计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2017,26(11):1−10 [doi: 10.15888/j.cnki.csa.006022]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于组件的信息物理系统描述语言①

时雨霖[1,2], 慈铁为[1], 杨秋松[1]

[1](中国科学院 软件研究所 基础软件国家工程研究中心, 北京 100190)
[2](中国科学院大学, 北京 100049)

摘　要: 本文设计实现了一种基于组件的信息物理系统描述语言 CDL(CPS Description Language). 基于面向参量模型, 将信息物理系统中的传感器、执行器和计算组件封装成具有统一抽象的组件, 将系统描述分解为系统中包含的组件、组件之间的关联以及系统约束三部分. 实现了基于 CDL 的信息物理系统的设计实现工具, 提供 CDL 生成、验证和安装的功能. 最后设计实现了两个应用实例, 验证了通过 CDL 来实现系统, 可以减少开发者编写的代码量, 提高编程效率, 一定程度上降低系统设计实现的难度.

关键词: 信息物理系统; 面向参量; 系统描述; 描述验证; 图形化编辑; XML

引用格式: 时雨霖,慈铁为,杨秋松.基于组件的信息物理系统描述语言.计算机系统应用,2017,26(11):1–10. http://www.c-s-a.org.cn/1003-3254/6022.html

Component-Based Description Language of Cyber-Physical System

SHI Yu-Lin[1,2], CI Yi-Wei[1], YANG Qiu-Song[1]

[1](National Engineering Research Center of Fundamental Software, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)
[2](University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: This paper presents a component-based language named CDL (CPS Description Language) to describe Cyber-Physical Systems according to the actor-oriented model. The sensors, actuators and compute components in a Cyber-Physics system are encapsulated into components with uniform abstraction. Therefore, a system description is decomposed into the components contained in the system, the directed relationships between components, and the system constraints. In addition, we design and implement the development tool of CPS based on CDL, which provides the functions of system description generation, verification and installation. At last, two system examples are implemented to prove that the amount of code written by developers could be reduced and the efficiency of programming could be improved to a certain extent by using CDL.

Key words: cyber-physical system; actor-oriented; system description; description validation; graphical edit; XML

信息物理系统 (CPS, Cyber-Physical System) 是计算和物理过程的整合, 嵌入式计算机和网络通过反馈回路监视和控制物理过程, 在反馈回路中, 物理过程与计算过程相互影响[1]. 信息物理系统被称为下一代计算革命, 将为人类社会带来巨大改变[2], 可以广泛应用在医疗健康系统, 交通安全控制, 环境控制, 智能电网, 分布式机器人, 智能家居等领域等[3]. 一般的 CPS 实例可以采用图 1 的结构进行描述[4], 系统中包含了多个计算组件、传感器组件和执行器组件, 组件之间通过网络进行交互. 传感器是一种嵌入式设备, 用于监测物理环境的状态, 比如声音, 光照等; 计算组件是决策控制单元, 负责接受传感器数据, 并实现对应的控制逻辑, 根

① 基金项目: 国家“核高基”科技重大专项 (2014ZX01029101-002)
收稿时间: 2017-01-25; 修改时间: 2017-02-15; 采用时间: 2017-02-23

据传感器的数据, 决定发送何种指令给执行器; 执行器是嵌入式设备, 接收到指令后, 执行对应的动作, 从而影响物理环境的状态, 形成反馈循环.

CPS 是异构混合系统, 将计算、通信和物理动态过程融于一体. 如何对 CPS 系统进行描述是一个重要的问题. 要对 CPS 的系统进行描述, 首先要构建 CPS 的的系统模型, 文献[5]提出了一种面向参量(actor-oriented) 模型, 对嵌入式硬件和软件系统进行建模.

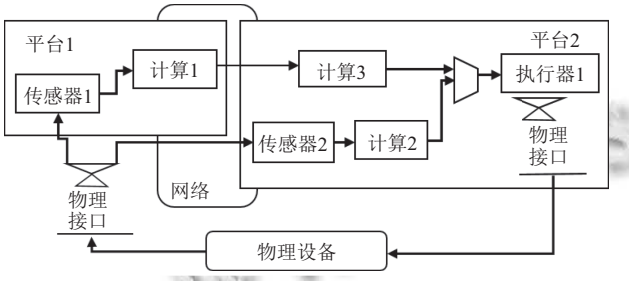

图 1 CPS 结构实例

基于面向参量模型, 本文提出了一种基于组件的信息物理系统描述语言 CDL(CPS Description Language). CDL 是对面向参量模型的一种实现, 提供对 CPS 系统的静态描述, 将参量所表示的计算组件、传感器和执行器都封装成具有统一抽象的组件. CDL 描述了系统中包含的组件, 组件之间的关联和系统约束. 此外, CDL 提供了系统描述在组件可达性方面的模拟验证, 一定程度上保证了系统的正确性. 本文的第 1 节介绍了相关研究, 第 2 节介绍了 CDL 的设计, 第 3 节介绍了 CDL 的实现, 第 4 介绍了 CDL 的验证, 第 5 节介绍了 CDL 的应用, 第 6 节介绍了 CDL 应用的实验结果, 第 7 节总结了 CDL 的优势与不足.

