



# 主动服务模型的形式化描述 与求解方法研究

2006 年 4 月

# 目 录

---

- ✓ 一、研究背景与研究内容
- 二、主动服务模型的形式化描述
- 三、主动服务的实现过程与关键技术
  - 1、资源的虚拟化与组织过程
  - 2、问题的描述与求解过程
  - 3、服务的实例化与执行过程
- 四、主动服务模型的体系结构与相关协议

# 一、研究背景

## 1、网格技术的发展：

网络 → Web → 网格：试图实现Internet上所有资源的互通和共享

## 2、服务网格的相关研究

Web服务成为广域环境下实现互操作的一种主要机制，有助于解决网格中的应用集成、资源共享、系统互操作和标准化等问题。

已有的研究：OGSA，WSRF等

## 3、软件体系结构和构件技术的发展

构件技术的发展：粒度更大，功能级别的构件

软件体系结构研究：从软件实体本身更多地转向实体间的动态协同

## 4、用户需求的变化

用户应用需求的智能化、个性化和综合化的趋势，

# 前期的相关研究

## 互联网中主动服务模型的研究

——立足于提高互联网应用的智能化程度，使其能够主动适应用户需要和应用环境的变化。

- 1、互联网中主动服务的概念、模型与结构
- 2、主动服务的实现方法
- 3、实现原型系统



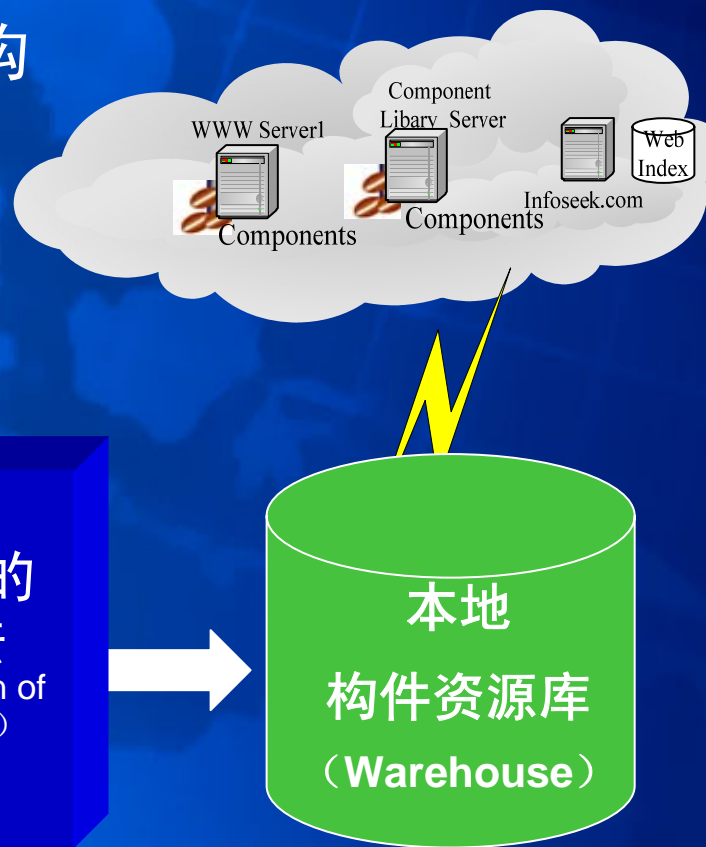
主动服务  
人机接口  
(The Interface for  
Active Services)



主动服务的  
实现方法  
(Implementation of  
Active Services)



本地  
构件资源库  
(Warehouse)





# 目 录

---

一、研究背景与研究内容

✓ 二、主动服务模型的形式化描述

三、主动服务的实现过程与关键技术

1、资源的虚拟化与组织过程

2、问题的描述与求解过程

3、服务的实例化与执行过程

四、主动服务模型的体系结构与相关协议

# 主动服务的形式化描述

## 几个概念的引入和定义

### 相互关系

### 形式化定义：

服务资源

服务构件子集

功能子集

功能集合

服务需求

定义 1: 服务资源集合  $S_{Set} = \bigcup_{i=1}^n s_i$  , 即是所有类型资源的总和。

定义 2: 功能集合  $F_{Set} = \bigcup_{j=1}^m f_j$  , 其中  $f_1 \cap f_2 \dots \dots f_{m-1} \cap f_m = 0$

定义 3: 功能子集  $F_{subset} = \prod_{i=k}^l f_i$  , 即功能子集是功能集合  $F_{Set}$  的一个子集合, 同时, 这些子功能之间又有一定的逻辑关系, 而不是简单的联合。

定义 4: 服务构件子集  $S_{subset} = \prod_{i=k}^l s_i$  , 即服务构件子集是服务资源集合  $S_{set}$  的一个子集合, 与功能子集一样, 这些子服务构件之间有一定的组装关系。

定义 5: 用户的服务需求 P

# 主动服务问题的描述

给定：

- (1) 一个服务资源集合  $S_{set}$
- (2) 一个用户的服务需求  $p$  ,

问题：

在服务资源集合  $S_{set}$  , 找到合适的服务构件子集  $S_{subet}$  , 使得服务构件子集  $S_{subet}$  能够对应于用户服务需求  $p$  的所有功能需求, 并通过对  $S_{subet}$  的解释执行, 最终满足用户的服务需求  $p$

# 主动服务求解的过程

- 1、给定服务集合，导出其功能集合
- 2、如何根据用户需求，在已有的功能集合中，导出一个功能子集
- 3、根据功能子集，导出服务构件子集，并定义相互调用关系
- 4、执行该服务构件子集，最终满足用户的服务需求

## 求解过程：

(1)  $\exists S_{set}$ ，如何导出  $F_{set}$ ，使得  $F_{set}$  包含服务资源集合  $S_{set}$  中所有服务所提供的功能。

(2)  $\exists F_{set}$ ， $\exists p$ ，如何根据用户的服务需求  $p$ ，导出某一个功能子集  $F_{subset}$ ，使得  $F_{subset}$  能够满足  $p$ 。

(3)  $\exists F_{subset}$ ，如何导出某一个服务构件子集  $S_{subset}$ ，使得  $F_{subset}$  中的每一个子功能  $f_i$  在  $S_{subset}$  中都有一个服务构件  $s_j$  一一对应，且服务构件子集  $S_{subset}$  中服务构件之间的组装关系与功能子集  $F_{subset}$  中定义的相一致。

