UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

WILLIAM LOSINA BRANDÃO

Geração automática de código para agentes SNMP e CLI

Trabalho de Diplomação.

Prof. Dr. Luigi Carro
Orientador

Porto Alegre, novembro de 2010.

AGRADECIMENTOS

Agradeço a minha família pelo apoio e carinho nesta longa jornada. Aos amigos por compreenderem minhas ausências freqüêntes nestes últimos tempos. E especialmente a minha namorada Letícia, pois sem ela nada disso seria possível.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

CLI	Command Line Interface
IDE	Integrated Development Enviroment
JVM	Java Virtual Machine
MIB	Management Information Base
OID	Object Identifier
OSI	Open Systems Interconnection
RFC	Request For Comments
RTOS	Real Time Operational System
SNMP	Simple Network Management Protocol
SQL	Structured Query Language
UDP	User Datagram Protocol

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

O processo de desenvolvimento de software embarcado exige cada vez mais atenção, devido ao aumento constante na complexidade, além do alto custo de manutenção dos equipamentos uma vez em campo. As técnicas tradicionais de engenharia de software não são suficientes para lidar com os requisitos não funcionais do projeto de sistemas embarcados, tais como o tempo de colocação do produto no mercado, uso de memória, tempo de resposta e custo de projeto e manutenção.

Na área de telecomunicações, a manutenção de um equipamento em campo pode tornar-se inviável, seja pela quantidade de equipamentos distribuídos em uma área muito grande, ou pela dificuldade de acesso nos casos em que esses equipamentos se encontram em áreas muito remotas. Além disso, como será visto, o tempo de colocação do produto no mercado é de extrema importância, sendo decisivo para o sucesso ou fracasso de um projeto.

Tendo em vista esses fatores, este trabalho propõe uma forma de aumentar a automatização na geração de código para sistemas embarcados de gerência de equipamentos de telecomunicação, com o uso de uma arquitetura que propicia uma redução no número de linhas a serem escritas pelo programador, através do uso de uma interface bem definida entre o sistema de gerência e os dados do equipamento. Desta forma, pode-se obter uma redução no tempo de desenvolvimento, bem como nos gastos com a manutenção dos equipamentos.

Palavras-Chave: SNMP, CLI, telecomunicações, geração automática de código, sistemas embarcados.

Automatic code generation for SNMP agents and CLI: software engineering applied to embedded systems

ABSTRACT

The development process of embedded software demands an ever increasing attention due to the growth in complexity, besides the high cost to repair equipment once in field. The traditional software engineering techniques are not enough to deal with the non-functional requirements of the embedded systems design, such as time-to-market, memory footprint, performance and development/maintenance cost.

In the telecommunications field, the maintenance of equipment once in field may be unviable, be it due to the existence of a large number of equipments distributed in a vast area, or due to the difficulty in reaching such equipments when they are located in remote areas. Besides that, as it will be presented, the time-to-market is of extreme importance, consisting in a decisive factor to the success or failure of a project.

Having all these factors in mind, this work proposes a way to increase the code generation automation in management software for embedded telecommunication systems, with the use of an architecture that allows for a reduction in the number of lines of code to be written by the programmer, by using a well-known interface between the management system and the equipment data. Therefore, a reduction in both the development time and development/maintenance costs can be achieved.

Keywords: SNMP, CLI, telecom, automatic code generation, embedded systems.

1 INTRODUÇÃO

Sistemas de telecomunicação incluem dispositivos de uma vasta gama de complexidade, desde equipamentos de chaveamento não-gerenciáveis até grandes roteadores nas centrais das operadoras de telecomunicações. O aumento na utilização de serviços de comunicação de dados demanda um aumento no tamanho e na área de abrangência das redes de comunicação. Desta forma, as operadoras de telecomunicações buscam cada vez mais equipamentos gerenciáveis remotamente – em oposição aos equipamentos ditos não-gerenciáveis, que devem ser configurados manualmente por um técnico no momento da instalação, e exigem uma manutenção presencial e portanto muito dispendiosa –, tanto para permitir uma gerência mais rápida e eficaz dos recursos, quanto para uma redução nos custos de manutenção.

De acordo com Sridhar (2003) equipamentos de telecomunicação tipicamente executam um sistema operacional de tempo real, apresentando memória RAM e flash limitadas, e oferecem um terminal e/ou uma interface ethernet para controle e configuração. Os equipamentos ditos gerenciáveis são os que apresentam uma interface de comunicação ethernet, possuindo sistemas embarcados que, além de efetuar as funções de controle do equipamento, implementam algum tipo de protocolo de gerenciamento, como por exemplo SNMP. O uso de protocolos deste tipo, visa construir um laço de comunicação com a central, de modo a permitir que os seus recursos sejam configurados de acordo com a demanda.

Sistemas embarcados são dispositivos eletrônicos que incorporam microprocessadores, sendo que um dos grandes objetivos da inclusão destes, é propiciar uma flexibilidade na remoção de bugs, adição de novas funcionalidades, e alterações em geral: basta alterar o software de controle (LEWIS, 2002). Entretanto, o custo de alteração de uma versão de software de um equipamento embarcado, uma vez em campo, pode ser suficientemente alto para inviabilizar a mesma, ainda que o equipamento permita efetuar remotamente a carga de software. Segundo Lewis (2002), a confiabilidade é um parâmetro muito importante, de modo que não se pode simplismente reiniciar um equipamento com problemas. Portanto, deve-se tomar extremo cuidado no desenvolvimento, de forma que não seja necessária uma alteração no software do equipamento uma vez em campo, pois esta deve exigir o reinício do equipamento e uma subsequente perda momentânea do serviço.

Como pode ser visto em Carro et al. (2009) devido ao baixo suporte e à menor automação, o software embarcado tende a apresentar mais erros do que o software tradicional. Levando ainda em consideração o fato de o custo de validação do software embarcado ser maior devido ao maior tempo necessário para a verificação do mesmo por conter componentes de hardware e software a serem validados, visualiza-se a importância de uma solução que reduza a quantidade de erros de software.

Além disso, o tempo de colocação do produto no mercado é um fator crucial para o sucesso de um projeto. Desta forma, o tempo de desenvolvimento deve ser reduzido ao máximo, e além do reuso de software, uma das técnicas que favorecem a redução no tempo de desenvolvimento é o uso de ferramentas de geração automática de código. Através destas, pode-se reduzir significativamente o tempo de desenvolvimento e a probabilidadade de falhas de software, uma vez validado o código gerado.

2 COMPONENTES E FERRAMENTAS

Nesta seção são brevemente descritos alguns dos componentes existentes em um software embarcado para gerência de equipamentos de telecomunicações, além de algumas ferramentas que se pretende utilizar na abordagem proposta.

2.1 CLI

Uma CLI (Command Line Interface) é um mecanismo bastante simples de interação com um equipamento, aonde os comandos devem ser digitados manualmente para a execução das tarefas desejadas. Consiste em um interpretador de linhas de comando, que frequentemente utiliza alguma ferramenta de análise léxica e sintática e um conjunto de funções associadas aos comandos interpretados.

É um componente comumente utilizado nos equipamentos de telecomunicação para prover uma gerência completa do produto: permite efetuar configurações em todas as propriedades configuráveis, bem como verificar o estado atual do equipamento e do(s) enlace(s).

2.2 SNMP

O SNMP (Simple Network Management Protocol) é um protocolo baseado em UDP utilizado para monitorar e gerenciar equipamentos de rede. Um agente SNMP é um componente de software responsável por receber requisições de um gerente através de uma mensagem no protocolo SNMP e processá-la.

Os dados que o agente SNMP disponibiliza são descritos em uma base de informações conhecida como MIB. O protocolo opera na camada de aplicação (camada 7 do modelo de referência OSI) e é descrito na RFC 1157, e os seus comandos mais básicos são do tipo get (GetRequest) e set (SetRequest).

