软件体系结构
(Software Architecture)

三、软件体系结构描述语言

- 增加UML“4+1”View模型

Recommend
火龙果软件

软件体系结构的现状

- 体系结构的理解停留在直觉、逸事和传说层面
- 非形式化描述
  - 框线图
  - 非形式化的陈述
- 惯用的词汇传递了丰富的语义
  - RPC
  - Client-Server
  - Pipe and Filter
  - Layered
  - Distributed
  - OO
- 该层次的非形式化是严重的问题吗？

Recommend
火龙果软件

体系结构描述语言的必要性

- 体系结构是降低开发成本的关键
  - 开发焦点转移到粗粒度系统成分
- 形式化的体系结构模型是需要的
- 体系结构描述语言(ADLs)作为一种可能的解决方案被提出

ACME	MetaH
Aesop	Rapide
ArTek	SADL
C2	UniCon
Darwin	Weaves
LILEANNA	Wright

- 若干原型ADLs被开发出来

  →ADL及其扮演的角色仍是一个开放的问题

Recommend
火龙果软件

ADL扮演的角色

- 提供了描述构件及其交互的模型、符号体系和工具
- 支持大规模、高层设计
- 支持体系结构风格的理性选择和应用
- 支持抽象
  - 用户定义
  - 应用特定(application-specific)
- 支持设计的实现
  - 系统化的
  - （可能）自动化的

→语言和环境之间相互作用

- 语言使能精确的规约
- 环境使得规约被复用

Recommend
火龙果软件

ADL描述的例子(1)

- Rapide构件

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

- Wright连接件

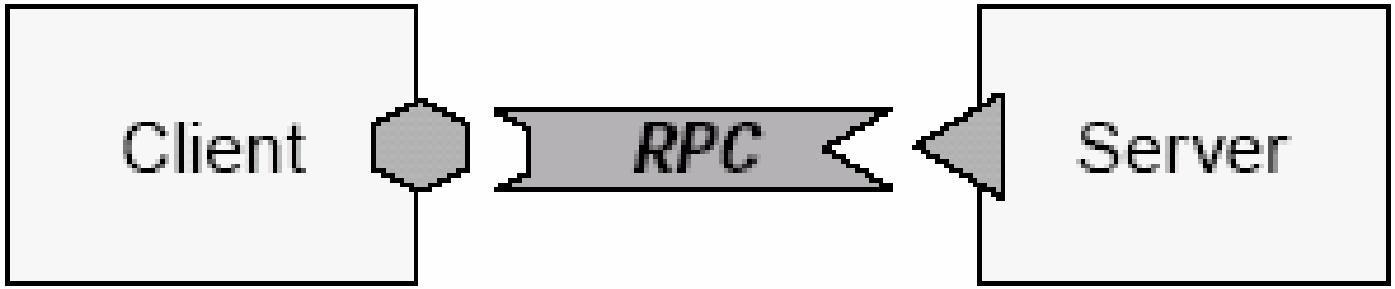
```
connector Pipe =
  role W = write → W ⊓ close → √
  role R =
     let Exit = close → √
     in let DoR = (read → R
                   ▯ read-eof → Exit)
     in DoR ⊓ Exit
```

```
glue = let ROnly = R.read → ROnly
           ▯ R.read-eof → R.close → √
           ▯ R.close → √
       in let WOnly = W.write → WOnly
           ▯ W.close → √
       in W.write → glue
        ▯ R.read → glue
        ▯ W.close → ROnly
        ▯ Reader.close → WriteOnly
```

Recommend
火龙果软件

ADL描述的例子(2)

- ACME描述的体系结构实例




```
System simple_cs = {
    Component client = {Port send-request}
    Component server = {Port receive-request}
    Connector rpc = {Roles {caller, callee}}
    Attachments : {
      client.send-request to rpc.caller;
      server.receive-request to rpc.callee
    }
}
```

Recommend
火龙果软件

ADLs分类的努力

- 现存ADL的考察
  - Kogut and Clements
  - Vestal
- 来自个别系统的观察
  - Luckham and Vera
  - Shaw et al.
- 识别基本的ADL特征
  - Tracz
  - Shaw and Garlan
  - Medvidovic, Taylor, and Whitehead
  - Medvidovic and Rosenblum
- 体系结构交换(interchange)语言
  - ACME

