文章编号: 1000-6788(2014)07-1857-11 中图分类号: TP301 文献标志码: A

信息物理融合系统可信软件形式化建模与分析

于振华 [1,2], 蔡远利 [3], 付 晓 [1,2], 谢文军 [1], 徐海平 [4]

(1. 空军工程大学 信息与导航学院, 西安 710077; 2. 飞行器控制一体化技术重点实验室, 西安 710065; 3. 西安交通大学 电子与信息工程学院, 西安 710049; 4. Department of Computer and Information Science, University of Massachusetts Dartmouth, North Dartmouth 02747)

摘 要 从多 Agent 系统的角度, 以 Petri 网和 π 演算为语义基础, 建立了一种信息物理融合系统 (cyber-physical systems, CPS) 可信软件形式化模型 (high-confidence software formal model, HCSFM). HCSFM 以 Petri 网形象地描述 CPS 可信软件静态结构模型及动态行为, 用 Petri 网分析方法和支持工具对模型进行分析和验证; 利用 π 演算刻画 CPS 可信软件中 Agent 的加入、退出、更新和体系结构重配置等动态演化机制, 并研究 Agent 的演化策略及演化后 CPS 的一致性, 确保动态演化后 CPS 软件能正常交互, 从而为 CPS 软件设计提供可信保障. 通过 HCSFM 在无人驾驶车辆编队 CPS 中的应用, 表明 HCSFM 可以有效地对 CPS 可信软件进行建模和分析.

关键词 信息物理融合系统; 可信软件; Petri 网; π 演算; 建模; 演化

Formal modeling and analyzing high-confidence software of cyber-physical systems

YU Zhen-hua[1,2], CAI Yuan-li[3], FU Xiao[1,2], XIE Wen-jun[1], XU Hai-ping[4]

(1. School of Information and Navigation, Air Force Engineering University, Xi'an 710077, China; 2. Science and Technology on Aircraft Control Laboratory, Xi'an 710065, China; 3. School of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an 710049, China; 4. Department of Computer and Information Science, University of Massachusetts Dartmouth, North Dartmouth 02747, USA)

Abstract From the perspective of multi-agent systems, a high-confidence software formal model (HCSFM) of cyber-physical systems (CPS) based on two complementary formalisms, namely Petri nets and π-calculus, is proposed. Petri nets are employed to visualize the architecture and model the behaviors of CPS software, and the structural analysis techniques allow the qualitative analysis of properties that may be proved directly on the structure of Petri nets. π-calculus is used to describe CPS software evolution, including agent joining, exiting, updating, and architecture reconfiguration. The evolving strategy of agents and the consistency of CPS software can also be analyzed using π-calculus. HCSFM will improve the dependability of CPS software. HCSFM is applied to unmanned ground vehicles CPS, which shows that it can effectively describe and analyze the high-confidence software of cyber-physical systems.

Keywords cyber-physical systems; high-confidence software; Petri nets; π-calculus; model; evolve

1 引言

信息物理融合系统 (cyber-physical system, CPS) 是一种集成计算、通信与控制功能的新型智能系统, 系统中计算进程和物理进程在开放环境下持续交互、深度融合, 从而实现大规模物理系统的实时感知、远程精确控制和信息服务 [1−3]. CPS 可广泛应用于国防、航空航天、智能交通、智能电网、医疗设备、分布式机器人等领域 [4−7]. 如同 Internet 改变了人与人交互的方式一样, CPS 的出现将改变人与物理世界交互的方式 [6]. CPS 包含传感器节点 (sensors)、执行器节点 (actuators) 和控制器节点 (controller), 各节点通过有线或无线网络联接, 能实现节点之间的高效协作、精确控制、资源的动态组织与协调分配、体系结构重配置等 [3,8].

收稿日期: 2012-09-07
资助项目: 国家自然科学基金 (61202128); 航空科学基金 (20100796004, 20125896020); 陕西省自然科学基金 (2011JQ8011)
作者简介: 于振华 (1977–), 男, 汉, 山东乳山人, 副教授, 博士, 研究方向: 信息物理融合系统, 可信软件, E-mail: zhenhua_yu@163.com.

软件是 CPS 的灵魂, 在 CPS 中所占的比重越来越高, 对系统的影响也越来越大. CPS 主要用于安全关键系统中, 生存环境恶劣, 极易遭受干扰、外部攻击或恶意代码侵蚀; 随着 CPS 功能需求的不断增加, 软件系统变得日趋庞大和难以驾驭, 缺陷和漏洞难以避免, 系统也越来越脆弱, 一旦发生故障或失效, 将会给国防安全和生命财产等方面带来巨大损失. 因此, CPS 软件必须是可信的, 软件的可信性对 CPS 的安全性和有效性至关重要 [9], 我们亟需推动 CPS 可信软件基础理论与关键技术研究.

CPS 是基于计算进程与物理进程深度融合的新型计算模式, 其软件体系结构经常随需求和环境的变化而进行动态调整与重构, 对其软件可信性研究的难点在于如何精确地描述、分析和验证系统的动态演化、离散事件部分与连续时间部分及其交互 [10−11], 从而尽早发现需求和设计中的不一致、歧义和不完备等情况. 传统的计算机模拟和测试费效比较高, 而且不能够完全验证系统的正确性和可靠性 [12]. 形式化方法现已成为提高软件可信性的重要手段, 它在保证软件的正确性、揭示软件设计的本质特征和一般规律方面具有重要的意义 [9,13−14]. 因此, 为了提高软件可信性, 需要采用形式化方法对 CPS 软件进行描述、分析和验证, 从而减少在系统早期设计阶段潜在的错误, 降低开发成本, 有效提升系统的正确性、可用性、安全性和可靠性等可信属性, 这对于高可信 CPS 的建设和发展具有十分重要的理论和实际意义.

近年来, 模型驱动架构 (model driven architecture, MDA)[15] 作为一种提高软件质量的方法在复杂嵌入式系统领域得到了广泛应用. 在 MDA 中, 通过建立规范的高抽象层次模型, 将软件需求描述与特定平台的实现分离, 从而适应不断变化的需求. OMG 推荐 UML2 作为 MDA 的建模语言, 但 UML2 为半形式化语言, 不能充分对模型进行分析和验证 [9]. 目前, 在 CPS 软件研究中也广泛采用 MDA 框架 [9,16], 并采用不同形式化方法建立了相应的形式化模型. Lee 等 [9] 和 Jiang 等 [16] 在 MDA 框架下, 都利用时间自动机描述医疗 CPS, 通过模型检测工具 UPPAAL 对模型进行分析和验证, Lee 最后通过工具 TIMES 进行代码自动生成, 有效提高了医疗 CPS 软件的可信性; Bruce 等 [17] 利用进程代数 SPA 对 CPS 进行建模和分析, 然后利用模型检测技术验证系统的 BNDC 属性; Platzer 等 [18] 利用微分动态逻辑描述 CPS, 开发了 KeYmaera 工具验证系统相关性质; Hunt[19] 利用 ACL2r 逻辑描述 CPS, 然后利用定理证明验证系统的安全性质; Zhang 等 [20] 使用时态逻辑来描述和分析 CPS 软件的时间约束, 并采用模型检测来验证系统的性质; Bujorianu 等 [21] 利用 Hilbertean 方法建立了 CPS 的形式化框架, 并研究了面向用户控制的 CPS 逻辑描述和动态性质分析; Susuki 等 [22] 采用混合自动机对基于 CPS 的电网进行建模, 然后对模型进行可达性分析, 并研究系统的稳定性; Thacker[23] 利用一种扩展标记混合 Petri 网对 CPS 进行建模, 使用离散变量表示软件变量, 并增加一种表达式对数学运算进行描述, 然后提出一种快速搜索系统状态空间的方法对模型进行分析; 范贵生等 [24] 基于 Petri 网建立了 CPS 软件形式化模型, 对设备、计算与物理交互、构件及通信过程等要素进行建模, 并对系统的可靠性保障策略进行研究. 综上所述, 虽然现有方法可以为 CPS 软件提供形式化描述, 但没有全面对 CPS 软件的静态结构、动态结构和行为进行建模, 尤其缺乏对软件的动态结构建模, 而且对模型的分析和验证能力不足, 缺乏全面、系统的模型分析和验证方法, 如分析模型的死锁性、有界性、可达性、并发性、互斥性及演化后系统一致性等, 因此 CPS 软件可信性不能得到有效保障.