1 相关研究

CPS 的关键问题不是物理和计算的组合, 而是需要一种抽象能够同时包容两者[1]. 而现有计算模型大多仅能建模系统的部分行为和特性, 如连续时间模型主要用于建模化学反应、机械运动等过程, 离散事件模型 DE[6]主要用于建模电子电路、队列系统等具有离散动态的系统. 对于异构混合系统, 重点是软件和物理动态的混合建模, 文献[7,8]基于面向参量模型, 提出了一种建模 CPS 的模型 PTIDES, 证明了通过面向参量模型建模 CPS 的可行性与合理性.

介绍参量模型之前, 首先说明参量的概念. 在面向参量模型中, 系统中的计算组件和物理组件被建模为参量. 如图 2 所示, 一个参量是一个具有输入端口和输出端口的盒子, 在参量的输入端口, 按照时间顺序, 接受一系列时间标记为 t 的事件 e, 每个事件 e 指示该参量应该在逻辑时间 t 执行的动作; 在参量的输出端口, 产生输出事件, 发送给下一个参量. 一个系统可以被描述为多个参量相连构成的系统模型.

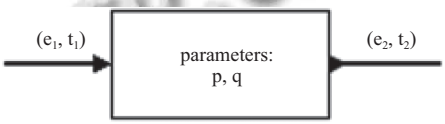

图 2 参量模型

文献[5]对面向参量模型进行了阐述. 面向参量模型是一种基于组件的模型, 适合建模分布式嵌入式硬件和软件系统. 被称为参量的组件具有定义明确的组件接口, 接口封装了参量的内部状态和行为, 并限制参量如何与其环境交互. 接口包括参量之间通信的端口, 以及用于配置参量操作的参数, 端口之间的信道用于传递事件. 模型, 参量, 端口, 参数, 信道等概念共同构成了面向参量模型.

面向参量模型有很多种实现方法, 如通过图形来表示, 通过 XML 来表示, 或者通过特定的 API 编程实现等. 工具 Ptolemy[9]是用于对嵌入式系统进行建模仿真的工具, 同时提供了这三种表示方式. 经调研发现, 首先, Ptolemy 要求用户了解硬件和软件组件的内部实现, 对开发者的嵌入式硬件和软件开发的知识要求较高; 其次, 该工具提供的 XML 描述中, 元素和属性繁多, 组件和组件之间的关系的描述没有分离, 词法语法复杂; 并且 Ptolemy 只提供建模仿真的功能, 并不提供生成系统实现的所需的代码或数据的功能. 我们希望能实现以下目标: ① 隐藏组件复杂的内部实现, 用户只需要选择封装好的、符合要求的组件使用; ② 提供尽可能简单的系统描述语言; ③ 通过系统描述直接生成系统配置安装所需的数据, 从而实现系统的自动安装, 降低系统设计实现的难度. 综上, 本文没有使用 Ptolemy 工具, 而是基于同样的模型——面向参量模型, 实现了 CPS 描述语言 CDL, 通过 CDL 来描述系统, 一定程度

上降低了系统设计的难度, 并实现了的其他工具与 CDL 配合使用, 实现系统的自动安装.

基于参量模型, 将参量表示的计算组件、传感器和执行器都封装成统一抽象的组件, 组件之间通过端口相连, 以统一的格式发送事件. 由参量构成的系统, 根据继承抽象, 具有同样定义良好的外部接口. 基于组件实现了一种 CPS 静态描述语言 CDL, CDL 中包含的组件表示构成系统的参量, 关联表示组件之间发送事件的方向. CDL 还描述了系统中的循环约束和超时约束, 其中, 循环约束用于描述系统中组件循环执行的情况; 超时约束指的是, 在现实世界中, 事件的产生和处理产生有一定的时间限制, 错过了截止期的事件可能在处理过程中夭折[10], 比如网络等原因造成比如网络等原因造成的延迟, 导致事件到达处理它的组件时, 以及超过了时间限制, 那么这种事件不应该被调度执行.

