实例、端口和数据元素，并分别赋值到变量receive_cp，receive_p和de中。

```
function-name = "Rte_Receive_<receive_cp>_
<receive_p>_<de>"
function-argument = "<de.Type>* value"
p = conn[p2], c = find receive swc for conn
re = find receive runnable-entity for calling this function
if re has DataReceivePoint of p and
      c has trigger point "DataReceivedEvent" of re then
    event = get DataReceivedEvent of re
    call "Rte_WaitEvent_<receive_p>_<event>" to block receiving
end if
if receive_queue is empty then
    status = "RTE_E_NO_DATA"
else
    if receive_queue is overflow then status = "RTE_E_LOST_DATA"
```

```
    data = receive_queue.pop_front()
    *value = data
end if
return status
```

非队列化接收生成算法为：

输入：conn　top_{ECU}中的连接器，attr
输出：conn的非队列化接收代码
通过conn，获取**接收端**构件实例、端口和数据元素，并分别赋值到变量receive_cp，receive_p和de中。

```
function-name = "Rte_Read_<receive_cp>_<receive_p>_<de>"
p = conn[p2], c = find receive swc for conn
re = find receive runnable-entity for calling this function
if re has DataReceivePoint of p and
      c has trigger point "DataReceivedEvent" of re then
   event = get DataReceivedEvent of re
   call "Rte_WaitEvent_<receive_p>_<event>" to block receiving
end if
*value = Rte_ReceiveBuffer_<receive_cp>_<receive_p>_<de>.value
return status "RTE_E_OK"
```

“数据收到”回调函数生成算法为：

输入：conn　top_{ECU}中的连接器
输出：COM模块的ComNotification回调函数代码
通过conn，获取通信数据元素de。

```
在信号映射中找到de所对应的com_signal_id
function-name = "Rte_ComCbk_< com_signal_id >"
call "Com_ReceiveSignal(com_signal_id, &tmp)" to receive signal
if receive successfully then
   if de.IsQueued == true then receive_queue.
   push_back(tmp)
  else
    copy tmp to receive_buffer
  end if
  raise event: "DataReceivedEvent"
else
  raise event: "DataReceiveErrorEvent"
end if
```

3.3　算法复杂度分析

由于代码生成过程是在软件设计阶段，与运行时性能无关，本文分析生成出的运行时代码复杂度。

全局数据区在系统启动时由Bootloader加载，而RTE并未要求对其进行初始化，因此存储空间申请部分时间复杂度为Θ(1)。空间复杂度与通信变量的数量成线性关系：

$$S(n)=\left|\{\mathrm{de}\mid \mathrm{de[Queue]}=\mathrm{false}\wedge\forall \mathrm{de}\in \mathrm{DCONN}_{\mathrm{ECU}}\}\right|+\left|\{\mathrm{de}\mid \mathrm{de[Queue]}=\mathrm{true}\wedge\forall \mathrm{de}\in \mathrm{DCONN}_{\mathrm{ECU}}\}\right|\times \mathrm{Q\_SIZE}$$

因此复杂度为O(n)。

对于Intra-ECU通信，仅拷贝发送端数据到接收缓冲，对于队列化的情况由于使用线性有界队列，因此通信实现部分时间和空间复杂度均为Θ(1)。对于Inter-ECU部分，需要调用COM模块的代码，并注册相应回调函数。除去COM机制，RTE对系统负载增加部分的算法复杂度为Θ(1)。

在含有等待点和事件的通信模式中，RTE使用OSEK OS事件机制[5]进行运行管理，由于OS采用位图算法，调度时间复杂度为Θ(1)，不会影响接收端算法的复杂度。

4　实验分析

4.1　实验的描述

设计一个应用系统，通过实验结果验证算法的有效性。该应用系统由两个swc组成，每个swc上各有3个端口，每个端口实现一种通信语义，通过连接器一一相连。每个swc内设置3个运行体，分别用来收发数据，如图2所示。

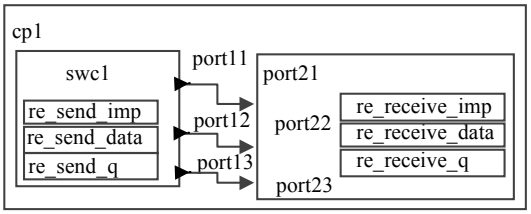

图2　实验应用系统

实验参数配置如表2所示。设计两种类型的部署，第一次将两个swc部署到同一个ECU上，测试通信算法是否正常运行；第二次将swc1部署到ECU1上，swc2部署到ECU2上，测试通信算法是否正常运行。