针对上述问题, 本文集成 Petri 网 [25] 和 π 演算 [26] 等两种形式化方法, 在模型驱动架构 MDA 下, 建立 CPS 可信软件形式化模型. 首先利用 Petri 网形象地描述系统的静态结构模型和动态行为, 同时 Petri 网所具有的多种数学分析方法和支持工具又能对系统模型进行分析和验证; 当系统发生演化时, 利用 π 演算对系统动态演化进行建模并分析动态演化后系统的一致性, 确保系统的正常交互, 从而为 CPS 软件设计提供可信保障. 本文将为 CPS 可信软件的设计与开发提供理论依据和技术手段, 进而推进其在各个领域中的应用.

2 信息物理融合系统可信软件形式化建模研究

CPS 由传感器、执行器和控制器等组成, 属于复杂的嵌入式系统, 并具有自治性、异构性、并发性等特点. 复杂的嵌入式系统一般由多个计算子系统构造而成, 具有较高的构件化特征 [27], 目前构件化软件开发技术已成为嵌入式软件开发的发展趋势. 为了提高 CPS 软件的自治性, 本文首先将 CPS 软件中各子系统的构件作为一个具有特定信念 (belief)、愿望 (desire) 和意图 (intention) 的 Agent, 然后根据我们建立的多 Agent 系统体系结构模型 [28], 在 MDA 框架下, 利用 Petri 网和 π 演算描述系统中各 Agent 的结构和行为及其动态演化, 建立 CPS 可信软件形式化模型 (high-confidence software formal model, HCSFM), 从而为 CPS 可信软件开发提供具有高抽象层次、独立于执行平台的系统模型.

CPS 可信软件形式化模型由 CPS 可信软件体系结构模型和动态演化模型构成. 下面分别进行研究.

2.1 CPS 可信软件体系结构模型

定义 1 CPS 可信软件体系结构模型 (high-confidence software architecture model, HCSAM) 是一个三元组, $HCSAM = (AComp, AConn, AConf)$, 其中 $AComp = (AComp_1, AComp_2, \cdots, AComp_o)$ 表示计算 Agent 集合, $AConn = (AConn_1, AConn_2, \cdots, AConn_p)$ 表示连接 Agent 集合, $AConf$ 表示体系结构配置.

1) 计算 Agent($AComp$)

计算 Agent 是一个数据单元或一个计算单元, 是具有扩展性和集成性的计算或状态存储的场所. 根据功能的不同, 计算 Agent 可分为反应 Agent 和混合 Agent.

定义 2 反应 Agent 是八元组, $ReaAComp = (P, P_r, T, IT, OT, F, W, \Pi)$, 其中:

① 库所 P_r 是抽象库所, 用椭圆表示, 表示 "条件/动作" 规则库, 用 IF-THEN 规则表示, 可以根据系统需求进行求精;

② 变迁 IT 和 OT 分别表示接口变迁, IT 表示反应 Agent 的输入接口, OT 表示反应 Agent 的输出接口; Π 利用 π 演算描述反应 Agent 的演化机制.

反应 Agent 模型如图 1 所示. 反应 Agent 可以将感知与动作连接起来, 能及时而快速地响应外来信息和环境的变化.

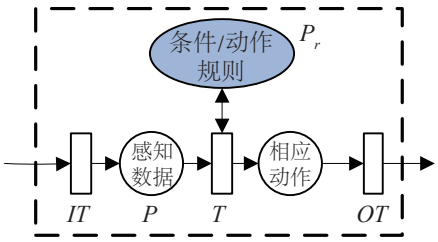

图 1　反应 Agent 模型

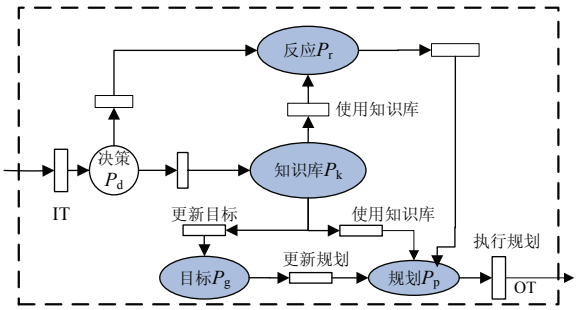

图 2　混合 Agent 模型

定义 3 混合 Agent 以 BDI (belief-desire-intention) 模型为基础, 是一个 12 元组, $HybridAComp = (P, P_d, P_r, P_k, P_g, P_p, T, IT, OT, F, W, \Pi)$, 其中 P_r, P_k, P_g, P_p 为抽象库所, 用椭圆表示,

① P_d 为决策模块, 主要用于消息派遣和事件捕获, 把感知的信息按类型派遣到反应模块或 BDI 模型中.

② 反应模块 P_r 表示 Agent 利用知识库存储的 IF-THEN 规则, 可以对紧急或简单情况做出快速反应.

③ 知识库模块 P_k 对应于 BDI 模型中的信念 (beliefs), 主要描述了环境和其他 Agent 的信息以及条件 – 动作规则库.

④ 目标模块 P_g 对应于 BDI 模型中的愿望 (desires), 主要描述了 Agent 的动机和最终目标, 具体的实现中可以表示为一个变量、数据结构或表达式.

⑤ 规划模块 P_p 对应于 BDI 模型中的意图 (intentions), 主要描述了节点 Agent 完成任务所需要的行为.

⑥ Π 利用 π 演算描述混合 Agent 的演化机制.

混合 Agent 模型如图 2 所示, 阴影库所为抽象库所, 可以根据系统需求进行求精. 反应 Agent 和混合 Agent 都是通过接口进行交互, Agent 的内部实现对于其他 Agent 是不可见的, 这样可以保持较好的封装性, 同时也可以简化对系统模型的分析. 在系统的开发中, 可以根据需求, 选择合适的 Agent 实现相应的功能. 关于 Agent 演化机制, 我们将在下一节研究.

