Uso de semântica de ações na especificação de um sistema de *data warehousing*

Josiane Michalak Hauagge
Departamento de Informática
Universidade Estadual do Centro Oeste
85015-430 Guarapuava, Paraná
jhauagge@unicentro.br

Martin A. Musicante
Departamento de Informática
Universidade Federal do Paraná
81531-970 Curitiba, Paraná
mam@inf.ufpr.br

(*Recebido: 17 de julho de 2002*)

Resumo: Data warehouses *são grandes repositórios de dados integrados, construídos para armazenar informações para auxiliar no processo de tomada de decisão. Na Universidade Federal do Paraná, UFPR, foi desenvolvido o projeto SAGU (Sistema de Apoio ao Gerenciamento Universitário), que é um sistema de suporte à decisão, que tem por objetivo prover os administradores daquela universidade com informações que auxiliarão na direção de seus objetivos. O principal componente deste sistema é um* data warehouse*. Neste artigo nós apresentamos a especificação formal (utilizando Semântica de Ações) do componente Integrador do* data warehouse *no contexto do projeto SAGU. A especificação obtida foi utilizada para auxiliar no processo de construção do* data warehouse *e pode ser considerada parte da documentação do projeto de um* data warehouse *real.*

Palavras-chave: Data warehouse, *Semântica de Ações, especificação formal*

Abstract: *Data warehouses are large, integrated data repositories, constructed to maintain information to help in decision making, market trends identification and implicit information discovery. The SAGU project, developed at Federal University of Paraná, is working on the development of a decision suport system for the university managers. One of the main*

components of this system is a data warehouse. We present a formal specification (using Action Semantics) of the components of a data warehouse, in the context of the SAGU project. It was used to help in the warehouse design process. The formal description can be regarded as part of the documentation of an actual data warehousing project.

Key words: *Data warehouse, Action Semantics, formal specification*

1 Introdução

A informação vem desempenhando um papel fundamental em todos os setores da sociedade. Cada vez mais o sucesso das organizações torna-se dependente da utilização da informação, em particular no uso relativo à tomada de decisão [1]. Porém, a maioria das aplicações informatizadas existentes atualmente nas organizações não são apropriadas para possibilitar o entendimento do funcionamento dos negócios como um todo pois foram projetadas para suportar as transações que ocorrem no dia-a-dia das organizações, ou seja, realizar o processamento operacional das mesmas [2].

Nesse contexto, surge a necessidade de criação de um ambiente adequado para o armazenamento e gerenciamento eficiente dos dados provenientes das aplicações operacionais, com o objetivo de se produzir informações que auxiliarão seus usuários na direção e alcance de suas metas.

Para a construção de tal ambiente, normalmente é necessária a criação de um novo repositório de dados, único e centralizado, populado com dados derivados das diversas aplicações operacionais da organização e que geralmente precisam ser combinados, consolidados e sumarizados. *Data warehouse* é a materialização desse conceito.

Também nas instituições de educação e pesquisa, a qualidade das atividades, bem como o controle rigoroso e transparente dos recursos, pode ser influenciado pelo uso da informação. O projeto SAGU-Sistema de Apoio ao Gerenciamento Universitário, desenvolvido na Universidade Federal do Paraná, UFPR, tem por objetivo a disponibilização de um ambiente para auxiliar seus dirigentes no processo de tomada de decisão e proporcionar uma melhor gestão das informações administrativas e acadêmicas.

O principal objetivo deste artigo é apresentar a especificação formal obtida para o componente Integrador do *data warehouse* no contexto do projeto SAGU. Nossa especificação é escrita usando Semântica de Ações [3] e pode ser vista como uma documentação formal de um projeto de *data warehousing* real.

O restante deste artigo é organizado da seguinte forma: a seção 2 introduz a terminologia de *data warehousing*; a seção 3 aborda alguns conceitos de Semântica de Ações; a seção 4 apresenta o projeto SAGU e suas particularidades; a seção 5 mostra a especificação formal do componente Integrador do *data warehouse* do SAGU e as conclusões são apresentadas na seção 6.

2 Data warehousing

Um *data warehouse* é um grande repositório de dados integrados, coletados de fontes distribuídas, autônomas e heterogêneas, tais como: sistemas legados, aplicações operacionais e fontes externas [4, 5].

O propósito específico de um *data warehouse* é criar um ambiente apropriado, tipicamente mantido separado das bases de dados operacionais da organização [2], onde sistemas de suporte à decisão possam produzir informações para auxiliar seus usuários nos processos de tomada de decisão, descoberta de informações implícitas nos dados e identificação de tendências de mercado.