(4)  $\exists S_{subset}$ ，如何解释执行，并最终满足用户的服务需求。



# 目 录

---

一、研究背景与研究内容

二、主动服务模型的形式化描述

✓ 三、主动服务的实现过程与关键技术

1、资源的虚拟化与组织过程

2、问题的描述与求解过程


3、服务的实例化与执行过程

四、主动服务模型的体系结构与相关协议

# 主动服务的求解过程

## □ 如何导出构件、服务等资源的功能集合


- 1、搜索、挖掘网格环境中的各类资源
- 2、服务化（将各类资源包装成服务）和统一描述
- 3、功能提取和聚类
- 4、形成功能集合和功能语义网络



资源的虚拟化与组织过程

## □ 如何导出能满足用户需求的功能子集


- 1、用户需求的理解与描述
- 2、用户需求的功能分解
- 3、确定子功能之间的逻辑关系，最终导出功能子集



问题的描述与求解过程

## □ 如何从功能子集导出相应的服务子集，并解释执行，最终生成能满足用户需求的服务

- 1、服务的实例化（抽象功能需求映射到具体服务构件）
- 2、服务组装方案的形成
- 3、服务的执行与监控



服务的实例化与执行过程

# 一、资源的虚拟化与组织过程

过程1:

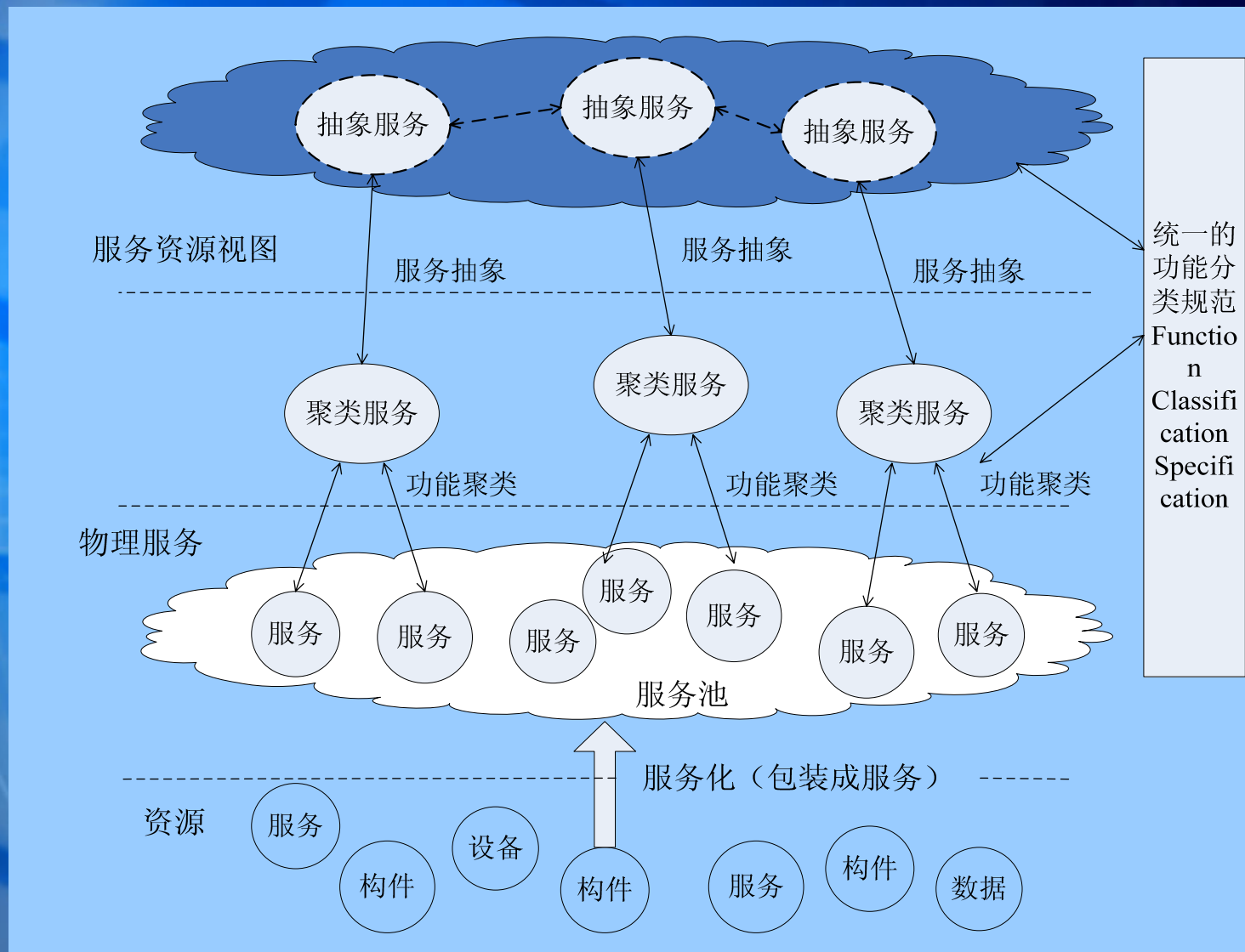
资源的服务化

过程2:

服务功能聚类

过程3:

服务功能语义  
关系定义



# 过程1：资源的服务化

过程 1： 资源的服务化过程

$\forall r_i \in R_{set}, 1 \leq i \leq n$

If  $r_i \in WS$  then  $r_i \rightarrow R_{set}$

else

$GridServices(r_i \vee deploy(r_i)) \rightarrow R_{set}$

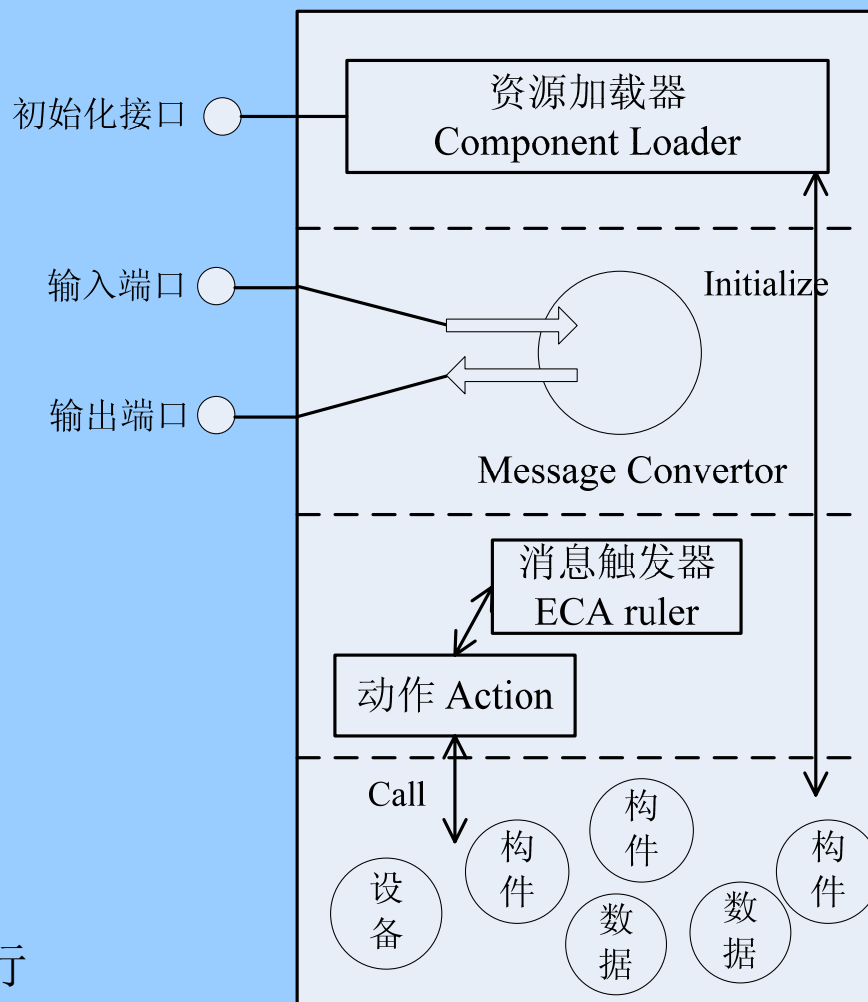
Endif

注：

$Deploy(r)$ ：将资源r在网格环境中进行部署

$GridServices(r \& deploy(r))$ ：

将资源r在网格环境中进行部署后，并进行服务化的包装，我们采用的是服务构件代理机制



服务构件代理结构图



## 过程2、基于刻面分类的服务功能聚类方法:

- ❑ 功能聚类是以服务资源的分类为基础的。
- ❑ 刻面分类是我们对服务资源采用的分类方法 → 刻面, 术语, 术语空间
- ❑ 在聚类抽象过程中, 术语构成了一个聚类的基本单元。我们将在功能刻面上术语取值相同的服务实例组织在一起形成一个服务目录项。
- ❑ 服务目录项再按照刻面术语空间中的层次结构进行组织, 形成服务功能聚类目录树。其中每一个服务目录项链接的服务, 就是经过聚类后的一组功能相同的服务。

### 过程 2: 服务的功能聚类过程

$\forall s_i \in S_{set}, 1 \leq i \leq n$

If (  $\exists funcItem_k \in FS, map(Func(s_i), funcItem_k) = TRUE$  )

Then

$ServiceCluster(s_i, funcItem_k)$

else

(  $New(funcItem, m + 1)$  **and**  $funcItem_{m+1} = Func(s_i)$  )

**and**  $ServiceCluster(s_i, funcItem_k)$

Endif

## 过程3：形成服务功能语义Ontology

服务功能语义Ontology：将聚类后的抽象服务与服务之间的语义关系组织起来（如同一功能的不同实现例），形成一个具有层次结构的服务关系网络。

**过程 3：** 形成服务功能语义 Ontology

（1） 抽象服务

$AS(funcItem_k) = \bigcup s_i$ , 其中对于  $\forall s_i, map(Func(s_i), funcItem_k) = TRUE$

（2） 抽象服务之间的语义关系定义：（前置关系  $\rightarrow$ ，并行相关  $\leftrightarrow$ ，置换关系//）

（3） 服务功能语义 Ontology = (  $A_{ASsubset}$ ,  $A_{Message}$ ,  $A_{Scenario}$  )

在这些相关定义的基础上，抽象服务之间的语义关系定义过程如下：

$\forall as_i \in AS_{set}, \exists funcItem_i, as_i = AS(funcItem_i)$ , 其中  $1 \leq i \leq n$ ,

$\forall as_i, as_j \in AS_{set}$ , 其中  $1 \leq i, j \leq n$

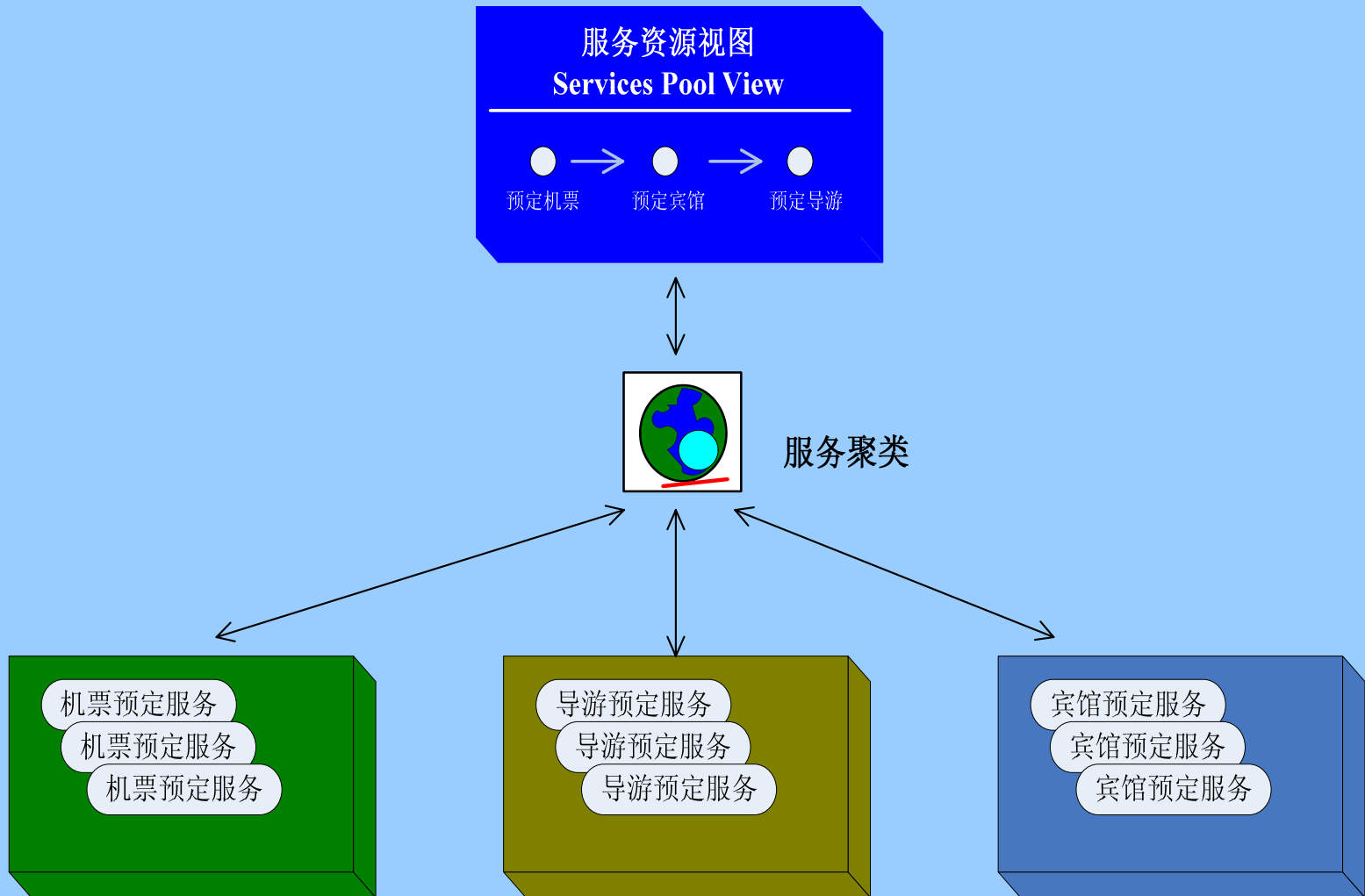
If (  $Relation(as_i, as_j) = TRUE$  )

