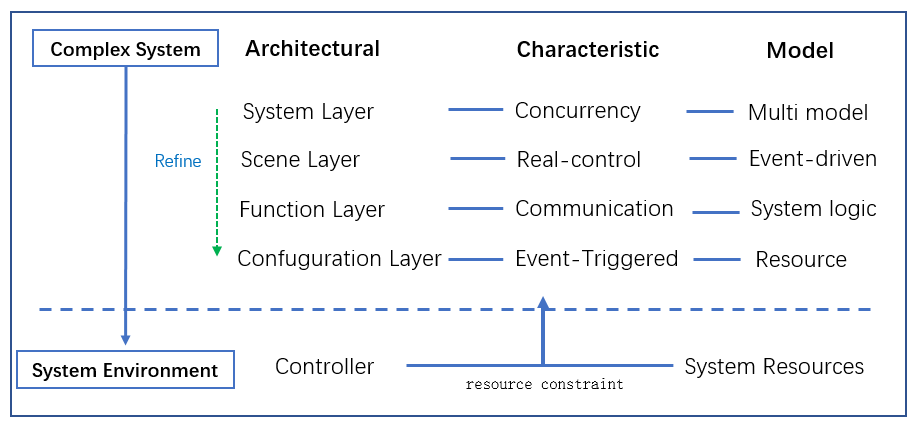
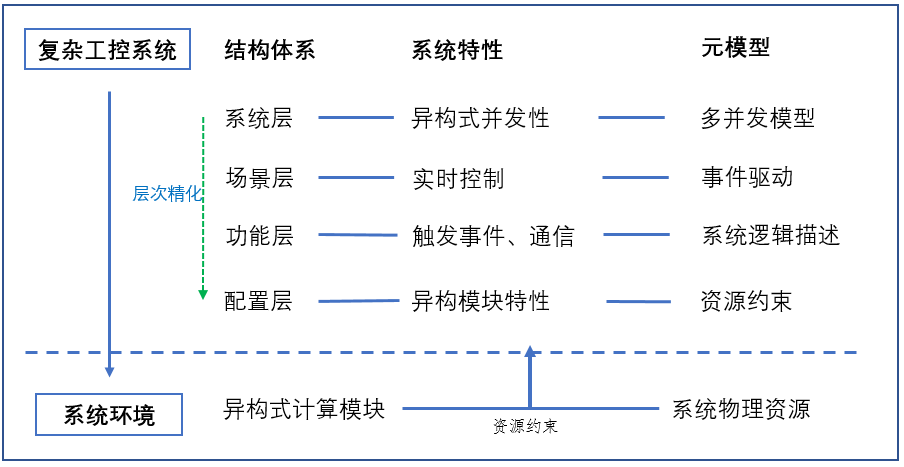
IMCL转61131-3规则

1. **IMCL建模**

**（1）IMCL 模型**

IMCL建模语言设计的目的在于对实际的工控领域系统进行建模。对复杂的工业系统的建模有不同的刻画角度。IMCL所采用的是一种基于精化思想，层次化地对系统功能、控制逻辑、系统资源等进行建模。下图表示的是使用 IMCL 简历一个复杂工控系统模型其系统结构，系统特性以元模型特点。



1. 系统层： 系统层体现的是一个模型直观构成，通常系统都是有模块化或功能化组成。对于工控领域异构式系统而言，其内部具备计算控制能力的控制器或处理器都是可相互独立运行的，从系统整体行为来看其运行过程是高度并发的。

2. 场景层： 场景层描述的主要是系统中独立的组件之间共同运行时的逻辑关系，即控制流程和交互规则。所有场景都是由系统特定的任务执行顺序、事件触发、消息传输等来共同体现的。一个系统的操作过程，可以看作是场景的变化，也可以看作是系统内部的交互过程。在IMCL的建模过程中，场景层对应的是系统中的事件。