Recommend
火龙果软件

理解ADLs的尝试

- Shaw and Garlan
  - composition
  - abstraction
  - reusability
  - (re)configuration
  - heterogeneity
  - analysis
- Tracz
  - components
  - connectors
  - configurations
  - constraints

Recommend
火龙果软件

ADL定义

- 定义
  - ADL是一种语言，提供了对软件系统概念体系结构进行建模的特性(features)
- 基本特性：对以下成分进行显式规约
  - 构件(components)
    - 接口
  - 连接件(connectors)
  - 配置(configurations)
- 期望的特性
  - 构件、连接件和配置的特定方面
  - 工具支持

Recommend
火龙果软件

区 分ADLs

- 模型化配置的方法
  - 隐式的配置(implicit configuration)
  - 内嵌的配置(in-line configuration)
  - 显式的配置(explicit configuration)
- 关联体系结构和实现的方法
  - 实现约束(implementation constraining)
  - 实现无关(implementation independent)
- 相关符号体系
  - 高层设计符号体系
  - 模块连接语言 (MIL)
  - 面向对象符号体系
  - 编程语言
  - 形式化规约语言

Recommend
火龙果软件

ADL构件

- 定义
  - 构件是计算或数据存储单元
  - 构件是计算和状态的场所
- 所有的ADLs支持构件建模
- 不同的术语
  - 构件(component)
  - 接口(interface)
  - 进程(process)

Recommend
火龙果软件

构件分类范畴

- 接口(**Interfaces**)
  - 模型化要求的(required)和提供的(provided)服务
- 类型(**Types**)
  - 相同的功能能被复用和多重实例化(multiple instances)
- 语义(**Semantics**)
  - 方便分析，强制约束，体系结构不同层次细化的映射
- 约束(**Constraints**)
  - 确保符合预期的构件使用，使用边界和构件之间的依赖
- 演化(**Evolution**)
  - 作为设计元素的构件演化
  - 通过子类型化和细化进行支持
- 非功能属性(**Non-Functional Properties**)
  - 使能运行时行为的模拟，分析，约束，处理器规约和项目管理

Recommend
火龙果软件

ADLs比较：构件

Components	ACME	Aesop	C2	Darwin	MetaH	Rapide	SADL	UniCon	Weaver	Wright
Interface	■	■	■	■	■	■	■	■	■	■
Types	■	■	■	■	■	□	■	□	■	■
Semantics	□	□	■	■	■	■	□	□	□	□
Constraints	□	□	□	□	■	■	□	■	□	■
Evolution	□	□	■	□	□	□	□	□	□	□
Non-Funct. Properties	□	□	□	□	■	□	□	■	□	□

Recommend
火龙果软件

ADL连接件

- 定义
  - 连接件是组成体系结构的“积木”，用于模型化构件之间的交互，以及指导这些交互的规则
- 所有的ADLs支持连接件建模
  - 一些ADLs不把连接件作为第一类实体建模
  - 所有的ADLs至少支持语法连接(interconnection)
- 不同的术语
  - 连接件(connector)
  - 连接(connection)
  - 绑定(binding)

Recommend
火龙果软件

连接件分类范畴

- 接口**(Interfaces)**
  - 确保构件之间正确地连接和通讯
- 类型**(Types)**
  - 抽象和复用复杂的交互协议
- 语义**(Semantics)**
  - 分析构件交互，强制约束，确保一致地细化
- 约束**(Constraints)**
  - 确保符合预期的交互协议，使用边界和连接件之间的依赖
- 演化**(Evolution)**
  - 通过修改或细化现有的连接件，使复用最大化
- 非功能属性**(Non-Functional Properties)**
  - 使能运行时行为的模拟，分析，约束和OTS连接件的选择

Recommend
火龙果软件

ADLs比较：连接件

Connections	ACME	Aesop	C2	Darwin	Meta-H	Rapide	SADL	UniCon	Wright	Weaves
Interface	■	■	■				▒	■	■	■
Types	■	■	■		▒		■	▒	■	■
Semantics	▒	■	▒			■	▒	▒		■
Constraints	▒	▒	▒				▒	■	▒	■
Evolution	▒	■	▒				▒		▒	
Non-Funct. Properties	▒	▒						■		

Recommend
火龙果软件

ADL配置

- 定义
  - **体系结构配置**或**拓扑**是构件和连接件的连接图，描述了系统的体系结构
- 通过定义，ADLs必须显式地建模配置
- 配置有助于确保体系结构属性
  - 合适的连接
  - 并发和分布属性
  - 遵守设计启发规则和风格规则

