

学术观点

物联网软件低代码开发

李博睿 董 玮 高 艺 等
浙江大学

关键词：物联网 低代码 云-端一体

概述

低代码开发

根据著名咨询公司 Gartner 的预测，在 2018 年至 2023 年间，全球将创建超过 5 亿个应用程序，超过过去 40 年的总和，越来越多的企业和组织希望通过借助软件集成和自动化来简化工作流程、提高运营效率。2020 年新冠肺炎疫情开始在世界范围内蔓延，进一步迫使公司尽快完成数字化转型，将办公模式由线下转变为线上。在这个软件数量飞速增长的时代，企业和组织机构对程序员产生了前所未有的迫切需求。然而，大量的人才需求带来了更高的薪资溢价，增加了企业软件化转型的成本，对于中小企业而言，这一负担尤为沉重。

低代码开发平台（Low-Code Development Platform, LCDP）正是为了降低软件开发成本而生的。2014 年，LCDP 在一篇名为“New development platforms emerge for customer-facing applications”的文章中被著名咨询公司 Forrester 正式提出，它是一类无须编码或通过少量代码就可以快速生成应用程序的开发平台。LCDP 的理念核心可以追溯到 20 世纪 90 年代备受学术界和工业界关注的第四代编程语言（Fourth Generation of Languages, 4GL）。4GL 通常具备简单易学、用户界面友好、非过程化程度高、面向问题等特点，即开发者只需告知计算机“做什么”，而不必告知计算机“怎么做”。Gartner 预测，

全球低代码开发技术市场规模将从 2021 年的 138 亿美元快速攀升至 2024 年的 523 亿美元。届时，65% 的应用程序将在低代码平台中开发出来。一般来说，一个低代码开发平台应包含以下三个特征：

1. 提供应用程序平台即服务（Application Platform as a Service, aPaaS）能力：平台不仅需要提供常规的云服务，还要具备覆盖整个应用程序生命周期（如开发、编译、部署、运维等）的快速开发与交付的能力。

2. 提供移动应用开发平台（Mobile Application Development Platform, MADP）能力：用来快速定制场景化、覆盖桌面端和移动端的应用。

3. 提供业务流程管理（Business Process Management, BPM）能力：平台需要注重流程化开发，为用户提供业务逻辑与落成的可视化设计能力。

近两年，国内外的行业巨头纷纷布局低代码领域。被高德纳公司（Gartner）称为低代码平台行业领先者的 Mendix 被西门子以 7 亿美元收购；另一家行业领先者 OutSystems 获得了 3.6 亿美元融资，估值超 10 亿美元。在国内，腾讯和华为也推出了自己的低代码开发平台“微搭”和“应用魔方”。

近期，低代码开发也逐渐在学术界掀起浪潮。由北京大学谢涛教授在 CNCC2021 上组织的“软件自动化：从低代码到无代码开发”专题论坛，聚集了来自北京大学、复旦大学、微软研究院等高校与科研院所以及华为、腾讯等知名企业的专家学者，分享他们对于低代码的探索与思考。

物联网与低代码

已有的低代码开发平台（如 OutSystems、Mendix 等）主要关注企业数字化运营、机关无纸化办公等应用场景。然而，随着传感器、通信芯片等技术的不断发展与突破，物联网（Internet of Things, IoT）应用快速发展起来。据 Analytics 统计数据显示，2020 年，全球物联网设备连接数达到 117 亿台，首次超越非物联网设备连接数，预计到 2025 年，物联网设备连接数将增长到 309 亿台，是非物联网设备连接数的近 3 倍。同时，我国也将物联网列为“新基建”的重要组成部分，大力促进相关行业发展。在政策利好与市场青睐的双重加持下，面向物联网的低代码开发也是未来低代码产业的重要组成部分。

考虑到物联网应用独有的设备属性，除了前面提到的传统低代码开发的三大特征（aPaaS、MADP 与 BPM）之外，面向物联网软件的低代码开发还应提供设备应用开发平台（Device Application Development Platform, DADP）能力，以适配物联网设备端编程、云服务接入以及云-端融合等开发需求。因而，物联网低代码开发不可避免地面临以下三方面的挑战：（1）软硬耦合难开发，（2）云-端一体难构建，（3）设备异构难融合。我们将从这三点挑战入手，分别探讨其中的关键研究问题与国内外研究现状，分享我们的思考与探索。