3. 功能层： 功能层对应的是系统的行为过程。相对与场景层，功能层是可以看作是场景层的精化，主要是刻画场景层里不同场景的实现细节过程，包括场景中发生的各类消息的请求、数据计算和交互，以及控制器和设备之间存在的调度关系。功能层对应场景层里的不同场景，一步步精细地刻画系统特征，包括组件之间的通道、事件触发后的任务执行流程。

4. 配置层： 配置层反映的是系统中具备控制计算能力的计算单元与系统物理资源之间的映射关系，在本文中我们用资源约束来刻画该关系。每个计算单元都能只能够控制和调度特定的物理资源，这种制约关系的其实描述的就是不同处理单元各自的功能。通过配置这种资源约束关系，不仅可以更加真实地反映复杂系统的内部结构，更是后续的复杂模型拆分生成群体模型的重要准则。

整个系统建模的过程主要包括以下3个方面：

（1）资源统一定义: 系统环境包括系统运行时所需的物理资源。基于不同的工业环境，物理资源是多样式的，例如传感器、可控设备或其他设备资源，他们之间存在着客观的差异性，在不同的控制器上其驱动方式、软件库等都是不一样的。考虑到它们对整个系统影响的实现细节不同，为了能够统一刻画他们的功能，在本研究中，将其具体的实现方式高度抽象，使用统一的建模方式来描述其系统资源，把这些资源描述为统一定义的变量集合。

（2）工业控制系统建模: 结合具体的工业控制系统，其系统的本质可以看作是一个由特定数量的功能事件的集合，包括输入输出、逻辑控制等。与物理资源的统一定义类似，在建模过程中，可以会将这些系统行为看作是不同的事件。每个事件则是有不同的最小执行任务单元所构成。

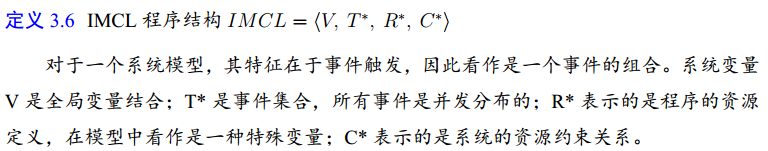
（3）资源约束: 系统环境除了包括物理设备资源，还包括核心的处理控制模块。这些异构计算处理单元与设备之间的控制关系存在调度控制约束的。如果一个系统太复杂，资源调度关系存在多样性，那么通常情况下系统是不可能由一个计算单元就实现所有的任务。系统的实现是需要多个计算单元来相互协同的。因此我们需要在模型中清晰地定义异构式系统中计算单元和物理资源之间存在约束关系。

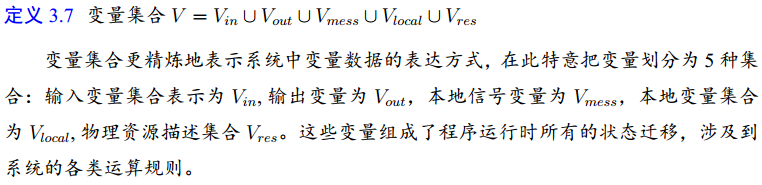
**（2）形式化语义**

为了更好的理解IMCL模型的特点，接下来我们从形式化角度来介绍模型的特点。对于任意的系统，从系统触发事件角度用 IMCL 语言进行建模，其系统模型可以表示为如下形式：