Tipicamente o desenvolvedor não necessita implementar o protocolo SNMP do princípio. O código que implementa o protocolo é reutilizado, e acessado através de uma interface bem conhecida, como o AgentX, descrito na RFC 2741.

Este protocolo faz parte de equipamentos que necessitam gerência remota e, portanto, como será mostrado posteriormente, da maior parte dos equipamentos.

2.3 MIB

Uma MIB (Management Information Base) é uma base de dados de gerenciamento, contendo o nome dos parâmetros (associados a um identificador ou OID), o seu tipo ou sintaxe (Integer, DisplayString, etc.), uma definição escrita do parâmetro, o tipo de acesso (read-only, read-write, write-only ou not-accessible) e o seu status (mandatory,

optional ou obsolete), conforme descrição na RFC 1156. Esta coleção de informações é utilizada para acessar os dados de um equipamento através do protocolo SNMP.

Um equipamento gerenciado via SNMP deve responder a pelo menos uma MIB, porém tipicamente responde a diversas MIBs, aonde cada uma responde por uma funcionalidade do equipamento.

2.4 mib2c

O mib2c é uma ferramenta disponível no pacote Net-SNMP, projetada para gerar o esqueleto de um código em C que implementa a leitura e escrita dos objetos descritos na MIB, tomando como base um arquitvo MIB ou uma porção definida do mesmo. Além da leitura e escrita, o código gerado já apresenta o modelo de conexão com o AgentX. Existem diversas opções de configuração para o mib2c, e o esqueleto gerado deve ser preenchido pelo programador com as funções de acesso aos dados do equipamento.

2.5 SQLite

O SQLite é uma biblioteca compacta que implementa uma base de dados SQL e, de acordo com o site da própria ferramenta, é bastante utilizado em sistemas embarcados, apresentando um baixo consumo de memória e uma boa performance. Ainda de acordo com o site, a ferramenta é testada cuidadosamente antes de cada release e tem a reputação de ser muito confiável.

3 SOLUÇÕES EXISTENTES

Este capítulo trata das diversas ferramentas comerciais existentes que se propõem a auxiliar no desenvolvimento de soluções para o gerenciamento de equipamentos através do protocolo SNMP e de CLIs. Deseja-se com isto, demonstrar a necessidade da criação de uma nova ferramenta, que vá além da automação presente nas ferramentas atuais.

3.1 NuDesign Visual Embedded xAgenBuilder for C++

A ferramenta disponibilizada pela NuDesign Technologies Inc. consiste em um conjunto de aplicações que auxiliam os desenvolvedores na criação de Agentes SNMP Multiprotocolo, suportando o protocolo SNMP nas versões SNMP v1/v2c ou v1/v2c/v3, e com acesso aos objetos gerenciados através de CLI e opcionalmente HTTP. A ferramenta é disponibilizada para o uso em ambiente windows, apesar de suportar a geração de código para o sistema operacional da plataforma alvo. A relação de IDEs e ambientes suportados pela ferramenta podem ser vistos na Tabela 3.1.

Tabela 3.1: Relação entre IDEs e ambientes suportados pela ferramenta Visual Embedded xAgentBuilder for C++

Ambiente	**IDE**
MS Windows	Visual Studio (Versão 6 a 2005)
MS CE.NET	MS eMbedded Visual C++
PC Linux	GNU Make files
MontaVista	DevRocket IDE
SnapGear / uCLinux	GNU Make files
ENEA OSE (Epsilon, Delta e Softkernel)	CodeWarrior e Make files
QNX Neutrino	Momentics IDE
Windriver VxWorks	Tornado IDE

Fonte: dados obtidos em NuDesign

A solução da NuDesign fornece todo o código (bibliotecas para os protocolos SNMP, CLI, HTTP, além de utilitários) construído com base em bibliotecas padrão ANSI C++, para facilitar a portabilidade aos diversos ambientes embarcados. A ferramenta é capaz de utilizar uma MIB como entrada para gerar um projeto completo para um software de gerenciamento, fornecendo o esqueleto de código a ser preenchido com as rotinas de acesso aos dados.

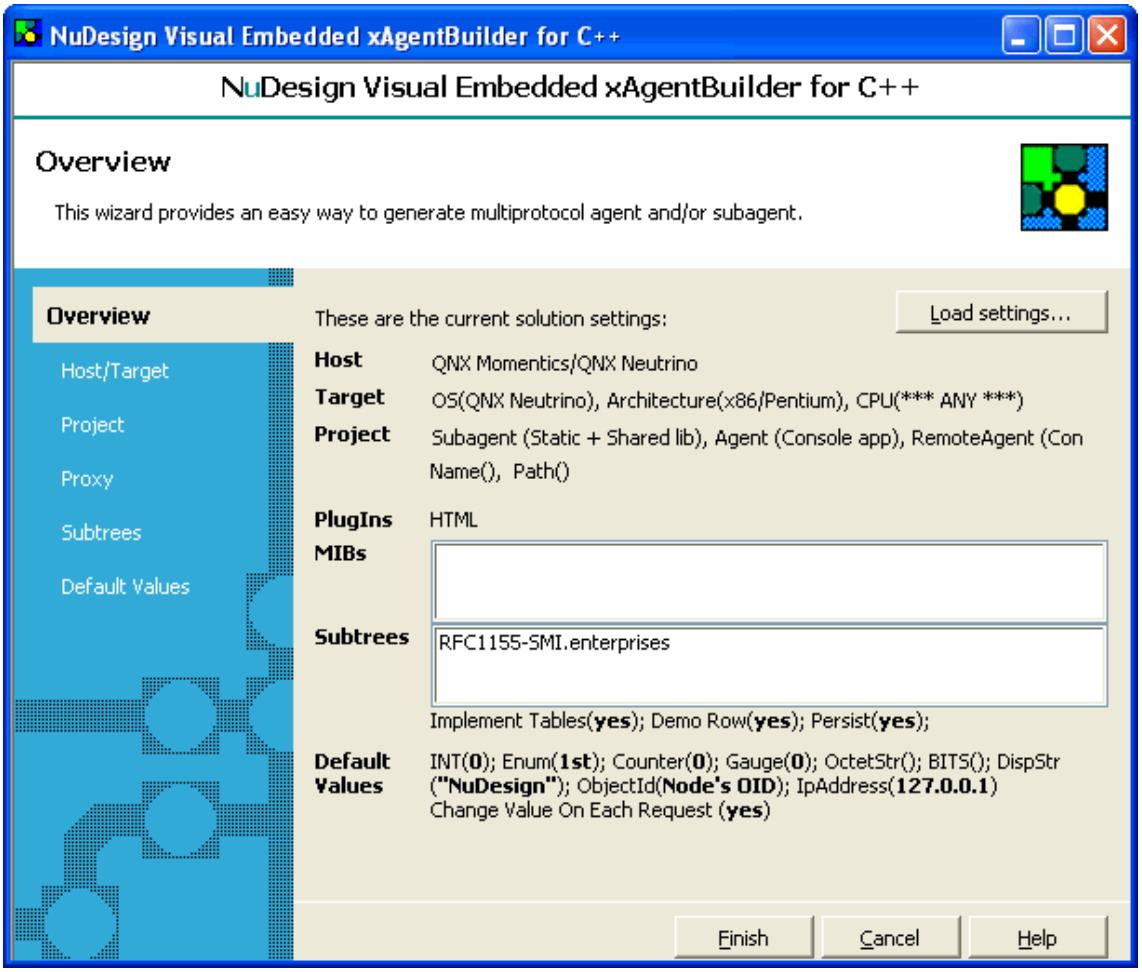

Figura 3.1: Visão geral da ferramenta Visual Embedded xAgentBuilder for C++ (NuDesign).

3.2 Web NMS SNMP Agent Toolkit C Edition 6.4.0

A Web NMS disponibiliza uma ferramenta de desenvolvimento para a criação de agentes SNMP (v1, v2c e v3) e CLI em ANSI C que visa auxiliar os engenheiros e desenvolvedores de aplicações de telecomunicações na criação de agentes SNMP multiprotocolo, CLIs e interfaces HTTP. A ferramenta é independente de plataforma e é baseada em Java, podendo ser utilizada em qualquer sistema com suporte a JVM. De acordo com a Web NMS, o agente gerado apresenta um overhead em performance no dispositivo alvo próximo a zero.

