4
Desenvolvimento da Solução Utilizando Web Semântica

O capítulo 3 explorou a vulnerabilidade dos sistemas de gerenciamento de conhecimento. Os fatores limitantes foram analisados e identificados no estudo de caso, uma pesquisa de satisfação com os usuários do sistema, um levantamento dos principais indicadores associados ao sistema do estudo de caso e a descrição das metas e requisitos de um novo sistema de gerenciamento de conhecimento completou a fase de análise.

De acordo com as metas e requisitos descritos no capítulo anterior, um novo sistema de gerenciamento de conhecimento baseado nas tecnologias da Web Semântica será desenvolvido para suprir as principais deficiências do sistema atual: integração e pesquisa do conhecimento. Neste capítulo será apresentada uma solução seguindo esta abordagem para o estudo de caso.

Para desenvolver um novo sistema de gerenciamento de conhecimento, primeiro é necessário analisar os requisitos do sistema e em seguida modelar e desenvolver o novo sistema. Os requisitos já foram abordados na seção anterior e neste capítulo as etapas de modelagem e desenvolvimento do novo sistema serão expostas.

4.1.
Casos de Uso

Um caso de uso é uma descrição de um conjunto de seqüência de ações, inclusive, variantes que um sistema executa para produzir um resultado de valor observável por um ator. A figura 9, apresentada a seguir, ilustra o diagrama de caso de uso do novo sistema de gerenciamento de conhecimento.

O novo sistema de gerenciamento de conhecimento pode ser dividido em quatro ações principais executadas por quatro sujeitos diferentes. Como explanado anteriormente, a *TEC* é responsável por registrar novos alarmes, o *ITCM* é responsável por inventariar o *hardware* e *software* dos elementos monitorados, o Especialista deve registrar os conhecimentos necessários para tratamento de

alarmes, o CORS é o usuário do conhecimento registrado pelo especialista, ele deve consultar os conhecimentos necessários para ter capacidade de tratar os alarmes do ambiente de monitoração.

Todas as consultas executadas pelo CORS serão suportadas pela Web Semântica para contornar os problemas levantados no capítulo 3. As consultas serão detalhadas na seção 4.7.

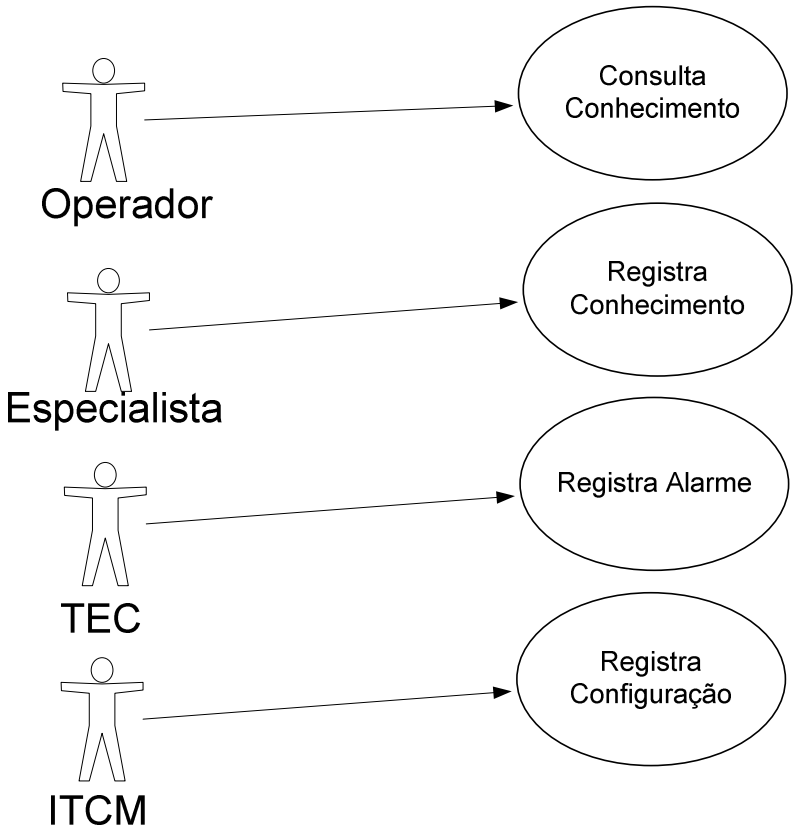

Figura 9 – Diagrama de Casos de Uso.

1. Consulta Conhecimento

Ator: Operador do CORS.

Propósito: Permitir que os operadores do CORS tenham o conhecimento necessário para tratar um alarme.

Pré-condições: -

Pós-condições: Os conhecimentos relacionados aos parâmetros semânticos procurados são exibidos.

Roteiro Principal:

1. O Operador seleciona a opção de consulta.
2. A aplicação solicita os parâmetros que serão usados para buscar o conhecimento.
3. O Operador insere os parâmetros para busca.
4. A aplicação executa uma busca Semântica (SPARQL) na camada de integração.
5. A camada de integração traduz a busca Semântica numa *query* (SQL) e busca a informação na camada de dados.
6. A camada de integração converte a informação encontrada na camada de dados para forma Semântica (RDF) e envia para a aplicação.
7. A aplicação converte as triplas RDF num formato legível para o usuário e exibe o resultado da busca para o Operador.

2. Registra Conhecimento.

Ator: Especialista.

Propósito: Permitir que o especialista registre seu conhecimento sobre um determinado tipo de incidente. Esse conhecimento poderá ser usado pelos operadores do CORS no futuro.

Pré-condições: -

Pós-condições: O conhecimento do especialista é registrado no sistema.

Roteiro Principal:

1. O Especialista seleciona a opção de registro de conhecimento do *KMS*.
2. O *KMS* exibe a interface de edição e solicita os dados que serão usados para registrar o conhecimento.
3. O Especialista registra o conhecimento no *KMS*.

4. O *KMS* exibe o conhecimento inserido para o Especialista e solicita a confirmação da operação de inserção.
5. O Especialista confirma a operação.
6. O *KMS* registra o conhecimento.

Roteiro Alternativo:

5. O Especialista não confirma a operação de inserção.
6. O *KMS* volta para o passo 2.

3. Registra Alarme.

Ator: TEC – Tivoli Enterprise Console.

Propósito: Permitir que o CORS identifique incidentes de infra-estrutura de TI.

Pré-condição: Servidor ou serviço indisponível ou fora da operação normal.

Pós condições: Gerar e registrar alarme.

Roteiro Principal:

1. Sistema de monitoração envia evento para TEC.
2. TEC avalia e classifica o evento em algum tipo de alarme.
3. TEC exibe o alarme na console, registra incidente no ARS e registra o incidente em sua base de alarmes.

4. Registra Configuração.

Ator: ITCM – IBM Tivoli Configuration Manager.

Propósito: Viabilizar o processo de gerenciamento de configuração dentro da empresa.

Pré-condição: -

Pós-condições: Registrar configuração de hardware e software das estações, notebooks e servidores.

Roteiro Principal:

1. Administrador do ITCM inicia o processo de inventário de hardware e software para um determinado grupo de máquinas.
2. ITCM coleta as informações de inventário dos nós gerenciados.
3. ITCM registra a informação de hardware e software dos nós gerenciados na base de configuração.

4.2. Modelo de Entidade e Relacionamento

O modelo de entidades e relacionamentos é um modelo abstrato cuja finalidade é descrever, de maneira conceitual, os dados a serem utilizados em um sistema ou que pertencem a um domínio. Nesta seção, vamos estudar os dados manipulados pelo novo sistema de gerenciamento de conhecimento.

Esta seção servirá de referência para construção da ontologia descrita na seção 4.5. Normalmente na implementação de um sistema baseado em Web Semântica esta etapa não é necessária, pois a ontologia já descreve um domínio assim como o MER. Alem disso, a ontologia também adiciona formalismos semânticos ausentes no MER e que trazem uma série de benefícios, detalhados nas seções 3.1 e 3.2. Entretanto, o sistema atual já está em produção e não adota nenhuma ferramenta da Web Semântica.

Neste caso, usamos o processo inverso, onde, a partir dos modelos relacionais das três bases distintas presentes no estudo de caso, descrevemos facilmente um único MER correspondente. Em seguida, usando este MER como referência, definimos uma ontologia.

No estudo de caso, o CORS precisa de informações de três fontes diferentes para tratar os alarmes recebidos. Para consultar essas informações, o operador precisa interagir com três sistemas diferentes, não existe nenhuma integração dos dados.

O modelo de entidade e relacionamento (MER), da figura 10, representa os dados necessários para o CORS tratar os alarmes. O modelo é abstrato e contém

informações das diferentes bases de dados. Apenas os dados essenciais para tratamento dos alarmes são modelados, as bases de configuração, alarmes e conhecimento podem manter outras informações irrelevantes para o novo sistema de gerenciamento de conhecimento proposto.

Um servidor (*server*) pode ter vários processadores, por outro lado, um mesmo modelo de processador pode estar instalado em vários servidores diferentes, caracterizando uma relação n:m. O relacionamento é o mesmo entre servidor x *software* e servidor x memória. Neste cenário, não é relevante a informação dos discos rígidos dos servidores, mas dos sistemas de arquivos (*file system*), que traz informações como espaço livre e espaço ocupado. Um servidor pode ter vários sistemas de arquivos. Cada servidor é de responsabilidade de uma equipe, que pode ser responsável também por outros servidores.

Um procedimento (*procedure*) é escrito por uma equipe de especialistas que, normalmente, escreve vários procedimentos para tratamento de alarmes. Os procedimentos podem estar associados a um servidor específico ou a vários servidores com características em comum, como sistema operacional ou aplicação. Os procedimentos podem não estar associados com os servidores diretamente, mas sim aos tipos de alarmes.

Um alarme (*alarm*) de queda de serviço pode ser tratado sempre da mesma maneira independente do servidor, por exemplo, um alarme de queda de serviço de um servidor pode ser sempre tratado pelo CORS da mesma maneira, independente da aplicação ou localidade do servidor. Neste sistema, vamos assumir que um alarme está associado sempre a um único servidor, embora existam poucas exceções. Um servidor pode ter vários alarmes associados como: disco, memória, CPU, aplicações e etc.