2 CDL 设计

CPS 系统包括物理部分和计算部分, 其中物理部分是由传感器等嵌入式物理组件构成, 而计算部分是由软件组件构成, 基于面向参量模型, 设计实现了 CPS 系统描述方法 CDL. CDL 是面向系统开发者的、描述了系统的组成、结构和约束的标记语言, 通过扩展 XML[11]实现. 参考面向参量模型, CDL 中将参量描述为组件, 将参量之间通过端口相连的关系描述为组件之间的关联, 并添加了系统在时间维度的超时约束. 由以上三方面的描述组合成对一个 CPS 系统的描述. 下文简单介绍 CDL 的关键内容.

① 组件. CPS 系统一般由传感器、执行器和计算组件构成, 在 CDL 中, 它们被统一描述为称为组件 (component). 组件的内部实现对系统开发者来说是透明的, 开发者通过组件提供的接口调用组件的方法, 实现对组件的控制. CDL 中按组件在系统中的位置, 将组件分为开始组件、中间组件和终止组件三种类型, 分别对应传感器、计算组件和执行器. 开始组件一般为传感器或其他可以被物理环境触发而产生事件的组件, 这类组件是系统的启动点; 中间组件一般是实现控制逻辑的计算组件, 接收事件进行处理, 并产生事件发送给下一个中间组件或终止组件. 终止组件一般为执行器, 是最终执行动作的组件, 不会产生新的事件. 每个组件有一个或多个输入端口和输出端口, 在组件的输入端口, 接收事件; 在组件的输出端口, 产生新的事件, 做为下一个组件的输入. 组件之间通过端口相连的关系称为组件之间的关联.

② 关联. 在面向参量模型中, 参量之间通过端口相连. 在 CDL 中, 参量被封装成了组件, 组件之间通过端口相连的关系被描述为关联 (connection). 组件在输入端口接收事件进行处理后, 在输出端口产生新的事件, 发送给与之关联的其他组件. 关联是有向的, 每个关联可以表示一个一对一的连接, 或一个一对多的连接, 或一个多对一的连接, 而多对多的关联可以用多个一对多的连接或多个多对一的连接表示.

③ 约束. CDL 中实现的约束类型有超时约束和循环约束. 超时约束是对一个事件 e 从产生到处理完成的时间约束 t, 约束的范围是从产生 e 的组件 c1 的输出端口开始, 到需要处理 e 的组件 c2 的输出端口结束, 如果 e 从 c1 产生到在 c2 处理完成的时间超过了 t, 那么该事件将被抛弃, 不再处理; 循环约束用于描述系统中组件循环执行的情况, 描述了从该循环约束的起点组件到终点组件, 循环执行的次数或循环退出的条件.

3 CDL 实现

CDL 通过扩展 XML 实现, XML Schema 可以对 XML 文档定义进行约束, 下文通过 XML Schema 解释 CDL 的实现.

① 系统描述声明. 通过 cdl 元素来声明一个系统描述, 该标签是一个系统描述文档的根元素, 每个系统描述文档中只有一个根元素. cdl 元素必须的属性有系统名 name 和系统版本号 version, 这两个属性用于唯一标识一个系统描述文档; 可选子元素包括三个分别描述组件、关联和约束的列表, 列表元素见②. 系统声明的实现如下:

```
<xs:element name="cdl">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="list"/>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string"/>
    <xs:attribute name="version" type="xs:string"/>
  </xs:complexType>
</xs:element>
```

② 列表元素. 系统中对组件的描述, 对关联的描述, 对系统约束的描述, 分别用三个列表来表示. 列表用 list 元素表示, 每个 list 有一个 name 属性, 用来该表示 list 描述的内容的类型, 该属性的值域是{component, connection, rule}, 这三种类型描述的具体内容见③④⑤. 列表元素的实现如下:

```
<xs:element name="list">
  <xs:complexType>
    <xs:choice>
      <xs:element ref="component"/>
      <xs:element ref="rule" />
      <xs:element ref="connection"/>
    </xs:choice>
    <xs:attribute name="name"/>
  </xs:complexType>
</xs:element>
```

③ 系统中包含的组件的描述. 在 CDL 中, 每个组件用一个 component 元素元素表示, 该元素必须的属性包括唯一表示该组件的 name 属性; 必须的子元素有描述组件类型的 type 元素, 描述组件行为的 behavior 元素, 描述组件位置的 position 等. 其中 type 元素描述了组件是什么类型, 可类比于面向对象中的类; behavior 元素描述了组件接收到事件时执行的一组动作及参数, 包括 name 和 value 两个子元素, name 描述该组件接收到事件时执行的动作, value 包含了执行该动作需要的参数; position 元素描述组件是开始组件, 中间组件或终止组件. 组件描述的定义如下:

```
<xs:element name="component">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="type" type="xs:string"/>
      <xs:element ref="behavior"/>
      <xs:element name="position" type="xs:string"/>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string"/>
  <xs:complexType>
</xs:element>
```

