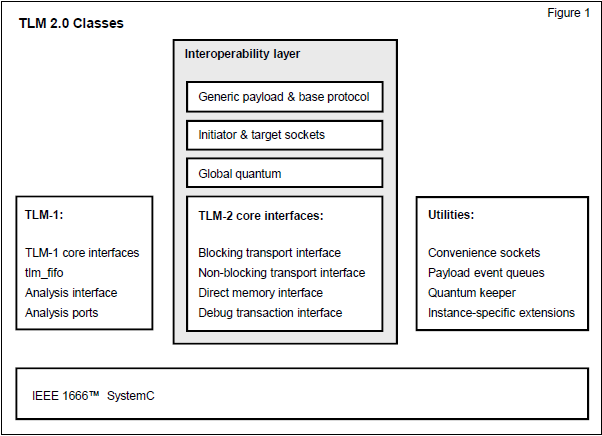
1. Transaction Level Model
   1. 概述

TLM-2.0的组成：core interfaces、global quantum、initiator、target sockets、generic payload和base protocol、utilities。另外，TLM-1.0的core interface、analysis interface和analysis port也被包含在内，尽管这些不属于TLM2.0标准。

TLM-2.0的core interfaces组成：blocking和non-blocking transport interfaces、the direct memory interface(DMI)和debug transport interface。

TLM-2.0在systemc库的上层，为了最大的互操作性、memory-mapped bus modeling。推荐TLM-2.0 core interfaces，sockets，generic playload 和 base protocol 在一起按照一致顺序使用。这些类在一起被称为互操作层。如果有使用generic payload有不合适的地方，那么可能是core interfaces和initiator和target sockes，或者单独core interfaces，被用来作为alternative transaction type。Generic payload直接使用core interfaces，不经过initiator和target sockets，从技术上是可以的，只是这种方法不推荐使用。



Generic payload 主要用来做memory-mapped bus建模，但是也有可能被用来做相似属性的非总线协议的建模。

Generic payload的attributes和phases能够扩展，用于建模制定的协议。但是这种扩展会导致互操作性降低。

最后，loosely-timed模型希望使用blocking transport interface，direct memory interface和temporal decoupling。Approximately-timed模型希望使用non-blocking transport interface和payload event queues。这些说明仅仅是编码风格的建议，并非是TLM-2.0标准协议的一部分。

* 1. 介绍
     1. 背景

TLM-1标准定义了core interface的一个集合，用于通过value和const reference做transproting transcation。这些interface集合在一些应用上成功使用，但是在对memory-mapped buses和其他on-chip 通信网络的建模方面存在3个缺点。

1、TLM-1没有标准的transaction class；所以，每个application都必须创建非标准的classes，这就导致在不同代码间的models互操作性就非常差。TLM-2.0使用generic payload来处理该缺点。

2、TLM-1不支持timing annotation。所以，在models之间没有标准的timing通信信息。TLM-1对delay典型的实现是通过调用wait，这种方法会降低simulation的速度。TLM-2.0增加blocking和non-blocking transport interface处理该缺点。

3、TLM-1接口需要所有的transaction objects和data都通过value或者const reference来传输，这种方法降低simulation速度。一些applications在transaction objects中嵌入pointers，但是这是非标准和无互操作性的。TLM-2.0对此做了处理。

* + 1. Transaction-Level Modeling，用例和抽象

在esl社区，长期存在一个争论，什么最适合分类抽象transaction level modeling。建模曾经根据range of criteria， including granularity of time，frequency of model evalution，functional abstraction，communication abstract，和use cases进行类。TLM-2.0活跃明确地表明，存在多种多样的use cases用于transaction level modeling，而不是围绕每个use case定义一个abstraction level。TLM-2.0标准定义了一套APIs，这些APIs被视为low-level编程机制，用于实现transaction-level models，适合于描述大量coding styles。

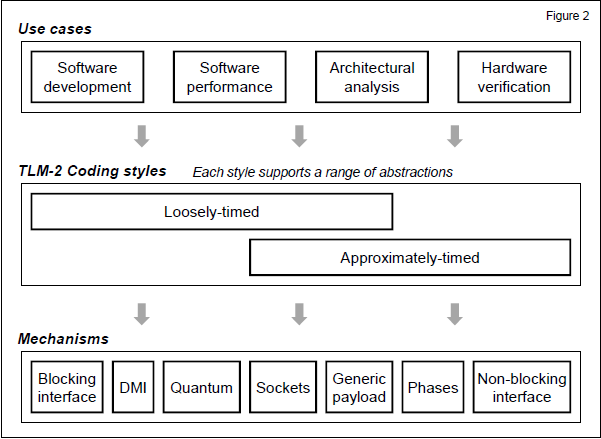
TLM-2.0定义的interfaces标准与descriptions of coding styles是分离的。TLM-2.0 interfaces形成标准的部分规范，并确保互操作性。每种coding style能够支持一系列贯穿functionality、timeing和communication的抽象。原则上，用户可创建自己的coding style。

An untimed functional model consisting of a single software thread can be written as a C function or as a single SystemC process, and is sometimes termed an algorithmic model。这种模型本质上不属于transaction-level，因为按照定义是一个通信的抽象，但进程model没有进程间通信。一个transaction-level 模型需要多个systemc 进程仿真并行执行和通信。

一个抽象的transaction-level model包含多个进程，需要一些机制，能够使这些进程能够被其他进程控制。这就是为什么systemc使用多进程协作的模型。Systemc在thread类型的进程下调用wait()用于控制，在method类型的进程下则直接返回到kernel。调用wait()通常会隐藏在programming API之后，使用这种方法的model通常用于建模不依赖于timing信息的特定抽象或具体的协议。

Synchronization就能够预先精确的确认通信事件的顺序而言，显得很重要，而对于通信事件的顺序由单个进程的具体timing确定而言，显得稍微不那么重要。在systemc中通过使用FIFOs或semaphores可以轻松实现健壮的synchronization，也允许完全untimed modeling style，原则上simulation能够不需要time运行。On the other hand, a fast virtual platform model allowing multiple embedded software threads to run in parallel may use ether strong or weak synchronization. In this standard, the appropriate coding style for such a model is termed **loosely-timed**

A more detailed transaction-level model may need to associate multiple protocol-specific timing points with each transaction, such as timing points to mark the start and the end of each phase of the protocol. By choosing an appropriate number of timing points, it is possible to model communication to a high degree of timing accuracy without the need to execute the component models on every single clock cycle. In this standard, such a coding style is termed **approximately-timed.**



* + 1. Coding styles

Coding style是编程语言风格的一个集合，不特指具体的抽象层，也不是软件编程接口。为了清晰和简单起见，该文档详细说明两种具体coding styles：

1. loosely-timed
2. approximately-timed

按其性质，coding styles不被精确定义，TLM-2.0核心接口规则独立于这些coding styles。原则上，基于TLM-1和TLM-2.0机制定义其他coding style是被允许的。

**Loosely-timed coding style and temporal decoupling**