第31卷 第23期 计 算 机 工 程 2005年12月
Vol.31 № 23 Computer Engineering December 2005
·基金项目论文· 文章编号：1000—3428(2005)23—0025—03 文献标识码：A 中图分类号：TP393.07

通用性转换代理的研究与实现

吴黎兵[1]，吴产乐[1]，崔建群[2]，张沪寅[1]
（1. 武汉大学计算机学院国家多媒体软件工程技术研究中心，武汉 430072；2. 华中师范大学计算机科学系，武汉 430079）

摘 要：根据网管软件的现状，对SNMP转换代理存在的通用性问题进行了较深入的研究，提出通过构造通用适配器和动态载入XML设备模板的方法，实现具有较好通用性的SNMP转换代理，使基于SNMP的网络管理延伸至更广泛的领域。
关键词：SNMP协议；网络管理；转换代理；通用性和可扩展性

Research and Implementation of Universal SNMP Proxy Agent

WU Libing[1], WU Chanle[1], CUI Jianqun[2], ZHANG Huyin[1]
（1. National Engineering Center of Multimedia Software, School of Computer, Wuhan University, Wuhan 430072；
2.Computer Science Department, Central China Normal University, Wuhan 430079）

【Abstract】The paper researches on universal character of SNMP proxy agent after analyzing the status of network management software. A method to implement universal SNMP proxy agent through constructing universal adaptors and dynamic loading XML device models is put forward. The method enlarges the field of network management based on SNMP.
【Key words】SNMP; Network management; Proxy agent; Universal and scalability

SNMP协议真正被各大产品供应商广泛应用于其网络设备中，是在20世纪90年代后期，在此之前的设备通常没有内置SNMP代理，SNMP管理者无法通过标准的SNMP协议操作获得设备的相关数据，从而对设备进行管理。一些非网络核心设备只提供较弱的管理功能，通常没有提供SNMP接口，无法将其纳入网络管理系统的管理对象中。

解决上述问题的一条较好的途径就是在网络管理系统和不支持SNMP的被管设备中间增设一个SNMP转换代理（Proxy Agent）。通过这个转换代理，将网管系统发出的SNMP协议操作转换成被管对象（硬件或软件）可以识别的命令，从而获得所需的管理数据或对设备进行相关的设置，然后再将设备的响应信息转换为SNMP响应PDU（协议数据单元），传送回网络管理系统。这种方法可以在基本不改变网络管理系统结构的情况下，实现对不支持SNMP协议对象的统一管理。

因此，研究与实现SNMP转换代理，可以扩大网络管理的被管对象，使网络管理延伸至更广泛的领域。

1 转换代理通用性研究

1.1 研究内容

要实现SNMP转换代理，并使其具有通用性，需要进行以下几个方面的研究。

（1）SNMP协议PDU与被管对象可识别协议或命令之间的转换。将NMS的SNMP命令PDU转换为被管对象可识别协议或命令，是SNMP转换代理需要完成的最基本也是最重要的功能。通过对SNMP协议PDU进行解码，可以将该PDU所要完成的操作命令、操作对象等相关信息分离出来，然后根据目标被管对象的类型，进行相应的二次编码，转换成被管对象可识别的信息；同样当接收到被管对象的反馈信息或主动发出的故障信息时，转换代理也需要将其封装成相应的SNMP协议PDU，发送给NMS。

（2）MIB变量与被管对象可识别变量之间的映射与转换。由于基于SNMP的NMS只能对MIB变量进行操作，因此当NMS向被管对象发出SNMP请求时，需要将标准的MIB变量转换为被管对象内部的可访问变量，这就要求在转换代理内部保存一张可动态改变的MIB变量映射表，通过这张表来得到MIB变量与设备可识别变量之间的映射关系，从而达到转换的目的。

（3）转换代理与被管对象之间的通信机制。由于被管对象类型的多样性，使得转换代理与被管对象之间的通信问题也变得复杂起来。目前不支持SNMP管理的被管对象通常支持telnet、http等方式的管理，这些都是建立在TCP/IP协议栈之上，既可以直接采用应用层协议发送命令，也可以通过socket接口进行相互通信。除此之外，大部分设备还可以直接通过计算机的串口与设备的Console口相连，来对设备实行管理，这可以通过RS232通信机制对设备进行访问。如果这些通信机制都不支持，也可以通过加载特有的协议通信包来实现转换代理与被管对象之间的通信。

1.2 构造通用适配器

前面提到的3方面的研究，如果只针对某一个或某一类特定的被管对象，实现起来难度相对较小，但如果希望转换代理具有良好的通用性，则比较困难。为此本文提出一种动态加载通用适配器的模型，使得在不改动Proxy Agent核心模块的前提下，达到支持尽可能多类型设备的转换目的。

