清 华 大 学

综 合 论 文 训 练

题目：一种基于Renew的有色网元对象协议的设计与实现

系　　别：计算机科学与技术系

专　　业：计算机科学与技术

姓　　名：齐大伟

指导教师：王生原

2008 年 6 月 15 日

关于学位论文使用授权的说明

本人完全了解清华大学有关保留、使用学位论文的规定，即：学校有权保留学位论文的复印件，允许该论文被查阅和借阅；学校可以公布该论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存该论文。

(涉密的学位论文在解密后应遵守此规定)

签　名：_____ 导师签名：_____ 日　　期：_____

中文摘要

随着软件系统变得越来越复杂，对系统的建模的可重构性要求越来越高。Petri网作为一种在模拟并行系统中广泛应用的工具，对Petri网的模拟的可重构性要求也越来越迫切。在本文中针对Petri网模拟工具，设计了相应的元对象协议从而使得Petri网工具具有可重构行。同时在Petri网模拟工具中增加元对象协议可以达到简化基级网模型，获取基级网运行信息和动态改变基级网结构等目的。以Renew工具作为基础，实现了元对象协议，并以Hurried Philosophers 作为例子来验证了元对象协议的功能。

关键词：Petri网　元对象协议　Renew

ABSTRACT

As the software system are becoming more and more complicated, the requirement of reconfigurability are more and more serious. Petri Net is a popular simulation tool in parallel system simulating,so Petri Net simulation tools are also required to be reconfigurable. In this article,a MetaObject Protocol for the Renew Petri Net simulation tool is designed in order to add reconfigurability to Petri Net simulation tools. This work could also simplify the base level net model, gather information during the net simulation and change the base level net dynamically. Then the MetaObject Protocol has been immplemented based on the Renew tool. The Hurried Philosophers problem is taken as an example for the usage of the MetaObject Protocol.

Key words: Petri Net MetaObject Protocol Renew

目 录

主要符号对照表

Place	Petri网中的库所
Transition	Petri网中的变迁
Arc	Petri网中的转移弧
Token	Petri网中的托肯
CPN	有色Petri网
Renew	德国汉堡大学编写的模拟有色网的软件
rnw	Renew中的网的文件的后缀名

第 1 章　引言

现在软件的发展非常迅速，软件在其生命周期中经常需要不断的更改。为了减少不断更改的开销，现在通常采用的办法是在开始设计的时候尽量的考虑到未来可能的需要，并将这些未来可能的需要集成到当前的系统中来。这种做法会使得当前的系统变得更加的复杂，使得当前的系统需要考虑一些与当前的系统不相关的因素。另外，并不是所有的未来的需求都是可以在软件设计的时候被预见到的。Petri网是对离散并行系统的数学表示，被广泛用于离散并行系统的建模当中。当Petri网被用于对这些不断改变的系统进行建模的时候，同样会遇到这些问题。系统的行为和系统的改变应该是正交的两个方面[1]，即它们可以分开进行考虑。为了在利用Petri网建模的系统中实现这种分开考虑的做法，我们在基于Renew 工具的基础上设计并实现了一种元对象协议。

针对Petri网建模工具设计并实现元对象协议的相关研究并不多见，在[2,3]中提出了一种Petri网模拟工具中增加元对象协议的方案，在他们的方案中基级网采用的是基本网。在他们的论文中提到要以GreatSPN为基础来实现该设计，但是至今未见相关工作的发表。

Renew[4,5]是德国汉堡大学开发的有色网的模拟工具，它采用Java语言来进行编写，结构清晰，功能强大。对于我们这个项目其最重要的一点是，它完全的公开其源代码。

在Renew中增加元对象协议，首先要保证原有的系统不受影响，并且增加的内容能够和原有的系统融为一体。因此在对Renew的结构进行了分析之后定义了对Renew中运行状态的一致性的要求，之后的改动都是按照此规则来进行。具体的实现中采用Renew的原来的运行时的网结构作为元对象协议中的基级，在此基础上增加元级。设计并实现了的元对象接口主要包括对基级的Token的增加和删除，对基级Place，Transition，Arc的增加和删除。基级和元级之间的因果联系的建立是在实现过程中的一个重点。

增加元对象协议，对Renew工具的建模能力会有所增强。

对于普适的情况

- 可以简化基级别的网模型
- 可以在不改变基级的情况下获取基级信息

- 可以通过对基级的控制来改变基级网结构

  对于处理器描述
- 简化分支预测恢复的建模
- 获取处理器运行时的一些信息

在我们实验室中，有色网被用于处理器的描述，在对于复杂处理器的分支指令预测等特征的描述中，利用普通的有色网工具会遇到一些困难[6],利用在工具中增加元对象协议的方式可以解决这些问题。

为了验证Renew中元对象协议的功能，采用了Hurried Philosophers作为例子，通过Hurried Philosophers的例子验证了实现的正确性和整体上的功能。

论文的余下部分组织如下:第二部分介绍了本工作中涉及到的背景知识，包括Renew、有色网、元对象协议和反射机制。第三部分主要介绍了整体的元对象协议的设计与实现。第四部分具体的介绍了如何使用所设计的元对象协议。第五部分是Hurried Philosophers的例子，用来验证元对象协议的实现。最后第六部分对整体的工作进行了总结，并对未来的工作进行了展望。

第 2 章　背景知识

2.1　Renew

Renew是德国汉堡大学编写的有色网的模拟工具，作为有色网的工具，它采用Java语言来编写，因此可以在大多数的操作系统上使用。采用面向对象的方式，在实现中网和网元素都是Java中的对象。在Renew中可以使用任何的外部的Java类，使得扩展变得很容易。Renew还采用完全开放其源代码的方式，这使得用户可以对整个系统更加的了解，使得用户可以根据自己的需求来自行修改整个系统。在具体的参照的网模型方面，Renew也有很多的优点：在Renew中支持同步通道的机制，支持众多的转移弧的类型，支持面向对象的机制等等。下面对Renew的结构和具体的执行过程做一下简要的分析。

2.1.1　Renew结构的分析

Renew整个软件的结构从层次上来讲共分为四层：graphic，shadow，template，instance层（图2.1）graphic和图形界面相关，shadow层为了脱离图形界面而设置的，为了支持不同类型的网，提供了shadow到template的转换，根据shadow的不同类型采用不同的compiler进行编译，但是编译之后的template层是统一的。在运行的时候会由template层生成instance层的对象。

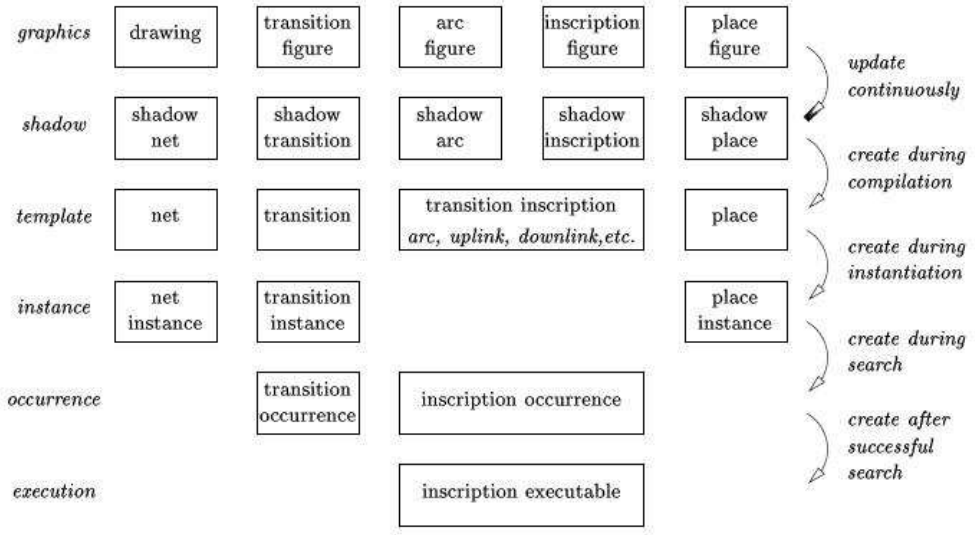

图 2.1　Renew的结构分析

2.1.2　Renew的运行过程

一个Petri网的运行过程就是寻找可以发射的Transition，然后进行发射的过程。在Transition发生的过程中会对整个网的Marking产生改变，在改变之后重复以上查找发射的过程，直到找不到可以发射的Transition为止。Renew的内部实现当然也要遵守上面的规则。在Renew中增加了时序的机制，也就是某些token在某时刻之前是不可见的，这样做是为了提高效率。对于Transition也增加了时间的标签。通过这些标签来标明一个Transition在某时刻之前是不可能enable的，因此也就没必要进行搜索，从而减小了搜索的空间，提高了运行的效率。在Renew中采用SearchQueue来保存所有的TransitionInstance，在SearchQueue中会保存一个全局的时钟，在某一个时刻可以从SearchQueue中获得所有当时可能enable的TransitionInstance的集合。在获得这个集合之后，从中选取一个可能搜索消耗最小的TransitionInstance来进行搜索，由于是有色网，搜索采用合一算法，进行递归的搜索。如果搜索成功，即找到一个有效的合一，那么便执行这个TransitionInstance。执行的时候由于一个Transition会有很多相关的动作，这些动作是存储在Transition中的，被称为TransitionInscription Set，其中就包含有Arc,（简单的Arc的动作包括从某个place移除Token，或者将Token加到某个Place里面）。这些存储的并不是真正执行的动作，而是类似于一个没有具体化的动作，比如一个input Arc要移除一个Token，只有在成功的找到Token的绑