2) 连接 Agent($AConn$)

连接 Agent 是计算 Agent 交互协议的实现, 定义了计算 Agent 之间交互的规则并且给出了一些实现的机制. 连接 Agent 从全局层次上协调、监督各个计算 Agent, 可以随时与每个计算 Agent 进行通信, 并对系统的目标、资源等进行合理安排.

连接 Agent 是一个五元组, $AConn_p = (ILP, Gate, KBP, Role, \Pi)$, 其中 ILP (intelligent link place) 为系统中的智能连接库所, 用双椭圆表示, 负责建立计算 Agent 之间的消息传递通道; Gate 表示 Agent 间的消息传递通道; KBP (knowledge-base place) 为知识库所, 存储系统中 Agent 的相关信息 (如名字、地址和接口信息等), 用椭圆表示; Role 是连接 Agent 中的角色, 为与连接 Agent 相交互的所有 Agent 的抽象集合; Π 利

用 π 演算描述了连接 Agent 的演化, 主要描述连接 Agent 建立计算 Agent 之间交互通道及负载平衡等演化过程, 我们将在下一节中研究.

CPS 软件各节点系统可以通过连接 Agent 和计算 Agent 组装而成. 如图 3(a) 所示, 多个计算 Agent 通过一个连接 Agent 交互形成一个小的系统. 一个复杂的系统可能由若干个子系统构成, 子系统之间也通过连接 Agent 进行交互, 如图 3(b). 通过计算 Agent 与连接 Agent 组装, 我们在 CPS 软件设计初始阶段关注较大粒度的 Agent, 随后再对这些大粒度的 Agent 进行精化, 使得整个软件设计过程更容易被管理和控制 [29].

3) 体系结构配置 ($Conf$)

$Conf$ 为 CPS 可信软件的配置, 主要描述了由计算 Agent 和连接 Agent 构成的 CPS 软件体系结构拓扑.

CPS 软件体系结构配置如图 4 所示, 主要从宏观层次上描述 CPS 软件体系结构, 侧重于软件系统的整体行为和 Agent 之间的交互, 也描述了 CPS 软件体系结构模型的静态语义. 图 4 中计算 Agent 1 和 2 通过通道 $G1$ 交互, 计算 Agent 3 和 4 通过通道 $G2$ 交互. 计算 Agent 都通过接口挂靠在连接 Agent 上, 形成了一种树状的拓扑结构.

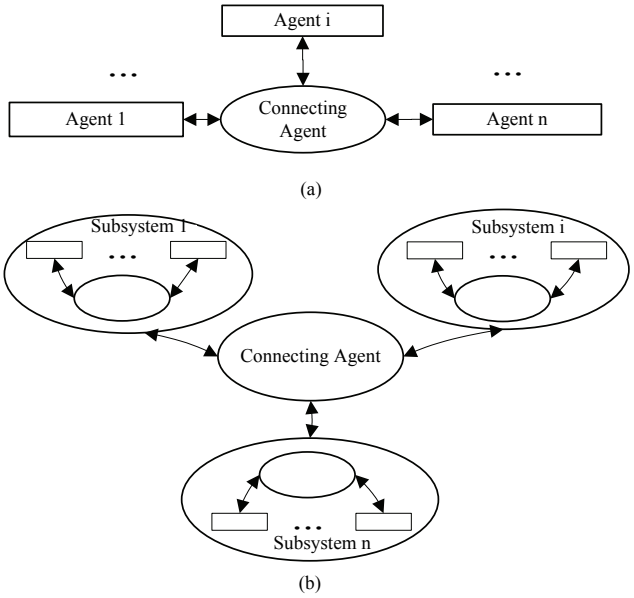

图 3 连接 Agent 示意图

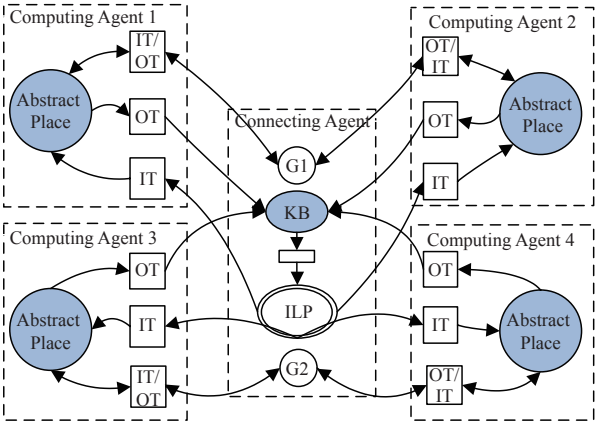

图 4 体系结构配置

2.2 CPS 可信软件动态演化建模

CPS 所处的环境和配置经常发生改变, 导致其软件体系结构经常随需求和环境的变化而进行动态调整与重构. CPS 软件的动态演化, 不是简单地进行各类 Agent 的创建或删除, 还需要确定软件演化的起因和时间, 给出系统演化方案, 并推理演化产生的后果, 从而确保系统正确、完整地进行动态演化 [30]. 否则, 在演化过程中可能会引入一定的风险, 导致系统崩溃, 带来巨大的损失.

研究 CPS 可信软件动态演化时, 需要考虑的问题包括:

1) 系统动态演化的起因. 大致可以分为两类: 一类是系统内部的原因, 即 Agent 内部出现异常. 另一类是系统外部原因, 如用户需求、负载平衡、外部干扰和攻击等.

2) 系统动态演化的时间. 在系统运行中, 我们不能随时、随意地对系统进行动态调整, 否则会导致系统死锁, 造成数据丢失或系统异常, 带来极大的风险. 只有当相关的 Agent 处于相对安全状态的时候, 才能对系统进行动态调整.

3) 系统动态演化的基本操作. 常见的动态演化包括各类 Agent 的创建或删除、Agent 的更新和动态配置等几种情况.

4) 系统动态演化的分析. 综合考虑演化的各种因素后, 给出系统动态配置的完整方案, 对体系结构的动态演化进行形式化描述, 然后进行分析和验证, 保证演化的正确性、完整性和一致性.

针对以上所述的 CPS 可信软件动态演化存在的诸多问题, 我们利用 π 演算对其进行描述, 在 HCSFM 中采用如下方案:

1) 在计算 Agent 和连接 Agent 中设置监控进程. 计算 Agent 的监控进程向连接 Agent 发送消息, 描述引发动态配置的内部原因, 如内部计算错误或出现异常, 请求进行动态配置. 连接 Agent 的监控进程与计算 Agent 的监控进程和外部环境进行交互, 可以描述演化的外部原因, 在全局层次上控制体系结构的动态演化.

2) 连接 Agent 作为系统监督控制器, 定义算子 $Create, Delete, Update$ 等表示动态演化的基本操作.

3) 描述了系统的演化后, 需要进一步分析最终体系结构的正确性、一致性等特性, 这将在下一节进一步研究.

下面首先给出计算 Agent 与连接 Agent 内部的监控进程, 然后研究系统动态演化的几种情况, 如 Agent 的加入和退出、更新和体系结构重配置等 [29]. 需要注意的是, 在某个时刻计算 Agent 的退出和更新, 必须确保在这一时刻计算 Agent 的状态为静止, 即不存在与其他 Agent 的交互. 在以下的分析中, 假定计算 Agent 退出和更新时, 其状态均为静止.