Os componentes de um sistema de *data warehousing* podem ser divididos em dois grupos de software: *componentes de integração* e *componentes de análise e consulta* [6].

2.1 Componentes de integração

São os componentes responsáveis por realizar o carregamento inicial dos dados e verificar as ocorrências de atualizações nas fontes de dados, ou seja, notificá-las ao *data warehouse*, obtê-las das fontes, carregá-las e executar os procedimentos necessários para integrá-las e armazená-las no *warehouse*.

Os componentes de integração podem ser divididos em módulos de *software* responsáveis por desempenhar tarefas específicas. Existe um componente responsável por monitorar as fontes de dados, notificar a ocorrência de atualizações de interesse ao *data warehouse* e carregar as atualizações encontradas. Como as fontes são heterogêneas, faz-se necessário a existência de um componente para realizar a tradução das informações do formato externo para o formato utilizado no *data warehouse*.

Como os dados são originários de diferentes fontes, antes deles serem armazenados no *warehouse*, eles são validados por um conjunto de programas de integração e transformação de dados [2], responsáveis por realizar as operações de consolidação, formatação e depuração dos dados obtidos, com o intuito de prover uma visão unificada de todos os dados da organização.

Num ambiente de *data warehousing*, os dados são consultados e analisados, mas não são alterados, sendo que os próprios usuários finais podem propor as consultas ao *warehouse*, utilizando ferramentas que possuem facilidades para interação.

2.2 Componentes de análise e consulta

São os programas de *front-end*, cuja finalidade específica dentro das aplicações informacionais é a utilização dos dados disponibilizados no *warehouse* e a transformação dos mesmos em informações úteis à tomada de decisão, permitindo um gerenciamento de causas e efeitos mais eficiente e, conseqüentemente, a antecipação de políticas para tal.

As ferramentas para análise e consulta, de acordo com Kimball [7], podem ser um pacote de *software* independente, que possibilita a elaboraçao de consultas *ad*

hoc e a utilização de instruções SQL[1] para exibir e formatar relatórios, tanto no modo texto, como no modo gráfico ou uma aplicação monolítica construída sobre um domínio específico, com uma interface de usuário personalizada.

Outra variedade de programas utilizados nesse contexto são as ferramentas de mineração de dados (*data mining* [8-11]), que são *softwares* que exploram os dados para identificar relacionamentos em variáveis previmente independentes, de acordo com critérios pré-determinados [12].

3 Semântica de Ações

Semântica de Ações é um formalismo desenvolvido para prover descrições compreensíveis de aplicações e linguagens da vida real [3, 13].

A Semântica de Ações foi desenvolvida a partir da *Semântica Denotacional* [14], combinando formalidade com aspectos pragmáticos desejáveis, tais como:

- facilidade de leitura, fazendo uso de termos utilizados na linguagem informal;
- modularidade;
- capacidade de tratar abstrações;
- facilidade para provar propriedades algébricas para produzir semânticas equivalentes.

A Semântica de Ações utiliza entidades *ad hoc* que são chamadas de ações e que são operacionais, ou seja, quando executadas, processam as informações gradualmente [3].

O formalismo de Semântica de Ações utiliza uma metalinguagem formal para descrever ações, denominada *Notação de Ações*, que apresenta ações primitivas e combinadores para formar ações complexas, mas que pode ser ampliada pela inclusão de novos tipos de dados e pela criação de abreviaturas para entidades já especificadas, sem a necessidade de reformular as descrições já realizadas.

Os símbolos usados na Notação de Ações são palavras tais que frases em inglês − desde que completamente formais − podem ser usadas para expressar a maioria dos conceitos presentes em computação.

3.1 Ações, dados e *yielders*

Ações são entidades semânticas utilizadas para representar o controle e o processamento de informações, apresentando vários resultados possíveis:

- terminação normal (*complete* − execução completa);
- terminação excepcional (*escape* − saída anormal);

[1]Structured Query Language - Linguagem de Consulta Estruturada. É uma linguagem comercial para definição e manipulação de dados para banco de dados relacionais.

- terminação sem sucesso (*fail* - falha);
- não-terminação (*diverge* - divergência).

A Notação de Dados é usada para descrever a informação processada pelas ações, provendo uma coleção de dados de tipos abstratos definidos algebricamente, incluindo números, caracteres, strings, conjuntos, tuplas, mapeamentos entre outros que podem ser especificados *ad hoc*.

