基于体系结构模型检查分布式控制系统*

汪 洋[1,2,3+], 魏 峻[1,3], 王振宇[2,3]

[1](中国科学院 软件研究所,北京 100080)

[2](武汉数字工程研究所,湖北 武汉 430074)

[3](武汉大学 软件工程国家重点实验室,湖北 武汉 430072)

Model Checking Distributed Control Systems Based on Software Architecture

WANG Yang[1,2,3+], WEI Jun[1,3], WANG Zhen-Yu[2,3]

[1](Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

[2](Wuhan Digital Engineering Institute, Wuhan 430074, China)

[3](State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, China)

+ Corresponding author: Phn: +86-10-62630989, E-mail: yangwang@public.wh.hb.cn, http://www.iscas.ac.cn

Received 2003-08-13; Accepted 2003-11-27

Wang Y, Wei J, Wang ZY. Model checking distributed control systems based on software architecture. *Journal of Software*, 2004,15(6):823~833.

http://www.jos.org.cn/1000-9825/15/823.htm

Abstract: Distributed control systems are a category of high complex systems that include a large number of devices controlled and harmonized by computer systems. Their reliability and functional correctness always need to be guaranteed as their mission-critical feature. The analysis process for complex control systems consists of proving or verifying that the designed system indeed meets certain specifications. However, both the design and analysis may be formidable due to the complexity and magnitude of the system. From an analysis perspective, the complexity of a system can be reduced by imposing a hierarchical structure and abstraction on the architectural design. Currently, model checking has been demonstrated by more and more successes. It is an effective way to verify that the construction of a complex system satisfies to the requirements of reliability and correctness. In this paper, an approach for formally analyzing distributed control systems at architectural level by applying software architecture description and model checking techniques is presented. Through study on a building comprehensive control system, it is shown that the method could improve the quality of design of distributed control systems.

Key words: correctness requirement; pattern of property specification; model checking; software architecture; distributed control system

* Supported by the National Natural Science Foundation of China under Grant No.60203029 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2001AA113010 (国家高技术研究发展计划(863)); the National Grand Fundamental Research 973 Program of China under Grant No.2002CB312005 (国家重点基础研究发展规划(973))

作者简介: 汪洋(1968－),男,湖北麻城人,博士生,主要研究领域为软件工程,分布式系统,中间件技术;魏峻(1970－),男,博士,副研究员,主要研究领域为软件工程,软件理论,网络分布式计算技术;王振宇(1936－)男,研究员,博士生导师,主要研究领域为软件工程,软件理论与工具开发.

摘 要: 分布控制系统是大量硬件设备通过计算机系统得以控制和协调的高度复杂系统,它们也是任务关键的系统,需要保障其功能的高度正确性和可靠性.分析复杂控制系统的过程包含了证明或验证设计的系统确实满足某种需求.但由于系统的复杂度,有效分析系统是相当困难的.从系统设计和分析的角度看,基于体系结构方法可以运用层次化构造和抽象的方法来减小模型复杂度.模型检查技术是分析复杂系统构造满足正确和可靠性需求的有效方法.结合软件体系结构描述方法和模型检查技术,提出了基于体系结构的分布式控制系统形式分析方法,通过楼宇综合控制系统实例研究,展示了该方法在提高分布式控制系统设计质量方面的效果.

关键词: 正确性需求;性质规范模式;模型检查;软件体系结构;分布式控制系统

中图法分类号: TP311 **文献标识码**: A

大规模分布控制系统[1]例如智能公路控制系统和航空控制系统,包括大量硬件设备,如各种传感器、传动器、电子设备、机械设备等,通过计算机系统得以控制和协调.各种硬件设备、复杂通信协议和控制算法造就了它们组成的是高度复杂的系统.同时,这些系统也是任务关键的,需要保障其功能高度正确和可靠.设计大规模控制系统的过程包括了系统体系结构设计和运用、设计通信和控制算法.而分析复杂控制系统的过程包含了证明或验证设计的系统确实满足某种需求.但是,由于系统复杂度问题,系统的设计和分析是相当困难的.

在设计过程中,可以通过对系统体系结构运用层次化构造和抽象的方法来减小复杂度.运用体系结构方法,既能重用设计成果,又能在高层次关注更抽象的功能,从而提升整个系统开发的效率,并且可能最终使系统实现具有更好的性能.从分析的角度看,减小复杂度是通过聚焦关注的方面进行的.例如,控制系统中组件通信的方式可以多种多样,对组件交互协议建模就会十分复杂,要证明该组件交互满足需求规范就更加复杂.若采用软件体系结构方法,则将组件交互抽象为连接器,分离功能性与非功能性.先通过分析分离结构,再进行组合分析,这样就可以减小分析的复杂性.这也是软件体系结构方法的本质思想.一旦运用软件体系结构方法获得系统抽象,标准分析方法就可以运用于抽象模型.

模型检查[2]是在系统行为的有限状态模型上进行正确性检查的技术.它运用于系统抽象模型,而非实际模型.模型检查的优势在于能自动穷尽地搜索完全状态空间,若检查性质失败,工具会找到导致违反性质的行为.其最大的局限性之一就是状态爆炸问题.一方面,技术层面上应对状态爆炸的有自动抽象、抽象解释、数据归纳与数据独立等技术研究,其目标是减小模型,将无限状态空间向有限状态空间映射;另一方面,则是在系统开发的早期阶段运用模型检查技术.在体系结构设计阶段,系统具体细节还没有涉及,系统状态空间更容易抽象到有限状态空间.因此,在体系结构层次运用模型检查技术是克服方法局限性的有效切入点.同时,在系统设计的早期阶段发现更多设计问题,可以减小后续步骤的开发代价.