定之后才会知道要移除的是哪个具体的Token。所以在搜索成功之后，再由这些TransitionInscription来生成具体的Executables来具体的改变网的Marking并且来产生高级网的一些其它的改变。

2.2 有色网

Petri网是由德国科学家Carl Adam Petri 于1962年在其博士论文中提出来的。Petri网是一种离散分布系统的数学表示，作为一种建模语言，它能够图形化的表示离散分布系统。Petri网最重要的一个特征就是建立了真正的并发语意。这也是它在描述分布式或者并发系统方面得到广泛应用的最主要的原因。Petri网是建立在一些很简单但功能强大的数学概念之上的，这使得它具有很好的理论基础。根据这些理论基础，现在已经有一些很有效的模拟、分析、验证的方法。用Petri网建立的模型的可验证性是Petri网得到广泛应用的另一个主要的原因。由于最原始的Petri Net语言是十分简单的，直接用Petri Net进行建模，往往使得模型太过复杂。因此，在Petri Net被提出后的四十多年间人们对它引入了各种各样不同的特性来加强它的建模的方便性，出现了各种各样的变体。如在Petri网中引入对象的概念形成了对象Petri网、在Petri网中引入Color的概念形成了有色网、引入随机数学中的概念形成了随机Petri网等。当然，C.Petri的Petri网奠定了所有Petri网变体的语意基础，在理论上原始的Petri网与这些Petri网变体具有相同的建模能力。我们的工作是使用有色网，这也是应用最广泛的一种变体。

在基本的Petri网中，Place中的token是不可以区别的，每个Place中只能存在一个token。在有色网中对token增加了颜色的概念，这样一个Place中可以存在多个不同颜色的Token，这样原来的基本的网可以通过折叠的方式变成有色网。通过引入有色网的概念可以简化系统的模型，是整个系统的模型变得更加清晰明了。

从形式化来说, 一个有色网就是一个元组CPN = (S, P, T, A, N, C, G, E, I) 满足如下约束:

- S 是一非空的token类型的有限集，也称作颜色集合。
- P 是库所的有限集。
- T 是变迁的有限集。
- A 是弧的有限集，集合A有如下约束:

  $P \cap T = P \cap A = T \cap A = \emptyset$

- N 是一个结点函数，它定义了从A到$P \times T \cup T \times P$的映射。
- C 是一个颜色函数，它定义了从P到S的映射。
- G 是一个哨卫函数，它定义了从T到表达式的映射。映射满足如下约束：
  $\forall t \in T : [Type(G(t)) = \mathbf{B} \wedge \mathbf{Type(Var(G(t))) \subseteq \Sigma}]$
- E 是一个弧表达式函数，它定义了从A到表达式的映射。映射满足如下约束：
  $\forall a \in A : [Type(E(a)) = C(p)_{MS} \wedge Type(Var(E(a))) \subseteq \Sigma]$ where p is the place of N(a)
- I 是一个初始化函数。它定义了从P到闭表达式的映射。映射满足如下约束：
  $\forall p \in P : [Type(I(p)) = C(p)_{MS}]$

有色网和基本的Petri网一样也是由库所，变迁，和转移弧来组成的。库所中可以存在标有颜色的token，在建模中token可以代表现实生活中的资源。库所中token的数目则代表了相应的资源的数目。一个库所中所包含的token的状态称为marking，在一个时刻所有库所的marking代表了整个系统的状态。在有色网中，变迁上设有卫哨，当有色网执行的时候，变迁触发前首先要对token进行绑定，当绑定的token符合卫哨的条件的时候，变迁才可以触发。

2.3 计算反射性及元对象模型

2.3.1 计算反射性中的基本概念

随着软件的功能日趋复杂，应用的需求越来越多，观察自身行为并且改变自身行为的能力逐渐成为一种需求。下面首先介绍一些计算反射性中的基本的一些概念[7]：

定义 2.1： **反射** (reflection)

反射是指计算系统能够观察自身行为，并且在某些特定条件下改变自身行为的能力。

定义 2.2： **基级系统** (base system)

基级系统是指为了解决实际问题而开发的系统，是相对于元级系统而言的。

定义 2.3： **元级系统** (meta system)

元级系统是用来观察基级，改变基级系统的系统。通常元级系统对于基级系统是不可见的。

定义 2.4： **具体化** (refication)

具体化是指将基级的信息在元级中表示出来，便于元级对基级进行观察与操作。

定义 2.5： **因果联系** (causal link)

因果联系是指基级和元级之间的联系，基级可以因为运算而符合元级的某些条件而引发元级的计算，元级也可以主动的改变基级的行为。因果联系对于程序员应该是不可见的。

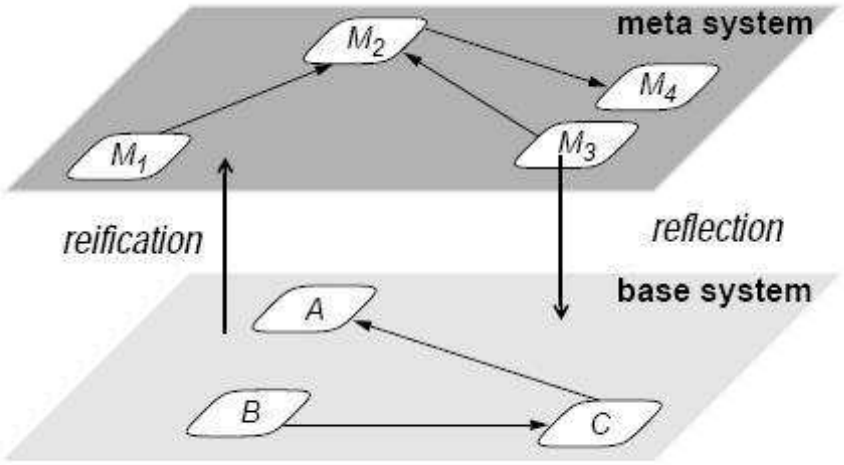

图 2.2　计算反射系统的整体结构

图2.2显示了上面这些概念之间的相互联系，以及整个计算性反射系统的框架结构。

2.3.2　计算反射性的分类

计算反射性可以根据基级和元级的关系的建立时间来分为编译时反射、加载时反射和运行时反射。根据具体的反射的信息的不同可以划分为结构反射和行为反射。在反射系统中，基级和元级直接的因果联系是其中的关键，通过对反射进行具体的分类可以便于对反射系统有更深入的了解。

2.3.2.1 编译时、加载时、运行时反射

根据反射系统中基级和元级关系的建立时间，我们可以将反射分为编译时反射、加载时反射和运行时反射。不同阶段的因果联系的建立会对整个系统的效率和灵活性产生影响。编译时的反射效率最高，同时灵活性也最差。运行时的反射效率最低，但是拥有最好的灵活性。加载时的反射则介于两者中间。

编译时反射

编译时反射指的是在编译基级代码的时候是由元级代码来控制的。编译时反射的一个最大的好处是它拥有很高的效率，一旦编译之后执行的时候便不会元级代码的影响。同时由于编译是的反射只能是通过改变源代码来完成，因此灵活性比较差。图2.3反映的是编译时的反射。实现编译时反射的典型系统有OPEN C++。