Existe uma terceira classe de entidades chamada *yielders* (produtores) que podem ser avaliadas para gerar dados, cujo valor é dependente da informação disponível para a ação primitiva em que ele ocorre, visando a facilitar a transformação dos diversos tipos de informações processadas.

4 O projeto SAGU

Na Universidade Federal do Paraná - UFPR, está sendo desenvolvido o Sistema de Apoio ao Gerenciamento Universitário - SAGU [15], que tem como objetivo a criação de um ambiente para produzir informações relevantes visando a apoiar e agilizar o processo decisório dos diretores daquela universidade e, conseqüentemente, proporcionar a gestão eficaz de informações acadêmicas e administrativas.

Esse projeto visa a integrar informações heterogêneas, originárias de diferentes setores da UFPR, por meio da construção de um *data warehouse*, a partir do qual técnicas de análise possam ser utilizadas para extrair informações de qualidade sobre a realidade da universidade, proporcionando a percepção rápida das mudanças e tendências, permitindo ao setor público uma melhor alocação dos recursos disponíveis.

Todas as funções necessárias para o funcionamento do *data warehouse* do SAGU estão sendo implementadas em componentes de *software* distintos, responsáveis por desempenhar tarefas específicas e interagir entre si, quando necessário.

Cada um desses componentes possui um identificar único, conhecido por todos os outros componentes e utilizado para a comunicação entre eles, por meio dos protocolos de comunicação TCP/IP.

4.1 Arquitetura

Na arquitetura do *data warehouse* do projeto SAGU (Figura 1), o Integrador é o componente responsável pela inicialização do sistema, contratação de outros agentes (processos) e especificação das ações que eles devem executar, coordenação do processo de carregamento e manutenção dos dados existentes.

O agente Monitor é responsável por carregar os dados das fontes que são externas, autônomas e heterogêneas. Existe um Monitor específico para cada fonte de dados, como uma forma de se esconder detalhes particulares de cada fonte e possibilitar a utilização de um método de *interface* uniforme no *warehouse*. O Monitor

periodicamente verifica a fonte, procurando por atualizações nos dados que são de interesse para o *data warehouse*.

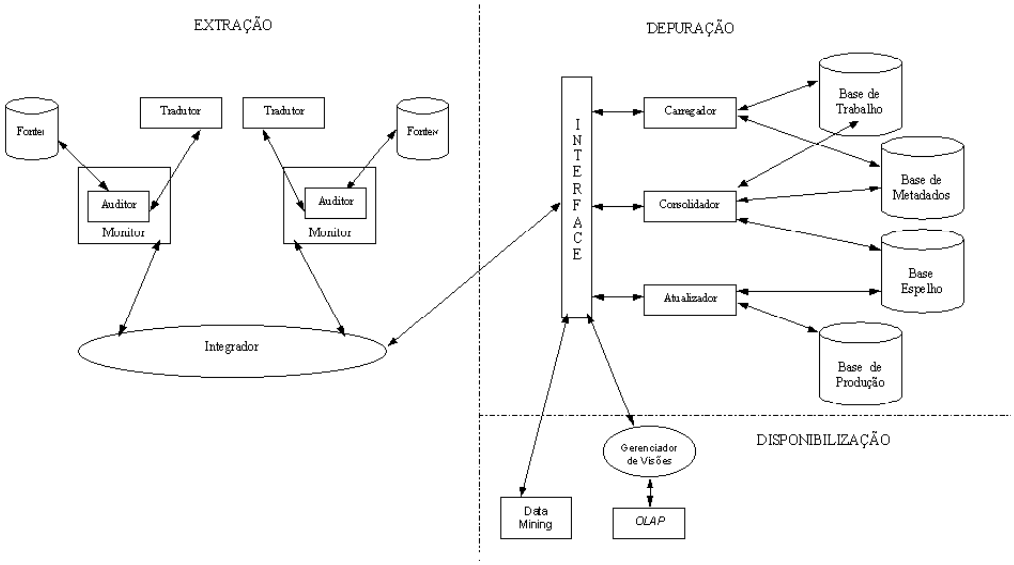

Figura 1. Arquitetura do *data warehouse* do projeto SAGU.

As atualizações são enviadas para o agente Tradutor (também específico para cada fonte), que as traduz para o formato utilizado no *warehouse*. Após a tradução das atualizações, o Monitor as envia para o Integrador, que notifica o agente Carregador para transportá-las para a Base de Dados de Trabalho, onde são armazenadas as atualizações dos dados das fontes.

