



LarKC 架构简介

LarKC架构概述

- ◆ LarKC的结构设计和原型实现是协同进行的。LarKC的结构最初被设计为的简单的线性pipeline，现已演化为更加灵活的workflow方式。
- ◆ Pipeline / workflow 采用灵活的插件结构，这种结构便于设计和测试各种新的推理技术。
- ◆ LarKC中的推理过程被分解为 workflow 中的若干步,每步对应一个插件，整体对应一个 workflow，以解决大规模知识库上的推理问题。
- ◆ LarKC的目标是设计一个开放的，灵活的，可扩展的平台，又能够满足LarKC用例需求。因此要求灵活性和性能的折衷。

LarKC架构组成

◆ LarKC平台的主要组成部分：

◆ 数据层及其API

- ◆ 用于存储和交换数据

◆ workflow支持系统

- ◆ 使Decider插件能建立、执行和管理LARKC workflow

◆ 插件API

- ◆ 用于和外部插件交互

◆ 资源适配器

- ◆ 屏蔽处理异构的和分布式运行环境的复杂性，充分利用不同的计算资源来运行平台和插件

◆ 插件注册组件

- ◆ 用于保存可用插件的信息

版本一 (v1)

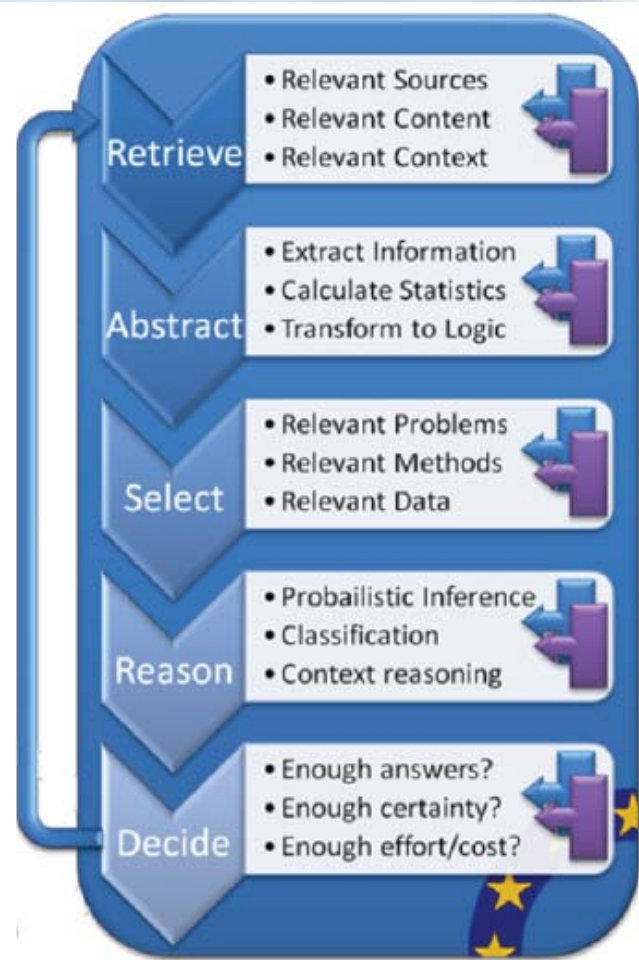


Figure 1 Initial LarKC plug-in architecture

- 最初的设计：线性 pipeline 结构，把整个推理过程分解到五个插件中，是一个循环迭代过程，由最后的 Decider 插件判定是输出结果还是继续迭代。
- 该设计仅支持所有步骤的重复循环，不能灵活的控制每一步。

版本二 (v2)

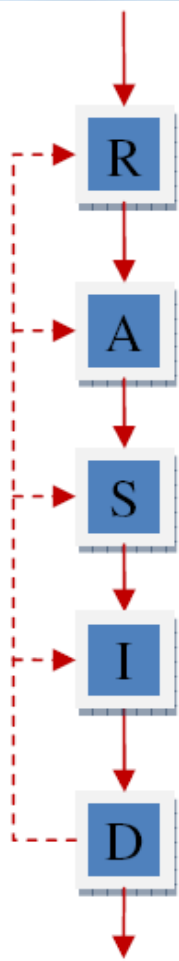


Figure 2 LarKC plug-in architecture v2

- v2的设计是对v1不足之处的改进。
- 图中各字母含义：
R(Retrieve,检索),
A(Abstract, 提取),
S(Select, 选择),
I(Inference, Reason, 推理), D(Decide, 判定)。
- 实线表示数据流, 虚线表示控制流。
- 该设计的问题: 判定只能在两步之间进行。

