Especificação Formal de Entidades SNMPv3 Usando Semântica de Ações

Diógenes Cogo Furlan[1], Elias Procópio Duarte Jr.[2], and Martin Musicante[2]

[1] Universidade Tuiuti do Paraná, Fac. Ciências Exatas Tecnologicas, Curitiba, PR
diogenes.furlan@utp.br
[2] Universidade Federal do Paraná, Depto. Informática, Curitiba, PR
{elias,mam}@inf.ufpr.br

Resumo A arquitetura padrão para gerência de redes da Internet é o SNMPv3 (*Simple Network Management Protocol version 3*). Um sistema baseado em SNMPv3 é composto por entidades de gerência (agentes e gerentes) que se comunicam usando o protocolo de gerência. As entidades são compostas por um engenho, além de aplicações padrão. As entidades são descritas informalmente em documentos do IETF. Este trabalho apresenta um estudo de caso de aplicação do formalismo denominado Semântica de Ações para gerar uma especificação formal do SNMPv3. Semântica de Ações alia o rigor matemático com clareza e facilidade de expressão. O principal propósito desta descrição é especificar a estrutura e o comportamento do SNMPv3 sem ambiguidades. Inconsistências foram detectadas na descrição informal e são discutidas nesta especificação.

Palavras-chave: Semântica de Ações, SNMPv3, Gerência de Redes, Formalismo Semântico.

1 Introdução

A necessidade de sistemas eficazes de gerência de redes aumenta na medida em que as redes têm se tornado maiores e mais complexas. O propósito destes sistemas é auxiliar gerentes humanos no trabalho do monitoramento e controle de redes de computadores. *Simple Network Management Protocol* (SNMP) é a arquitetura padrão de gerência de redes da Internet, estando especificada sua terceira versão (SNMPv3) [8, 2, 10]. Há atualmente um grande número de sistemas disponíveis baseados no SNMP, tanto comerciais quanto de domínio público.

Um sistema de gerência de redes baseado no SNMPv3 é composto por *entidades* de gerência que se comunicam usando um protocolo de gerência [3, 16], como pode ser visto na Figura 1. As entidades SNMPv3 são tradicionalmente chamadas de *gerentes* e *agentes*. Cada sistema tem pelo menos uma entidade na função de gerente, que é responsável por avaliar o desempenho dos dispositivos gerenciados, ou por diagnosticar falhas, entre outras tarefas. Cada dispositivo gerenciado contém uma entidade na função de agente, que é responsável por

armazenar as informações de gerência coletadas neste dispositivo, em estruturas de armazenamento virtuais chamadas de *Management Information Base* (MIB).

Uma entidade SNMP é composta por uma ou mais *aplicações SNMP*, que são os módulos que irão executar as operações de gerência, e por um núcleo, aqui denominado de *engenho SNMP*, que provê serviços para estas aplicações, tais como enviar e receber mensagens, fazer autenticação e criptografia dos dados, e controle de acesso aos objetos gerenciados.

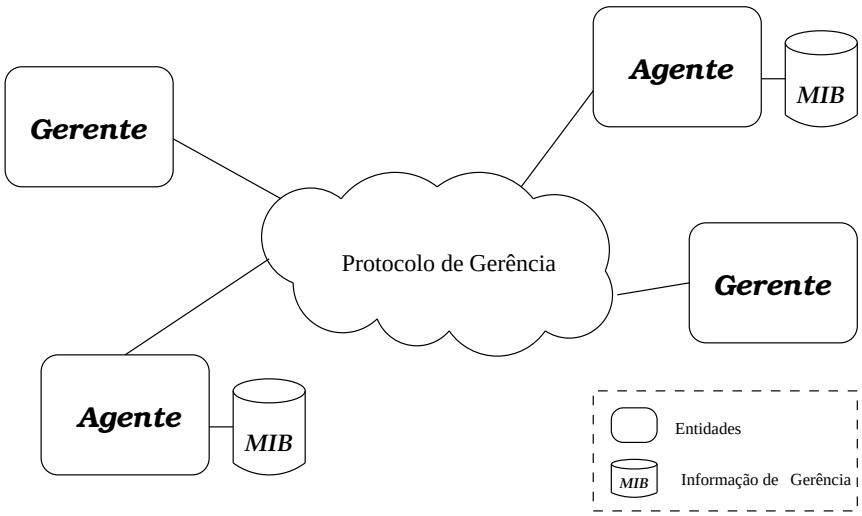

Figura 1: Sistema de Gerência de Redes baseado no SNMP.

Atualmente, a semântica dos componentes do SNMP é dada informalmente em uma série de descrições informais publicadas em RFC's(*Requests for Comments*) [3, 4, 8, 2, 10, 1, 19, 11]. A natureza informal destas descrições pode levar a interpretações incorretas e abrir a possibilidade de implementações inconsistentes.

