软件体系结构 SEI/BUAA
北航计算机学院 School of Computer Science & Engineering

第二部分　软件体系结构建模

软件体系结构描述语言（ADL）
----------形式化描述

张莉　教授
北京航空航天大学
软件工程研究所
lily@buaa.edu.cn

非形式化描述存在的问题

- 语义模糊
  - 自然语言的二义性
  - 图形语言直观，同样具有二义性
- 由语义模糊引起的沟通障碍
- 无法实现系统验证
  - 检查系统的一致性与完备性等
- 不适于描述体系结构行为
  - 语义模糊妨碍了系统行为描述的准确性
  - 行为无法通过计算来预测
  - 不能描述运行时的动态行为，不能发现协议间的不一致、死锁等问题。

为什么需要形式化方法

- 在计算机科学，形式化方法指的是用于软件与硬件系统的说明、开发与验证的数学化方法。
- 形式化方法的特点：
  - 可以用于系统描述，且可以在不同层次上进行描述：精确的语义，消除语义模糊。
  - 在体系结构行为描述的优势：
    - 更利于机器表达和计算
    - 提供了形式化且准确的定义用于描述行为、行为模式、行为分析。如：进程代数提供了代数法则用于处理分析等
    - 对行为的分析和建模是系统验证的主要组成部分

件体系结 SEI/BUAA

  - 形式化方法使得系统验证变得可行
    - 自动化理论证明：在给定系统描述的情况下，基于逻辑原语与推理规则来进行形式化证明
    - 模式识别：系统通过对可能到达的状态进行完全搜索的方式验证特定的属性。
- 常见的形式化方法
  - Petri网：主要用于描述分布式系统。
  - Z语言：用于计算系统的建模和描述
  - B-Method(Abrial，1996）
  - CSP（Hoare, 2004）
  - 。 。 。 。 。 。

北京航空航天大学软件工程研究所 lily@buaa.edu.cn

- 软件体系结构描述语言ADL（Architecture Description Language）
  - ADL是软件体系结构研究领域为进行软件体系结构描述和分析而专门定义的一种语言符号.
  - “ADL指用来表示和分析软件体系结构的形式化符号”。一般认为，ADL的核心元素主要包括：
    - 构件(Component):表示系统中主要的计算元素和数据存储，如客户端、服务器、数据库等；
    - 连接件(Connector):定义构件之间的交互关系，如过程调用、消息传递、事件广播等；
    - 软件体系结构配置(Architecture Configuration):描述构件、连接件之间的拓扑关系。构件、连接件的外部可见特性由接口定义。

分类

- 根据各个ADL的研究范围，可归纳为：
  - 描述软件体系结构配置的ADL
  - 描述软件体系结构实例的ADL
  - 软件体系结构风格的ADL
- 典型的ADL有：Aesop，C2，Darwin，Rapide，SADL，UniCon，Meta-H，和Wright等。

部分ADL

ADL	研究单位和代表人物	适用范围	主要设计元素/特点	具备的主要能力
Aesop	美国卡内基-梅隆大学软件工程研究所，David Garlan等	同一设计环境中多种软件体系结构风格的应用；特定风格软件体系结构设计环境的快速生成	定义了六种通用对象类型：构件、连接件、端口、角色、表示和绑定；以子类型方式对通用类型进行扩展可定义新类型；每个对象类型用一个C++类表示，软件体系结构风格信息内嵌在C++类的代码实现中	外部工具集成；语法制导的类型检查；代码编译；可执行系统的生成；循环、资源冲突、调度可行性检查
C2	美国加利福尼亚大学软件研究所，Richard N. Taylor等	主要适用于分布式异构环境中、基于消息的图形用户界面应用系统的软件体系结构描述	主要设计元素包括构件、连接件、以及它们之间的拓扑关系；构件间只能通过连接件相连，具有“受限的可见性”；异步通知消息和请求消息是构件间唯一的通信方式	软件体系结构演化；软件体系结构动态配置；多形式的类型检查；设计决策过程支持；可执行代码生成
MetaH | AADL	美国霍尼韦尔(Honeywell)公司	一种特定领域的ADL，支持对可靠性和安全性要求较高的多处理器实时嵌入式系统的创建、分析和验证，主要适用于航空电子控制软件系统	语义基于形式化调度和数据流模型；提供预定义的构件和连接件类型，与领域密切相关；构件类型包括事件、端口、子程序、包、监视器和进程等；连接类型包括事件连接、端口连接、等价连接和存取连接等	软硬件绑定；实时调度；可靠性和安全性分析；代码自动生成、编译和链接

部分ADL

ADL	研究单位和代表人物	适用范围	主要设计元素/特点	具备的主要能力
Unicon	美国卡内基-梅隆大学软件工程研究所，Mary Shaw等	一种通用类型的ADL，支持多种常用构件和连接件类型的综合应用	主要设计元素为构件和连接件，构件和连接件类型以枚举方式预定义，并定义了构件类型和连接件类型之间的匹配规则；连接件机制内嵌到工具的实现当中	类型检查；编译、链接和可执行系统的自动生成；外部工具集成；进程调度分析
Rapide	美国斯坦福大学，David C. Luckham等	基于事件的、复杂、并发、分布式系统的软件体系结构描述	主要设计元素由接口（相当于构件）和连接规则组成，接口定义包括动作、服务、行为和约束；构件的计算和交互语义通过偏序事件集（posets）定义；构件的行为约束通过抽象状态和状态转移规则定义	软件体系结构模拟执行；模拟结果分析（约束检查器、posets浏览器、模拟动画）；软件体系结构的动态配置和代码生成
Wright	美国卡内基-梅隆大学(CMU)软件工程研究所，Robert J. Allen等	一种完全形式化的ADL，对连接件语义进行了形式化表示，支持用户将复杂的交互模式定义成新的连接件类型	主要设计元素包括构件、连接件和配置，构件定义包括端口和计算，连接件定义包括角色和胶水（Glue）；支持用户定义接口类型和风格；以CSP作为语义基础	用户自定义软件体系结构风格；风格约束检查；模型一致性、完整性检查；死锁分析

北京航空航天大学软件工程研究所 lily@buaa.edu.cn

ADL	研究单位和代表人物	适用范围	主要设计元素/特点	具备的主要能力
Darwin	英国科学、技术和医药皇家大学，Jeff Magee 和 Jeff Kramer等	主要用来对基于消息传递的分布式系统进行描述	主要设计元素由构件组成，构件定义包括提供的服务和要求的服务；没有对连接件提供显式的描述，构件之间的交互通过绑定关系表示；以π-calculus为语义基础	系统动态配置；代码自动生成和编译
ACME	美国卡内基-梅隆大学软件工程研究所，Garlan等	一种软件体系结构交换语言，为ADL及其工具之间信息的共享和交换提供了一个公共的集成架构	主要包括七个核心设计元素：构件、连接件、系统、端口、角色、表示和表示映射	软件体系结构信息交换和共享
XYZ/ADL	中国科学院软件研究所，唐稚松等	一种通用类型的ADL，具有较强的形式化理论基础；能够支持多种设计方法和多种软件体系结构风格的综合运用	主要设计元素包括构件（包括接口和计算）和连接件（接口和交互协议），语义基于时序逻辑语言XYZ/E，可对设计元素的静态和动态语义进行严格的形式化描述	支持软件体系结构的逐层细化、分析、验证和自动代码的生成；支持用户自定义软件体系结构风格

- 众多的ADL及支持工具，方法和形式各不相同，强调不同侧面，作用重要。但也有负面影响：
  - 各ADL都以独立的形式存在，不兼容，选择。
  - 大多与领域相关的，不利于对不同领域进行分析。
  - 某些方面大同小异，有很多冗余的部分。
  - 使设计人员很难选择一种合适的ADL.