版本三 (v3)

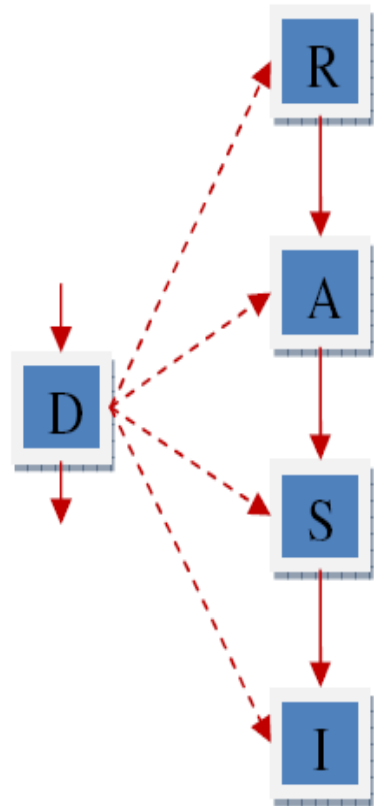


Figure 3 LarKC plug-in architecture v3

- v3对v2的改进。
- Decider成为一个“元”插件，可以在插件工作流程中的任意点进行判定和控制。
- v1到v3的过渡发生在项目早期，此时还没有稳定的原型系统。

版本三（v3） -- 插件名变更

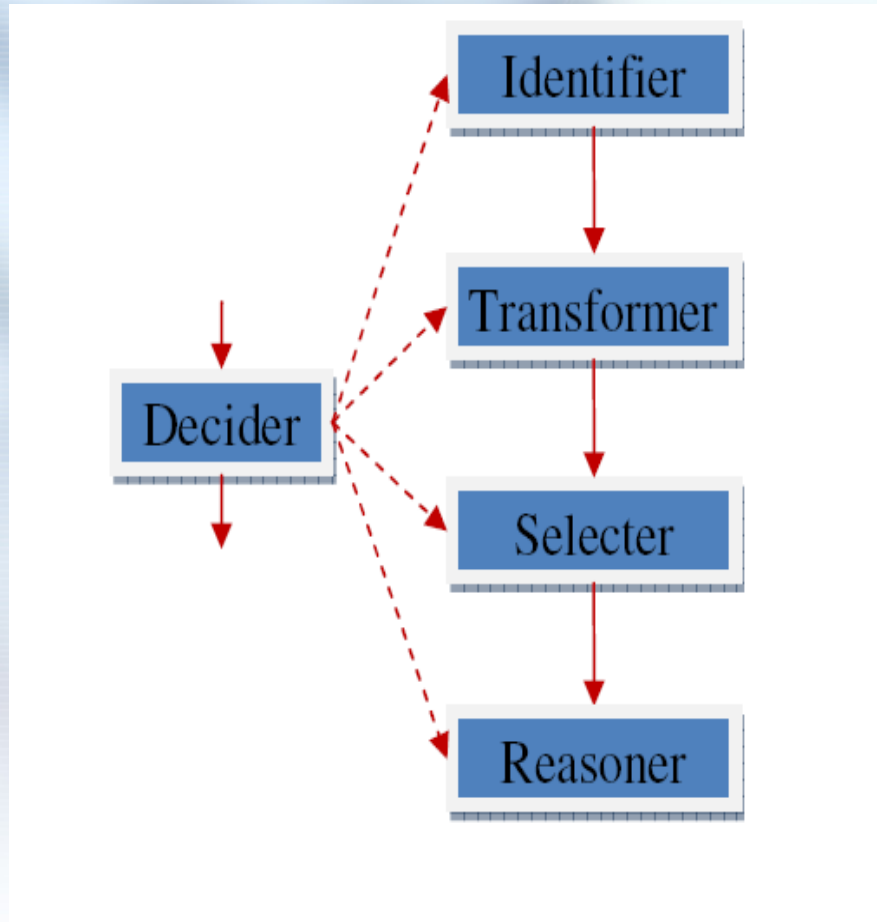


Figure 4 Plug-in architecture v3 with new plug-in names

- 插件名称有所变更，以更好的反映其功能。
- 这一版本成为设计第一版原型的基本结构。

版本四 (v4)