Recommend
火龙果软件

配置分类范畴(1)

- 可理解性**(Understandability)**
  - 使能相关人员(stakeholders)之间的通讯
  - 系统结构应当仅从配置中体现出来
- 可组装性**(Compositionality)**
  - 在不同细节层次上的系统建模和表示
- 异构性**(Heterogeneity)**
  - 具有各种不同性质的现存成分的大型系统开发
- 约束**(Constraints)**
  - 刻画构件和连接件之间的依赖

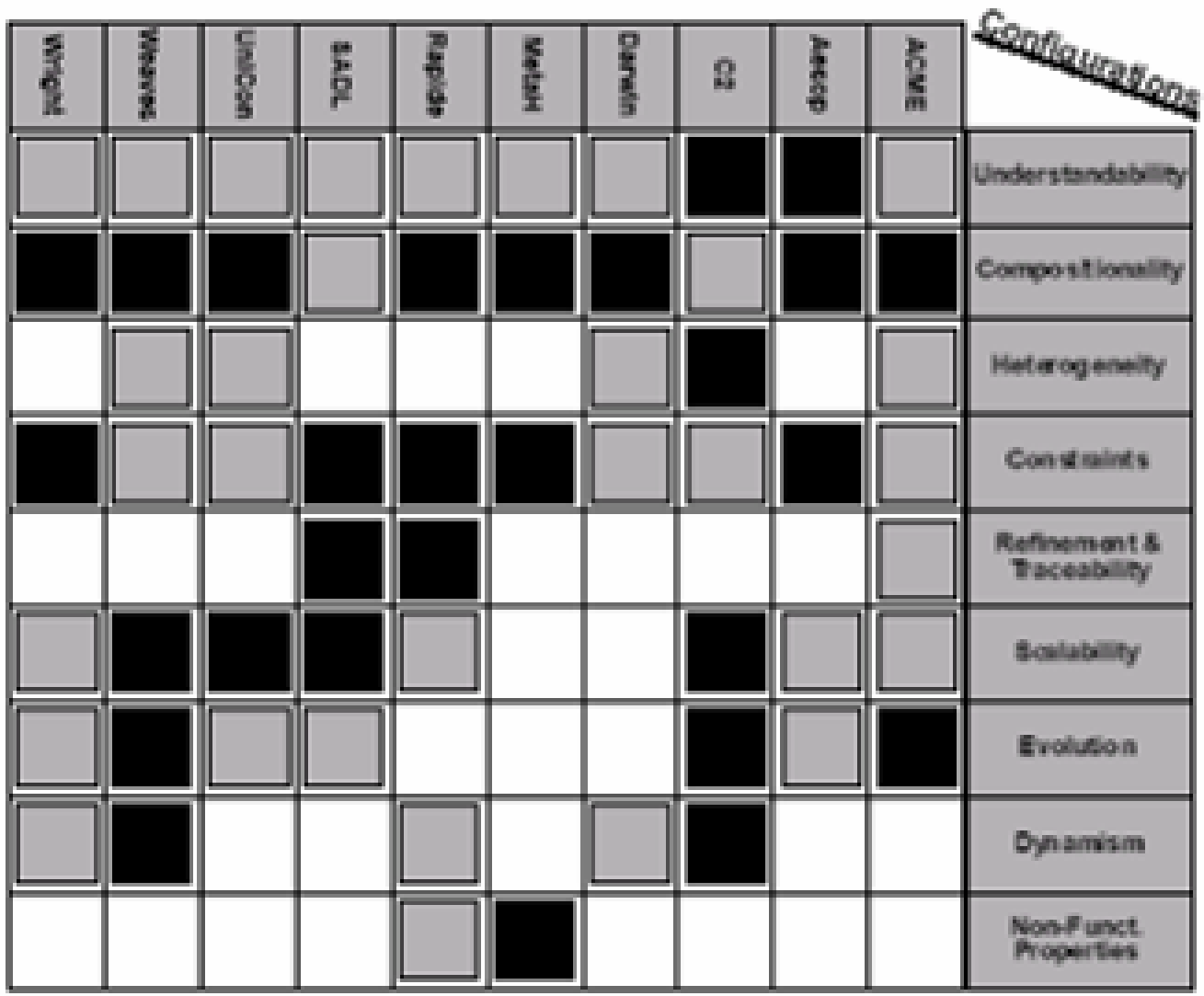
Recommend
火龙果软件

配置分类范畴(2)

