Universidade Federal do Rio de Janeiro

Núcleo de Computação Eletrônica

Pedro Felipe de Souza Lopes

GERÊNCIA DE NÍVEL DE SERVIÇOS PARA APLICAÇÕES:

Considerações sobre o uso de RMON2 e tecnologias correlatas

Rio de Janeiro

2010

Pedro Felipe de Souza Lopes

GERÊNCIA DE NÍVEL DE SERVIÇOS PARA APLICAÇÕES:

Considerações sobre o uso de RMON2 e tecnologias correlatas

Monografia apresentada para obtenção do título de Especialista em Gerência de Redes de Computadores no Curso de Pós-Graduação Lato Sensu em Gerência de Redes de Computadores e Tecnologia Internet do Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro – NCE/UFRJ.

Orientador:

Moacyr Henrique Cruz de Azevedo, M.Sc., UFRJ, Brasil

Rio de Janeiro

2010

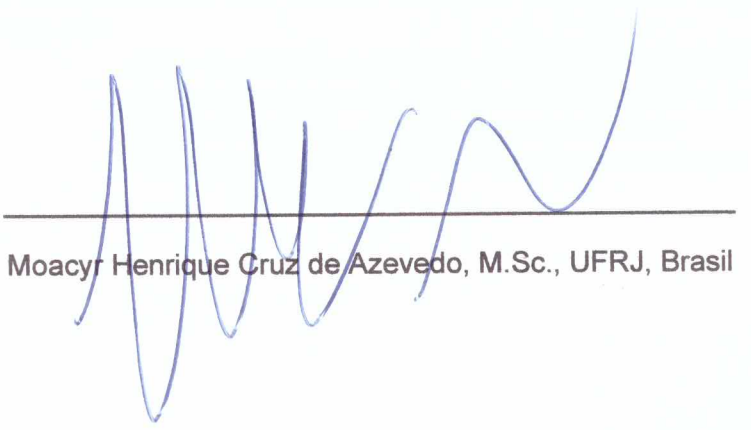
Pedro Felipe de Souza Lopes

GERÊNCIA DE NÍVEL DE SERVIÇOS PARA APLICAÇÕES:

Considerações sobre o uso de RMON2 e tecnologias correlatas

Monografia apresentada para obtenção do título de Especialista em Gerência de Redes de Computadores no Curso de Pós-Graduação Lato Sensu em Gerência de Redes de Computadores e Tecnologia Internet do Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro – NCE/UFRJ.

Aprovada em março de 2010.



Moacyr Henrique Cruz de Azevedo, M.Sc., UFRJ, Brasil

A todos aqueles que vieram antes de mim e aplainaram o caminho para mim.

Junto ao monte, a lavoura, esplêndida, aparece!
Uns plantam: e outro chega e é quem recolhe a messe.
Recém-chegado, não te alegres; a esperar
Por tua vez está alguém: -- outro que irá chegar
CHUEHWEN

AGRADECIMENTOS

Ao Bartholomeu e ao Philipe, por não me perturbarem (só um pouquinho...). E a Clery, por até ter me deixado fazê-lo.

RESUMO

LOPES, Pedro Felipe de Souza. **Gerência de nível de serviços para aplicações: Considerações sobre o uso de RMON2 e tecnologias correlatas**. Monografia (Especialização em Gerência de Redes e Tecnologia Internet). Núcleo de Computação Eletrônica, Universidade Federal do Rio de Janeiro. Rio de Janeiro, 2009.

Considerações sobre a viabilidade da gerência de nível de serviços focada em aplicações usando RMON2 e tecnologias correlatas (extensões da RMON2-MIB, API's padronizadas para geração de relatórios de *performance*). O estado atual destas tecnologias e algumas considerações sobre desenvolvimentos futuros desta abordagem são apresentados neste trabalho.

ABSTRACT

LOPES, Pedro Felipe de Souza. **Gerência de nível de serviços para aplicações: Considerações sobre o uso de RMON2 e tecnologias correlatas**. Monografia (Especialização em Gerência de Redes e Tecnologia Internet). Núcleo de Computação Eletrônica, Universidade Federal do Rio de Janeiro. Rio de Janeiro, 2009.

Appreciations about feasibility of service level management focused on applications using RMON2 and correlate technologies (RMON2-MIB extensions, standard API's for performance report generation). Present day status of these technologies and some considerations about future developments of this approach are shown in this work.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 **INTRODUÇÃO**

A integração da Informática com os processos de negócios é fato notório atualmente. A Informática serve não só para potencializar os processos de negócios como permite criar processos de negócios inexistentes, que podem gerar vantagem competitiva significativa. Existe então uma dependência dos processos de negócios com relação aos serviços que os suportam. Esta dependência gera a necessidade de gerenciar estes serviços de modo a garantir a continuidade e qualidade dos processos de negócios. E mais: a natureza desta gerência deveria ser de mais alto nível; ela visa antes o negócio como um todo que aspectos técnicos específicos e isolados (como rede, tráfego, sistemas, aplicações). Assim, ela seria feita sobre abstrações montadas sobre a rede corporativa: os serviços. Essa gerência é a chamada gerência de níveis de serviço (*Service Level Management*).

O objetivo deste documento é analisar alguns aspectos da gerência de serviços e o relacionamento desta gerência com os meio práticos disponíveis para implementá-la (a saber: SNMP, RMON e suas extensões e algumas tecnologias complementares). Estes meios são suficientes ou podem ser facilmente adaptados para atenderem às demandas do gerenciamento de serviços? Se não, quais seus limites neste cenário?

2 **GERENCIAMENTO DE NÍVEIS DE SERVIÇO (SERVICE LEVEL MANAGEMENT)**

Como vimos na introdução, o propósito deste documento é tecer algumas considerações sobre o uso dos recursos de SNMP e RMON no chamado gerenciamento de nível de serviço. Sendo este conceito fundamental para a discussão que se segue, seria conveniente definirmos melhor este termo e seus correlatos e situá-los melhor num contexto que permita a inter-relação.

2.1 CONCEITOS DE SLM

2.1.1 **Processo de negócios**

São as maneiras particulares de cada empresa pelas quais estas coordenam e organizam suas atividades produtivas e informações para produzir bens e serviços comercializáveis. Um processo de negócios típico pode incluir vários serviços, alguns deles dependentes da rede corporativa. E, no nosso caso, apenas os dependentes da rede corporativa serão considerados.

2.1.2 **Serviço**

É uma função que a rede corporativa oferece para o negócio. Um serviço é uma abstração, montada sobre componentes da rede. Um serviço pode ser “decomposto” num agregado de funções concretamente observáveis.

2.1.3 **Rede corporativa**

Para os fins de nos interessam, podemos defini-la como um conjunto de equipamentos de transmissão, linhas de transmissão entre os equipamentos de transmissão e os sistemas de computadores, sistemas de computadores e aplicações executadas nos sistemas de computadores.

2.1.4 **Gerência de nível de serviço (Service Level Management, SLM)**

É todo processo relacionado à identificação, definição e gerenciamento dos serviços oferecidos pela rede corporativa.