④ 组件之间的关系的描述. CDL 中通过一组 connection 元素描述组件之间的端口的关联. 该元素必须的属性包括唯一标识该关联的 name 属性. 必须的子元素有一组 from 元素, 描述该关联的源组件; 一组 to 元素描述该关联的目标组件. 关联描述的定义如下:

```
<xs:element name="connection">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="from"/>
      <xs:element ref="to"/>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string"/>
  </xs:complexType>
</xs:element>
```

⑤ 系统中存在的约束的描述. CDL 通过一组 rule 元素来描述系统中的约束. 每个 rule 元素必须的属性包括唯一标识该约束的 name 属性; 必须的子元素包括, 一个 from 元素标识约束的起点, 一个 to 元素标识约束的终点; 一个 type 元素表示该约束的类型, 一个 value 元素描述该约束的参数. 目前约束描述的定义如下:

```
<xs:element name="rule">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="from"/>
      <xs:element ref="to"/>
      <xs:element name="value" type="xs:string"/>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string"/>
  </xs:complexType>
</xs:element>
```

以上是 CDL 中主要内容的概述, 为了节省篇幅, 在上面的例子中省略了对一些属性和标签的约束, 但不影响对本文的理解.

4 CDL 验证

CDL 在词法语法和组件可达性两方面提供正确性验证. 通过定义了词法语法约束的 XML Schema 来验证 CDL 的词法语法的正确性; 通过将 CDL 中的信息转换成图的数据结构, 设计算法来验证系统中组件的可达性.

CDL 通过扩展 XML 实现, 作为结构化的文档, 在交给 CDL 处理程序之前, 需要保证 CDL 处理程序能

正确地处理这份文档, 提取需要的信息, 因此需要对 CDL 文档进行词法语法的验证, 保证文档是格式良好的且有效的. XML Schema 可以规范 XML 文档的结构和数据类型, 因此根据 CDL 的定义, 编写了对应的 Schema 文档, 实现对 CDL 的词法语法验证.

将 CDL 中的组件集合视为节点集合 V, 组件之间的关联集合视为图中的边的集合 E, 可以将系统描述转换成一个有向图 G. 如果从组件 v1 到组件 v2 存在一条路径, 那么认为组件 v1 到组件 v2 是可达的. CDL 中的 position 元素描述组件是开始组件, 中间组件或终止组件, 系统中的开始组件构成了系统的启动点集合 S, 如果从 S 中的所有组件出发, 系统中的每个组件都是可达的, 那么认为系统是满足组件可达性的, 反之则不满足组件可达性. 系统不满足组件可达性表示系统中有一些组件不会被接收到事件而执行动作, 也不会产生新的事件, 因此这个组件也是多余的.

组件可达性的验证的实现过程是:首先解析 CDL, 获得组件, 关联的相关数据;之后根据这些数据将系统描述转换成一个有向图 G, 其中组件构成了图的顶点集合 V, 组件之间的关联是图的边集合 E; 最后通过构造开始组件集合的传递闭包, 检查传递闭包是否包含系统中所有的组件. 构造传递闭包的核心算法如下:

TRANSITIVE-CLOSURE(G)

$n = |G.V|$

let $T^{(0)} = \left(t_{i,j}^{(0)}\right)$ be a new $n*n$ matrix

for $i = 1$ to n

  for $j = 1$ to n

    if $i == j$ **or** $(i,j) \in G.E$

      $t_{i,j}^{(0)} = 1$

    else $t_{i,j}^{(0)} = 0$

for $k = 1$ to n

  let $T^{(k)} = \left(t_{i,j}^{(k)}\right)$ be a new $n*n$ matrix

  for $i = 1$ **to** n

    for $j = 1$ **to** n

      $t_{i,j}^{(k)} = t_{i,j}^{(k-1)} \vee \left(t_{i,k}^{(k-1)} \wedge t_{k,j}^{(k-1)}\right)$

return $T^{(n)}$

5 CDL 应用

笔者实现了提供 CDL 生成、验证、安装功能的开发工具, 做为 CDL 应用的开发环境, 系统架构如图 3 所示. 系统主要包括两个大的功能模块: ① 面向系统开发者的系统描述及验证模块, 主要由图形化描述工具 Arcobaleno 和 CDL 编译器构成. Arcobaleno 是笔者实现的图形化编辑器, 可以根据图形化设计生成 CDL. 系统描述生成与验证的流程是, 开发者通过 Arcobaleno 来设计系统, 系统设计在中间代码生成器生成 CDL, 在中间代码编译器中验证其正确性, 之后通过目标代码生成器生成目标语言, 存储在数据管理层; ② 面向系统用户的系统发布及安装模块 Vitamin, 用户可以从 Vitamin 上安装系统. 安装流程是, 用户在 Vitamin 上选择符合需求的系统描述, 确定系统安装的位置, 通过数据管理层调用组件管理层进行系统安装. 接下来的部分将介绍系统两个主要功能模块的实现.