物联网低代码开发的学术研究挑战及解决方法

软硬耦合难开发

一般来说，开发一个物联网节点端软件需要经历需求分析、硬件选型、软件编写、软硬联调四个阶段，如图 1 所示。在硬件选型过程中，开发者不仅需要关注硬件外设的功能与性能指标是否满足自己软件开发的需求，还要关注模块之间的电气连接特性；同样地，在软件程序编写时，开发者不仅需要关注软件本身的应用逻辑，还要处理不同

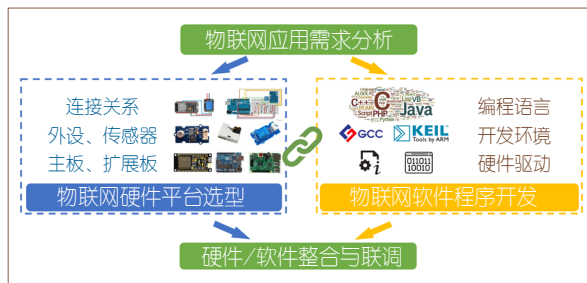


图1 物联网端侧应用开发流程

硬件平台开发环境的独特性与底层硬件驱动的差异性带来的问题。这种软件开发与底层硬件紧密耦合的情况给物联网端侧软件开发带来了极大的困难。

现今物联网应用的硬件平台、开发工具与操作系统纷繁复杂且相互依赖，如图 2 所示。以操作系统为例，现有面向资源受限设备的 FreeRTOS，重点关注无线网络协议栈的 Mynewt OS，以及注重多设备联动的鸿蒙操作系统，等等，它们各有所长。开发者在开发时经常会遇到某操作系统仅能支持有限的几种硬件平台，且需要使用指定的开发工具进行开发的情况。与当前平台与工具稳定、接口相对统一的移动应用和人工智能应用相比，物联网应用的开发难度更高。

因此，如何设计一个能够松弛软硬件耦合，且能够弥合硬件平台、开发工具以及操作系统之间碎片化现状的低代码开发框架是一个亟待解决的问题。

为了降低开发一个物联网应用所需的代码量，一个直接的想法就是让机器帮助我们自动生成代码。现在已经有一些工作在此方向上进行了探索，例如芬兰奥斯陆 SINTEF 信息与通信技术中心设计的物联网软件建模语言 Thing Modeling Language



图2 移动应用、人工智能应用以及物联网应用开发平台、工具、系统的复杂性对比

(ThingML) 和与其对应的自动代码生成框架^[1]。顾名思义, ThingML 借鉴了软件工程中“模型驱动的代码生成”的思想, 将物联网软件建模为“组件”和“配置”两部分。“组件”用于表示物联网设备上某个硬件模块(如传感器、执行器)的对外接口和内部的状态机实现,“配置”用来记录如何使用多个组件组成一个物联网设备,并定义组件之间的通信方式与协议。ThingML 的用户可以使用基于统一建模语言(Unified Modeling Language, UML)的建模方式来描述一个硬件。随后, 用户可以使用配置文件组合多个硬件, 并使用 ThingML 内置的代码生成器生成可编译的代码。

与 ThingML 类似, 利用模型驱动的思想进行物联网应用代码生成的工作还有 UML4IoT^[2]、SysML4IoT^[3]、IoTML^[4]等, 它们分别使用 UML、SysML 与 Eclipse Modeling Framework 等建模语言对物联网软件进行建模和自动生成, 从而实现低代码开发。然而, 这些建模方式都建立在一个假设上: 物联网软件开发所需的硬件已经确定, 无须开发者自己挑选。但是对于一个初学者来说, 在数以万计的开发板、外设组合中选择符合自己应用逻辑和性能需求的硬件无异于大海捞针。