计算 Agent 内部的监控进程为

$$Monitor(request, config) = \overline{request}\langle id, updinfo\rangle.config(x, y)([x = id, y = begin]\ Update).RegInfo\langle id, s\rangle.Monitor(request, config) \tag{1}$$

其含义为: 进程 $Monitor$ 通过通道 $request$ 把 id 和更新信息 $updinfo$ 发送到连接 Agent 中的监控进程, 然后通过通道 $config$ 等待连接 Agent 通知是否可以进行更新, 利用进程 $Update$ 更新完毕后通过进程 $RegInfo$ 向连接 Agent 重新注册.

连接 Agent 内部的监控进程为

$$Supervisor(request, info, config) = request(u, v).\overline{info}\langle ids, wait\rangle.\overline{config}\langle id, begin\rangle.\ CRegInfo\langle x, y\rangle.Supervisor(request, info, config) \tag{2}$$

其含义为: 进程 $Supervisor$ 通过通道 $request$ 接收需要动态配置的计算 Agent 信息, 通过通道 $info$ 通知与动态配置 Agent 相关的 Agent 暂停服务, 然后接受新 Agent 的消息注册.

1) Agent 的加入和退出

当新 Agent 加入时, 新 Agent 首先在连接 Agent 中注册, 然后通过 ILP 建立和其他 Agent 的交互通道. Agent 创建进程为

$$NewAComp(id, s) = \tau.Create(id, s) \tag{3}$$

表示创建了一个标志符为 id, 服务为 s 的 Agent. Agent 注册进程可以描述为

$$RegInfo(register) = (\nu id, s)(\overline{register}\langle id, s\rangle) \tag{4}$$

上式表示 Agent 通过通道 $register$ 传递其标志符 id 和服务 s 到连接 Agent 中. 在连接 Agent 中相应的注册进程为

$$CRegInfo(register) = (\nu x, y)(register(x, y).UpdateKB) \tag{5}$$

表示连接 Agent 通过通道 $register$ 获取新 Agent 的注册信息 x 和 y, 同时更新其知识库 KBP.

当新 Agent 请求一个服务的时候, 由于事先不知道哪一个 Agent 能提供所需的服务, 需要通过连接 Agent 查找提供相应服务的 Agent. 如果存在相应的服务 Agent, 连接 Agent 负责建立这两个 Agent 之间的交互通道; 如果不存在相应的服务 Agent, 请求 Agent 可以向连接 Agent 定购这个服务, 一旦有提供这类服务的 Agent 注册, 连接 Agent 就向请求 Agent 发送消息, 通知可以提出相应的服务请求. 新 Agent 请求服务的进程为

$$RequestService(i, r, subscribe, l) = \overline{i}\langle a\rangle.r(z).([z = nil]\overline{subscribe}\langle a\rangle + \overline{z}\langle l\rangle) \tag{6}$$

表示新 Agent 通过通道 i 发送请求 a 到连接 Agent 查询相应的服务 Agent, 然后通过通道 r 等待连接 Agent 的答复. 一旦新 Agent 收到了服务 Agent 的标志符 z, 就通过 z 发送请求服务的地址 l 到服务 Agent. 如果新 Agent 收到的服务 Agent 的标志符为空, 则向连接 Agent 定购这个服务.

在连接 Agent 中相应的服务查询进程为

$$QueryService(i, kb, r, subscribe) = i(y).(\overline{kb}\langle y\rangle|Belief(y)).(\overline{r}\langle nil\rangle.subscribe(y) + \overline{r}\langle p\rangle) \tag{7}$$

表示连接 Agent 通过通道 i 接收请求 Agent 的请求 y, 然后通过知识库判断是否存在相应的服务 a, 如果存在, 通过通道 r 发送该 Agent 的标志符 p, 否则的话, 发送空并定购该服务.

在服务 Agent 中相应的服务提供进程为

$$ProvdService(p,s) = p(x).\overline{x}\langle s\rangle \tag{8}$$

表示服务 Agent 通过通道 p 接收服务请求的地址 x, 并通过 x 发送相应的服务 s 到请求 Agent.

如果某个 Agent 达到其目标, 需要从系统中退出, 则要把它的信息从当前的连接 Agent 中删除, 与其他 Agent 的交互通道也要删除. Agent 退出进程可以描述为:

$$AComp(id,s) = Delete(id,s) \tag{9}$$

表示删除了一个标志符为 id, 服务为 s 的 Agent.

2) Agent 更新

由于 CPS 出现故障、升级或遭到攻击, 需要对 Agent 进行更新, Agent 更新步骤如下: 1) 更新发起者向连接 Agent 发送 Agent 替换的命令, 包括原 Agent 和更新 Agent 的信息; 2) 连接 Agent 把更新信息通知与原 Agent 发生交互的其他 Agent, 并禁止其他 Agent 向原 Agent 发起新的事务, 并且缓存新的事务; 3) 连接 Agent 向更新 Agent 发送激活信息, 使其进行初始化, 然后完成替换过程; 4) 连接 Agent 将其他 Agent 新发起的事务引导到更新 Agent, 并向原 Agent 发出退出消息.

Agent 的更新方式分为两种情况, 第一种情况由于 Agent 内部出现错误或算法效率低, 需要对 Agent 内部进行更新, 即 Agent 的内部演化, 称为保持语义更新; 第二种情况, 由于系统出现新的需求, 需要具有新功能的 Agent 对原 Agent 进行更新, 称为扩展性更新.

对于第一种情况, 可以利用 π 演算的弱等价关系判断新旧 Agent 能否进行更新, 弱等价关系可以判断两个具有不同内部结构的系统从外部看来是否等价. 如果两个 Agent 行为等价, 则可以用一个 Agent 代替另外一个 Agent, 外部环境不能察觉替换的发生. 保持语义更新规则如下:

规则 1 [Agent 保持语义更新] 设 $Agent_0$ 和 $Agent_1$ 的行为分别被表示为进程 P 和 R, 若 $P \approx R$, 则 $Agent_1$ 可以对 $Agent_0$ 进行保持语义更新, 记为 $Agent_1 \rhd Agent_0$.

Agent 保持语义更新的一种直观意义就是 Agent 外部行为保持不变, 仅对 Agent 内部进行更新. 对于第二种情况, 需要有功能更为强大的 Agent 更新旧 Agent. 扩展性更新规则如下:

规则 2 [Agent 扩展性更新] 设 $Agent_0$ 和 $Agent_1$ 的行为分别被表示为进程 P 和 R, P 和 R 满足如下条件:

1. $fn(P) \subseteq fn(R)$;
2. 如果 $P \xrightarrow{\tau} P'$, 则 $\exists R'$, $R \Rightarrow R'$;
3. 如果 $P \xrightarrow{x(z)} P'$, 那么 $\exists R', R \xrightarrow{x(z),\cdots,x_i(z_i)} R'$ ($\xrightarrow{x(z),\cdots,x_i(z_i)}$ 表示 R 可响应若干个输入行为);
4. 如果 $P \xrightarrow{\overline{x}(y)} P'$, 那么 $\exists R', R \xrightarrow{\overline{x}(y),\cdots,\overline{x}_i(y_i)} R'$($\xrightarrow{\overline{x}(y),\cdots,\overline{x}_i(y_i)}$ 表示 R 可执行若干个输出行为);
5. 如果 $P \xrightarrow{\overline{x}y} P'$, 那么 $\exists R', R \xrightarrow{\overline{x}y,\cdots,\overline{x}_i y_i} R'$.