Quando o Carregador finaliza o processo de transporte de dados, o Integrador notifica o agente Consolidador, que é responsável primeiramente por resolver as possíveis inconsistências dos dados armazenados na Base de Dados de Trabalho, corrigir as diferenças existentes entre os tipos de dados e definir os relacionamentos entre as múltiplas fontes de dados para combinar as atualizações recebidas com os dados existentes no *warehouse*.

Então, o Consolidador atualiza a Base de Dados Espelho[2] do *data warehouse* para que, durante esse processo, a Base de Dados de Produção não tenha de ficar *off-line*.

Estando a Base de Dados Espelho em um estado consistente, para finalizar o processo de atualização, o Integrador notifica o agente Atualizador para transportar a Base de Dados Espelho para a Base de Dados de Produção do *data warehouse*.

Durante o processo de construção do ambiente do *data warehouse*, realizou-se a especificação formal do sistema como forma de orientação no desenvolvimento de todo esse processo, melhorando a comunicação entre as partes envolvidas no projeto, facilitando o desenvolvimento modular e proporcionando um *feedback* para validar a descrição obtida.

[2]É idêntica à Base de Dados de Produção e armazena todos os dados utilizados no *warehouse*.

5 Descrição formal do Integrador

Os componentes do *data warehouse* do projeto SAGU estão implementados como um sistema de agentes distribuídos, sendo que cada agente é responsável por desempenhar uma tarefa específica, podendo comunicar-se com os outros agentes, enviando e recebendo mensagens, mesmo que residindo fisicamente separados.

Inicialmente, somente um único agente distinto, conhecido como *user agent*, está ativo que é o Integrador no projeto SAGU, responsável pela inicialização e coordenação de todo o processo.

Na inicialização do sistema (realizada pela ação *Init-System*), o Integrador recebe uma lista, chamada *MSList*, que contém informações sobre os agentes Monitores e as fontes de dados. Também recebe o inteiro *Consolidation-Time*, referente à periodicidade de consolidação das atualizações das fontes externas no ambiente do *data warehouse*.

É através das informações da *MSList*, que o Integrador inicializa cada um dos agentes Monitores e especifica as ações que os mesmos devem executar de acordo com as características específicas do tipo de fonte de dados à qual ele está relacionado. Cada elemento da *MSList* é uma tupla contendo os seguintes dados:

- *Monitor Identification* - identificador único para comunicação com o Monitor.
- *Source Identification* - identificador único para comunicação com a fonte de dados.
- *Source Type* - especifica o tipo da fonte de dados - que pode ser "*Cooperative-Source*" ou "*Non-Cooperative-Source*".
- ABSN - *Audit Block Serial Number* - inteiro que corresponde ao último registro anteriormente verificado na busca de atualizações no arquivo de *log* das fontes não-cooperativas.
- *Wait Time* - inteiro que determina a periodicidade para a checagem das atualizações nas fontes não-cooperativas.

A seguir são apresentadas as seis ações executadas pelo Integrador. A ação *Init-System* está dividida em duas outras ações: a ação *Integrator-Init-Action* inicializa o sistema de *data warehousing* e a ação *Integrator-Action* coordena todo o processo de atualização do ambiente.

- MSList=listof(string,agent,string,integer,integer).
- ConsolidationTime=integer.
- Init-System(_,_)::MSList,Consolidation-Time→action[diverging].

(1) Init-System(L: MSList,T: Consolidation-Time)=
  | Integrator-Init-Action(L)
  hence
  | Integrator-Action(L, T)

A ação *Integrator-Init-Action* é descrita a seguir:

- Integrator-Init-Action(_) :: MSList → action [binding | communicating | completing][using current bindings | current buffer].

```
(2) Integrator-Init-Action(L: MSList) =
        | give L
        then
        | | | unfolding
        | | | | | chek not(the given list is empty-list)
        | | | | and then
        | | | | | | | subordinate an agent
        | | | | | | and give component #1 of the head of the given list
        | | | | | then bind the given agent #1 to the given string #2
        | | | | and given tail of the given list
        | | | | then unfold
        | | | or check(the given list is empty-list)
        | | and subordinate an agent then bind it to "Semaphore WDB"
        | | and subordinate an agent then bind it to "Loader"
        | | and subordinate an agent then bind it to "Consolidator"
        | | and subordinate an agent then bind it to "Updater"
        | | before
        | | | | unfolding
        | | | | | | | check not(the given list is empty-list)
        | | | | | | and then
        | | | | | | | | give head of the given list
        | | | | | | and then
        | | | | | | | | | check (the given string#3 is "Cooperative-Source")
        | | | | | | | | and then send a message [to the agent bound to the given string #1]
        | | | | | | | |       [containning the application of closure abstraction of
        | | | | | | | |       CS-Monitor-Function to the given agent #2]
        | | | | | | | or
        | | | | | | | | | check (the given string #3 is "Non-Cooperative-Source")
        | | | | | | | | and then send a message [to the agent bound to the given string #1]
        | | | | | | | |       [containning the application of closure abstraction of
        | | | | | | | |       NCS-Monitor-Function to the given agent #2]
        | | | | | | and give tail of the given list
        | | | | | | then unfold
        | | | | | or check (the given list is empty-list)
        | | | and send a message [to the agent boud to "SemaphoreWDB"]
        | | | |             [containning the closure abstraction of SemaphoreWDB-Function]
        | | | and send a message [to the agent bound to "Loader"]
        | | | |             [containning the closure abstraction of Loader-Function]
        | | | and send a message [to the agent bound to "Consolidator"]
        | | | |             [containning the closure abstraction of Consolidator-Function]
        | | | and send a message [to the agent bound to "Updater"]
        | | | |             [containning the closure abstraction of Updater-Function]
```