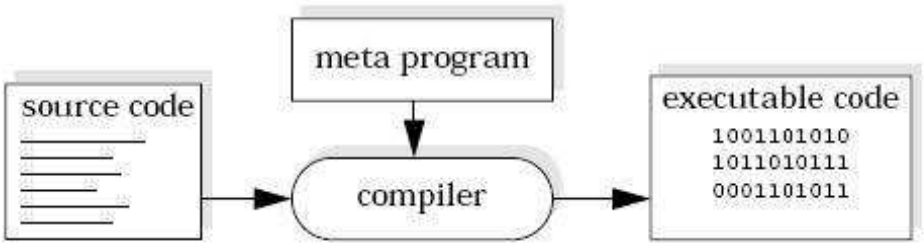

图 2.3　编译时反射

加载时反射

加载时反射指的是由元级控制基级程序的装入或者加载的过程。加载时反射的典型系统有OMOS。图2.4反映的是加载时反射的情况。

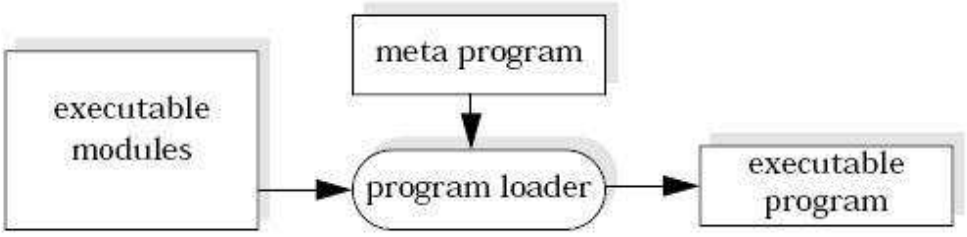

图 2.4　加载时反射

运行时反射

运行时反射的情况下元级程序可以控制基级程序的执行，因此可以获得很高的灵活性。图2.5反映的是运行时的反射的情况。运行时反射的典型系统有MetaJava。

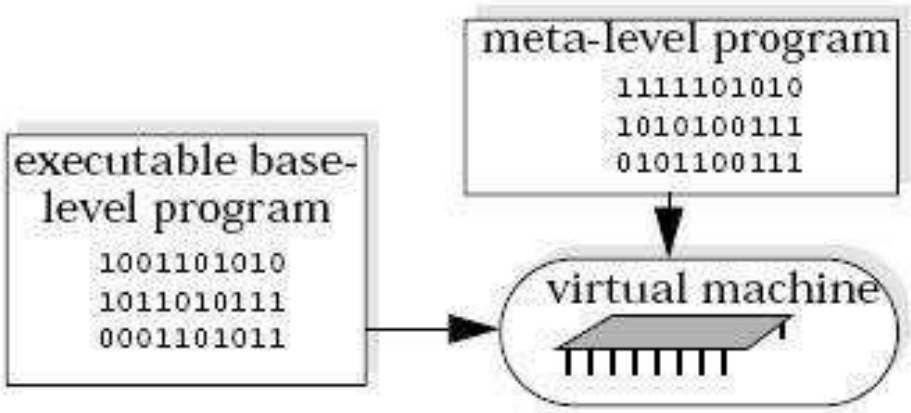

图 2.5　运行时反射

2.3.2.2　结构反射与行为反射

从在具体化的内容方面来划分，反射可以被划分为结构反射和行为反射。结构反射具体化程序的结构方面，比如继承、类的结构和数据类型等。行为反射具体化程序的运算和具体的行为。一个方法调用是典型的行为反射所关心的方面。

2.3.3　元对象协议在面向对象语言中的实现

从前几小节的基础知识中可以知道元对象协议的关键在于基级系统在元级的表示以及基级与元级之间因果联系的建立。下面就这两个方面来说明在面向对象的语言中如何来实现元对象协议。

在面向对象的语言中，基级系统中的内容都是用对象来表示的，在元级系统中同样采用面向对象的机制，可以使得基级与元级的对象相互对应。反射是通过元级的计算来完成的，而元级的计算由基级计算来触发。在基级系统中的运算符合预先定制好的条件的时候便会陷入元级，引发元级的计算，对基级进行观察和改变。

发射根据因果联系的建立时间的不同被区分为编译时发射，加载时反射和运行时反射。决定采用哪种反射的一个主要的因素是元级所需要获得饿信息和

所需要改变的基级的内容。在面向对象的语言中一般可以用事件机制来实现因果联系的建立。

第 3 章　Renew中元对象协议的设计与实现

3.1　整体设计

以Petri网在Renew中原来的表示作为基级，这个表示既包括静态的表示（网的结构）也包括动态的表示（某一时刻的Marking）。新增加元级来加强Renew的功能，在元级可以实现对网的结构和Marking的动态的改变，并且可以获得基级的运行是的信息（图 3.1）。元级的具体的操作是由用户通过元级的程序来自己定义的。用户使用Renew中的元对象协议的一个整体的过程可以是：a.在Renew中进行基级网的编辑，b.对应基级网编辑相应的元级程序，c.模拟运行。如果用户并不需要元级的功能，那么只是简单的省略掉步骤b，便可以按照Renew中原来的执行了，整个Renew不会受任何影响。

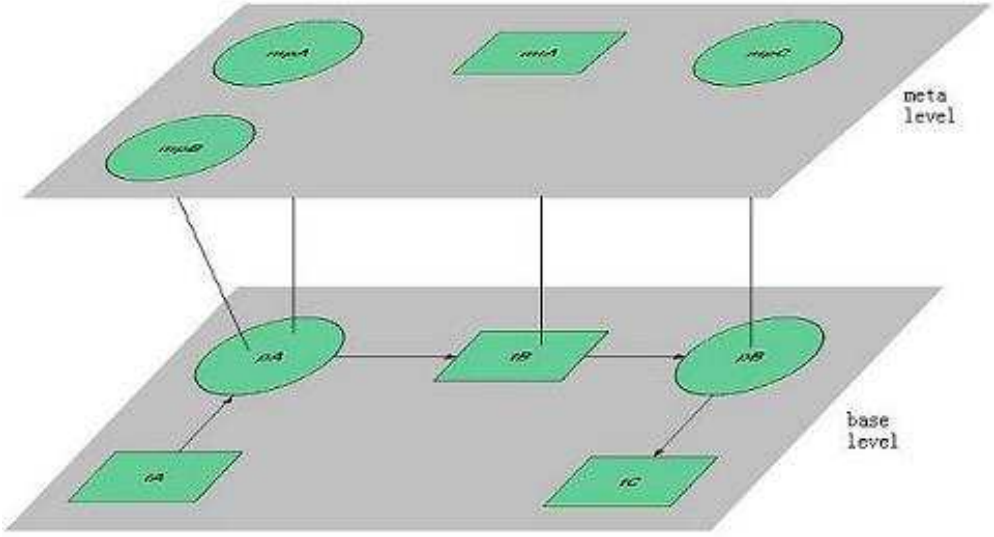

图 3.1　Renew中元对象协议整体结构

3.2　基级对象在元级中的表示

对于一个运行时的Petri网，如果需要完全的描述它的状态，需要采用Place，Transition，Arc的集合并且用Marking来对每个Place中的Token进行表

示。这只是基本网的要求，如果是高级网，就需要对Transition和Arc上的表达式进行表示。联系到Renew的系统，采用NetInstance，Net，SerachQueue来表示基级对象可以对基级对象进行完全的表示。Net中包含的网中对象的集合，在这个集合中包含Transition和Place，Arc包含在Transition中。每个Place的Token储存在PlaceInstance里面。Arc是储存在Transition中的。对于基级的修改，最关键的就是一致性的问题，也就是修改后的基级的表示仍然可以被renew当作一个正常的状态来继续进行执行，而renew系统的其它部分对整个修改不会有所察觉。如果一个状态是符合如下条件的我们认为这个状态是一致的：

1) NetInstance和Net中的对象相互一致：如果在NetInstance中存在一个PlaceInstance（TransitionInstance）那么在Net中应该存在相应的Place（Transition），反之亦然。
1) 在SearchQueue中存在的TransitionInstance必然在NetInstance中存在
1) 在任意一个Transition中存在的Arc对应的两端的Transition和Place必然在Net中存在。

对于所有的元级操作成功的一个必要条件是保持基级的一致性。

3.3 元对象协议的结构和用户接口

提供两种元对象，MetaPlace 和MetaTransition 它们都是MetaObject的子类；基级对象相应的为PlaceInstance和TransitionInstance。