首先我们在设计Proxy Agent主模块时，并不针对某一特定设备来实现，而是在需要与非SNMP设备进行数据交换时，

基金项目：国家“863”引导项目（2003AA001032）；软件工程国家重点实验室开放基金资助项目
作者简介：吴黎兵(1972－)，男，博士生，研究方向：网络管理，网络通信；吴产乐，教授、博导；崔建群，硕士；张沪寅，副教授
收稿日期：2004-10-13 **E-mail：**wufox@sina.com

提供 3 个适配器抽象接口：传输适配器 TransAdaptor、私有数据转换适配器 PrivateDataAdaptor，陷阱适配器 TrapAdaptor，和一个对象接口：私有数据接口 PrivateDataIf。

适配器模型如图 1 所示。

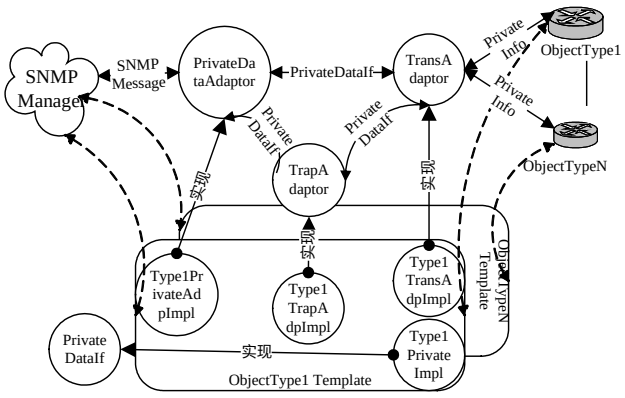

图 1 适配器模型

适配器模型中的 TransAdaptor 负责 Proxy Agent 与非 SNMP 设备之间的通信，包括连接、发送数据、接收数据以及断连等功能；PrivateDataAdaptor 负责标准 SNMP 请求与非 SNMP 设备可识别协议或命令之间的转换；TrapAdaptor 负责定时轮询非 SNMP 设备并设置陷阱阈值，一旦达到发送陷阱条件，即构造 SNMP Trap 报文，将报文发送至 SNMP 管理者。私有数据接口 PrivateDataIf 则为上述适配器提供互相传递的对象。这些接口被主程序调用直接与 SNMP Manager 和被管对象进行通信。当需要对某类非 SNMP 对象,例如 ObjectType1 进行管理时，只需要提供 ObjectType1 Template，在 Template 中分别实现上述接口所提供的所有方法，就可以被系统实例化后用于实现对 ObjectType1 的实际管理。

1.3 动态载入适配器

对于不同的设备类型需要实例化不同的适配器，这些适配器通过设备模板的 XML 定义通知系统。设备模板的 XML Schema 结构图如图 2 所示。

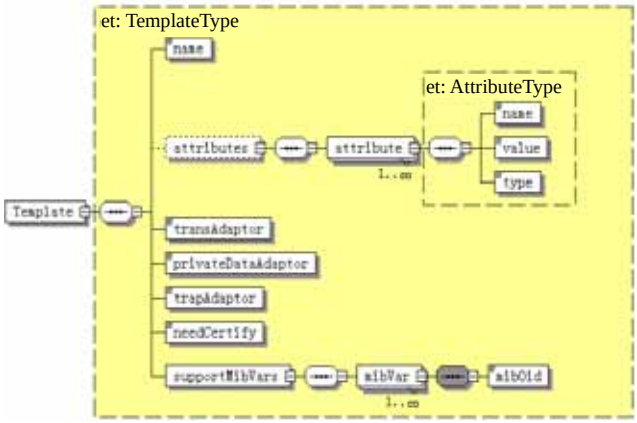

图 2 设备模板的 XML Schema 结构

其中 name 描述设备模板的类型名称，transAdaptor、privateDataAdaptor 和 trapAdaptor 分别描述适配器所在的类名，attributes 用来设置适配器可能用到的与设备自身特性相关的各种属性，needCertify 标识该设备是否需要身份验证，supportMibVars 则给出该类型设备支持的 MIB 变量标识符。

符合上述 XML Schema 的设备模板 XML 文件通过 Proxy Agent 的配置客户端，导入数据库相关数据表中。当 Proxy Agent 的服务器端启动时，会根据数据表中适配器类所在的位置，在模板构造函数中将其动态加载至设备模板对象实例的属性中，成为适配器接口的实现类：

```
template = new Template(templateId , templateName,
(TransAdaptorIf)Class.forName(transAdaptor).newInstance(),
(PrivateDataAdaptorIf)Class.forName(privateDataAdaptor).newInstance(),
(TrapAdaptorIf)Class.forName(trapAdaptor).newInstance(),
needCertify);
```