Esta ferramenta apresenta um processo de desenvolvimento em seis passos, como pode ser visto na Figura 3.2. Após definir uma MIB e gerar o código do agente com a ferramenta, é necessário preencher o código gerado com as rotinas de acesso aos dados, para então compilar, testar e implantar a solução.

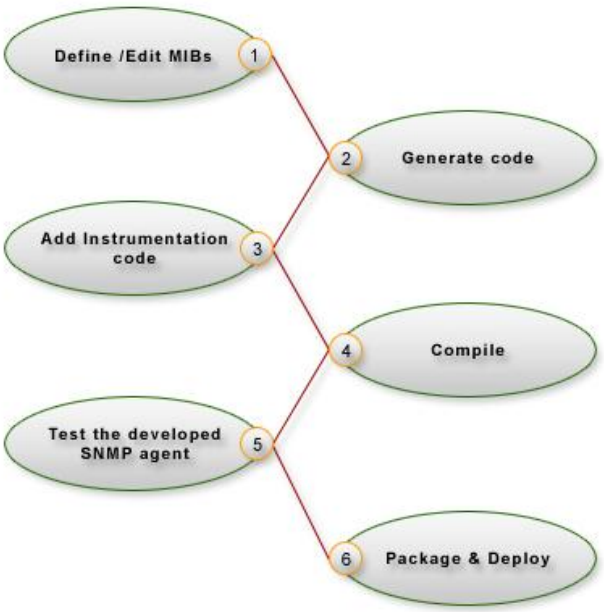

Figura 3.2: Modelo de desenvolvimento em seis passos (WebNMS)

O kit de desenvolvimento de agentes da Web NMS oferece uma plataforma modular para a construção de agentes, desde ferramentas de edição e compilação, até utilitários para facilitar a integração do agente gerado com aplicativos de gerenciamento existentes.

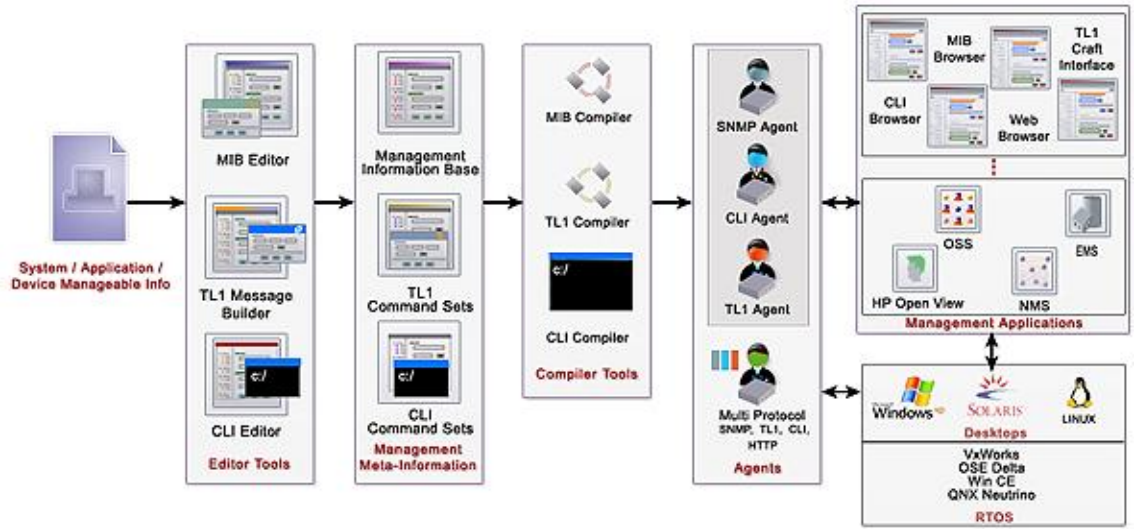

Figura 3.3: WebNMS SNMP Agent Toolkit C Edition (WebNMS)

3.3 EMANATE: SNMP Agent Development Framework

O EMANATE consiste em um agente SNMP extensível, que contém um conjunto de ferramentas de desenvolvimento para subagentes. Sua arquitetura é baseada em Mestre/Subagente, e permite que os subagentes sejam carregados dinamicamente. A ferramenta possui suporte ao protocolo SNMP v1, v2 e v3. O EMANATE (Enhanced

MANagement Agent Through Extensions ou Agente de Gerenciamento Aprimorado Através de Extensões) é desenvolvido pela SNMP Research Inc, e fornece uma API independente de plataforma, podendo ser utilizado em diversas plataformas.

Tabela 3.2: Relação entre RTOS, processador e sistema de desenvolvimento suportados pela ferramenta EMANATE

RTOS	**Reference Processor**	**Cross-Development OS**
Chorus ClassiX 3.1	Intel 80x86	Sun Solaris 2.x
HP/RT 3.01B	HP 9000/743	HP/UX 10.x
LynxOS 2.5.0	Intel 80x86	LynxOS 2.5.0
Nucleus NET 4.0*	AMD 186 (i80c186)	32-bit Windows
ENEA OSE 4.3	Soft Kernel	Solaris 2.x/32-bit Windows
OS/9 3.03*	Motorola 68030/68040	32-bit Windows
pSOS 2.2	Motorola 68030/68040	SunOS 4.1.x
VRTX	Motorola 68030/68040	SunOS 4.1.x/32-bit Windows
VxWorks 5.3/Tornado	Motorola 68030/68040	Solaris 2.x/32-bit Windows
Windows CE*	Intel 80x86	32-bit Windows

*Only available for EMANATE/Lite

Fonte: SNMP Research International, Inc.

A arquitetura da ferramenta permite que o desenvolvedor se concentre em desenvolver subagentes para a gerência do seu sistema, e os conecte a um agente principal responsável por gerenciar o protocolo SNMP. O processo de desenvolvimento é iniciado com uma MIB, a partir da qual o esqueleto do subagente é gerado, para então ser preenchido pelo desenvolvedor.

3.4 Outras ferramentas e considerações

Existem diversas outras ferramentas para auxiliar no desenvolvimento de agentes SNMP. A ModLink Networks, por exemplo, disponibiliza o Axonet SNMP Embedded Agent, capaz de gerar agentes SNMP v1, v1/v2c e v1/v2c/v3, e stubs para subagentes com base em arquivos MIB. Já a InterWorking Labs apresenta ferramentas para automação de testes, gerenciamento e monitoração de código SNMP, além de outras.

As ferramentas existentes para automatizar o desenvolvimento de software embarcado para a gerência de equipamentos de telecomunicações, apesar de reduzirem bastante o trabalho e agilizarem o processo de desenvolvimento, ainda deixam o programador com a trabalhosa tarefa de preencher manualmente centenas de linhas de

código. Sendo este trabalho muitas vezes puramente mecânico, ele é portanto ainda passível de ser automatizado.

O motivo desta baixa automatização no processo de desenvolvimento deve-se principalmente ao ponto comum a ambas as interfaces externas do equipamento: a interface com o hardware. Como pode ser visto na Figura 3.4, a interface com o hardware é a maneira com a qual o módulo CLI e o agente SNMP podem interagir entre sí. Entretanto, devido a especifidades do hardware de cada equipamento, esta interface não é padronizada, impossibilitando o aumento na automatização.