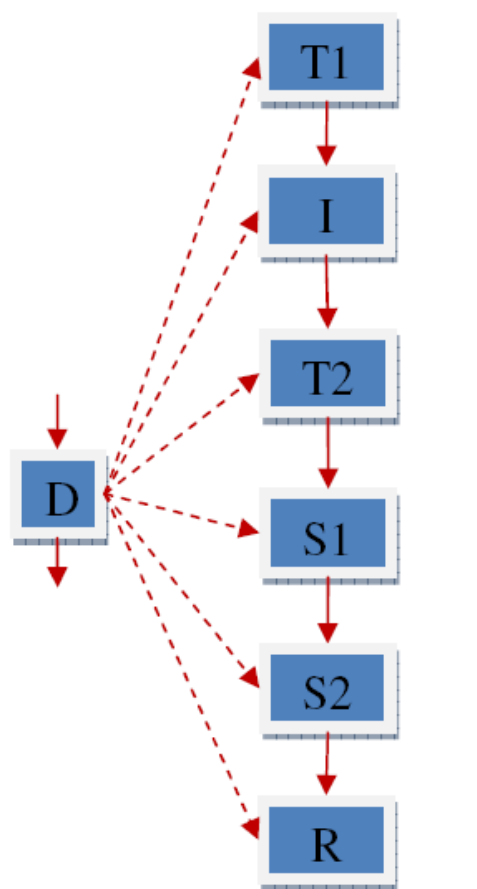


Figure 5 LarKC plug-in architecture v4

- 对v3的改良，主要基于开发新插件和工作流的需求，特别是用例和数据层开发的需求。
- 改良了用于在插件间传递数据的数据层结构，使之更加灵活通用。
- workflow 组建更灵活，允许 workflow 中包含多个同类型插件。已在 LarKC alpha 1 release 中实现。

版本五 (v5)

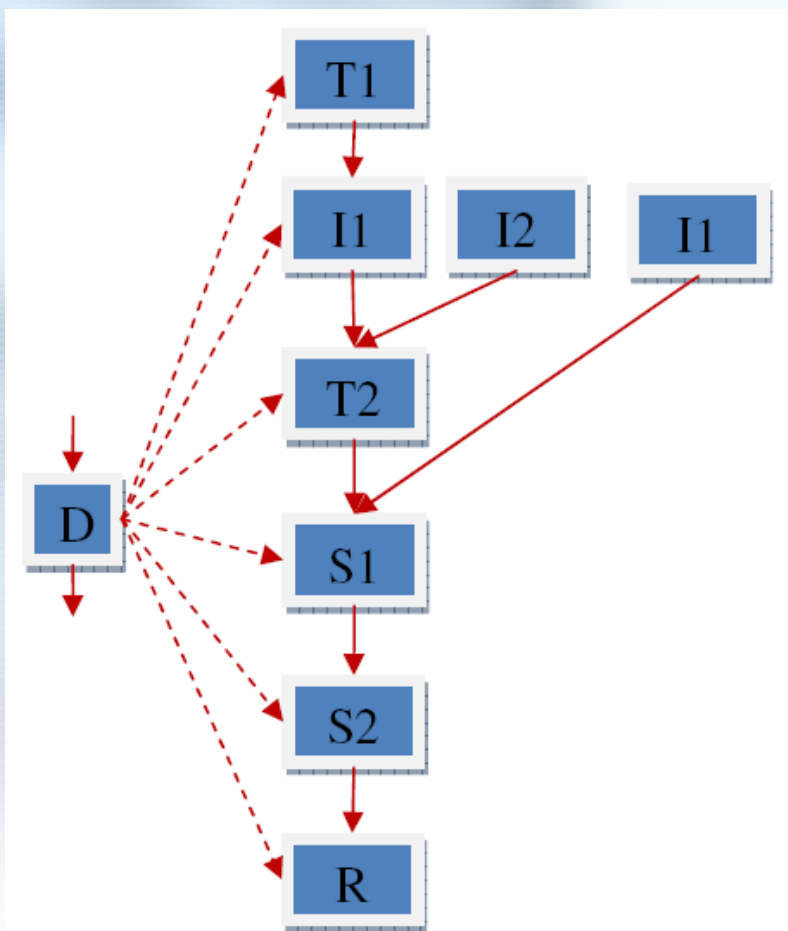


Figure 6 Parallel LarKC plug-in architecture (v5)