5.1 系统描述生成及验证

因为 CDL 是基于组件对系统进行描述, 因此实现了基于组件的图形化编程工具, 让开发者通过图形化表示来描述系统. 目前已有的图形化编程工具主要分为两种, 一种是类似积木块码放的方式. MIT 的软件团队开发的 Scratch[12], 以及基于 Scratch 的思想实现的 Hopscotch[13]. 这种编码软件的缺点是, 当系统中存在多个对象时, 每一个对象的设计都在不同的页面进行模块搭建, 难以直观地把握系统整体的逻辑运算关系. 另一种是系统流程图方式, 常见的基于流程图的编码软件有 Illumination Software Creator 等, 缺点即系统搭建需要大量的连线, 导致操作复杂. 为了直观地展示系统的整体逻辑关系, 并降低操作的复杂性, 基于 Blockly 工具[14]设计实现了图形化编程工具 Arcobaleno, 以积木块表示系统中的组件, 以拼接积木块的方式, 来描述系统, 工具实现如图 4 所示.

系统描述及验证流程如图 5 所示. 首先, 开发者根据系统目标, 按照系统需求从组件库中选择符合需求的组件, 载入 Arcobanelo 中, 以拼接积木块的方式, 形成图形化的系统设计. 图形化的系统设计通过中间代码生成器生成器生成 CDL, 之后在中间代码编译器中进行验证, 对 CDL 文档和 CDL Schema 进行匹配, 验证 CDL 的词法语法是否正确; 组件可达性通过闭包算法进行验证. 经过验证的 CDL 用于生成目标语言.

5.2 系统安装

系统描述经过验证后, 生成目标语言, 存储在数据管理层, 并发布在网站 Vitamin 上. 用户可以从

Vitamin 上进行系统安装, 用户数据和系统配置数据被发送到数据管理层, 由数据管理层调用组件管理层自动进行系统安装. 至此完成了系统描述, 验证, 转换, 安装的过程.