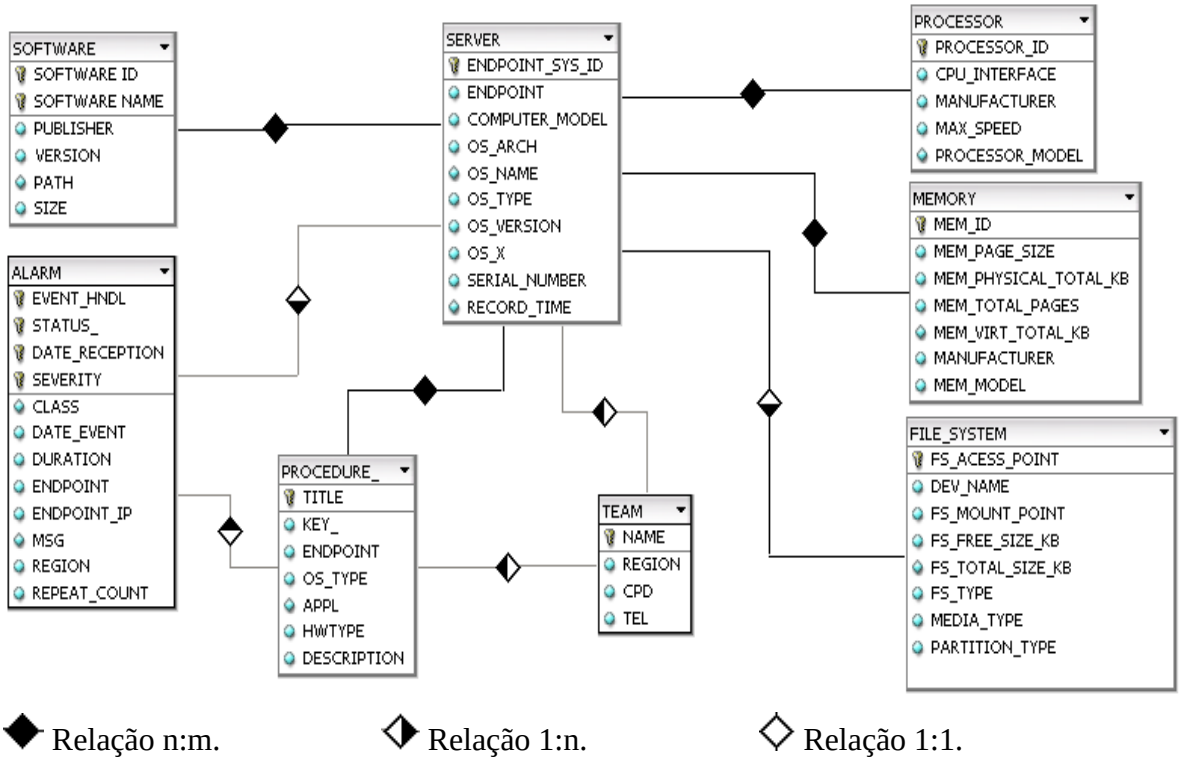

Figura 10 – MER do Sistema Proposto.

Os dados do servidor (*server*), processador (*processor*), *software*, memória (*memory*) e *file system* são informações coletadas pelo ITCM e armazenadas na base de configuração. Os dados do alarme (*alarm*) são informações passadas pela TEC e armazenadas na base de alarmes. Os procedimentos (*procedure*) são dados gerenciados pelo *KMS* e armazenados na base de conhecimento. Os dados das equipes (*team*) são gerenciados por outro sistema que não vamos detalhar, entretanto, esses dados são exportados para a base de configuração do *ITCM* diariamente.

É importante destacar que o MER é um modelo abstrato das entidades, atributos e relacionamentos do sistema. Os nomes das entidades e atributos podem ser diferentes dos nomes das tabelas e colunas do banco de dados. O modelo que pode ser usado como um mapa do banco de dados é o modelo relacional (MR) que surge a partir de uma derivação do MER podendo criar novas entidades correspondentes a tabelas do banco de dados, conforme técnicas de normalização.

O dicionário de dados (DD) consiste numa lista organizada de todos os elementos de dados que são pertinentes para o sistema. Sem o dicionário de dados

o modelo não pode ser considerado completo, pois este descreve entradas, saídas, composição de depósitos de dados e alguns cálculos intermédios. O DD consiste num ponto de referência de todos os elementos envolvidos na medida em que permite associar um significado a cada termo utilizado.

O dicionário de dados associado ao MER apresentado na figura 10 é descrito nas tabelas a seguir:

SERVER		
ATRIBUTO	**DESCRIÇÃO**	**TIPO**
ENDPOINT_SYS_ID	Código Identificador do Servidor	Inteiro
ENDPOINT	Nome do Servidor (hostname)	Texto
COMPUTER_MODEL	Modelo do Servidor	Texto
OS_ARCH	Arquitetura do Sistema Operacional do Servidor	Texto
OS_NAME	Nome do Sistema Operacional	Texto
OS_TYPE	Tipo do Sistema Operacional	Texto
OS_VERSION	Versão do Sistema Operacional	Inteiro
OS_X	Kernel Mode	Inteiro
SERIAL_NUMBER	Numero Serial do Servidor	Texto
RECORD_TIME	Data da Última Atualização do Inventário	Data

Tabela 3 – Dicionário de Dados da Entidade Server.

ALARM		
ATRIBUTO	**DESCRIÇÃO**	**TIPO**
EVENT_HNDL	Número Seqüencial do Evento neste Segundo	Inteiro
STATUS	Status do Evento	Texto
DATE_RECEPTION	EPOCH Timestamp do Horário de Recebimento do Evento	Inteiro
SEVERITY	Severidade do Evento	Texto
CLASS	Classe do Evento	Texto
DATE_EVENT	Data de Recepção do Evento	Data
DURATION	Duração do Evento em Segundos	Inteiro
ENDPOINT	Nome do Servidor (hostname)	Texto
ENDPOINT_IP	IP do Servidor	Texto
MSG	Mensagem do Alarme	Texto
REGION	Regional do Servidor	Texto
REPEAT_COUNT	Número de Eventos Duplicados	Inteiro

Tabela 4 – Dicionário de Dados da Entidade Alarm.

PROCEDURE		
ATRIBUTO	**DESCRIÇÃO**	**TIPO**
PROCEDURE_ID	Código do Procedimento	Inteiro
TITLE	Título do Procedimento	Texto
KEY	Chave/Login do Autor	Texto
ENDPOINT	Nome do Servidor(hostname)	Texto
OS_TYPE	Tipo do Sistema Operacional	Texto
APPL	Nome da Aplicação	Texto
COMPUTER_MODEL	Modelo do Servidor	Texto
DESCRIPTION	Descrição do Procedimento	Texto

Tabela 5 – Dicionário de Dados da Entidade Procedure.

TEAM		
ATRIBUTO	**DESCRIÇÃO**	**TIPO**
NAME	Nome da Equipe	Texto
REGION	Regional do CPD	Texto
CPD	CPD do Servidor	Texto

Tabela 6 – Dicionário de Dados da Entidade Team.

FILE_SYSTEM		
ATRIBUTO	**DESCRIÇÃO**	**TIPO**
FS_ACESS_POINT	Local de Montagem do File System	Texto
DEV_NAME	Nome do File System	Texto
FS_MOUNT_POINT	Ponto de Montagem do File System	Texto
FS_FREE_SIZE_KB	Quantidade de Espaço Livre no FS em Kbytes	Texto
FS_TYPE	Tipo de FS	Texto
MEDIA_TYPE	Tipo de Mídia que o FS aponta (hd, cd, usb...)	Texto
PARTITION_TYPE	Tipo de Partição (lógica, remota...)	Texto

Tabela 7 – Dicionário de Dados da Entidade File System.

SOFTWARE		
ATRIBUTO	**DESCRIÇÃO**	**TIPO**
SOFTWARE ID	Código Identificador do Software	Inteiro
SOFTWARE NAME	Nome do Software	Texto
PUBLISHER	Fabricante do Software	Texto

VERSION	Versão do Software	Texto
PATH	Local de Instalação do Software	Texto
SIZE	Tamanho em Kbytes do Software	Inteiro

Tabela 8 – Dicionário de Dados da Entidade Software.

PROCESSOR		
ATRIBUTO	**DESCRIÇÃO**	**TIPO**
PROCESSOR_ID	Código Identificador do Processador	Inteiro
CPU_INTERFACE	Informação sobre Interface da CPU (tipo de socket)	Texto
MANUFACTURER	Fabricante do Processador	Texto
MAX_SPEED	Freqüência do Processador	Inteiro
PROCESSOR_MODEL	Modelo do Processador	Texto

Tabela 9 – Dicionário de Dados da Entidade Processor.

MEMORY		
ATRIBUTO	**DESCRIÇÃO**	**TIPO**
MEM_ID	Código Identificador da Memória	Inteiro
MEM_PAGE_SIZE	Tamanho da Página em Bytes	Inteiro
MEM_PHYSICAL_TOTAL_KB	Tamanho Total de Memória Instalada em Kbytes	Inteiro
MEM_TOTAL_PAGES	O número de Páginas Disponíveis no Sistema	Inteiro
MEM_VIRT_TOTAL_KB	Quantidade de Memória Virtual Disponível	Inteiro

Tabela 10 – Dicionário de Dados da Entidade Memory.

4.3. Arquitetura Lógica

Para atender os requisitos do novo sistema de gerenciamento de conhecimento, uma nova arquitetura foi implementada. A arquitetura é dividida em três camadas: dados, integração e aplicação. A camada de dados é composta pelas três bases de dados: alarmes, configuração e conhecimento. A camada de integração tem o papel de *middleware* semântico. A camada de aplicação faz

interface direta com o usuário e será responsável por realizar as consultas Semânticas.

A camada de dados é formada pela base de alarmes que contém informações sobre os eventos recebidos pela *TEC*, pela base de configuração que contém o inventário de hardware e software dos computadores/servidores gerenciados pelo *ITCM* e pela base de conhecimento que contém os conhecimentos relacionados ao tratamento dos alarmes. A figura 2, do capítulo 2, ilustra a arquitetura do ambiente atual e mostra o relacionamento dessas bases.