则 R 对 P 进行了扩展性更新, 因而 $Agent_1$ 对 $Agent_0$ 进行了扩展性更新, 记为 $Agent_1 \rhd \rhd Agent_0$.

条件 1 说明了 P 的自由名集合为 R 自由名集合的子集; 条件 2 说明了如果 P 能执行一个内部动作到 P', 那么 R 至少可以执行一个内部动作到 R'; 条件 3、4、5 分别说明了 R 除了响应和执行 P 的动作, 还可以执行其它的动作. 扩展性更新说明新 Agent 除了保持系统原有行为外, 还提供了一些新的行为满足系统需求.

3) 体系结构重配置

为了提高 CPS 软件的稳定性, 系统中需要加入一些备份 Agent, 这样当主 Agent 发生错误时, 可以启用备份 Agent, 同时主 Agent 与其他 Agent 的交互也需要切换到备份 Agent 上, 从而系统体系结构发生改变.

例如 Agent 1 和 2 通过通道 G1(记为 y) 进行交互, 在系统运行时加入一个备份 Agent 1' 通过通道 bak 备份其数据. 当 Agent 1 发生异常需要临时关闭时, 需要通知 Agent 2 把链接通道切换到备份 Agent 1', 通道切换过程如下:

$$(\nu bak)(\overline{y}\langle bak\rangle.Agent_1|Agent_{1'})|y(x).Agent_2 \xrightarrow{\tau} (\nu bak)(Agent_1|Agent_{1'}|Agent_2\{bak/x\}) \tag{10}$$

表示 Agent 1 在停止服务前, 通过通道 y 将原来属于自己和备份 Agent $1'$ 之间的私有通道 bak 传递给 Agent 2, 系统的拓扑结构发生改变, 使得 Agent 2 可以通过通道 bak 访问备份 Agent $1'$.

3 CPS 可信软件动态演化分析

在 CPS 可信软件演化过程中, 主要研究系统演化过程中软件的一致性, 保证演化后的系统能正常交互. 在动态演化分析阶段, 我们主要关注各类 Agent 之间的交互而忽略 Agent 的内部实现细节, 由于 Agent 通过接口进行交互, 因此可以使用 π 演算进程表达式来描述 Agent 接口的动态行为, 进而利用 π 演算的相关分析方法分析演化模型的一致性. 我们首先分析 Agent 内部实现与接口的兼容性, 然后分析体系结构的一致性.

3.1 Agent 内部实现与接口的兼容性分析

Agent 由接口和内部实现组成. 接口代表了 Agent 提供或需要的服务, 表示从一个特定的逻辑角度观察 Agent 时所看到的 Agent 能够执行的行为. 因此, Agent 的内部实现必须和接口的相关行为保持兼容, 即 Agent 必须正确执行接口所规定的行为. 为了便于理解及清晰地描述模型, 我们把模型中复杂的部分替换为语义上相等的模型, 把 Agent 接口用 π 演算的进程表示, 利用 π 演算的名字隐藏等概念 [31] 对兼容性进行分析.

定义 4 (名字隐藏) 给定进程 P 和名字集合 $\mathscr{N} \subseteq fn(P)$, $P/\mathscr{N}$ 表示隐藏进程 P 中属于集合 $\mathscr{N}$ 中的名字后所得到的进程, 则 $P/\mathscr{N} = (\nu\mathscr{N})\Big(P|\prod\limits_{n\in\mathscr{N}} Hide(n)\Big)$, 其中 $Hide(n) = n(m).(Hide(n)|Hide(m)) + (\nu m)(\overline{n}\langle m\rangle.(Hide(n)|Hide(m)))$.

对于每一个名字 $n \in \mathscr{N}$, 进程 $Hide(n)$ 把它隐藏在 P 中. $Hide(n)$ 的含义是提供名字 n 的输入或输出前缀, 与 P 结合并与 P 中 n 的输出或输入前缀进行交互, 从而约简为内部行为 τ, 达到隐藏 n 的目的.

隐藏名字与限制名字不同. 限制名字是进程内部通信使用的名字, 不能与外界直接交互, 但可以通过通道将限制名字作为值传递给另外一个进程, 从而能够在原有的进程和传送值的进程之间使用限制名字进行通信. 隐藏则是将进程的部分自由名字隐藏, 通过使用隐藏可以将特定事件转化为内部事件或者将特定的通信通道局部化以免受到外部环境的干扰.

定义 5 (Agent 接口) 假设一个 Agent $Acomp$ 和进程 P, 如果 $fn(p) \subseteq fn(AComp)$ 并且 $P \approx AComp/(fn(AComp) - fn(P))$, 那么称进程 P 为 $Acomp$ 的一个接口.

定义 5 表示接口执行的行为是 Agent 内部实现的一个子集, 因此接口的自由名集合为 Agent 自由名集合的一个子集, 同时在 Agent 中把非本接口的名字进行隐藏, 把它们作为内部行为后得到的进程应与 P 弱等价. 同理可以得出 Agent 的其他接口.

定义 5 表明可以通过名字隐藏获得相应的接口, 但并不是每一个满足定义 5 的进程都能正确地描述 Agent 的内部实现, 可以根据如下结论判断 Agent 接口与实现的兼容性.

结论 1 假设一个 Agent $Acomp$ 及其接口的集合 $P = \{P_1, P_2, \cdots, P_n\}$, 满足如下条件:

1. $fn(P_i) \cap fn(P_j) = \emptyset, \forall i \neq j$;
2. $AComp \Rightarrow 0$ iff $\forall i$, $P_i \Rightarrow 0$(这里 $\Rightarrow$ 代表 $(\xrightarrow{\tau})^*$, 表示存在 0 个或多个内部演化序列);
3. 如果 $\exists\alpha(\alpha \neq \tau)$, $AComp \xrightarrow{\alpha} AComp'$, 那么 $\exists i, P_i', P_i \xrightarrow{\alpha} P_i'$ 且 $AComp'$ 内部实现与 $P' = \{P_1, \cdots, P_i', \cdots, P_n\}$ 也是兼容的.

那么 $Acomp$ 内部实现与接口 P 是兼容的.

3.2 CPS 可信软件动态演化一致性分析 [28]

在 CPS 软件系统运行时, 由于新 Agent 的加入或 Agent 的更新, 导致系统发生演化, 可能会影响系统的一致性 (consistency)[32]. 一致性是指系统中各个部分必须成功的进行交互, 而不会彼此冲突. 一致性对于 CPS 软件动态演化尤为重要, 因为一旦经过演化后的 Agent 和连接 Agent 彼此发生冲突, 那么由此演化出来的系统就一定存在问题, 不能正常工作. 因此, CPS 软件动态演化的一致性研究具有重要意义 [33].