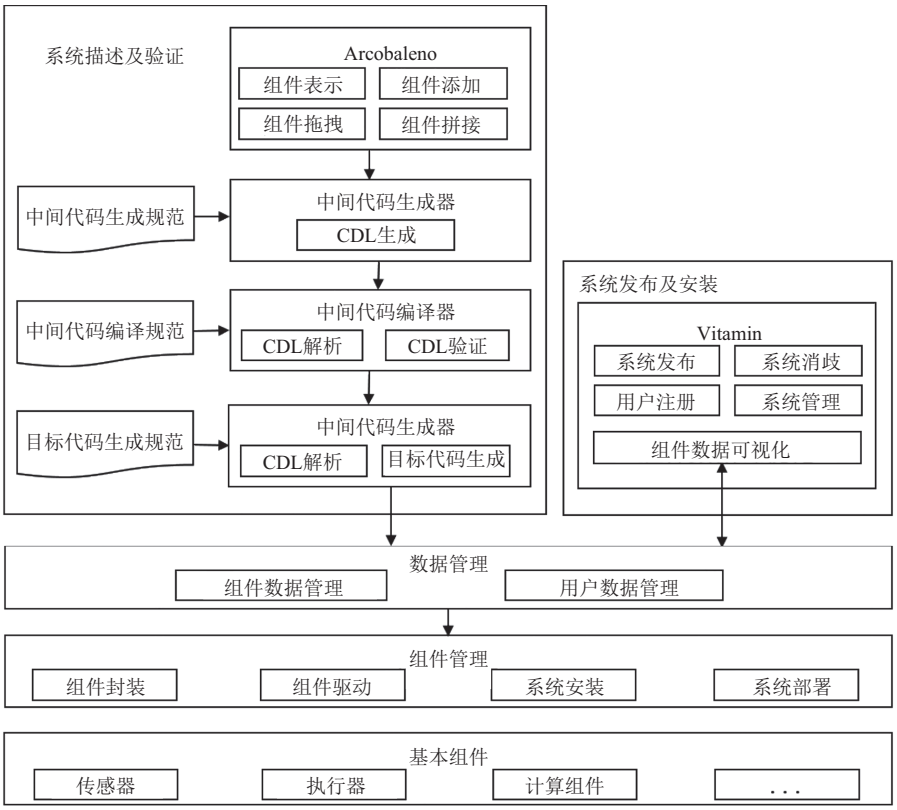

图 3 系统架构图

图 4 图形化工具 Arcobanelo

6 实验结果

通过以上开发环境, 实现了两个应用实例来验证 CDL 可以减少系统实现需编写的代码量, 提高开发效率. 在实验中, 将在 Arcobaleno 中使用的每个组件视为一个代码行, 通过统计在 Arcobaleno 和 CDL 以及以 python 和 html 等高级编程语言分别实现同一个应用所需编写的代码量, 对比通过这三种方式来实现同一个应用的难度. 结果如表 1 与表 2 所示. 表中除了最后一行数据, 每行数据分别是实现组件的代码行数, 最后一行的数据, 是实现各个组件以及组件之间的关系的代码总量, 且通过高级语言实现整个应用的代码量, 必然大于实现各个组件的代码量之和. 由于篇幅限制, 用 python/html 等高级语言实现的各个组件的代码见 https://github.com/virtdev/vdtools/tree/unstable/vdtools/drivers.

① 室内环境监测应用

通过 CDL 能够快速实现一个室内环境检测应用, 实时检测室内的环境变化, 可以广泛应用于检测家庭, 医院, 工厂等场所的环境数据. 参考室内环境检测的相关文献[14,15], 应用主要实现了对环境中温度, 湿度, 光照, 尘埃数据和室内图像数据的实时采集. 应用模型如图 6 所示, 主要分为两个计算平台, 平台 1 连接了温度湿度传感器, 光照传感器, 尘埃传感器, 摄像头等传感器组件, 中心电脑包括数据处理组件和可视化组件. 数据处理组件按照设定的频率读取传感器的数据, 发送给可视化组件, 可视化组件对采集到的数据进行展示.

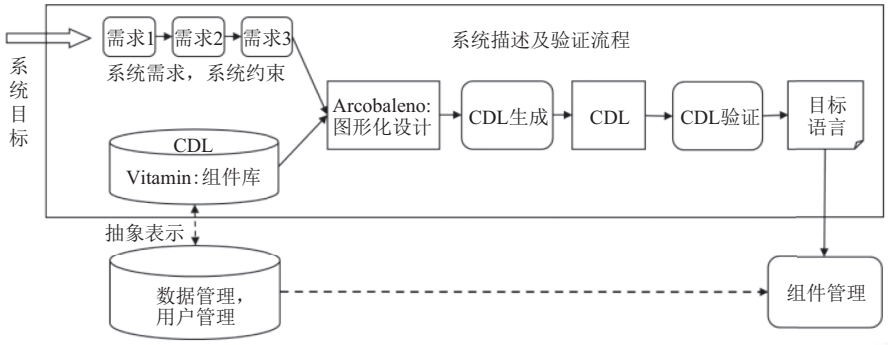

图 5 系统描述及验证流程

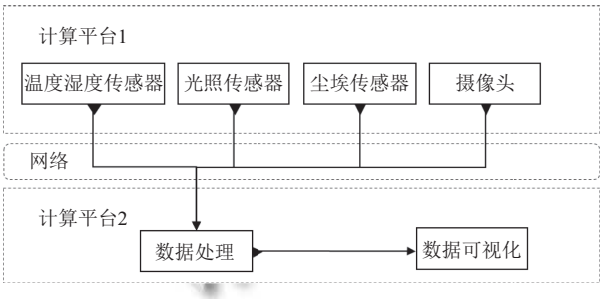

图 6 室内环境检测应用参量模型

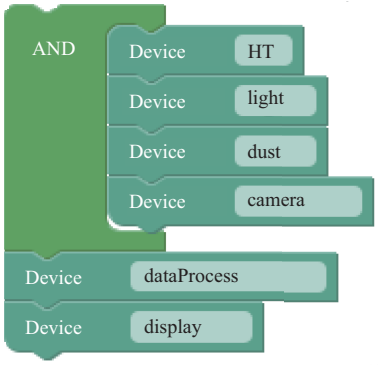

图 7 室内环境检测应用图形化表示

根据该应用的面向参量模型, 通过 Arcobanelo 生成的 CDL 代码如下:

```
<?xml version="1.0" encoding="utf-8"?>
  <cdl name="EnvironmentMonitoring" version="0.0.1">
    <list name="component">
      <component name="HTID0">
        <type>HT</type>
        <behavior>
          <name/>
          <value/>
        </behavior>
        <begin_end_type>sc</begin_end_type>
      </component>
      <component name="lightID0">
        <type>light</type>
        ...
      </component>
      <component name="dustID0">
        <type>dust</type>
        ...
      </component>
      <component name="cameraID0">
        <type>camera</type>
        ...
      </component>
      <component name="AndFinishID0">
        <type>VDev</type>
        ...
      </component>
      <component name="dataProcessorID0">
        <type>dataProcessor</type>
        ...
      </component>
      <component name="displayID0">
        <type>display</type>
        ...
      </component>
    </list>
    <list name="connection">
      <connection name="con_0">
        <from name="HT0"/>
        <from name="lightID0"/>
        <from name="dustID0"/>
```

```
        <from name="cameraID0"/>
        <to name="AndFinishID0"/>
      </connection>
      <connection name="con_1">
        <from name="AndFinishID0"/>
        <to name="dataProcessorID0"/>
      </connection>
      <connection name="con_2">
        <from name="dataProcessorID0"/>
        <to name="displayID0"/>
      </connection>
    </list>
    <list name="rule"/>
  </cdl>
```

通过 Arcobanelo, CDL 和高级语言分别实现该室内环境应用的代码量对比见表 1.