- 某些用例需要数据的“多路”功能 (multiplexers)，即能够对数据进行合并/分割。
- v5的设计支持树形 workflow，能够对各插件结果进行合并/分割，从而支持支持并行任务。

总体结构

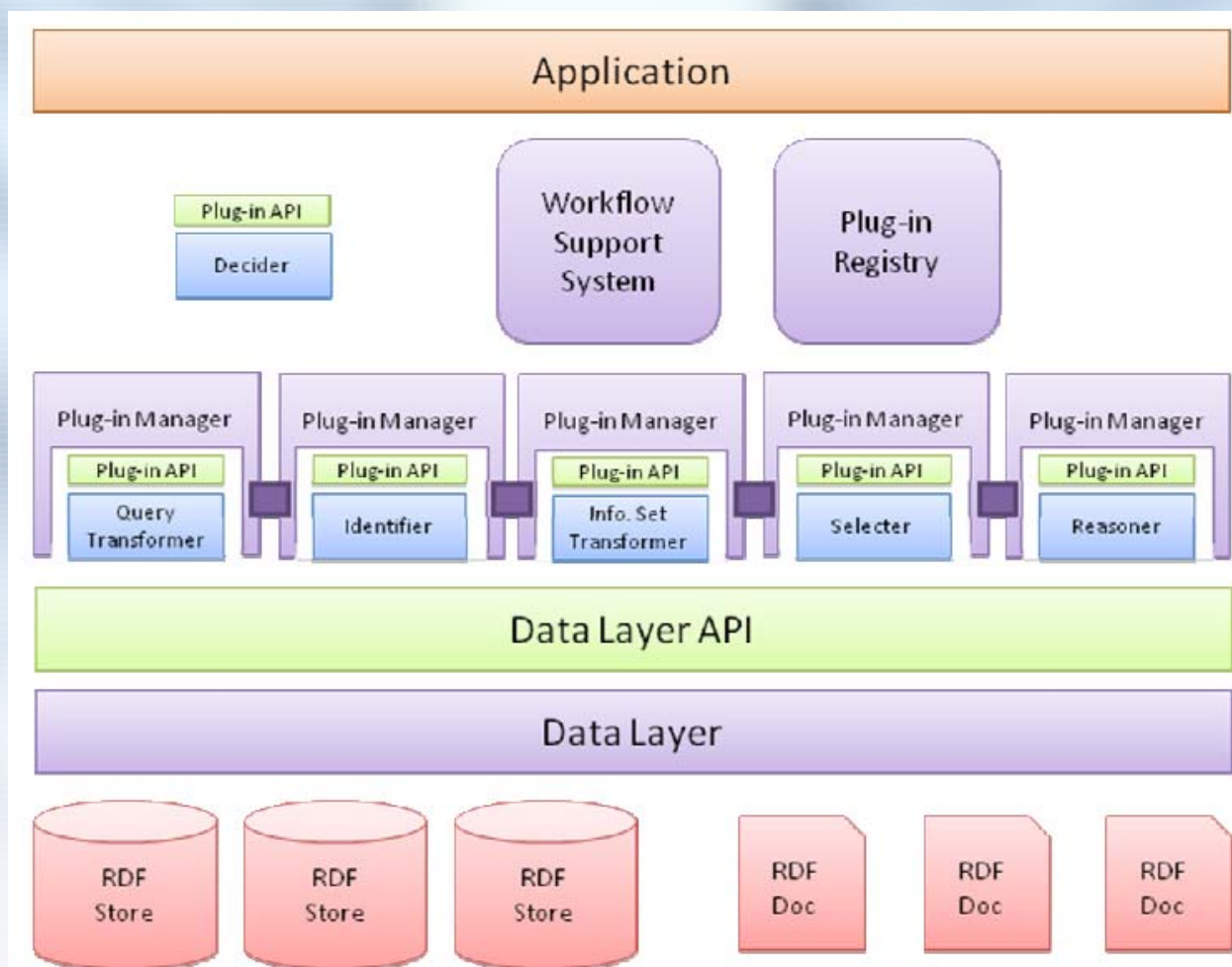


Figure 7 Architecture Overview

总体结构

- ◆ LarKC把推理过程分解为若干步，每步对应一类插件。
- ◆ 插件组合成 workflow，处理大规模知识库上的推理任务。
- ◆ 平台已提供组建、实例化、监视和控制 workflow 所需要的支持。