Essa ação recebe uma lista de informações, do tipo *MSList*, denominada L. Na primeira parte dessa ação (antes do combinador *before*) são criados os novos processos representando os agentes Monitores da lista L e todos os outros componentes do sistema: Carregador, Consolidador, Atualizador e Semáforo. A lista L e as associações da primeira parte da ação são passadas à segunda parte pelo combinador *before*.

Na segunda parte da ação, o Integrador envia as ações que cada agente deverá executar. Essas ações podem ser vistas na especificação completa do *data warehouse* do projeto SAGU apresentada em [16].

Todas as associações realizadas nessa ação são passadas à ação *Integrator-Action* através do combinador *hence* da ação *Init-System*.

A seguir, a ação *Integrator-Action* é apresentada:

- Integrator-Action (_ , _) :: MSList, Consolidation-Time → action[binding | storing | diverging | communicating][using current bindings |current buffer | current storage].
- Consolidator.Log = list of tuples.
- Updater.Log = list of tuples.

```
(3) Integrator-Action(L: MSList, T: Consolidation-Time) =
| | give L
| then
| | unfolding
| | | | check not(the given list is empty-list)
| | and then
| | | | give head of the given list
| | | then Integrate-Source
| | and
| | | | give tail of the given list
| | | then unfold
| | or check(the given list is empty-list
and
| unfolding
| | | Sleep T
| | then send a message [to the agent bound to "Consolidator"]
| |       [containning an "OK"]
| | then receivea message [from the agent bound to "Consolidator"]
| |       [containning a Consolidator.Log]
| | then send a message [to the agent bound to "Updater"]
| |       [containnning an "OK"]
| | then receive a mesage [from the agent bound to "Updater"]
| |       [containning a Updater.Log]
| | then unfold
```

Essa ação recebe uma lista L, do tipo *MSList* e o inteiro T, representando o intervalo de tempo para a consolidação das atualizações dos dados e, através de dois *loops,* realiza a manutenção dos dados do *warehouse*. Esses *loops* podem ser executados simultaneamente (de acordo com as características do combinador *and*).

No primeiro *loop*, a ação *Integrate-Source* (apresentada a seguir) é responsável por notificar os Monitores da lista L para realizarem o carregamento das atualizações ocorridas nas fontes de dados externas ao *data warehouse*.

No segundo *loop* (que é infinito), a ação *Sleep* deixa o processo em estado de espera, de acordo com o valor de T, determinando assim a periodicidade de execução da ação. Na sequência, o Integrador envia uma mensagem ao Consolidador para realizar a consolidação das atualizações obtidas e integrá-las à Base de Dados Espelho do *data warehouse*. Em seguida, o Integrador aguarda o recebimento da mensagem de retorno do Consolidador, contendo a lista de *log* das operações realizadas (*Consolidator.Log*), indicando a finalização do processo de consolidação.

Então, o Integrador envia uma mensagem ao Atualizador para que este realize o transporte dos dados da Base de Dados Espelho, agora atualizada, para a Base de Dados de Produção e aguarda o recebimento de uma mensagem de retorno, contendo a lista de *log* das operações realizadas (*Updater.Log*), indicando a finalização do processo de transporte. A ação retorna ao início do *loop* recomeçando o processo.

São realizados procedimentos não automatizados para a verificação dos arquivos de *log* das operações realizadas pelo Consolidador e pelo Carregador e, para os possíveis erros encontrados, são realizados procedimentos que não são especificados aqui, pois são feitos de forma manual e esporádica.