北京航空航天大学软件工程研究所 lily@buaa.edu.cn

ADL与其它语言的比较

- 构造能力：ADL能够使用较小的独立体系结构元素来建造大型软件系统；
- 抽象能力：ADL使得软件体系结构中的构件和连接件描述可以只关注它们的抽象特性，而不管其具体的实现细节；
- 重用能力：ADL使得组成软件系统的构件、连接件甚至是软件体系结构都成为软件系统开发和设计的可重用部件；
- 组合能力：ADL使得其描述的每一系统元素都有其自己的局部结构，这种描述局部结构的特点使得ADL支持软件系统的动态变化组合；
- 异构能力：ADL允许多个不同的体系结构描述关联存在；
- 分析和推理能力：ADL允许对其描述的体系结构进行多种不同的性能和功能上的多种推理分析。

- 介绍几种典型的ADL
  - Wright
  - ACME
  - Aesop
  - C2
  - UniCon
  - Darwin
  - Rapide
  - MetaH
  - XYZ/ADL
  - AADL

Wright

- Wright是美国卡内基-梅隆大学软件工程研究所的Robert J. Allen和David Garlan等人定义的完全形式化的软件体系结构描述语言。它为体系结构中连接提供了形式化基础。
- 可以描述体系结构风格、系统族、体系结构实例和单个系统。
- 提供了对计算构件和连接件的描述
- 关键思想：
  - 将体系结构连接件定义为明确的语义实体。这些实体用协议的集合来说明，而这些协议代表了交互中的各个参与角色及其相互作用。

Wright

- 特点：使用显式的、独立的连接件类型作为交互模式。
  - 构件定义由3部分组成：构件标识符、端口集合、表示构件计算行为的CSP进程。
  - 构件通过端口与其它构件进行交互。连接件也有类似于构件的端口。粘接规约描述参与交互各方的行为如何协调、组合。
- 符号定义如下：
  - e?x表示x是事件e的输入值。在通道e上输入x。
  - e！y表示y是事件e的输出值。在通道e上输出y。
  - e→P前缀操作符，表示一进程执行事件e，然后成为进程P。
  - P∏Q表示一进程不确定的选择进程P或Q执行。
  - P□Q表示一进程根据与其发生交互的其它进程从进程P和进程Q中选择一个执行。
  - √终止操作符，表示进程可以终止。

例子

- 下图是一个简单的客户机/服务器模型。
  - 该模型由一个连接件和一个系统配置组成。客户的request端口和服务器的provide端口分别连接到连接器的Client和Server角色上。

例子

- 一个简单的客户服务器结构的体系结构

```
System SimpleExample
   Component Server =
            port provide [provide protocol]
              spec [Server specification]
  Component Client =
            port request [request protocol]
              spec [client specification]
  Connector C-S-connector =
            role client [cient protocol]
            role server [server protocol]
              glue [glue protocol]
  Instances
        s: Server
        c: Client
        cs: C-S-connector
  Attachments:
       s.provide as cs.server
       c.request as cs.client
end SimpleExample
```

第一部分：定义构件和连接件类型

第二部分：定义构件和连接件实例的集合

第三部分：定义构件和连接件实例被组合在一起的方式

体系结构描述和推理

- 体系结构描述包括：协议（用CSP进程表示）和协议轨迹预测说明
- 支持端口和角色的一致性检查。
- CSP(Communicational Sequential Processes)是C A R.
  - Hoare在1978年提出的分布式程序设计原型“通讯顺序进程”。它是进程代数的重要组成部分，并被广泛用于并发分布式设计的建模和分析中。
  - C A R. Hoare, Communicating Sequential Processes, Prentice Hall. 1985 （英国图灵奖获得者）
- CSP提供了丰富的描述进程行为的语义集，主要包括进程的语义、实施、行为、并发、通信等内容。

- CSP(Communicational Sequential Processes)
- 语言的目的：为了克服传统程序设计语言在多处理器体系上的障碍。
  - 基本单元是一系列并发的进程，同时能很方便地描述进程之间的通信和同步。
  - 一段CSP描述由一组按顺序执行的命令组成。
  - CSP可以通过并行命令生成新的进程。一个新进程不能使用进程之间共享的变量。并行命令中的进程是同步执行的。并行命令的结束是以所有进程都结束为标志的。

- **Wright只选用了CSP的一个小型子集，包括：**
  - 进程与事件（Processes and Events）
  - 前缀(profixing)
  - 选择
  - 决策

特点

- Wright没有提供图形化的描述方式。
- 主要特点：
  - 将连接件语义进行了显示的形式化表示，支持用户将复杂的交互模式定义成新的连接件类型。
  - 支持用户自定义软件体系结构风格，并可对风格的约束进行检查。与Aesop将软件体系结构风格的约束信息内嵌到实现对象的方法中的做法不同，Wright对软件体系结构及其约束信息的定义都是显示表示的。
- Wright的语义建立在CSP基础之上，通过事件和进程来定义构件、连接件、以及软件体系结构配置的行为，支持模型的一致性、完整性检查，以及构件之间连接关系的静态死锁分析。

ACME

- ACME是一种软件体系结构交换语言
  - 目的：解决目前出现的众多软件体系结构描述语言之间的不兼容问题。
  - 可以作为体系结构设计工具中的通用交换格式，也可以作为开发新的体系结构设计和分析工具的基础。
- ACME项目开始于1995年，由Carnegie Mellon大学的David Garlan等人开发。
  - 当时的目标是在不同的体系结构开发工具之间提供一种能够相互交换体系结构描述的工具。
  - 目前，ACME语言和工具开发库（Acmelib)为软件体系结构的描述、表示、生成和分析，提供了一种通用的、可扩展的基础设施。
- 尽管从严格意义上讲，ACME并不是一种体系结构描述语言，但是它包含了大量ADL特点，实用性很高。我们也作简要介绍。