为了提高分布式控制系统设计的正确性和可靠性,本文结合软件体系结构描述方法和模型检查技术,提出了基于体系结构的分布式控制系统形式分析方法,通过楼宇综合控制系统实例研究,展示了方法在提高系统设计质量方面的有效性.本文第 1 节介绍了本文工作的相关知识和研究成果.第 2 节给出了结合软件体系结构方法和模型检查技术的形式分析方法.第 3 节展示了基于分析方法的楼宇综合控制系统 BCS 实例研究.最后是结束语,并指出进一步的工作.

1 背 景

1.1 分布式控制系统

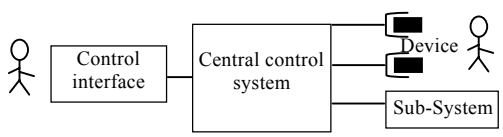


Fig.1 Typical distributed control system
图 1 典型分布式控制系统

控制系统广义上看是一类范畴很广的系统,具备控制单元的系统一般都可以归为此类.但随着计算机技术和网络技术的发展,现代分布式控制系统被认为是一类由硬件设备和计算机控制软件组成的复杂分布式系统.典型的分布式控制系统构成如图 1 所示,包括各种子系统和大量硬件设备,如各种传感器、传动器、电子设备、机械设备等,

各子系统或设备通过计算机软件系统——中央控制系统控制,它们之间也可以各自直接交互与访问.智能高速公路系统、航空控制系统、企业制造控制系统、智能楼宇管理系统、智能寓所控制系统等都是分布式控制系统的例子.本质上,分布式控制系统属于软件密集的分布式系统.

1.2 模型检查技术与工具

近年来,模型检查[2]正逐步发展成为一项系统检查和验证的有力技术.模型检查技术的基本思想是产生系统的有限表示,即系统的一个模型,通过穷尽地搜索有限模型的所有行为,验证系统的关键性质.现在有很多用于并发反应式系统的模型检查工具.著名的有SPIN[3],SMV[4],LTSA[5],FDR等.

模型检查技术包括两个基本方面:模型和性质的形式描述与模型行为验证方法.下面介绍选定的工具.

1.2.1 SPIN/Promela

模型检查器SPIN[3,6]是分析分布式系统逻辑一致性的工具.它基于自动机理论和“on-the-fly”验证技术,用于追踪如操作系统、数据通信协议、交换系统、并发算法等分布式系统设计中的错误.该工具检查规范中的逻辑一致性,报告死锁、未指定的消息接收、标识不完整、竞争状态、进程相对速度未保证等问题.

SPIN进行系统正确性需求检查使用的是完全状态空间搜索方法.其接受的正确性需求可以是系统或进程不变式(断言),或通用线性时序逻辑LTL(linear temporal logic)[7]需求.这些需求或以无“next”算子的LTL公式表示,或以Buchi自动机的形式表示(称为never断言).若某性质非有效,该工具则给出一个出错轨迹.而且,SPIN还能作为有效的模拟执行器调试模型.

SPIN 使用 Promela 作为建模语言.Promela 是一种类 C 的进程建模语言,它提供了描述分布式系统的必须元素,包括消息发送/接收原语、并发进程并行和异步的组合、通信通道等.详细的语言定义参见文献[6].

1.2.2 线性时序逻辑

配套 SPIN 用于描述系统正确性需求的形式系统是线性时序逻辑.在 LTL 中,状态模式被定义用于刻画有限状态系统的所有可能行为.在 SPIN 中用 ASCII 符号来描述 LTL 算子,包括标准命题逻辑连接符&&,||,->,!,还有 3 种时序算子:<>p 表示 p 在未来的某个状态点成立;[]p 表示在未来的所有状态点都成立;二元算子 p∪q 表示 p 在所有状态点都成立直到 q 成立的第 1 个状态点.LTL 可以描述系统的事件和状态如何与时间相关联,以及表达大量复杂的需求.

在 SPIN 模型检查工具中,一般将需要检验的性质以“非”形式表示为“never claim”,即将这类 LTL 公式生成 Buchi 自动机,与系统模型同步执行.若发现行为错误,则说明系统模型不满足 LTL 公式表示的性质,并给出反例的状态转移轨迹.

1.3 相关研究分析与比较

基于软件体系结构进行软件质量分析主要有定性和定量分析两类技术,本文关注基于形式化方法的定性分析方法.

定性分析方法一般基于具有形式基础的体系结构描述语言(architecture description language,简称ADL)进行软件体系结构建模,针对ADL规范描述使用特定的分析方法.Allen和Garlan等人基于CSP[8]语义设计了形式描述语言WRIGHT[9],刻画体系结构元素的连接与交互,使用CSP进程求精方法,检查体系结构元素交互行为兼容匹配,以及系统配置有无死锁;Inverardi,Wolf等人[10]基于化学抽象机(chemical abstract machine)开发的Cham体系结构描述语言,用于刻画和分析各种软件体系结构;Magee,Kramer等人[5,11]开发FSP进程代数描述软件体系结构,使用LTSA分析器进行分布式系统的模拟、可达性分析、安全性和活性检查;Garlan等人[12]基于SMV工具提出一种模型检查Publish/Subscribe系统的框架,由于直接采用SMV作为建模语言,框架中没有显式地描述体系结构元素;Inverardi,Muccini等人[13]基于UML状态图和序列图刻画体系结构各元素的交互,使用SPIN/Promela工具进行软件体系结构规范不完整和视角不一致的分析和验证工作;Issarny,Kloukinas等人[14]扩展UML(主要是结构图和OCL)刻画软件体系结构,使用SPIN作为中间件体系结构组合的分析器,进行了中间件非功能性质的分析.