平台与插件的关系

- ◆ 为便于部署和扩展，对平台与插件在功能上进行了彻底的区分。
- ◆ 插件通过plug-in API连接到平台，通过Data Layer来访问数据。插件和数据源可以在同一台本地机器上，也可以是分布式的，支持远程访问。
- ◆ 平台本身不包含任何插件，推理功能来自于工作流中一组插件的结合。

层次结构

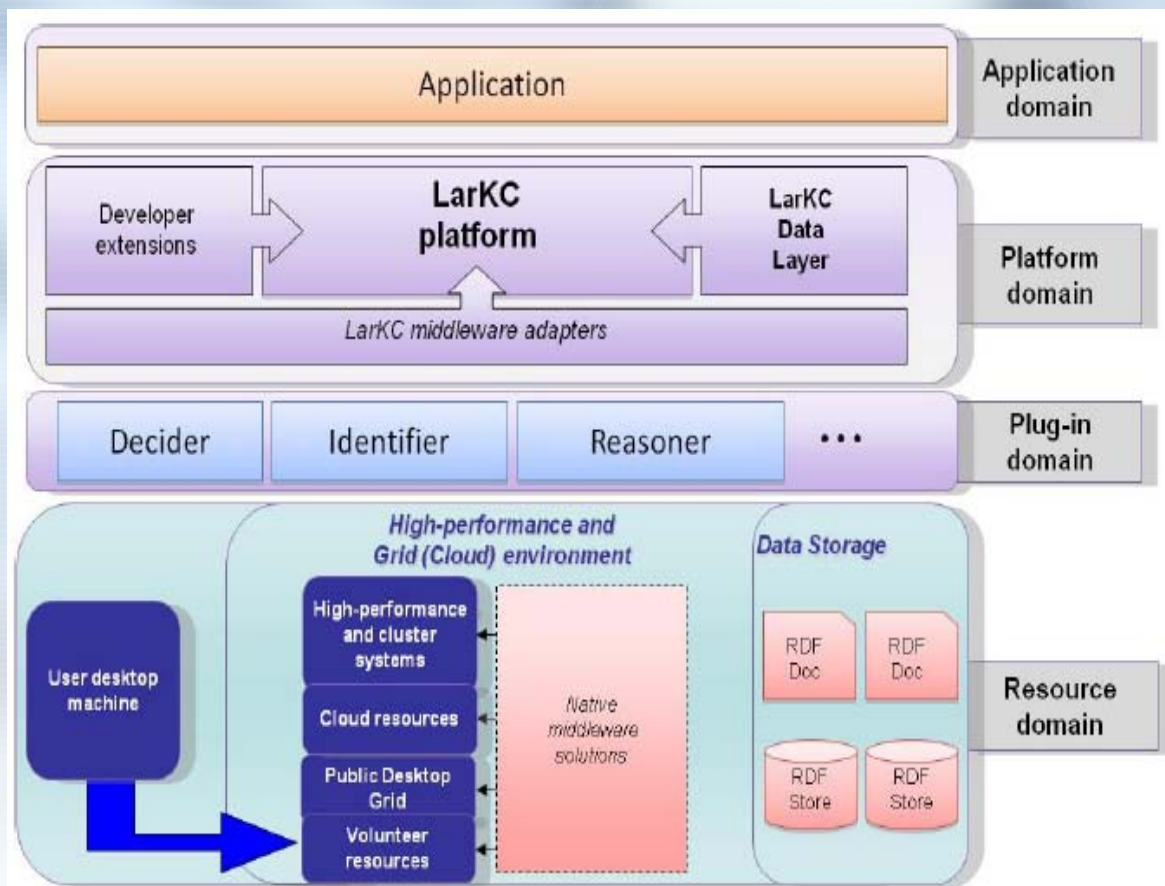


Figure 8 The hierarchical view of the LarKC architecture

WP6, WP7a, WP7b
负责用例开发，
主要工作在应用
域和插件域。

WP2, WP3, WP4
，负责方法研
究，主要工作
站插件域。

WP5负责整体
架构设计和平
台开发，主要
工作在平台域。

层次结构

- ◆ 应用程序/用户域：包括最终用户交互的应用程序接口，用户包括与LarKC交互的外部系统。
- ◆ 平台域：包括平台工具（插件注册组件和工作流支持系统，包括插件管理器和其他工具），数据层，数据层API，插件API，支持分布式和远程执行的组件（LarKC中间件适配器），及其他可能扩展（如开发者扩展，外部程序开发者的应用可能需要支持）。
- ◆ 插件域：包括插件本身，插件作为独立组件存在，通过插件注册组件注册到平台中。
- ◆ 资源域：包括运行工作流和插件的计算和存储设备，及数据集（包括临时存储和缓存）；插件和平台通过数据层API访问外部数据源，通过中间件适配器访问计算资源。

平台域子系统和接口

- ◆ **Data Layer**

规定和实现了供所有插件使用的通用数据模型。该模型的定义依从W3C规范和推荐。数据层实现了LarKC数据模型规范。

- ◆ 对于海量RDF数据的有效存储；ORDI数据模型的评价实施；评估与评价数据流；可分解的RDF数据识别；通过标准SPARQL端点的数据索取；可选的前向链推理；允许流数据处理的接口；能够查询远端数据或者多路数据集的实用方法。