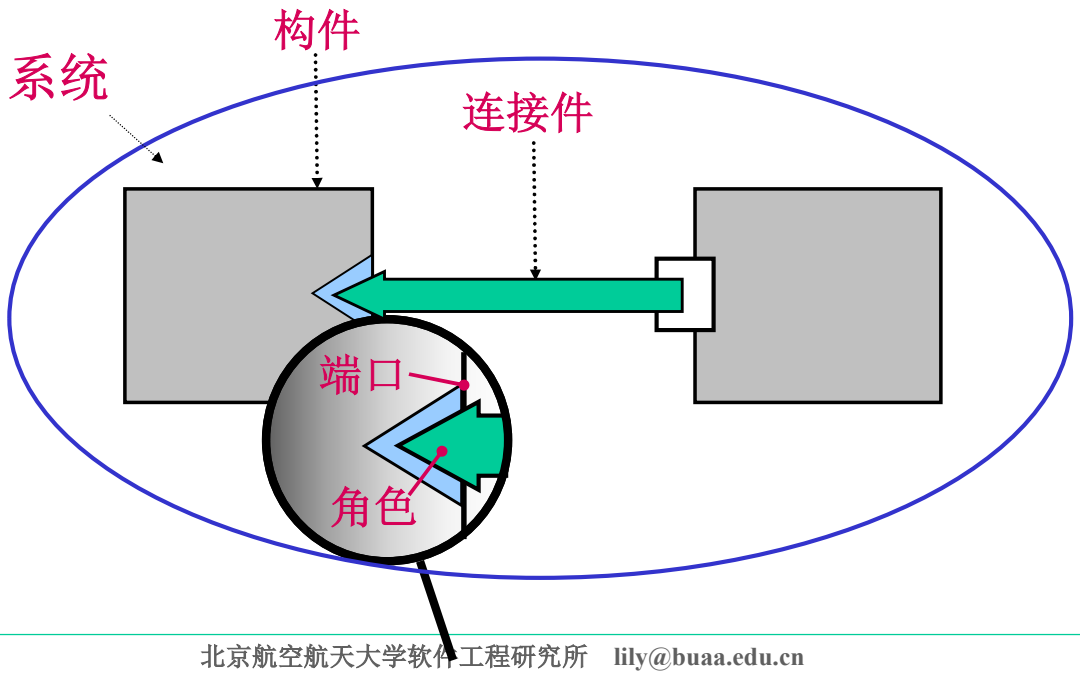
- ACME语言和工具包提供3种功能：
  - 体系结构的相互交换。
    - 提供一种体系结构设计的交换格式。
    - 允许各种开发工具与其他补充工具协同工作。
  - 为新的体系结构设计和分析工具提供了可扩展的基础。
    - 体系结构设计和分析工具需要一个用于描述、存储、操纵体系结构设计的表示法
    - ACME及其开发工具提供了一个很好的基础。
  - 体系结构描述。
    - ACME本身是很好的开发语言，虽然不是适合所有的应用系统。
    - ACME可以让开发者很好地认识体系结构建模，提供了一个相对容易的对简单软件系统的描述方法。

特点

- ACME有如下主要特点：
  - 用7种基本的设计元素来表示软件体系结构
  - 提供了一种灵活的注释机制，支持将体系结构和用子语言表示的非结构化信息结合起来。这些子语言是外部定义的。
  - 提供了一种类型机制，用于抽象出共同的、可重用的体系结构惯用法和风格。
  - 提供了一个开放的语义框架，用于体系结构描述的推理。

ACME体系结构设计元素类型

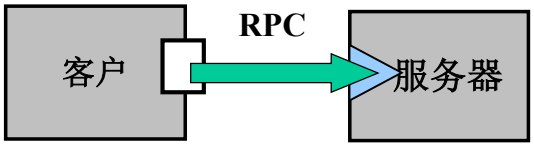
- 7类基本元素：
  - 构件、连接件、系统、端口、角色、表述和表述图。
- 核心是：构件、连接件和系统



- 构件：代表系统中基本的计算元素和数据存储。
  - 典型的构件有：客户、服务器、过滤器、对象、黑板、数据库等。
- 连接件：构件之间的通信和交互方式。
  - 例子：简单的交互形式，如管道、过程调用、事件广播等；复杂的交互，如客户/服务器协议、数据库和应用程序之间的SQL连接。
- 系统：代表构件和连接件的配置。

- 构件的接口：定义为端口(port)的集合。一个端口表示构件与外部环境的一个交互点。
- 连接件的接口：可以用角色来定义。每个角色定义了连接件所代表的交互中的一个参与方。
  - 二重连接有两个角色。如RPC中的Caller和Callee;管道连接件中的Reading和Writing；消息发送连接件中的sender和receiver等。

例子

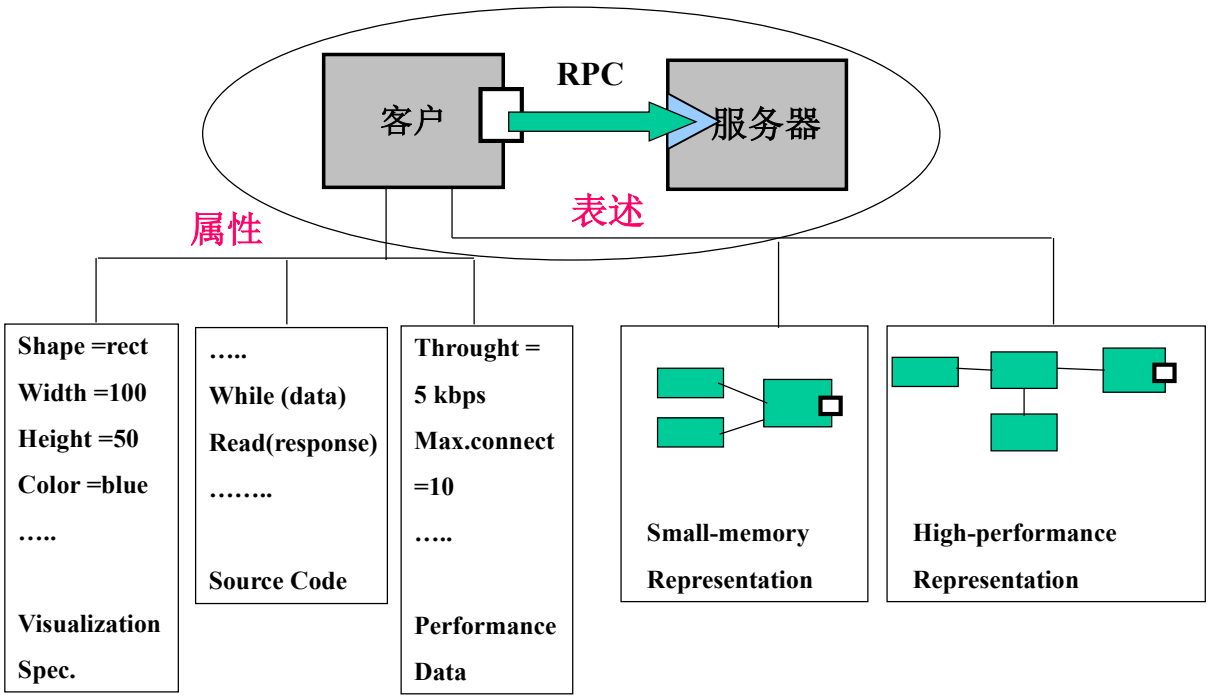


- 以C/S模型为例。图中包含两个构件和连接件。ACME描述如下：

```
System simple_CS={
        Component Client = { Port send-request }
        Component Server = { Port receive-request }
        Connector RPC = { Roles (Caller, Callee) }
        Attachment: { Client. send-request to RPC.Caller
              Server. receive-request to RPC.Callee
               }
        }
```