由于模型检查技术在形式分析方法中具有自动执行、给出反例的优点,因而以上使用模型检查工具如SMV,LSTA,SPIN的方法更为有效.但直接使用模型检查工具的形式建模语言不直观,很难获得正确的形式模型,使得

Magee,Kramer和Garlan等人形式分析体系结构的工作具有一定的局限性.而基于UML可视建模,进行可视模型到形式模型转换,结合SPIN模型检查工具的研究方法[13,14]能成为我们很好的研究基础.不同的是,我们关注研究在体系结构层次分布式控制系统设计的功能正确性.

2 基于软件体系结构方法形式设计分布式控制系统

2.1 SA方法确定分布式控制系统体系结构风格

软件体系结构设计是现代软件系统开发中重要的一部分.各种软件体系结构风格都可以用于开发分布式控制系统.例如,数据流式的管道-过滤器可以用于各种设备的信号转换;数据中心式的数据库或仓库系统等是中央控制系统必须使用的组件;Client/Server风格更是普遍使用.但是,分布式控制系统的中央控制系统必须基于一种主导的体系结构风格来开发,因为系统的核心控制逻辑承载于该核心中间件之中.适用于应用集成的中间件体系结构风格有数据中心、分布对象、事件系统等.选择适合中央控制系统的体系结构风格需要由分布式控制系统的特点来决定[15].

分布式控制系统中包括了各种硬件设备,它们由计算机网络连通和协调,由中心软件系统集中控制,相互操作和访问.支持中央控制系统的软件体系结构,从功能方面考虑,主要包括:

(1) 必须支持灵活智能的组件集成:支撑实现各种设备和子系统之间可裁剪、可定制的交互.

(2) 支持多种访问模式:同步与异步、可靠与弱连接等不同模式,可以应对各种设备的接入访问需求.

从非功能性方面考虑,应该支持:

(1) 可变化性:支持随便增减设备或子系统,支持动态修改设备或子系统功能接口.

(2) 互操作性:支持各子系统或设备的信息的高度可互交换.

分析适用于应用集成中间件的各种体系结构风格,事件系统被认为是既能很好满足分布式控制系统功能性需求,又能满足非功能性需求的一种软件体系结构.事件系统包括了事件管理器核心组件和事件订阅/发布两种应用组件.订阅组件向事件管理器登记感兴趣信息,发布组件向事件管理器发布这些信息.事件管理器的分发模块负责匹配发布与订阅,向感兴趣的订阅者转发信息.通过对组件交互时间和空间的松耦合,基于事件处理的体系结构风格已经被广泛认可,作为大规模复杂系统开发的有效范型.现在它已在各种环境中得以部署应用.如消息通信中间件(MOM)在组件集成和移动计算[16],C2体系结构风格在GUI软件中的应用[17].但是,基于这种范型开发分布式控制系统是特定应用采用专门方法,系统化检查系统正确性需求的方法仍然缺乏.

针对这个问题,下面我们结合形式化描述和体系结构设计方法,运用模型检查技术来提高分布式控制系统设计的质量.

2.2 事件系统的ADL/Promela体系结构模型

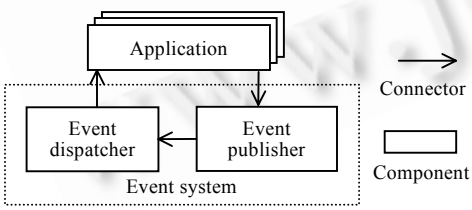


Fig.2 Conceptual model of event-based system
图 2 事件系统概念模型

基于软件体系结构设计方法,需要从逻辑视图勾画系统的概念模型,进而确定主要的子系统、组件和连接器的概念模型.针对本文研究的分布式控制系统中间件体系结构——事件系统,我们给出了其简化概念模型,如图2所示.

事件系统由事件管理和事件分发两个核心部分组成.事件管理器提供事件发布、存储、检索、订阅、注册等功能.而事件分发器主要根据一定事件分发策略,将发布的事件传送给事件订阅者.由于本文主要关心事件系统在进行系统集成和控制时交互方面时间/空间的松耦合特征,因此该概念模型中简化了事件管理器的功能,只关注事件发布.同时,简化事件分发的各种调度和分发策略,采用最原始的单线程同步分发策略.