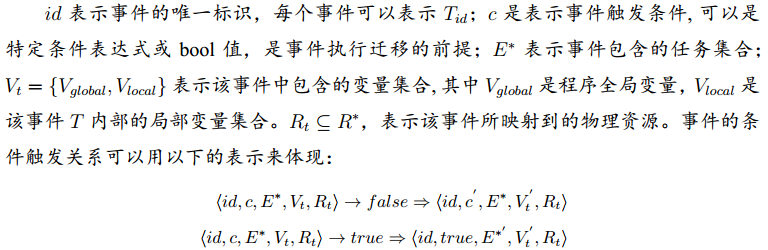


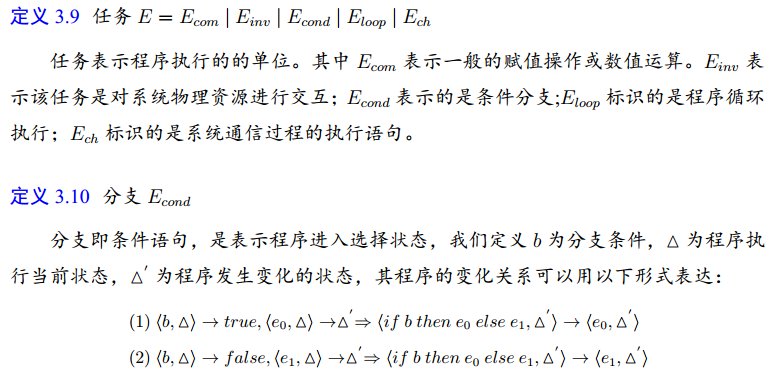
从模型精化地思想来看，每个系统Prog程序可以看作是触发事件的集合，然后每个触发事件Ti是一组有序的命令表达式，每个命令表达式又可以看作是系统执行的最小计算单元。