- 细化和可追踪性**(Refinement and Traceability)**
  - 在高层模型和代码之间架构桥梁
- 可伸缩性**(Scalability)**
  - 支持对不断增长的系统进行建模
- 演化**(Evolution)**
  - 单个系统或系统家族的演化
- 动态性**(Dynamism)**
  - 使能长期运行的进行运行时修改
- 非功能属性**(Non-Functional Properties)**
  - 使能模拟，分析，约束，处理器规约和项目管理

Recommend
火龙果软件

ADLs比较：体系结构配置



Recommend
火龙果软件

ADL工具支持

- ADLs的形式化使得它们能够被工具操作
  - 工具集不是ADL语言的组成部分
  - 一种ADL的有用性依赖于它对基于体系结构开发的支持
- 每种ADL提供某些工具支持
- 通常聚焦于特定的领域和/或技术
- 有限的总体支持促进体系结构交换的支持
  - ACME

Recommend
火龙果软件

工具支持分类范畴

- 主动规约(Active Specification)
  - 支持设计人员降低认知负担
  - 主动 vs. 被动
- 多重视图
  - 支持不同的相关人员(stakeholders)
- 分析
  - 向前回溯评估大型、分布、并发系统
- 细化
  - 增强正确性和一致性的信心
- 代码生成
  - 体系结构建模活动的最终目标
  - 手工方法导致不一致性和缺乏可跟踪性
- 动态
  - 在执行过程中，体系结构发生变化

Recommend
火龙果软件

ADLs比较：工具支持

Tool Support	ACME	Aesop	C2	Darwin	MetaH	Rapide	SADL	UniCon	Weaver	Wright
Active Specification	■	■	■	■	▒			▒		
Multiple Views	■	■	■	■	■	■		■	▒	▒
Analysis	▒	■	■	■	■	■	▒	▒	■	■
Refinement						■	■			
Code Generation		▒	▒	▒	▒	▒		▒	▒	
Dynamism	▒	■	■	▒		▒			■	

Recommend
火龙果软件

讨论

- 目的：区分不同类型的ADLs
- ADL定义是一个简单的“试金石”
- 若干ADLs跨在边界
  - 实现约束语言(implementation constraining languages)
  - 内嵌的配置语言(in-line configuration languages)
- 在某些领域支持较好，其他领域支持较弱
  - 复杂连接件的实现
  - 非功能属性
  - 细化
  - 动态
- 决定一个ADL的相对“价值”
- 辅助ADLs开发
- 辅助体系结构交换
  - 识别互补的ADLs

Recommend
火龙果软件

ADL实例 — Wright

- CMU开发
- 一个体系结构（系统）包括
  - 构件类型规约
    - 一组端口 (**port**s)
    - 构件行为规约 (**comp spec**)
  - 连接件类型规约
    - 一组角色 (**role**s)
    - 角色协作规约 (**glue**)
  - 类型实例 **(instances)**
  - 实例之间的连接 **(attachments)**
- 所有的规约都是CSP协议
- 用于检测体系结构的死锁

Recommend
火龙果软件

ADL实例 — UniCon

- CMU开发
- UniCon: ***Uni***versal ***Con***nector language
- 预定义的构件和连接件类型
  - 构件类型：模块 (module)，计算(computation)，过滤器 (filter)，进程 (process)，……
  - 连接件类型：管道(pipe)，文件IO (file IO)，过程调用 (procedure call)，数据访问 (data access)，远程过程调用 (remote proc call)，……
- 为构件连接生成“胶水”代码
  - 实现约束
- 允许非功能属性规约

Recommend
火龙果软件

UniCon的语法 (1)

- 一个体系结构是包含以下成分的复合构件 (composite **component**)
  - 原子(atomic)或复合构件类型包括
    - 接口 (**interface**)
      - 预定义的演员(**player**s)和属性(attributes)集合
      - 预定义的每个演员的属性集合
    - 实现 (**implementation**)
  - 连接件类型
    - 协议 (**protocol**)
      - 预定义的角色和属性集合
      - 为每个角色预定义的属性集合
    - 实现 (**implementation**)
  - 构件类型和连接件类型的实例化 (**uses**)
  - 实例的配置 (**connect**)

Recommend
火龙果软件

UniCon的语法 (2)