在 HCSFM 中, Agent 把相应的信息注册在连接 Agent 中, 通过连接 Agent 建立 Agent 之间的交互通道, 然后 Agent 通过接口进行交互. 由于 Agent 接口与内部实现是兼容的, 因而可以在接口层次上判断 Agent 交互的一致性 [34]. 根据定义 5, Agent 接口用进程描述, 我们首先定义进程之间的一致性关系, 进而得出 Agent 的一致性定理.

如果两个进程 P 和 Q 是一致的, 则 P 和 Q 至少能借助一个共享的对偶名字进行交互, 如 $P = \overline{x}\langle y\rangle.P'$, $Q = x(z).Q'$. 这样的进程之间存在同步关系.

定义 6 (一致性关系 [31]) 设同步进程集合上的二元关系 R, 对于 PRQ, 且对所有的替代 $\sigma \notin fn(P) \cup fn(Q)$, 都满足 $P\sigma R Q\sigma$, 而且

1. 如果 $P \xrightarrow{\tau} P'$, 则 $P'RQ$;
2. 如果 $P \nsim 0 \wedge P \neq 0$, $P \xrightarrow{x(z)} P' \wedge Q \xrightarrow{\overline{x}y} Q'$, 则 $P'\{y/z\}RQ'$;
3. 如果 $P \nsim 0 \wedge P \neq 0$, $P \xrightarrow{x(n)} P' \wedge Q \xrightarrow{\overline{x}(n)} Q'$, 则 $P'RQ'$.

则称 R 为进程间半一致性关系. 如果 R 和 R^{-1} 都是半一致性关系, 则称其为进程间一致性关系, 记为 $\backsimeq$.

进程的一致性强调进程的正常交互, 表明进程之间能够通过对偶名字进行通信从而成功到达最终状态. 如果存在进程 P', 使得 $P \Rightarrow P'$, 且 $P' \xrightarrow{\tau}$ 或 $P' \equiv 0$($P' \xrightarrow{\tau}$ 表示 $\exists P'', P' \xrightarrow{\tau} P''$), 则称进程 P 可正常运行, 能成功到达最终状态; 如果 $P \Rightarrow P', \neg(P' \xrightarrow{\tau})$ 且 $P' \neq 0$, 就表示进程死锁, 进程不能再执行内部演化, 其行为也并未完全执行.

定理 1 (Agent 一致性) 设 P 和 Q 分别为 $Agent_1$ 和 $Agent_2$ 的接口, 而且 P 和 Q 具有一致性关系, 即 $P \backsimeq Q$, 则 $Agent_1$ 和 $Agent_2$ 满足一致性, 即 $Agent_1|Agent_2$ 能正常运行.

证明 由于 Agent 接口描述 Agent 提供的动作, 因此证明 Agent 之间满足一致性只需证明如果 $P \backsimeq Q$, 那么 $(\nu fn(P) \cup fn(Q))(P|Q)$ 可正常运行, 即存在一个进程 S(success), 使得 $(\nu fn(P) \cup fn(Q))(P|Q) \Rightarrow S$, 其中 $S \xrightarrow{\tau}$ 或 $S \equiv 0$.

设 $fn(P) \cup fn(Q) = \mathscr{N}$, 则 $(\nu fn(P) \cup fn(Q))(P|Q) = (\nu\mathscr{N})(P|Q) \Rightarrow S$, 我们利用数学归纳法证明经过 n 次内部演化后进程仍能正常运行.

1. 当 $n = 0$ 时, 证明 $(\nu\mathscr{N})(P|Q) \xrightarrow{\tau}$ 或 $(\nu\mathscr{N})(P|Q) \equiv 0$ 成立. 由于 $P \backsimeq Q$, P 和 Q 为同步进程, 有 $\exists\alpha, P \xrightarrow{\alpha}$ 和 $Q \xrightarrow{\overline{\alpha}}$, α 和 $\overline{\alpha}$ 为借助一个共享的对偶名字进行交互的动作, 为进程的内部演化, 因此 $(\nu\mathscr{N})(P|Q) \xrightarrow{\tau}$ 成立.
2. 假设 $n = k, k > 0$ 时, $\forall P', Q', P' \backsimeq Q'$, $(\nu\mathscr{N})(P'|Q')(\xrightarrow{\tau})^k S$ 成立, 其中 $S \xrightarrow{\tau}$ 或 $S \equiv 0$.
3. 当 $n = k + 1$ 时, 要证明 $(\nu\mathscr{N})(P|Q) \xrightarrow{\tau} (\nu\mathscr{N})(P'|Q')(\xrightarrow{\tau})^k S$ 成立. 根据定义 6, 对于第一次内部演化, 可以得到
   - $P \xrightarrow{\tau} P'$, 由于 $P \backsimeq Q$, 则 $P' \backsimeq Q$.
   - $Q \xrightarrow{\tau} Q'$, 由于 $P \backsimeq Q$, 则 $P \backsimeq Q'$.
   - $P \xrightarrow{x(z)} P'$, $Q \xrightarrow{\overline{x}y} Q'$, 由于 $P \backsimeq Q$, 则 $P'\{y/z\} \backsimeq Q'$.
   - $P \xrightarrow{x(n)} P'$, $Q \xrightarrow{\overline{x}n} Q'$, 由于 $P \backsimeq Q$, 则 $P' \backsimeq Q'$.
   - $Q \xrightarrow{x(z)} Q'$, $P \xrightarrow{\overline{x}y} P'$, 由于 $P \backsimeq Q$, 则 $P' \backsimeq Q'\{y/z\}$.
   - $Q \xrightarrow{x(n)} Q'$, $P \xrightarrow{\overline{x}n} P'$, 由于 $P \backsimeq Q$, 则 $P' \backsimeq Q'$.

因此若 $(\nu\mathscr{N})(P|Q) \xrightarrow{\tau} (\nu\mathscr{N})(P'|Q')$, 则 $P' \backsimeq Q'$, 根据假设得 $S \xrightarrow{\tau}$ 或 $S \equiv 0$ 成立, 因此 $(\nu fn(P) \cup fn(Q))(P|Q)$ 可成功运行, $Agent_1$ 和 $Agent_2$ 满足一致性, 得证.

定理 1 保证了 Agent 行为的匹配性, 然而它只分析了 Agent 由一个接口描述其行为, 实际上 Agent 可能由若干个接口组成, 这样的 Agent 一致性证明可以通过对定理 1 进一步扩展得到.

定理 2 如果 CPS 软件系统中相互交互的 Agent 完全满足一致性, 则该系统可以正常交互, 执行期望的行为.

由定理 1 可以导出, 证明比较简单, 从略.

通过对 CPS 可信软件动态演化的一致性分析, 可以确保演化后系统能正常交互, 执行用户期望的行为.

4 建模、分析与仿真

本节以一支无人驾驶车辆 (unmanned ground vehicle, UGV) 编队执行作战任务为应用背景. 敌方派遣一辆 UGV 进入我方边境, 我方指挥中心调遣一支 UGV 编队 (包括两辆 UGV) 对目标实施侦察. 我们将整个系统作为一个 CPS, 每一辆 UGV 是 CPS 中的传感器节点, 指挥中心作为控制器节点, 节点之间通过电台进行沟通和联络. 将每一辆 UGV 视为一个 Agent, 指挥中心作为一个监督控制 Agent(supervisor), 是整个系统中的连接 Agent, 负责制定作战计划, 将目标信息和初始作战任务分配给 UGV Agent 编队, 并在全局层次上进行协调. 监督控制 Agent 还可以随时与每辆 UGV Agent 进行通信, 了解任务的完成情况及战场的态势信息等. 为了建立 UGV Agent 编队的形式化模型, 根据 HCSFM, 并对目标和规划库所进行求精, 得到该

CPS 的形式化模型如图 5 所示.