Data Layer API

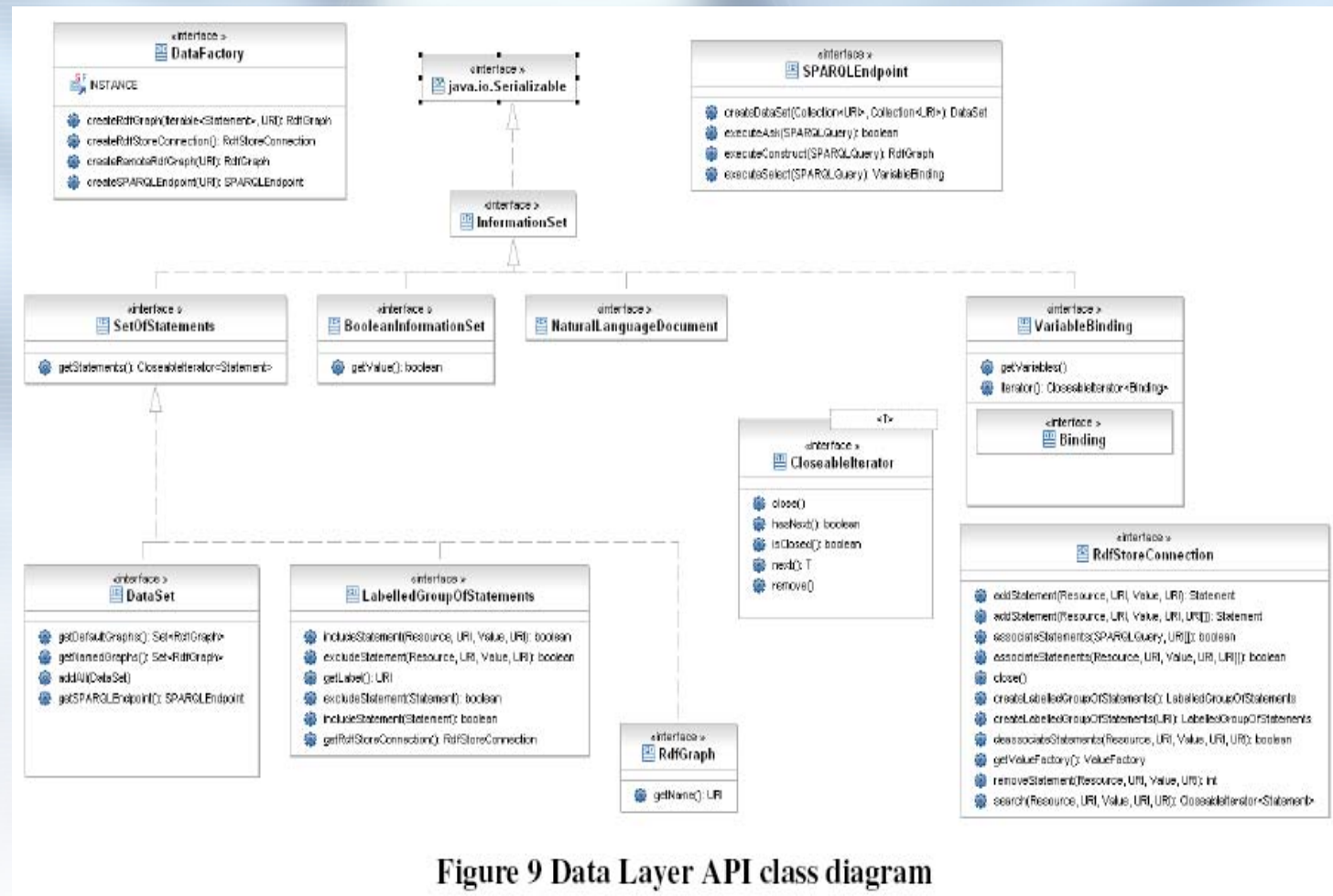


Figure 9 Data Layer API class diagram

Data Layer API

- ◆ SetOfStatement是所有RDF数据类型的接口

```
public interface SetOfStatements extends InformationSet {  
    public CloseableIterator<Statement> getStatements();  
}
```

Figure 10 The most abstract RDF data type

内存需求估计

使用不同的SetOfStatement实现，传递1亿条RDF语句所需要的内存大小：

- ◆ -SetOfStatementsImpl，默认内存实现，值传递，约需内存350MB；
- ◆ -RdfGraphImMemory，默认命名图实现，值传递，约需内存350MB；
- ◆ -HTTPRemoteGraph，RUI检索的命名图，引用传递，只需几KB；
- ◆ -RdfGraphDataSet，从远程/本地SPARQL终端检索的命名图，引用传递，只需几KB；
- ◆ -DataSet，从远程/本地SPARQL终端检索的命名图的集合，引用传递，只需几KB；
- ◆ -LabelledGroupOfStatementsImpl，按各种标准任意选出的一组RDF语句，引用传递，只需几KB，需要可兼容ORDI的实现

。

插件API

- ◆ 插件API: 目的在于提供所有插件开发者必须遵守的一致接口, 好处是插件更容易组合形成 workflow, 必要时也更容易插接或替换其他插件。
- ◆ 所有LARKC插件实现一个通用接口, 该接口提供对所有插件类型公共部分的访问。
- ◆ 所有的插件用URI作为标识名, 并且提供QoS(服务质量)信息, 关于它们怎样执行他们提供的功能。

```
public interface Plugin {  
    public URI getIdentifier();  
    public QoSInformation getQoSInformation();  
}
```

Table 1 The Plugin Interface from the LarKC Plug-in API

Identify

- ◆ 根据给定查询，确定需要的信息集。能从所有可用信息集中找到能回答查询的信息集，缩小推理任务的范围

```
public interface Identifier extends Plugin {  