A camada de integração não trabalha como um *middleware* tradicional, que fornece uma integração sintática e estrutural, mas como um *middleware* semântico que fornece uma integração Semântica. Os benefícios dessa metodologia foram apresentados no item 3.2.2 e ilustrados na figura 8.

Mas como transformar bases relacionais distintas e heterogêneas em conteúdo semântico? Um dos principais desafios do novo sistema de gerenciamento de conhecimento é traduzir e integrar os dados das diferentes bases de dados para conteúdo semântico representado por triplas RDF/RDFS. A camada de integração fornece conteúdo semântico para a camada de aplicação através de um ponto central, abstraindo a camada de dados que é composta por diferentes bases de conteúdo relacional.

Acima da camada intermediária uma aplicação é responsável por tratar os conhecimentos semanticamente e faz interface direta com os usuários do sistema, o CORS. A camada de aplicação é responsável por transformar a navegação do usuário pela interface do sistema em consultas Semânticas. As consultas Semânticas são formatadas em SPARQL [W3C, 2008] (linguagem usada para executar consultas em triplas RDF/RDFS), essas consultas são encaminhadas pela camada de aplicação para o SPARQL *endpoint* mantido pela camada de integração. A figura 11, abaixo, ilustra a arquitetura lógica usada pelo novo sistema de gerenciamento de conhecimento baseado em tecnologias da Web Semântica.

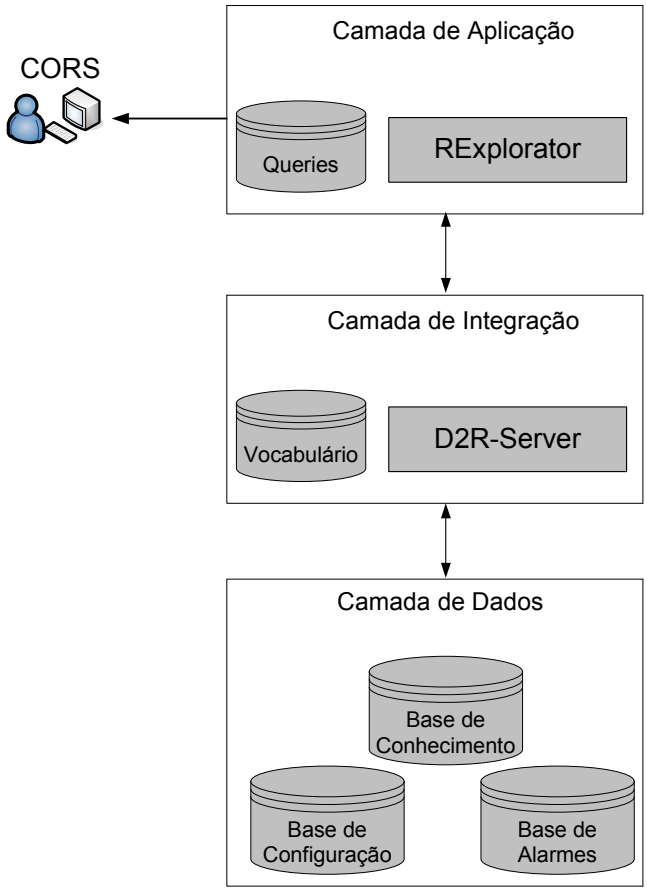

Figura 11 – Arquitetura Lógica do Novo Sistema de Gerenciamento do Conhecimento.

Para a camada de integração atuar como mediadora entre a camada de dados e a camada de aplicação foi necessário definir um vocabulário semântico responsável por suportar o processo de mapeamento e tradução em tempo real dos recursos RDF/RDFS representados por URIs e manipulados pela camada de aplicação para tabelas e atributos de todas as bases da camada de dados. A camada de integração recebe *queries* em SPARQL da camada de aplicação, em seguida, traduz as *queries* SPARQL para *queries* SQL com auxílio do vocabulário. Uma *query* SPARQL da camada de aplicação pode ser traduzida em várias *queries* SQL. Depois de convertidas para SQL, as *queries* podem ser encaminhadas pela camada de integração para diferentes bases na camada de dados.

A camada de dados executa as *queries* SQL em cada base relacional e retorna os resultados para a camada de integração. Então, a camada de integração executa o processo inverso, traduzindo os dados das diferentes bases para triplas RDF que são encaminhadas para tratamento da camada de aplicação.

O D2R-Server [D2R Server] é uma ferramenta para publicar bases de dados relacionais em Web Semântica, ele permite que aplicações executem consultas *SPARQL* sobre bases de dados relacionais. Também é possível consultar os dados das bases relacionas através de browsers RDF e HTML, mas no sistema de gerenciamento de conhecimento proposto o usuário não fará interface com o D2R-Server, apenas a camada de aplicação, responsável por encaminhar as consultas SPARQL.

O D2R-Server foi instalado e configurado para atuar como a camada de integração do novo sistema. Para atuar como integrador das diversas bases, o D2R-Server consulta um vocabulário construído pelo administrador do sistema. A camada de integração e vocabulário serão detalhadas posteriormente. A figura 12 ilustra a arquitetura básica do D2R Server.

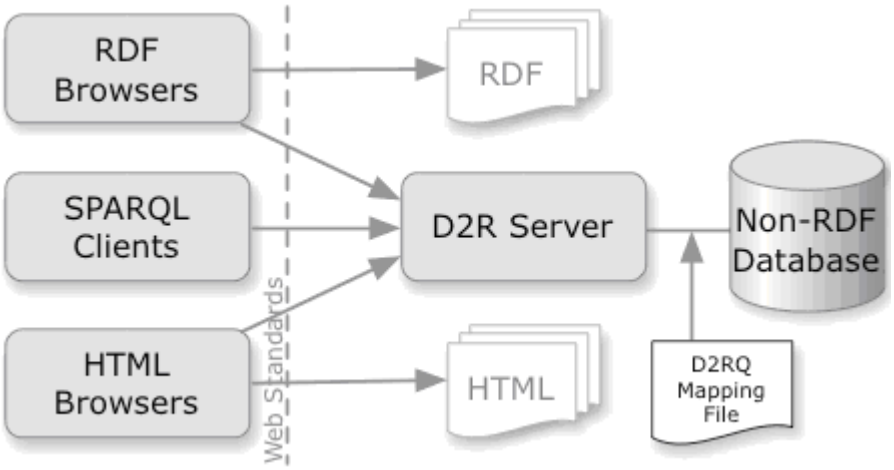

Figura 12 – Arquitetura do D2R Server.

O RExplorator [TecWeb, RExplorator Tool] versão que adiciona algumas funções ao Explorator [TecWeb, Explorator Tool] será responsável por coordenar a camada de aplicação, fazendo interface direta com os operadores do CORS e com a camada intermediária através de consultas SPARQL. Tanto RExplorator como Explorator são ferramentas que permitem aos usuários explorar bases RDF/RDFS semi-estruturadas para aquisição de conhecimento.

As bases de dados da camada de dados são detalhadas na próxima seção e no anexo 1. A camada de integração será detalhada na seção 4.5 e a camada de aplicação será abordada na seção 4.6.

4.4. Camada de Dados

O sistema de gerenciamento de conhecimento em produção atualmente é um sistema Web com base de dados proprietária**.** O principal problema deste ambiente é a não independência dos dados, ou seja, não é possível reutilizar os conhecimentos registrados atualmente se implantarmos um novo sistema de gerenciamento de conhecimento, também não é possível consultar os conhecimentos registrados na base proprietária através de outro sistema. O problema é gravíssimo e um novo sistema de gerenciamento de conhecimento não poderá ler os conhecimentos registrados no sistema atual.

Em conseqüência do grave problema de não independência dos dados do sistema atual será necessário criar uma nova base de conhecimento, essa nova base fará parte da camada de dados do novo sistema. Na fase de implantação será descrita uma solução de contorno para popular essa nova base com os conhecimentos registrados no sistema atual e viabilizar os testes do novo sistema de gerenciamento de conhecimento baseado em Web Semântica.

O modelo relacional da nova base de conhecimento do sistema proposto é apresentado no Anexo 1, assim como os modelos da base de configuração e da base de alarmes. É importante lembrar que o MER, descrito na seção 4.3, é um modelo abstrato e descreve apenas as entidades, atributos e relacionamentos usados pelo sistema proposto.

Os modelos relacionais do Anexo 1 são modelos físicos resultantes de derivações de modelos de entidade e relacionamento e do emprego de técnicas de normalização de banco de dados. Os modelos relacionais das bases de configuração e alarmes possuem dados irrelevantes para o sistema de gerenciamento de conhecimento proposto (não modelados no MER da seção 4.3), mas importantes para suas respectivas aplicações: ITCM e TEC.

4.5. Camada de Integração

Para viabilizar o uso do RExplorator na camada de aplicação é necessário configurar a camada de integração antes. O D2R Server precisa traduzir a

informação contida nas diferentes bases dados para triplas RDF/RDFS. Para possibilitar essa tradução é necessário construir um vocabulário que será usado como base pelo D2R Server.

O vocabulário foi construído com referência no MER apresentado na seção 4.2, e nos modelos relacionais (MR) das bases de configuração, alarmes e conhecimento descritas no Anexo 1 deste documento.

O primeiro passo para construir um vocabulário é definir as classes e propriedades manipuladas pela camada de integração e suas respectivas URIs. Em seguida é necessário descrever o relacionamento dessas classes e propriedades.

O Anexo 2 ilustra todo o vocabulário usado pelo D2R-Server para mapear de forma Semântica os dados contidos nas três bases distintas da camada de dados. Nesta seção será resumido como o Anexo 2 foi construído.

Algumas entidades (tabelas) das diferentes bases de dados manipuladas pela camada de integração (alarmes, configuração e conhecimento) são traduzidas para classes (*rdfs:class*) e seus respectivos atributos são traduzidos para propriedades (*rdfs:type*).