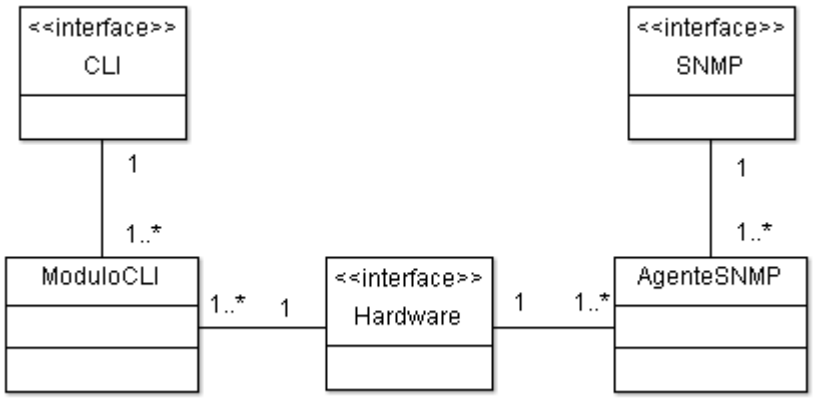

Figura 3.4: Arquitetura típica de um software embarcado para a gerência de equipamentos de telecomunicações.

Com a adoção de uma arquitetura mais apropriada, pode-se eliminar a necessidade de alocar um projetista específicamente para o desenvolvimento da gerência do produto via SNMP (como será visto posteriormente), e até mesmo para o desenvolvimento da interface de comunicação via terminal (Interface de Linha de Comando, ou CLI). Desta forma, estes recursos podem ser empregados na pesquisa e no desenvolvimento de novas funcionalidades, além de otimizações para o produto.

4 ABORDAGEM PROPOSTA

Como pode ser visto em Sridhar (2003), a ênfase na redução do custo de Pesquisa e Desenvolvimento de embarcados acarretou a adoção não só de hardware, mas também de componentes de software de terceiros na composição dos produtos, levando ao surgimento de um desenvolvedor que deve compreender a construção e a integração de uma vasta gama de componentes reusáveis de software.

Comumente, um desenvolvedor é alocado para realizar a implantação de um sistema operacional sobre uma plataforma de hardware e, além de gerenciar o equipamento em um nível mais baixo, prover as interfaces necessárias para a comunicação com o equipamento às camadas de gerenciamento externas. Um segundo desenvolvedor deve tomar conhecimento das especificidades das interfaces de comunicação para este produto específico (já que estas não são padronizadas), e prover uma CLI (geralmente padronizada) ao usuário. Um terceiro desenvolvedor deve também conhecer as interfaces de comunicação do produto e disponibilizar o acesso às configurações através de uma interface de gerenciamento SNMP.

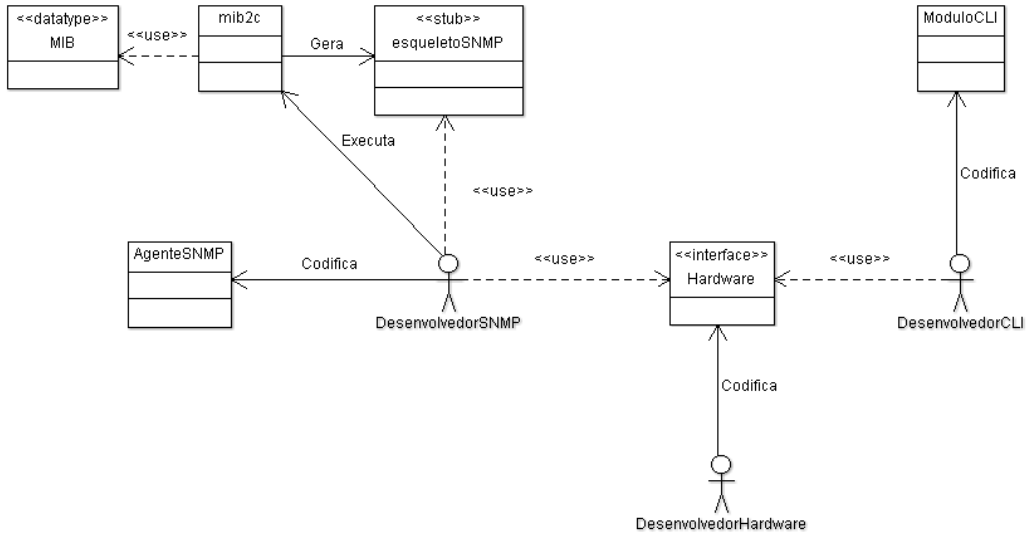

Figura 4.1:Estrutura de desenvolvimento típica.

Como pode ser visto na Figura 4.1, os três desenvolvedores estão envolvidos diretamente com a codificação de seus módulos, e há pouca automatização nesta tarefa.

Devido às especificidades da implementação de cada produto, as tarefas do desenvolvedor do agente SNMP e do desenvolvedor da CLI tornam-se bastante passíveis de erro, devido ao gargalo existente na interface do hardware. Desta forma,

esta interface entre o hardware e o agente/CLI é um ótimo candidato a uma padronização e, consequentemente, a uma automatização na geração do seu código.

Grande parte dos equipamentos de telecomunicação atuais conta com um banco de dados integrado. Este banco de dados é geralmente utilizado para permitir que sejam realizados processos de auditoria nesses equipamentos. Outro uso comum do banco de dados é no armazenamento de um histórico de eventos: no caso de haver uma interrupção na comunicação entre o equipamento e o sistema de gerência, deve ser possível recuperar a informação referente aos estados pelos quais este equipamento passou durante esse período sem comunicação. Havendo a existência de um banco de dados no equipamento, pode-se utilizá-lo para armazenar as informações de gerenciamento do mesmo. Sendo a interface de comunicação com o banco de dados bem conhecida, ele se torna uma opção viável para intermediar a comunicação entre o hardware e o software de gerenciamento.

Com o uso de um banco de dados centralizado para manter as informações de configuração e estado do hardware pode-se criar uma ferramenta que realize um grau a mais de automação na geração de código a partir da definição de uma MIB.

As informações que antes seriam buscadas diretamente do equipamento agora passam a ser buscadas de um banco de dados. No momento da configuração do equipamento o processo se dá de modo análogo: as informações são escritas no banco de dados. A interface de hardware deve então interagir com o banco de dados de modo a publicar eventuais alterações em seu estado no banco. No momento da programação do equipamento, um processo integrado à interface de hardware deve ser notificado, de modo a buscar os parâmetros do banco de dados e configurá-los no equipamento.

Desta forma, o desenvolvedor da interface de hardware deve se apropriar destes conceitos de modo a utilizar o banco de dados como uma interface para a programação do equipamento e publicação do estado atual. Ao invés de publicar uma função que permita buscar os dados das variáveis de estado e configuração, qualquer mudança que ocorreria em uma destas variáveis deverá ocorrer nas variáveis do banco de dados (através de uma rotina de escrita em banco). Da mesma forma, ao invés de publicar uma função que recebe os parâmetros a serem configurados no equipamento, deve-se fornecer um método que ao ser sinalizado busque as informações no banco de dados e as efetue no equipamento.

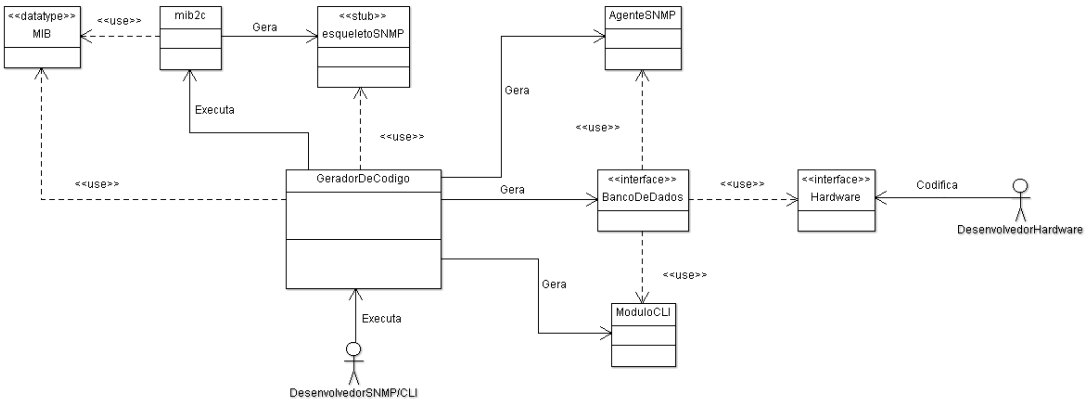

Figura 4.2 Estrutura de desenvolvimento proposta.