- UniCon构件演员

```
PLAYER input IS StreamIn
    MAXASSOCS (1)
    MINASSOCS (1)
    SIGNATURE (“line”)
    PORTBINDING (stdin)
END input
```

- UniCon连接件角色约束

```
ROLE output IS Source
    MAXCONNS (1)
    ACCEPT (Filter.StreamIn)
END input
```

Recommend
火龙果软件

UniCon的语法 (3)

- UniCon配置

```
USES p1 PROTOCOL Unix-pipe
USES sorter INTERFACE Sort-filter
CONNECT sorter.output TO p1.source
USES p2 PROTOCOL Unix-pipe
USES printer INTERFACE Print-filter
CONNECT sorter.input TO p2.sink
```

Recommend
火龙果软件

ADL实例 — Rapide

- Stanford Uni.开发
- 集中于动态行为的建模和模拟
  - 偏序事件集 (posets)
  - 基于posets的约束
  - 识别posets的事件模式
- 包含若干子语言
  - 类型 (types)
  - 约束 (constraint)
  - 模式 (pattern)
  - 体系结构 (architecture)
  - 执行 (executable)

Recommend
火龙果软件

Rapide事件模式运算

- Dependent
- Both
- Distinct
- Either
- Independent
- After
- Iteration
- Guarded pattern

  →用于规约动态重新配置

Recommend
火龙果软件

Rapide的语法 (1)

- 一个体系结构包括
  - 构件类型 (**interfaces**)
    - 接口元素(functions)的同步通讯
      - **provides**
      - **requires**
    - 接口元素(events)的异步通讯
      - **in action**
      - **out action**
  - 连接件 (**connections**)
  - 约束 (**constraints**)

Recommend
火龙果软件

Rapide的语法 (2)

- Rapide事件模式

  Evt1(?prm) and Evt2(?prm);

  Evt1→Evt2;

  Evt1 < Evt2;

  Evt1(?prm) where ?prm < 0;

- Rapide构件

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

Recommend
火龙果软件

ADL实例 — C2SADEL

- UC Irvine开发
- C2SADEL: ***C2 S***oftware ***A***rchitecture ***D***escription and ***E***volution ***L***anguage
- 在形式化和实用化之间进行平衡
  - 少量的语言构造成分
  - 语义采用一阶逻辑描述
- 描述C2构件内部对象
  - 不包括requests/notifications的描述
  - Provided服务和required服务相分离
  - 接口(interface)和操作(operations)相分离

Recommend
火龙果软件

C2SADEL语法 (1)

- 一个体系结构包括
  - 构件类型 (**component types)**
    - **subtype**规约
    - 状态变量
    - 构件不变式 **(invariant)**
    - 接口 **(interface)**
    - 行为 (**behavior**)
      - 局部变量 (local variables)
      - 前置条件 (preconditions)
      - 后置条件 (postconditions)
    - 从接口元素到行为操作的映射 **(map)**
      →该映射是一个满射函数

Recommend
火龙果软件

C2SADEL语法 (2)

- 连接件类型 (**connector types**)
  - 消息过滤策略
    - **no filtering**
    - **notification filtering**
    - **message filtering**
    - **prioritized**
    - **message sink**
- 配置 (**architectural topology**)
  - 构件实例 (**component instances**)
  - 连接件实例 (**connector instances**)
  - 实例之间的连接 (**connections**)

Recommend
火龙果软件

变量和基本类型

- 变量声明类似于程序设计语言

  capacity : Integer;

- 变量可以被声明为函数

  well_at : Integer -> Color;

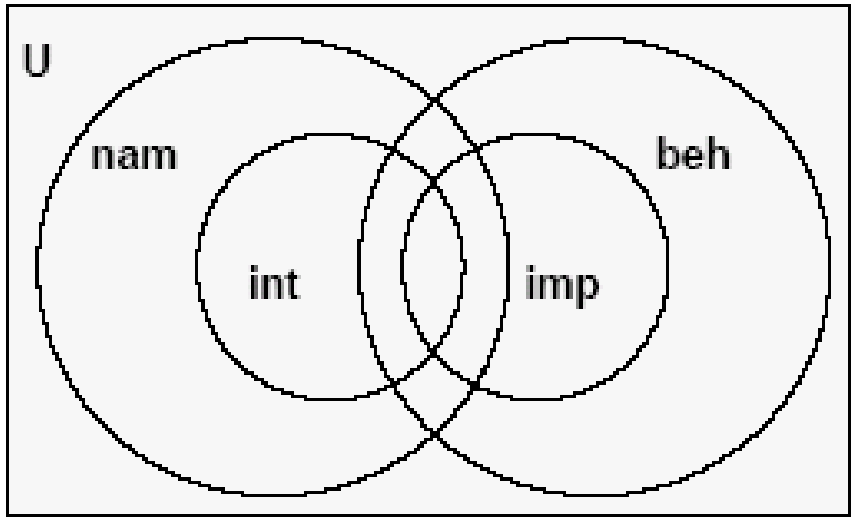
- C2SADEL只支持基本类型的声明
  - 不支持基本类型的语义
- 基本类型之间的子类型关系是允许的
  - 对构件演化是有用的