A tabela 11, abaixo, ilustra este de/para, onde as tabelas das diferentes bases são traduzidas para classes RDFS com suas respectivas URIs. Note que duas novas entidades foram criadas (*Has_software* e *Has_processor*) em conseqüência da derivação dos relacionamentos n:m do modelo de entidade relacionamento (MER) apresentado na seção 4.2. Essa derivação é equivalente as tabelas *inst_nativ_sware* e *inst_processor* do ilustradas no Anexo 1.

prefixo vocab: http://serverX:2020/vocab/resource/

Entidade	**rdfs:class**
Event	vocab:event
Computer	vocab:computer
File_system	vocab:filesystem
Software	vocab:software
Has_software	vocab:inst_software
Processor	vocab:processor
Has_processor	vocab:inst_processor

Memory	vocab:memory
Team	vocab:computer
Procedures	vocab:procedures

Tabela 11 – Classes e URIs.

O prefixo http*://serverX* é a URL definida no DNS do servidor que hospeda o D2R-Server. A porta 2020 indica a instância do SPARQL *endpoint* mantido pelo D2R-Server.

É importante destacar que as entidades *computer* e *team* apesar de serem tabelas diferentes foram definidas como mesma classe: *vocab:computer*. Esta decisão de modelagem viabiliza uma maior integração dos dados. Como todos os servidores estão sempre associados a uma equipe, sempre que as propriedades do servidor forem exploradas as informações da equipe também serão resgatadas.

Depois de definir as classes e suas respectivas URIs é necessário fazer o mesmo para suas propriedades (*rdfs:type*), também é importante definir um *label (rdfs:label*). O objeto da propriedade *label* é quem representa um determinado sujeito. As tabelas 12, 13 e 14 ilustram as URIs das propriedades e *labels* definidas para as classes *vocab:event*, *vocab:computer* e *vocab:procedures*, respectivamente. As demais classes não serão detalhadas, mas seguem o mesmo princípio.

prefixo vocab: http://serverX:2020/vocab/resource/

Entidade	**rdfs:class**
EVENT	vocab:event
Propriedades	**rdfs:type**
EVENT_HNDL	vocab:event_hndl
STATUS	vocab:status
DATE_RECEPTION	vocab:date_reception
SEVERITY	vocab:severity
CLASS	vocab:class
DATE_EVENT	vocab:date_event

DURATION	vocab:duration
ENDPOINT	vocab:endpoint
ENDPOINT_IP	vocab:endpoint_ip
MSG	vocab:msg
REGION	vocab:region
REPEAT_COUNT	vocab:repeat_count
Label	**rdfs:label**
-	vocab:endpoint

Tabela 12 – Propriedades e URIs de Event.

prefixo vocab: http://serverX:2020/vocab/resource/

Entidade	**rdfs:class**
COMPUTER	vocab:computer
Propriedades	**rdfs:type**
ENDPOINT_SYS_ID	vocab:endpoint_sys_id
ENDPOINT	vocab:endpoint
ENDPOINT_IP	vocab:endpoint_ip
COMPUTER_MODEL	vocab:computer_model
SERIAL	vocab:serial
OS_TYPE	vocab:os_type
OS_VERSION	vocab:os_version
OS_ARCH	vocab:os_arch
OS_X	vocab:os_x
TEAM_NAME	vocab:team
PHONE	vocab:phone
REGION	vocab:region
Label	**rdfs:label**
-	vocab:endpoint

Tabela 13 – Propriedades e URIs de Computer.

prefixo vocab: http://serverX:2020/vocab/resource/

Entidade	**rdfs:class**
PROCEDURES	vocab:procedures
Propriedades	**rdfs:type**
TITLE	vocab:title
ID_PROCEDURE	vocab:id_procedures
TEAM_NAME	vocab:team
CLASS	vocab:class
ENDPOINT	vocab:endpoint
OS_TYPE	vocab:os_type
APPL	vocab:appl
COMPUTER_MODEL	vocab:computer_model
DESC	vocab:desc
Label	**rdfs:label**
-	vocab:title

Tabela 14 – Propriedades e URIs de Procedures.

Note que alguns atributos têm o mesmo significado, apesar de serem colunas em tabelas de bases de dados diferentes. Como estão em bases diferentes, não há nenhum tipo de relacionamento entre esses dados. Usando o vocabulário é possível integrar esses dados na camada de integração que atuará como um *middleware* semântico. Para dar o mesmo significado a esses dados é necessário apenas definir a mesma URI para identificá-los. A propriedade *vocab:endpoint* é um exemplo, ela está presente nas três bases de dados e expressa o mesmo significado, representa o nome lógico do servidor. Na base de configuração o nome lógico do servidor é representado pelo atributo *tme_object_label*, na base de alarmes e procedimentos pelo atributo *hostname*, vide modelos relacionais do Anexo 1. Outros exemplos são as propriedades: *vocab;computer_model*, *vocab:os_type, vocab:team e vocab:class*.

Para que o D2R Server possa traduzir as informações armazenadas em diferentes bases de dados, organizadas em várias tabelas e atributos para sujeitos,

predicados e objetos em formato RDFS, é necessário construir um vocabulário formatado de acordo com determinados padrões e regras. O trecho a seguir ilustra parte do vocabulário apresentado por completo no Anexo 2.

```
# Table TEC.EVENT_OPEN
map3:TEC.EVENT_OPEN a d2rq:ClassMap;
d2rq:dataStorage map3:database;
d2rq:uriPattern"event/@@TEC.EVENT_OPEN.DATE_RECEPTION|urlify@@/@@TEC.EVENT_OPEN.EVENT_HNDL|urlify@@/@@TEC.EVENT_OPEN.HOSTNAME|urlify@@";
d2rq:class vocab:event;
d2rq:classDefinitionLabel "Events";
.
map3:TEC.EVENT_OPEN__label a d2rq:PropertyBridge;
d2rq:belongsToClassMap map3:TEC.EVENT_OPEN;
d2rq:property rdfs:label;
d2rq:pattern "@@TEC.EVENT_OPEN.HOSTNAME@@";
.
map3:TEC.EVENT_OPEN_DATE_RECEPTION a d2rq:PropertyBridge;
d2rq:belongsToClassMap map3:TEC.EVENT_OPEN;
d2rq:property vocab:DATE_EVENT_EPOCH;
d2rq:propertyDefinitionLabel "TEC DATE_RECEPTION";
d2rq:column "TEC.EVENT_OPEN.DATE_RECEPTION";
d2rq:datatype xsd:decimal;
d2rq:valueRegex "^[0-9]{10}$";
d2rq:valueMaxLength "10";
.
```

O trecho em negrito instrui o D2R Server a definir a tabela *TEC.EVENT_OPEN* da base de dados de alarmes (definida anteriormente como *map3*) como *rdfs:class*. A linha em azul define o nome "*Events*" (*rdfs:label*) para está classe que é identificada pela URI: *vocab:event* (linha acima). A linha em vermelho define o padrão de URI atribuído aos recursos da classe "*Events*" (ou *vocab:event*). As URIs seguem o padrão:

prefixo/event/DATE_RECEPTION/EVENT_HNDL/HOSTNAME

onde *DATE_RECEPTION*, *EVENT_HNDL* e *HOSTNAME* são campos chave da tabela *EVENT_OPEN*. Esses campos foram descritos no dicionário de dados apresentado na seção 4.2, tabela 4.

O trecho em verde, define o padrão de nome (*rdfs:label*) usado pelos recursos da classe "Events" (os alarmes) da classe. O *rdfs:label* de cada recurso será o seu respectivo atributo *HOSTNAME* da tabela *EVENT_OPEN*.

O último parágrafo do trecho transcrito acima, define a propriedade "*TEC DATE_RECEPTION*", identificada pela URI *vocab:date_event_epoch* (linha amarela). Os objetos desta propriedade apontarão para o valor do atributo *DATE_RECEPTION* da tabela *EVENT_OPEN* da base de alarmes.

A figura 13, a seguir, ilustra a visualização HTML do *D2R Server* de um evento traduzido do banco de dados de alarmes para formato RDFS. O evento é identificado pela URI: *http://serverX:2020/resource/event/1255269973/212/xxxx* (observe como respeita o padrão descrito acima), este recurso faz parte da classe *vocab:event*, conforme descrito em *rdf:type*. A propriedade *rdfs:label* representa o nome lógico do servidor cujo objeto é xxxx, portanto, igual a *vocab:endpoint*. A propriedade *vocab:severity*, com objeto 50, indica que o evento é "*Critical*" e a propriedade *vocab:status*, com objeto 30, indica que o evento está aberto. As demais propriedades traduzem os respectivos atributos da base de alarmes, descritos no dicionário de dados da seção 4.2, tabela 4.

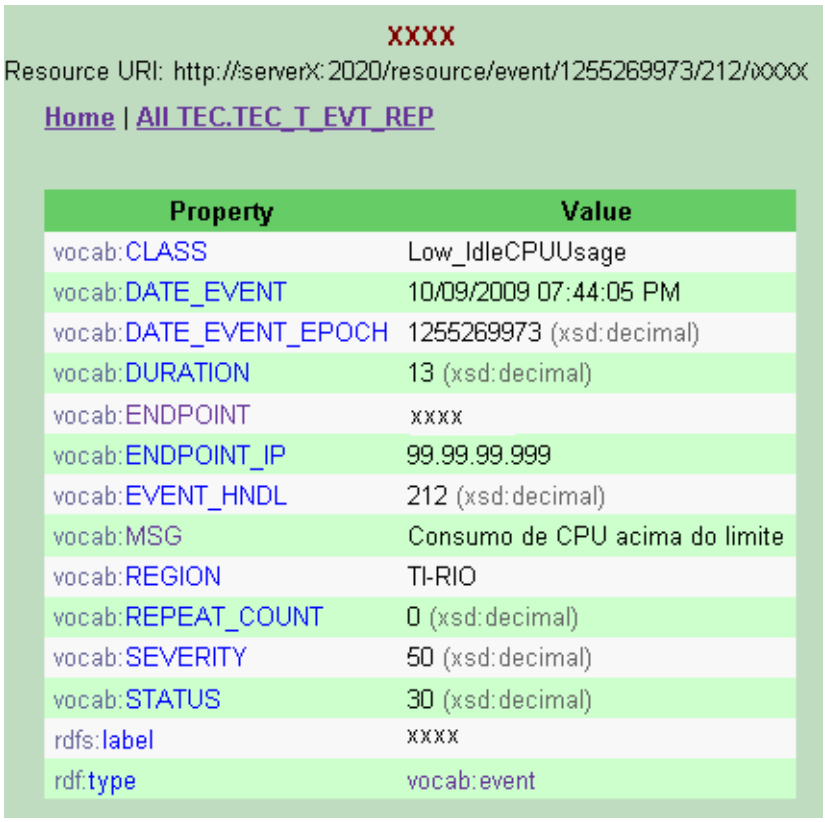

Figura 13 – Exemplo de informação mapeada pelo D2R-Server.