Semântica de Ações é um formalismo introduzido em [12, 18] para prover descrições legíveis de linguagens de programação. Descrições em semântica de ações mapeiam sintaxe abstrata em entidades *ad hoc* chamadas *ações*, que podem ser executadas. Descrições em semântica de ações são modulares e facilmente extensíveis ou modificáveis. Ações permitem separar o fluxo de controle dos diversos fluxos de dados durante a execução, provendo uma forma natural para descrever computações.

Este trabalho é parte de uma série de descrições usando Semântica de Ações, em que cada componente do SNMPv3 vem sendo descrito. A série inicia com [5], onde propõe-se o uso de semântica de ações para especificar o comportamento de objetos de gerência de redes. Em [6], apresenta-se uma descrição formal das operações SNMPv3, que fazem a transferência dos objetos de gerência entre as diversas entidades. Em [7], apresenta-se uma descrição formal do Despachante SNMPv3, que é o módulo principal do núcleo das entidades. Neste trabalho,

usa-se Semântica de Ações para descrever as entidades SNMPv3 por completo, seguindo as definições apresentadas em [8].

O resto deste artigo é organizado como segue. A Seção 2 apresenta uma introdução à Semântica de Ações. A Seção 3 contém a especificação formal das entidades SNMPv3. A Seção 4 traz os principais resultados alcançados através desta especificação formal. A Seção 5 contém as conclusões.

2 Semântica de Ações

Semântica de ações [12] é um formalismo semântico para a especificação de conceitos de programação. Seu uso é encontrado na descrição de linguagens de programação de uso real e outras aplicações, entre elas [14, 15, 9, 13].

Semântica de ações descreve a semântica de linguagens e aplicações usando uma metalinguagem formal denominada *Notação de Ações*. Os símbolos usados na notação de ações são intencionalmente prolixos, sendo utilizados termos da língua inglesa - mas completamente formais, pois funcionam como funções - para expressar a maioria dos conceitos presentes na computação (facilidade de leitura). Esta notação pode ser facilmente estendida através da inclusão de novos tipos de dados ou através da criação de abreviaturas para entidades já especificadas (modularidade).

Em semântica de ações, o significado de cada frase da aplicação é representado por entidades especiais chamadas de *ações*. Ações são essencialmente dinâmicas, entidades computacionais representando controle e processamento de informação. *Ações* são entidades que podem ser *executadas* e seu desempenho corresponde a possíveis comportamentos da computação, que pode tanto terminar normalmente (complete), terminar excepcionalmente (escape), terminar sem sucesso (fail) ou não terminar (diverge). Notação de ações provê algumas *ações primitivas*, e vários *combinadores de ações* para formar ações complexas, correspondendo aos principais conceitos fundamentais de linguagens de programação.

A *notação de dados* é usada para descrever os dados processados pelas ações. A notação de dados incluída na notação de ações padrão provê uma coleção de tipos de dados definidos algebricamente, incluindo números, valores-verdade, caracteres, *strings*, listas, conjuntos, tuplas, mapeamentos, etc.; também são incluídos identificadores, células e agentes, que são utilizados para acessar outros dados, e algumas entidades compostas com componentes de dados, tais como mensagens e contratos.

A terceira classe de entidades deste formalismo é o *produtor*, que representa um dado não avaliado, cujo valor depende da informação correntemente disponível como entrada à ação onde o produtor está inserido. *Produtores* são entidades que podem ser *avaliadas* para gerar dados. Um exemplo de produtor padrão é the value stored in the given cell, cujo valor depende do conteúdo corrente da memória e da informação passada para a ação em que o produtor aparece.

O comportamento das ações é dividido em facetas, conforme o tipo de informação por elas processado. Cada faceta provê algumas ações primitivas e combinadores para formar ações complexas, correspondendo aos principais conceitos

fundamentais da computação. Além disto, provê um mecanismo para manipular dados e para gerar dados partindo de um tipo específico de informação.

Faceta básica é aquela cujos componentes estão relacionados apenas com o *fluxo de controle* na execução de diversas ações.

Faceta funcional é aquela cujos componentes estão relacionados com o *fluxo de informação transitória*, que são tuplas de dados trocadas entre as ações.

Faceta declarativa é aquela cujos componentes estão relacionados com o *fluxo de informação com escopo*, que é aquela proveniente da ligação de identificadores a dados identificáveis. Esta associação é denominada de *binding*.

Faceta imperativa é aquela cujos componentes estão relacionados com o *fluxo de informação estável*.

Faceta reflexiva é aquela cujos componentes estão relacionados com o *encapsulamento de ações como dados*.

Faceta comunicativa é aquela cujos componentes estão relacionados com o *processamento de informação em sistemas distribuídos de agentes*.

A notação $\square$ representa que um item é definido em outro nível da especificação.

