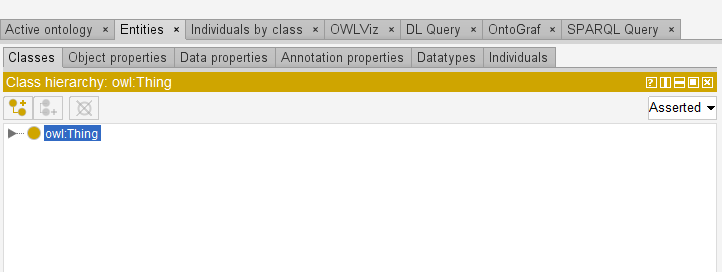
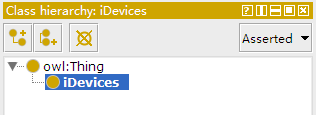
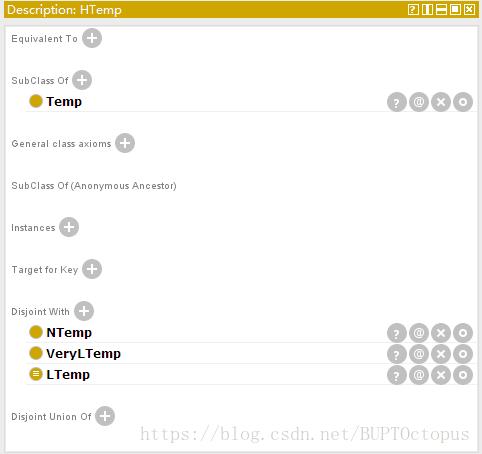
菜单点击file, new.每个本体有一个总的类owl:Thing。



我们在这个类的下面声明一个子类(subclass)，比如输入设备iDevices：   
点击图2-1（Add subclass）,Name就叫iDevices。   
  
再声明输出设备oDevices类，可以点回到owl:Thing然后声明子类，也可以直接声明sibling class   
图2-3

同理我们再声明一些类，就能实现一个类的架构出来。

类的属性有：   
  
点击后面的加号就可以声明相应的属性，其中Equivalent To属性声明的是当满足这一条件时这个类等同于另一个指定的类；Subclass of,Disjoint with等顾名思义。

声明个体（Individual）

点击声明个体，声明个体之后在这个个体的Description里可以声明它属于哪个类，在 Property Assertion里声明它具有的对象属性数据属性等。

2.3 对象属性

在本体中如何表示“我是你师兄”这样一种关系呢？那就要用到对象属性。比如“人”这个类下的个体“我”具有“isFatherOf”，啊不是，具有“isBrotherOf”的对象属性，对象是“你”：

在 Object Property 里创建isBrotherOf这一对象属性，在 对象“我”的property assertion里声明

这就行了。

2.4 数据属性

跟对象属性差不多，顾名思义数据属性，比如先声明Temperature这一数据属性，在代表温度传感器的个体XTemp里声明它具有Temperature这一数据属性，值是30 （℃）：

通过制定规则来完成推理，SWRLTab的使用

由于用OWL语言直接写规则很麻烦，可读性也较差（看个例子？：），因此可以用SWRL这个很棒的插件。

例：温度高过阈值了开空调并保持其他制热设备关闭 的规则OWL语言实现：

SWRL的官网：http://www.w3.org/Submission/SWRL/

当然，如果你直接按照官网给的例子写规则是会报错的，大概是他们太久没更新了语法不太对……至于SWRL的语法，看上面那个例子也就很清楚了吧。简单解释一下就是：

iDevices里的某个个体(?p)，它具有对于另一个体(?y)的对象属性(hasTemp)并且( ^ )它也是高温(Htemp)类的个体，那么(->)就让个体Control具有对于Air的对象属性TurnOn以及 吧啦吧啦吧啦后面一串规则。

当你搞了一坨规则以后：2.6 本体内的参数决策

利用Class 的Equivalent To属性，比如温度Temp类下有高温HTemp低温LTemp和常温NTemp等等不想disjoint的子类，可以通过让Temp类的个体，若其Temperature数据属性大于某个值就等同于高温类HTemp，就可以完成本体内的决策。

以LTemp为例

最后，

保存你创建的本体，会生成相应的.owl文件

用Protege构建本体就到这吧。

OWL API

基于OWL API可以做些简单的开发，对本科生来说目的肯定是入门一下Java和HTML。主要包括：

读取本体，创建本体Model

修改本体中某些数值、属性等

调用推理机，根据本体中的规则库进行推理

1.读取本体

1.1 读本体文件

1.2 创建本体Model

OWLOntologyManager OOM = OWLManager.createOWLOntologyManager();

OWLOntology myOntology = OOM.loadOntologyFromOntologyDocument(InFilePosition);

OWLDataFactory DF = OOM.getOWLDataFactory();