A figura 13 ilustra somente os atributos de um determinado evento da base de alarmes traduzidos para formato RDFS. O processo é semelhante para a base de configuração e conhecimento. Não será descrito aqui todo o processo de tradução de todas as bases, entretanto, no Anexo 2 é possível consultar todo o vocabulário definido para o D2R Server.

A seguir, o trecho em RDFS do evento correspondente a figura 15:

```
@prefix vocab:  <http://serverX:2020/vocab/resource/> .
<http://serverX:2020/resource/event/1255269973/212/xxxx>
    a       vocab:event ;
    rdfs:label "xxxx" ;
    vocab:CLASS "Low_IdleCPUUsage" ;
    vocab:DATE_EVENT "10/09/20010” 07:44:05 PM" ;
    vocab:DATE_EVENT_EPOCH   "1255269973"^^xsd:decimal ;
    vocab:DURATION "13"^^xsd:decimal ;
    vocab:ENDPOINT "xxxx" ;
```

vocab:ENDPOINT_IP "99.99.99.999" ;
vocab:EVENT_HNDL "212"^^xsd:decimal ;
vocab:MSG "Consumo de CPU acima do limite" ;
vocab:REGION "TI-RIO" ;
vocab:REPEAT_COUNT "0"^^xsd:decimal ;
vocab:SEVERITY "50"^^xsd:decimal ;
vocab:STATUS "30"^^xsd:decimal .

Depois de configurar o vocabulário da camada de integração, conforme o Anexo 2, podemos garantir a integração Semântica das diferentes fontes de dados usadas no tratamento de alarmes. O próximo passo é implantar o ambiente de exploração e visualização, RExplorator sobre a camada de integração e usufruir de uma busca mais eficiente e precisa. O RExplorator permite buscar os conhecimentos de forma Semântica (através de consultas SPARQL) sem necessidade de interagir diretamente com as bases de dados.

4.6. Camada de Aplicação

Nesta seção é descrito como o RExplorator foi adaptado para atuar na camada de aplicação fazendo interface direta com os usuários do sistema (CORS) e com a camada de integração, o D2R Server.

O Explorator permite a exploração de bases RDF/RDFS semi-estruturadas através de uma interface Web amigável para os usuários. O RExplorator adiciona novas funcionalidades sobre o Explorator, dentre elas, a possibilidade de criar aplicações de consulta que podem ser compartilhadas de diversas formas e aplicadas a outras bases que sigam as mesmas ontologias, ou seja, o RExplorator cria um ambiente de desenvolvimento de aplicações.

O RExplorator possui duas interfaces principais: a primeira é um ambiente de autoria, onde é possível criar consultas SPARQL através da manipulação direta via interface Web das classes, objetos e propriedades das bases RDF/RDFS conhecidas, ou seja, as consultas podem ser construídas sem a necessidade conhecer a sintaxe SPARQL. A segunda interface é uma interface exclusivamente de consulta, onde é possível executar e visualizar os resultados das consultas

construídas no ambiente de autoria, sem necessidade de entendimento do modelo subjacente.

Os usuários do sistema consultarão os conhecimentos necessários para tratamento dos alarmes através da interface de consulta do RExplorator usando consultas pré-definidas. As consultas pré-definidas serão construídas através da interface de autoria pelos administradores da aplicação ou por usuários avançados do sistema.

Antes de criar as consultas pré-definidas e disponibilizá-las para os usuários, é necessário instalar e configurar o RExplorator. O procedimento de instalação não será detalhado, da mesma forma como não foi descrito o procedimento de instalação do D2R Server, apenas algumas configurações relevantes serão descritas a seguir.

O RExplorator precisa traduzir a manipulação da estruturas RDFS em sua interface para consultas SPARQL, em seguida, as consultas SPARQL devem ser encaminhadas para o *endpoint* do D2R Server na camada de integração. Para encaminhar as consultas SPARQL para o D2R Server é necessário configurar o RExplorator com o endereço IP e porta do SPARQL *endpoin*t da camada de integração, isto pode ser feito editando o *script ruby dataload.rb* com o seguinte trecho:

adapter =ConnectionPool.add_data_source :type => :sparql,:title=>'Conhecimento', :url => "http://serverX:2021/sparql", :results => :sparql_xml, :caching =>true

A figura 14, a seguir, ilustra a interface de autoria do RExplorator. O usuário busca o recurso “xxxxxx” e o RExplorator busca todos os sujeitos, predicados e objetos relacionados ao recurso procurado através de uma consulta SPARQL encaminhada para o SPARQL *endpoint* do D2R Server. O resultado ilustrado na figura 14**,** mostra as classes e propriedades associadas ao recurso “xxxxxx”. Em seguida, é possível criar consultas avançadas através da manipulação direta de recursos e com auxílio de operações pré-definidas de conjuntos como intercessão, união e diferença. Note que os dados associados aos recursos podem estar armazenados em bases de dados diferentes (Alarmes, Configuração e Procedimentos).

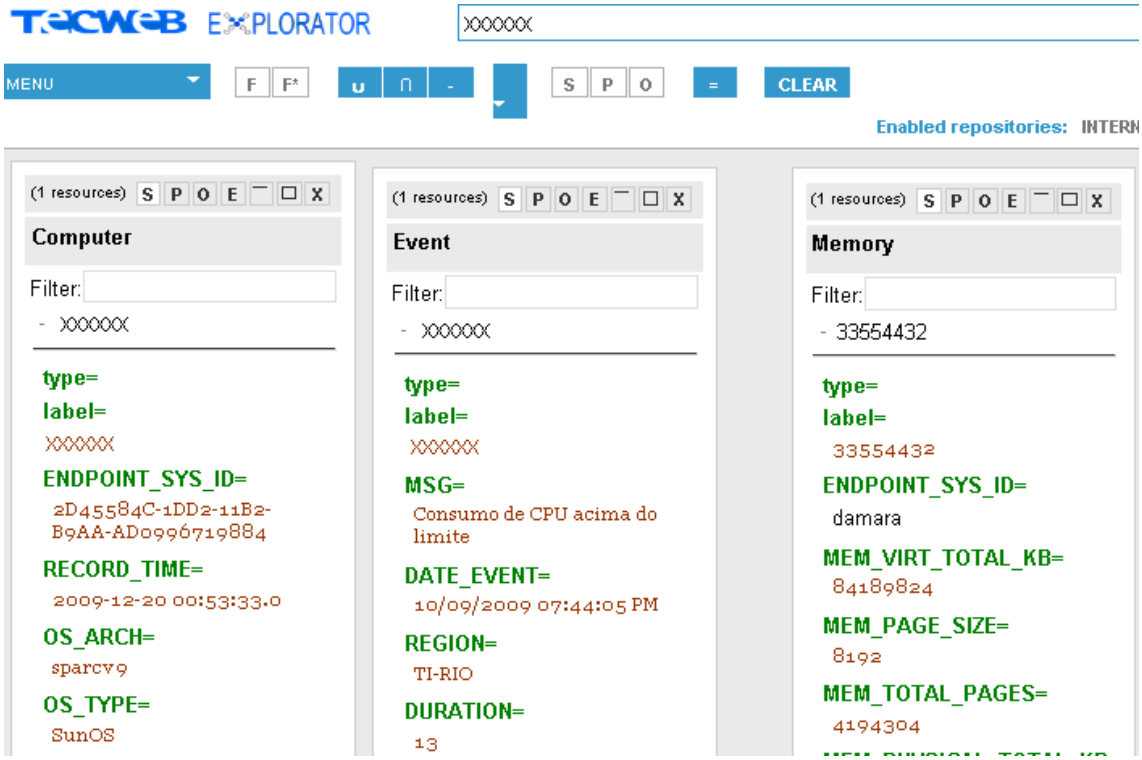

Figura 14 – Exemplo de busca no RExplorator sobre o D2R-Server.

A figura 15, a seguir, ilustra a interface de consulta do RExplorator. Nesta interface o operador pode executar consultas Semânticas pré-definidas compostas de várias consultas SPARQL com apenas um clique. Ainda na figura 15 é ilustrada a execução de uma consulta pré-definida, ela lista os servidores com alarmes abertos (servidores com alarmes associados e com a propriedade vocab/status=30). Ao clicar sobre o nome do servidor os detalhes do alarme, configuração e os procedimentos associados serão exibidos, as figuras 16, 17 e 18 ilustram algumas propriedades dessas classes.