- ACME支持体系结构的分层描述。每个构件或连接件都能用一个或多个更详细、更底层的描述来表示－－－称为一个表述（representation).
- 表述图：当某个构件或连接件有体系结构表述时，表述图用于说明在这个表述的系统内部和此表述所代表的构件或连接件的外部接口之间，存在何种关系。

构件的表述和属性



小结

- ACME为公共的软件体系结构信息提供了一个统一的结构化框架，主要元素包括构件、连接件和系统。
  - 构件:系统的计算和数据存储单元，构件接口由一组端口定义，每个端口标识了构件和环境之间的一个交互点，一个构件可以具有若干不同类型的端口。
  - 连接件:表达了构件之间的交互，协调构件之间的通讯和合作
  - 系统是构件和连接件的具体配置，一个系统包括一组构件、一组连接件和一组描述系统拓扑结构的连接。
- ACME允许不同的软件体系结构描述语言和相应的支持工具共享相同的软件体系结构信息，同时提供开放的语义框架，支持特定软件体系结构描述语言表示的信息规约。这样，ACME为不同的软件体系结构描述语言和相应的支持工具提供了公共的软件体系结构信息交换格式。

Aesop

- Aesop提供了描述软件体系结构的机制，但不是严格意义上的ADL，是卡内基-梅隆大学软件工程研究所的Garlan等人设计开发的一个软件体系结构设计环境生成器(Software Architecture Design Environment Generator)。
- Aesop建立在一个通用的对象模型基础之上，该模型中定义了六种用于对软件体系结构进行描述的对象类型：Component、Connector、Port、Role、Representation和Binding。
- 通过对这些通用类型进行扩展（某种风格所具有的设计元素类型作为这些通用类型的子类型创建），用户可以快速创建适用于某种特定软件体系结构风格的设计环境。在工具实现中，每个对象类型用一个C++类来表示，软件体系结构风格的约束信息则在相关对象类型的方法操作中编码实现。

- Aesop的主要特点是支持特定风格的软件体系结构设计环境的快速生成。
- 最初的目标是提供一个用于构造开放的、支持体系结构风格的体系结构环境的开发工具包。
- 基本思路：使定义新的风格变得容易，从而让设计者能够利用这些风格进行体系结构设计。
- 基于以上想法，Aesop提供下列支持：
  - 一个开发工具包，用于快速开发体系结构设计环境，着重于特定领域的体系结构风格；
  - 一个开放的工具集成框架，用于支持与其他工具的协作；
  - 一个“有风格意识的”库，用于存储、检索和重用体系结构设计元素；
  - 一个可以维持体系结构设计状态的面向对象数据库，用于存储和操作体系结构设计；
  - 一个用户可定制的基于Tcl/Tk的图形用户界面

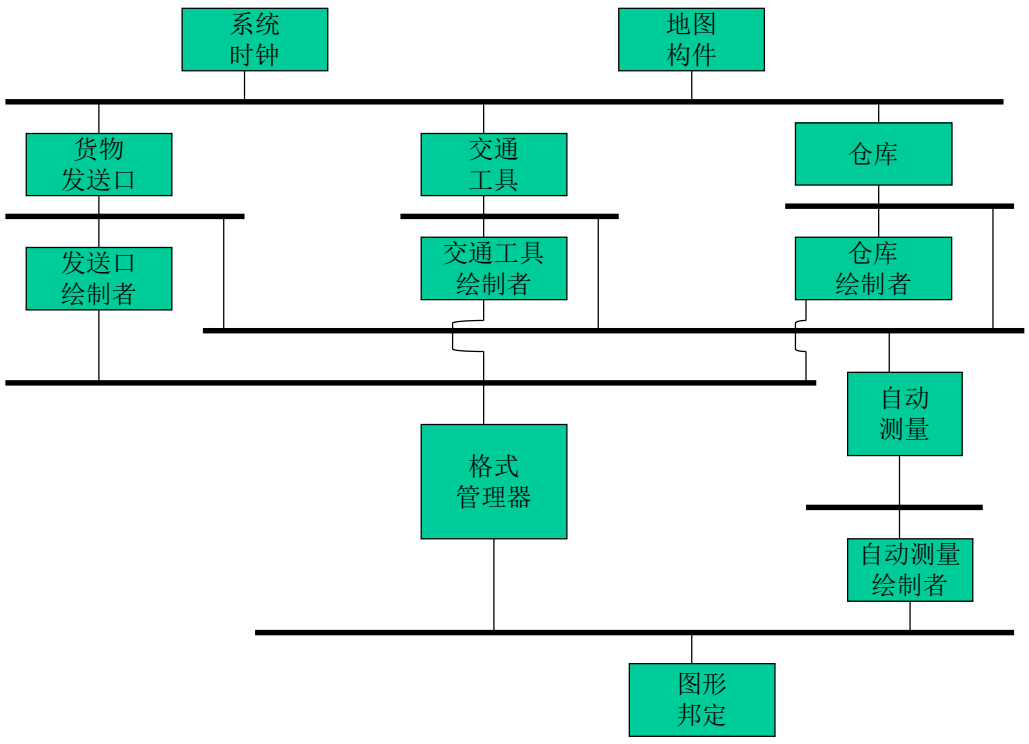
C2

- C2是加利福尼亚大学软件研究所ISR（Institute of Software Research at University of California）的Richard N. Taylor等人定义的，主要是对C2风格的软件体系结构进行描述。
- C2风格是由ISR定义的一种软件体系结构风格，
  - 主要适合于分布式异构环境中、基于消息的图形用户界面应用系统的设计。
  - 其主要的设计元素包括构件、连接件，以及它们之间的拓扑关系。
- 一个C2风格的软件体系结构，可以看作由并行构件挂接在消息路由设备上连接而成的一个具有层次性的网络。
  - 每个构件和连接件都有一个“顶层域”和一个“底层域”。
  - 构件的“顶层域”连接到某个连接件的“底层域”，构件的“底层域”连接到另一个连接件的“顶层域”，连接数目不受限制。
  - 构件之间的交互只能通过连接件进行，异步通知消息和异步请求消息是构件之间进行交互的唯一方式。
  - 连接件负责对消息进行路由或多点传送。