Para uma descrição mais detalhada da notação de ações, refira-se a [12] ou [18]. (o primeiro aborda todos os aspectos relatados deste sistema formal; o último não aborda a faceta comunicativa nem as operações semânticas da notação).

3 Composição de Entidades SNMPv3

Toda a especificação a seguir é efetuada a partir da especificação informal de uma entidade como mostrado em [8]. As entidades padronizadas pelo protocolo serão estudadas: entidades com a função de gerente e com a função de agente, conforme o padrão estipulado em [8].

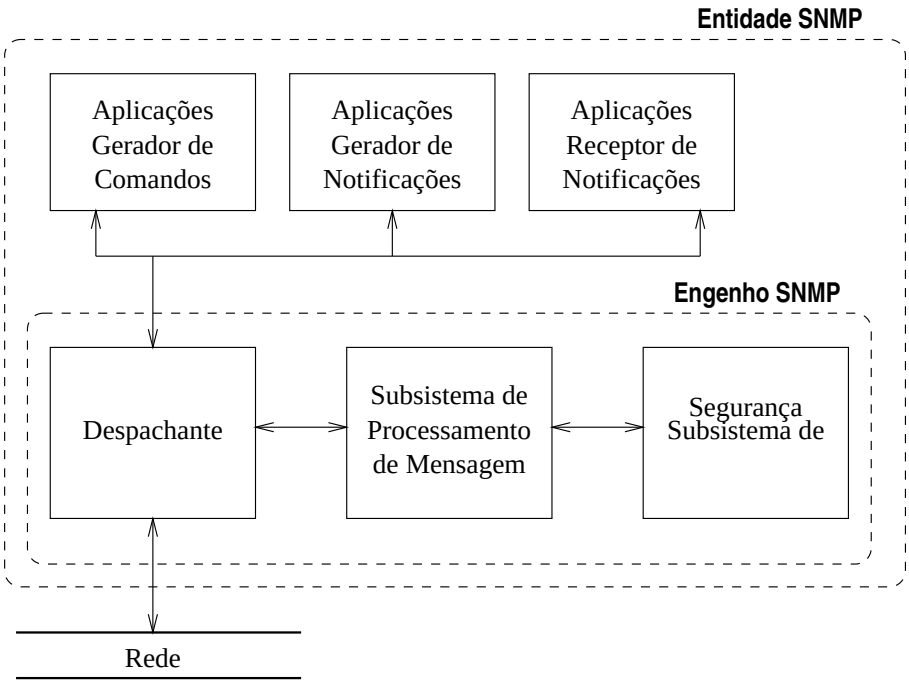

Figura 2: Gerente SNMP.

Uma entidade com a função de *gerente* (Figura 2) contém os seguintes módulos: um Despachante, um Subsistema de Processamento de Mensagens, um Subsistema de Segurança, pelo menos uma aplicação Receptor de Notificações e diversas aplicações usuárias executando as aplicações Gerador de Comandos e Gerador de Notificações. Cada um deles será executado por um agente da semântica de ações, como visto na ação Manager-daemon abaixo:

```
• Manager-daemon :: action
(1) Manager-daemon =
        |subordinate a dispatcher then bind "dispatcher" to it
        and
        |subordinate an application then bind "CR" to it
        and initiate user-applications
        and initiate MPS
        and initiate SS
        hence
        |send a message[to the agent bound to "dispatcher"]
        |              [containing closure abstraction of Dispatcher-daemon]
        and
        |send a message[to the agent bound to "CR"]
        |              [containing closure abstraction of CR-daemon]
        and activate user-applications
        and activate MPS
        and activate SS
```

A forma geral de uma entidade consiste em subordinar todos os agente da semântica de ações necessários, atribuindo-lhes um identificador (ações subordinate), e em seguida lhes fornecer a ação que deverão executar (ações send). Desta forma, todos os módulos podem referenciar aos outros, trocando mensagens.

As ações initiate e activate são criadas pois neste nível da especificação não é possível especificar o número de subcomponentes de uma entidade (genérica) e seus identificadores. As ações do tipo initiate serão compostas por uma ação subordinate para captar um agente da semântica de ações do tipo necessário e então uma ação bind para associar um identificador a este agente. As ações do tipo activate serão compostas por uma ação send para enviar a ação que deve ser executada pelo agente antes subordinado pela ação initiate correspondente. Genericamente, para inserir um novo módulo X na entidade, faz-se necessário construir uma ação initiate X, para subordinar um agente da semântica de ações para representar o módulo, e uma ação activate X para passar a ação a ser executada por este agente. As aplicações usuárias variam para cada entidade formada. O combinador hence propaga os *bindings* criados pelas ações subordinate e initiate para as ações send, a fim de que todos os módulos possam ter uma referência dos outros e assim se comunicar.