2.2 EVOLUÇÃO DOS CONCEITOS DE SLM

Como é natural, o conceito de SLM não surgiu de repente. Ele é fruto de uma evolução na visão de gerenciamento de recursos, cuja abrangência e cronologia pode ser resumida no quadro abaixo (adaptado de [5]):

Tabela 2.1: Evolução cronológica dos conceitos de monitoração

Ano	Estágio
1975	Gerência de equipamentos
1990	Gerência de redes
1991	Gerência de tráfego
1992	Gerência de sistemas
1994	Gerência de aplicações
1996	Gerência corporativa operacional (redes + tráfego + sistemas + aplicações)
1998	Gerência corporativa tática (corporativa operacional + SLM)
2000	Gerência corporativa estratégica (corporativa tática + gerência de processos de negócios)

3 **VISÃO SINTETICA DO RELACIONAMENTO ENTRE SLM, ELEMENTOS DE REDE E PROCESSOS DE GERÊNCIA/MONITORAÇÃO**

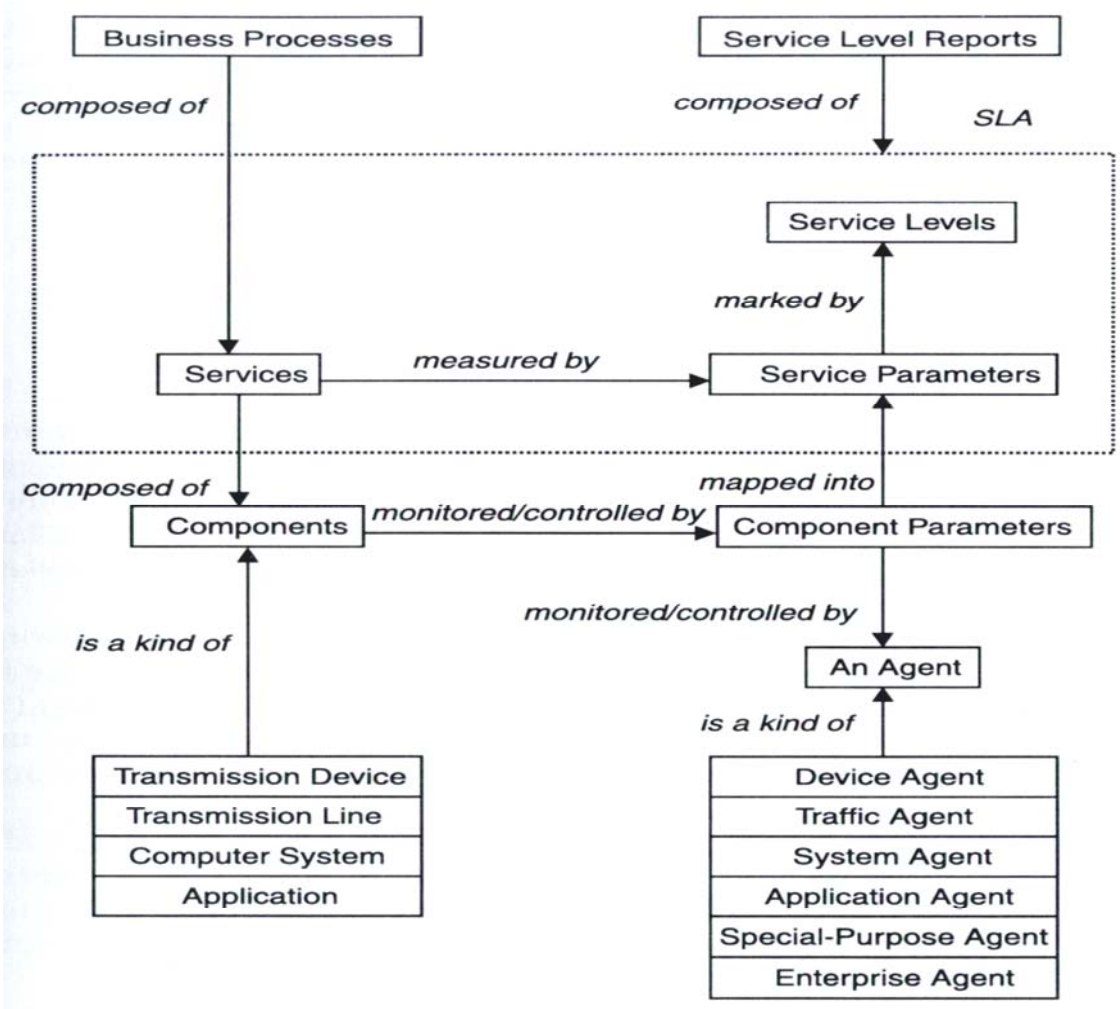


Figura 3.1: Arquitetura de uma solução para SLM

A figura acima sintetiza muito bem um dos aspectos mais complexos da SLM: o problema da disparidade semântica [5]. Em termos simples, a disparidade semântica ocorre porque os parâmetros que são fáceis para os técnicos em Informática medirem não são significativos para o usuário (“bytes/segundo”, por exemplo). Por outro lado, os parâmetros que interessam aos usuários muitas vezes são de natureza extremamente “vaga” e de difícil mensuração (“satisfação do cliente”, por exemplo). Para que a SLM atinja os objetivos a que se propõe alguma solução para esse problema deve ser dada. A figura 3.1 propõe uma “solução de

compromisso”: estabelecer algum tipo de mapeamento entre o que se pode quantificar e medir e as expectativas do cliente.

Cada “ramo” da figura propõe um modelo para os pólos da disparidade semântica. O ramo esquerdo modela um processo de negócios como um conjunto de serviços; assim, “medir” o processo pode ser mapeado em medir o serviço. Um serviço é considerado composto pela interação de diferentes componentes. Um componente, por sua vez, é um conjunto de elemento de rede corporativa (como definido em 2.1.3). O ramo direito apresenta a solução de compromisso propriamente dita: podemos medir os elementos de rede usando parâmetros bem definidos, através do uso de um agente (idéia já amplamente empregada no SMNP). Ao juntarmos os elementos para gerarmos um componente, este poderá também ser medido por um “agente de componente”. E para que o cliente possa interagir a figura propõe a criação de uma entidade conectando os dois ramos: o ***SLA*** (*Service Level Agreement* – acordo ou contrato de nível de serviço). Esta entidade introduzirá valores mais mensuráveis e mais próximos da realidade do cliente. É importante enfatizar que o cliente não precisa saber dos meandros da definição dos parâmetros. Cada conjunto de parâmetros define um nível de serviço; cada nível de serviço implica num comportamento do processo de negócios. Para cada nível de serviço gera-se um ***SLR*** (*Service Level Report,* relatório de nível de serviço). Cotejando estes relatórios com o comportamento do processo de negócios o cliente é capaz de escolher o que mais satisfaz. A partir daí cabe aos técnicos ajustarem os parâmetros na cadeia “abaixo”, analisando e correlacionando os diversos elementos envolvidos. Nesse processo de “ajuste de cima para baixo” nos parece haver alguns pontos ainda não suficientemente maduros e que podem causar dificuldades no uso efetivo do esquema apresentado pela figura: (a) as dificuldades inerentes ao processo de análise/correlação; (b) a falta um elemento essencial: gerenciamento do elemento “aplicação”.

O ponto focal deste trabalho é o item (b). Pela própria natureza desse elemento, construir agentes para ele não parece tarefa fácil. O que veremos a seguir são possíveis alternativas para o gerenciamento deste componente essencial.

4 **ESTUDO DE SOLUÇÕES POSSÍVEIS PARA MONITORAÇÃO DE APLICAÇÕES**

Vejamos se já seria possível desenvolver alguma coisa para a monitoração do componente “aplicação” usando tecnologias já disponíveis e *vendor independent*. Estes são pontos importantes nesta análise: as possíveis soluções devem ser “padrões abertos”, não soluções proprietárias. E estas soluções devem já ter alguma maturidade ou serem viáveis no atual estágio de conhecimento, mesmo que demandando algumas alterações.

4.1 APPLICATION RESPONSE MEASUREMENT (ARM) API

4.1.1 **O que é**

É um conjunto de funções chamadas por uma aplicação (API) para passar informações relevantes de uma transação da aplicação para um agente. O agente é o encarregado de coletar, contabilizar, organizar e disponibilizar as informações para as aplicações de gerência.

A idéia é permitir que as aplicações (ao invés de, digamos, “a rede”) “respondam” à pergunta essencial: as aplicações estão fazendo aquilo que se espera delas da maneira desejada, com um tempo de resposta aceitável?

Uma abordagem detalhada da ARM-API pode ser vista em [11]. Neste trabalho apresentaremos apenas uma visão sintética baseada no documento citado, objetivando fornecer uma base para as considerações que apresentaremos sobre a solução.

4.1.2 **Arquitetura**

A arquitetura básica da solução ARM-API pode ser resumida na figura 4.1.