针对这个问题, 我们课题组提出了基于软件定义的物联网节点端低代码开发技术 TinyLink^[5]。区别于传统的先硬件选型、后软件开发这种“自底向上”的模式, 我们提出了先软件代码编写、后节点配置生成的“自顶向下”的开发模型。开发者可以不用先做具体的硬件选型, 而是在一个所有感知、执行、网络等功能都已具备的虚拟硬件平台上, 利用 TinyLink 提供的与硬件无关的低代码 API(如使用 `TL_Light.read()` 获取光照强度) 直接开始应用逻辑代码的编写。除此之外, TinyLink 还允许开发者声明自己对物联网软件性能的需求, 例如希望生成的物联网节点价格最低或是功耗小于一个阈值。在应用代码编写完成后, TinyLink 系统会根据开发者编写的

应用逻辑和性能需求, 自动生成最符合其需求的硬件组件清单与连接关系, 将使用 TinyLink API 编写的代码自动扩展为可编译的、平台相关的代码, 并最终编译为二进制文件。用户只需按照生成的硬件连接关系连接硬件, 并烧写入对应的二进制代码, 就能快速完成物联网节点端软件的低代码开发。

TinyLink 不仅利用平台无关的低代码开发 API 降低了代码量, 还可以帮助用户进行按需硬件选型, 并且支持多种硬件平台的云端编译, 使得用户侧无须安装复杂的编译环境, 既解决了物联网端侧软件耦合难以开发的问题, 也缓解了平台和开发工具碎片化带来的开发困难。

云-端一体难构建

一个典型的物联网应用通常由物联网节点端、边缘端、云端以及移动端组成, 如图 3 所示。节点端用于感知周围环境并与物理世界进行联动; 边缘端(如网关、基站等)用于将局域网协议(如 BLE、ZigBee)转化为互联网协议, 或进行一些基础数据处理; 云端通常被用来对物联网数据进行深度分析, 为人们的决策提供指导; 移动端主要与终端用户进行交互, 用于展示数据、实现手动控制等。共享单车就是一个典型的物联网应用, 其中物联网设备为车锁, 其采集的相关数据首先传到移动基站, 再到云平台进行处理, 最后通过手机应用在移动端与用户交互。

由图 3 也可以看出, 节点、边缘、云、移动这四端的开发框架和编程语言各不相同, 在编程时还要考虑各端之间复杂的交互逻辑, 因此开发一个四

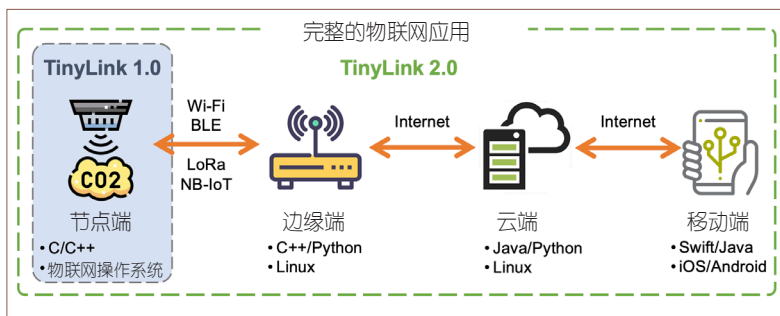


图3 典型物联网应用的构成与各端对应开发框架

端完整的物联网应用对开发者提出了更高的要求。TinyLink 虽然简化了物联网节点端的开发,但并没有简化包括边缘端、云端以及移动端的云-端一体的开发过程。为了进一步简化云-端一体应用的开发,我们提出了一种云-端一体的编程框架,它可以与 TinyLink 结合使用,是 TinyLink 思想的延伸,我们称之为 TinyLink 2.0^[6]。

TinyLink 2.0 的核心是提出了一种云-端一体的编程语言。在节点端, TinyLink 2.0 沿用了 TinyLink 的节点端低代码编程模型;在云端, TinyLink 2.0 遵循 IFTTT (IF This Then That, 一个被广泛应用的条件-触发编程模型) 的思想设计了简单易用的基于规则的编程方法;在移动端, TinyLink 2.0 提供了用于数据展示和交互的用户界面组件,供开发者编程调用。TinyLink 2.0 一体化语言有以下四个特点:

1. **微应用模型** 在 TinyLink 2.0 中,各个节点的能力和代码实现都封装在微应用模型 (TinyApp) 中,使用微应用模型可以快速实例化出多个节点。