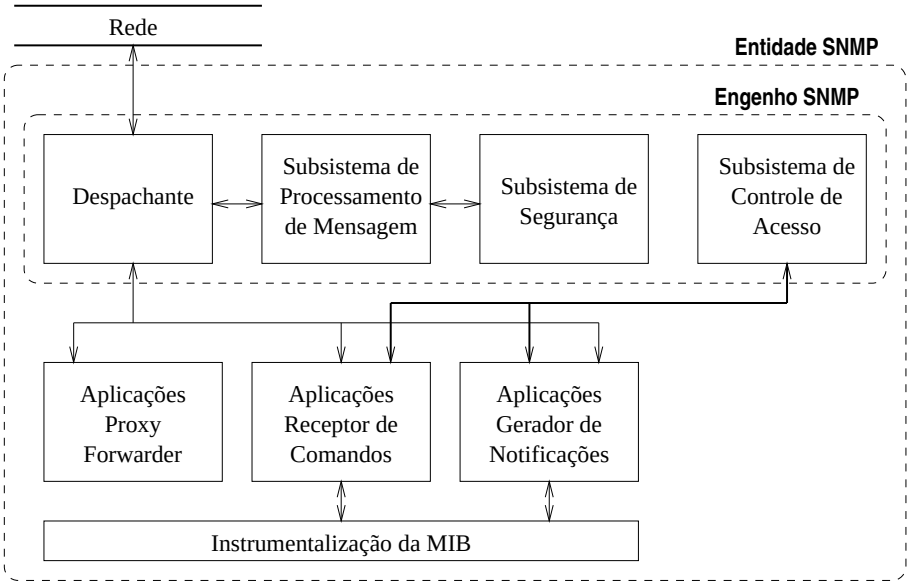

Figura 3: Agente SNMP.

Uma entidade com a função de *agente* (Figura 3) contém os seguintes módulos: um Despachante, um Subsistema de Processamento de Mensagens, um Subsistema de Segurança, um Subsistema de Controle de Acesso, pelo menos uma aplicação Receptor de Comandos e diversas aplicações usuárias executando as aplicações Gerador de Notificações ou *Proxy Forwarder*. Cada um deles será executado por um agente da semântica de ações, como visto na ação Agent-daemon abaixo:

- Agent-daemon :: action

(2) Agent-daemon =
```
    |subordinate a dispatcher then bind "dispatcher" to it
    and
    |subordinate an application then bind "NR" to it
    and initiate user-applications
    and initiate MPS
    and initiate SS
    and initiate ACS
    hence
    |send a message[to the agent bound to "dispatcher"]
    |              [containing closure abstraction of Dispatcher-daemon]
    and
    |send a message[to the agent bound to "NR"]
    |              [containing closure abstraction of NR-daemon]
    and activate user-applications
    and activate MPS
    and activate SS
    and activate ACS
```

As ações dependentes da implementação são explicadas a seguir. A ação initiate user-applications deve contratar um agente da semântica de ações para cada aplicação usuária que exista na entidade. A ação initiate MPS deve contratar um agente da semântica de ações para cada modelo de processamento de mensagens que exista na entidade, a ação initiate SS para cada modelo de segurança, e a ação initiate ACS para cada modelo de controle de acesso. A ação activate user-applications envia a ação a ser executada por cada aplicação usuária que exista na entidade. A ação activate MPS envia a ação a ser executada por cada modelo de processamento de mensagens, a ação activate SS por cada modelo de segurança, e a ação activate ACS por cada modelo de controle de acesso. Todas estas ações são especificadas como $\Box$ por serem definidas em outro nível da especificação.

(3) initiate user-applications = $\Box$
(4) initiate MPS = $\Box$
(5) initiate SS = $\Box$
(6) initiate ACS = $\Box$
(7) activate user-applications = $\Box$
(8) activate MPS = $\Box$
(9) activate SS = $\Box$
(10) activate ACS = $\Box$

O modelo já existente para o Subsistema de Processamento de Mensagens é o v3MP, o modelo já existente para o Subsistema de Segurança é o *User-based Security Model* (USM) e o modelo já existente para o Subsistema de Controle de Acesso é o *View-based Access Control Model* (VACM). Suas especificações formais ficam propostas como trabalho futuro. A seguir, vemos a especificação do engenho e, em seguida, das aplicações padrão.

3.1 Engenho de uma Entidade SNMPv3

Esta seção define os agentes da semântica de ações executando as ações referentes aos módulos do engenho SNMP. Como visto anteriormente, o engenho SNMP é formado por um Despachante, um subsistema de processamento de mensagens, um subsistema de segurança e um subsistema de controle de acesso, sendo que a sua especificação em semântica de ações é organizada como um conjunto de agentes da semântica de ações, cada um responsável por um destes módulos. A comunicação entre eles se dá através da troca de mensagens.