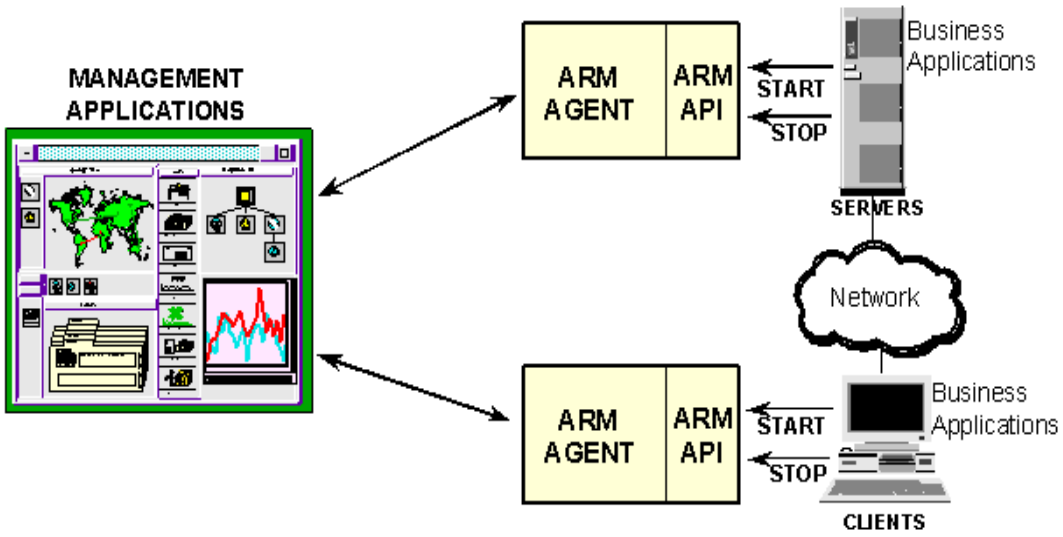


Figura 4.1: Arquitetura ARM-API

Assim, a solução ARM-API tem uma estrutura similar à do SNMP, com agentes e aplicações de gerência. No entanto, não há uma MIB definida e nem protocolo de comunicação como na arquitetura SNMP. Mas a arquitetura proposta é bastante flexível, podendo inclusive ser "acoplada" a uma estrutura SNMP já existente: como as API's se comunicam com o agente, nada impediria que o agente "formatasse" os dados numa ("futura") MIB e usasse o "*framework* SNMP" para transferir as informações.

4.1.3 **Como funciona**

Como visto a ARM-API é, basicamente, um conjunto de funções chamadas por uma aplicação para passar dados relevantes para um elemento consolidador externo à aplicação (o agente). A aplicação deve, então, definir quais pontos devem ser monitorados e, usando as funções, disponibilizar dados para análise. Isso significa que a aplicação deve ser (re)escrita especificamente para usar a ARM-API.

A ARM-API está atualmente na versão 2. Analogamente ao SNMP, a versão 2 estende a versão anterior mas é totalmente compatível com esta. Os conceitos básicos de uso permanecem de uma versão para outra.

4.1.3.1 ARM-API v1

O conjunto de funções disponibilizado é deliberadamente pequeno e simples, de modo a impactar pouco na aplicação sendo monitorada. Basicamente a ARM-API identifica a aplicação, a transição, o usuário (opcionalmente) e informa um código de estado ao fim da transação. Para a versão 1 da API estão disponíveis as seguintes funções:

a) *arm_init*: Permite definir um nome para a aplicação e, opcionalmente, uma identificação de usuário. Retorna um identificador único de aplicação que será usado por outras funções;

b) *arm_getid*: Permite definir um nome para a transação e, opcionalmente, detalhes da transação. O identificador único vindo da *arm_init* é usado para associar transações ⇔ aplicações. Retorna um identificador único de transação que será usado por outras funções;

c) *arm_start*: Indica o início de uma transação. Deve usar o identificador único de transação retornado pelo *arm_getid*. Como podem existir várias instâncias de uma mesma transação ocorrendo no sistema, cada chamada *arm_start* retorna um identificador único de instância de transação;

d) *arm_update*: Uso recomendado para transações muito longas. Serve para indicar que a transação ainda está em curso (a transação não “congelou”). Usa o identificador único de instância de transação retornado pela *arm_start* para identificar qual a instância “ainda viva” de uma transação. Não é uma função essencial, podendo não ser usada;

e) *arm_stop*: Indica o fim de uma de instância de transação. Usa o identificador único de instância de transação retornado pela *arm_start* para identificar qual instância de transação foi concluída. Outro parâmetro importante dessa função é o código de estado da transação, que será retornado para o agente para registro e contabilização. Este código pode ser GOOD, para uma transação completada com êxito, ERROR para transação concluída com falha e ABORT para transação não concluída;

f) *arm_end*: Indica que a aplicação não mais fará uso da ARM-API. Tipicamente, indica o fim de uma aplicação, liberando todos os recursos alocados desde a *arm_start*. Mas é importante perceber que os dados coletados pelo agente não são liberados, ficando disponíveis para consultas, exceto para o caso de transações não concluídas. Neste caso (transações em progresso quando do fim da aplicação) as informações das transações pendentes se perdem.

Todos os identificadores únicos são fornecidos pelo agente (o que seria de se esperar, visto que é ele quem implementa as funções).

Assim, o uso da ARM-API pode ser esquematizado da seguinte forma:

```
d_apl = arm_init( “apl X”, “usu X” )
      id_trans_A = arm_getid( “Trans A”, “transacao que faz A” )
      id_trans_B = arm_getid( “Trans B” )
      id_trans_C = arm_getid( “Trans C”, “transacao que faz C quando
não faz D” )

      id_inst_trans_A = arm_start( id_trans_A )
            /* Faz um  monte de coisas da transação  “A”,
            entre elas chamar outras transações */

            id_inst_trans_B = arm_start( id_trans_B )
            stat_B = transacao_B( … )
            arm_stop( id_inst_trans_B, stat_B )

            id_inst_trans_C_1 = arm_start( id_trans_C )
            stat_C = transacao_C( … )

            se( stat_C … )
                  {
                        id_inst_trans_C_2 = arm_start( id_trans_C )
                              ….
                              arm_update( id_inst_trans_C_1 )
                                    stat_C = ….
                        arm_stop( id_inst_trans_C_2, stat_C )
                  }
            arm_stop( id_inst_trans_C_1, stat_C )

      arm_stop( id_trans_A , stat_A )
arm_end( id_apl )
```

4.1.3.2 ARM-API v2

A grande diferença conceitual entra a v1 e a v2 pode ser colocada da seguinte forma: A v1 permitiria observar o quê está ocorrendo; a v2 permitiria observar porquê está ocorrendo. A v2 permite estabelecer coisa tais como uma correlação entre transações “pai-e-filha”, métricas específicas para uma transação e informações de diagnóstico. A v2 é compatível retroativamente com a v1, ou seja, não é necessário recompilar “aplicações v1” para funcionarem com “agentes v2”.

Uma situação para a v2 seria a apresentada na figura 4.2.