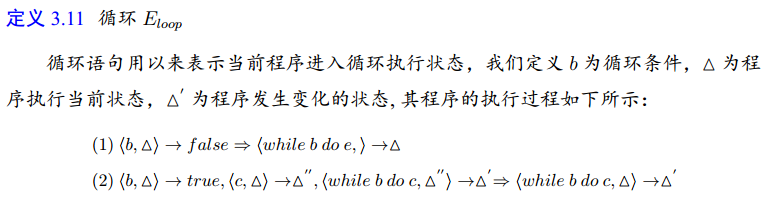




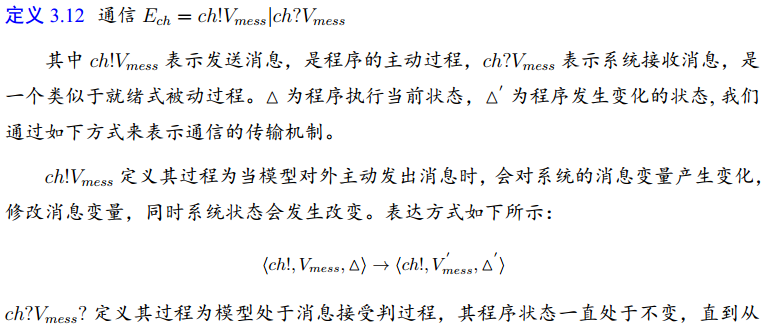


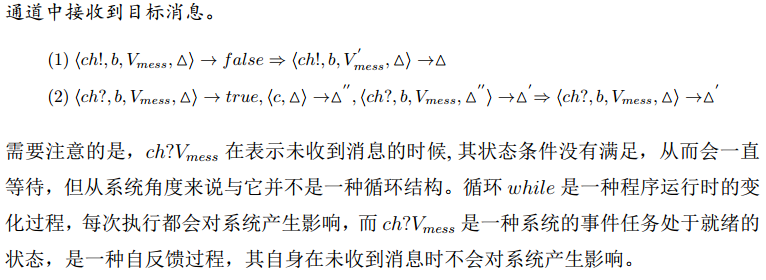


Econd = <cond, ，>



>= <cond, > = / >= <, ch!, ch?>





**定义3.13** 调度= <, ε, Dev>

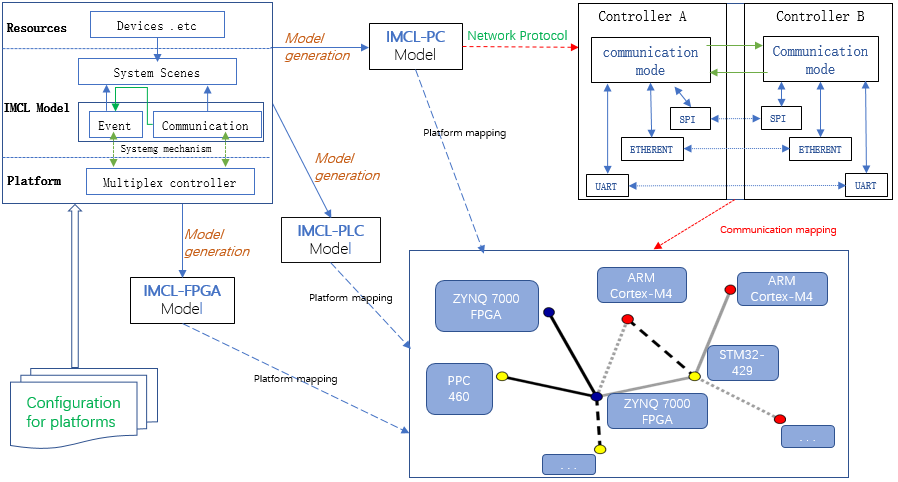
资源调度反映的是控制器和物理资源之间的调度关系，ε有两种类型“<<Dev”和“>>Dev”。“<<Dev”表示控制器向物理设备调度获取数据；“>>Dev”表示控制器向物理设备调度传输数据。由于IMCL模型的目的是研究系统的逻辑功能，所以我们用ε的这两种形式来描述控制器和物理设备之间的调度功能。

（3）群体模型生成

群体模型生成技术的具体实现细节在我们之前的发表的论文中已经提出。使用ICML建模方法的优势在于，我们可以将一个复杂模型在给定的资源与控制器约束条件前提下，智能化地拆分成多个子模型。子模型之间能够相互通信协同，实现与原始模型一致的功能。生成的子模型分别对应具体的目标平台，and本文的主要研究工作是实现看IMCL模型到不同目标平台上的代码生成工作。

1. **Approach**

**介绍框架图**



在上一节我们介绍了IMCL表示复杂系统的模型方法以及模型形式化定义。在此基础上，我们将介绍如何从群体IMCL模型来生成目标代码。

如图所示，是从IMCL模型到多种目标平台代码生成的原理图。IMCL模型在给定资源约束的前提下，可以生成异构式系统模型。在给定具体的目标平台配置信息后，通过制定代码生成规则，自动化地实现模型到对应平台的代码生成。

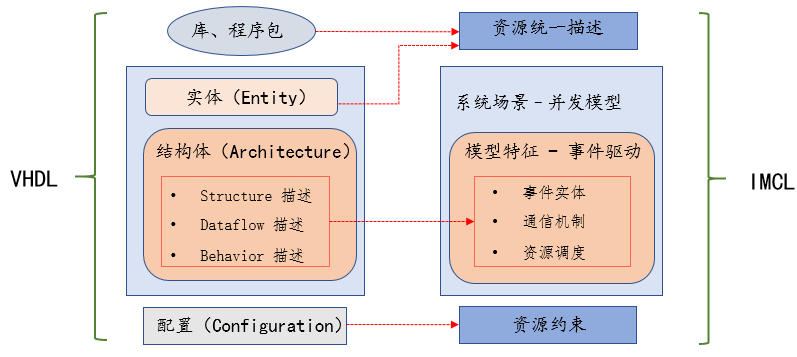
1. **模型与不同控制器程序转换研究**

我们针对不同的平台，包括FPGA、PLC、PC来进行研究。研究主要包括如何用IMCL表示这些异构系统。

1. **FPGA系统与IMCL模型转换**

FPGA(Field－Programmable Gate Array,现场可编程门阵列)由于其可定制的特点，被广泛应用在医疗设备，轨道交通控制等领域。FPGA所使用的VHDL语言开发的系统，其架构包括以下几个基础部分：

1. Library声明区：声明程序所需使用的资源库，包括标std、work 和用户定义库等。Library包含了多样的设计单元，从程序角度来说，可以看作是一个数据的集合。
2. Use声明区：该部分和Library相关，所声明的是对Library中对应资源库所使用的具体资源。
3. Entity声明区：该区是VHDL程序的实体声明，主要描述系统电路的输入、输出和端口之间关系。
4. Architecture架构：架构是电路的行为部分，架构支持程序的并行和串行，主要描述的是其内部的实现过程，包括数据流（Dataflow）、结构描述（Structure）、行为描述（Behavior）等。
5. Configuration配置：配置的主要目的是从库中选择所需要的单元，组成所需要的系统。从系统资源角度来说，是一个选择和组合的过程。