Um agente da semântica de ações executando a ação referente ao Despachante é descrito a seguir. Os outros módulos do engenho não serão especificados neste trabalho, sendo propostos para um trabalho futuro.

- Dispatcher-daemon :: action

(11) Dispatcher-daemon =

```
|initialize LCD-Dispatcher
hence unfolding
|||procedure Dispatcher
|trap complete
and then unfold
```

Um Despachante é um agente da semântica de ações executando a ação procedure Dispatcher, (descrita em [7]) que espera por mensagens endereçadas ao Despachante.

Cada módulo de uma entidade pode possuir um conjunto de variáveis locais, para salvar informações temporariamente, definidas na forma de uma MIB chamada de *Local Configuration Datastore* (LCD). Estas variáveis podem ser utilizadas para salvar informações recebidas numa requisição ou para salvar dados que são utilizados ao se gerar novas mensagens, indicando contextos ou modelos, por exemplo. A única variável local que o despachante necessita é a variável *"sender"*, que serve para indicar qual aplicação está enviando a mensagem atual, caso necessite de uma resposta. A ação initialize LCD-Dispatcher produz a MIB contendo a variável descrita acima.

- initialize LCD-Dispatcher :: action

(12) initialize LCD-Dispatcher =

```
allocate a cell then bind "sender" to it
```

A ação trap complete termina normalmente, impedindo a propagação do estado de terminação excepcional, gerado por uma ação interna á ação procedure Dispatcher, o que fará o Despachante aguardar por uma nova mensagem a ser processada. Este tipo de comportamento é necessário quando a ação deve terminar normalmente para que o processamento que vem a seguir (fora do contexto da ação) possa continuar.

O combinador unfolding faz com que a ação procedure Dispatcher seja executada infinitamente, e o agente da semântica de ações passa a atuar como se fosse um servidor com a função de executar esta ação. O combinador hence propaga os *bindings* criados pela ação initialize LCD-Dispatcher para a ação procedure Dispatcher.

3.2 Aplicações SNMPv3

Esta seção define os agentes da semântica de ações executando as ações referentes às aplicações padrão do SNMP. O conjunto de aplicações de uma entidade envolve aplicações do tipo Gerador de Comandos, Receptor de Comandos, Gerador de Notificações, Receptor de Notificações e *Proxy Forwarder*.

As aplicações Receptor de Comandos e Receptor de Notificações serão aqui produzidas como sendo um agente da semântica de ações executando uma ação que corresponde ao procedimento que a aplicação deve seguir.

Já as aplicações Gerador de Comandos e Gerador de Notificações não serão executadas por um agente da semântica de ações, e sim serão chamadas por aplicações usuárias. *Aplicações usuárias* são aplicações não padronizadas, escritas para controlar e monitorar informações de gerência. Isto ocorrerá porque na especificação informal não há nenhuma referência a respeito da comunicação das aplicações Gerador de Comandos e Gerador de Notificações com aplicações usuárias na forma de primitivas. Já as aplicações Receptor de Comandos e Receptor de Notificações possuem esta referência quando, dentro do processamento das operações *Inform* e *Trap*, uma aplicação usuária deve ser escolhida para realizar um processo dependente da implementação em cada caso.

3.2.1 Gerador de Comandos. Este tipo de aplicação não funcionará como um servidor, mas será implementado dentro de uma aplicação usuária que necessite enviar consultas ou alterações para MIBs em outras entidades. Ou seja, não é necessário que um agente da semântica de ações fique responsável por controlar todo o processamento efetuado por esta aplicação. Entretanto, qualquer aplicação usuária que deseje enviar uma operação de leitura ou de escrita irá executar a ação procedure CommandGenerator, que envia uma única requisição e recebe sua respectiva resposta.

Abaixo, vê-se um exemplo de aplicação usuária atuando como emissora de operações SNMP de leitura e de escrita.

```
• user-application _ :: integer → action
(13) user-application X:integer =
        |initialize LCD-Generate
        hence
        |...
        |procedure CommandGenerator
        |...
```

A ação procedure CommandGenerator é descrita em [6].

Inicialmente, a LCD da aplicação Gerador de Comandos é construída pela ação initialize LCD-Generate. Então, em algum ponto da especificação, a ação procedure CommandGenerator é chamada, recebendo a informação transitória correta, que é constituída por dados referentes ao PDU a ser enviado. As variáveis locais que um Gerador de Comandos necessita são: *"transpDom"*, *"transpAdd"*, *"MPModel"*, *"secModel"*, *"secName"*, *"secLevel"*, *"pduVersion"*, *"contextEngID"*, *"contextName"* e *"request-id"*. A ação initialize LCD-Generate produz a MIB contendo as variáveis descritas acima.