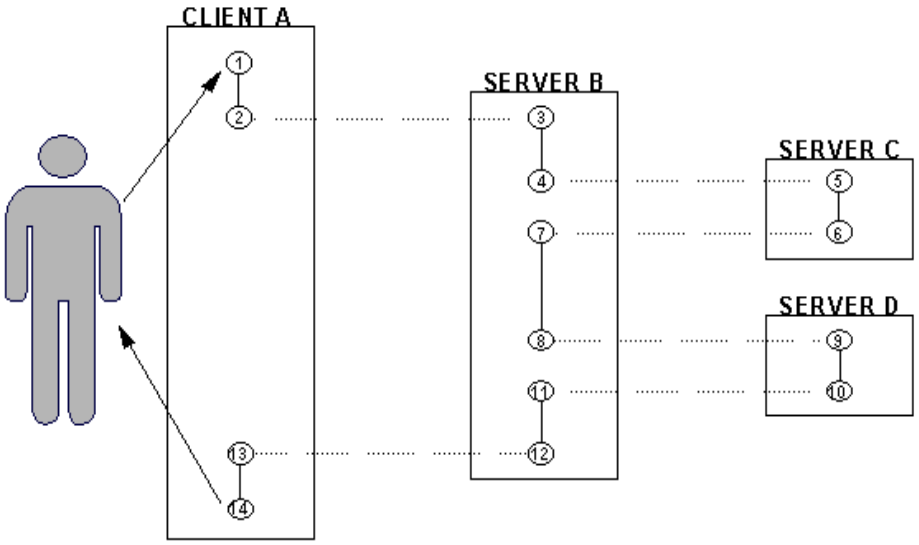


Figura 4.2: O problema das transações aninhadas

Observe que, do ponto de vista do usuário, o que interessa é o trecho 1-14. Este pode ser medido com os recursos já disponíveis na v1 (e todos os outros também, supondo que a programação tenha sido feita corretamente). O problema é que a correlação entre as transações não é registrada. Isto tornaria a tarefa de análise de gargalos bem mais complexa, já que as transações apareceriam “independentes” umas das outras. A v2 fornece meios de descrever a correlação entre elas. A correlação é estabelecida na chamada da arm_start. A transação originadora, ao chamar a arm_start, solicita um identificador de correlação. O agente

pode ou não fornecer o identificador. No caso de “agentes v1” ou quando existe uma política que bloqueia este mapeamento o identificador não é fornecido. De forma análoga, a transação originada pode fornecer um identificador de correlação para a arm_start, estabelecendo assim o relacionamento “pai-filho”. A figura 4.3 resume este processo:

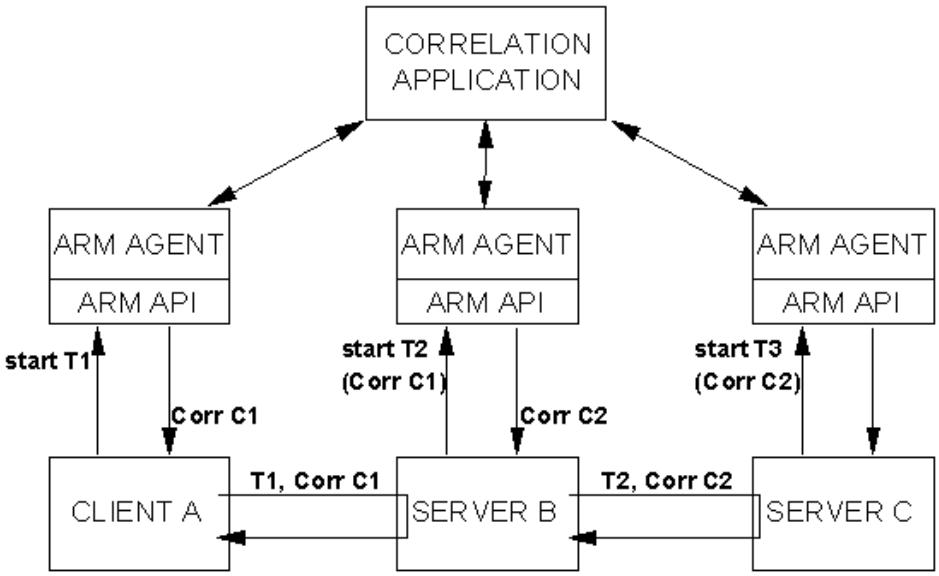


Figura 4.3: Controle de transações na ARM-API v2

A aplicação de correlação é a encarregada de identificar as correlações a partir dos identificadores e pode também filtrar as correlações que interessam segundo um dado critério (sobre os identificadores de correlação).

A capacidade de definição das métricas da v2 permitiria, por exemplo, saber quantos bytes (ou registros ou...) foram processados numa transação. Dessa forma, seria possível quantificar os recursos alocados numa dada transação. Correlacionando estes recursos com o tempo de resposta seria possível identificar a causa dos “gargalos” internos de uma transação e alterar esta alocação para melhorar a performance. Os parâmetros de definição das métricas são definidos na

arm_getid (e não podem ser posteriormente alterados) e passados nas arm_start, arm_update e arm_stop.

A v2 também fornece um mecanismo para que as aplicações passem informações de diagnóstico associadas às transações. São usadas as mesmas funções da definição de métricas e os mesmos campos disponíveis, mas com outra semântica.

4.1.4 **Considerações**

Como pontos positivos da solução ARM-API podemos citar:

a) Permite alto grau de granularidade na análise;

b) É *vendor independent*; a solução não é proprietária e está bem definida e padronizada, podendo ser usada (e portada) com facilidade em qualquer plataforma;

c) Ela se encaixa facilmente numa estrutura SNMP já existente, podendo ser vista como “mais uma MIB”, mas endereçando o que realmente interessa ao usuário: as aplicações.

Como pontos negativos da solução ARM-API podemos citar:

a) Exige agentes capazes de suportar a API. E esses agentes não são amplamente disponíveis. Aliás, este é um dos grandes problemas da solução; apesar de lançada em 1996, a solução até hoje não “decolou”. Algumas razões para isso talvez estejam nos itens abaixo;

b) Implica em reescrita das aplicações. Ora, como se sabe, reescrita de código é uma operação cara. Além disso, mesmo para aplicações novas, ela demandaria mais esforço e atenção do programador e maiores custos de desenvolvimento e manutenção. Isso é particularmente sensível no caso da v2, onde o uso dos recursos de correlação exige atenção na programação. Afinal, o relacionamento entre as transações é estabelecido “manualmente”; erros de programação (inseridos por alterações evolutivas, por exemplo) podem prejudicar seriamente a análise de co-relacionamentos, inserindo ainda mais confusão um quadro já confuso;

c) Uma coisa que nos pareceria interessante seria uma API para “ligar/desligar” a monitoração (ou talvez definir níveis de detalhamento da monitoração). Assim seria possível só ativar a monitoração (e seu respectivo ônus) quando fosse realmente necessário e/ou apenas para determinados níveis de utilização;

d) Embora as API’s em si sejam “leves”, o agente pode não ser. Se o agente estiver na mesma plataforma da aplicação pode ocorrer disputa de recursos que degrade a aplicação. Se o agente for “externo à aplicação” o problema é “empurrado” para a rede, continuando a existir sobre outra forma;

e) Parece duvidoso que a solução escale bem para o ambiente distribuído atual, “*webizado*”. A solução funcionaria para aplicações que fazem uso de *Web services* disponíveis em sites da Internet? Mesmo que esses web services também usassem a API, como ficaria a coleta dos dados? Que agente os recolheria? Como a aplicação de gerência os acessaria?

Resumidamente, a solução é interessante, permite realmente grande granularidade e oferece muitos recursos para monitoração de aplicações. Mas os custos parecem muito altos para sua efetiva difusão e utilização, além das incertezas quanto a escalabilidade e adequação para o novo paradigma de aplicações distribuídas. Mas acreditamos que ela, combinada com outras ferramentas, pode ser bastante efetiva no que se propõe.

4.2 APPLICATION PERFORMANCE MEASUREMENT (APM) MIB

4.2.1 **O que é**

É uma MIB que “estende” a RMON-MIB para permitir analisar a *performance* de aplicações do ponto de vista do usuário final. Ou seja, usando esta MIB seria possível medir a qualidade do serviço disponibilizada ao usuário final por uma aplicação, sempre levando em conta que, do ponto de vista do usuário, o que interessa são disponibilidade e “responsividade” (tempo de resposta).

A MIB está definida na RFC 3729 e basicamente introduz alguns novos grupos (com suas respectivas tabelas) que coletam dados sobre aplicações.

Uma abordagem detalhada da APM-MIB pode ser vista em [12]. Neste trabalho apresentaremos apenas uma visão sintética baseada no documento citado, objetivando fornecer uma base para as considerações que apresentaremos sobre a solução.

4.2.2 **Arquitetura**

A arquitetura básica da solução APM-MIB, como já vimos, estende a RMON-MIB (cria mais uma MIB); de resto, usa a solução SNMP/RMON já existente. Esta extensão introduz novos grupos e apresenta novos conceitos que são usados na solução.