2. **分布式交互** 各端均可定义公有数据和服务接口,在编写代码时可以直接引用其他端提供的公有接口,实现跨设备访问,简化编程开发。

3. **云定义节点行为** 在 TinyLink 2.0 中,考虑到云服务器计算能力强,且拥有全局信息,我们采用节点行为云端定义的模式。云端可以决定何时读取数据、以什么频率读取、何时发送指令等,节点端仅实现公用接口和简单逻辑。

4. **虚拟传感器** 有些传感器并不能直接产生我们所需要的事件,例如在门上安装的惯性传感器仅能提供原始的惯性传感器数据,而无法实现门开关状态的监测。为此, TinyLink 2.0 提出虚拟传感器的概念,只要开发者规定输入传感器的集合和事件的输出,以及内置的推理模型, TinyLink 2.0 就会生成一个虚拟传感器,自动选择合适的传感器并训练推理模型。

在完成开发后, TinyLink 2.0 将编译好的节点端二进制文件提供给用户,供用户烧写,也会自动化部署移动端页面和云端规则,用户可以直接访问。除此之外, TinyLink 2.0 还通过优化传感器上传间隔

的方式,将多个传感器数据合并上传以降低功耗。

TinyLink 2.0 将注意力集中于各端可执行代码的生成与分布式逻辑的构建上,而在 2021 年的操作系统领域顶级会议 ACM SOSP 上,加州大学伯克利分校的希尔维亚 (Sylvia Ratnasamy) 教授团队提出的 dSpace 编程框架^[7]则更加聚焦于多设备物联网应用逻辑的整体性抽象与构建。dSpace 使用模块化、层级化的思想进行云-端一体应用的开发,它包含两个基本要素: Digivice 和 Digidata。Digivice 用于对节点设备建模, Digidata 用来实现数据处理。多个 Digivice 可以通过 mount 原语组合成一个更高层次的 Digivice,例如可以将台灯和空调两个低层语义的 Digivice 组成一个书房的高层语义 Digivice,这样可以方便地定义不同的场景。

设备异构难融合

物联网节点受限于能耗或成本,通常无法完成重量级的计算任务,需要将任务卸载至计算能力更强的节点或者边缘端/云端进行计算。著名咨询机构 IDC 也提出“云边融合”是 2021 年的十大技术趋势之一。然而,在云-边-端这个多层架构中,层级内部与跨层间的显著异构性使得应用逻辑难以在设备间自由地流转与融合,具体来说,有以下两方面原因。

一方面,物联网节点端硬件指令集架构的异构性强。如图 4 所示,全球物联网微处理器市场占有率前 10 的公司几乎都拥有自研指令集架构(如 AVR、MSP、PowerISA 等),移动计算时代的霸主 ARM 指令集也仅占 25% 的份额,加之在 Wi-Fi 芯片市场占有率全球第一的国产微处理器 ESP32 使用的 Xtensa 指令集和不断吸引工业界和学术界关注的开源指令集 RISC-V,物联网节点端的异构性是 PC 时代(x86 架构主导)和移动计算时代(ARM 架构主导)无法比拟的。

另一方面,云端(x86 架构)、边缘端(通常为 ARM 架构)与物联网设备的跨层间异构性进一步阻碍了计算任务在各个计算层级之间的无缝流转,导致物联网难以充分利用云、边、端丰富的计算资