A ação *Sleep* deixa o sistema em estado de espera, de acordo com o valor especificado em T, determinando, assim, a periocidade da atualização do *warehouse*. Essa função é dependente de implementação e não é especificada aqui.

- Sleep _ :: integer → action

(1) Sleep I = □

A notação □ indica que este item não é definido nesta parte da especificação.

A ação *Integrate-Source*, utilizada pela ação *Integrator-Action* apresentada acima, é especificada a seguir:

- Integrator-Source :: action [receiving a tuple | completing]
  [using current bindings | current buffer].

(4) Integrate-source

```
| check(the given string #3 is "Cooperative-Source") then
| | give the given string #1
| and then Integrate-Cooperative-Source
or
| check(the given string #3 is "Non-Cooperative-Source") then
| | give (the given string #1, the given integer #4, the given integer #5)
| and then Integrate Non-Cooperative-Source
```

Nessa ação, o Integrador recebe uma tupla de dados da lista *MSList*. Então, de acordo com o tipo da fonte, o Integrador executa uma das seguintes ações:

- *Integrate-Cooperative-Source*: para fontes cooperativas, que repassam automaticamente as atualizações ocorridas ao Monitor. O identificador do Monitor é enviado à ação.
- *Integrate-Non-Cooperative-Source*: para fontes não cooperativas, onde o Monitor é responsável por verificar e carregar as atualizações ocorridas na fonte. O identificador do Monitor, o último ABSN pesquisado e a periodicidade de verificação das atualizações são enviados à ação.

A ação Integrate-Cooperative-Source é apresentada a seguir:

- Integrate-Cooperative-Source::action[receiving a
  string | binding | storing | diverging | communicating]
  [using current bindings | current buffer | current storage].
- Translated-Update-List = list of tuples.
- Log-List = list of tuples.
- Errors-List = list of tuples.

```
(5) Integrator-Cooperative-Source =
    | | allocate a cell
    then
    | | | bind the given cell to "Monitor"
    | | and store the given string#1 in it
  hence
    | unfolding
    | | | | receive a message [from the agent stored in the cell bound to "Monitor"]
    | | | |       [containning a Translated-Update-List]
    | | | then send a message [to the agent bound to "Loader"
    | | |       [containning the contents of the given message]
    | | then receive a message [from the agent bound to "Loader"]
    | | |       [containning a (Log-List, Errors-List)]
    | | then
    | | | | | check not(the given list#2 is empty-list)
    | | | | and then Carry-Errors(the contents of the given message)
    | | | or check(the given list#2 is empty-list)
    | | and then unfold
```

Na primeira parte dessa ação a identificação do agente Monitor é armazenada numa célula de memória, chamada Monitor, para posteriores utilizações.

Na segunda parte da ação (após o *hence*), dentro de um *loop* infinito, o Integrador aguarda a chegada de uma lista chamada *Translated-Update-List*, contendo as atualizações da fonte externa do *data warehouse*. A periodicidade com que essas atualizações são enviadas é ajustada na própria fonte de dados por meio de uma rotina pré-programada.

Quando a lista *Translated-Update-List* é recebida, o Integrador a envia ao componente Carregador, e aguarda o retorno de duas listas por ele enviadas, indicando a finalização das atividades:

- *Log-List*: arquivo de *log* contendo as operações realizadas no processo de carregamento de dados para as tabelas da Base de Dados de Trabalho.
- *Errors-List*: arquivo de erros ocorridos durante o processo de carregamento de dados para as tabelas da Base de Dados de Trabalho, que poderá ser vazio quando não existirem erros.

Após o recebimento dessas listas, o Integrador verifica se a lista de erros *Errors-List* recebida está vazia e em caso negativo, notifica o sistema da necessidade de realizar-se a correção de erros através do procedimento *Carry-Errors*. Então, a ação volta ao início do *loop* para recomeçar o processo.

A ação *Carry-Errors* é dependente de implementação e representa um procedimento não automatizado para a correção dos erros ocorridos no carregamento das atualizações das fontes para as tabelas da Base de Dados de Trabalho.

- Carry-Errors(_,_)::Errors-List,Log-List→action

(1) Carry-Errors (L1, L2) = □

A ação Integrate-Non-Cooperative-Source é apresentada a seguir.