4.2.2.1 Conceitos usados pela solução

Os novos conceitos de interesse no escopo deste trabalho são:

a) *Métricas*: A rigor, métricas não são um novo conceito criado por esta solução. O que a solução efetivamente propõe é definir métricas focadas nas duas "categorias" básicas que realmente interessam ao usuário final: disponibilidade (*availability*) e "responsividade" (ou tempo de resposta – *responsiveness*). Além disso, a solução define os domínios (conjuntos de valores válidos) de cada métrica.

b) *Transação*: Uma transação no contexto da APM-MIB é toda ação iniciada pelo usuário que causa a ativação (e conseqüente conclusão) de uma função de processamento distribuído. Por exemplo, clicar num botão de um formulário de consulta web: ao clicar no botão o usuário inicia um processo, que é completado quando uma resposta é devolvida ao usuário. A transação é a entidade mensurada pela APM-MIB. A métrica de disponibilidade é a mesma para qualquer transação. Já a métrica de responsividade (*responsiveness type*) define três possíveis domínios:

i. *Transaction-oriented*: A métrica é tempo de resposta (em ms). Por exemplo, é o tempo entre o usuário submeter um formulário web e receber uma resposta;

ii. *Throughput-oriented*: A métrica é a de quantidade de informação transmitida no tempo (kb/s). É o caso típico de transferências de arquivo;

iii. *Streaming-oriented*: A métrica é a razão entre o tempo em que o serviço ficou degradado e o tempo total de transação (ppm – partes por milhão ou porcentagem). São aplicações do tipo *streams* de vídeo, que usualmente trabalham com uma taxa constante mas que sofrem degradação/interrupção quando não há capacidade de atender esta taxa.

c) *Identificação de uma aplicação:* Para fins de referência (principalmente em relatórios) uma aplicação é definida por uma tupla (aplicação, servidor, cliente). Um relatório mostrará a identificação da aplicação (a tupla citada), uma medida de disponibilidade e uma de responsividade. Lembrar que a responsividade será uma das três mencionadas anteriormente (transaction, throughput, streaming). A disponibilidade indicará sucesso ou não na realização da transação.

d) *Agregação de resultados:* É a capacidade de sumarizar dados para emissão de relatórios, agregando itens de um relatório segundo um critério específico. Existem quatro possíveis tipos de agregação, cada um deles gerando um número menor de registros resultantes (atentar que só faz sentido agregar métricas iguais):

i. *Flows*: É o tipo mais simples. Todas as aplicações com os mesmo valores de (aplicação, servidor, cliente) são agrupadas;

ii. *Clients*: Todas as aplicações com os mesmo valores de (aplicação, cliente) são agrupadas. Esta agregação gera menor número de resultados que a anterior;

iii. *Servers*: Todas as aplicações com os mesmo valores de (aplicação, servidor) são agrupadas;

iv. *Applications*: Todas as aplicações com os mesmo valores de (aplicação) são agrupadas.

e) *Application protocol e verb:* Application protocol é o termo usado na RFC para descrever a aplicação. Este termo tanto pode se referir a um protocolo já conhecido (HTTP, por exemplo) quanto a uma aplicação definida pelo usuário. Um verb é um dos possíveis comandos implantados por um application protocol (o *get* do HTTP, por exemplo). Os identificadores de protocolo da RMON-2 definem protocolos até o nível de aplicação mas não se aprofundam na aplicação em si. Para permitir identificar os diferentes tipos de operações realizados por uma aplicação (um *download* ou uma consulta, por exemplo) é importante ter um identificador de protocolo para cada um destes "verbos". Estes identificadores podem ser os já existentes na MIB RMON-2 *applicationDir* ou podem ser dinâmicos; ambos os casos estão relacionados ao *appLocalIndex* (que veremos mais adiante).

f) *Push Model:* Refere-se ao mecanismo de transferência das informações do agente para a gerência. Como a proposta da APM-MIB é monitorar uma aplicação até o usuário final, pode ser necessário instalar agentes nas estações usadas pelo usuário. Ora, neste caso estaremos falando de máquinas que apresentam uma série de características que inviabilizam um modelo de *polling* para coleta de dados. Neste caso, adota-se o modelo de push, onde os agentes de usuário enviam dados em momentos por eles definidos. A RFC explicita as condições e os objetos que devem ser enviados quando de um push.

4.2.2.2 Estrutura da APM-MIB

Os novos grupos introduzidos são:

a) *APM Application Directory Group:* Contém os objetos de configuração para cada aplicação (application protocol) e verbo (verb) monitorado. Composto da tabela *apmAppDirTable*; esta tabela é indexada pelo *AppLocalIndex*. A tabela é populada pelo agente com todas as aplicações/verbos de qualquer resposivenessType que ele saiba monitorar. As entradas são *read/write* (o que permite modificar parâmetros desta tabela, inclusive desabilitando as entradas) mas não podem ser removidas nem inseridas. A apmAppDirTable está sempre associada com as tabelas *apmHttpFilterTable* e

apmUserDefinedAppTable (do grupo APM User Defined Application), registrando qualquer alteração feita nestas outras duas. Ou seja, caso seja criado um novo filtro HTTP ou aplicação definida pelo usuário, uma entrada equivalente será criada na apmAppDirTable. Uma aplicação pode ter sido definida no diretório de aplicações RMON2 (grupo protocolDir) ou numa outra tabela da APM-MIB (apmHttpFilterTable ou apmUserDefinedAppTable); em qualquer caso, um índice AppLocalIndex único e distinto será atribuído. Esta tabela é básica para todos os relatórios e alertas gerados pelo agente e deve estar configurada para refletir as características de operação do ambiente sendo monitorado, de modo a permitir analisar o desempenho da aplicação nas condições em que ela irá efetivamente ser executada.

b) *APM User Defined Application Group*: Registra aplicações/verbos que não estão na protocolDirTable. Consiste nas tabelas apmHttpFilterTable e apmUserDefinedAppTable. Cada entrada numa dessas tabelas irá gerar uma ou mais entradas correspondentes na apmAppDirTable (uma para cada responsivenessType que o agente souber monitorar). A apmHttpFilterTable permite criar uma aplicação “virtual” que medirá a performance de certas páginas web (por exemplo, uma aplicação de *e-commerce*). A apmUserDefinedAppTable é onde são registradas as aplicações que não tem um protocolo definido (ou definível) na protocolDirectory. Assim, é aqui que são inseridas as aplicações específicas do usuário (as que implementam a lógica de negócios, por exemplo). Esta tabela seria então a mais importante do ponto de vista que estamos abordando neste trabalho. Mas é aqui que reside o maior problema da APM-MIB: a maneira do agente monitorar estas aplicações (logo, dele coletar resultados para as tabelas da APM-MIB) é *implementation-dependent*. Ou seja, seria necessário escrever um agente específico para uma aplicação de negócio. É certo que isso faz sentido e que, uma vez existindo o agente, todo framework SNMP/RMON/RMON2/APM estaria disponível. Mas recaímos no problema da ARM-API: (re)escrita de código. Na seção de considerações nos estenderemos mais sobre este aspecto, apontando algumas diferenças e sinergias entre esta solução e a ARM-API.

c) *APM Report Group*: Define relatórios de performance de aplicações. Os relatórios permitem agregar informações, gerando sumários de mais alto nível (inclusive estatísticos). Consiste nas tabelas *apmReportControlTable* e *apmReportTable*. A apmReportControlTable controla a criação de conjuntos de relatórios agregados da performance de uma aplicação. A apmReportTable contém os dados gerados pelas agregações definidas.

d) *APM Transaction Group*: Mostra as transações correntes e as recentemente acabadas, junto com suas respectivas métricas. Uma observação importante aqui é que as entradas neste grupo podem sumir muito rapidamente, já que algumas transações podem executar muito rapidamente (assim como outras podem demorar bastante). Assim, é preciso critério para usar este grupo ou podem-se perder informações importantes sobre transações “rápidas” mas significativas. Existe um parâmetro que permite ajustar a retenção das informações mas, dependendo do caso, pode ser necessário um certo esforço para determinar este valor. Por outro lado, transações “longas” estarão disponíveis para consulta por toda sua duração. Isso é bom porque permite acompanhar a performance de uma transação mesmo antes de seu término (já que as notificações de fim de transação só vêm ao fim da transação, quando mais nada pode ser feito). Este grupo é basicamente a tabela *apmTransactionTable*.

e) *APM Exception Group*: Gera notificações para aplicações que ultrapassaram determinados valores-limite (*thresholds*). A tabela *apmExceptionTable* registra os valores-limite para as transações especificadas. Dois tipos de notificação podem ocorrer: *apmTransactionResponsivenessAlarm*, quando a responsividade de uma transação viola o *threshold* e *apmTransac-tionResponsivenessAlarm*, quando uma aplicação com *threshold* estabelecido falha (*abort*).

f) *APM Notification Group*: Contém as duas notificações que são emitidas quando os *tresholds* da *apmExcetionTable* são atingidos.