- initialize LCD-Generate :: action

(14) initialize LCD-Generate =
```
|rebind
and allocate a cell then bind "transpDom" to it
and allocate a cell then bind "transpAdd" to it
and allocate a cell then bind "MPModel" to it
and allocate a cell then bind "secModel" to it
and allocate a cell then bind "secName" to it
and allocate a cell then bind "secLevel" to it
and allocate a cell then bind "pduVersion" to it
and allocate a cell then bind "contextEngID" to it
and allocate a cell then bind "contextName" to it
```

3.2.2 Receptor de Comandos. Esta aplicação será representada por um agente da semântica de ações, executando a seguinte ação:

- CommandResponder-daemon :: action

(15) CommandResponder-daemon =
```
||initialize MIBs
and
||initialize LCD-Process
and
||register CR
hence unfolding
|||procedure CommandResponder
||trap complete
|and then unfold
```

As MIBs de uma entidade podem ser inicializadas logo que uma aplicação Receptor de Comandos começar a executar. O processo para inserir novos objetos de gerência nestas MIBs é muito simples, bastando especificar como o objeto será atribuído e consultado (ações get e set para um objeto de gerência) e inserir uma célula de memória, caso necessário, usando a ação initialize MIBs.

Então, a LCD da aplicação Receptor de Comandos é construída pela ação initialize LCD-Process. As variáveis locais que um Receptor de Comandos necessita são: *"MPModel"*, *"secModel"*, *"secName"*, *"secLevel"*, *"pduVersion"*, *"contextEngID"*, *"contextName"*, *"request-id"*, *"non-repeaters"*, *"max-repetitions"* e *"variable-bindings"*.

- initialize LCD-Process :: action

```
(16) initialize LCD-Process =
        |rebind
        and allocate a cell then bind "MPModel" to it
        and allocate a cell then bind "secModel" to it
        and allocate a cell then bind "secName" to it
        and allocate a cell then bind "secLevel" to it
        and allocate a cell then bind "pduVersion" to it
        and allocate a cell then bind "contextEnglD" to it
        and allocate a cell then bind "contextName" to it
        and allocate a cell then bind "request-id" to it
        and allocate a cell then bind "non-repeaters" to it
        and allocate a cell then bind "max-repetitions" to it
        and allocate a cell then bind "variable-bindings" to it
```

Antes que uma aplicação Receptor de Comandos possa processar mensagens, ela deve associar-se com um engenho SNMP, para que os tipos de operações que ela irá processar estejam indicados. Isto é feito na ação register CR. A primitiva usada para este propósito é *registerContextEngineID*, sendo formalizada na ação associate A with D, que associa o agente que está executando a ação com o dado D, que contém o identificador do engenho ao qual o agente da semântica de ações está se cadastrando e o respectivo tipo de operação. É comum que apenas uma aplicação Receptor de Comandos esteja cadastrada por engenho. Uma vez que isto aconteça, ela pode esperar por mensagens do tipo especificado.

```
  • register CR :: action
(17) register CR =
        |get("snmpEngineID")
        then
        ||associate performing-agent with (it, Get)
        |and associate performing-agent with (it, GetNext)
        |and associate performing-agent with (it, GetBulk)
        |and associate performing-agent with (it, Set)
```

Então, a ação procedure CommandResponder é executada eternamente dentro de um *loop*, processando todas as operações de leitura e escrita recebidas pela entidade. A ação trap complete termina normalmente, impedindo que qualquer escape gerado internamente à ação procedure CommandResponder flua para fora do contexto atual, e permitindo que uma nova mensagem seja tratada.

A ação procedure CommandResponder é descrita em [6].

3.2.3 Gerador de Notificações. Da mesma forma como na aplicação Gerador de Comandos, este tipo de aplicação não funcionará como um servidor, mas será implementado dentro de uma aplicação usuária que necessite enviar notificações para outras entidades.

Abaixo, vê-se um exemplo de aplicação usuária atuando como emissora de notificações SNMP.

```
(18) user-application X:integer =
        |initialize LCD-Generate
        hence
        |... procedure NotificationOriginator ...
```

Antes de sua execução, a aplicação usuária que for utilizá-la deverá determinar as entidades de destino para as quais uma notificação será enviada, determinar o contexto atual com as informações a serem transmitidas e determinar a lista de variáveis a ser enviada para cada destino. Só então, a ação procedure NotificationOriginator será executada para cada destino, com os respectivos nomes e valores das variáveis passados para a ação como informação transitória.

A ação initialize LCD-Generate já foi explicada na aplicação Gerador de Comandos. A ação procedure NotificationOriginator é descrita em [6].

3.2.4 Receptor de Notificações. Da mesma forma como na aplicação Receptor de Comandos, esta aplicação será representada por um agente da semântica de ações, executando a seguinte ação:

```
(19) NotificationReceiver-daemon =
        ||initialize LCD-Process
        |and register NR
        hence unfolding
        |||procedure NotificationReceiver
        ||trap complete
        |and then unfold
```

Inicialmente, a LCD da aplicação Receptor de Notificações é construída pela ação initialize LCD-Process.