表2　应用系统参数配置

re symbol	Trigger/ms	api	data
re_send_de1	timing 100	Rte_IWrite	1
re_send_de2	timing 100	Rte_Write	2
re_send_de3	timing 100	Rte_Send	3,4,5
re_receive_de1	timing 100	Rte_IRead	-
re_receive_de2	data receive event	Rte_Read	-
re_receive_de3	timing 300	Rte_Receive	-

4.2　实验的数据分析

通过将程序下载到MPC5634 ECU上运行，并通过串口返回获取数据。

第353次到369次操作的串口反馈数据为：

[353]: [re_send_de1] Rte_IWrite<port11:de1> = 1
== re_send_de3 ==
[354]: Rte_Send<port13:de3> = 3
[355]: Rte_Send<port13:de3> = 4
[356]: Rte_Send<port13:de3> = 5
======
[357]: [re_receive_de1] Rte_IRead<port21:de1> = 1
== re_send_de3 ==
[358]: Rte_Send<port13:de3> = 3
[359]: Rte_Send<port13:de3> = 4
[360]: Rte_Send<port13:de3> = 5
======
[361]: [re_send_de1] Rte_IWrite<port11:de1> = 1
[362]: [re_send_de2] Rte_Write<port12:de2> = 2
[363]: [re_receive_de2] Rte_Read<port22:de2> = 2
== re_receive_de3 ==
[364]: Rte_Receive<port23:de3> = 3
[365]: Rte_Receive<port23:de3> = 4
[366]: Rte_Receive<port23:de3> = 5
[367]: Rte_Receive<port23:de3> = 3
[368]: Rte_Receive<port23:de3> = 4
[369]: Rte_Receive<port23:de3> = 5

操作353和357展示了隐式接口调用和异步语义，数据发送后并未立即激活接收者，而是等待接收者的TimingEvent到期才读取数据。

操作362和363展示了显式接口调用和同步语义，数据发送后立即激活接收者，对数据进行处理。

操作354-356,358-360,364-369展示了队列化数据的发送和接收语义，两次间隔数据发送，每次3个数据，导致接收队列中保存6个数据，然后再依次输出。

5 结 论

运行时环境是支持现代汽车软件系统分布式功能开发和软件复用的关键。因此，研究运行时环境的通信问题具有重大意义。本文在研究汽车电子下一代架构AUTOSAR虚拟功能总线通信语义的基础上，结合应用构件部署的需求，重点提炼出下一代架构运行时环境的通信问题，设计出通信算法且分析算法复杂度为线性。实验结果表明，设计的通信算法能实现虚拟功能总线的全部通信语义。

参 考 文 献

[1] AUTOSAR GbR. AUTOSAR specifications[S]. AUTOSAR Development Partnership, 2008.
[2] AUTOSAR GbR. AUTOSAR software component template[S]. AUTOSAR Development Partnership, 2011.
[3] AUTOSAR GbR. AUTOSAR specification of the system template[S]. AUTOSAR Development Partnership, 2011.
[4] AUTOSAR GbR. AUTOSAR specification of RTE[s]. AUTOSAR Development Partnership, 2011.
[5] AUTOSAR GbR. AUTOSAR specification of operating system[S]. AUTOSAR Development Partnership, 2011.
[6] AUTOSAR GbR. AUTOSAR specification of communication[S]. AUTOSAR Development Partnership, 2011.
[7] JO H C, PIAO S, CHO S R, et al. RTE template structure for AUTOSAR based embedded software[C]//IEEE/ASME International Conference on Mechtronic and Embedded Systems and Applications. Beijing: IEEE, 2008: 233-237.
[8] PIAO S, JO H, JIN S, et al. Design and implementation of RTE generator for automotive embedded software[C]//Seventh ACIS International Conference on Software Engineering Research, Management and Applications. Haikou: IEEE, 2009: 159-165.
[9] WANG Da-fang, LIU Shi-qiang, HUANG Bo, et al. Communication mechanisms on the virtual functional bus of AUTOSAR[C]//International Conference on Intelligent Computation Technology and Automation. Changsha: IEEE, 2010: 982-985.
[10] Razvan Racu, Arne Hamann, Rolf Ernst, et al. Automotive software integration[C]//Design Automation Conference. SauDiego, California: IEEE, 2007.
[11] LONG Rong-shen, LI Hong, WEI Peng, et al. An approach to optimize Intra-ECU communication based on mapping of AUTOSAR runnable entities[C]//International Conference on Embedded Software and Systems. Hangzhou: IEEE, 2009.
[12] KUM D, PARK G, LEE S, et al. AUTOSAR migration from existing automotive software[C]//International Conference on Control, Automation and Systems. Seoul: IEEE, 2008.

编 辑 税 红