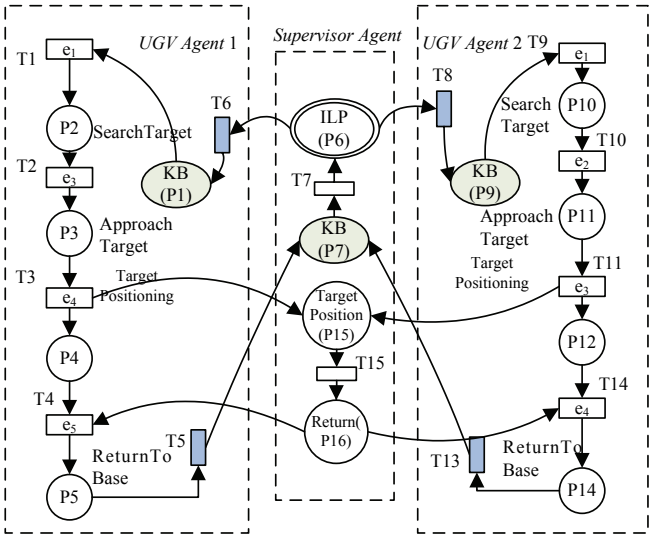

图 5　UGV Agent 编队 CPS 形式化模型

图 5 所示的模型可以较好地描述系统的静态语义, 动态语义可以通过变迁的使能和发射规则来描述. 为了确保系统的安全性和无死锁性等关键属性的正确性, 需要对模型进行分析和验证. 我们利用 Petri 网分析工具 INA[35] 对图 5 的系统模型进行相应的分析. 分析结果表明, 模型是有界的, 可达状态有 104 个, 且模型是活的, 不存在死锁, 可以保证系统的正常运行, CPS 中的各节点能达到预期的目标.

根据图 5 所示的模型, 对该 CPS 进行仿真. 假设系统初始时 UGV Agent 编队与目标相距最远. 系统初始配置如图 6 所示, 而且环境中有障碍物存在, 仿真结果如图 7 所示. 图中三角形代表 UGV Agent 和目标的初始位置, 圆圈代表 UGV Agent 追捕上目标 UGV 时所处的方位,

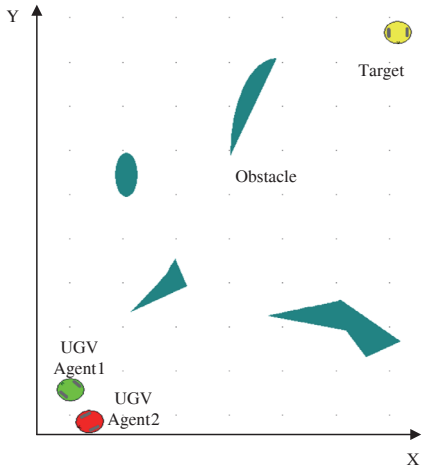

图 6　系统初始配置

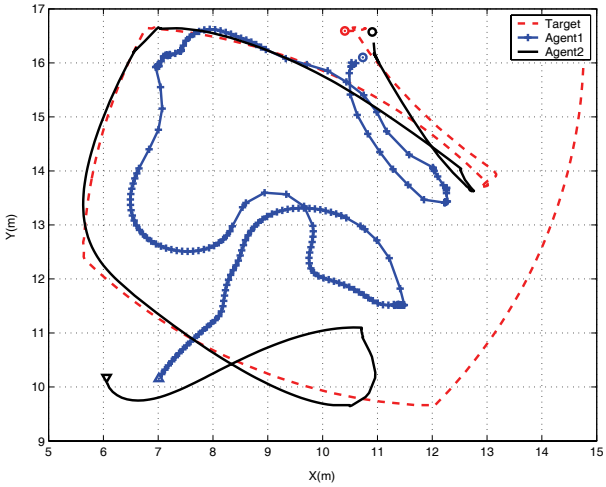

图 7　UGV Agent 与目标运动轨迹

为了实现打击目标的需要, 指挥中心决定派出一辆攻击 UGV 对目标实施打击. 攻击 UGV 作为 CPS 中的执行器节点, 根据 CPS 可信软件动态演化建模, 攻击 UGV 首先在监督控制 Agent 中进行注册, 然后建立它们之间的交互通道 *attack*. 此时主要关注的是攻击 UGV 到达后系统的一致性问题, 系统能否正常交互. 因此根据上一节的 CPS 可信软件动态演化分析, 把新加入攻击 UGV 的接口和与攻击 UGV 有交互行为的 Agent 接口用进程描述出来, 然后根据接口判断攻击 UGV 的加入是否会影响系统的一致性. 攻击 UGV 只与监督控制 Agent 交互, 攻击 UGV 与监督控制 Agent 增加的接口可以定义如下:

攻击 UGV:

$$AttackUGVAgent(attack) = attack(x).AttackUGVAgent(attack) \tag{11}$$

监督控制 Agent:

$$SupervisorAgent(attack) = \overline{attack} < location > .SupervisorAgent(attack) \tag{12}$$

监督控制 Agent 由于提供了新的功能, 接口发生了变化, 通过通道 *attack* 发送目标位置信息 *location* 与攻击 UGV 进行交互. 根据规则 2, 监督控制 Agent 发生了扩展性更新, 然后根据定义 6 很容易得到 $AttackUGVAgent \backsimeq SupervisorAgent$, 那么根据定理 1, $AttackUGVAgent|SupervisorAgent$ 能正常运行, 进而根据定理 2, 攻击 UGV 加入该 CPS 后, 系统仍能保持正常交互, 确保任务的顺利执行.

5 结束语

本文以两种互补的形式化方法 Petri 网和 π 演算为语义基础, 建立了 CPS 可信软件形式化模型 HCSFM, 利用 Petri 网形象地描述 CPS 软件的体系结构模型及动态行为, 用 π 演算描述系统的动态演化. 为了确保演化后 CPS 能正常交互, 利用 π 演算的相关分析方法讨论了演化后 CPS 可信软件的一致性. HCSFM 可以比较直观地刻画系统的整体和个体特性, 能促进客户、体系结构设计者和开发人员之间的交流和理解, 比较适合描述复杂的 CPS 软件系统, 具有一定的理论研究和实际意义.

参考文献

[1] National Science Foundation of the United States. Cyber-physical system (CPS) program solicitation[EB/OL]. [2012-08-01]. http://www.nsf.gov/pubs/2010/nsf10515/nsf10515.htm.

[2] Lee E A. Cyber physical systems: Design challenges[C]// Proceedings of the 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing, 2008: 363–369.

[3] 王中杰, 谢璐璐. 信息物理融合系统研究综述 [J]. 自动化学报, 2011, 37(10): 1157–1166.
Wang Zhongjie, Xie Lulu. Cyber-physical systems: A survey[J]. Acta Automatica Sinica, 2011, 37(10): 1157–1166.