- Integrate-Non-Cooperative-Source::action

[receiving a (string, integer, integer) | binding | storing | diverging | communicating]
[using current bindings | current buffer | current storage]

- Translated-Update-List = list of tuples
- Log-List = list of tuples
- Errors-List = list of tuples

```
(6) Integrator-Non-Cooperative-Source =
    allocate a cell
  then
    bind the given cell to "Monitor"
    and store the given string#1 in it
  and
    allocate a cell
  then
    bind the given cell to "ABSN"
    and store the given integer#2 in it
  and
    allocate a cell
  then
    bind the given cell to "Wait-Time"
    and store the given integer #3 in it
hence
  unfolding
      send a message[to the agent stored in the cell bound to "Monitor"]
            [containning the integer stored in the cell bound to "ABSN"]
    then receive a message [from the agent stored in the cell bound to "Monitor"]
            [containning a (ABSN,Translated-Update-List)]
    then store the given ABSN#1 in the cell bound to "ABSN"
    and
        check not(given list#2 is empty-list)
      and then
        send a message [to the agent bound to "Loader"]
                       [containning the given list#2]
      then receive a message [from the a gent bound to "Loader"]
                       [containning a (Errors-List,Log-List)]
      then
          check not(the given list#1 is empty-list)
        and then Carry-Errors (the contents of the given message)
      or check(the given list#1 is empty-list)
    or check(the given list#2 is empty-list)
  and then Sleep "Wait-Time"
  and then unfold
```

Essa ação recebe uma tupla de dados contendo a identificação do Monitor, o último ABSN verificado e a periodicidade para verificar as atualizações nas fontes e armazena essas informações em células de memória chamadas de Monitor, ABSN e *Wait-Time* respectivamente.

A segunda parte da ação (após o *hence*) é responsável por verificar periodicamente as atualizações ocorridas na fonte de dados. A periodicidade com que essas verificações são realizadas é determinada pela variável *Wait-Time*, cujo valor foi recebido no início da ação.

Dentro de um *loop* infinito, o Integrador envia uma mensagem para o Monitor, contendo o último ABSN verificado no arquivo de *log* da fonte, a partir do qual serão pesquisadas as atualizações na fonte de dados.

Então, o Integrador aguarda a chegada de uma mensagem do Monitor, contendo o último ABSN pesquisado no arquivo de *log* da fonte e uma lista chamada *Translated-Update-List*, contendo as atualizações ocorridas nos dados da fonte.

Em seguida, o ABSN recebido é armazenado na célula de memória chamada ABSN e, se a lista *Translated-Update-List* não estiver vazia (indicando que existem atualizações nos dados da fonte), ela é enviada ao componente Carregador, responsável por carregar as atualizações para as tabelas da Base de Dados de Trabalho.

Em seguida, com a ação *Sleep*, o Integrador permanece em estado de espera, de acordo com o valor armazenado no produtor *Wait-Time*, antes de verificar novamente as atualizações das fontes de dados. Essa ação é a mesma utilizada pelo Integrador na ação *Integrator-Action* apresentada acima.

6 Conclusões

A utilização de *data warehouses* está sendo considerada uma boa solução para os problemas relativos à construção de sistemas de suporte à decisão.

Por se tratar de uma tecnologia relativamente recente, ainda existem questões a serem resolvidas, não constituindo um consenso, sendo a definição da arquitetura do sistema de *data warehousing* um exemplo.

Algumas arquiteturas de *data warehousing* foram definidas [1, 6, 17-19], mas essas definições não são formais. A descrição formal pode ser utilizada para chamar a atenção a detalhes que muitas vezes são negligenciados durante o projeto de um *data warehouse,* contribuindo para o processo de padronização desta tecnologia.

No início do desenvolvimento do projeto SAGU, participamos da escolha e da definição da arquitetura do *data warehouse.* Dentre as várias abordagens estudadas [1, 17, 18], tomamos por base a arquitetura utilizada no Protótipo WHIPS [6, 17, 19], por se tratar de uma proposta genérica e bastante aceita na literatura.

A especificação formal foi de extrema importância na concepção da arquitetura do projeto SAGU, sendo feita simultâneamente à definição da mesma, proporcionando *feedback* para todas as partes envolvidas, antecipando problemas que possivelmente poderiam surgir e agilizando o desenvolvimento de algumas operações, por prover um material concreto de apoio.

A escolha do formalismo de Semântica de Ações para realizar as especificações dos módulos de *software* do *data warehouse* do projeto SAGU deve-se ao seu alto grau de modularidade, extensibilidade, reusabilidade, que são especialmente desejáveis durante projetos do *software* [20].

Outra característica importante que influenciou a escolha do formalismo de Semântica de Ações é que as ações são escritas como frases em inglês (mas completamente formais), facilitando sua leitura e entendimento (pelo menos superficial) por pessoas leigas no assunto.