  Natural is basic_subtype Integer;

Recommend
火龙果软件

构件演化

- 通过子类型化支持构件演化
  - 子类型关系是类型空间中的区域



注：nam − name, int − interface,

beh − behavior, imp − implementation

component WellADT is subtype Matrix (beh)

component WellADT is subtype Matrix (beh \and \not int)

Recommend
火龙果软件

前置条件，后置条件和不变式

- 一阶逻辑公式
- 不变式应用于整个构件
  - 必须只用构件状态变量表达

```
invariant {
        (num_tiles \eqgreater 0) \and
        (num_tiles \eqless capacity);
}
```

- 前置和后置条件应用于单独的操作
  - 可以用构件状态变量或局部操作变量表达

```
pre (pos \greater 0) \and (pos \eqless num_tiles);
post \result = well_at(pos) \and
        ~num_tiles = num_tiles - 1;
```

Recommend
火龙果软件

接口和行为相分离

```
component WellADT is subtype Matrix (beh) {
    state {
        capacity : Integer;
        num_tiles : Integer;
        well_at : Integer -> GSColor; }
    invariant { (num_tiles \eqgreater 0) \and (num_tiles \eqless capacity); }
    interface {
        prov gt1: GetTile (loc : Integer) : Color;
        prov gt2: GetTile (i : Natural) : GSColor; }
    operations {
        prov tileget: {
            let pos : Integer;
            pre (pos \greater 0) \and (pos \eqless num_tiles);
            post \result = well_at(pos) \and ~num_tiles = num_tiles - 1; } }
    map {
        gt1 -> tileget (loc -> pos);
        gt2 -> tileget (i -> pos); } }
```

Recommend
火龙果软件

接口和行为的结合

```
component WellADT is subtype Matrix (beh) {
    state {
        capacity : Integer;
        num_tiles : Integer;
        well_at : Integer -> GSColor; }
    invariant { (num_tiles \eqgreater 0) \and (num_tiles \eqless capacity); }
    services {
        GetTile (loc : Integer) : Color {
            pre (loc \greater 0) \and (loc \eqless num_tiles);
            post \result = well_at(loc) \and ~num_tiles = num_tiles - 1; }
        GetTile (i : Natural) : GSColor {
            pre (i \greater 0) \and (i \eqless num_tiles);
            post \result = well_at(i) \and ~num_tiles = num_tiles - 1; } } }
```

Recommend
火龙果软件

C2SADEL接口元素 → C2消息

- Provided接口元素 → incoming requests
  → outgoing notifications
- Required接口元素 → outgoing requests
  → incoming notifications
- 这是非常简化的通讯！

Recommend
火龙果软件

ADL实例 — ACME

- Acme是一种体系结构交换语言，旨在解决目前出现的众多ADLs之间的不兼容问题
  - Acme提供了一个统一的结构化框架，描述公共的体系结构信息
  - 通过注释的方法，刻划特定于某种ADL的信息
- Acme使用七种成分描述一个系统的体系结构：
  - 构件(components)
  - 连接件(connectors)
  - 端口(ports)
  - 角色(roles)
  - 系统(systems)
  - 表示(representations)
  - 性质(properties)