这样对于不同设备类型，系统就会通过相同的抽象接口调用不同的适配器实现类，来完成转换代理的转换功能。

综上所述，在出现新的需要进行 SNMP 管理的非 SNMP 设备时，按照以上的设计模式，只需编写 4 个适配器类，实现适配器接口提供的所有方法，再通过设备模板的 XML 文件导入转换代理系统，而不需要更改转换代理的其它任何模块，就可以实现对新设备的 SNMP 管理，具有很好的通用性。

2 通用转换代理的实现

2.1 系统模型

图 3 描述了采用上述方法对不支持 SNMP 对象进行管理的通用转换代理的系统模型。

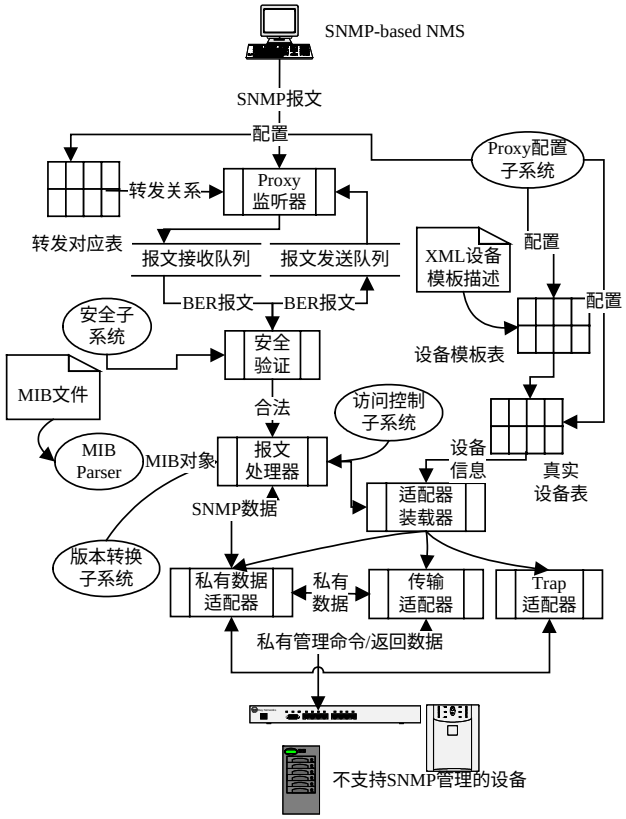

图 3 系统模型

NMS 管理站对非 SNMP 设备发出 SNMP 管理操作，只要其目的地址是发往转换代理所在的主机，且端口已经在转换代理的转发对应表中注册过，转换代理就会对发往该端口的 SNMP 标准报文进行转换。

转换代理的 Server 端在启动时的初始化过程中，首先注册转发对应表中需进行监听的端口，然后启动多个线程进行监听。如某个端口有 SNMP 请求报文到来，则首先经过队列缓冲，然后通过源地址的比对以及事先配置好的安全信息进行安全验证，通过安全验证的报文交由报文处理器分析处理。

报文处理器首先根据报文目的地址获得被管设备的 ID，根据提供的共同体名，通过访问控制子系统对该设备支持的 MIB 对象和可操作权限的检查后，调用适配器装载器装载该

模板的 3 个适配器：传输适配器 TransAdaptor，私有数据转换适配器 PrivateDataAdaptor 和陷阱适配器 TrapAdaptor。在适配器的互相协作下完成对设备管理信息的采集。最后按照上述相逆的过程，将采集到的数据转换为标准 SNMP 报文，返回给请求报文的发出者。

2.2 转换代理相关数据表设计

通用 SNMP 转换代理的实现至少需以下数据表的支持：

（1）设备模板表 XspaTemplateTable：存放模板的基本信息；

（2）设备模板属性表 XspaTemplateAttrTable：存放模板的其它属性；

（3）设备访问控制表 XspaTemplateMibTable：存放模板支持的 MIB 变量及访问权限；

（4）Snmp 报文转换表 XspaMessageProxyTable：存放已转换过的报文和私有数据对应表；

（5）真实设备表 XspaNodeTable：存放实际被管对象的基本信息；

（6）设备属性表 XspaNodeAttrTable：存放实际被管对象的其它信息；

（7）设备安全表 XspaNodeSecurityTable：存放 SNMP 安全参数与实际被管对象安全参数；

（8）转发对应表 XspaForwardTable：存放 Proxy 接收和发送的对应关系表。

其中设备模板表 XspaTemplateTable 中的 templateId 和真实设备表 XspaNodeTable 中的 nodeId 由数据库系统按照递增序列自动生成，其他数据表中的 templateId 和 nodeId 则都是以上述两表的相应字段作为外码，受其约束。