结合之前分析的IMCL模型特点，我们可以看到两种架构之间有共同之处的，因此当忽略些无关的平台细节后，IMCL可以对VHDL所描述的行为进行建模。如图所示，在忽略平台相关性后用IMCL建模的过程中，VHDL中表示的库和程序包结构，可以抽象成资源。实体表示的是系统电路结构的设计图，也可以用作是资源来描述。VHDL中的架构主要包括结构描述、数据流描述、系统行描述，本质上都是描述内部结构的功能特征，对应的在IMCL中用事件来描述。

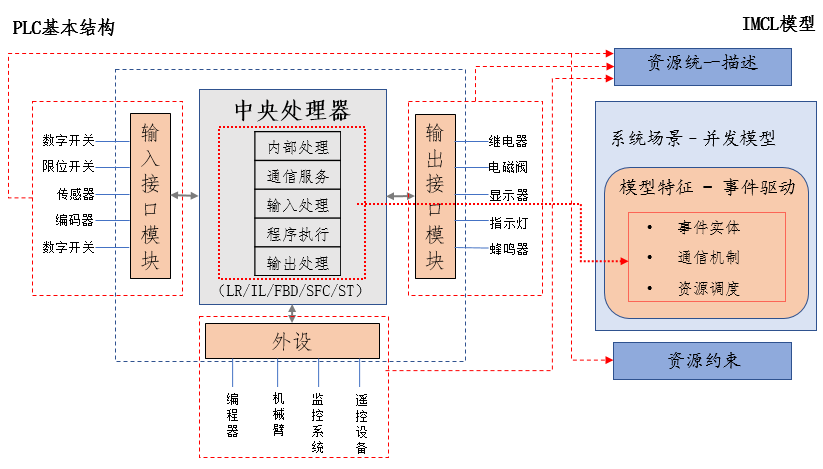
1. **PLC系统与IMCL模型转换**

PLC(Programmable Logic Controller，可编程逻辑控制器)是一种可编程控制的工业控制计算机，PLC以微处理器为核心，通过软件实现对系统的控制。PLC种类繁多，但其结构原理都基本相同：包括处理器，存储、I/O口、网络通信等部分。以PLC开发的语言61131-3语言为例，其设计语言包括五种形式：

* 梯形图语言（LD）：采用图形化设计方式来描述其内部逻辑的控制关系。
* 指令表语言（IL）：类似于汇编语言，是一种命令式程序，通过操作码和操作数结合的形式来控制系统的执行过程。
* 功能模块图语言（FBD）：该语言是采用模块化的设计思想来刻画系统的逻辑控制流程。不同模块高内聚、低耦合，可以清晰地表达系统的功能。
* 顺序图表语言（SFC）：其特点在于采用step和条件转换关系（while、if），能够按照系统执行过程顺序地设计系统。
* 结构化文本语言（ST）：ST一种结构化的高级语言，常用来控制目标系统的程序过程。ST可以完成复杂的控制运算，从最精细的层次刻画系统。

上述5种语言是PLC的主要编程语言，目前主流的PLC对这五种语言都会同时支持。如图所示，我们可以使用IMCL来表示PLC的运行模式，PLC的工作阶段是周期性地循环扫描，在无中断等情况发生时会周而复始的工作下去。通用的 PLC运行模式可以分为5个阶段：

* 内部处理阶段：检测系统当前就绪状态，对内部定时器复位；
* 通信服务阶段：PLC带有通信功能，外部控制模块可以相互通信，也可以接受到其他控制器的信号命令。
* 输入处理阶段：读取挂载外设的信息数据，一次性把数据采样到系统中。
* 程序处理阶段：该阶段是PLC控制过程的核心阶段，是PLC程序主体：包括条件控制、数值计算、逻辑转换等。该阶段反映的正是系统的功能行为特点。
* 输出阶段：主程序运行结束后，通过输出机制把数据加载到外部。