A aplicação se cadastra junto ao engenho da entidade da mesma forma que a aplicação Receptor de Comandos, pela ação register NR, só que com as operações *Inform* e *Trap*.

```
(20) register NR =
        |get("snmpEngineID")
        then
        ||associate performing-agent with (it, Inform)
        |and associate performing-agent with (it, Trap)
```

Então, a ação procedure NotificationReceiver é executada eternamente dentro de um *loop*, processando todas as operações de notificação recebidas pela entidade. O funcionamento da ação trap complete é idêntico ao encontrado na ação CommnadResponder-daemon.

A ação procedure NotificationReceiver é descrita em [6].

3.3 Outras Ações de uma Entidade SNMP

Esta seção contém definições de dados, ações e produtores auxiliares utilizados na especificação de uma entidade SNMP, incluindo abreviaturas e entidades semânticas.

A ação associate A with D associa uma aplicação executada pelo agente da semântica de ações A com os dados fornecidos pela tupla D, no caso, o identificador do engenho onde será feito o cadastro e o tipo da operação.

(21) associate A:agent with (C:contextEngineId, O:pduType) = □

A ação initialize MIBs é dependente da implementação e inicializa as MIBs de uma entidade.

(22) initialize MIBs = □

A ação allocate a cell escolhe uma célula dentre as células não alocadas, faz sua reserva e a devolve como informação transitória. É uma abreviatura para:

```
(23) allocate a cell =
         indivisibly
         ||choose a cell [not in mapped-set of the current storage]
         |then reserve the given cell and give it
```

A ação get(N) retorna o valor associado ao objeto de nome N numa MIB local como transitório.

(24) get(N:ObjectName) = □

Da forma como a entidade foi especificada, todos os agente da semântica de ações que são por ela ativados se conhecem e podem se referenciar. A forma usada nas seções anteriores, pela qual qualquer módulo referenciava o Despachante era através do produtor the Dispatcher of Entity.

(25) the Dispatcher of Entity = give the agent bound to "dispatcher"

Outros produtores podem ser definidos desta forma, como por exemplo the MPModel associated with _, dependendo apenas que os modelos existentes sejam retornados conforme seu numero definido em [8]. Caso contrário, *nothing* deve ser retornado para que a ação que chamou o produtor termine com falha.

4 Inconsistências Detectadas nos Documentos do IETF

O objetivo desta seção é apresentar algumas inconsistências e ambiguidades detectadas durante o extenso trabalho de formalização da parte fundamental do protocolo SNMPv3.

A seguir, é apresentado um exemplo de inconsistência detectada no engenho SNMP: Quando uma indicação de erro é retornada pelo modelo de processamento de mensagens no envio de uma resposta, esta indicação é retornada para a aplicação que processou a operação e gerou a resposta. Isto é feito pela ação send back em procedure Disp send-response B. "If the result is an errorIndication, the errorIndication is returned to the calling application." [2, página 10]. **Inconsistência:** Só que segundo a especificação informal, a aplicação que enviou a

operação de resposta (Receptor de Comandos ou Receptor de Notificações) não espera por esta indicação de erro.

A seguir, é apresentado um exemplo de inconsistência detectada nas aplicações SNMP: Uma aplicação armazena *sendPduHandle* após enviar uma requisição para poder correlacioná-la com sua resposta. "The command generator should store the sendPduHandle so that it can correlate a response to the original request." [10, página 7]. **Inconsistência:** O índice *sendPduHandle* salvo pela aplicação Gerador de Comandos nunca mais é utilizado, pois o Despachante só manda para uma aplicação as respostas destinadas a ela. Portanto, não fica claro como este índice será manipulado quando for retornado.

A seguir, são apresentados algumas inconsistências detectadas na composição de uma entidade SNMP: Existem informações utilizadas por um módulo que devem ser guardadas durante o processamento de uma operação. Estas informações são armazenadas em uma LCD: "The subsystems, models, and applications within an SNMP entity may need to retain their own sets of configuration information. The collection of these sets of information is referred to as an entity's Local Configuration Datastore (LCD)." [3, página 27]. A seguir encontra-se um exemplo deste tipo de informação: "The request-id is extracted from the PDU and saved." [10, página 11]. **Inconsistência:** Não está claro quais informações devem existir na LCD de cada módulo, pois não há uma seção que a descreva inteiramente.

Uma relação completa com todas as inconsistências e ambiguidades encontradas pode ser obtida por e-mail através dos autores.