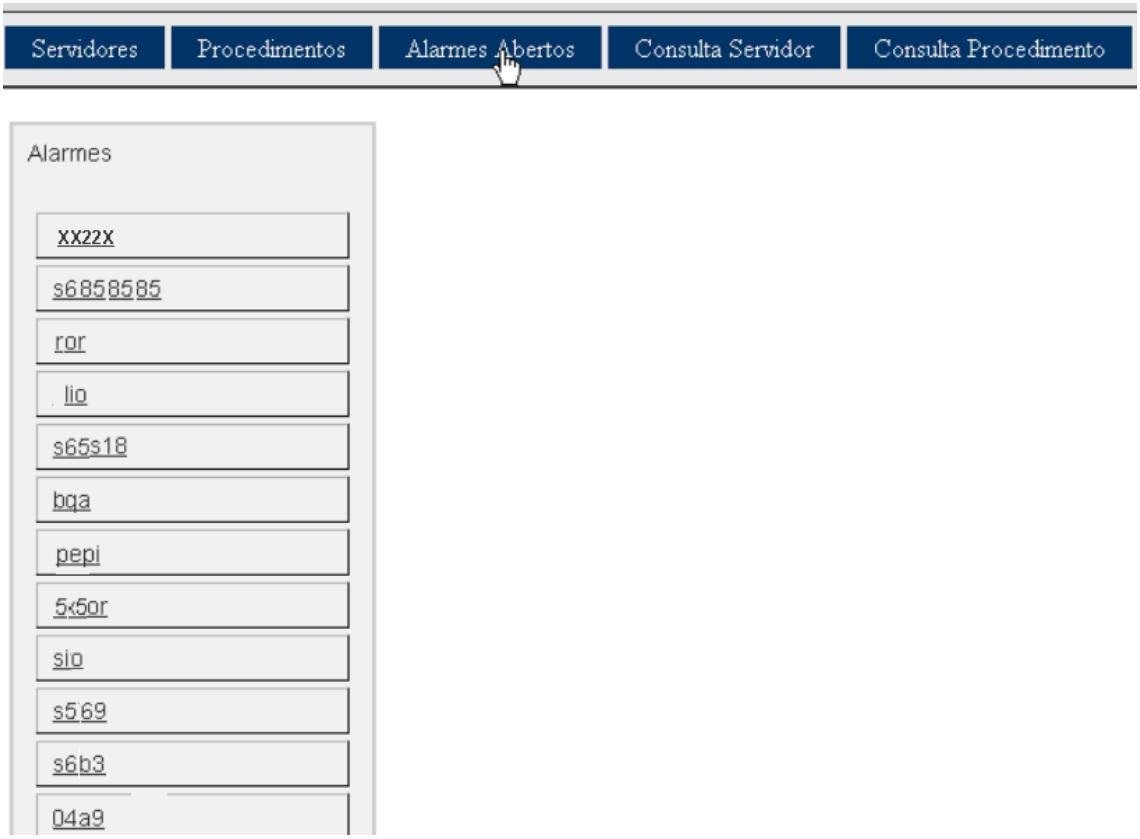

Figura 15 – Interface de consulta do RExplorator.

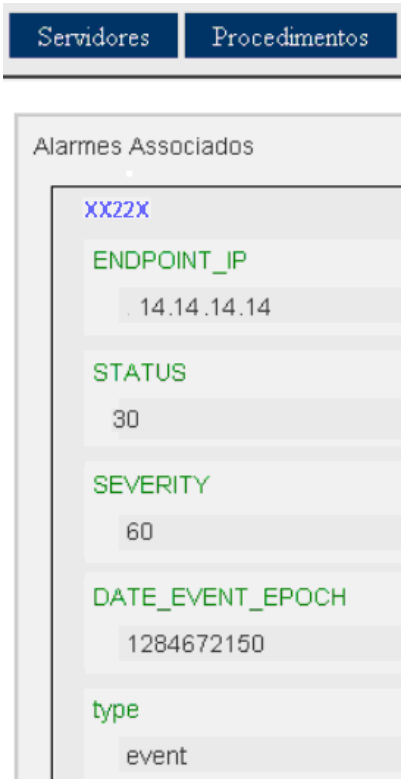

Figura 16 – Algumas propriedades da classe alarme associadas ao servidor XX22X.

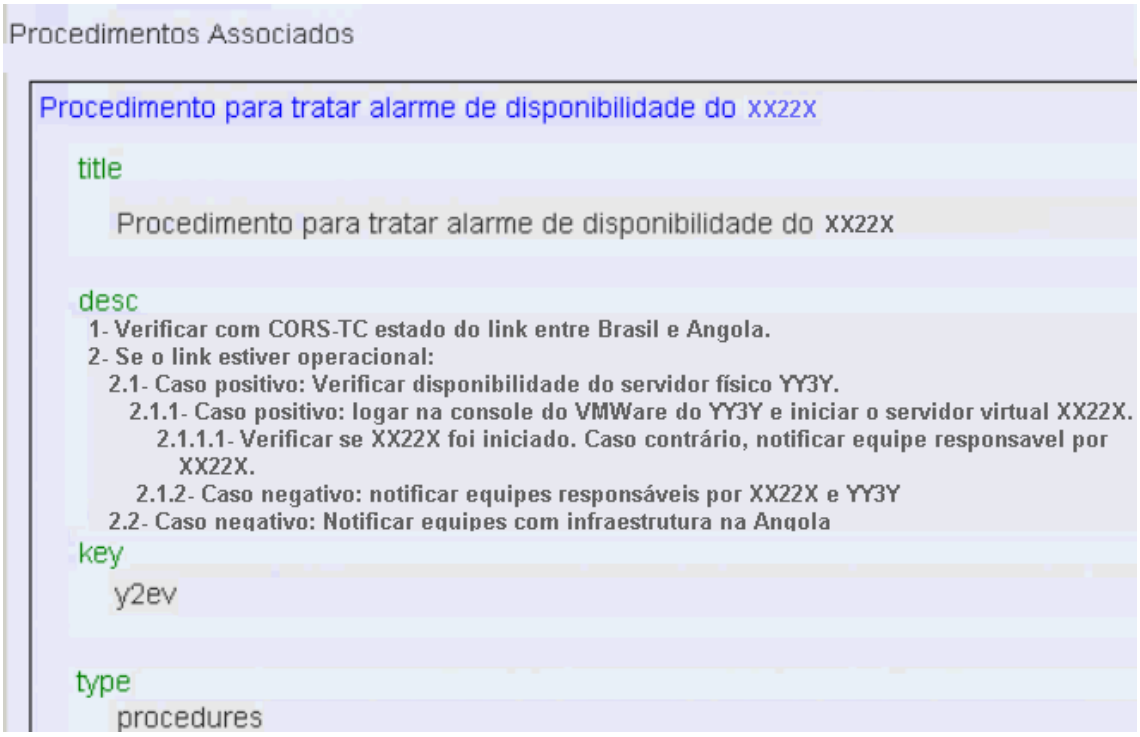

Figura 17 – Algumas propriedades da classe procedures associadas ao servidor XX22X.

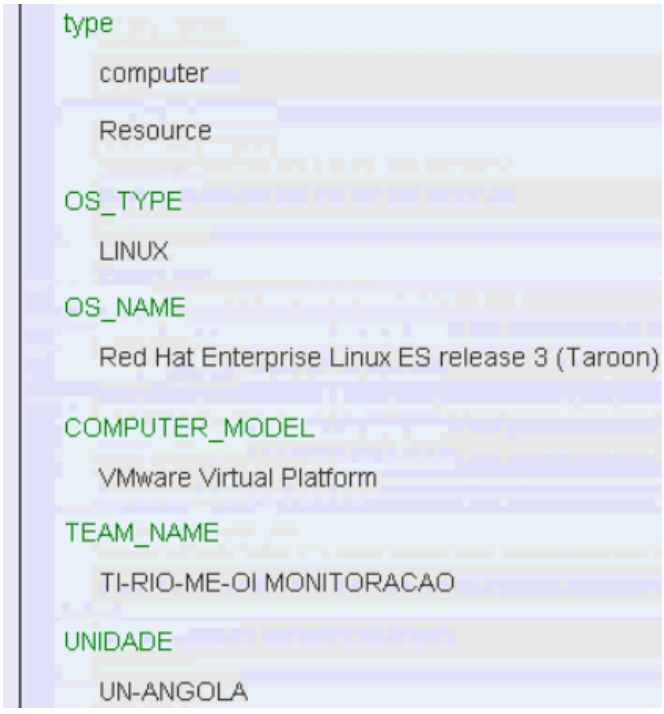

Figura 18 – Algumas propriedades da classe computer associadas ao servidor XX22X.

Para exibir todas as informações associadas ao recurso procurado, o RExplorator formula uma série de consultas SPARQL encaminhadas para o D2R Server, o D2R Server por sua vez, com auxílio do mapa definido no Anexo 2, deriva as consultas SPARQL em várias consultas SQL que são repassadas para as bases de dados. A figura 19 ilustra consultas SPARQL formuladas pelo RExplorator na camada de aplicação e repassadas para o D2R Server na camada de integração, a figura 20 ilustra as respectivas consultas SQL derivadas pelo D2R Server e encaminhadas para as bases de dados na camada de dados.

```
SELECT DISTINCT ?p ?o WHERE { <http://serverX:2020/resource/event/1282452617/1/ogr> ?p ?o .   }
EXPLORATOR(Local)
SELECT DISTINCT ?p ?o WHERE { <http://serverX:2020/resource/event/1282452617/1/ogr> ?p ?o .   }.
INTERNAL
SELECT DISTINCT ?p ?o WHERE { <http://serverX:2020/resource/event/1282452617/1/ogr> ?p ?o .   }.
Conhecimento
SELECT DISTINCT ?p ?o WHERE { <http://serverX:2020/resource/event/1283101193/1/s09> ?p ?o .   }
EXPLORATOR(Local)
SELECT DISTINCT ?p ?o WHERE { <http://serverX:2020/resource/event/1283101193/1/s09> ?p ?o .   }
INTERNAL
SELECT DISTINCT ?p ?o WHERE { <http://serverX:2020/resource/event/1283101193/1/s09> ?p ?o .   }
Conhecimento
SELECT DISTINCT ?p ?o WHERE { <http://serverX:2020/resource/event/1283083438/1/s29> ?p ?o .   }
EXPLORATOR(Local)
SELECT DISTINCT ?p ?o WHERE { <http://serverX:2020/resource/event/1283083438/1/s29> ?p ?o .   }
INTERNAL
SELECT DISTINCT ?p ?o WHERE { <http://serverX:2020/resource/event/1283083438/1/s29> ?p ?o .   }
Conhecimento
SELECT DISTINCT ?p ?o WHERE { <http://serverX:2020/resource/event/1283092276/2/scss69> ?p ?o . }
EXPLORATOR(Local)
SELECT DISTINCT ?p ?o WHERE { <http://serverX:2020/resource/event/1283092276/2/scss69> ?p ?o . }
INTERNAL
```