C2

- C2风格的一个主要特点是其“受限的可见性”，即对于层次结构中的任何构件来说，它仅知道其上层构件的存在，而不知道其下层构件的存在。
  - 通过在构件内部定义一个消息转换器来实现的。消息转换器可以将其发送的请求消息转换为接收方可理解的形式，并把接收的通知消息转换为自身可理解的形式。
  - 这种机制提高了构件的可替换性和重用性，以及系统的可扩展性和一定的动态配置能力。
- ArchStudio是一个建立在C2基础之上的基于软件体系结构的开发环境。
  - 包含一个Java和一个C++的面向对象类框架，这两个框架为其构件、连接件和消息交互机制的实现提供了底层支持，可以从软件体系结构模型生成可执行代码。

C2风格的货物路由系统的体系结构



北京航空航天大学软件工程研究所 lily@buaa.edu.cn

C2

- 对构件的描述

```
Component::=
  component component_name is
        interface component_message_interface
        parameters component_paremeters
        methods component_methods
        [behavior  component_behavior]
        [context   component_context]
end component_name;
```


- 对接口的描述

```
component_message_interface ::=
  top_domain_interface
  bottom_domain_interface

top_domain_interface ::=
    top_domain  is
          out  interface_requests
           in   interface_notifications

bottom_domain_interface ::=
    bottom_domain  is
          out  interface_notifications
         in   interface_requests

interface_requests ::=
    { request; } | null;

interface_notifications ::=
    { notification; } | null;

request ::=
    message_name (request_parameters)

request_parameters ::=
  [to component_name][parameter_list]

Notification ::=
   message_name[parameter_list]
```

Unicon

- Unicon(UNIversal CONnection)是SEI/CMU 的Mary Shaw等人定义的一个通用型软件体系结构描述语言， 主要目的：
  - 支持对现有组件的使用。
  - 提供对组件和连接器的统一的访问。
  - 区分不同类型的组件和连接器以便对体系结构配置进行检查。
  - 支持不同的表示方式和不同的分析工具。
- 主要元素包括构件和连接件。
  - 构件代表软件系统中的计算和数据的处所，用于将计算和数据组织成多个部分。这些组成部分都有完善定义的语义和行为。
  - 连接件代表构件间的交互作用的类。它们在构件之间的交互中起中介作用。

- Unicon中构件和连接件的结构非常对称，都包括如下几个部分：
  - 名字Name；
  - 规格说明Specification（构件的规格说明称为构件的接口Interface，连接件的规格说明称为连接件的协议Protocol）；
  - 构件和连接件类型Type；
  - 一个特性列表Property List（每个特性用“特性名－数值”的形式表示，是对设计元素的属性、约束信息等的进一步说明，Unicon有35个预定义的特性）；
  - 一系列的关联单位Association Unit（构件的关联单位称为演员Player，连接件的关联单位称为角色Role）；
  - 实现Implementation。构件和连接件的实现部分都有原子Primitive和复合Composite两种形式。原子构件直接由程序设计语言或系统外壳脚本编码实现。复合构件则通过构件和连接件的组合表示。原子连接件通常直接通过编程语言或系统中的内嵌机制（如过程调用、数据共享、消息机制等）来实现。目前Unicon还不能对复合连接件提供支持。

北京航空航天大学软件工程研究所 lily@buaa.edu.cn

- Unicon以枚举的方式预定义了多种构件和连接件类型。
  - 预定义的构件类型包括Module，Computation，SharedData，SeqFile，Filter，Process，SharedProcess，General。
  - 预定义的连接件类型包括Pipe，FileIO，ProcedureCall，DataAccess，PLBundler，RemoteProcCall，RTScheduler。Unicon还定义了每个构件类型的演员和每个连接件类型的角色，以及约束演员和角色匹配关系的内部规则。
- Unicon提供了图形和文本两种描述方式，可以在这两种方式之间进行转换，提供了编译器支持文本形式编译。
- Unicon将各种连接件的实现机制都内嵌到工具的实现当中，编译器对实现构件功能的源文件或对象文件的位置进行跟踪。
  - 当模型经过解析并将所有连接关系都建立起来之后，Unicon可以产生一个类似Make文件的描述文件，然后触发目标语言的编译和链接工具来生成可执行系统。
  - Unicon提供了与外部开发工具进行集成的机制，通过集成RMA。
  - Unicon可以对实时处理过程中进程的调度情况进行分析。

- 构件的定义：

<Component>:== COMPONENT <identifier>
<interface>
<component_implementation>
END < identifier >

<interface>:== INTERFACE IS
TYPE <component_type>
<property_list>
<player_list>
END INTERFACE

<component_type>:==Module|Computation|SharedData|SeqFile|
Filter|Process|SchedProcess|General

<component_implementation>:==<primitive_implementation>|
<composite_implementation>

……

- 连接件的定义：

<Connector>:== CONNECTOR <identifier>
<protocol>
<connector_implementation>
END < identifier >
<protocol>:== PROTOCOL IS
TYPE <connector_type>
<property_list>
<role_list>
END PROTOCOL
<connector_type>:==DataAccess|FileIO|Pipe|PLBundler|
ProcedureCall|RemoteProcCall|RTScheduler
<connector_implementation>:== IMPLEMENTATION IS
BUILTIN
END IMPLEMENTATION

- 连接件的实现方式是UniCon语言内建的，不支持用户定义的连接件。

Darwin

- Darwin主要用来对基于消息传递的分布式系统进行描述，通过π-calculus来描述结构元素的语义,对系统行为建模，利用其强类型系统进行静态检查。
- Darwin没有对连接件提供显式的描述，构件之间的交互关系通过绑定关系来表示。
  - 提供了惰性实例化（Lazy Instantiation）和显示动态构建（Explicit Dynamic Constructions)两种机制来支持系统的动态配置，支持软件体系结构执行时期的演化。
- Darwin对软件体系结构风格的支持通过参数配置实现。Darwin具备编译器，这使得它们可以根据软件体系结构描述生成可执行系统。

Rapide

- Rapide是美国Standford大学的David C. Luckham等人定义的一种可执行的软件体系结构描述语言，主要适用于对基于事件的、复杂、并发、分布式系统的软件体系结构进行描述。
- Rapide是一种基于事件的、并发的面向对象语言，为系统的体系结构建立快速原型。
- Rapide中构件的计算和交互语义通过偏序事件集（称为posets）定义。构件计算由构件接受到事件触发，并进一步产生事件传送到其他构件，由此触发其他计算。
- Rapide构件类型（称为Interfaces）的定义包括动作、服务、行为和约束，其中对动作和服务的执行和调用表现为事件的接收和发送。构件间的连接通过监视和产生事件的匹配来实现，并不能对连接件提供显示的支持。

Rapide特点

- 与Unicon、Darwin等直接通过实现来定义构件行为的方式不同，Rapide通过抽象状态和状态转移规则来定义该类构件的行为约束。
  - 利用这种机制Rapide可以对软件体系结构进行模拟执行，模拟执行过程中可以进行模型的一致性检查和分析，模型的执行结果被表示为一系列具有因果（Causal）和时序（Timing）关系的事件集合。
- Rapide具备编译器，可以根据软件体系结构描述生成可执行系统。