我们开发了类似于Wright的体系结构描述方法ADL/Promela,描述软件体系结构的各种结构元素,即组件、连接器和配置.为了便于形式分析,其中直接采用Promela语言描述每种结构的行为.组件由主体行为和端口构

成.主体行为BodyBehavior描述组件的功能性行为,端口PortProtocol刻画交互通道的协议.连接器包括Coordination和Role两部分,分别表示独立于角色的交互行为和相关角色的交互行为.下面以事件系统体系结构的描述为例来展示ADL/Promela的语法成分.下表包括了RPC连接器Call、Request连接器Publish、系统组件EventPublisher和EventDispatcher,以及它们的绑定的ADL/Promela描述.

```
Connector Call = {
Coordination =
    proctype Call (chan caller, callee)
   { mtype m ;
    do :: caller ? m -> callee ! m ->
         callee ? m -> caller ! m;
    od };
Role Caller = {
   RoleProtocol =
   inline Caller (caller, m)
   { caller ! m -> caller ? m }
};
Role Callee = {
   RoleProtocol =
   inline Callee (callee, m)
   { callee ? m -> caller ! m}
};
Connector Publish = {
Coordination =
proctype Publish (chan puber, pubee)
{ mtype m ;
   do :: puber ? m -> pubee ! m ->
   od }
Role Pubee = {
   RoleProtocol =
   inline pubee (pubee, m)
   { pubee ? m }
};
Role Puber = {
   RoleProtocol =
   inline puber (puber, m)
   { puber ! m    }
};
```

```
Component EventPublisher = {
BodyBehavior =
   proctype EventPublisher ()
   { chan    pubee[REG], pusher, pbuffer;
    mtype m ;
    C! pusher;
     forall C1[i] ! pubee[i] ;
     do
     :: forall { EP_Pubport(pubee[i], m) ;
        if    :: m != NULL -> pbuffer!m;
        if }
     :: EP_Pushport(pusher, pbuffer);
     od
   }
Port EP_Pubee = {
   PortProtocol =
   inline EP_Pubport (pubee,    m)
   { if :: pubee ? [m] ->    pubee ? m;
    fi
   }
};
Port EP_Pusher = {
   PortProtocol =
   inline EP_Pushport (pusher,    pbuffer)
   { if :: pbuffer ? [m] ->
        pbuffer ? m -> pusher!m;
    fi
   }
};
```

```
Component EventDispatcher = {
BodyBehavior = {
   proctype EventDispatcher ()
   { chan caller[REG], pushee, dbuffer ;
     mtype m ;
     C ! pushee ;
     forall C2[i] ! caller[i] ;
     do
     :: ED_PushPort (pushee, dbuffer);
     :: dbuffer ? m ->
      if :: m==reg[i] ->
            ED_CallPort (caller[i], m);
         ...
      fi;
     od
   }
};
Port ED_Pushee = {
   PortProtocol =
   inline ED_PushPort (pushee, dbuffer)
   { if :: pushee?[m] ->
         pushee?m -> dbuffer ! m;
     fi
   };
};
Port ED_Caller = {
   PortProtocol =
   inline ED_CallPort (caller, m)
   { caller ! m -> caller ? m};
   };
};
```

```
Bindings {
   EventPublisher.EP_Pusher <-> Publish.Puber;        EventDispatcher.ED_Pushee <-> Publish.Pubee;
   EventPublisher.EP_Pubee <-> AppComp i.Puber;     EventDispatcher.ED_Caller <-> AppComp j.Callee;
}
```

2.3 ADL/Promela体系结构模型的形式转换

尽管ADL/Promela体系结构模型中的行为部分可以直接映射为Promela描述,但是完整模型并不能直接成为模型检查工具SPIN可以分析的形式模型.因此,还需要将ADL/Promela规范转换为完整的Promela规范.

通常刻画分布式系统的Promela模型包含了一系列独立进程,它们之间通过消息传递方式,以共享变量或通信通道为渠道进行通信.因此,体系结构元素——组件、连接器、端口、角色也可以像Wright语言一样直接映射为独立进程.但是,这种方式将产生大量进程,不但没有充分利用在体系结构层次可以减小状态空间的优势,反而

引起计算资源的急剧增加.因而,采用如下方法进行ADL/Promela规范到Promela规范的映射:

(1) 组件和连接器实例映射为 Promela 进程结构 Proctype,描述独立执行线程.

(2) 组件端口映射为 Promela inline 结构,描述被各个进程调用的函数.

(3) 连接器的角色在形成完整的 Promela 体系结构模型时,不再显式存在.原因是定义角色行为协议的目的就是用于检查对应端口的通信协议是否被遵守.不像 CSP/FDR,Promela/SPIN 中没有检查进程求精关系的方法,而且,即使端口不遵照角色协议,SPIN 也会检查到其造成的死锁.

(4) 对组件每个端口,在组件 Proctype 中定义一个对应同名 Promela 通道.在配置初始化时,这些通道将以参数方式传给那些有角色绑定到该端口的连接器.

(5) Bindgs 映射为 Promela init 进程,完成组件和连接器之间的绑定.根据 Promela 的计算语义,init 进程通过 run 函数实现组件和连接器即时实例化,使用 Promela 通道将端口和角色联系起来.

(6) 在组件和连接器的 Promela 映射中的一点差异.组件行为定义中以 Chan 结构声明了端口使用的通信通道,但连接器的定义中没有对应的通信通道.它们之间完成绑定是在启动连接器进程时,以参数方式将通信通道传递进去.

基于该方法,转换事件系统 ADL/Promela 规范中的 EventPublisher, EventDispatcher 组件,Call, Publish 连接器都只需直接取相应的行为描述部分,只有 Bindings 部分复杂一些,这里给出其对应的 Init 进程描述:

```
Init {
  Chan Cout, Cin, puber, pubee, caller, callee;
  run EventPublisher();
  run EventDispatcher();
  C?puber;
  C?pubee;
  run Publish(puber, pubee);
  C1[i]?pubee;
  run AppCompi();
  Cout?puber;
  run Publish(puber, pubee);
  C2[j]?caller;
  run AppCompj();
  Cin?callee;
  run Call(caller, callee);
}
```

2.4 基于模式建立分布式控制系统的正确性需求

性质规范模式是对系统有限状态模型中的状态/事件序列发生需求的一个概括性描述.性质规范模式描述了系统行为某些方面的本质特性,以常用形式化描述方法表现这些特性.有了性质规范模式,写关于系统行为的断言就很简单了.实例化断言模式就是直接将符号用明确的命题替换.