现在提供的都是一对多的绑定的方式，即一个元对象可以绑定一个基级对象，但是一个基级对象可以被多个元对象所绑定（图3）。对于元对象提供的接口包括：

3.3.1 用于元级和基级关系的建立的接口

```
public void attach(String placeName);
public void detach(String placeName);
String getName();
```

用户可以通过attach来将元对象绑定到相应的基级对象上，通过detach来解除这种绑定。用户可以通过getName()来获得所绑定的基级对象的名字，这个名字和在图形界面上进行的命名的名字是相同的。

3.3.2 用户对基级改变的接口

表 3.1 对基级进行改变的接口

改变基级行为的	改变基级结构的
Mp.insertToken(Object token)	addPs(String ... names)
Mp.removeToken(TokenRemoveCondition condition)	public static void addTs(String ... names)
Mp.addToken(Object [] tokens)	addArc(String place, String transition, int arcType) removeArc(String place, String transition, int arcType transitionInstance,int arcType) void remPFs(String ... names) void remTFs(String ... names) void remPs(String ... names) void remTs(String ... names)

对于基级行为的改变主要是对Marking的改变，其中加入Token相对比较容易，当然这个Token要由用户进行定义，如果需要用到Renew中的有色的概念，就需要了解Renew中的Token的类型的概念。同时可以加入多个Token，采用数组的形式。

对于Token的删除，相对比较困难，因为用户并不知道在某个Place中有哪些Token，然后还要制定删除哪些token，如果要用户把place中的所有token都自行的检查一遍，就会显得不简洁。这里我们提供的删除的机制是由用户提供一个条件类TokenRemoveCondition，由用户自行定义这个类，来确定删除哪些Token。这个类只有一个函数就是

boolean delete(Object token); 用户可以继承这个类自行定义判断是否删除的条件。

3.3.3 对基级进行观察的接口

MP.Marking();

这个函数提供对Marking的观察，这个函数会返回一个MetaPlace对应的PlaceInstance中所有的token，返回的token是以数组的形式进行组织的。

Object base();

base()函数会返回所绑定的基级的对象。

3.3.4 其它的一些辅助函数

getMetaPlace(String name);

getMetaTransiton(String name);

上面这两个函数用于方便用户对元对象的获得，并且可以节省资源的使用，也就是说在这两个函数中帮助用户完成了元对象的定义，绑定的过程，如果之前有元对象绑定到相应的基级的对象上的时候，此时直接返回该元对象，而不是重新定义一个新的元对象。

void register(String event,String function,String name, int type);

void unRegister(String event,String name, int type);

这两个函数用来实现用户自定义的事件和Renew中事件的联系。register 函数用来建立联系在元对象协议的实现中，我们允许用户在元级通过程序的方式来定义某些事件发生后的行为，我们可以通过函数的方式将这个行为在元级程序中表示出来。这类函数的框架是这样的：

```
public static void yourActionName(MetaObject metaObject){
    ...
}
```

3.3.5 元级接口汇总

表 3.2 元级接口汇总

接口	作用
public static void addFIFOPlaceInstance(String name)	向基级网加入FIFO 类型的Place
public static void addMultisetPlaceInstance(String name)	向基级网加入Multiset 类型的Place
public static void addPlaceInstance(String name)	向基级网加入Place
public static void addPs(String ... names)	向基级网加入多个Place
public static void addTransitionInstance(String name)	向基级网加入Transition

表 3.2 (续元级接口汇总)

public static void addTs(String ... names)	向基级网加入多个Transition
public static void addArc(String placeInstance, String transitionInstance, int arcType)	向基级网加入Arc，参数分别为对应的Place，Transition的名称和Arc的类型①
public static void addArc (PlaceInstance placeInstance, TransitionInstance transitionInstance,int arcType)	向基级网加入Arc，参数分别为对应的PlaceInstance、TransitionInstance和Arc的类型
public static TransitionInscription getArc(Transition transition,Place place ,int arcType)	获取基级网的Arc，注意返回的类型可以强制转为Arc
public static void removeArc(PlaceInstance placeInstance,TransitionInstance transitionInstance,int arcType)	删除基级网中的Arc，参数分别为对应的PlaceInstance、TransitionInstance和Arc的类型
public static void removeArc(String placeInstance,String transitionInstance,int arcType)	删除基级网中的Arc，参数分别为对应的Place，Transition的名称和Arc的类型
public static void removePlace(PlaceInstance placeInstance)	删除基级网中的Place，参数为对应的PlaceInstance
public static void removeTransition(TransitionInstance transitionInstance)	删除基级网中的Transition，参数为对应的TransitionInstance
public static boolean checkPlaceArcs(PlaceInstance placeInstance)	检查一个Place 是否有相关联的Arc，如果有返回fasle，没有返回true
public static boolean checkTransitionArcs(TransitionInstance transitionInstance)	检查一个Transiton是否有相关联的Arc，如果有返回false，没有返回true
public static void register(String event,String function,String name, int type)	将基级类型为type的名称为name的基级对象的事件evenet与元级程序的function关联起来②
public static void unRegister(String event,String name, int type)	解除基级类型为type的名称为name的基级对象的事件evenet与元级的函数关联

① Arc的类型有Arc.in ,Arc.out,Arc.both等，更多的Arc类型请参考源代码中Arc.java，in和out是相对于Place的

② type 有TRANSITON和PLACE两种类型

表 3.2 (续元级接口汇总)

public static MetaObject getMetaObject(String name,int type)	获取类型为type的名称为name的元级对象
public static MetaObject getMO(String name,int type)	获取类型为type的名称为name的元级对象
public static Set getMTs(String ... name)	获取元级Transition
public static MetaPlace getMetaPlace(String name)	获取元级Place
public static MetaTransition getMetaTransition(String name)	获取元级的Transition
public static void removeTransitionF(TransitionInstance transitionInstance)	删除基级网中的Transition，同时删除相关联的Arc
public static void remTFs(MetaTransition ... mts)	删除基级网中的Transition，同时删除相关联的Arc
public static void remTFs(String ... names)	删除基级网中的Transition，同时删除相关联的Arc
public static void removeTransitionF(String name)	删除基级网中的Transition，同时删除相关联的Arc
public static void removePlaceF(String name)	删除基级网中的Place，同时删除相关联的Arc
public static void removePlaceF(PlaceInstance placeInstance)	删除基级网中的Place，同时删除相关联的Arc
public static void remPFs(MetaPlace ... mps)	删除基级网中的Place，同时删除相关联的Arc
public static void remPFs(String ... names)	删除基级网中的Place，同时删除相关联的Arc
public static boolean exist(String name)	判断是否已经存在元级对象名为name
public static double getTime()	获取SearchQueue中的时间
(MetaPlace MetaTransition)public Object base()	用来获取对应的基级对象
(MetaPlace) public Object[] Marking()	用来获取对应基级对象的marking
(MetaPlace) public void addTokens(Object[] tokens)	添加多个token
(MetaPlace) public void insertToken(Object token)	添加一个token

表 3.2 (续元级接口汇总)

(MetaPlace) public void removeToken(TokenRemoveCondition condition)	根据参数condition来删除符合条件的token

在这个函数中会传入发生时间的元级对象作为一个参数，通过这个元级对象用户可以很容易的获得相应的基级对象，这样在用户自定义的程序中便可以灵活的使用这个参数，进行丰富的扩展。

3.4 对基级的观察

这个方面主要是利用在attach的时候存储了基级的对象来实现的，对于base()即直接返回存储的对象，对于Marking来说，也是利用存储的PlaceInstance来获得相应的Marking来实现的。

3.5 对基级的改变

对基级的petri网的改变包括两个方面：行为方面的改变和结构方面的改变。行为方面的改变目前主要包括对网的Marking的改变，包括增加和删除一些Token。对于网的Marking的改变相对比较容易，因为在元级的每个MetaPlace中都存储着相应的基级的PlaceInstance，所以只是需要调用PlaceInstance的相应的函数就可以实现了。在这里注意的一点就是在进行改动的时候要获得相应对象的互斥锁，这样才可以保证改动的一致性。至于删除token的时候的困难及具体的解决办法在上面的元对象协议的结构和用户接口一小节有详细的描述。