Figura 19 – Consultas SPARQL do RExplorator.

```
19:42:27 INFO  SPARQL              :: Query: SELECT DISTINCT ?p ?o WHERE { <http://serverX:2020/resource/event/128
2452617/1/ogr > ?p ?o .   }
19:42:27 DEBUG QueryExecutionIterator :: SELECT DISTINCT "TEC"."EVENT_OPEN"."DATE_RECEPTION" FROM "TEC"."EVENT_OPEN" W
HERE ("TEC"."EVENT_OPEN"."DATE_RECEPTION" = 1282438238 AND "TEC"."EVENT_OPEN"."EVENT_HNDL" = 1 AND "TEC"."EVENT_OPEN".
"HOSTNAME" = 'ogr ')
19:42:27 INFO  SPARQL              :: OK/select: SELECT DISTINCT ?p ?o WHERE { <http://serverX:2020/resource/event/128t
2452617/1/ogr > ?p ?o .   }
19:42:27 DEBUG QueryExecutionIterator :: SELECT DISTINCT "TEC"."EVENT_OPEN"."REPEAT_COUNT" FROM "TEC"."EVENT_OPEN" WHE
RE ("TEC"."EVENT_OPEN"."DATE_RECEPTION" = 1282438238 AND "TEC"."EVENT_OPEN"."EVENT_HNDL" = 1 AND "TEC"."EVENT_OPEN"."H
OSTNAME" = 'ogr ')
19:42:27 DEBUG QueryExecutionIterator :: SELECT DISTINCT "TEC"."EVENT_OPEN"."EVENT_HNDL" FROM "TEC"."EVENT_OPEN" WHERE
 ("TEC"."EVENT_OPEN"."DATE_RECEPTION" = 1282438238 AND "TEC"."EVENT_OPEN"."EVENT_HNDL" = 1 AND "TEC"."EVENT_OPEN"."HOS
TNAME" = 'ogr ')
19:42:27 DEBUG QueryExecutionIterator :: SELECT DISTINCT "TEC"."EVENT_OPEN"."ORIGIN" FROM "TEC"."EVENT_OPEN" WHERE ("T
EC"."EVENT_OPEN"."DATE_RECEPTION" = 1282438238 AND "TEC"."EVENT_OPEN"."EVENT_HNDL" = 1 AND "TEC"."EVENT_OPEN"."HOSTNAM
E" = 'ogr ')
```

Figura 20 – Derivação das consultas SPARQL em SQL pelo D2R Server.

Depois de configurar o RExplorator na camada de aplicação é necessário criar algumas consultas Semânticas pré-definidas no ambiente de autoria, que serão usadas pelos operadores do CORS. As consultas Semânticas serão descritas na próxima seção (4.7).

4.7. Consultas Semânticas

O primeiro caso de uso apresentado na seção 4.1 ilustra o cenário de consulta de conhecimento. A consulta pode ser realizada através de diferentes parâmetros, de acordo com a necessidade do operador. As consultas mais comuns serão implementadas através da interface de autoria do RExplorator e disponibilizada para os operadores na interface de consulta. Nesta seção as consultas serão detalhadas de um ponto de vista semântico.

Cada consulta envolve um conjunto de sujeitos, predicados e objetos diferentes, esses elementos se relacionam semanticamente através das ontologias definidas em RDFS para retornar o conhecimento desejado pelo operador.

Na seção 4.1.1 com auxílio da Figura 4, mostramos a diferença entre a metodologia de integração tradicional e a metodologia de integração Semântica. Na metodologia baseada em Web Semântica os agentes de software entendem o significado dos dados através do processamento automático a partir do grafo RDF e de ontologias.

Cada consulta está associada a um grafo RDF e a uma seqüência de ações desempenhadas pelo operador e pelo sistema para chegar a um determinado resultado. Nesta seção os grafos RDF e as seqüências de ações associadas a cada consulta serão ilustrados. As consultas a seguir foram pré-implementadas:

1. Consulta alarmes abertos.
2. Consulta detalhes de alarme.
3. Consulta relação de servidores.
4. Consulta detalhes do servidor.
5. Consulta procedimentos de equipe.
6. Consulta procedimentos de alarme.
7. Consulta detalhes de procedimento.

A consulta número 1 (Consulta alarmes abertos) exibe o nome de todos os servidores (*rdfs:label*) com alarmes abertos. A seqüência de ações desempenhadas para executar a consulta e grafo RDF correspondente ao resultado são ilustrados a seguir:

Seqüência de ações:

1. Operador seleciona botão para exibir alarmes abertos.
2. Sistema exibe relação de alarmes abertos em ordem decrescente de data.

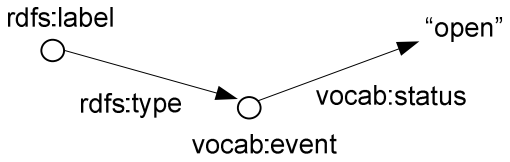

Figura 21 – Grafo RDF para Consulta Alarmes Abertos.

A consulta número 2 (consulta detalhes de alarme) exibe todos os objetos e propriedades associadas a um determinado alarme aberto (identificado por determinada URI), isto inclui as classes: alarmes, configurações e conhecimentos (*vocab:event*, *vocab:computer* e *vocab:procedures*). Nesta consulta, os procedimentos retornados podem estar associados diretamente ao servidor alarmado (*vocab:endpoint*) ou ao seu sistema operacional (*vocab:os_type*), seu modelo (*vocab:computer_model*), sua equipe (*vocab_team*) ou ao tipo de alarme (*vocab_class*). Isto é interessante porque alguns procedimentos podem não estar relacionados diretamente a um determinado servidor, podem ser genéricos, associados a uma característica do servidor ou alarme. Exemplos:

1. Todos os alarmes de CPU oriundos de servidores com sistema operacional AIX 4.3.3 devem ser tratados da mesma maneira pelo CORS independente da aplicação ou equipe responsável.
2. Todos os alarmes do deamon lcfd.sh oriundos de servidores da equipe de 'Correio Eletrônico' devem ser tratados seguindo determinado procedimento.

A consulta deve ser feita a partir do nome do servidor alarmado (rdfs:label da classe event) ou a partir da consulta número 1. Em ambos os casos, o RExplorator consultará os dados relacionados a URI do sujeito com rdf:type= vocab:event, vocab:status = "open" e rdfs:label = 'parâmetro informado' (*query*). A seqüência

de ações desempenhadas para executar a consulta e grafo RDF correspondente ao resultado são ilustrados a seguir:

Seqüência de ações:

1. Operador executa consulta 1 – Alarmes abertos.
2. Operador seleciona detalhes de determinado alarme.
3. Sistema exibe todos os detalhes do alarme, as informações básicas de configuração e todos os procedimentos associados (ao servidor, sistema operacional, modelo ou tipo de alarme).

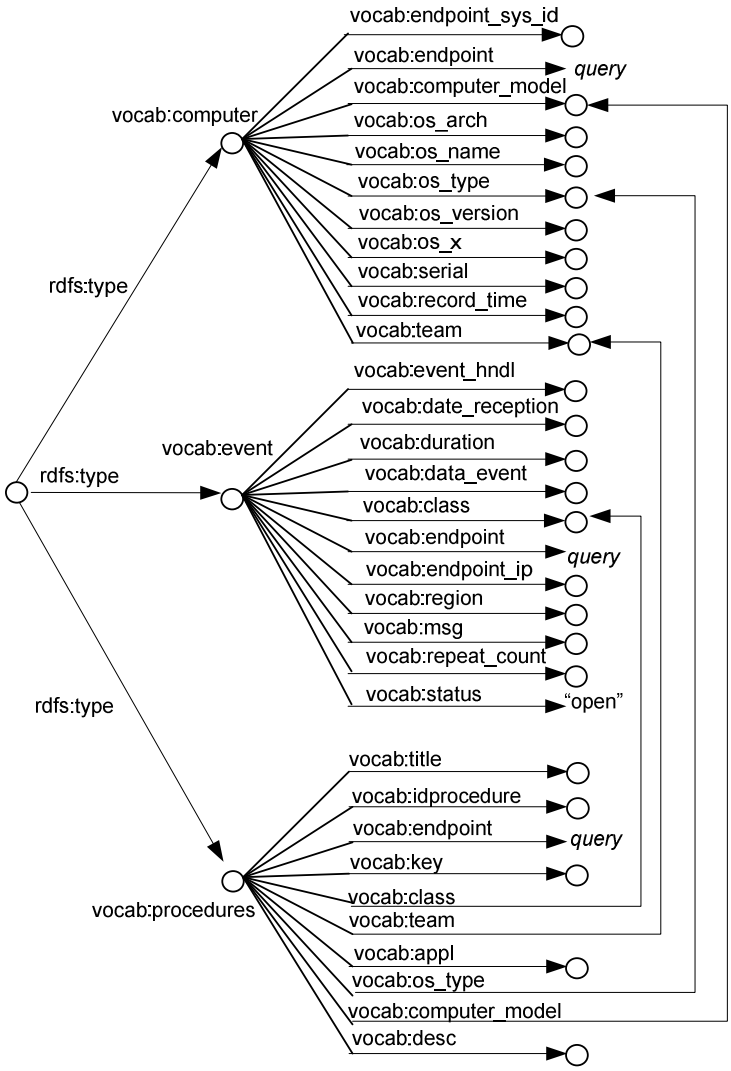

Figura 22 – Grafo RDF para Consulta Detalhes de Alarme.

A consulta número 3 (Consulta relação de servidores) exibe a relação de todos os servidores (rdfs:label da classe computer) presentes na base de

configuração. A seqüência de ações e grafo RDF correspondentes à consulta são ilustrados a seguir:

Seqüência de ações:

1. Operador seleciona botão para exibir relação de servidores.
2. Sistema exibe relação de servidores.

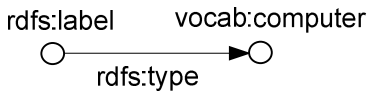

Figura 23 – Grafo RDF para Consulta Relação de Servidores.

A consulta número 4 (Detalhes do servidor) exibe todos os objetos e propriedades associadas a um determinado servidor, isto inclui as classes: alarmes, configurações e conhecimentos (*vocab:event*, *vocab:computer* e *vocab:procedures*). Da mesma forma que a consulta 2, os procedimentos retornados podem estar associados ao servidor alarmado (*vocab:endpoint*), sistema operacional (*vocab:os_type*), modelo (*vocab:computer_model*), equipe (*vocab:team*) ou ao tipo de alarme (*vocab_class*).