Lamport 很早就进行了并发系统的安全性(safety)和活性(liveness)的规范描述.Manna 和 Pnueli[7]从语法结构上对 LTL 公式描述并发系统性质进行了更为细粒度的划分.Dwyer 等人[18]全面、系统地设计了有限状态系统的性质规范模式,从时序作用范围和使用条件等方面,为有效而精确地表示并发反应式系统的性质需求提供了一个有力的表示手段.基于这些结果,针对分布式控制系统正确性需求主要关心全局事件的发生与多事件关联的特点,我们总结了一系列面向事件的性质规范模式.

性质规范模式采用线性时序逻辑(LTL)作为描述机制,以状态公式表示事件发生.与文献[19]中的性质规范模式相比,这里给出的是 LTL 模型检查工具可以检验的时序逻辑公式.

(1) 事件发生(关注事件发生的性质规范模式)

名称	表示	说明
没有发生	[] (!P)	可用于刻画系统不会发生某些不好的事件.
持久发生	[](P -> []P)	该模式描述某个事件发生后促成某个状态保持不变,其使用不同于系统不变式.
无限经常发生	[]<>P	刻画重复出现的事件,但不能描述特定的频率.
最终发生	<>P	某事件最终要发生,发生多少次不关心.
Until	P Until Q	某事件发生使 P 为真直到另一事件发生使 Q 为真.
互斥发生	[]!(P & Q)	关于两个事件不能同时发生的安全条件.
同时发生	[](P->Q) & [](Q->P)	刻画两事件同时发生.
一次发生	P	使用状态公式描述某事件在某时刻发生.一般在模型中直接使用,如 Promela 模型中使用 assert(p).

(2) 事件关联(关注事件之间关联的性质规范模式)

名称	表示	说明
因果	[] (P -> <>Q)	某事件发生(P)引起另一事件的发生(Q).因果模式形式上与文献[1,2]中的 response 模式一致,但它强调事件的关联性,而不只是事件的发生顺序.
单因多果	[](P -> (<>Q & <>R)	这是多个结果非同时发生的形式,是[](P-><>Q)/\[](P-><>R)的简化形式.多个结果同时发生的模式是[](P-><>(Q&R)).
多因单果	[](P & Q -> R)	这里是多个原因同时发生的形式,而原因非同时发生的情况很难刻画.P 与 Q 都能引起 R 发生,但两者关系不能确定.可以采用分而治之的方法,用两个因果模式表示:[](P-><>R)/\[](Q-><>R).

(3) 其他

这一类主要是关于系统全局状态和系统模型正确性的需求.

名称	表示	说明
不变式	[] P	刻画系统或进程模型生命周期不变的状态.这里,状态公式 P 不是针对某个事件,而是关于模型的性质.
无死锁	无	系统无死锁在不同验证工具中有不同的检查方法,无法使用 LTL 或 CTL 表示通用的无死锁需求.例如:SPIN 通过检查模型的无效结束状态(invalid end-state)来发现死锁.
无活锁	无	同样无活锁没有通用的需求规范描述.在 SPIN 中,通过检查无限循环和无进展(non-progress)来发现活锁.

3 分布式控制系统实例 BCS 的形式分析

楼宇综合控制系统(building controlling system,简称 BCS)是典型软件密集的分布式控制系统.它作为楼宇的中枢神经,调控智能大厦内各子系统之间互操作、快速响应与联动控制[20].通过中心控制系统,实现了:(1) 集中监视各子系统,特别是现场信息的监视;(2) 集中控制各子系统,特别是现场设备的控制;将控制信息传送给相应子系统的控制管理器,由各子系统完成对最终设备的控制;(3) 综合管理各子系统.通过分析各个子系统之间在内的联系,对相应信息进行综合、加工、处理,完成对大楼内综合信息的集成.

楼宇综合管理系统 BCS 由 BCS 控制中心、楼宇设备控制系统(building equipment administrating subsystem,简称 BAS)、火灾报警系统(fire alarming subsystem,简称 FAS)、保安系统(security assurance subsystem,简称 SAS)、门禁系统(door controlling subsystem,简称 DCS)等主要功能组件和事件系统中间件构成.BCS 控制中心组件基于该中间件而开发,而其他应用组件(子系统)都是该中间件的应用客户端.系统体系结构如图 3 所示.

3.1 BCS系统体系结构的形式模型

对应于如图 3 所示的 BCS 系统结构,采用软件体系结构设计方法,将系统组件和应用组件通过组件交互集成在一起形成一个完整的 BCS 体系结构配置,如图 4 所示.这里,由于 BCS 控制中心基于事件系统开发,其控制管理逻辑体现在事件系统之中,控制中心因而成为系统组件的一部分.4 个子系统分别通过 CALL 和 PUBLISH 连接器与事件系统交互.

3.1.1 应用组件

4 个应用组件 BAS,SAS,DCS,FAS 基于行为状态机模型来模拟楼宇设备控制系统、保安系统、门禁系统、

火灾报警系统的事件交互行为.