Parte-se de uma descrição das interfaces de gerenciamento a partir de um arquivo MIB. A partir deste arquivo, é possível criar as tabelas de um banco de dados, mantendo-se tanto os OIDs quanto os nomes dos atributos como possíveis chaves para os campos. Utiliza-se a ferramenta mib2c para se obter um esqueleto do agente. Este esqueleto pode então ter os seus métodos de leitura e configuração preenchidos com chamadas ao banco de dados – cujas tabelas foram geradas de forma automática e, portanto são bem conhecidas – utilizando como chave o próprio OID das variáveis da MIB. A partir do banco de dados, pode-se derivar uma CLI; esta pode utilizar os nomes dos atributos MIB (após possivelmente um *parse* de modo a hierarquizar os parâmetros e facilitar o uso) tanto como nome para as funções de configuração quanto como chave (após a reconstrução hierárquica destes) para acesso ao banco de dados.

Ao final do ciclo obtém-se três artefatos: o agente SNMP finalizado, já com a ligação com a interface de hardware (através da base de dados) e com o AgentX (através do esqueleto SNMP tomado como base); o módulo CLI finalizado, com conexão com a interface de hardware obtida de modo análogo ao do agente SNMP, e com uma estrutura de acesso aos comandos previamente definida; e uma base de dados, que serve como interface para o desenvolvedor da interface do hardware publicar os seus métodos (leitura e escrita).

A proposta então consiste em dois passos. Inicialmente deve-se alterar a arquitetura do software de gerência para incluir um banco de dados que centralize a informação do equipamento e padronize o acesso a ela, como pode ser visto na Figura 4.3.

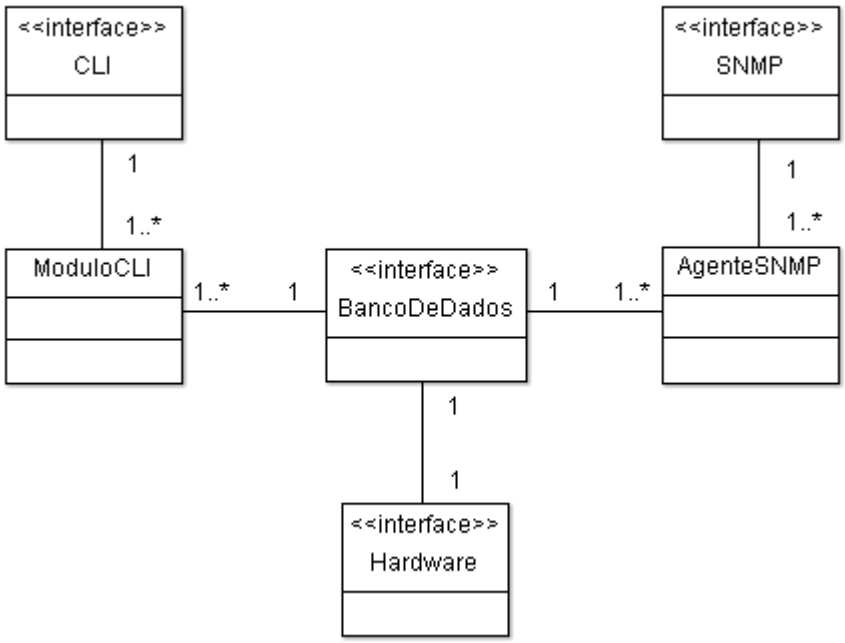

Figura 4.3: Arquitetura proposta.

Como visto na Figura 4.2, deve-se então criar uma ferramenta que permita a geração automática do código e criação das tabelas do banco de dados. A ferramenta deve também produzir os três artefatos descritos anteriormente, aumentando assim a automação, reduzindo a possibilidade de erros e acelerando o processo de desenvolvimento. Desta forma, deve haver um forte impacto no tempo de colocação do produto no mercado.

5 IMPLEMENTAÇÃO

A implementação da solução proposta foi feita utilizando a linguagem Python. Sendo uma linguagem multiparadigma e de alto nível de abstração, ela se presta muito bem às tarefas de automação, e as facilidades na manipulação de arquivos e strings simplificam bastante a tarefa.

O banco de dados utilizado na aplicação é o SQLite. A escolha deste banco se deve ao fato de já estar disponível no equipamento. O SQLite segue a sintaxe SQL e seu código é de domínio público, além de ser leve (ocupa atualmente cerca de 150KB com a remoção de algumas funcionalidades não padronizadas de acordo com a sintaxe SQL).

A entrada para o gerador de código é um arquivo MIB. Este arquivo é então submetido à ferramenta mib2c, de modo que seja gerado o esqueleto básico do subagente, contendo os stubs das funções de comunicação com o agente, bem como as entradas para as funções de leitura e escrita das variáveis SNMP. O OID de cada variável tratada pelo subagente é atrelado a uma variável local, e esta é utilizada na determinação das respostas aos comandos GetRequest e SetRequest enviados ao agente.

Após esta etapa, o gerador passa a tratar o arquivo gerado pela ferramenta mib2c, de modo a incluir as bibliotecas necessárias para o acesso ao banco de dados, bem como a criação e inicialização do mesmo. O fluxo de exeução segue então para a criação das tabelas no banco de dados, como pode ser visto no diagrama simplificado da máquina de estados da Figura 5.1. A partir daí são inseridas as linhas na tabela, concomitantemente com a inserção das chamadas ao banco de dados através dos métodos de leitura no código do subagente, além da criação das entradas para as rotinas de leitura na CLI.

Finalmente, são tratados os métodos de escrita, etapa na qual o gerador cria as entradas na CLI para as rotinas de escrita e insere as chamadas às funções de escrita no banco de dados. Nesta etapa também é verificada a conformidade das funções de escrita nas devidas tabelas do banco de dados, gerando uma advertência caso alguma das variáveis declaradas inicialmente na MIB como sendo de leitura/escrita não possua uma entrada correspondente no banco.