Recommend
火龙果软件

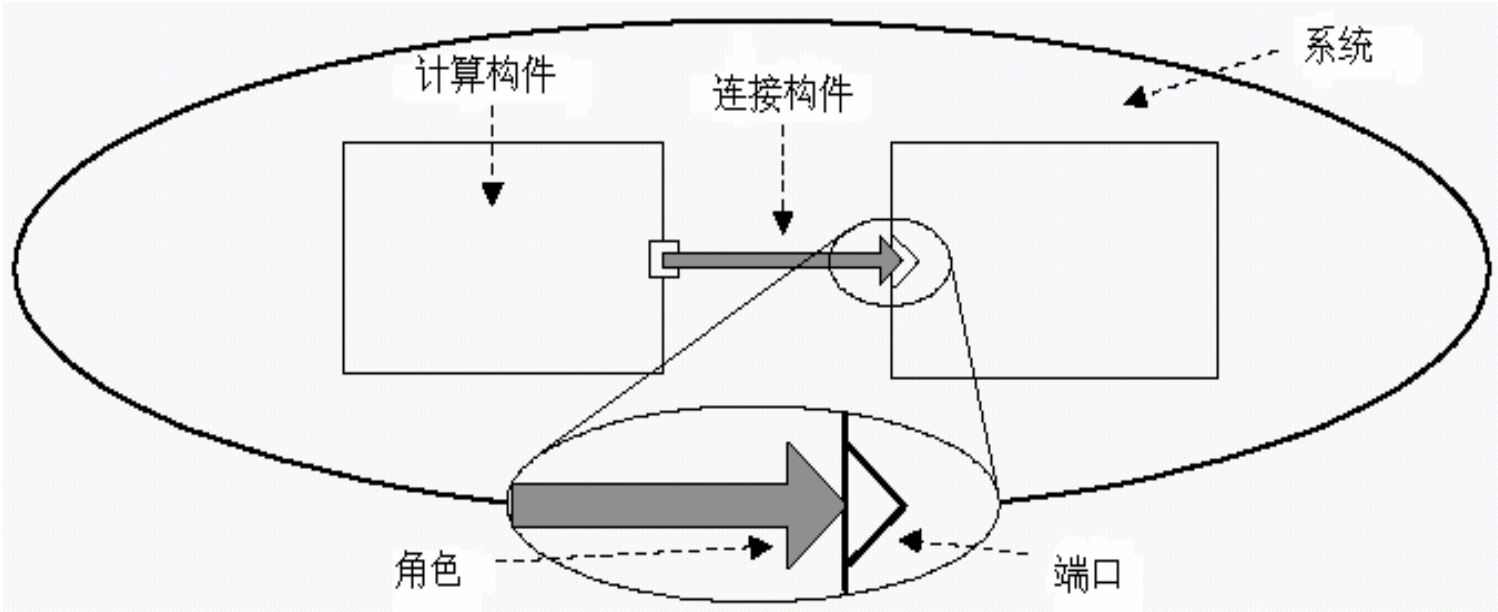
系统成分

- 在Acme中，软件体系结构描述了构件和构件之间的相互关系。
- 构件是实施计算和存放状态的场所，每个构件通过接口规约(interface specification)定义了一组属性，包括功能属性和非功能属性两个方面，每个构件属于某个类型或子类型（例如，过滤器、客户、服务器等）。
- 连接件是表达构件之间关系的场所，每个连接件通过协议规约(protocol specification)定义了一组属性，规定了它能连接的接口类型、事件发生的顺序、对交互的承诺等，每个连接件属于某个类型或子类型（例如，远程过程调用、管道、消息广播、事件等）。
- 打个形象的比喻，每个构件通过接口规范定义了一组“演员”，每个连接件通过协议规范定义了一组“角色”，只有当构件中的“演员”和连接件中的“角色”匹配时，它们才能连接在一起。

Recommend
火龙果软件

系统成分之间的关系

Acme中的主要成分：



Recommend
火龙果软件

ACME的例子

```
System simple_cs = {
   Component client = {
      Port send-request;
      Properties { request-rate : float = 17.0;
            source-code : external-file = "CODE-LIB/client.c" }}

   Component server = {
      Port receive-request;
      Properties { idempotence : boolean = true;
            max-concurrent-clients : integer = 1;
            source-code : external-file = "CODE-LIB/server.c" }}

   Connector rpc = {
      Role caller;
      Role callee;
      Properties { synchronous : boolean = true;
            max-roles : integer = 2 }}

   Attachments {
      client.send-request to rpc.caller ;
      server.receive-request to rpc.callee }
}
```

Recommend
火龙果软件

练习题

1. 阅读材料
   - Mary Shaw, David Garlan, *Software Architecture: perspectives on an emerging discipline*: *P147~P212*
2. 体系结构描述语言同一般程序设计语言的最大区别是什么？

Recommend
火龙果软件

谢谢大家！

Questions or comments?

Recommend
火龙果软件