    public Collection<InformationSet> identify(Query q, Contract c,  
        Context co);  
}
```

Table 2 The Identifier Interface from the LarKC Plugin API

Transform

- ◆ 把数据从一种表示形式转换为另一种。在LARKC API中。
- ◆ 转换分两类：对查询的转换，和对信息集的转换。所以有两种接口：
QueryTransformer和
InformationSetTransformer

Transform API

```
public interface QueryTransformer extends Plugin {  
    public Set<Query> transform(Query q, Contract c, Context co);  
}
```

Table 3 The QueryTransformer Interface from the LarKC Plugin API

```
public interface InformationSetTransformer extends Plugin {  
    public InformationSet transform(InformationSet i, Contract c,  
    Context co);  
}
```

Table 4 The InformationSetTransformer Interface from the LarKC Plugin API

Select

- ◆ 从输入中创建一个陈述 (statements) 集

```
public interface Selector extends Plugin {  
    public SetOfStatements select(SetOfStatements s, Contract c,  
    Context co);  
}
```

Table 5 The Selector Interface from the LarKC Plugin API

Reason

- ◆ 在提供的陈述集上执行给定的SPARQL查询，支持SPARQL查询的四种标准方法：
select(选择), describe(描述), construct(构建), ask(询问)
- ◆ select方法：返回变量绑定输出，变量对应于查询中的变量
- ◆ describe方法：返回RDF图，包括描述查询变量的三元组
- ◆ construct方法：返回RDF图，根据查询构建
- ◆ ask方法：返回布尔信息集，如果查询中的模式被找到返回真，否则假