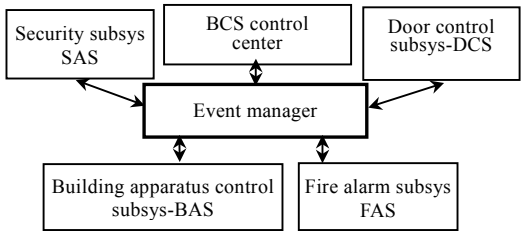


Fig.3 BCS system structure

图 3 BCS 系统结构

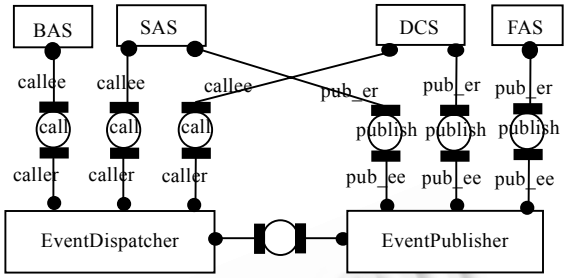
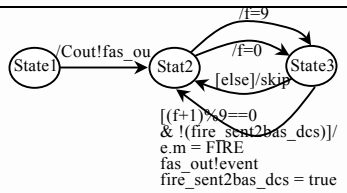
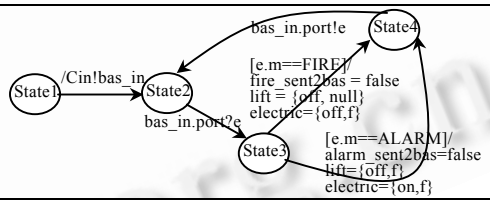
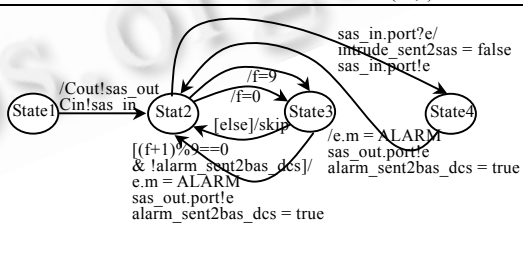
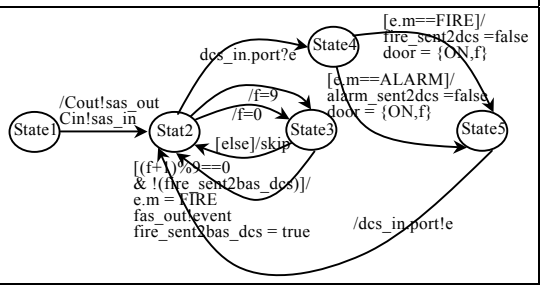


Fig.4 Architectural configuration of BCS

图 4 BCS 体系结构配置

行为状态机 BSM=⟨S_0,S,T,L⟩.S_0 是初始状态集合,S 是状态集合,L 是状态转移标记,形式为 $E[C]/A \in L$,$T \subseteq S \times L \times S$ 是状态转移集合.状态转移标记 $E[C]/A$ 的 E 代表触发状态转移的事件(动作),一般为 chanA?e 接收某个事件;C 表示约束转移分支的条件,是基于模型变量的布尔表达式;A 表示转移发生进行的操作,主要是发送消息 chanA!e 和内部赋值操作.转移标记中的 E,C 可选存在.

下表右列对应于应用组件的行为状态机模型.将每个行为状态机对应于一个 Promela Proctype 结构,产生组件的 Promela 模型是相当直接的.

(1) 楼宇设备控制系统 BAS BAS 模型设计为只订阅事件,通过 bas_in 通道使用 Call 连接器与 EventDispatcher 连接.BAS 接收 FIRE 和 ALARM 事件,并会激活自身相应方法采取行动.	
(2) 火灾报警系统 FAS FAS 模型设计为只随机发布 FIRE 事件,通过 fas_out 通道使用 Publish 连接器,pub 与 EventPublisher 连接.	
(3) 保安系统 SAS SAS 模型设计为一方面模拟产生并发布 ALARM 事件,另一方面接收控制中心的门禁非法入侵事件,并向控制中心发布 ALARM 事件.SAS 通过 sas_in 通道,使用 call 连接器与 EventDispatcher 连接;通过 sas_out 通道,使用 Publish 连接器与 EventPublisher 连接.	
(4) 门禁系统 DCS DCS 模型设计为一方面模拟产生并发布 INTRUDE 事件,另一方面接收控制中心传来的 ALARM 和 FIRE 事件.DCS 通过 dcs_in 通道,使用 Call 连接器与 EventDispatcher 连接;通过 dcs_out 通道,使用 Publish 连接器与 EventPublisher 连接.	

3.1.2 系统组件、连接器和系统配置

BCS 系统体系结构模型中使用了两个系统组件 EventPublisher 和 EventDispatcher,连接 BAS,SAS,DCS 与

EventDispatcher 的 3 个 Call 连接器,连接 FAS, SAS, DCS 与 EventPublisher 的 3 个 Publish 连接器和一个连接 EventDispatcher 与 EventPublisher 的 Publish 连接器.这些系统组件和连接器的 Promela 模型在第 2.2 节已给出.