5 Conclusão

A arquitetura padrão para gerência de redes da Internet é o SNMPv3 (Simple Network Management Protocol version 3). Um sistema baseado em SNMPv3 é composto pelas chamadas entidades de gerência, que se comunicam usando o protocolo de gerência. As entidades são compostas por um engenho, além de aplicações padrão. As entidades são descritas informalmente em documentos do IETF. Neste trabalho apresentamos sua descrição formal utilizando a notação chamada Semântica de Ações, que alia o rigor matemático com clareza e facilidade de expressão.

Esta tradução da parte fundamental do SNMPv3 para um formalismo semântico permitiu a detecção de inconsistências e ambiguidades do mesmo. Com este resultado, pode-se gerar novas implementações automaticamente, automatizar a validação de implementações já existentes do SNMP, além de facilitar a correção de futuras atualizações do protocolo. Além disto, possibilita a definição explícita da semântica dos objetos de gerência de redes. Num contexto mais amplo, a utilização de formalismos para descrever semanticamente aplicações computacionais é uma atitude importante mas geralmente desprezada, devido a sua complexidade. Neste sentido, semântica de ações contribui proporcionando um caminho plauzível para esta mudança tão necessária a fim de que se possa produzir aplicações livres de interpretações dúbias e incompletas. Como desvantagem deve-se

mencionar o entendimento do formalismo Semântica de Ações que, apesar de mais simples que outros formalismos, tem também uma notação complexa em alguns aspectos, além de limitações com relação à faceta comunicativa [12].

Como trabalhos futuros coloca-se a produção da especificação do SNMPv3 usando-se outros formalismos, como SDL, LOTOS ou ESTELLE [17].

Referências

1. V. Blumenthal and B. Wijnen. User-based Security Model (USM) for the SNMPv3. Request for Comments 2574, April 1999.
2. J. Case, D. Harrington, R. Presuhn, and B. Wijnen. Message Processing and Dispatching for the SNMP. Request for Comments 2572, May 1999.
3. J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Protocol Operations for Version 2 of the SNMP. Request for Comments 1905, January 1996.
4. J. Case, R. Mundy, D. Partain, and B. Stewart. Introduction to Version 3 of the Internet-Standard Network Management Framework. Request for Comments 2570, April 1999.
5. E. P. Duarte Jr. and M. A. Musicante. Formal Specification of SNMP MIB's Using Action Semantics: The Routing Proxy Case Study. In *Proc. of the Sixth IFIP/IEEE Int'l Symp. on Integrated Network Management*, pages 417–430, Boston, USA, May 1999. IEEE Publishing.
6. D. C. Furlan, M. A. Musicante, and E. P. Duarte Jr. A Formal Description of SNMPv3 Standard Applications using Action Semantics. *BRICS Notes (ISSN 0909-3206), NS-00-06, Denmark*, August 2000.
7. D. C. Furlan, M. A. Musicante, and E. P. Duarte Jr. An Action Semantics Description of the SNMPv3 Dispatcher. *Anais do IV Simpósio Brasileiro de Linguagens de Programação (SBLP2000)*, pages 186–199, 2000.
8. D. Harrington, R. Presuhn, and B. Wijnen. An Architecture for Describing SNMP Management Frameworks. Request for Comments 2571, May 1999.
9. S. B. Lassen. Action semantics reasoning about functional programs. Technical report, University of Aarhus, Department of Computer Science, 1995. Available at URL: `http://www.brics.dk/~thales/docs/ldpl.dvi.gz`.
10. D. Levi, P. Meyer, and B. Stewart. SNMPv3 Applications. Request for Comments 2573, May 1999.
11. K. McCloghrie, D. Perkins, and J. Schoenwaelder. Structure of Management Information Version 2 (SMIv2). Request for Comments 2578, April 1999.
12. P. D. Mosses. *Action Semantics.* Cambridge University Press, Cambridge, UK, 1992.
13. P. D. Mosses and M. A. Musicante. An action semantics for ML concurrency primitives. Number 873 in Lecture Notes in Computer Science, Barcelona, Spain, October 1994. FME, Springer-Verlag.
14. M. A. Musicante. The Sun RPC language semantics. In *Proceedings of PANEL'92, XVIII Latin-American Conference on Informatics.* Universidad de Las Palmas de Gran Canaria, 1992.
15. J.P. Nielsen and J.U.Toft. Formal specification of ANDF, existing subset. Technical Report 202104/RPT/19, issue 2, DCC International A/S, Lundtoftvej 1C, DK-2800 Lyngby, Denmark, 1994.
16. M. T. Rose. *The Simple Book: an Introduction to Internet Management.* Prentice Hall, second edition, 1994.

17. K. J. Turner. *Using formal description techniques: an introduction to Estelle, LOTOS and SDL.* John Willey and Sons, 1993.
18. D. A. Watt. *Programming Language Syntax and Semantics.* Prentice Hall, UK, 1991.
19. B. Wijnen, R. Presuhn, and K. McCloghrie. View-based Access Control Model (VACM) for the SNMP. Request for Comments 2575, April 1999.