The Reason Interface

```
public interface Reasoner extends Plugin {  
    public VariableBinding sparqlSelect(SPARQLQuery q,  
    SetOfStatements s, Contract c, Context co);  
    public SetOfStatements  
    sparqlConstruct(SPARQLQuery q, SetOfStatements s,  
    Contract c, Context co);  
    public SetOfStatements  
    sparqlDescribe(SPARQLQuery q, SetOfStatements s,  
    Contract c, Context co);  
    public BooleanInformationSet  
    sparqlAsk(SPARQLQuery q, SetOfStatements s,  
    Contract c, Context co);  
}
```

Table 6 The Reasoner Interface from the LarKC Plugin API

Decide

- ◆ 负责建立和维护 workflow（包含着其他类型的插件），管理插件间的控制流
- ◆ 接口类似于推理插件，因为 LARKC 是一个大规模推理平台，故也可以看作一个 reasoner，主要的区别在于实际用来推理的数据没有显式地指出。
- ◆ 因为 decider 是 LARKC 平台的外部接口，故没有接口中不提供 contract/context 信息，而是通过最终用户提供服务质量参数来指导 decider 选用合适插件回答查询，并且知道何时可以停止。例如，用户可以指定最少返回多少结果，workflow 最多花多少时间回答查询

The Decider Interface

```
public interface Decider extends Plugin{  
    public VariableBinding sparqlSelect(SPARQLQuery q,  
    QoSParameters p);  
    public SetOfStatements  
    sparqlConstruct(SPARQLQuery q, QoSParameters p);  
    public SetOfStatements  
    sparqlDescribe(SPARQLQuery q, QoSParameters p);  
    public BooleanInformationSet  
    sparqlAsk(SPARQLQuery q, QoSParameters p);  
}
```

Table 7 The Decider Interface from the LarKC Plugin API

插件注册

- ◆ 插件注册是**LARKC**平台的一个部分，为平台存储和提供已注册的插件及其位置，可访问性和描述。

Plug-in Registry自身可分为三部分：

- ◆ -插件和插件本体输入(plug-in ontology importer)
- ◆ -存储输入数据的知识库（**KB**）
- ◆ -用于检索插件数据的**API**

插件注册

- ◆ 第一部分在新插件注册到平台时启用。关于插件的 **SAWSDL** 描述文件,须阅读 **D1.3.1**,描述形式类似于 **web services**, **LARKC** 平台从描述文件中抽取插件的名称、位置、类型和一个指向 **RDF** 文件的指针,此 **RDF** 文件包含插件本体,插件的输入输出、行为、速度、服务质量信息,及平台需要的其它详细描述。
- ◆ 第二部分: 插件信息从存储在 **Plug-in Registry** 的内部 **KB** 中的 **SAWSDL** 和 **RDF** 中检索得到。这部分数据与平台数据层不在一起,数据层专用于存储插件要处理的数据。内部 **KB** 和 **LARKC** 本体一起预先载入, **LARKC** 本体使插件本体的导入和扩展更容易。插件注册 **KB** 建立在 **Cyc** 的 **KB** 结构上,此结构可缓存,易操作,访问和检索快,为推理认为做准备,几乎没有额外代价。

插件注册API

<code>Collection<Plugin ></code>	<code>askPluginQuery(String _query)</code> <p>Result of this method is the list of the plug-ins, based on the query passed as input. The result can already be the workflow if the query is asking for that, or just plain list, for example the list of all Reasoners.</p> <p>This method can already take an advantage of the platform's internal reasoner (see section 3.4.1) and answer more complicated queries based on QoS and other plug-in parameters.</p> <p>The query have to be in CycL format, but this is soon to be switched with SPARQL</p>
--	---

(Partial)Table 8 The Plug-in registry API, taken from Javadoc

内部Reasoner

- ◆ 由OpenCyc所用的reasoner衍生而来，有同样的能力，但只有与LARKC相关的模块，与插件注册KB直接相关，提供在已导入的插件本体上推理的元推理能力。对于插件开发者来说，外显为askPlugInQuery方法。

workflow支持系统

- ◆ LARKC平台内的一个工具集合，作用是使Decider插件能够构建、执行和管理LARKC workflow，目标是使decider插件作者用存在的LARKC插件快捷容易的建立 workflow，主要包括：
 - ◆ 插件管理器
 - ◆ 插件实例化
 - ◆ 队列与multiplexer
 - ◆ 日志