结构方面的改变就是在运行时改变基级网的结构，包括对Place，Transition，Arc的增删。在前面提到过对于网结构的改变最重要的一点是一致性的保持。

增加网元素：在增加网元素的时候，增加单独的Place，或者是单独的Transition都没有很复杂的情况，只是把在基级把PlaceInstance，TransitionInstance和Place，Transition加入到相应的结构中就可以了。这里对于Transition的加入需要注意的一点是，在加入Transition的时候要把相应的Transition加入到SearchQueue中，否则这个Transition将永远都不会得到执行。对于增加网元素来讲，比较复杂的是增加Arc，因为增加Arc需要依赖相应的Transition和Place，如果相应的Transition或者Place不存在的话，必然会导致错

误。另外Arc有很多类型，这个需要用户在写元级程序的时候进行指定。删除网元素：在删除网元素的时候，和增加的过程恰恰相反，删除Arc会相对比较容易一些，但是删除Place或者Transition的时候就要注意到其相关联的Arc了。在具体的实现中我们提供两种操作（以删除Place为例）

void remPFs(String ... names)

void remPs(String ... names)

一种是强制删除的版本，另外一种需要用户自行解决相关的Arc的问题。

对于强制删除，就是不管有没有关联的Arc，只要确定删除就会把Place或Transition给删除掉，然后如果有想关联的Arc的话，相关联的Arc也会被删除掉。这种删除是和在图形界面上编辑基级网时候的删除是一致的。对于非强制性的删除，如果有相关联的Arc的话，删除就会失败。

在实现过程中，非强制删除相对容易，对于强制性删除，在上面提到过，Arc是作为TransitionInscription保存在Transition中的，因此对于Transition来讲，强制删除只是将Transition和TransitionInstance给删除掉就可以了，这样相应的Arc也就没有了。但是对于Place来讲，一个Place所相关联的Arc可能会存储在不同的Transition中，而且在Renew原来的实现中，无法知道这些Arc存储在哪些Transition中除非你对所有的Transition进行完全的扫描。在我们的具体实现中，在Place增加了相关联的Arc的集合，这样便于在删除Place时对是否有相关联的Arc的判断。

3.6 基级和元级之间的因果联系

元级和基级的因果联系包括两个方面，一方面是从元级到基级的联系，另一方面是从基级到元级的联系。元级到基级的联系：在每个元对象中，存储这相应的基级对象的引用。这个基级对象是在attach操作的时候获得的。通过获得相应的基级对象，我们可以在之后的操作中使用这个基级对象来直接进行。那么在元对象这一级实际上对基级对象进行了封装，对用户看起来像是在对元对象进行的操作，在实际的实现中，元对象一级只是进行了一个转换的过程。

比如insertToken的操作

```
public void insertToken(Object token){
    _placeInstance.lock.lock();
```

```
        try{
        _placeInstance.insertToken(token);
        }catch(Exception e){
        System.out.println("MetaPlace.insertToken "+e.toString());
        }finally{
            _placeInstance.lock.unlock();
        }
    }
```

基级到元级的联系：基级到元级的联系同样是在attach的时候建立起来的。这里利用了Renew中原有的EventListener的机制来实现的。在Renew中Transition，Place，TransitionInstance，PlaceInstance都有想对应的EventListenerSet，当相应的事件发生的时候，在EventListenerSet中的元素相应的函数就会被调用。这个是TransitionInstance中的代码

```
    synchronized void firingStarted(FiringEvent fe) {
        pendingEvents.add(fe);
        listeners.firingStarted(fe);
        transition.getListenerSet().firingStarted(fe);
    }
```

这是变量listener对应的TransitionEventListenerSet类的firingStarted的代码

```
public synchronized void firingStarted(final FiringEvent fe) {
    dispatch(new ListenerSetDispatcher() {
        public void dispatchTo(Object listener) {
            ((TransitionEventListener) listener).firingStarted(fe);
        }
            });
    }
```

利用这个机制，在元级每一个元对象都实现了基本的EventListener的接口，然后在attach的时候，相应的元对象会被加入到相应的基级对象的EventListenerSet的里面，所以当基级对象发生某些事件的时候，就会陷入到元级，从而实现了基级到元级的联系。

第 4 章　具有元对象协议的Renew的使用方法

4.1　获取程序

具有元对象协议的Renew的程序可以在http://learn.tsinghua.edu.cn:8080/2004011351/renew/renew/index.html 这个网页上面来获取，在download子页面中，有两个版本的压缩文件可以下载：full package, source package。full package 是包含了编译后的可执行文件的版本。source package 是没有包含可执行文件的版本。如果用户不想自行编译整个系统或者在整个系统编译的时候遇到问题时可以直接下载full package来直接运行。

4.2　目录结构

整个Renew的压缩包解压之后的目录结构是这样的，顶级目录是src，src下面包含的主要的子目录有ant,dist,Meta,user,example,Simulator等等。下面对主要的目录进行一下介绍：

ant目录里面包含着很多的配置文件，其中比较重要的有local.properties,其余的配置文件一般不需要修改。

dist目录下包含的是编译好的可执行文件，在dist/config 目录下面有两个配置文件需要根据实际的需要来进行配置，分别为log4j.properties和renew.properties。

Meta目录下是本项目的主要代码的文件夹，对Renew中增加的元对象协议的部分代码在Meta文件夹下面。

user目录是用户的元级程序的目录，目前情况下用户的元级程序只能放在user/src目录下面,这主要是要使用build.xml进行自动编译的原因。

example目录放置了提供预置示例的rnw文件，相应的元级程序在user目录下，文件名相互对应，用户可以参照示例来编写相应的元级程序。

Simulator目录是整个Renew系统最核心的部分，这里的一些代码由于项目的需要进行了修改，如果用户想了解Renew的实现机制也可以从这部分入手。

4.3 安装前的准备

对带有元对象协议的Renew进行编译首先需要安装有jdk1.5 以上的版本，并且保证jdk的可执行文件在环境变量中进行了正确的配置。

另外需要的软件或者库文件有apache-ant,jalopy-ant,javacc,junit。对于apache-ant 需要将可执行文件加入到环境变量中去，以便能在任意目录下使用ant命令来进行编译。其余的配置将在4.4节中详细的进行说明。

4.4 配置文件的配置选项

共有三个配置文件需要用户进行配置，分别为ant/local.properties,dist/log4j.properties,dist/renew.properties。

ant/local.properties

ant.home=E:\\Renew\\apache-ant-1.6.5

ant.home设置成apache-ant对应的文件夹

bin.javacc=E:\\Renew\\javacc-4.0\\bin\\lib

bin.javacc 设置成javacc 的lib文件夹

option.compile.debug=true

option.compile.optimize=false

dir.jalopy.lib=E:\\Renew\\jalopy-ant-0.1-1.5rc3

jalopy.lib 设置成jalopy-ant 对应的文件夹

dist/log4j.properties

log4j.logger.de=INFO, RenewLog, RenewConLog

log4j.logger.CH=INFO, RenewLog, RenewConLog

log4j.logger.jamr=INFO,RenewLog, RenewConLog

log4j.logger.de.renew.meta=LOG, RenewLog, RenewConLog

在Renew中采用log4j来进行输出信息的控制，这里的设置主要是用来用来控制log4j的输出信息的，log4j的等级有五种分别为LOG,DEBUG,INFO,WARN,ERROR，在LOG状态下输出的信息最多，ERROR状态下输出的信息最少。

log4j.logger.de.renew.meta是针对在Renew中添加的元对象协议的部分而添加的，一般情况下用户只需要修改这个设置就可以了。如果用户想看到调试信

息的话就设置成为DEBUG,否则的话设置成INFO则可将DEBUG的信息过滤掉，不至于显得很乱。

至于输出的信息会在用户的home目录下的renewlogs子目录中以log文件的形式进行保存，以便用户查看。在不同的系统中home目录的位置不同，若用户名为abc，在linux系统中用户目录为/home/abc，在windows系统中用户目录为c:\Documents and Settings \abc

dist/renew.properties

de.renew.classPath=E:\\src\\user\\build\\classes

de.renew.classPath 需要设置成为用户的放置的Renew系统中的user/build/classes 子目录，这样才能保证元级程序被Renew系统所找到。

de.renew.netPath=E:\\workspace\\r10k\\r10k

de.renew.classReinit=true

de.renew.classReinit 一定要设置为true

de.renew.SimulatorMode=-1

4.5 对系统和元级程序的编译

在安装了必要的软件并且对系统进行了正确的配置之后就可以对整个带有元对象协议的Renew进行编译了。对Renew的编译很简单，只需要在src目录下面执行ant命令就可以对整个系统进行编译了。大部分的用户应该并不需要对Renew系统进行编译，只是需要对用户自己编写的元级程序进行编译，那么只需要在src目录下执行ant User 命令就可以编译在user/src目录下用户自己编写的与基级相对应的元级程序了。