Rapide

- Rapide由5种子语言组成：
  - 类型语言：定义接口类型和函数类型，支持通过继承已有的接口来构造新的接口类型。
  - 模式语言：定义具有因果、独立、时序等关系的事件所构成的事件模式。
  - 可执行语言：包含描述构件行为的控制结构。
  - 体系结构语言：通过定义同步和通信连接来描述构件之间的事件流。
  - 约束语言：定义构件行为和体系结构所满足的形式化约束，其中约束为需要的或禁止的偏序集模式。

Rapide

- 对构件的描述

```
type Application is interface
   in action Request ( p: params);
   out action Request (p : params);
 behavior
  (?M in string) Receive(?M=>Results(?M);
end Application;
```

MetaH

- MetaH是由Honeywell定义的，支持对可靠性和安全性要求较高的多处理器实时嵌入式系统的创建、分析和验证，主要适用于航空电子控制软件系统。
- MetaH的语义主要基于形式化调度和数据流模型。
  - 预定义了一些构件类型，包括事件（Event）、端口(Port)、子程序（SubProgram)、包（Package)、监视器（Monitor）和进程(Process)；
  - 预定义连接类型包括事件连接、端口连接、等价连接和存取连接等。在构件连接件的定义中嵌入了很多与领域相关的信息。
- MetaH支持实时调度、可靠性和安全性分析，以及代码的自动生成、编译和连接功能，可以根据软件体系结构描述生成可执行系统。

AADL（Architecture Analysis and Design Language）

- AADL是一种体系结构描述语言，主要是用于对嵌入式实时系统进行建模。
- AADL的背景：
  - 在设计关键任务和实时性系统时，设计师必须满足功能需求和非功能需求，如性能(吞吐量和服务质量)、保护、可靠性、时间紧迫性、安全和容错等。随着嵌入式软件系统的硬件多样性和复杂性的不断提高，可以采用模型驱动的开发方法来满足开发早期阶段出现的系统集成问题。基于模型的设计方法的要点之一是要选择合适的设计语言来描述具体平台架构。

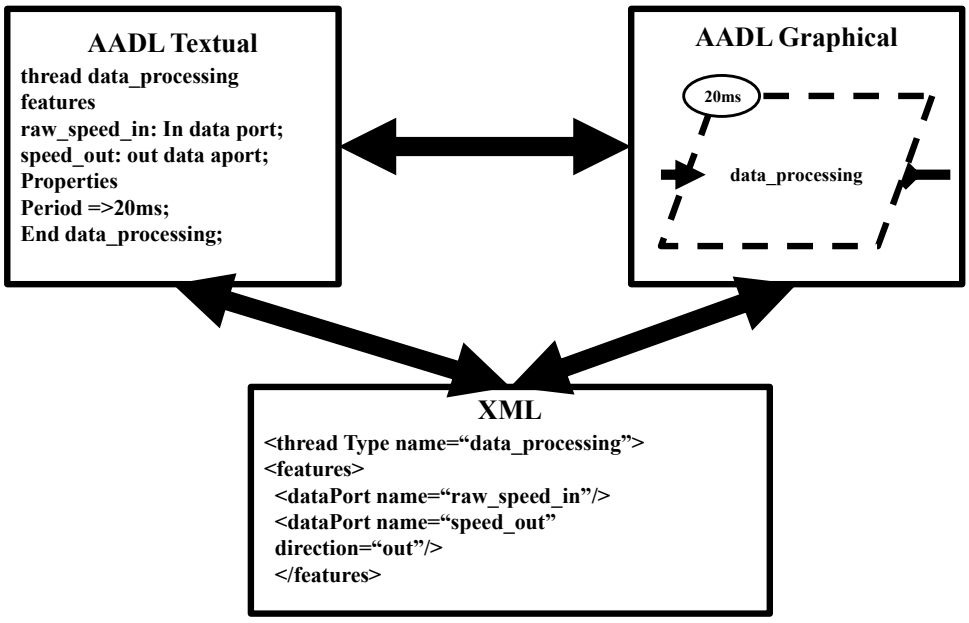
AADL的产生

- 1991年Honeywell实验室基于嵌入式系统提出了MetaH语言
- 2001年美国汽车电子协会SAE提出了航空架构描述语言AADL（Avionics Architecture Description Language），由于以上两个语言不足以解决现实中航空系统存在的一些问题。
- 2004年SAE，卡内基梅隆大学CMU和Honeywell实验室在MetaH和AADL的基础上提出了架构分析和设计语言AADL（Architecture Analysis and Design Language）。
- 在2004年推出基于EclipseOsate的1.0版本（Open Source AADL Tool Environment），
- 2009年推出Osate2.0版本。

AADL介绍

- AADL标准定义了三种模型描述的方式：文本化，XML和图形化。正是由于这种多样化的描述方式，AADL可以被许多不同的工具所使用。对UML的支持同样允许我们在UML的工具中插入AADL。
- 作为一种体系结构描述语言，AADL通过描写构件和连接来建立系统的体系结构。一个AADL的描述由一系列构件的描述组成，通过对这种描述的实例化来建立系统体系结构的模型。

AADL描述方式（文本化、XML 、图形化）

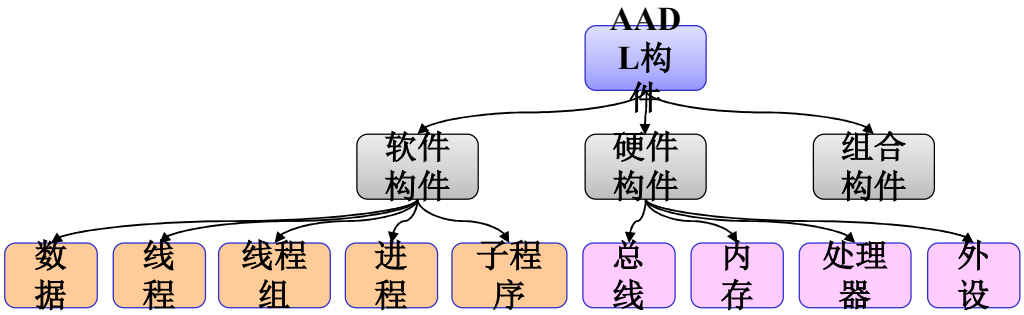


AADL的构件

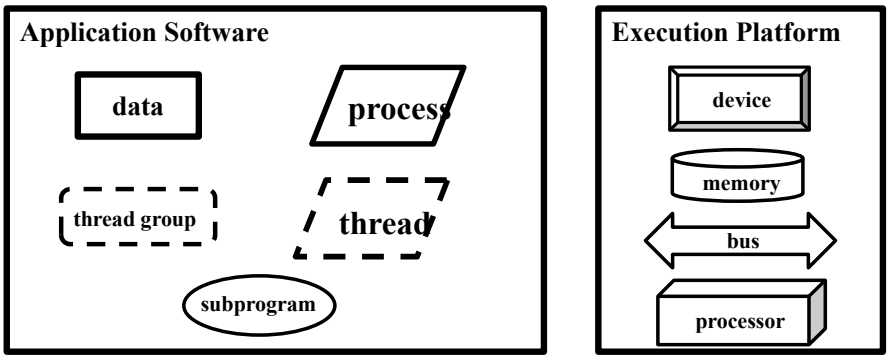
- AADL的构件的定义包括两部分：构件的类型和构件的实现。一个AADL的构件拥有一个类型(component type)以及对应的零个、一个或者很多个实现(component implementation)。
- 构件的类型描述了对外的功能接口(进出端口，属性等)。
- 构件的实现描述了构件的内部结构(子构件，其他的属性等)。
- 一个构件的类型可以有多个实现，和构件类型一样，一个实现同样可以扩展为其他的实现。