4.2.3 **Como funciona**

A RFC não é explícita sobre o mecanismo de preenchimento das tabelas. Alguns campos são preenchidos automaticamente pelo agente mas outros (até pelo

que já foi visto antes) deveriam ser preenchidos dinamicamente. Mas a RFC não explicita como; supomos que seja possível fazê-lo usando os mecanismos disponíveis no SNMP (SNMP Set), o que abriria caminho para que estes valores pudessem ser preenchidos diretamente pelo responsável pela monitoração.

4.2.3.1 Exemplo de agregação

Vejamos através de um exemplo, como funcionaria a agregação de relatórios.

Tabela 4.1: Exemplo de tabela de transações

Tabela de transações					
ID transação	**Application protocol**	**Cliente**	**Servidor**	**disponibilidade**	**Responsividade (s)**
1	http	José	web01	0	-
2	http	José	mail01	1	12
3	http	José	apl01	1	7
4	http	José	web01	1	5
5	smtp	José	pop01	1	12
6	http	Sonia	web01	1	3
7	SAP/R3	Sonia	sap01	1	19
8	smtp	Sonia	pop01	1	16
9	http	João	mail01	1	18

Tabela 4.2: Relatório gerado por *flow aggregation*

Flow aggregation							
Application protocol	**Cliente**	**Servidor**	**contador**	**disponibilidade**	**Responsividade**		
					méd	**mín**	**Máx**
http	José	web01	2	1	5	5	5
http	José	mail01	1	1	12	12	12
http	José	apl01	1	1	7	7	7
smtp	José	pop01	1	1	12	12	12
http	Sonia	web01	1	1	3	3	3
SAP/R3	Sonia	sap01	1	1	19	19	19
smtp	Sonia	pop01	1	1	16	16	18
http	João	mail01	1	1	18	18	18

Tabela 4.3: Relatório gerado por *client aggregation*

Client aggregation						
Application protocol	**Cliente**	**contador**	**disponibilidade**	**Responsividade**		
				méd	**mín**	**Máx**
http	José	4	3	8	5	12
smtp	José	1	1	12	12	12
http	Sonia	1	1	3	3	3
SAP/R3	Sonia	1	1	19	19	19
smtp	Sonia	1	1	16	16	16
http	João	1	1	18	18	18

Tabela 4.4: Relatório gerado por *server aggregation*

Server aggregation						
Application protocol	**Servidor**	**contador**	**disponibilidade**	**Responsividade**		
				méd	**mín**	**Máx**
http	web01	3	2	4	3	5
http	mail01	2	2	15	12	18
http	apl01	1	1	7	7	7
smtp	pop01	2	2	14	12	16
SAP/R3	sap01	1	1	19	19	19

Tabela 4.5: Relatório gerado por *application aggregation*

Application aggregation					
Application protocol	**contador**	**Disponibilidade**	**Responsividade**		
			méd	**mín**	**Máx**
http	6	5	9	3	18
smtp	2	2	14	12	16
SAP/R3	1	1	19	19	19

4.2.3.2 Relacionamento entre *apmAppDirTable*, *apmHttpFilterTable*, *apmUserDefinedTable* e *AppLocalIndex*

Como vimos, existe uma relação estreita entre as tabelas *apmAppDirTable*, *apmHttpFilterTable* e *apmUserDefinedTable* e os valores de *AppLocalIndex*. Quando entradas são preenchidas na *apmHttpFilterTable* e/ou na *apmUserDefinedTable* isto é automaticamente refletido com ***n*** entradas correspondentes na *apmAppDirTable*. Existem mais entradas na *apmAppDirTable* porque existe uma entrada para cada métrica (*responsivenessType*) que o agente saiba medir (em teoria, então, poderiam haver três entradas para uma aplicação. Na prática, dificilmente uma aplicação teria três métricas diferentes).

O *AppLocalIndex* é definido pelo agente de modo a ser único e unívoco. Assim, se um *application protocol/verb* (ou uma aplicação...) registrado for um dos já definidos na RMON2 *protocolDirTable*, o *AppLocalIndex* deste será igual ao índice na RMON2 *protocolDirTable* e este valor não poderá mais ser associado a nenhuma outra entrada. Da mesma forma para entradas na *apmHttpFilterTable* e *apmUserDefinedTable*: uma entrada numa dessas tabelas receberá um *AppLocalIndex* ainda não usado, diferente de qualquer índice na RMON2 *protocolDirTable* e que não será atribuído a mais ninguém. Resumindo: um *AppLocalIndex* estará definido em uma e apenas uma das tabelas RMON2

protocolDirTable , *apmHttpFilterTable* ou *apmUserDefinedTable* e será registrado também na *apmAppDirTable*, indicando a aplicação sendo monitorada.

4.2.3.3 *apmHttpFilterTable*

Esta tabela é ideal para medir aplicações web. Alguns campos relevantes são:

a) *apmHttpFilterServerAddress*: O endereço de rede origem das transações;

b) *apmHttpFilterURLPath*: URL envolvida nas transações;

c) *apmHttpFilterMatchType*: O algoritmo usado para comparar a URL atual com a especificada em (b). Pode ser *exact*, *stripTrailingSlash* ou *prefix*;

d) *apmHttp4xxIsFailure*: Uma *flag* para indicar se transações http com códigos de retorno entre 400 e 499 devem ser tratadas como falha ou não.

Em resumo, podem-se definir quais protocolos/verbos de qual URL se desejam monitorar.

4.2.3.4 *apmUserDefinedAppTable*

Esta tabela é para as aplicações específicas do usuário e que não se encaixam nos protocolos já conhecidos. Ela é que deve ser “customizada” por um agente específico para permitir a monitoração de aplicações proprietárias. Alguns campos relevantes são:

a) *apmUserDefinedAppParentIndex*: O *protocolDirLocalIndex* que indica o protocolo de mais alto nível do qual esta aplicação é filha. Não há esclarecimentos na RFC que nos permitisse entender qual a relevância disso, ainda mais que uma aplicação pode ser bem complexa de descrever nestes termos;

b) *apmUserDefinedAppApplication*: É um nome legível para aplicação, um rótulo;

c) *apmNameTable*: É uma tabela com todas as identificações da máquina. No final, esta tabela relacionará nome de host, nome do usuário e endereços do cliente. Esse campo mostra o cuidado com as características dos clientes (junto com o mecanismo de *push*) já que clientes podem, por exemplo, usar DHCP, o que levaria a possíveis mudanças de IP do cliente. Existe uma série de recomendações na RFC em como preencher, manter e limpar esta tabela já que a consistência dela é essencial para os relatórios.