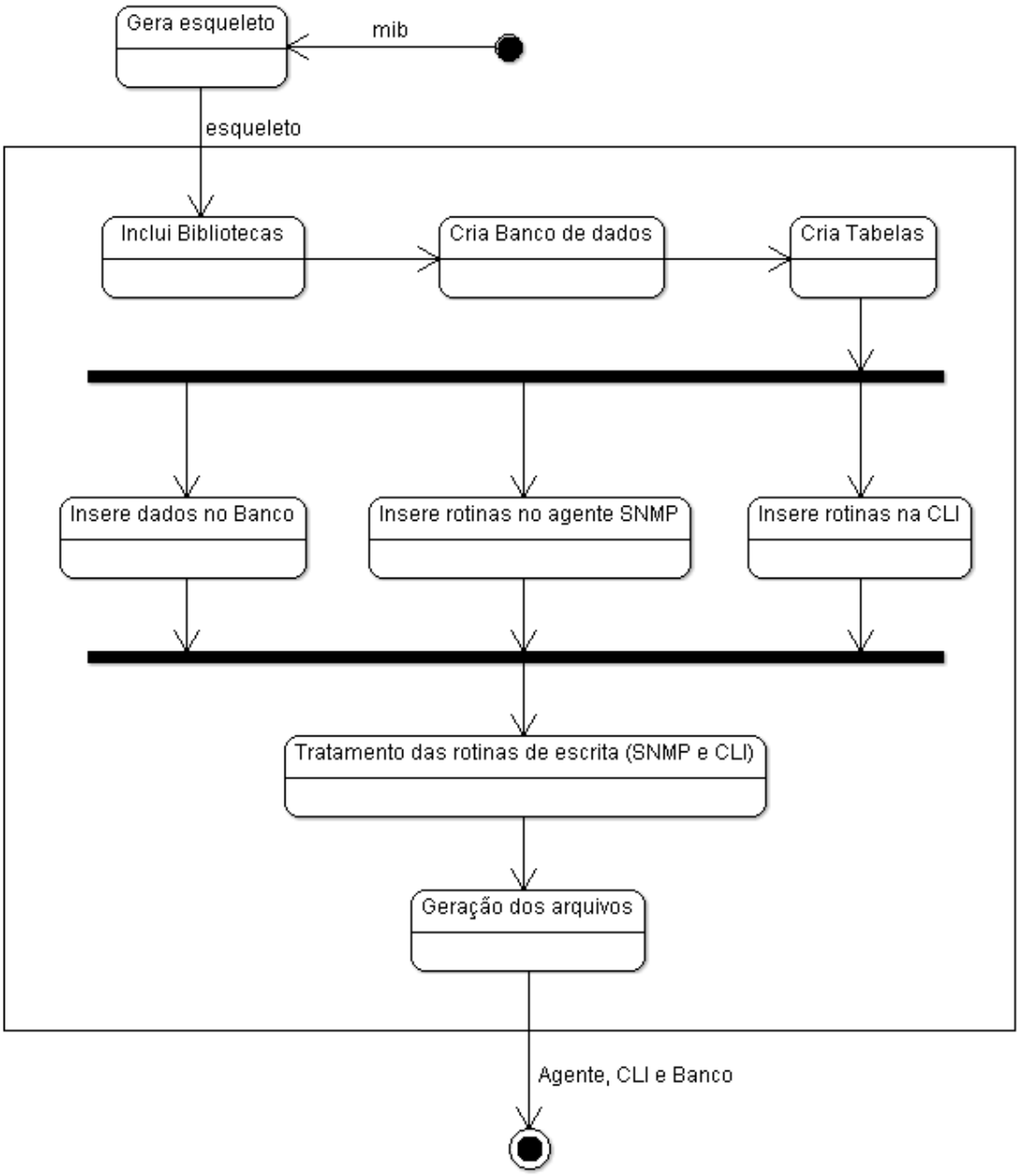

Figura 5.1: Diagrama simplificado da máquina de estados do gerador.

Ao final do processo são gerados uma série de arquivos: dois relacionados ao agente, sendo um arquivo de implementação do agente em C e o outro contendo os headers; além de arquivos correspondentes à CLI, que consistem em arquivos para a ferramenta de análise léxico (lex) e para a ferramenta de análise sintática (yacc); por fim também é gerado o arquivo correspondente ao banco de dados. O gerador pode ser facilmente adaptado para ao invés de gerar uma CLI, gerar um arquivo de configuração para uma CLI proprietária, e esta adaptação foi testada com êxito.

Após esta etapa de geração de código, basta ao programador realizar a ligação do subagente com o daemon SNMP e, de modo análogo, garantir a execução da CLI.

Graças às facilidades providas pelo uso da linguagem Python, o código do gerador possui apenas 361 linhas, incluindo comentários. Ao analisar o tamanho do código do gerador, deve-se levar em conta o fato de ele conter todo o código que será inserido nos arquivos fonte do agente SNMP e da CLI, além de todos os comandos de criação de tabelas e inserção de dados no banco.

Tomando-se como exemplo o uso do gerador aplicado à MIB descrita na RFC 5066, são geradas 3647 linhas de código para o agente SNMP, 96 linhas de código para o analisador léxico da CLI e 973 para o analisador sintático da CLI, além de um banco de dados de 96KB.

Em comparação com as ferramentas utilizadas antes da criação do gerador, o esqueleto do agente SNMP gerado pela ferramenta mib2c possuía 3007 linhas. São portanto acrescentadas 640 linhas de código pelo gerador, ou 21% de código. O código gerado para o agente SNMP pela ferramenta suprime comentários desnecessários gerados pela ferramenta mib2c, caso contrário a diferenca seria ainda maior. Conforme descrito anteriormente, não havia automação na geração de código para a CLI, portanto são geradas 1069 linhas para a cli contra 0 linhas geradas anteriormente. É conveniente ressaltar que o número de linhas geradas varia diretamente com a quantidade de parâmetros existentes na MIB, bem como o tipo de acesso permitido a esses dados (somente leitura ou leitura e escrita).

Considerando o exemplo descrito anteriormente, as 361 linhas de código do gerador foram capazes de gerar 4716 linhas de código, além de um banco de dados. Isso corresponde (não levando em consideração o banco de dados) a 13 linhas geradas para cada linha de código do gerador.

O desenvolvimento foi feito sobre uma plataforma linux com kernel na versão 2.6, e o código gerado em C foi compilado para plataformas linux com kernel na versão 2.6 para processadores ARM e PowerPC. O teste e validação do código gerado foi feito com o uso de ferramentas de teste que exercitam as interfaces por meio de testes exaustivos.

6 CONCLUSÃO E RESULTADOS

Com o uso de um banco de dados leve pode-se simplificar um ponto muito crítico no desenvolvimento de sistemas de gerenciamento de produtos de telecomunicações: a interface entre o software que gerencia os dados obtidos do equipamento e os agentes externos que provêem o acesso a estes. Este mesmo passo leva à possibilidade de uma automatização na geração de código, permitindo uma redução no tempo de desenvolvimento e na necessidade de manutenção do software.

Além disso, a desvinculação do código gerado (tando para o agente SNMP quanto para a CLI) da plataforma de hardware – através do uso do banco de dados como interface – acaba por gerar componentes reutilizáveis de software. Uma vez que muitas das funcionalidades configuradas através de uma MIB estão presentes em diversos equipamentos semelhantes, essa possibilidade de reutilizar componentes acaba por impactar em projetos futuros, não só no tempo de desenvolvimento, mas principalmente em testes.

O desenvolvimento do agente é realizado em paralelo com outras tarefas, sendo responsável por uma pequena parcela do projeto, estimada em 5%. Entretanto com o uso da ferramenta de geração de código criada, o tempo de desenvolvimento em relação ao agente SNMP foi reduzido em mais de 90%. Com isso, libera-se um programador para trabalhar em outra parte do projeto.

Com o uso da ferramenta estima-se uma redução de mais de 10% no tempo de desenvolvimento dos equipamentos de telecomunicações, graças à possibilidade de o desenvolvedor do agente SNMP e da CLI se dedicar a outras tarefas dentro do projeto. Esta redução no tempo de desenvolvimento permite à empresa colocar o produto mais cedo no mercado, evitando a perda no lucro em função a um atraso no lançamento.

REFERÊNCIAS

BEILI, E. **Ethernet in the First Mile Copper (EFMCu) Interfaces MIB**: RFC 5066. [S.l.]: Internet Engineering Task Force, Network Working Group, 2007.

CARRO, Luigi; WAGNER, Flávio Rech. *Metodologias e técnicas de engenharia de software para sistemas embarcados*. In CARVALHO, André C. P. L. F. de; KOWALTOWSKI, Tomasz (organizadores). **Atualizações em informática 2009**. Rio de Janeiro: PUC-Rio, 2009.

CASE, J. et al. **A simple Network Management Protocol (SNMP)**: RFC 1157. [S.l.]: Internet Engineering Task Force, Network Working Group, 1990.

DANIELE, M. et al. **Agente Extensibility (AgentX) Protocol Version 1**: RFC 2741. [S.l.]: Internet Engineering Task Force, Network Working Group, 2000.

LEWIS, Daniel W. **Fundamentals of embedded software**: where C and Assembly meet. Upper Saddle River, New Jersey: Prentice Hall, 2002.

MCCLOGHRIE, K.; ROSE, M. **Management Information Base for Network Management of TCP/IP-Based internets**: RFC 1156. [S.l.]: Internet Engineering Task Force, Network Working Group, 1990.

Net-SNMP. **mib2c**. Disponível em http://www.net-snmp.org. Acesso em 10/11/2010.