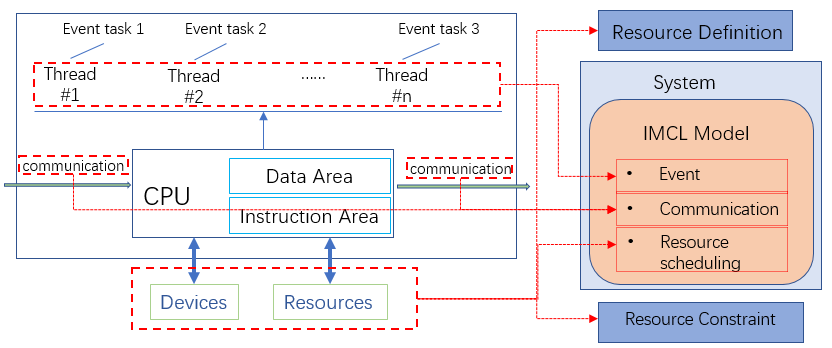


输入和输出模块、外设等关联的外部物理资源的统一描述，抽象资源对象为程序变量，可以便于资源调度以及设置约束条件。对于 PLC 的主体程序，我们提取主体部分的通信服务、程序执行过程，然后使用事件驱动的方式来描述。最终，可以通过IMCL模型来描述PLC应用。

1. **PC系统与IMCL模型转换**

https://blog.csdn.net/lu\_embedded/article/details/50907378

PC，个人计算机，由于其具有高速的处理速度、可靠的操作平台、海量存储、网络化、友好的人机交互等优点，因此也常常被广泛适用于工业系统中。例如PCBCS，PC能够其他PC或主流的PLC等系统进行通信协作实现复杂的功能需求。PCBCS中，PC的通信技术是其最大的优势之一，PC能够兼容主流的几乎所有的通信协议，因此非常有利于与应用于复杂系统的设计上。常见的PC系统设计程序语言是C，由于其备良好的兼容性、可移植性、执行效率高等特点，被广泛应用工业级系统设计。



如图所示，一个典型的PC形式的控制系统设计可以表现如上图所示，其中CPU负责程序的执行，而整个系统是由多个独立的线程组成的，每个线程代表的是相关的任务。系统具备独立的通信，包括数据的输入与输出。当我们使用IMCL来进行建模时，我们把多线程表现为并发的触发事件集合；系统功能的通信可以使用IMCL的抽象通信协议来表示；系统与外界设备资源的控制关系，使用IMCL中的资源调度来建模。

**B、代码生成配置**

系统模型本质是抽象掉一些无关细节信息，只关注研究对象特征的一种研究方法。因此，当我们希望能够从模型自动化生成代码时，我们需要对缺失的细节信息进行补充。从IMCL到具体目标平台的代码生成过程中，需要补充的内容包括变量转换，系统之间的通信协议方法以及控制器和特定设备的驱动关系。在此我们使用配置表示这些内容：**Conf = <Vmap，Cmap, Dmap>**

（1）Vmap表示模型中变量与指定控制器中变量映射：

Vmap = Vimcl -> （Vplc|Vfpga|Vpc）

其中Vimcl指的是IMCL中的变量，而Vplc | Vfpga | Vpc指的是具体不同平台的变量定义表示

（2）Cmap表示模型中通信方法和特定平台使用的通信协议之间映射关系： Cmap = Cimcl -> （Cplc|Cfpga|Cpc）

其中Cimcl=ch!Vmess|ch？Vmess，指的的是IMCL中通信的形式化表示。Cplc|Cfpga|Cpc指的是不同平台的具体通信协议的定义与实现。

（3）Dmap表示模型中驱动与指定控制器驱动程序映射关系。

Dmap = Dimcl -> （Dplc|Dfpga|Dsoc）

其中Dimcl=<<Dev|>>Dev，指的是IMCL中控制器与外围物理设备之间的调度关系，（Dplc|Dfpga|Dsoc）对应到具体的目标平台时需要实现设备调度驱动程序。