BCS 系统体系结构模型中的绑定具体化了 FAS,SAS,DCS,BAS 这 4 个应用组件,相应地,Promela init 进程与第 2.2 节的抽象描述相比,不过是将 AppComp*i* 分别具体化为 FASComp,SASComp,DCSComp,AppComp*j* 具体化为 BASComp,SASComp,DCSComp,细节不再赘述.

3.2 BCS系统的正确性需求

由于我们只关心 BCS 系统全局的功能正确性需求,即各子系统协调联动的正确性,而不关心 BCS 的子系统和设备的正确性,根据第 3.1 节 BCS 系统的体系结构模型,我们确定出 3 个全局关键事件:FIRE,ALARM, INTRUDE.这 3 个全局事件影响 4 个子系统之间的互相作用:

(1) 火灾报警系统↔楼宇自控系统

发生火灾报警时,关闭电梯使用,切断相应楼层供电.

(2) 火灾报警系统↔门禁管理系统

发生火灾报警时,门禁管理系统中与火警部位有关的各管制门应自动处于开启状态,以便内部人员疏散和火灾人员进入.

(3) 综合保安系统↔楼宇自控系统

当保安系统报警时,通知 BAS 电梯控制系统,使电梯不停靠报警层,闭锁楼层通道门,同时开启照明以便自动摄录现场.

(4) 门禁管理系统↔楼宇自控系统

当门禁管理系统有非法入侵时,楼宇自控系统将照明开启,以便自动摄录现场.

根据第 2.3 节的面向事件的性质规范模式和这些子系统的相互作用,我们提出了如下的性质需求:

p∪q	性质需求	性质规范	定义说明
P1	电梯运行直到发生火灾	LiftRunning∪Fired	#define Fired (Fired(0)‖…Fired(9)) #define Fired(i) (fire.status==true && fire.flr==i) #define LiftRunning (lift.status==ON)
P2	保持供电直到发生火灾	PowerOn(i)∪Fired(i)	#define PowerOn(i) (electric.status ==ON && electric.flr==i)
因果	性质需求	性质规范	定义说明
P3	发生火灾报警 leads-to 关闭电梯	[](Fired-><>LiftStopped)	#define LiftStopped (lift.status==OFF && lift.flr==FFFF)
P4	发生火灾报警 leads-to 置管制门于开启状态	[](Fired(i)-><>DoorOpen(i))	#define DoorOpen(i) (door.status==ON && door.flr==i)
P5	发生火灾报警 leads-to 关相应楼层照明	[](Fired(i)-><>PowerOff(i))	#define PowerOff(i) (electric.status == OFF && electric.flr==i)
P6	发生保安报警 leads-to 置管制门于关闭状态	[](Alarm(i)-><>Doorcls(i))	#define Alarm(i) (alarm.status==true && alarm.flr=i) #define Doorcls(i) (door.status==OFF && door.flr==i)
P7	发生保安报警 leads-to 电梯不停靠相应楼层	[](Alarm(i)-><>Liftnstp(i))	#define Liftnsp(i) (lift.status==OFF && lift.flr==i)
P8	发生保安报警 leads-to 相应楼层照明开启	[](Alarm(i)-><>PowerOn(i))	
P9	发生门禁入侵 leads-to 产生保安报警	[](Intruded(i)-><> Alarm(i))	#define Intruded(i) (intrude.flag == true && intrude.flr==i)
P10	发生门禁入侵 leads-to 系列同保安报警的动作	[](Intruded(i)-><> Doorcls(i) or PowerOn(i) or Liftnstp(i))	这里不再细述

还可以定义单因多果的性质规范.如将 P3,P4,P5 组成“发生火灾报警 leads-to 关闭电梯、置管制门于开启状态、关相应楼层照明”的复合因果性质.类似地,可以构造保安报警和门禁入侵的单因多果的性质规范.

3.3 模型检查结果和分析

将 BCS 系统的完整 Promela 模型输入 SPIN 模型检查器,进行正确性需求验证.所有的模型检查使用 SPIN Ver.4.04,在 Pentium III 800,内存 512M 的 PC 机上进行.

我们首先进行模型正确性的检查.通过穷尽式验证,寻找全局模型中的非有效终止状态和不进展状态,排除模型由于大量通信交互和使用共享变量造成的死锁和活锁.同时剔除应用组件中造成非可达状态的冗余代码.在模型正确的情况下,我们检查 BCS 系统模型是否满足选定的性质需求,结果如下:

(1) 两个 p∪q 性质 P1 和 P2 都满足;

(2) 因果性质 P9 满足;

(3) 因果性质 P3~P8,P10 都不满足.

3.4 分析与讨论

分析违反性质 P5 的反例,发现火灾事件发生后的状态序列中有与“关闭楼层照明”动作相反的操作,即开启照明电路,这是保安报警事件后 BAS 系统采取的操作.对比性质 P5 与 P8,两个“effects”是对立的.这样的情况同样出现在 P3 与 P7,P4 与 P6 之间.而 P10 不满足是因为门禁入侵引起保安报警事件,等同于 P6,P7,P8.

不同的全局事件对同一组件的同类操作带来不同、甚至相反的影响,这类似于通信领域中的“特征影响(feature interaction)”问题.我们试着将火灾报警系统 FAS 从 BCS 系统模型中移出,再次验证性质 P6~P8,P10 可满足.同样的方法也可让 P3~P5 满足.这种方法是通过消除影响对方的“特征”或服务来使部分正确性需求得以满足的,并不是根本的办法.