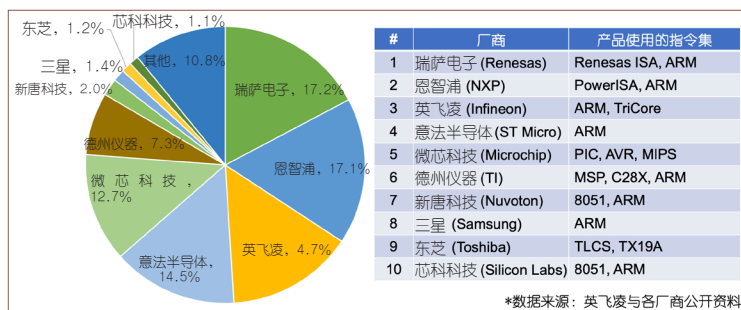


图4 2020年全球物联网微处理器芯片市场占有率与其产品所使用的指令集

源高效地完成任

实现跨异构指令集的计算融合主要有两种思路：一种是将代码直接编译为各个指令集的二进制并下发至对应设备，在需要迁移时，通过远程过程调用 (Remote Procedure Call, RPC) 或 RESTful API 调用等机制，远程调用远端程序进行计算；另一种方法是使用跨平台的虚拟机 (如使用 Java 虚拟机 JVM)，在需要迁移时，将跨平台的字节码传输至目标设备，恢复现场后继续执行。

已有的工作对于这两种思路均有探索。在基于远程调用机制的工作中，Quecc^[8]在调度计算时折中考虑了计算精度与计算时间，EdgeProg^[9]基于二进制文件的动态加载与链接技术实现了在迁移时按需加载应用逻辑。在跨平台虚拟机迁移方向，英特尔实验室的研究团队提出的 CloneCloud^[10]和德国电信实验室的团队提出的 ThinkAir^[11]都是基于 Java 虚拟机实现了节点端和云端之间的代码卸载。杜克大学库尔沃 (Cuervo) 等人提出的 MAUI^[12]借助 C# 语言的通用执行环境实现了手机与云服务器之间的计算卸载。

然而，基于远程调用与跨平台虚拟机的方法均存在一定缺陷。前者为了构建 RPC 或 RESTful API 接口导致冗余代码较多；后者支持的语言单一，且相对于原生 C 语言，程序运行效率较低。

我们最近提出了将 WebAssembly 技术引入云-边-端一体开发的新思路。WebAssembly (简称 Wasm) 是一个二进制字节码格式，规范由谷歌、微软、火狐等主流浏览器厂商组成的万维网联盟负责制定。2019 年 12 月，发布仅两年的 Wasm 成为

继 HTML、JavaScript 和 CSS 之后的第四个 Web 标准。Wasm 的技术特点包括：

1. 支持语言多 Java、C/C++、Rust 等主流编程语言可以直接编译为 Wasm。

2. 执行效率高 Wasm 在物联网设备上可以提供与原生 C 语言编程近似的性能 (同等情况下执行时间仅是原生 C 语言程序的 1.47 倍)，相较于其他跨平台语言 (如 Java、C# 等)，Wasm

的运行速度能够提高 16.8%~65.0%。

3. 具有其他虚拟机的良好特性 如良好的平台移植性、安全隔离性等。

更重要的是，主流浏览器基本都已经支持运行 Wasm 模块，因此基于 Wasm 虚拟机的物联网操作系统能提供云端、边端、移动端与节点端一致的执行环境，从而为万物互联、无缝迁移提供关键的技术支撑。

基于 Wasm 技术，我们提出了 WiProg^[13]，一种云-边-端协同的编程与计算迁移框架。有了 WiProg，如果要开发一个多端融合可迁移的人脸识别应用，只需要写一份代码，再对这份代码进行简单的标注就可以实现在节点端、边缘端和云端的代码自动切分和迁移。具体而言，我们可将代码中的一个函数标注为 “device_only”，那么它就只能在物联网节点侧执行，若将其标注为 “remotable”，则可以在任何一端执行。具体在哪里执行，WiProg 会根据用户自定义的迁移策略进行自动迁移。此外，在 Wasm 模块迁移时，除了模块本身，其代码运行的上下文也要一同迁移，这样才可以在远端继续执行。对此，WiProg 还提出了一个轻量级的上下文快照机制，大幅度降低了 Wasm 的迁移代价。

工业界的物联网低代码开发系统与平台

前文介绍的学术研究通常侧重于物联网低代码开发中的某个关键挑战，而工业界研发的系统与平

台则需要同时解决多个问题，并为用户呈现方便易用的开发界面。

对于节点端的低代码构建，微软推出了 .NET Gadgeteer 电子积木及其开发平台。 .NET Gadgeteer 使用基于 C# 的面向对象的编程方法，其平台包含了一组小型嵌入式硬件设备，这些设备无须焊接即可组装成为一个完整的物联网节点。微软不仅向开发者提供了众多易于编程的接口，并将其集成在 Visual Studio 中来简化应用的开发，还提供了每个电子组件的精确 CAD 模型，帮助开发者进一步使用 SolidWorks 或 Rhino 等 3D 建模软件来为生成的物联网节点设计外壳，使其更加接近工业量产产品。