4.6 元级程序的编写

元级程序需要按照一定的规则来编写才能够被Renew系统所使用。在example目录下有样例程序可以供用户来参考。元级程序的基本的框架如下：

```
import ***
```

```
public class ClassName extends MetaScript {

    public static void setup(MetaPlugin _metaPlugin) {
    try{
        metaPlugin=_metaPlugin;
        register("firingComplete", "chase_away",
                            "chase_away2", TRANSITION);
        ......//其余的一些注册的函数
        }catch(Exception e){
            ......
        }

    }
    //下面是用户自定义的一些元级的响应的函数
    // 这里只是定义了一些动作，要想将这些动作和基级的网关联起来，要利
用register函数来实现

    public static void chase_away(MetaObject metaObject){
        //add the chased away philosopher to the set
        try{
                //在这里可以调用提供的用户接口
            }catch(Exception e){
                .......
            }
    }
}
```

元级程序中的ClassName必须和文件名和相应的rnw文件相对应。

4.7 整体的执行步骤

采用具有元对象协议的Renew进行系统建模验证，需要进行如下的一些步

骤：

(1) 在Renew系统中进行编辑rnw文件，假定其文件名为abc.rnw。

(2) 编写对应的元级程序，需要相应的文件名为abc.java（与rnw文件的文件名相对应）。

(3) 在src目录下运行ant User 命令，对元级程序进行编译，调试，直至通过。

(4) 在Renew系统中进行执行。

在用户不想使用Renew中的元对象协议的时候，只是简单的省略掉上面的第二步和第三步，便可以顺利的执行了。用户此时可能会得到一个warning。

第 5 章　具有元对象协议的Renew的实际应用的例子

5.1　Hurried Philosopher 的例子

这个例子主要体现在运行的时候动态的改变网的结构，并且也有对网的Marking的改变。利用加入了元对象协议的Renew可以很容易的完整的实现这个例子，在参考文献[8] 中说明了如果用普通的P/T网只能实现哲学家数目固定的情况，并且需要很熟练的Petri网建模的经验才能够完成，具体采用普通的P/T网对限定数目的哲学家问题的解决办法在参考文献[2] 中有具体的描述。

Hurried Philosophers例子介绍：

1. 当一个哲学家没有他的左边的叉子的时候，他可以向他左边的哲学家来请求叉子，同样适用于右边的情况。
2. 当一个哲学家有右边的叉子，并且他右边的哲学家请求这个叉子的时候，他必须答应这个请求。同样适用于左边的情况。
3. 只有一个哲学家同时具有两把叉子的时候才可以进餐。
4. 一些哲学家可以邀请其余的一些哲学家入座，当某个哲学家被邀请入座的时候他自己会携带一把叉子。
5. 一些哲学家可以请左边或右边的哲学家离开座位，或者自己离开座位，当某个哲学家离开的时候，他会带着一把叉子离开。
6. 按照以上的公平的规则，产生的现象是哲学家轮流进行就餐。
7. 在任意时刻，桌上至少有两个哲学家。

本文采用的例子是进行了如下具体限定的情形：

1. 哲学家刚入座的时候将餐叉放在自己的右边。
2. 每个哲学家都可以邀请其余的哲学家，哲学家的数目有一定的上限（≥ 2）。
3. 哲学家不可以自行离开餐桌，某个哲学家离开餐桌只能被其它哲学家驱逐。
4. 哲学家可以驱逐左边或者右边的哲学家。
5. 邀请或者驱逐的哲学家位于左边或者右边随机决定。
6. 初始的情况桌上有两个哲学家。

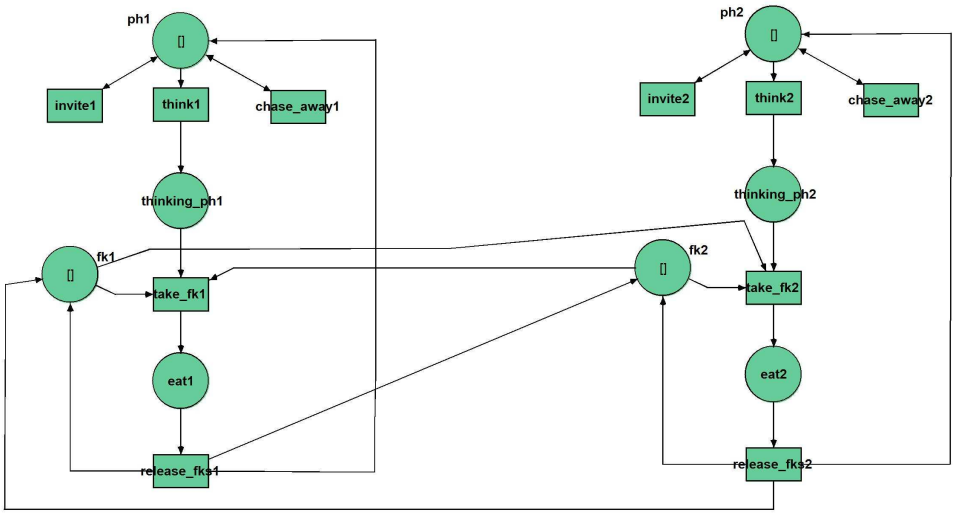

图 5.1　Hurried Philosophers例子的初始情况

在元级程序中我们定义了三种元级对象，分别对应基级对象中的Transition:invite,chase_away,eat。元级对象invite、chase_away对应的元级程序用来动态改变相应的基级的结构。invite的元级程序会加入一个philosopher所需要的所有基级的结构，包括图5.1中Place：ph,thingking_ph,eat,tk和Transiton:invite,chase_away,think,take_fk,release_fks。chase_away 对应的元级程序可以删除相应基级的Place和Transition。另外还要在invite和chase_away的元级程序中，进行一些Arc的加入和删除来维持元级的Philosophers之间的相邻的关系。元级对象eat用来监视基级的行为。在invite、chase_away和eat的相应的元级的代码中，进行输出一些信息来监视整个系统的运行状况。

下面就是实际的系统的一次运行结果：

```
2 invite 3 to sit right
1 chase away 2
1 invite 2 to sit right
1 chase away 3
1 invite 3 to sit right
3 invite 7 to sit right
3 chase away 1
3 chase away 7
3 invite 1 to sit left
3 chase away 2
1 invite 2 to sit left
1 chase away 2
1 invite 2 to sit left
1 chase away 3
1 invite 3 to sit right
1 invite 7 to sit right
3 invite 6 to sit left
6 chase away 7
6 invite 7 to sit right
3 invite 5 to sit left
5 chase away 3
2 is eating
7 invite 3 to sit right
7 chase away 6
2 invite 6 to sit right
2 invite 4 to sit left
2 chase away 4
6 chase away 2
7 is eating
5 is eating
7 invite 2 to sit right
2 chase away 7
6 invite 7 to sit right
2 chase away 1
2 chase away 7
2 is eating
2 chase away 6
2 chase away 5
2 invite 1 to sit right
1 is eating
2 is eating
2 invite 7 to sit left
1 is eating
7 chase away 3
7 chase away 1
```

第 6 章　工作总结及展望

6.1　工作总结

目前在关于Petri网的模拟工具的元对象协议扩展方面的研究并不多见，在参考文献[2,3] 中设计了一种实现方式，并且打算以GreatSPN工具作为基础进行扩展，但是目前为止并没有相关工作的结果发表。在本文中设计并基于Renew实现了在有色网上的元对象协议。Petri网是强大的用于系统建模和协议验证的工具，Renew是众多Petri网模拟工具中比较优秀的一个。在Renew中实现元对象协议，可以使得对不断改变的系统的建模变得更加的容易，更加的直观。通过Hurried Philosophers的例子来说明了其在建模方面的优势。在Renew中增加元对象协议会有如下的一些应用：

对于普适的情况：

- 可以简化基级别的网模型
- 可以在不改变基级的情况下获取基级信息
- 可以通过对基级的控制来改变基级网结构

对于处理器描述的方面

- 简化分支预测恢复的建模
- 获取处理器运行时的一些信息

本文中所提到的所有的具体实现都可以在http://learn.tsinghua.edu.cn:8080/2004011351/renew/renew/index.html获得源代码，具体的使用方法也请参考该网址。