彻底的解决方法应该是协调矛盾事件.由于事件系统中事件处理的不确定性,解决方法可以将事件根据重要性分级,事件产生影响取决于重要级高的事件效果.例如,在 BCS 系统中,FIRE 事件的重要性(显然)高于 ALARM 事件,所以,对于 BAS 子系统中来说,无论 ALARM 事件到来的先后顺序如何,只要发生 FIRE 事件,对 BAS 子系统的影响取决于 FIRE 事件带来的影响.使用这种解决方案改进模型后,重新检查性质 P3~P5 是可以满足的,但 P6~P8,P10 则由于事件重要级别低的缘故,只有加入“没有火灾发生”的前提约束才能满足它们的“effect”.

4 总结与将来工作

面对分布式控制系统设计和分析的高度复杂性,为了提高分布式控制系统设计的正确性和可靠性,本文基于软件体系结构描述和形式建模方法,在体系结构层次对复杂系统进行模型构造和正确性需求获取,运用模型检查技术对分布式控制系统进行严格的分析和检查,通过实例研究楼宇综合控制系统,说明了这种基于软件体系结构的复杂系统分析方法能够提高分布控制系统的设计质量.

进一步的工作包括研究该方法形成一个具体框架,集成 SPIN 开发出分析环境,以及研究事件系统可定制的分发和调度策略对分布式控制系统功能性和非功能性的影响.

References:

[1] Pappas GJ, Sastry S. Towards continuous abstractions of dynamical and control systems. In: Antsaklis P, Kohn W, Nerode A, Sastry S, eds. Hybrid Systems IV. Lecture Notes in Computer Science 1273, New York: Springer-Verlag, 1997. 329~341.

[2] Clarke E, Grumberg O, Peled D. Model-Checking. MIT Press, 1999.

[3] Holzmann GJ. The spin model checker. IEEE Trans. on Software Engineering, 1997,23(5):279~295.

[4] McMillan K. Symbolic Model Checking. Boston: Kluwer Academic Publishers, 1993.

[5] Magee J, Kramer J. Concurrency: State Models & Java Programs. Indianapolis: John Wiley & Sons, 1999.

[6] Holzmann GJ. Design and Validation of Computer Protocols. Englewood Cliffs: Prentice-Hall, 1991.

[7] Manna Z, Pnueli A. The Temporal Logic of Reactive and Concurrent Systems: Specification. New York: Springer-Verlag, 1991.

[8] Hoare CAR. Communicating Sequential Processes. Englewood Cliffs: Prentice-Hall, 1985.

[9] Allen R, Garlan D. A formal basis for architectural connection. ACM Trans. on Software Engineering and Methodology, 1997, 6(3):213~249.

[10] Inverardi P, Wolf AL. Formal specifications and analysis of software architectures using the chemical abstract machine model. IEEE Trans. on Software Engineering, 1995,21(4):100~114.

[11] Magee J, Kramer J, Giannakopoulou D. Behaviour analysis of software architectures. In: Donohoe P, ed. Proc. of 1st Working IFIP Conf. on Software Architecture (WICSA1). Boston: Kluwer Academic Publishers, 1999. 35~50

[12] Garlan D, Khersonsky S, Kim JS. Model checking publish-subscribe systems. In: Ball T, Rajamani SK, eds. Proc. of the 10th SPIN Workshop: Model Checking Software. Heidelberg: Springer-Verlag, 2003. 166~180.

[13] Inverardi P, Muccini H, Pelliccione P. Automated check of architectural models consistency using SPIN. In: Feather M, Goedicke M, eds. Proc. of the 16th IEEE Int'l Conf. on Automated Software Engineering (ASE 2001). Los Alamitos: IEEE Computer Society Press, 2001. 346~349.

[14] Issarny V, Kloukinas C, Zarras A. Systematic aid in the development of middleware architectures. Communications of the ACM, 2002, 45(6):53~58.

[15] Bachmann F, Bass L, Chastek G, Donohoe P, Peruzzi F. The architecture based design method. Technical Report, CMU/SEI-2000-TR-001, Pittsburgh: Software Engineering Institute, Carnegie Mellon University, 2000.

[16] Eugster P, Felber PA, Guerraoui R, Kermarrec AM. The many faces of publish/subscribe. ACM Computing Surveys, 2003,35(2): 114~131.

[17] Taylor RN, Medvidovic N, Anderson KM, Whitehead Jr. EJ, Robbins JE, Nies KA, Oreizy P, Dubrow DL. A component- and message-based architectural style for GUI software. IEEE Trans. on Software Engineering, 1996,22(6):390~406.

[18] Dwyer MB, Avrunin GS, Corbett JC. Patterns in property specifications for finite-state verification. In: Proc. of the 21st Int'l Conf. on Software Engineering. Los Alamitos: IEEE Computer Society Press, 1999. 411~420.

[19] Comella-Dorda S, Gluch D, Hudak J, Lewis G, Weinstock C. Model-Based verification: Claim creation guidelines. Technical Note, CMU/SEI-2001-TN018, Pittsburgh: Software Engineering Institute, Carnegie Mellon University, 2001.

[20] Wang Y, Wang ZY. Design and implementation of intelligent building management system based on CORBA. Computer and Digital Engineering, 2001,29(2):16~22 (in Chinese with English abstract).

附中文参考文献:

[20] 汪洋,王振宇.基于 CORBA 的智能建筑管理系统 IBMS 的设计与实现.计算机与数字工程,2001,29(2):16~22.