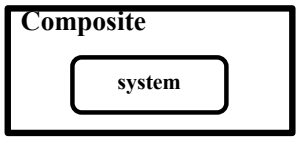
AADL构件的种类

- AADL 定义了很多构件的种类， 总的来说可以分成三大类：软件构件，执行平台构建以及组合构件。
- 软件构件用于软件体系结构建模,包括数据(data)、线程(thread)、线程组(thread group)、进程(process)、子程序(subprogram)构件。
- 执行平台构件用于硬件体系结构建模,包括处理器(processor)、存储器(memory)、总线(bus)、 外设(device)构件。
- 系统构件组合所有的构件，层次化地建立系统的体系结构。



AADL构件的描述



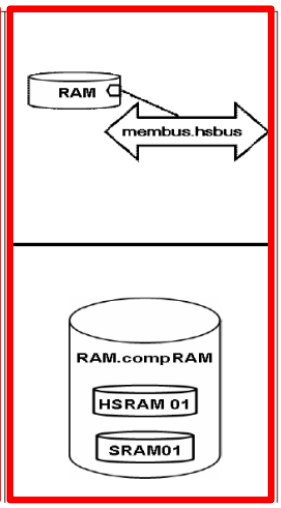


AADL 的特点

- AADL 支持文本和图
- AADL可以用来描述
  统的体系结构， 以及
  输入输出） 和性能关
- AADL可以用来描述
  入输
  定位
- AAD
  AADL

Memory表示存储数据和代码构建，Memory可以在process内，也可以作为单独的执行平台构建，通过bus和processor连接。

```
memory RAM
features
bus01: requires bus access membus.hsbus;
end RAM;
--
memory implementation RAM.compRAM
subcomponents
HSRAM01: memory XRAM.HSRAM;
SRAM01: memory XRAM.SRAM;
end RAM.compRAM;
--
memory XRAM
end XRAM;
--
memory implementation XRAM.HSRAM
end XRAM.HSRAM;
--
memory implementation XRAM.SRAM
end XRAM.SRAM;
--
bus membus
end membus;
--
bus implementation membus.hsbus
end membus.hsbus;
```



型运行模式以及模式之间的切换，
行时体系结构的动态行为 。

AADL研究现状

- AADL语言的研究
- AADL建模方法的研究
- AADL模型转换
- AADL模型转换
- AADL模型的验证与分析
- 基于AADL模型的自动代码生成

AADl应用实践：

- **TOPCASED**：TOPCASED由空中客车公司（空客）提出并应用的项目，采用模型驱动开发和形式化验证相结合、多种开源工具集成的思想，支持复杂嵌入式实时系统的设计、开发与实现，以保证系统的质量属性并降低开发时间和成本。
- **COTRE**：COTRE是空中客车公司的一个项目，它的目标是在软件设计到实现的工业过程中，将形式化方法和半形式化方法相结合，定义一种实用的，工具性的建模和验证方法，并且将这种方法应用到实际生产中。
- **ASSERT**: ASSERT项目的目标是对嵌入式实时系统的开发过程的改进。AADL被用来描述所开发系统的体系结构，由于它的良好扩展性，可以用于支持那些必要的信息，特别是基于验证机制的工业系统的特别数据(验证的描述，非功能性要求，条件的可行性等等)。

XYZ/ADL

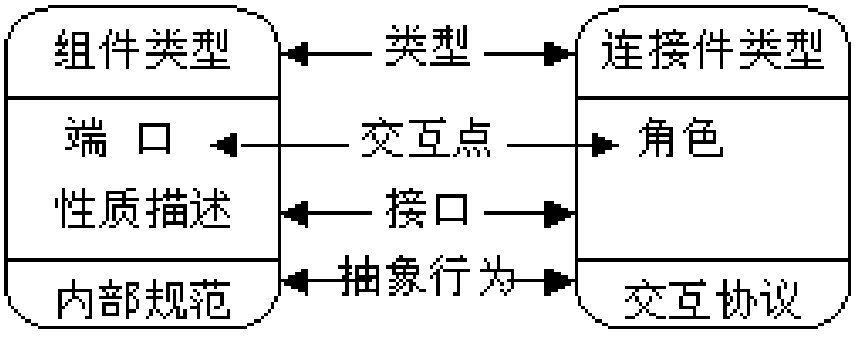
- XYZ/ADL是中国科学院软件工程研究所唐稚松院士等定义的一种基于时序逻辑的软件体系结构描述语言。
- XYZ/ADL以时序逻辑语言XYZ/E为形式化基础，可以在统一的时序逻辑框架下描述系统静、动态语义结构，并可进行相关性质分析。
- 有一套基于XYZ/E的CASE工具集作为底层支撑，可对软件体系结构进行描述、分析、求精、验证、直至最终系统的编译实现。

XYZ/ADL

- 特点：
  - 支持逐层设计软件体系结构，设计过程从外到内部的不断求精过程。
  - 是可视化的体系结构描述语言，支持软件系统的体系结构设计。
  - 在体系结构的每步设计中，XYZ/ADL能生成相应的XYZ/ADL程序。
- XYZ/ADL的基本概念有：组件、连接件、系统、端口、角色、性质、配置、体系结构和风格，以及粘接操作和绑定操作。其中组件、连接件和系统是最主要的实体元素。
  - 组件是一个具有一定功能的逻辑单元或者存储对象。典型组件的例子有：客户机，服务器，过滤器，对象，黑板，数据库等。组件由接口和内部规范组成。
  - 组件的接口包括一组端口和一个性质描述。
  - 连接件用作定义组件之间的交互方式和规则。
  - 连接件由接口和交互协议组成。