A utilização de métodos formais para realizar a descrição de sistemas de *software* é explicável pela necessidade de se construir sistemas que ofereçam propriedades como correção, confiabilidade e segurança. Os métodos formais oferecem a vantagem de permitir a verificação formal dessas propriedades, embora nem sempre isso seja fácil.

Como os módulos componentes do projeto SAGU são relativamente pequenos, conseguimos descrever com relativa simplicidade as propriedades de suas especificações, mas acreditamos ser imprescindível a utilização de ferramentas automatizadas, para sistemas grandes e complexos, sem as quais seria extremamente difícil a sua utilização.

Outra grande vantagem da utilização de métodos formais para especificação dos módulos componentes do sistema da *data warehouse* do projeto SAGU é que eles puderam ser usados como meio de comunicação entre os participantes do projeto, que concordaram sobre o seu significado, devido ao fato de apresentarem uma semântica não ambígua e totalmente precisa, assim como o fato da linguagem utilizada nas especificações formais ser bem definida levou a um rigor maior nas definições, facilitando o aparecimento e o tratamento de comportamentos errôneos de módulos do sistema especificado.

Dentre as dificuldades encontradas, identificamos a especificação de tempo em Semântica de Ações. Essa característica do formalismo deve ser mudada em futuras versões do mercado.[3]

Referências

[1] B. Devlin. *Data warehouse from arquitecture to implementation.* Addison-Wesley, 1997.

[2] S. Chaudhuri e U. Dayal. *An overview of data warehousing and OLAP technology.* In: Proceedings of the ACM SIGMOD International Conference, v. 26, n. 1, 1997, p. 65-74.

[3] P. D. Mosses. *Action semantics.* Cambridge University Press, Cambridge, UK, 1992.

[4] C. Squire. *Data extraction and transformation for the data warehouse.* In: Proceedings of the ACM SIGMOD Annual Conference, San Francisco, CA: May 1995.

[5] J. Widom. *Research problems in data warehousing.* In: Proceedings of 4th International Conference on Information and Knowledge Management (CIKM), November 1995.

[6] J. L. Wiener *et al. A system prototype for warehouse view maintenance.* In: Proceedings of the ACM Workshop on Materialized Views: Techniques and Applications, Montreal, Canada: June 1996, p. 26-33.

[7] R. Kimball. *The data warehouse toolkit.* John Wiley & Sons, 1996.

[3] http://www.brics.dk/projects/AS

[8] A. Berson e S. J. Smith. *Data warehouse, data mining and OLAP.* McGraw-Hill, first edition, 1997.

[9] M. Meja-Lavalle e C. Rodrigues-Ortiz. Obtaining expert system rules using data mining tools from a power generation database. *Expert System with Application*, v. 14, 1998, p. 32-47.

[10] P. Adrians e D. Zantinge. *Data mining.* Addison-Wesley, England, first edition, 1997.

[11] S. Muggleton. *Declarative knowledge discovery in industrial databases.* In: PADD'97: Proc. of First International Conference and Exhibition on the Practical Application of Knowledge Discovery and Data Mining, 1997.

[12] A. G. Oliveira. *Data warehouse: conceitos e soluções.* Advanced Editora, Florianópolis, SC: 1998.

[13] D. A. Watt. *Programming language syntax and semantics.* Prentice Hall, UK: 1991.

[14] D. A. Schmidt. *Denotational semantics: a methodology for language development.* Allyn & Bacon, 1986.

[15] A. T. Pozo, *et al. SAGU - Sistema de apoio ao gerenciamento universitário.* Relatório Técnico do Projeto, DINF - Departamento de Informática, UFPR: agosto 2000.

[16] J. M. Hauagge. *Uma proposta de especificação formal para data warehousing.* Dissertação de Mestrado, UFPR, Brasil: Dezembro, 2000.

[17] J. Hammer, *et al. The stanford data warehousing project.* IEEE Data Engineering Bulletin: June 1995.

[18] Y. Zhuge *et al. View maintenance in a warehouse enviroment.* In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 316-327, San Jose, California: May 1995.

[19] W. J. Labio *et al. The WHIPS prototype for data warehouse creation and maintenance.* In: Proceedings of the ACM SIGMOD Conference, Tucson, Arizona, May 1997.

[20] P. D. Mosses e M. A. Musicante. *An action semantics for ML concurrency primitives.* In: FME'94: Industrial Benefit of Formal Methods, Second International Symposium of Formal Methods Europe, Barcelona, Spain, October 1994.