对于云-端一体应用的低代码构建问题，工业界也做出了很多有益的尝试。国外的 Atmosphere IoT Studio 实现了覆盖云端、节点端和移动端的可视化开发，开发者仅需使用拖、拉、拽的方式就可以开发出一个完整的云-端一体应用。在用户完成开发后，Atmosphere 会自动将编写好的应用部署至云端，并提供节点端代码的下载，开发者可以自行将其编译并烧写至开发板上。类似地，开源低代码开发框架 NodeRED 也引起了开发者的广泛关注，它可以在树莓派、BeagleBone 等高性能物联网开发板上运行，帮助开发者完成应用的可视化逻辑构建与编排。

SmartThings 公司成立于 2012 年，专注于制造智能家居设备。这家公司起源于一个发布在众筹平台 kickStarter 上的项目。2014 年，SmartThings 被三星公司收购，如今已经发展成为一个综合性的智能家居平台，吸引了诸如飞利浦、霍尼韦尔等老牌电器品牌与其合作。SmartThings 可以通过统一的控制中心管理不同品牌和平台的物联网设备。除此之外，SmartThings 允许第三方开发者在平台上创建自定义的智能家居设备或应用。开发者可以根据不同的设备连接方式需求，灵活地使用平台支持的各类设备接入技术。

2021 年 11 月，亚马逊推出了 AWS IoT TwinMaker 来帮助开发人员借助低代码的开发方式构建建筑、工厂、生产线和设备等物理系统的数字孪生

(digital twin)。TwinMaker 内置了数据连接器供开发者将设备的数据传输至云端。它还会自动创建一个知识图谱，整合并理解所连接数据源的关系。另外，TwinMaker 还支持真实设备的 CAD 模型或点云扫描模型的导入，并将知识图谱中的数据叠加到 3D 视图上，大幅度简化了在云端构建数字孪生的流程。

涂鸦智能也是一家物联网低代码开发平台，成立于 2014 年，其平台单日最高服务设备数超过百万，并在 2021 年初成功赴美上市。在节点端，涂鸦智能提供了覆盖电工、照明等品类的“行业标准配置”，只要用户对于产品的功能需求能够在行业标准配置中找到，在涂鸦平台上通过可视化点选配置的方式就能完成节点端的开发。在云端和移动端，涂鸦平台也积累了很多公版应用方案供用户选择和进一步配置，大幅度提高了开发速度。

阿里云 IoT Studio 一站式开发平台覆盖了设备开发、云服务开发、Web 应用与移动应用开发等物联网开发的全过程。IoT Studio 使用物模型描述物联网设备的功能。物模型是物理空间中的实体(如传感器、车载装置、楼宇、工厂等)在云端的数字化表示，从属性、服务和事件三个维度，分别描述了该实体是什么，能做什么，可以对外提供哪些信息。定义了物模型这三个维度，即完成了对设备功能的定义。另外，IoT Studio 平台还支持 3D 场景的可视化低代码搭建，提供了大量三维空间组件，用于快速构建物理空间的数字孪生体。2018 年 5 月，浙江大学与阿里巴巴成立了浙大-阿里前沿技术联合研究中心物联网实验室。在此合作框架下，笔者所在研究团队在物联网低代码开发平台的一系列关键技术上取得突破(如差分空中更新技术、远程物联网实验平台构建技术等)，并在 IoT Studio 中进行了实际应用，取得了广泛的经济和社会效益。

总结与展望

物联网软件低代码开发顺应了后疫情时代的数字化转型浪潮，得到了快速发展。它能够帮助中小

型企业、个体开发者快速地构建物联网应用,降低数字化转型的成本和风险,真正实现降本增效。尤其是多端应用的协同构建技术和面向数字孪生的三维空间低代码应用搭建技术,在智能制造、智慧城市、智能家居等领域有着广阔的应用前景和巨大的发展空间。