6.2　工作展望

这项工作主要是由于在用Renew建模复杂处理器的时候引出的，现在具有的功能已经可以解决该问题。今后可以在处理器描述的更多的方面应用元级特性，同时也可以在应用的同时不断的改进元对象协议。在对可重构的系统的建模方面，具有元对象协议的Renew同样可以得到应用。

插图索引

表格索引

公式索引

参考文献

[1] H ¨ ursch W, Lopes V. Separation of Concerns. Proceedings of Technical Report, Northeastern University, Boston, 1995

[2] Capra L, Cazzola W. Self-Evolving Petri Nets. J. UCS, 2007, 13(13):2002–2034

[3] Capra L, Cazzola W. A Petri-Net Based Reflective Framework for the Evolution of Dynamic Systems. Electr. Notes Theor. Comput. Sci., 2006, 159:41–59

[4] Duvigneau O K F W M. Renew - User Guide. Technical report, Theoretical Foundations Group of the Department for Informatics of the University of Hamburg, May 26, 2006. http://www.renew.de

[5] Duvigneau O K F W M. Renew - Architecture Guide. Technical report, Theoretical Foundations Group of the Department for Informatics of the University of Hamburg, May 26, 2006. http://www.renew.de

[6] 林明. 基于有色网的处理器描述研究[硕士学位论文]. 北京: 清华大学, 2008

[7] Golm M. Design and Implementation of a Meta Architecture for Java[Doctor Thesis]. Germany: Computer Science Department, the Friedrich-Alexander University, April, 1997

[8] Sibertin-Blanc C. Concurrent object-oriented programming and petri nets, volume 2001. Springer-Verlag New York, Inc., 2001: 536–537

致　谢

衷心感谢导师王生原老师对本人的精心指导。感谢同组的林明、郭胜基两位师兄对本人的帮助，以及实验室全体同学和老师的帮助和支持。

本课题受到国家自然科学基金（60573017）的资助，特此致谢。

声　明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，独立进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人享有著作权的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。

签　名：_____ 日　期：_____

附录 A　开题时的调研阅读报告

A.1　Introduction

Nowadays, as processors are more and more complicated, and new processors are coming out every year, the compiler transplanting has become a much more difficult work than before. The compiler researcher try to use some tools to describe the processors and to get some useful information for transplanting compiler. The commonly used tools are dedicated processor description language, functional programming language, automata , and Petri net.

In our lab, a previous processor description work was *done*[3]. This work is mainly about using a Colored Petri Net Simulation tool (Renew) to describe the Mips R10000 SuperScalar Processor. Some difficulties appeared during this research, and the solution raised is to hack the Renew tool to and MetaObject Protocol to it. The primary goal of my work is to extend the Renew tool to meet the requirement of the processor description. And simultaneously, the extended tool could be more powerful and could be used in many areas for different applications.

A.2　Colored Petri Net

Colored Petri *Net*[1] is a kind of high-level petri net. The basic components of it are as same as other petri nets, they are places, arcs , transitions. The difference between Colored Petri Net and low level Petri Net is the tokens in Colored Petri Net are different while they are not in the case of low level Petri Net. Colored Petri Net could make the description more easily and decrease the scale using this property. There could be some inscriptions on the arcs to judge if a transition is active using the ”color”.

A.3 The Renew tool

A.3.1 Reference Net

The reference net used in renew tool is Colored Petri Net with java inscription language and some extentions . Some properties of of it is listed as follows:

- Each place and transition can be assigned a name.
- Arc inscription could be added so that the result of a transition is evaluated by it.
- Guard inscription could prevent a transition from happening if it is tested to be false.
- The inscription language of reference nets has been extended to include tuples.
- User can call Net from java using the stub mechanism provided.
- A timed mode is provided in which each token is assigned a time when the token is available.
- There are some extended arcs:
  - Flexible Arcs: allow mutiple tokens to be moved by an arc
  - Clear arcs: remove all tokens in a place
  - inhabitor arcs :make sure a certain kind of token is not a Place

A.3.2 *Anatomy*[3,6]

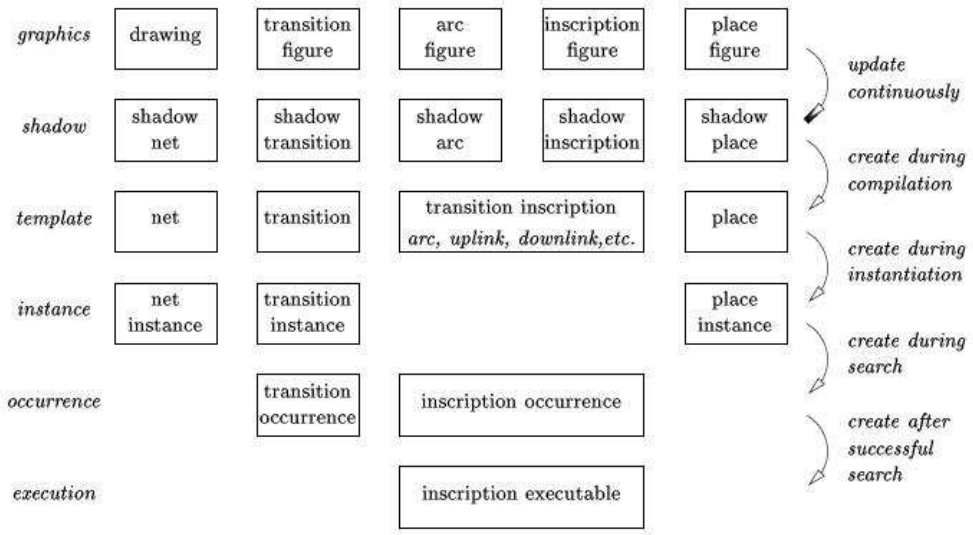

图 A.1 Renew的结构分析

The structure of the Renew immplementation is shown in A.1 . There are four levels in this structure, i.e. graphics , shadow ,template ,instance. The graphics in the interface for user to draw a figure using arcs , places ,transitions and so on . The compilation from graphics representation to shadow net is real-time. In the shadow net layer, some useless information are eliminated , such as the coordiantes of a place. And then the shadow net is compiled into template net ,the net layer representation is more suitable for a simuation. Since we called it template net layer , net instance is generated from the template at run time.

The simulation work is done by the simulation engine. I will give a brief explain of how the simulation engine works below. The central controller of the engine is a Simulator. Every transition instance is a searchable, which means they can be searched for suitable bindings and after that they can be fired. All the searchables are kept in a global search queue. And in order to get some efficiency , the search queue keep a time, and the engine can decide before a certain time , some transitions won't be able to be fired , so the scope of the search is reduced. The simulator will get a Searchable from the search queue and using the unifying algorithm to find a binding for this Searchable if there is . And if a binding is found, it will be executed, if not, the simulator will get another searchable from the queue until it is empty. There are some technic used in the search engine to make it efficient.

A.4 Reflection & MetaObject Protocol

Basic concepts:[2] Reflection:

Reflection is the capability of a computational system to "reason about and act upon itself" and adjust itself to changing conditions. The computationaldomain of a reflective system is the structure and the computations of the system itself.

Reification:

Reification is the act of making something accessible which normally is not available in the programming model or hidden from the programmer.

base level system:

The system actually solving a real application problem.

meta level system

It is constructed by base level representation , base level control and meta level interface.

Causal connection

The connection between base level and meta level . The changes of the base level must be reflected in the meta level.

MetaObject Protocol in Petri Net Simulation[4]

In this model, the base level would be a Petri Net Simulation tool, and the meta level would have the ability to got information from the base level and control the base level . The control over base level could be removing tokens ,add tokens and moreover it could be changing the Petri net of the base level. The most important point in Simulating Petri Net using a MetaObject Protocol is the base level reification. By choosing a good representation of the base level, the reflection would be easy.

A.5 My work

My work is about adding the MetaObject Protocol to the current Renew tool . This extention to the tool would make it more powerful . The base level of this protocol is chose to be the NetInstance level in the Renew architecture. And the meta level code would be the code implemented by users, probably the same as that draw the Nets.

The current NetInstance is good enough to represent the base level and could be used by the meta level for interaction.

The interface or the MetaObject Protocol to be provided would be fixed, which means users can't change it themselves. The basic method we provide would be partitioned into two kinds.

The first kind is like

```
transition_started(…){
    ...
}
```

which is to be implemented by user. The second kind is like

```
add_token(…){
    ...
```

```
        }
```

which could control the base level.