**三、代码生成**

IMCL模型特点具备事件触发、消息通信和资源调度等功能特点。本小节将从代码生成角度来介绍IMCL模型转换成PLC程序的转换规则。

代码生成的常用技术是基于AST来生成。抽象语法树（Abstract Syntax Tree）也称为AST语法树，指的是源代码语法所对应的树状结构。也就是说，对于一种具体编程语言下的源代码，通过构建语法树的形式将源代码中的语句映射到树中的每一个节点上。在IMCL树形结构基础上，通过给定的代码生成规则，可以实现模型到目标平台的代码生成。

**算法：基于AST的转换算法**

|  |
| --- |
| 输入：（1）IMCL code （2）rules of specific target platform  输出：Target platform code  过程：   1. 转换成AST树，通过开源工具ANTLR来转换 *AST<-antlr IMCLcode* 2. 广度遍历AST语法树结构，获取树中节点信息。 *Forall node \in Tree …* 3. 分析节点信息（GetTypeRule()），针对具体的目标平台实现具体转换。  * case Rule1：variableHandler() * case Rule2：eventHandler() * case Rule3：structHandler() * case Rule4：communicationHandler() * case Rule4：scheduleHandler() |

异构式多平台代码生成的第一步是将IMCL代码转换成结构化的语法树AST。接着，我们借助ANTLR工具，在给定的IMCL语法下生成IMCL模型的抽象语法树。然后通过深度遍历树形结构，对每个节点所属类型进行分析。最后，我们根据每个节点所属的类型，分别使用对应的规则来转换。

下面我们将介绍这些不同规则转换的具体细节：

**Rule1：IMCL变量转换**

IMCL程序的变量共有5种形式，这些变量覆盖了61131-3、VHDL、C的基础变量类型。由已经定义的Vmap = Vimcl -> （Vplc|Vfpga|Vsoc）可知，不同控制器程序中的所有变量都可以用全局变量、局部变量、静态变量来表示。针对每个控制器程序，转换成程序中特定的变量类型，如int、string等。

**Rule2：IMCL事件转换**

IMCL工控模型表现形式为事件的集合，在系统中事件是并发的。任意事件的触发条件得到满足后就会执行各自事件的主体。不同程序语言中的事件表达形式是不一样的，接下来将针对三种语言分别转换：

（1）在IEC 61131-3中，能够表示并发的只有顺序功能图标语言SFC。

定义：一个SFC程序被定义为一个元组SFC =<V,S∗,s0,L∗,pDv∗>，其中V表示变量集合，S∗ 表示步的集合，s0表示SFC的初始步，L ∗ 表示内部指令程序集合， pDv∗ 表示并发分支结构集合。

由上述定义，从事件驱动的角度来说，ICML所有事件都可以看作是SFC程序的分支，即每个pDV\*的分支都可以看作是IMCL的中的事件。因此我们把SFC作为描述整个系统结构的架构性语言。

在对于IMCL中具体触发事件的主体，我们使用61131-3中的ST语言来等价的刻画事件的细节逻辑。具体的规则细节见规则3、规则4和规则5。

（2）在VHDL中，程序是以*architecture*形式来表现的，一个程序中的多个事件可以使用process结构来表示。因为在architecture中，每个process就是一个程序块，而所有的程序块都是并行的，这就可以等价地用每个进程来表示一个事件。

定义：一个architecture中的process程序可以看作一个元组PROCESS=<name, V, cond\*,L\*>，其中name表示进程名字，令表示某个PROCESS结构，则V表示是该PROCESS的变量集合，cond\_\*表示触发条件，L\*表示的是内部顺序执行的程序集合。

由于VHDL程序中architecture可以看作是一个process的集合。中每个T\_i就是一个PROCESS。

（3）在C语言中，为了表示多个事件，以及事件之间的并发关系，我们采用线程的方式来表示事件。

定义：我们给每个线程添加一个触发变量。则所有的事件看作是一个元组Thread=<id，V\*, L\*>，其中id表示进程为唯一标识，V\*表示的是线程的数据空间，L\*标识的是线程的执行主体。