1.3 例：把前端输入的温度写进本体

（为啥要前端输入？还不是因为没有真的传感器）

OWLDataProperty Temperature = DF.getOWLDataProperty (IRI.create(base+"#Temperature"));

OWLNamedIndividual IndiXT = DF.getOWLNamedIndividual (IRI.create(base + "#XTemp"));

OWLDataPropertyAssertionAxiom ODPAA = DF.getOWLDataPropertyAssertionAxiom(Temperature, IndiXT,InsideTemp);

IRI documentIRI = OOM.getOntologyDocumentIRI(myOntology);

AddAxiom addAxiom = new AddAxiom(myOntology, ODPAA);

OOM.applyChange(addAxiom);

1.4 例：从本体读温度传感器数值

Map<OWLDataPropertyExpression, Set<OWLLiteral>> ODPV=IndiXT.getDataPropertyValues(myOntology);

String StringODPV = ODPV.toString();

StringODPV=StringODPV.trim();//去空格

//输出是标签形式的一长串，往前端显示数字的话可以百度个代码吧string里的数字拿出来就是了。

1.5 调用HermiT或Jena推理机

//ReasonerFactory RF = new ReasonerFactory(); //调用jena默认推理机

OWLReasonerFactory RF = new Reasoner.ReasonerFactory();//调用HermiT 推理机

OWLReasoner reasoner = RF.createNonBufferingReasoner(myOntology);

System.out.println(reasoner.getReasonerName()+"版本信息："+reasoner.getReasonerVersion());

reasoner.precomputeInferences();//Infer All

ArrayList<InferredAxiomGenerator<? extends OWLAxiom>> IAG = new ArrayList<InferredAxiomGenerator<? extends OWLAxiom>>();

6

下面的输出选项可供选择你想要输出的推理结果：

IAG.add(new InferredSubClassAxiomGenerator());

IAG.add(new InferredClassAssertionAxiomGenerator());

IAG.add(new InferredDataPropertyCharacteristicAxiomGenerator());

IAG.add(new InferredDisjointClassesAxiomGenerator());

IAG.add(new InferredEquivalentClassAxiomGenerator());

IAG.add(new InferredEquivalentDataPropertiesAxiomGenerator());

IAG.add(new InferredEquivalentObjectPropertyAxiomGenerator());

IAG.add(new InferredInverseObjectPropertiesAxiomGenerator());

IAG.add(new InferredSubDataPropertyAxiomGenerator());

IAG.add(new InferredObjectPropertyCharacteristicAxiomGenerator());

IAG.add(new InferredPropertyAssertionGenerator());

IAG.add(new InferredSubObjectPropertyAxiomGenerator());

1.6 读取个体的对象属性（可用于读取推理结果呀）

值得强调的是，RACER、FaCT、Pellet采用描述逻辑作为理论基础，算法采用Tableau算法。这些系统都做了大量的优化工作。

，针对本体的推理，越来越多地集中在了几种标准的本体语言上，如OWL、DAML、RDFS/RDF等，这样，研究者给出效率更高的，针对这些具体应用的推理机。象RACER、FaCT、Pellet等就是这类具体有强烈针对性的推理机。它们的优点是：效率很高，使用上也非常方便；缺点是：不是通用推理机，将推理能力限定在几种具体的本体语言上，而且用户很难对它进行扩展。

值得强调的是，RACER、FaCT、Pellet采用描述逻辑作为理论基础，算法采用Tableau算法。这些系统都做了大量的优化工作。Jena是面向语义Web的应用开发包，包含的内容比较全面，推理机只是其中一部分。Jena提供的推理机也和RACER、FaCT、Pellet等一样，是针对本体的推理机，但Jena本身并不是‘推理机设计专家’，它自身包含的推理机基本上就是一种CLIPS配合本体领域产生式规则的前向推理系统。因此，它的运行效率不是很高。好在现在有DIG接口，DIG有点象数据库中的ODBC，允许前端挂接到后台不同的推理引擎上。这样，在Jean中，

也可以使用RACER、FaCT、Pellet这样更‘专业’些的推理机。

Pellet是一种基于Tableau算法的描述逻辑推理机，由美国马里兰大学（College

Park分校）的MindSwap实验室开发。Pellet是基于Java的开放源码系统。

综上所述，可做如下区分：

1. Jena中的推理引擎和RACER都是针对具体本体语言的推理机，针对性强，效率高。

2. Jess是层面更低的，通用的CLIPS推理机。

这就可以解释：为什么当用户使用owl表示本体时，Protege会用Jena或RACER等推理，而当用户使用class/slot表示本体时，Protege仅能使用Jess。其实，从能力上，Jess也可以完成owl推理(需要输入相应的产生式规则库)，只是效率低。

————————————————

版权声明：本文为CSDN博主「蚂蚁1991」的原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/heliuwei1991/article/details/50161881