4.2.3.5 *apmAppDirTable*

Chegamos afinal na tabela mais representativa deste grupo. É nela que são registrados os dados relativos às aplicações e quais aplicações serão efetivamente monitoradas, bem como os *bucket boundaries*. *Bucket boundaries* são limites dos "escaninhos" do histograma de análise da aplicação. Ou seja, eles definem um número de "faixas" (de uma dada métrica) em que podem ser agrupados os valores monitorados de uma aplicação (são possíveis até 6 faixas por aplicação). Vejamos um exemplo: suponhamos uma aplicação http com as seguintes faixas definidas (todos os valores em ms)

bucket 1: 500

bucket 2: 1000

bucket 3: 2000

bucket 4: 5000

bucket 5: 15000

bucket 6: 60000

e com os seguintes tempos de resposta (em ms): 377, 8645, 1300, 487, 1405, 775, 1115, 850, 945, 1054, 7745, 9380. A distribuição nas faixas será de (em número de eventos):

bucket 1: 2 (377 e 487)

bucket 2: 3 (775, 850, 945)

bucket 3: 4 (1300, 1450, 1115, 1054)

bucket 4: 0

bucket 5: 3 (8645, 7745, 9380)

bucket 6: 0

Esta tabela deve ser preenchida pelo agente e embora suas entradas sejam read/write, as entradas em si não podem ser nem inseridas nem removidas pela estação de gerência. Alguns campos relevantes são:

a) *apmAppDirAppLocalIndex*: Já falamos sobre ele e suas particularidades;

b) *apmAppDirResponsivenessType*: Também já sabemos o que ele significa. Relembramos que podem existir aplicações com mais de uma métrica e que neste caso haverá uma entrada nesta tabela para cada métrica. Por exemplo, um download de arquivo: podemos medir o tempo (transaction) e a vazão (throughput);

c) *apmAppDirResponsivenessBoundary*[1..6]: São os *buckets boundaries* já vistos anteriormente. Quando um valor destes muda, todos os dados relativos a este escaninho devem ser apagados das tabelas que registram as métricas;

d) *apmAppDirConfig*: Indica se o agente deve monitorar esta entrada ou não. Se o estado mudar de *on* para *off*, todos os dados relativos a esta aplicação devem ser apagados de todas as tabelas que registram as métricas.

Tabela 4.6: Exemplo de RMON2 protocolDir

protocolDirectory		
ID	**parâmetros**	**LocalIndex**
www	--	1
www:get	--	2
SAP/R3	--	3

Tabela 4.7: Exemplo de apmHttpFilterTable

apmHttpFilterTable				
Index	**AppLocalIndex**	**Server**	**URLPath**	**MatchType**
5	20	www.nce.ufrj.br	/mot_cn	prefix

Tabela 4.8: Exemplo de apmAppDirTable

apmAppDir		
AppLocalIndex	**ResponsivenessType**	**Config**
1	transaction	on
1	throughput	on
2	transaction	on
2	throughput	on
3	transaction	on
20	transaction	on
20	throughput	on

4.2.3.6 *apmReportControlTable* e *apmReportTable*

Controla os relatórios dos dados obtidos, incluindo a agregação. Alguns campos relevantes são:

a) *apmReportControlDataSource*: É a identificação da origem dos dados sendo reportados. Se os dados vierem de um *probe*, identifica a porta ou interface que está recebendo os dados do probe. Se os dados estão sendo coletados localmente, deve identificar a interface primária que recebe os dados;

b) *apmReportControlAggregationType*: Define qual será a agregação feita no relatório. As métricas de responsividade já sabemos como são. A de disponibilidade é “0 ou 1” (“rodou/não rodou”) para uma transação individual e a média quando temos um conjunto de transações;

c) *apmReportControlInterval*: O tempo de coleta entre cada consolidação. É o tempo em que os dados são coletados antes de ser feita uma agregação;

d) *apmReportServerAddress*: É o endereço de rede do servidor envolvido na transação sendo reportada (lembrar do item 4.2.2.1.c);

e) *apmReportTransactionCount*: O número de transações agregadas neste relatório ou nesta “linha de um relatório”;

f) *apmReportSuccesfulTransactions*: O número de transações bem sucedidas agregadas nesta linha (ver 4.2.3.6.b);

g) *apmReportResponsivenessMean*: Média de todas as “responsividades” de uma transação. O relatório também dará a máxima e a mínima responsividade;

h) *apmReportResponsivenessB*[1..6]: Os buckets de novo, agora para os tempos de resposta.

4.2.3.7 *apmTransactionTabl*e

Mantém o registro das transações correntes ou recentemente encerradas. O significado de “recentemente” depende de um parâmetro que pode ser definido pela gerência. De novo, o registro da transação tem uma entrada para cada tipo de métrica que se está monitorando. Alguns campos relevantes são:

a) *apmTransactionServerAddress*: É o servidor envolvido na transação;

b) *apmTransactionAge*: Tempo decorrido desde o início da transação;

c) *apmTransactionSuccess*: Relacionado à métrica de disponibilidade (até o momento presente). Se uma transação falha, o estado dela não pode mais reverter para “disponível”.

4.2.3.8 *apmExceptionTable*

Implementa os filtros que permitem que a estação de gerência seja imediatamente avisada de condições críticas, sem esperar por *polling*. É muito importante para o modelo de *push* e/ou para casos de muitos agentes com poucas transações (pois evita *polling* de agentes que podem não ter nada para dizer). Ressalte-se que só é possível falar numa exceção após o fim da transação (seja fim “normal” ou por erro). Alguns campos relevantes são:

a) *apmExceptionResposivenessComparison*: Diz qual a relação de ordem que deve ser usada para comparar o *threshold* e o valor atual. Os valores podem ser *none*, *greater* e *less*;

b) *ApmExceptionResposivenessTreshold*: É o *threshold* propriamente dito (valor dele na métrica em questão);

c) *ApmExceptionUncessfulException*: Gera exceção quando a transação falha (é do tipo “on/off”);

d) *ApmThroughputExceptionMInTime*: É quase que exclusiva para transações *throughput-oriented*, já que estas transações são muito sensíveis a variações bruscas em pequenos intervalos. Por exemplo, a perda de apenas um pacote na transferência de um arquivo pequeno pode aumentar em muito o tempo de execução da transação (por causa da retransmissão), afetando o *throughput*. Também usada para dar “equilíbrio” à medição de *throughput*. Isso porque, neste tipo de transação existe uma componente “variável” (tamanho do arquivo a transferir, por exemplo) e uma fixa (*overhead* do protocolo). Para transferências

"pequenas" a parte fixa pode pesar muito, distorcendo o tempo efetivamente usado na transação. Este campo controlará o número mínimo de segundos que a transação precisará exceder o limite antes de gerar uma exceção. Colocando essa valor "igual" ao parte fixa poder-se-ia medir com mais precisão o *throughput* real da transação.

4.2.3.9 *apmNotifications*

É um objeto que representa a notificação gerada quando a exceção ocorre. São dois tipos de notificação:

a) *apmTransactionResponsivenessAlarm*: Notificação para casos em que o *threshold* é excedido. Contém um índice que permite relacionar a notificação com a entrada na *apmExceptionTable* que a motivou. A RFC diz que os agentes devem incluir mais informação para esclarecer melhor as razões da notificação (por exemplo, um dump dos recursos em uso no momento);

b) *apmTransactionUncessfulAlarm*: Análogo para o caso de transações que falham.

4.2.4 **Considerações**

Como pontos positivos da solução APM-MIB podemos citar:

a) Ela se encaixa perfeitamente na "estrutura SNMP" já existente, mas endereçando o que realmente interessa ao usuário: as aplicações. E do ponto de vista que interessa ao usuário (tempo de resposta e disponibilidade);

b) Como a solução se baseia em agentes, pelo menos em teoria seria possível aplicá-la sem precisar reescrever a aplicação. Isso até é verdade em certos casos (aplicações web usando HTTP, por exemplo). Mas, como já vimos, não é genérica o suficiente para lidar com as aplicações de negócio do usuário. Ainda assim, caso os fabricantes decidissem fornecer suporte em suas plataformas de aplicação (em termos práticos: se os sistemas operacionais dos servidores onde as aplicações são executadas

permitissem “acoplar” pontos de interação com os agentes nas *syscalls* do sistema operacional), algo já poderia ser feito sem necessidade de reescrita;

c) Já que a solução usa o “framework SNMP”, as estações de gerência não teriam grande dificuldade em se adaptar, permitindo uma maneira uniforme de apresentar as monitorações feitas. No caso da ARM-API, para usar o framework seria necessário quase que “uma nova MIB para cada aplicação”. No caso da APM-MIB, as MIB’s já estão definidas (a dificuldade é “populá-las”);

d) A solução já está na condição de RFC, o que significa que ela tem boas chances de se tornar um “*standard* da Internet”, o que é ótimo em termos de independência de fornecedor (portabilidade);

e) A escalabilidade da solução é a mesma do SNMP.