Then

**Add** ( (  $as_i, as_j, Relation(as_i, as_j), Message(as_i, as_j)$  ) , Ontology )

Endif

# 举例



## 二、问题的描述与求解过程

用户/应用的计算需求  $\longrightarrow$  待解决的问题 (Problem)

实现该需求的服务程序  $\longrightarrow$  问题的解决方法 (Solution)

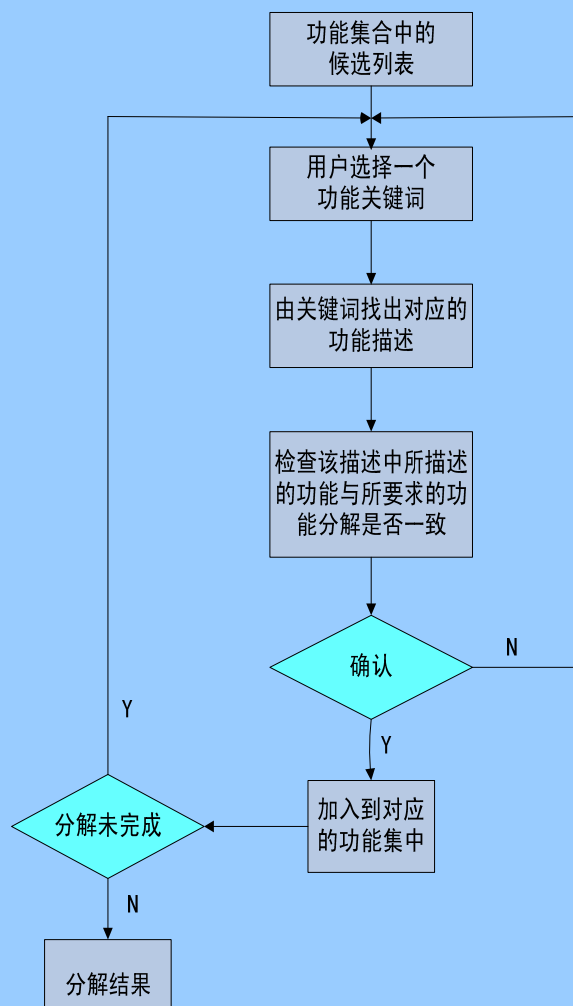
—— 主动服务就是一个 问题求解过程 (Human problem-solving process)

问题的描述： 三层描述模型

Level-1:	ProblemName: [ID: ProblemID	<u>需求输入界面</u>
	Classification: 分类信息	
	Description: 问题的描述	
	KeywordList: 关键词列表	
	SubProblem: {SubProblemIDSet}]	
Level-2:	SubProblem: [ID: SubProblemID	<u>问题分解算法</u>
	Signature: InputType $\rightarrow$ OutputType	
	Restriction: Invariants, Pre-, Post- Conditions]	
Level-3:	Component Composition Schemes	<u>服务组装方案</u>



# 递归分解算法



对应的算法框图

*If*  $\exists f_i \in F_{set}, Satisfy(f_i, P_{total}) = TRUE$

*Then* Add( $f_i$ )  $\rightarrow F_{subset}$

*Else If*  $\exists f_i \in F_{set}, Satisfy(f_i, P_{part}) = TRUE$

*Then*

Add( $f_i$ )  $\rightarrow F_{subset}$  ;

Recursive Solve(Decompose( $P_{total}$ ) -  $P_{part}$ );

*Else*

*Break*;

Endif

其中:

*Recursive -solve*: 递归求解算法

*Decompose*: 需求分解算法

# 二、服务的实例化与执行过程

## 1、抽象服务的实例化: (抽象服务到具体服务构件的映射)

采用服务代理机制，服务构件代理主要实现以下功能：

- ❑ 向组装工具提供所代理功能体的统一的访问方法。
- ❑ 根据用户对同一功能的不同要求，自动选用不同的实现例，来适应用户需求的动态变化。
- ❑ 功能体调用中的异常处理。

过程 1：抽象服务的实例化

*If* ( $\forall f_i \in F_{subset}, (i \geq 1)$ ): MappedAService( $f_i, P$ )  $\rightarrow as_i$

Then choose  $s_k$  in  $as_i$

satisfying Qos Condition( $Qos_p, s_k$ ) = true

MappedService( $f_i, P$ )  $\rightarrow s_k$

endchoose

if (Available ( $s_k$ )=FALSE)

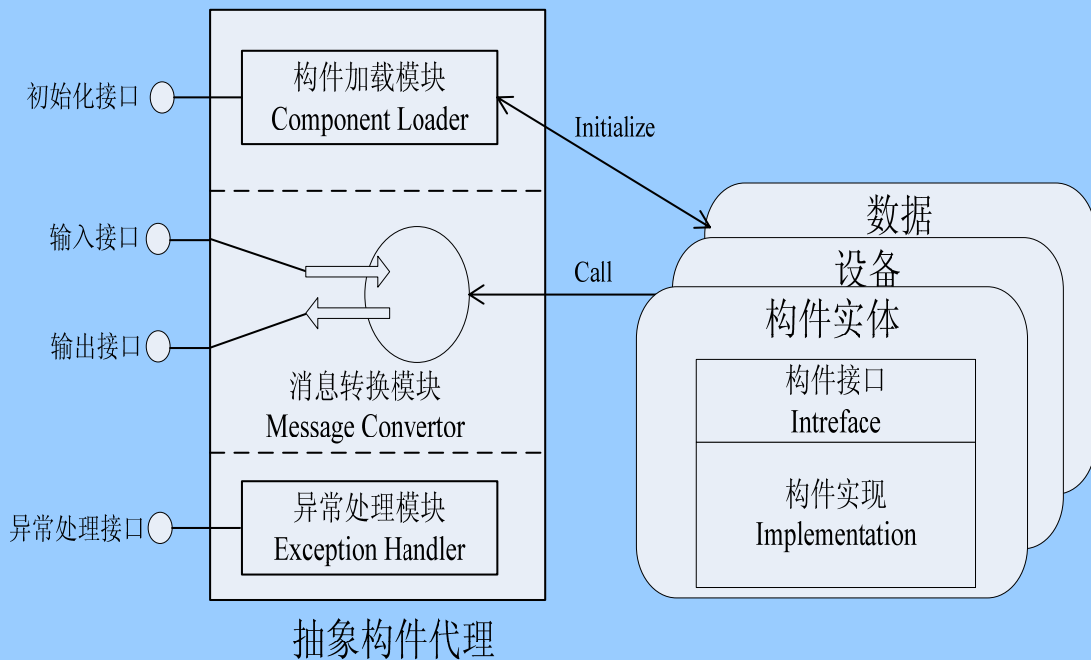
state( $P$ ) := failure; go end;