XYZ/ADL

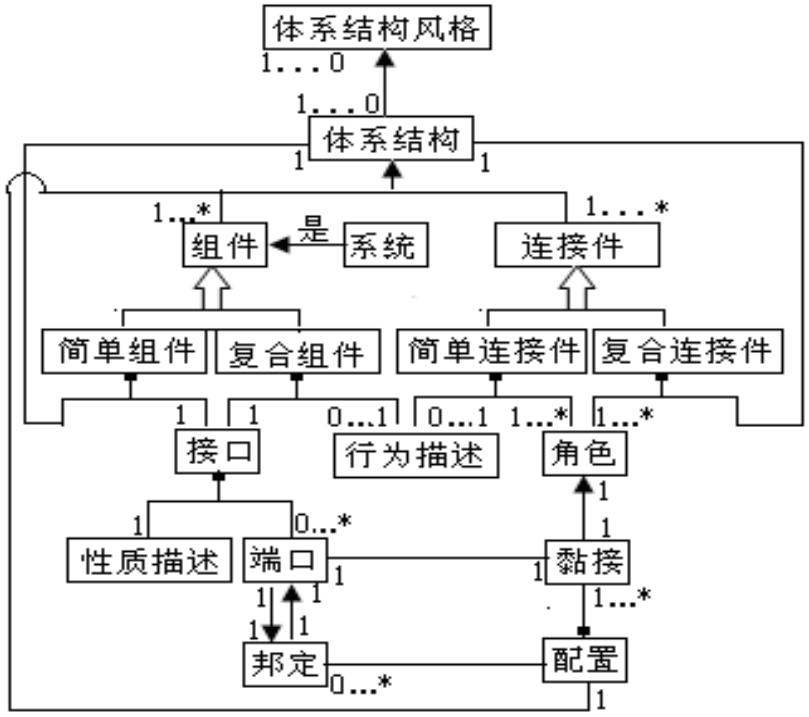
- 组件和连接件是体系结构的基本元素，它们的构成如图所示。



北京航空航天大学软件工程研究所 lily@buaa.edu.cn

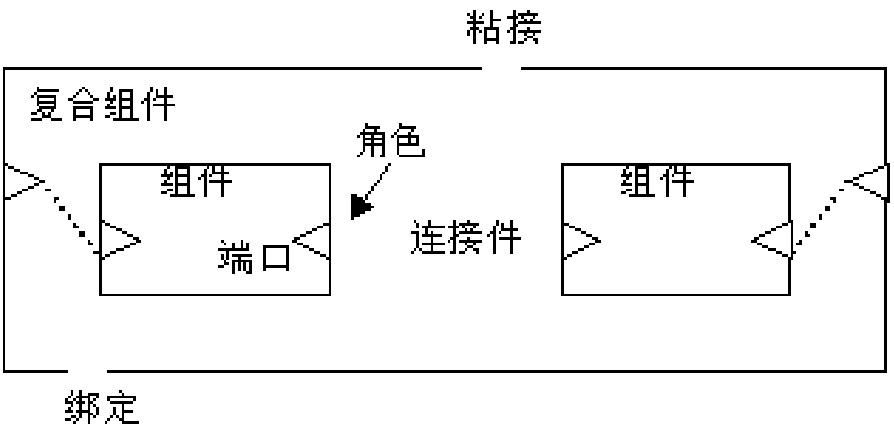
XYZ/ADL

XYZ/ADL中体系结构概念之间关系直观地表示如图。



XYZ/ADL

- 规定图形方式作为我们的体系结构框图表示。



- 例如：假设过滤器组件Filter1用来计算输入数的阶乘。有一个数据输入端口DataIn输入整数，和一个数据输出端口DataOut输出结果，那么Filter1可描述如下：

```
%COMPONENT Filter1=[
%PORT DataIn=INT;
    □[LB1= START⇒$0DataIn?x∧$0LB1=L1;
    LB1= L1∧～（x=EOF）⇒$0DataIn?x∧$0LB1=L1;
    LB1=L1∧(x=EOF)⇒$0LB1=STOP
      ]
```



```
%PORT DataOut==INT;
    □[LB2=START⇒$0LB2=L1;
    LB2= L1 ∧～（y=EOF）⇒$0DataOut?y∧$0LB2=L1;
    LB2= L1 ∧～（y=EOF）⇒$0DataOut!EOF∧$0LB2=STOP
  ]
%PROPERTY==□[(x>0⇒◊(y=x!))∧～DataOut!y）]
%COMPUTATION==
    □[LB=START⇒$0x=0∧$0y=0∧$0LB=L0;
    LB=L0⇒$0DataIn?x∧$0LB=L1;
    LB=L1∧(x=EOF) ⇒$0LB=L2;
    LB=L1∧(x=EOF) ⇒$0LB=End;
    LB=L2∧～（x<0）⇒◊(y=x! ∧LB=L3);
    LB=L2∧（x<0）⇒$0LB=L0;
    LB=L3⇒$0DataOut!y∧$0LB=L0;
    LB=End⇒$0DataOut!EOF∧$0LB=STOP
      ]
```

- 例如，组件Filterl的内部规范可以是上述的较抽象描述，也可以是具体的算法描述：

```
%COMPUTATION=
□[LB =START⇒$OLB=L0;
LB=L0⇒$0DataIn?x∧$OLB=L1;
LB=L1∧～(x=EOF)⇒$OLB=L2;
LB=L1∧(x=EOF)⇒$OLB=End;
LB=L2∧～(x<0)⇒$0i=1∧$0y=1∧$OLB=L3;
LB=L2∧(x<0)⇒$OLB=L0;
LB=L3∧(i<x)⇒$0y=y*i∧$0i=i+1∧$OLB=L3;
LB=L3∧～(i<x)⇒$0DataOut!y∧$OLB=L0;
LB=End⇒DataOut!EOF∧$OLB=STOP
]
```


ADL的发展方向

- 除此之外，还有北京大学信息科学技术学院梅宏等人提出的支持构件组装的描述语言ABC/ADL，中国东北大学软件中心定义的专用于多智能体系统的描述语言A-ADL (Agent-Based -ADL) 等。
- 现状：
  - 当前ADL研究主要侧重于对软件体系结构分析、演化、以及代码自动生成能力的研究。
  - 每种ADL分别侧重软件体系结构设计的某些方面，适于解决不同的问题。
  - 在软件体系结构研究初期，这种“百家争鸣”的情况为人们深入理解软件体系结构不同方面的特性，以及软件体系结构设计在开发过程中的作用有积极的意义。但是，随着软件体系结构技术的不断发展，尤其是经历了对基本概念的收集、整理阶段，逐渐向实际项目开发中推广和应用时，这种情况长时间维持下去，将不利于软件体系结构设计工具的研发和软件体系结构技术的推广。

- 要解决这一问题，归纳起来大致存在这样几种思路：
  - 定义一种“统一的 (Unified)”ADL
    - 为此发现和定义语言的核心 (Kernel)，并使其具备良好的扩展能力是至关重要的。
  - ADL 工具集成
    - Garlan等人正在为这一目标而努力
  - 定义面向特定领域的ADL
  - 将ADL的概念融合到统一建模语言UML当中
  - 维持现状