A.6 Reference

[1] K. Jensen , "A brief introduction to Colored Petri Nets", Workshop on the Applicability of Formal Models, 2 June 1998, Aarhus, Denmark.

[2]Golm M. Design and Implementation of a Meta Architecture for Java[Doctor Thesis]. Germany: Computer Science department, the Friedrich-Alexander University, April,1997

[3] Lin Ming.Study on Machine Description based on Colored Petri Net.Beijing Tsinghua University. 2008

[4] Capral L,Cazzola W. A Petri-Net Based Reflective Framework for the Evolution of Dynamic Systems.Electr.NotesTheor.Comput.Sci.,2006,159:41-59

[5] Duvigneau OKFWM. Renew - User Guide. Technical report, Theoretical Foundations Group of the Department for Informatics of the University of Humberg, May 26,2006. http://www.renew.de

[6]Duvigneau OKFWM. Renew - Architecture. Technical report, Theoretical Foundations Group of the Department for Informatics of the University of Humberg, May 26,2006. http://www.renew.de

附录 B　Hurried Philosophers 例子的元级程序

```
import java.lang.*;
import java.util.Set;
import java.util.HashSet;
import java.util.HashMap;
import java.util.Random;
import de.renew.meta.*;
import de.renew.net.arc.*;
import de.renew.net.Transition;
import de.renew.unify.Tuple;
import java.util.Iterator;

public class hp extends MetaScript {
    //folks are  placed on the right side of philosophers
    //the number of the philosophers
    public static int amount=8;
    //keep a set of unseated philosophers
    public static Set numSet;
    public static HashMap left;
    public static HashMap right;
    public static void setup(MetaPlugin _metaPlugin) {
            try{
        metaPlugin=_metaPlugin;
        numSet=new HashSet();
        left=new HashMap();
        right=new HashMap();
        left.put("1","2");
        left.put("2","1");
```

```
        right.put("1","2");
        right.put("2","1");
        for(int i=3;i<=amount;i++){
            numSet.add((new Integer(i)).toString());
        }
        register("firingComplete", "chase_away", "chase_away2", TRANSITION);
        register("firingComplete", "chase_away", "chase_away1", TRANSITION);
        register("firingComplete","invite","invite1",TRANSITION);
        register("firingComplete","invite","invite2",TRANSITION);
        getMTs("invite1","chase_away1","think1","take_fk1","release_fks1",
            "invite2","chase_away2","think2","take_fk2","release_fks2");
        register("firingComplete","take_fk","take_fk1",TRANSITION);
        register("firingComplete","take_fk","take_fk2",TRANSITION);
        }catch(Exception e){
            System.out.println("UserScript.setup "+e.toString());
        }

    }
    public static void chase_away(MetaObject metaObject){
        //add the chased away philosopher to the set
        try{
        Transition transition=
        ((MetaTransition)metaObject).getTransitionInstance().getTransition();
        String name=transition.getName();
        String mynum=name.replace("chase_away","");
        Random random=new Random();
        String num;
        if(left.get(mynum).equals(right.get(mynum)))//only two phils left
            return ;
        if(random.nextFloat()>0.5){
            num=(String)(left.get(mynum));
```

```
    MetaPlace tmpMp=getMetaPlace("fk"+num);
    if(tmpMp.getPlaceInstance().getNumberOfTokens()<=0)
        return;
    left.put(mynum,left.get(num));
    right.put(left.get(num),mynum);
    String leftnum=(String)(left.get(mynum));
    removeArc("fk"+num,"take_fk"+mynum,Arc.in);
    removeArc("fk"+num,"release_fks"+mynum,Arc.out);
    if(numSet.size()>-1){
    addArc("fk"+leftnum,"take_fk"+mynum,Arc.in);
    addArc("fk"+leftnum,"release_fks"+mynum,Arc.out);
    }

}else{
    num=(String)(right.get(mynum));
    MetaPlace tmpMp=getMetaPlace("fk"+num);
    if(tmpMp.getPlaceInstance().getNumberOfTokens()<=0)
        return;
    right.put(mynum,right.get(num));
    left.put(right.get(num),mynum);
    String rightnum=(String)(right.get(mynum));
    removeArc("fk"+num,"take_fk"+rightnum,Arc.in);
    removeArc("fk"+num,"release_fks"+rightnum,Arc.out);
    if(numSet.size()>-1){
    addArc("fk"+mynum,"take_fk"+rightnum,Arc.in);
    addArc("fk"+mynum,"release_fks"+rightnum,Arc.out);
    }
}
System.out.println(mynum+" chase away "+num);
left.remove(num);
right.remove(num);
```

```
    numSet.add(num);
    remPFs("ph"+num,"thinking_ph"+num,"eat"+num,"fk"+num);
    remTFs("invite"+num,"think"+num,
        "chase_away"+num,"take_fk"+num,"release_fks"+num);
        }catch(Exception e){
            System.out.println("UserScript.chase_away "+e.toString());
            }
}
public static void invite(MetaObject metaObject){
    //pick a num from the set randomly
    //decide the invited ph to sit left or right
    try{
    Transition transition=
    ((MetaTransition)metaObject).getTransitionInstance().getTransition();
    String name=transition.getName();
    String mynum=name.replace("invite","");
    Random random=new Random();
    Iterator iterator=numSet.iterator();
    String num=new String();
    if(iterator.hasNext()){
        num=(String)(iterator.next());
        iterator.remove();
        System.out.print(mynum+ " invite "+num+ " to sit ");
    }else{
        return;
    }
    String leftnum,rightnum;
    if(random.nextFloat()>0.5){//sit left
        System.out.println("left");
        leftnum=(String)(left.get(mynum));
        rightnum=mynum;
```

```
        right.put(num,mynum);
        left.put(num,left.get(mynum));
        right.put(left.get(mynum),num);
        left.put(mynum,num);
    }else{//sit right
        System.out.println("right");
        rightnum=(String)(right.get(mynum));
        leftnum=mynum;
        left.put(num,mynum);
        right.put(num,right.get(mynum));
        left.put(right.get(mynum),num);
        right.put(mynum,num);
    }
    addPs("ph"+num,"thinking_ph"+num,"eat"+num,"fk"+num);
    addTs("invite"+num,"think"+num,"chase_away"+num,
        "take_fk"+num,"release_fks"+num);
    addArc("ph"+num, "invite"+num, Arc.both);
    addArc("ph"+num,"chase_away"+num,Arc.both);
    addArc("ph"+num,"think"+num,Arc.in);
    addArc("thinking_ph"+num,"think"+num,Arc.out);
    addArc("thinking_ph"+num,"take_fk"+num,Arc.in);
    addArc("fk"+num,"take_fk"+num,Arc.in);
    addArc("eat"+num,"take_fk"+num,Arc.out);
    addArc("eat"+num,"release_fks"+num,Arc.in);
    addArc("ph"+num,"release_fks"+num,Arc.out);
    addArc("fk"+num,"release_fks"+num,Arc.out);
    removeArc("fk"+leftnum, "take_fk"+rightnum,Arc.in);
    removeArc("fk"+leftnum,"release_fks"+rightnum,Arc.out);
    addArc("fk"+num,"take_fk"+rightnum,Arc.in);
    addArc("fk"+leftnum,"take_fk"+num,Arc.in);
    addArc("fk"+num,"release_fks"+rightnum,Arc.out);
```

```
        addArc("fk"+leftnum,"release_fks"+num,Arc.out);
        getMetaPlace("ph"+num).insertToken(new Tuple(0));
        MetaPlace temp=getMetaPlace("fk"+num);
        temp.insertToken(new Tuple(0));
        getMTs("invite"+num,"chase_away"+num,"think"+num,
            "take_fk"+num,"release_fks"+num);
        register("firingComplete","invite","invite"+num,TRANSITION);
        register("firingComplete","chase_away","chase_away"+num,TRANSITION);
        register("firingComplete","take_fk","take_fk"+num,TRANSITION);
        }catch(Exception e){
                System.out.println("UserScript.invite "+e.toString());
        }
    }
    public static void take_fk(MetaObject metaObject){
        try{
        String name=((MetaTransition)metaObject).getName();
        String num=name.replace("take_fk","");
        System.out.println(" "+num+" is eating");
        }catch(Exception e){
            System.out.println("UserScript.eat "+e.toString());
        }
    }
}
```

在学期间参加课题的研究成果

个人简历

1987年04月02日出生于辽宁省北镇县。

2004年 9 月考入清华大学计算机科学与技术系计算机科学与技术专业至今。

发表的学术论文