NuDesign Technologies Inc. **Visual Embedded xAgentBuilder for C++**. Disponível em http://www.ndt-inc.com/SNMP/StudioC4RTOS.html. Acesso em 10/11/2010.

SNMP Research International, Inc. EMANATE – SNMP Agent Development Framework. Disponível em http://www.aethis.com/solutions/snmp_research/emanate.html. Acesso em 10/11/2010.

SQLite. Disponível em http://www.sqlite.org. Acesso em 10/11/2010.

SRIDHAR, T. **Designing communications software**. San Francisco: CMP Books, 2003.

WebNMS. SNMP **Agent Toolkit C Edition 6.4.0**. Disponível em http://www.webnms.com/cagent/index.html. Acesso em 10/11/2020.

APÊNDICE <CÓDIGO DO GERADOR>

```
import sqlite3
import os


mibFile=raw_input("Arquivo MIB:")
mibNode=raw_input("Nodo:")
os.system("MIBS=\"" + mibFile + "\" mib2c -c mib2c.old-api.conf " + mibNode)

#Arquivo de entrada
mibBasedFile = open(mibNode+".c",'r')
#Arquivo de saida temporario
outputFile = open('output.c','w')
database = ""
mibBasedText = mibBasedFile.read()
mibBasedTextLines = mibBasedText.splitlines()
mode="Header"
subM = 0
count = 0
list = []
writeCount = 0


#Variaveis de controle da CLI
#contador de tabelas/variaveis
tabS=0
tabList=[]
varS=0
varList=[]
#scanner
scanCmds="get                                                   return GET;\nset
                        return SET;\n"
#buffer de tokens
buffTokens="GET SET TAB VAR NUMBER "
#parser dos comandos get
parsGetCmds="           switch($2)\n          {\n"
#parser dos comandos set
parsSetCmds="           switch($2)\n          {\n"

#Verifica cada linha do arquivo
for line in mibBasedTextLines:
        #Copia os headers
        if mode == "Header":
                if "#include" not in line:
                        outputFile.write(line+"\n")
```

```
                else:
                        mode="Include"
                        outputFile.write(line+"\n")

        #Copia includes e insere libs e variaveis necessarias
        elif mode == "Include":
                if "#include" in line:
                        outputFile.write(line+"\n")
                else:
                        mode="Define"
                        #Insere as libs
                        outputFile.write("#include <string.h>"+"\n")
                        outputFile.write("#include <stdio.h>"+"\n")
                        outputFile.write("#include <stdlib.h>"+"\n")
                        outputFile.write("#include <sqlite3.h>"+"\n")

                        #Constantes
                        outputFile.write("\n"+"/*"+"\n")
                        outputFile.write(" * Constants"+"\n")
                        outputFile.write(" */"+"\n")
                        outputFile.write("\n"+"#define TABLE_SIZE 1"+"\n")

                        #Declaracao do banco de dados
                        outputFile.write("\n"+"/*"+"\n")
                        outputFile.write(" * Database name"+"\n")
                        outputFile.write(" */"+"\n")
                        outputFile.write("\n"+"sqlite3 *db;"+"\n")

                        #Funcao de callback (para os comandos SQL)
                        outputFile.write("\n"+"/*"+"\n")
                        outputFile.write(" * Callback function for SQL commands"+"\n")
                        outputFile.write(" */"+"\n")
                        outputFile.write("\n"+"char VAR[16];"+"\n")
                        outputFile.write("static int callback(void *NotUsed, int argc,
char **argv, char **azColName)"+"\n")
                        outputFile.write("{"+"\n")
                        outputFile.write("    int i;"+"\n")
                        outputFile.write("    NotUsed=0;"+"\n")
                        outputFile.write("    strcpy(VAR,argv[0]);"+"\n")
                        outputFile.write("    return 0;"+"\n")
                        outputFile.write("}"+"\n")

                        outputFile.write(line+"\n")

        #Copia os defines dos numeros magicos
        elif mode == "Define":
                if "init_" not in line:
                        outputFile.write(line+"\n")

                else:
                        mode="Init"
                        outputFile.write(line+"\n")
                        database=line.replace('init_','').split('(')[0]+".db"

        #Insere o codigo de inicializacao do banco de dados
        elif mode == "Init":
                if "{" in line:
                        outputFile.write(line+"\n")
```

```
                        outputFile.write("//sqlite3 *db;"+"\n")
                        outputFile.write("char *zErrMsg = 0;"+"\n")
                        outputFile.write("int rc;"+"\n")

                elif "/* place any other initialization junk you need here */" in line:
                        outputFile.write(line+"\n")
                        outputFile.write("    rc = sqlite3_open(\""+database+"\",
&db);"+"\n")
                        outputFile.write("    if( rc )"+"\n")
                        outputFile.write("    {"+"\n")
                        outputFile.write("       fprintf(stderr, \"Can't open database:
%s\\n\", sqlite3_errmsg(db));"+"\n")
                        outputFile.write("       sqlite3_close(db);"+"\n")
                        outputFile.write("       exit(1);"+"\n")
                        outputFile.write("    }"+"\n")
                        #outputFile.write(""+"\n")

                elif "}" in line:
                        mode="CreateDB"
                        outputFile.write(line+"\n")

                else:
                        outputFile.write(line+"\n")

        #Encontra o nome e cria o banco de dados
        elif mode == "CreateDB":
                if "var_" in line and "(" in line and "* " not in line:
                        mode = "SkipOrInsert"
                        con = sqlite3.connect(database)
                        con.isolation_level = None
                        cur = con.cursor()
                        statement=""
                        outputFile.write(line+"\n")

                else:
                        outputFile.write(line+"\n")

        #Insere as linhas nas tabelas ou copia conteudo
        elif mode == "SkipOrInsert":
                if "case" in line:
                        #Insere as linhas nas tabelas
                        statement = "INSERT INTO " + thisTable + " VALUES (\"" +
line.replace(':','').split()[1] + "\", NULL);"

                        if sqlite3.complete_statement(statement):
                                try:
                                        statement = statement.strip()
                                        cur.execute(statement)
                                except sqlite3.Error, e:
                                        print "An error occurred:", e.args[0]
                                statement = ""

                        thisEntry = line.replace(':','').split()[1]
                        #Atualiza o codigo da CLI
                        varS+=1
                        varList.append(thisEntry)
                        scanCmds+=thisEntry+"                   yylval="+repr(varS)+";
return VAR;\n"
```

```
                        parsGetCmds+="                  case "+repr(varS)+":\n"
                        parsGetCmds+="                     if (sqlite3_exec(db,\"SELECT
val FROM "+ tabList[tabS-1] +" WHERE var = \\\""+ varList[varS-1] +"\\\";\", callback, 0,
&zErrMsg)!=SQLITE_OK)"+"\n"
                        parsGetCmds+="                     {"+"\n"
                        parsGetCmds+="                        fprintf(stderr, \"SQL error:
%s\\n\", zErrMsg);"+"\n"
                        parsGetCmds+="                     }"+"\n"
                        parsGetCmds+="                     break;\n"

                elif "*write_method = write_" in line:
                        list.append(thisEntry)
                        writeCount=1
                        parsSetCmds+="                  case "+repr(varS)+":\n"
                        parsSetCmds+="                     sprintf(buf,\"UPDATE " +
tabList[tabS-1] + " SET val = \\\"%d\\\" WHERE var = \\\"" + varList[varS-1] + "\\\";\",
$4);\n"
                        parsSetCmds+="                     if (sqlite3_exec(db, buf, 0, 0,
&zErrMsg)!=SQLITE_OK)"+"\n"
                        parsSetCmds+="                     {"+"\n"
                        parsSetCmds+="                        fprintf(stderr, \"SQL error:
%s\\n\", zErrMsg);"+"\n"
                        parsSetCmds+="                     }"+"\n"
                        parsSetCmds+="                     break;\n"

                if "{" in line and subM == 0:
                        subM = 1;