else

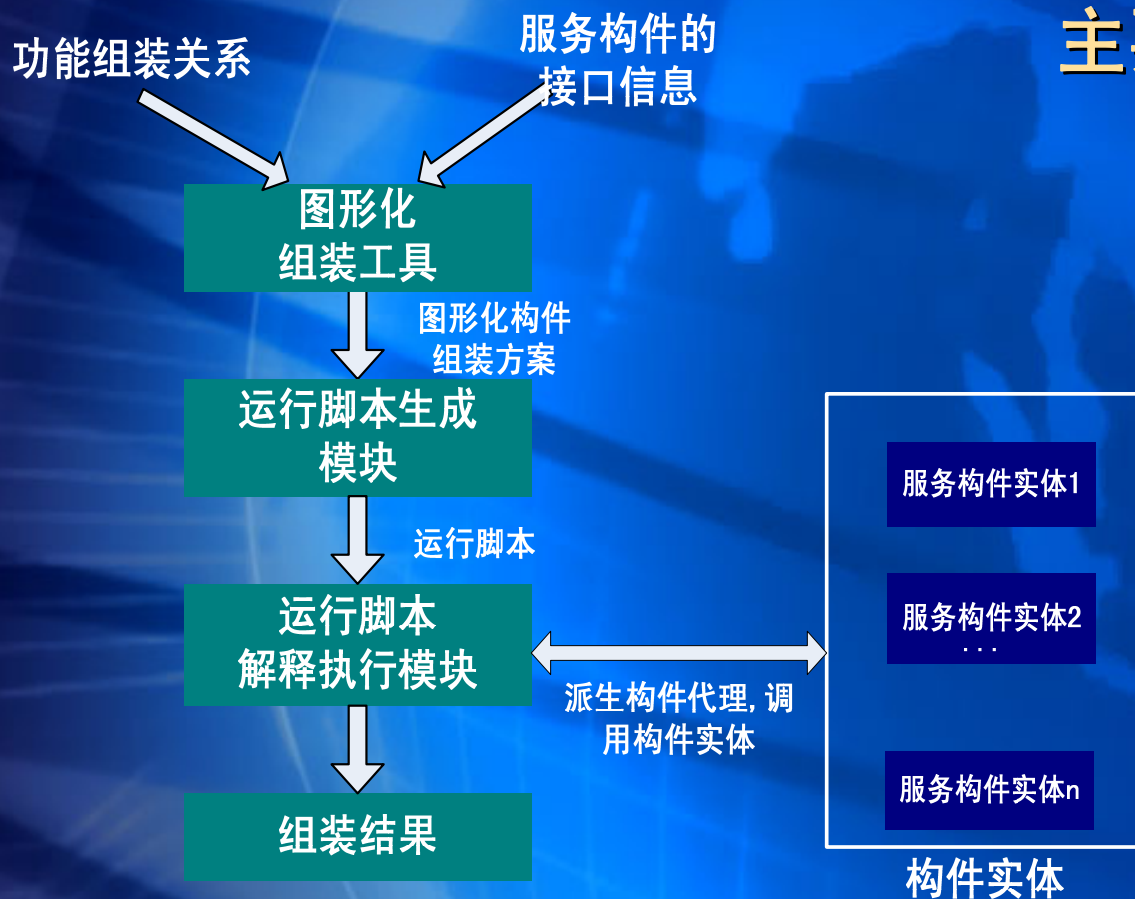
Delploy ( $s_k$ )

Endif

Endif



## 2) 图形化的服务构件组装



### 主要模块

- 图形化建模工具
- 运行脚本生成模块
- 运行脚本执行环境

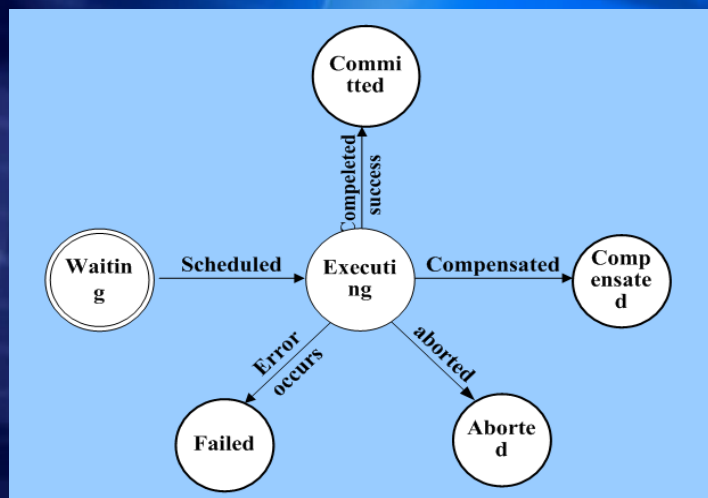
图形化构件组装过程示意图

## 2) 执行与监控

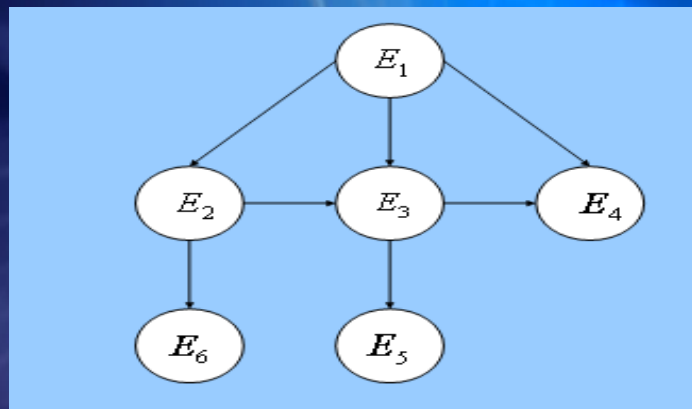
### ➤ 引入了主动服务执行状态模型

服务执行过程中的六种状态:

(Waiting, Executing, Failed, Aborted, Committed, Compensated)



### ➤ 深度优先调度算法



算法：服务构件执行调度算法

输入：

输出：SUCCESS 或 FAIL

```
{
    初始化服务构件依赖关系图 G;
    /* 从 G 的根节点出发以深度优先遍历方式生成调度方案 */
    DFS_Load (G, R, *Status_Load = 1);
    Return (Status_Load)
}

/* 服务构件加载深度优先递归算法 */
void DFS_Load (Graphic G, int v, int *Status_Load)
{
    if (!*Status_Load)
        return;
    visited[v] = TRUE;
    加载服务构件 v 对应的服务至执行环境中;
    if (服务加载成功)
    {
        将其状态置为 Waiting;
        在服务构件依赖图中查找出其依赖的所有服务构件;
        if (其所有依赖服务构件对应的服务均已成功提交)
            然后执行该服务;
    }
    else
    {
        *Status_Load = 0;
        if (加载服务构件 v 的重要度 == Vitality)
        {
            向所有其他服务构件发送 abort 请求并对已经成功提交的服务构件进行补偿;
            释放资源;
            退出整个服务;
        }
        else
            return;
    }
    for (w = FirstAdive (G, v); w; w = NextAdive (G, v, w))
        if (!visited[w])
            DFS_Load (G, w);
}
```