A consulta deve ser feita a partir do nome do servidor (*rdfs:label* de *vocab:computer*) ou a partir da consulta número 3. Em ambos os casos, o RExplorator consultará os dados relacionados a URI do sujeito com *rdf:type*= *vocab:computer* e *rdfs:label* = ‘parâmetro informado’ (*query*). A seqüência de ações desempenhadas para executar a consulta e grafo RDF correspondente ao resultado são ilustrados a seguir:

Seqüência de ações:

1. Operador executa consulta 3 – Relação de servidores.
2. Operador seleciona detalhes de determinado servidor.
3. Sistema exibe as configurações básicas do servidor, alarmes e procedimentos associados.

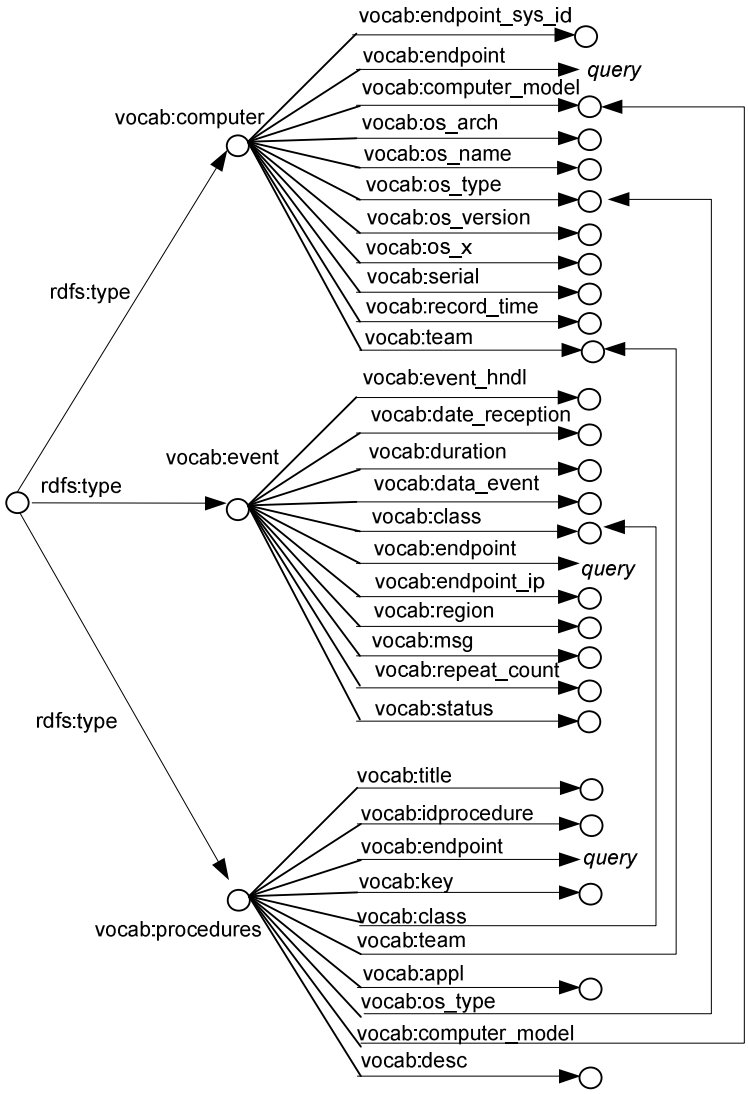

Figura 24 – Grafo RDF para Consulta Detalhes de Servidor.

A consulta número 5 (Consulta procedimentos de equipe) exibe os títulos dos procedimentos de autoria de uma determinada equipe (*rdfs:label* da classe procedures). A consulta deve ser feita a partir do nome da equipe (*vocab:team*). O RExplorator consultará os dados associados a URI do sujeito com *rdf:type= vocab:procedures* e *rdfs:label* = ‘parâmetro informado’ (*query*). A seqüência de ações desempenhadas para executar a consulta e grafo RDF correspondente ao resultado são ilustrados a seguir:

Seqüência de ações:

1. Operador seleciona botão para consultar procedimento.
2. Sistema solicita nome da equipe.

3. Operador informa nome da equipe.
4. Sistema exibe títulos dos procedimentos da equipe.

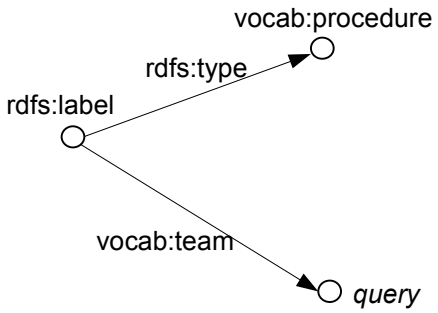

Figura 25 – Grafo RDF para Consulta Procedimento de Equipe.

A consulta número 6 (Consulta procedimentos de alarme) exibe os títulos dos procedimentos relacionados a um determinado tipo de alarme (*rdfs:label* da classe procedures). A consulta deve ser feita a partir do tipo de alarme (*vocab:class*). O RExplorator consultará os dados associados a URI do sujeito com *rdf:type= vocab:procedures* e *vocab:class* = 'parâmetro informado' (*query*). A seqüência de ações desempenhadas para executar a consulta e grafo RDF correspondente ao resultado são ilustrados a seguir:

Seqüência de ações:

1. Operador seleciona botão para consultar procedimento.
2. Sistema solicita tipo de alarme.
3. Operador informa tipo de alarme.
4. Sistema exibe títulos dos procedimentos relacionados ao alarme.

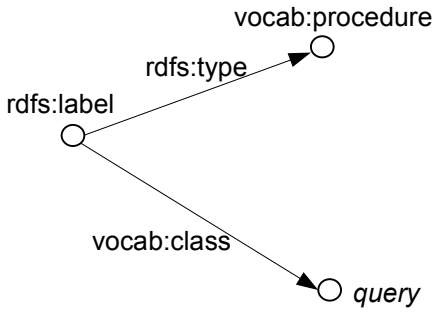

Figura 26 – Grafo RDF para Consulta Procedimento de Alarme.

A consulta número 7 (Detalhes de procedimento) exibe todos os objetos e propriedades de configurações básicas associadas a um determinado procedimento (da classe *vocab:procedures*) e também os servidores e alarmes associados (*vocab:endpoint*). Ou seja, através dessa consulta será possível conhecer os servidores e alarmes que determinado procedimento se aplica. Os servidores e alarmes retornados pela consulta podem estar associados ao nome do servidor (*vocab:endpoint*), sistema operacional (*vocab:os_type*), modelo (*vocab:computer_model*), equipe (*vocab:team*) ou ao tipo de alarme (*vocab_class*).

A consulta deve ser feita a partir da consulta número 5 ou 6. O RExplorator consultará os dados relacionados a URI do sujeito com *rdf:type= vocab:procedures* e *rdfs:label* = 'Título do procedimento selecionado' (*query*). A seqüência de ações desempenhadas para executar a consulta e grafo RDF correspondente ao resultado são ilustrados a seguir:

Seqüência de ações:

1. Operador executa consulta 5 – Consulta procedimentos de equipe.
2. Operador seleciona determinado procedimento (pelo título).
3. Sistema exibe detalhes do procedimento e servidores e alarmes associados.

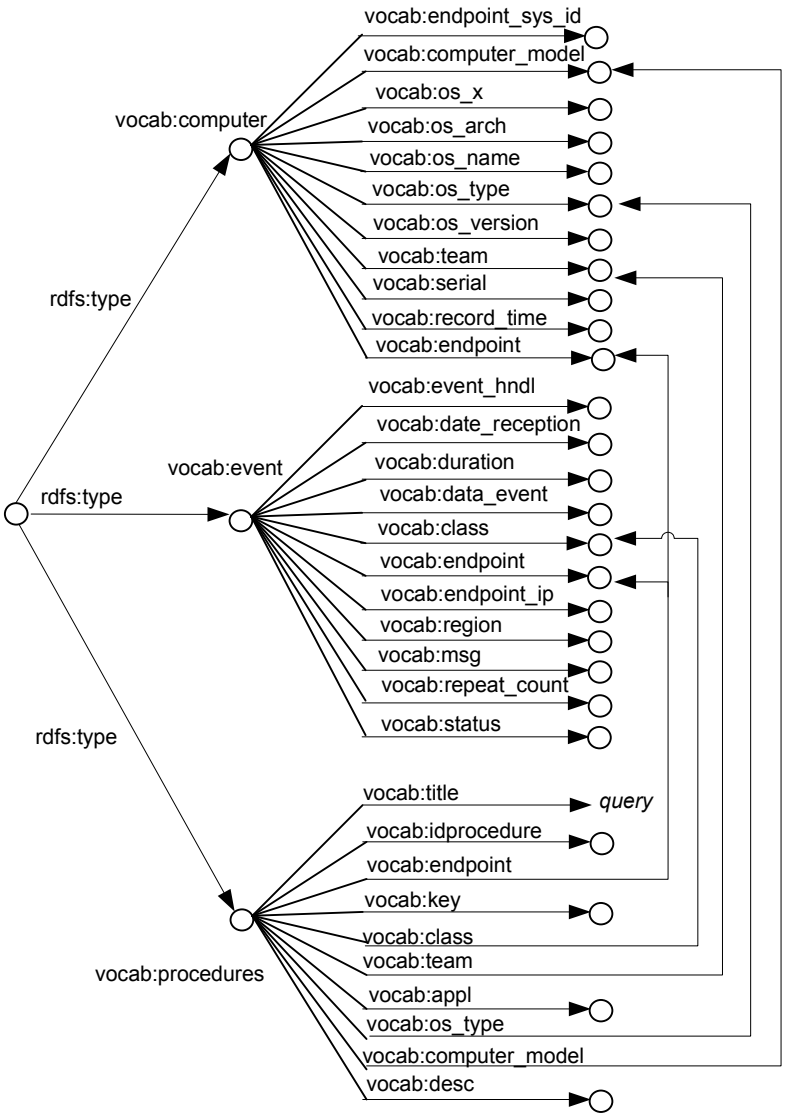

Figura 27 – Grafo RDF para Consulta Detalhes de Procedimento.