这些数据表将在配置子系统和 Server 启动时用到。

2.3 转换代理配置

转换代理第一次工作以前，需要通过 Proxy 配置子系统对上述数据表进行相应的配置，才能正常工作。

首先需要配置与设备模板相关的数据表：设备模板表 XspaTemplateTable，设备模板属性表 XspaTemplateAttrTable，设备访问控制表 XspaTemplateMibTable。这些数据表的配置非常简单，只需写好设备模板的 XML 文件，Proxy 配置子系统就会通过 XML 解析器和 XML Schema，将该设备模板的所有信息，一次性写入数据库，完成对此模板的配置。

第二步需配置真实设备相关的数据表：真实设备表 XspaNodeTable，设备属性表 XspaNodeAttrTable 和设备安全表 XspaNodeSecurityTable。配置这些表的前提是该真实设备的模板已存在于数据库中，否则无法配置这些真实设备。

最后，配置转发对应表 XspaForwardTable，通过配置该表可以通知转换代理，在某个端口上接收到的报文，应将其转发至何处，并可根据源地址的正确性与否决定是否转发收到的报文。要使转换代理可以正常监听这些端口，这些端口所对应的目的真实设备必须已经存在，否则无法找到该设备的模板，无法作进一步的处理。

将以上数据表配置好后，如果 Proxy Server 处于运行状态，则通过 Java RMI 调用，实时通知 Server 进行相应调整，否则只写入数据库，待 Proxy Server 启动时从库中载入。

对数据库的操作通过 JDBC 接口完成。

3 结论与展望

根据本文所讨论的这种构造通用 SNMP 转换代理的方法，我们已经完成了系统原型的开发，并针对不支持 SNMP 但支持 CLI 接口的交换机进行了具体实现，系统运行稳定。

参考文献

1 Thai B, Seneviratne A. The Use of Software Agents as Proxies. Fifth IEEE Symposium on Computers and Communications (ISCC 2000). Antibes, France: University of New South Wales, 2000-07: 546

2 John A, Vanderveen K, Sugla B. A Java-based SNMP Agent for Dynamic MIBs. IEEE/Global Telecommunications Conference, GLOBECOM '99, 1999, 1a: 396-400

3 Cisco StrataView Plus SNMP Proxy Agent Manual. http://www.cisco.com/en/US/products/sw/netmgtsw/ps2177/products_programming_reference_guide_chapter09186a00800d804c.html

（上接第 21 页）

```
        od;
        if bCompleteFlag = false then
            ReBuild(ROBDDNode);
        endif;
    od;
    return;
end OptimizateNode;
```

4 结论

模型检验是一种实用的自动验证方法，能在不满足系统规格说明时提供反例，来保证软件和硬件设计的正确性。本文在 GPVW 算法的基础上，给出一种从 LTL 公式导出识别该公式的 Büchi 自动机的优化方法，基于 ROBDD 描述结点和结点关系上，通过布尔优化方法，消除等价结点，从而得到优化的 Büchi 自动机。

参考文献

1 Pnueli A. The Temporal Logic of Programs. In: Proceedings of the 18th IEEE Symposium on Foundations of Computer Science, 1977: 46-77

2 Emerson E A, Clarke E M. Using Branching-time Temporal Logic to Synthesize Synchronization Skeletons. Science of Computer Programming, 1982,2(3): 241-266

3 Clarke E M, Emerson E A, Sistla A P. Automatic Verification of Finite-state Concurrent Systems Using Temporal-logic Specifications. ACM Transactions on Programming Languages and Systems, 1986,8(2): 244-263

4 Courcoubetis C, Vardi M, Wolper P. Memory-efficient Algorithms for the Verification of Temporal Properties[M]. In: Formal Methods in System Design. American: Kluwer Academic Publishers, 1992,1: 275-288

5 Gerth R, Peled D, Vardi M Y, et al. Simple on-the-fly Automatic Verification of Linear Temporal logic. In: Proceedings of the 15th Workshop on Protocol Specification Testing, and Verification, Varsaw, North-Holland, 1995-06

6 Cornelia van Wyk. An LTL Verification Based on Automata Theory[MSc Thesis]. Department of Computer Science, University of Stellenbosch, 1999-12

7 Bryant R E. Graph-based Algorithms for Boolean Function Manipulation. IEEE Transactions on Computers, 1986,C-35(8): 677-691

8 Bryant R E. Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams. ACM Computing Surveys, 1992,24(3): 293-318