# 目 录

---

一、研究背景与研究内容

二、主动服务模型的形式化描述

三、主动服务的实现过程与关键技术

1、资源的虚拟化与组织过程

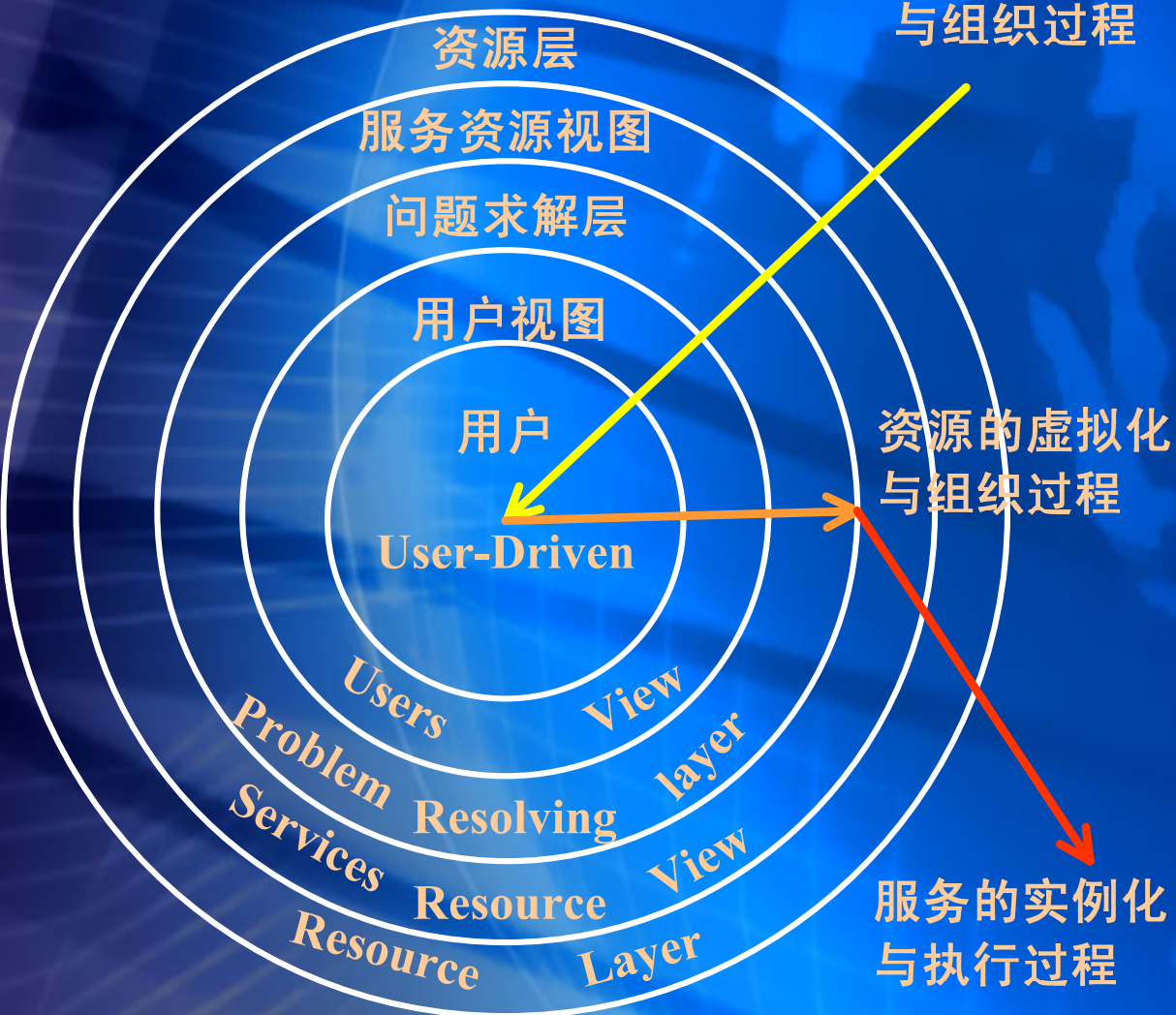
2、问题的描述与求解过程

3、服务的实例化与执行过程

✓ 四、主动服务模型的体系结构与相关协议

# 主动服务模型的层次结构图

## 网格计算环境



## 用户视图

UserView = <Users, UseCase, Relations, Requirements>

## 问题求解层:

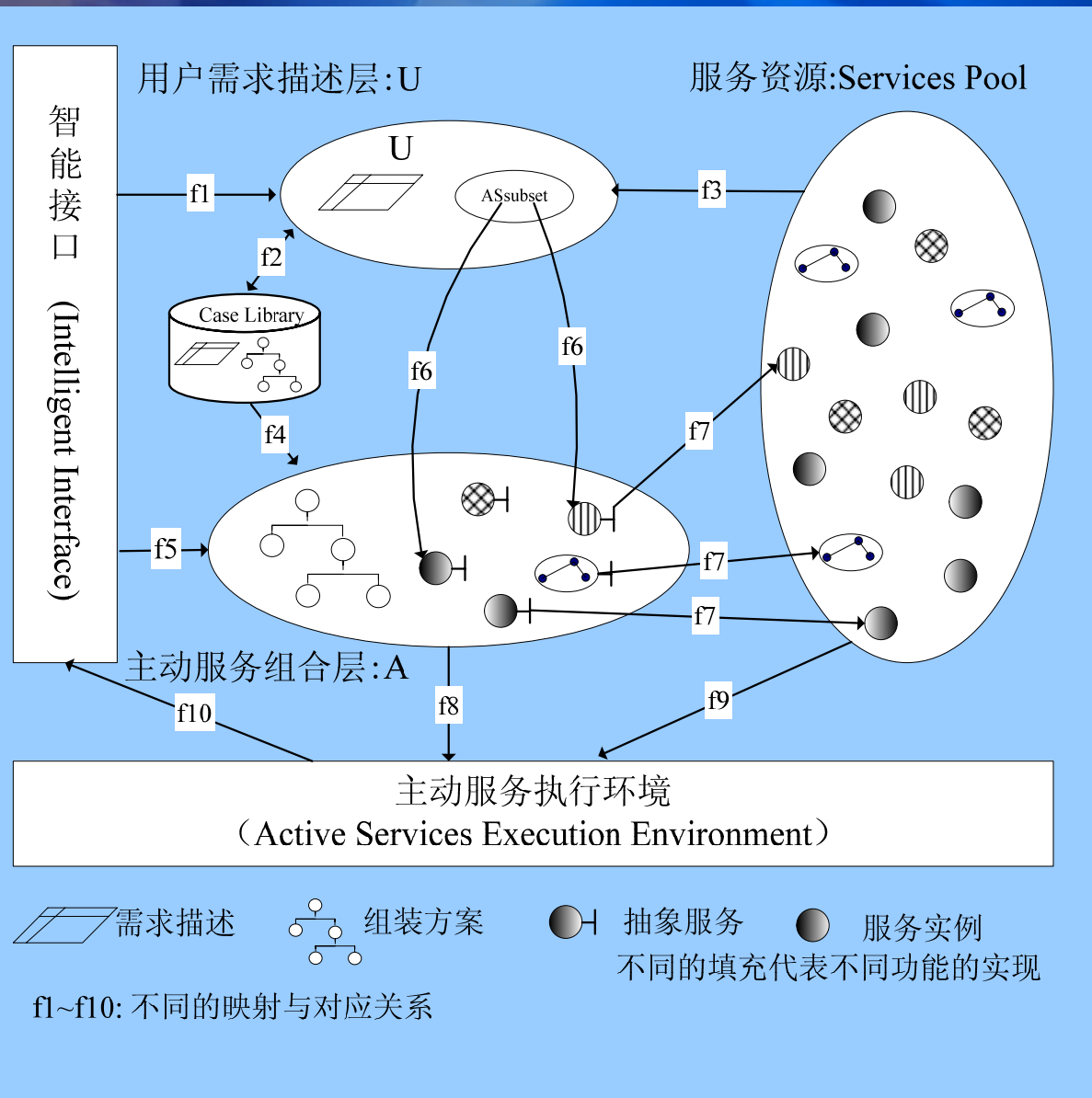
- 需求描述语言
- 功能分解算法
- 领域分析工具
- Case推理工具

## 服务资源视图

## 资源层:

- 各类网格资源的总和（技术资源，网络资源，构件资源，服务资源等）

# 主动服务求解过程的概念示意图



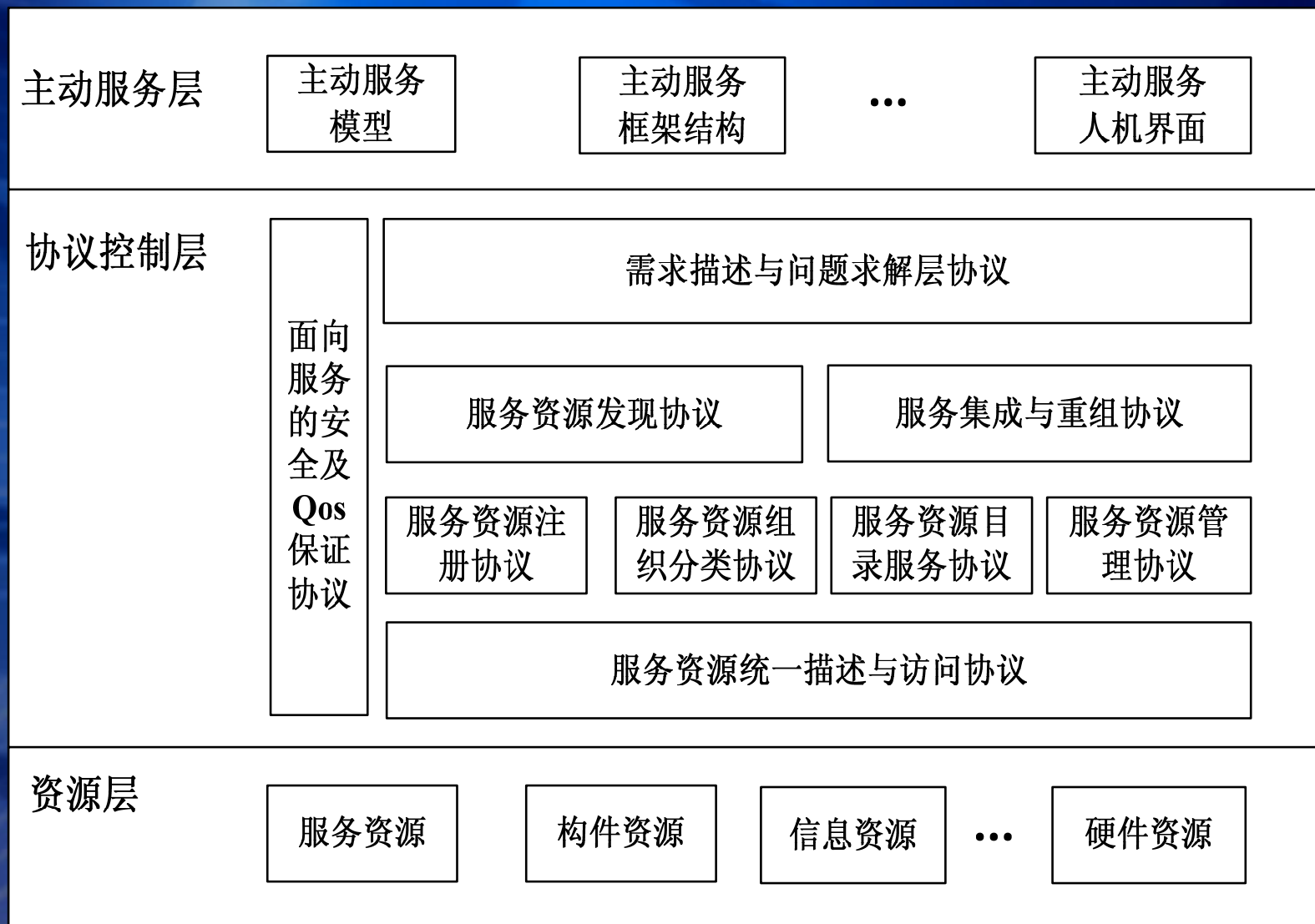
## 二个视图:

- 1、用户视图的需求定义
- 2、服务资源视图的功能分解与服务获取

## 三个过程:

- 1、资源的虚拟化与组织
- 2、问题的描述与求解
- 3、服务的实例化与执行

# 主动服务模型的相关协议结构图

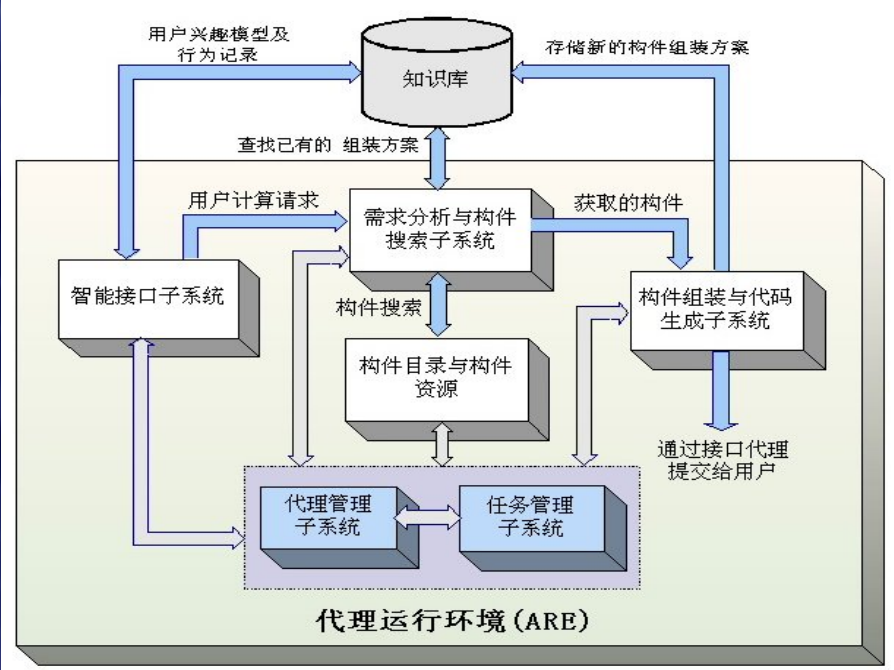




# 以上述算法和工具为基础，实现了相应的原型系统

## 软件著作权登记：

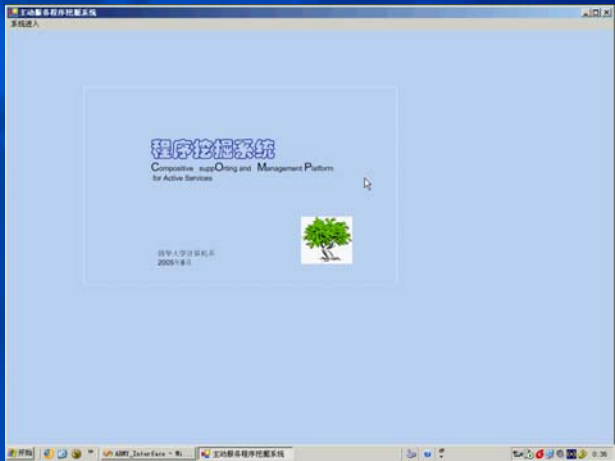
- 1、主动服务集成化管理与支撑平台软件
- 2、主持主动服务的构件库软件



程序挖掘原型系统模块划分



支持主动服务的构件库系统

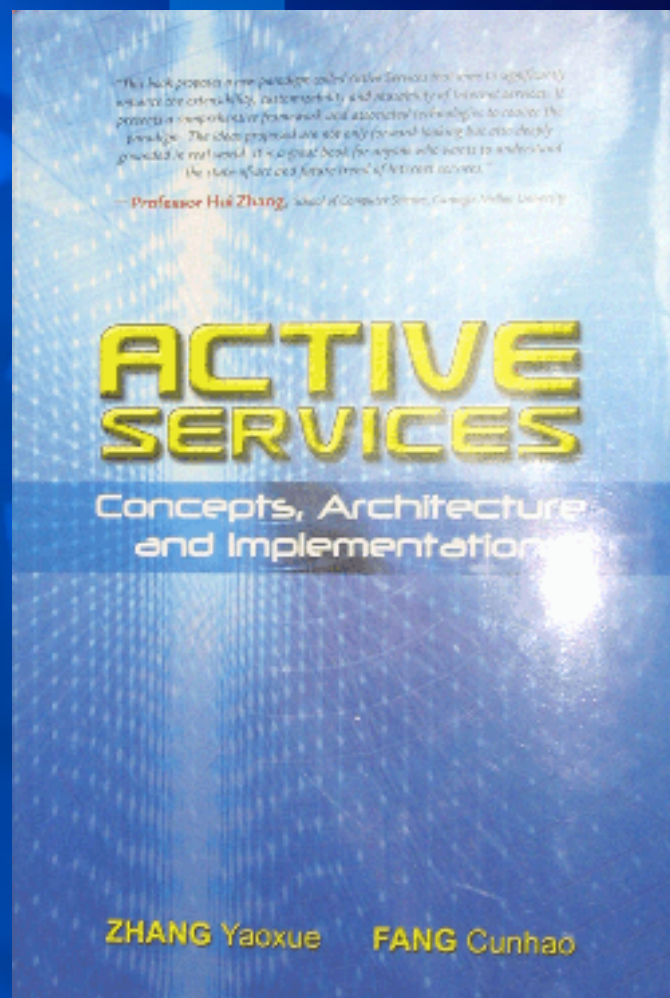


主动服务原型系统

# 中、英文学术专著



《主动服务——概念、结构与实现》  
(中文版：科学出版社)



Active Services: Concepts, Architecture and  
Implementation (英文版：Thomson Learning)



The background is a deep blue with a subtle, lighter blue grid pattern. A faint, semi-transparent globe is visible in the center, showing the outlines of continents. In the top right corner, there are three small squares: one light blue, one medium blue, and one dark blue, arranged in a small cluster.

*Thanks !*