在工业界多年积累的基础上,现在的物联网低代码开发已经形成了一套自成体系的技术方案和应用场景。学术界对于物联网低代码开发的研究虽然仍处于起步阶段,但是很多思路与工业界不谋而合,例如微软 .NET Gadgeteer 和 TinyLink 均采用了基于硬件 API 封装的低代码思想, dSpace 中的设备抽象 Digivice 与阿里云 IoT Studio 的物模型也是殊途同归。我们有理由相信,在日益增长的数字化、虚拟化、智能化需求的驱使下,工业界与学术界在物联网软件低代码开发方面将会有更多的合作机会。 ■



李博睿

CCF 学生会员。浙江大学计算机科学与技术学院博士研究生。主要研究方向为物联网与边缘计算系统、物联网低代码开发等。borui.li@zju.edu.cn



董 玮

CCF 高级会员。浙江大学计算机科学与技术学院教授。主要研究方向为物联网系统、低功耗无线网络、物联网感知与安全等。dongw@zju.edu.cn



高 艺

CCF 专业会员。浙江大学计算机科学与技术学院副教授。主要研究方向为网络测量理论与方法、无线与移动计算、物联网感知与安全等。gaoyi@zju.edu.cn

其他作者: 刘彦玮

(本文责任编辑: 郭得科)

参考文献

- [1] Harrand N, Fleurey F, Morin B, et al. ThingML: a language and code generation framework for heterogeneous targets[C]//*Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*. ACM, 2016: 125-135.
- [2] Thramboulidis K, Bochalis P, Bouloumpasis J. A framework for MDE of IoT-based manufacturing cyber-physical systems[C]//*Proceedings of the seventh international conference on the internet of things*. ACM, 2017: 1-8.
- [3] Hussein M, Li S, Radermacher A. Model-driven development of adaptive IoT systems[C]//*4th International Workshop on Interplay of Model-Driven Engineering and Component-Based Software Engineering. MODELS (Satellite Events)*. 2017: 17-23.
- [4] Dhouib S, Cuccuru A, Le Fèvre F, et al. Papyrus for IoT—A Modeling Solution for IoT[J]. *Proceedings l'Internet des Objets (IDO: Nouveaux Défis de l'Internet des Objets: Interaction Homme-Machine et Facteurs Humains)*, 2016.
- [5] Guan G, Dong W, Gao Y, et al. Tinylink: A holistic system for rapid development of iot applications[C]//*Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*. ACM, 2017: 383-395.
- [6] Guan G, Li B, Gao Y, et al. TinyLink 2.0: integrating device, cloud, and client development for IoT applications[C]//*Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*. ACM, 2020: 1-13.
- [7] Fu S, Ratnasamy S. dSpace: Composable Abstractions for Smart Spaces[C]//*Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. ACM, 2021: 295-310.
- [8] Li B, Dong W, Guan G, et al. Queec: QoE-aware Edge Computing for IoT Devices under Dynamic Workloads[J]. *ACM Transactions on Sensor Networks (TOSN)*, 2021, 17(3): 1-23.
- [9] Li B, Dong W. Edge-centric Programming for IoT Applications with Automatic Code Partitioning[J]. *IEEE Transactions on Computers*, 2021:1.
- [10] Chun B G, Ihm S, Maniatis P, et al. Clonecloud: elastic execution between mobile device and cloud[C]//*Proceedings of the sixth conference on Computer systems*. ACM, 2011: 301-314.
- [11] Kosta S, Aucinas A, Hui P, et al. Thinkair: Dynamic resource allocation and parallel execution in the cloud

for mobile code offloading[C]//2012 Proceedings IEEE Infocom. IEEE, 2012: 945-953.

[12]Cuervo E, Balasubramanian A, Cho D, et al. Maui: making smartphones last longer with code offload[C]// Proceedings of the 8th international conference on Mobile systems, applications, and services. ACM, 2010: 49-62.

[13]Li B, Dong W, Gao Y. WiProg: A WebAssembly-based Approach to Integrated IoT Programming[C]// IEEE INFOCOM 2021-IEEE Conference on Computer Communications. IEEE, 2021: 1-10.