Como pontos negativos da solução APM-MIB podemos citar:

a) Exige agentes capazes de suportar as aplicações específicas do usuário API. Pelas suas próprias características, estes agentes precisariam ser escritos (e mantidos) pelos próprios desenvolvedores da aplicação. Isso faz muito sentido mas implica em custo maior para as aplicações (mão-de-obra mais qualificada, ciclo de vida mais complexo, etc.);

b) Não existe uma arquitetura-padrão para a definição destes agentes específicos de aplicação. Ou seja, não existe uma maneira padrão destes agentes se comunicarem com a aplicação para coletar dados. Isso implica em dificuldade de padronização e curva de aprendizado dos programadores, em princípio, alta (o que significa mais custo...);

c) Como os agentes são específicos da aplicação, não existe escala para a produção de agentes “padronizados”; cada caso será um caso. Isto é custo...;

d) Da mesma forma que no caso da ARM-API, o agente pode não ser “leve”. E se ele estiver na mesma plataforma da aplicação, significa disputa de recursos que pode degradar a aplicação. Se o agente for “externo à aplicação”, o problema é “empurrado” para a rede, continuando a existir sobre outra forma. Mas, neste caso, a solução se aproveita das características já existentes do SNMP, que endereçam este problema;

e) Ao contrário da ARM-API, a solução parece escalar bem mas a mesma dúvida ainda persiste: como a solução funcionaria para aplicações que fazem uso de Web services disponíveis em sites da Internet? Como ficaria a coleta dos dados? Que agente os recolheria? Como a aplicação de gerência os acessaria? A solução óbvia seria encarar estes serviços externos como uma “caixa-preta”, computando apenas a “responsividade total” entre a chamado do serviço e seu retorno; isso é bastante razoável e atenderá bem ao que se pretende com a monitoração.

Assim, esta solução parece oferecer possibilidades interessantes. Ainda mais se combinada à idéia da solução anterior (ARM-API): uma API padrão que facilitasse a escrita de aplicações e agentes acoplados. Claro que esta solução híbrida ainda teria todos os problemas de custo apontados anteriormente. Mas não vemos como seja possível obter os dados de monitoração desejados sem investimento. E, neste caso pelo menos, o investimento seria “inicial”; na medida que muita coisa já estivesse pronta o problema poderia ser atacado onde ele realmente é mais complexo e mais necessário: na aplicação em si. Ressalte-se que não se diz que a ARM-API pode ser usada como está; há que se pensar se, como ela está definida, ela atende às necessidades da APM-MIB. Mas a idéia de uma API padrão para criar agentes APM-MIB parece interessante.

5 **CONCLUSÕES**

a) A combinação das abordagens APM-MIB e ARM-API parece ser uma boa abordagem, pelo menos no sentido de indicar uma direção para uma solução *vendor independent* para SLM;

b) Nos parece que o grande problema aqui é o custo de *software* envolvido, seja reescrevendo as aplicações seja escrevendo um agente. Não divisamos nenhum modo de evitar este custo, apenas de reduzi-lo;

c) Esta redução passaria pela definição de uma arquitetura padrão para a criação de agentes. Para gerência de ativos de rede a dependência do fornecedor para disponibilizar um agente/MIB acaba sendo aceita. Mas este esquema é prejudicial quando se trata de SLM para aplicativos, ainda mais os específicos e/ou "*home-made*";

d) Uma proposta para esta arquitetura padrão seria, como já dissemos, combinar a infra-estrutura da APM-MIB com a proposta da ART-API. Claro que existem muitos desdobramentos possíveis a partir dessa premissa;

e) Por exemplo, os agentes poderiam ser escritos no padrão de *Web services*, usando de todas as funcionalidades disponíveis para estes serviços. A API "dentro" da aplicação se comunicaria com estes "agentes Web service". Isso, claro, teria implicações no atual padrão de monitoração, o SNMP (já existem propostas sobre o uso de XML na descrição das MIB's ao invés de SMI). Outra opção seria a adoção dos padrões CORBA;

f) Observe que, no caso da adoção de um desses padrões, embora ainda continue existindo a necessidade de reescrita de aplicações, novas aplicações já poderiam ser escritas "prontas para monitoração SLM". E se ganharia em escala e curva de aprendizado, já que estes padrões (CORBA, Web services) são amplamente difundidos e apresentam bastante mão-de-obra disponível;

g) Quanto à compatibilidade com SNMP, o agente poderia facilmente atuar como "tradutor bilíngüe", oferecendo uma interface/linguagem para a API e SNMP com as estações de gerência, outros agentes ou *probes*;

h) É verdade que alguns destes padrões são considerados menos eficientes do ponto de vista de uso da rede que o SNMP (o SOAP, por exemplo, é visto como mais “pesado” que o SNMP). Mas este “peso” ficaria mais restrito à comunicação entre os *peers* aplicação (API) ⇔ agente; a função de tradução citada em (g) evitaria esse peso em toda a rede;

i) Existem ainda algumas outras características citadas para SLM de aplicações mas, em geral, elas nos parecem complexas demais para o momento. Por exemplo, o que é “usabilidade” de uma aplicação? Como medi-la? Características como essas envolvem um enorme grau de subjetividade e não têm sequer métricas definidas. Nos parece que as características citadas na APM-MIB são bastante apropriadas para o momento presente e permitem começar a construir uma infra-estrutura viável de SLM.

6 **REFERÊNCIAS**

[1] ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 10520**: informação e documentação. Citações em documentos:apresentação. Rio de Janeiro, 2002. 7p.

[2] _____. **NBR 14724**: informação e documentação: trabalhos acadêmicos: apresentação. Rio de Janeiro, 2002. 6 p.

[3] _____. **NBR 12225**: títulos de lombada. Rio de Janeiro, 1992. 2 p.

[4] UNIVERSIDADE FEDERAL DO RIO DE JANEIRO. SUB-REITORIA DE ENSINO PARA GRADUADOS E PESQUISA. **FATED**: Formato e apresentação de teses ou dissertações de pós-graduação. Rio de Janeiro, 1979. 26 p.

[5] LEWIS, Lundy. **Service level management for enterprise networks**. Boston; London: Artech House, 1999. 307 p.

[6] STADLER, Rolf; STILLER, Burkhard (ed.). **Active technologies for network and service management**. Berlin; Heidelberg; New York; Barcelona; Hong Kong; London; Milan; Paris; Singapore; Tokyo: Springer, 1999. 295 p.

[7] RIBEIRO, M. B.; GRANVILLE, L. Z.; ALMEIDA, M. J. B.; TAROUCO, L. M. R. **An Architecture to Monitor QoS in a Policy-Based Network**. Porto Alegre: UFRGS, 2003. Acesso em 09/08/2007.

[8] ADANEZ, X. G. **Service Level Agreement from the User Perspective**. Switzerland: Adventis Communications Engineering. Acesso em 09/08/2007.

[9] HAUCK, R.; RADISIC, I. **Service oriented application management — do current techniques meet the requirements?**. Carcóvia: Kluwer Academic Publishers, 2001. Acesso em 09/08/2007.

[10] CHATAUT, S.; MCCLELLAN, S.; GEMMILL, J. **Tools for application performance management**. Birmingham: University of Alabama at Birmingham. Acesso em 09/08/2007.

[11] JOHNSON, M. W.. **The Application Response Measurement (ARM) API, Version 2**. Tivoli Systems, 1997. Disponível em http://regions.cmg.org/regions/cmgarmw/marcarm.html. Acesso em 21/08/2007.

[12] WALDBUSSER, S. **Application Performance Measurement MIB**, RFC 3729. IETF: 2004. Disponível em http://www.faqs.org/rfcs/rfc3729.html. Acesso em 21/08/2007.