表 1 室内环境监测应用代码量对比

组件	代码		
	Arcobanalo	CDL	Python/html
温度湿度传感器	1	8	77
光照传感器	1	8	57
尘埃传感器	1	8	63
摄像头	1	8	39
数据处理	1	8	85
数据可视化	1	8	180
室内环境检测系统	7	79	>501

② 人脸识别应用

人脸图像具有唯一性和稳定性, 人脸识别可广泛应用于视频监控、日常考勤等场景, 作为身份验证的手段. 参考论文[16,17], 设计实现了一个简单的人脸识别应用, 如图 8 所示. 平台 1 的红外传感器检测到人的靠近, 摄像头采集用户的脸部图片, 经过平台 2 的人脸识别软件解析识别, 将识别结果发送到可视化组件, 对结果数据进行可视化. 在摄像头发送数据, 到人脸识别组件产生结果的过程添加了一个 3 秒的超时约束, 如果因为网络等原因, 从摄像头发送数据给人脸识别组件到可视化组件接收到识别结果数据的过程超过 3 秒, 数据可视化组件将会提示超时. 其中人脸识别组件通过封装 openCV 的人脸识别算法实现. 通过 Arcobanelo 表示该系统的图形化设计如图 9 所示.

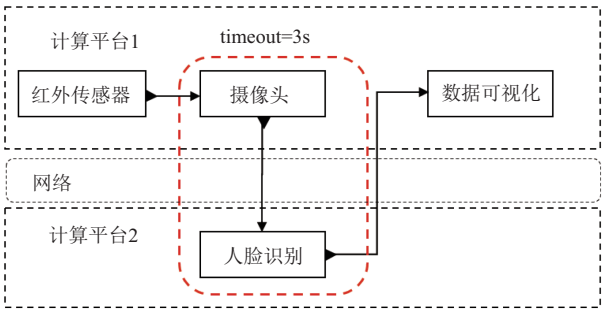

图 8 人脸识别应用参量模型

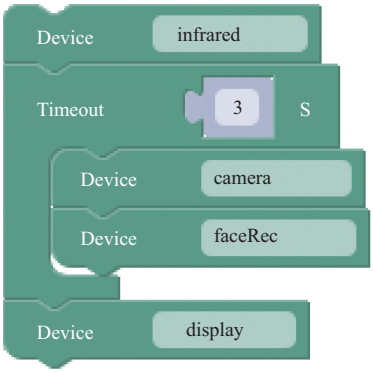

图 9 人脸识别应用图形化表示

根据该应用的面向参量模型, 通过 Arcobanelo 生成的 CDL 代码如下:

```
<?xml version="1.0" encoding="utf-8"?>
  <cdl name="FaceRecognition" version="0.0.1">
    <list name="component">
      <component name="infraredID0">
        <type>infrared</type>
        ...
      </component>
      <component name="cameraID0">
        <type>camera</type>
        ...
      </component>
      <component name="faceRecID0">
        <type>faceRec</type>
        ...
      </component>
      <component name="displayID0">
        <type>display</type>
        ...
      </component>
```

```
    </list>
    <list name="connection">
      <connection name="con_0">
        <from name="infraredID0"/>
        <to name="cameraID0"/>
      </connection>
      <connection name="con_1">
        <from name="cameraID0"/>
        <to name="faceRecID0"/>
      </connection>
      <connection name="con_2">
        <from name="faceRecID0"/>
        <to name="displayID0"/>
      </connection>
    </list>
    <list name="rule">
      <rule name="timeoutID0">
        <from name="cameraID0"/>
        <to name="faceRecID0"/>
        <value>3</value>
      </rule>
    </list>
  </cdl>
```

通过 Arcobanelo, CDL 和高级语言分别实现该人脸识别应用的代码量对比见表 2.