在C语言中，程序的主体可以看作是多个并行的线程，每个线程可以看作是事件任务。则一个多事件的IMCL模型可以转换成一个多线程的C程序。中每个T\_i就是一个Thread。

**Rule3：结构化语句转换**

定义：IMCL包含条件If语句。IMCL中一个Case语句被定义为一个元组<caseExp; caseEles∗>，其中caseExp表示case语句的表达式，caseElse定义为<caseValue; StmtList>，caseValue表示进入case分句的表达式值，L\*表示每个case分句执行的语句列表。

定义：IMCL包含循环语句。IMCL一个循环语句被定义为一个元组<whileExp; stmtList>，其中whileExp表示while语句中的循环条件stmtList表示每次循环执行的语句列表。

条件语句循环语句是通用编程语言里的通用结构，所有语言的条件结构和循环结构的语义都是等价的。因此IMCL和61131-3、VHDL、C语言在if和while结构都具有的共同特性，他们之间的转换是直接、等价的。

**Rule4：消息通信转换**

由于在模型中通信是抽象的，通信只包含通信方式、通信内容，而并不关注通信具体的实现细节。IMCL通过抽象通信的通讯模式，抽象化诸如UART、ENTHENT、SPI等协议，而不关注底层到底是怎么实现的。底层中包括通信不同链路，如网口、串口等。我们的研究中，由于我们并不需要关注通信的具体实现原理，而只关注通信过程。所有模型都可以通过消息绑定的方式，通过通讯模块将内容发送到通道中，需要接受该消息的子系统会将绑定的通道中的数据自动获取。因此在做实际的代码生成时需要针对设备之间具体的通信协议来实现通信功能。例如串口、以太网、无线通信等。在我们研究中，我们将主流的集中通信协议封装成函数接口，用户在生成代码时，只需要选择对应的接口就行。

针对之前提到的Cmap = Cimcl -> （Cplc|Cfpga|Cpc），在本研究中，我们定义通信规则下的接口如下：

|  |  |  |  |
| --- | --- | --- | --- |
|  | **PLC（61131-3）** | **FPGA（VHDL）** | **PC（C）** |
| UART | ST#UART | Entity#UART | C#UART |
| Entherent | ST#Entherent | Entity#Entherent | C#Entherent |
| SPI | ST#SPI | Entity#SPI | C#SPT |
| … | … | … | … |

特别地，由于各个控制器执行程序的方式不一样，所以在语言之间相互转化时要考虑程序语言的表达能力和方式。下面我们将介绍三种语言是如何对IMCL的消息通信进行转换的。

1. 在61131-3语言中，对于程序主体周期执行运行模式是从输入->处理->输出，是原子性连续不中断的。而IMCL的一段程序是连续的，可能在中间过程包含消息的输入和输出。因此为将IMCL中事件具体过程转换为ST语言，我们采用标记的方式来转换。具体原理是：将IMCL事件主体以通信语句为标记分成多个阶段，这样ST程序可以使用若干个执行周期去完成。
2. 在VHDL语言中，对于所有的功能模块，我们都可以用实体（Entity）来表示VHDL串口通信。无论是UART还是Entherent等方式，我们都用对应的实体来实现。对于特定的通信协议，我们将其实现成两个实体：接受实体和发送实体。例如表中的Entity#UART，表示对于UART通信模式，将其具体的通信过程用一个实体来实现。
3. 在C语言中，关于系统与外部控制器通信的代码生成，我们使用的是抽象接口来表示。对于不同协议的不同通信方式，我们把消息分成接受和发送两部分，并且预先实现其功能放在定义的库中。在进行代码生成时，我们调用预先定义库中具体通信函数来进行代码生成。

**Rule5：物理设备驱动调度转换**

工控系统环境中的设备物理资源是多样性的，这包括数量多样性、功能多样性。针对关系Dmap = Dimcl -> （Dplc|Dfpga|Dsoc），IMCL模型中所有的设备调度控制采用形式化表示方式，但当研究如何转换到具体目标设备时，需要根据实际的控制器和硬件之间的调度协议来实现调度函数。