[4] 何积丰. Cyber-physical systems[J]. 中国计算机学会通讯, 2010, 6(1): 25–29.
He Jifeng. Cyber-physical systems[J]. Communications of the CCF, 2010, 6(1): 25–29.

[5] 温景荣, 武穆清, 宿景芳. 信息物理融合系统 [J]. 自动化学报, 2012, 38(4): 507–517.
Wen Jingrong, Wu Muqing, Su Jingfang. Cyber-physical system[J]. Acta Automatica Sinica, 2012, 38(4): 507–517.

[6] 李仁发, 谢勇, 李蕊, 等. 信息 - 物理融合系统若干关键问题综述 [J]. 计算机研究与发展, 2012, 49(6): 1149–1161.
Li Renfa, Xie Yong, Li Rui, et al. Survey of cyber-physical systems[J]. Journal of Computer Research and Development, 2012, 49(6): 1149–1161.

[7] Song Z, Sastry C R, Chen Y. Optimal observation for cyber-physical systems, a fisher-information-matrix-based approach[M]. Springer, 2009.

[8] Al-Hammouri A, Liberatore V, Al-Omari H, et al. A co-simulation platform for actuator networks[C]// Proceedings of the 5th International Conference on Embedded Networked Sensor Systems, 2007: 383–384.

[9] Lee I, Sokolsky O, Chen S, et al. Challenges and research directions in medical cyber-physical systems[J]. Proceedings of the IEEE, 2012, 100(1): 75–90.

[10] Derler P, Lee E A, Vincentelli A S. Modeling cyber-physical systems[J]. Proceedings of the IEEE, 2012, 100(1): 13–28.

[11] Lin J, Sedigh S, Miller A. Modeling cyber-physical systems with semantic agents[C]// Proceedings of the 34th Annual IEEE Computer Software and Applications Conference Workshops, 2010: 13–18.

[12] Cortes L A, Eles P, Peng Z. Modeling and formal verification of embedded systems based on a Petri net representation[J]. Journal of Systems Architecture, 2003, 49(12–25): 571–598.

[13] 沈昌祥, 张焕国, 王怀民, 等. 可信计算的研究与发展 [J]. 中国科学: 信息科学, 2010, 40(2): 139–166.
Shen Changxiang, Zhang Huanguo, Wang Huaimin, et al. Research and development of trusted computing[J]. Scientia Sinica: Informations, 2010, 40(2): 139–166.

[14] 邓小妮. 基于模型检验与仿真的 C4ISR 系统需求验证方法研究 [D]. 长沙: 国防科技大学, 2008.
Deng Xiaoni. Research on the validation methods of the C4ISR system requirements based on model checking and simulation[D]. Changsha: National University of Defense Technology, 2008.

[15] OMG MDA guide version 1.0.1[EB/OL]. [2012-08-23]. http://ww.omg.org/cgi-bin/doc?omg/03-06-01, 2003.

[16] Jiang Z, Pajic M, Mangharam R. Cyber-physical modeling of implantable cardiac medical devices[J]. Proceedings of the IEEE, 2012, 100(1): 122–137.

[17] Bruce R A, McMillin M. Model-checking BNDC properties in cyber-physical systems[C]// Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference, 2009: 660–663.

[18] Platzer A. Verification of cyber physical transportation systems[J]. Intelligent Transportation Systems, 2009: 10–13.

[19] Hunt W A. Modeling and verification of cyber-physical systems[C]// Proceedings of National Workshop on High-Confidence Automotive Cyber-Physical Systems, 2009: 660–663.
[20] Zhang J, Goldsby H J, Cheng B. Modular verification of dynamically adaptive systems[C]// Proceedings of the 8th ACM International Conference on Aspect-Oriented Software Development, 2009: 161–172.
[21] Bujorianu M C, Barringer H. An integrated specification logic for cyber-physical systems[C]// Proceedings of the 14th IEEE International Conference on Engineering of Complex Computer Systems, 2009: 291–300.
[22] Susuki Y, Koo T J, Ebina H, et al. A hybrid system approach to the analysis and design of power grid dynamic performance[C]// Proceedings of the IEEE, 2012, 100(1): 225–239.
[23] Thacker R A. A new verification method for embedded systems[D]. The University of Utah, 2010.
[24] 范贵生, 虞慧群, 陈丽琼, 等. 策略驱动的可靠嵌入式系统建模及分析方法 [J]. 软件学报, 2011, 22(6): 1123–1139.
Fan Guisheng, Yu Huiqun, Chen Liqiong, et al. Strategy driven modeling and analysis of reliable embedded systems[J]. Journal of Software, 2011, 22(6): 1123–1139.
[25] Murata T. Petri nets: Properties, analysis, and application[J]. Proceedings of the IEEE, 1989, 77(4): 541–580.
[26] Milner R. Communicating and mobile systems: The Pi calculus[M]. Cambridge: Cambridge University Press, 2009.
[27] 胡军. 构件化嵌入式软件设计的分析与验证 [D]. 南京: 南京大学, 2005.
Hu Jun. Formal analysis and verification for component-based embedded software designs[D]. Nanjing: Nanjing University, 2005.
[28] 于振华, 蔡远利, 徐海平. 基于 π 网的多 Agent 系统建模与分析 [J]. 系统工程理论与实践, 2007, 27(7): 77–84.
Yu Zhenhua, Cai Yuanli, Xu Haiping. Modeling and analyzing multi-agent systems based on π -net[J]. Systems Engineering — Theory & Practice, 2007, 27(7): 77–84.
[29] 蔡远利, 于振华, 张新曼. 多 Agent 系统形式化建模方法研究 [J]. 系统仿真学报, 2007, 19(14): 3151–3157.
Cai Yuanli, Yu Zhenhua, Zhang Xinman. Formal modeling methodology for multi-agent systems[J]. Journal of System Simulation, 2007, 19(14): 3151–3157.
[30] 任洪敏. 基于 π 演算的软件体系结构形式化研究 [D]. 上海: 复旦大学, 2003.
Ren Hongmin. Research on software architectural formalism based on π-calculus[D]. Shanghai: Fudan University, 2003.
[31] Canal C, Pimentel E, Troya J M. Compatibility and inheritance in software architectures[J]. Science of Computer Programming, 2001, 41(9): 105–138.
[32] 马晓星, 余萍, 陶先平, 等. 一种面向服务的动态协同架构及其支撑平台 [J]. 计算机学报, 2005, 28(4): 467–477.
Ma Xiaoxing, Yu Ping, Tao Xianping, et al. A service-oriented dynamic coordination architecture and its supporting system[J]. Chinese Journal of Computers, 2005, 28(4): 467–477.
[33] Goudarzi K. Consistency preserving dynamic reconfiguration of distributed systems[D]. Imperial College London, 1998.
[34] Cimpan S, Leymonerie F, Flavio Oquendo F. Handling dynamic behaviour in software architectures[J]. Lecture Notes in Computer Science, 2005, 3527: 77–93.
[35] Roch S, Starke P H. INA: Integrated net analyzer, version 2.2[EB/OL]. [2012-09-01]. www2.informatik.huberlin.de/štarke/ina.html.