                count = count + line.count("{") - line.count("}")

                if count == 0 and subM == 1:
                        mode = "CreateTable"
                        subM = 0
                        count = 0

                if "VAR = VALUE;" in line:
                        outputFile.write("       if (sqlite3_exec(db,\"SELECT val FROM
"+thisTable +" WHERE var = \\\""+thisEntry+"\\\";\", callback, 0,
&zErrMsg)!=SQLITE_OK)"+"\n")
                        outputFile.write("       {"+"\n")
                        outputFile.write("          fprintf(stderr, \"SQL error: %s\\n\",
zErrMsg);"+"\n")
                        outputFile.write("       }"+"\n")

                elif "/* variables we may use later */" in line:
                        outputFile.write(line+"\n")
                        outputFile.write("    char *zErrMsg = 0;\n")

                else:
                        outputFile.write(line+"\n")

        #Cria cada tabela no banco de dados
        elif mode == "CreateTable":
                if "var_" in line and "(" in line and "* " not in line:
                        statement = "CREATE TABLE " +
line.replace('var_','').split('(')[0] + " (var VARCHAR(64), val VARCHAR(16));"
                        if sqlite3.complete_statement(statement):
                                try:
```

```
                                        statement = statement.strip()
                                        cur.execute(statement)
                                except sqlite3.Error, e:
                                        print "An error occurred:", e.args[0]
                                statement = ""

                        thisTable = line.replace('var_','').split('(')[0]

                        if writeCount == 0 and len(list) > 0:
                                list.pop()

                        list.append(thisTable)
                        writeCount=0
                        outputFile.write(line+"\n")

                        #Atualiza o codigo da CLI
                        tabS+=1;
                        tabList.append(thisTable)
                        scanCmds+=thisTable+"                   yylval="+repr(tabS)+";
return TAB;\n"

                        if tabS > 1:
                                parsGetCmds+="                  default:\n
printf(\"ERROR: No such variable in this table\\n\");\n                 break;\n"
                                parsGetCmds+="              }\n             break;\n\n"
                                parsSetCmds+="                  default:\n
printf(\"ERROR: No such variable in this table\\n\");\n                 break;\n"
                                parsSetCmds+="              }\n             break;\n\n"
                        parsGetCmds+="          case "+repr(tabS)+":\n
switch($3)\n            {\n"
                        parsSetCmds+="          case "+repr(tabS)+":\n
switch($3)\n            {\n"
                        mode = "SkipOrInsert"

                elif "write_" in line and "(" in line:
                        mode = "WriteMethods"
                        thisMethod = line.replace('write_','').replace('(int    action,','')
                        #print thisMethod
                        outputFile.write(line+"\n")

                else:
                        outputFile.write(line+"\n")

        #Trata os metodos de escrita
        elif mode == "WriteMethods":
                if "long value;" in line:
                        varType="%ld"
                        outputFile.write("    char *zErrMsg = 0;"+"\n")
                        outputFile.write("    char buf[200];"+"\n")
                        outputFile.write(line+"\n")

                elif "int value;" in line:
                        varType="%d"
                        outputFile.write("    char *zErrMsg = 0;"+"\n")
                        outputFile.write("    char buf[200];"+"\n")
                        outputFile.write(line+"\n")

                elif "char value;" in line:
```

```
                        varType="%s"
                        outputFile.write("    char *zErrMsg = 0;"+"\n")
                        outputFile.write("    char buf[200];"+"\n")
                        outputFile.write(line+"\n")

                elif "case COMMIT:" in line:
                        mode = "WriteCommit"
                        outputFile.write(line+"\n")

                else:
                        outputFile.write(line+"\n")

        #Insere as chamadas de escrita no banco de dados
        elif mode == "WriteCommit":
                if "break;" in line:
                        thisObject = list.pop(0)

                        if thisMethod.upper() not in thisObject:
                                thisWriteTable = thisObject
                                thisObject = list.pop(0)

                        outputFile.write("        sprintf(buf,\"UPDATE " + thisWriteTable
+ " SET val = \\\"" + varType + "\\\" WHERE var = \\\"" + thisObject.upper() + "\\\";\",
value);\n")
                        outputFile.write("        if (sqlite3_exec(db, buf, 0, 0,
&zErrMsg)!=SQLITE_OK)"+"\n")
                        outputFile.write("        {"+"\n")
                        outputFile.write("            fprintf(stderr, \"SQL error: %s\\n\",
zErrMsg);"+"\n")
                        outputFile.write("        }"+"\n")
                        outputFile.write(line+"\n")
                        mode = "FindNextMethod"

                else:
                        outputFile.write(line+"\n")

        #Busca o proximo metodo de escrita
        elif mode == "FindNextMethod":
                if "write_" in line and "(" in line:
                        mode = "WriteMethods"
                        thisMethod = line.replace('write_','').replace('(int     action,','')
                        outputFile.write(line+"\n")

                else:
                        outputFile.write(line+"\n")

        #Copia
        elif mode == "JustCopy":
                outputFile.write(line+"\n")

#Gera o scanner.l
scannerFile = open('/home/will/Documents/tc/scanner.l','w')
scannerFile.write("%{\n#include <stdio.h>\n#include \"y.tab.h\"\n%}\n%%\n[0-
9]+\t\t\t\tyylval=atoi(yytext);return NUMBER;\n"+scanCmds+"\n\\n\n[ \\t]+\n%%\n")
scannerFile.close()
```

```
#Gera o parser.y
parsGetCmds+="                }\n              break;\n\n            default:\n
printf(\"ERROR: No such table\\n\");\n              break;\n          }\n"
parsSetCmds+="                }\n              break;\n\n            default:\n
printf(\"ERROR: No such table\\n\");\n              break;\n          }\n"
parserFile = open('/home/will/Documents/tc/parser.y','w')
parserFile.write("%{\n#include <stdio.h>\n#include <string.h>\n#include
<stdlib.h>\n#include <sqlite3.h>\n\nsqlite3 *db;\n\n")
parserFile.write("char *zErrMsg = 0;\n")
parserFile.write("char buf[200];\n\n")
parserFile.write("static int callback(void *NotUsed, int argc, char **argv, char
**azColName)\n")
parserFile.write("{\n")
parserFile.write("    int i;\n")
parserFile.write("    for(i=0; i<argc; i++)\n")
parserFile.write("    {\n")
parserFile.write("        printf(\"%s\\n\", argv[i]);\n")
parserFile.write("    }\n")
parserFile.write("    printf(\"\\n\");\n")
parserFile.write("    return 0;\n")
parserFile.write("}\n\n")
parserFile.write("void yyerror(const char *str)\n")
parserFile.write("{\n")
parserFile.write("    fprintf(stderr,\"error: %s\\n\",str);\n")
parserFile.write("}\n\n")
parserFile.write("int yywrap()\n")
parserFile.write("{\n")
parserFile.write("    sqlite3_close(db);\n")
parserFile.write("    return 1;\n")
parserFile.write("}\n\n")
parserFile.write("main()\n")
parserFile.write("{\n")
parserFile.write("    int rc;\n")
parserFile.write("    rc = sqlite3_open(\""+database+"\", &db);\n")
parserFile.write("    if( rc )\n")
parserFile.write("    {\n")
parserFile.write("        fprintf(stderr, \"Can't open database: %s\\n\",
sqlite3_errmsg(db));\n")
parserFile.write("        sqlite3_close(db);\n")
parserFile.write("        exit(1);\n")
parserFile.write("    }\n")
parserFile.write("    yyparse();\n")
parserFile.write("}\n\n")
parserFile.write("%}\n\n")
parserFile.write("%token "+buffTokens+"\n\n")
parserFile.write("%%\n\n")
parserFile.write("commands:\n")
parserFile.write("        | commands command\n")
parserFile.write("        ;\n\n")
parserFile.write("command:\n")
parserFile.write("        GET TAB VAR\n        {\n"+parsGetCmds+"        }\n")
parserFile.write("       |SET TAB VAR NUMBER\n        {\n"+parsSetCmds+"        }\n
;\n")


con.close()
mibBasedFile.close()
outputFile.close()
```