表 2 人脸识别应用代码量对比

组件	代码		
	Arcobanalo	CDL	Python/html
红外传感器	1	8	54
摄像头	1	8	39
人脸识别软件	1	8	48
超时约束组件	1	8	50
数据可视化	1	8	180
人脸识别系统	5	58	>371

从实现以上应用的代码量数据可以看出, 通过 Arcobanelo 和 CDL 描述来实现应用, 与直接通过编程语言实现应用, 大大减少了开发者需要要编写的代码量, 降低了实现的难度. 通过这种基于组件的描述方法, 开发者能够快速设计实现一个 CPS 应用, 提高开发效率.

7 结束语

本文根据面向参量的系统模型的思想, 提出了基于组件的信息物理系统的描述方法 CDL, 并提供了 CDL 在组件可达性方面的模拟验证. 实现了 CDL 生成、验证、安装的验证环境, 并通过该环境实现了应用实例. 从实验结果可以发现, 开发者通过 CDL 能更简单地描述需要实现的系统, 减少自己实现的代码量, 提高编程效率, 降低系统设计实现的难度.

目前 CDL 还存在一些不足, 接下来将针对以下不足继续研究: ① CDL 中对 CPS 在时间维度的描述能力有限, 目前对时间维度的描述包括超时约束, 超时约束用于描述系统中组件之间在顺序执行时, 应该满足的时间限制, 超时约束是否足够描述 CPS 的时间维度还需继续研究. ② CDL 验证目前包括词法语法验证和组件可达性验证, 接下来会在时间约束的基础上, 加入对用户添加的时间约束的合理性的验证. ③ 词法语法的修改完善, CDL 基于 XML 实现, 接下来会继续修改其词法和语法, 使其更加简洁规范.

参考文献

1 Lee EA. CPS foundations. Proc. of the 47th Design Automation Conference. Anaheim, California. 2010.

2 Rajkumar R, Lee I, Sha L, *et al*. Cyber-physical systems: The next computing revolution. Proc. of the 47th ACM/IEEE Design Automation Conference. Anaheim, CA, USA. 2010.

3 Lee EA. Cyber physical systems: Design challenges. Proc. of 2008 the 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing. Orlando, FL, USA. 2008.

4 Lee EA, Sanjit SA. Introduction to embedded systems: A cyber-physical systems approach. Berkeley: Lee and Seshia, 2011.

5 Lee EA, Neuendorffer S, Wirthlin MJ, *et al*. Actor-oriented design of embedded hardware and software systems. Journal of Circuits, Systems and Computers, 2003, 12(3): 231–260. [doi: 10.1142/S0218126603000751]

6 Cassandras CG. Discrete event systems: Modeling and performance analysis. Homewood, IL: CRC Press, 1993.

7 Zhao Y, Liu J, Lee EA. A programming model for time-synchronized distributed real-time systems. Proc. of the 13th IEEE Real Time and Embedded Technology and Applications Symposium. Bellevue, WA, USA. 2007.

8 Derler P, Lee EA, Matic S. Simulation and implementation of the PTIDES programming model. Proc. of the 2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications. Vancouver, BC,

Canada. 2008.
9 Davis J, Hylands C, Kienhuis B, *et al*. Heterogeneous concurrent modeling and design in Java. Technical Memorandum UCB/ERL M01/12, California, USA: University of California at Berkeley, 2001.
10 鲁静, 张晶. 基于 PTIDES 执行策略的调度算法. 计算机工程, 2011, 37(18): 258–259, 263. [doi: 10.3969/j.issn.1000-3428.2011.18.086]
11 Textuality TB, Paoli J, Sperberg-Mcqueen CM, *et al*. Extensible Markup Language (XML) 1.1. 2nd ed. 2006.
12 Resnick M, Maloney J, Monroy-Hernández A, *et al*. Scratch: Programming for all. Communications of the ACM, 2009, 52(11): 60–67. [doi: 10.1145/1592761]
13 Gourlay AR. Hopscotch: A fast second-order partial differential equation solver. IMA Journal of Applied Mathematics, 1970, 6(4): 375–390. [doi: 10.1093/imamat/6.4.375]
14 Fraser N. Blockly: A visual programming editor. https://code.Google.com/p/blockly. [2013].
15 贾鹏辉. 基于 ARM 的智能家用空气质量检测系统的研究[硕士学位论文]. 淮南: 安徽理工大学, 2015.
16 王浩云, 刘佼佼, 侯思宇, 等. 信息物理系统 (Cyber-Physical System) 时空建模方法及在温室控制中的应用. 农业工程学报, 2015, 31(15): 183–190. [doi: 10.11975/j.issn.1002-6819.2015.15.025]
17 潘芬兰. 基于人脸识别技术的智能考勤系统研究[硕士学位论文]. 杭州: 浙江大学, 2014.
18 郎利影, 魏娜. 嵌入式人脸识别考勤系统设计与实现. 煤矿安全, 2012, 43(4): 68–70.