Especificando Protocolos em Z com Agentes

João Bosco M. Sobral * Jorge L. S. Leão Aloysio C. P. Pedroza

COPPE/Universidade Federal do Rio de Janeiro
Programa de Engenharia Elétrica
Caixa Postal 68504 - CEP 21945-700 - Rio de Janeiro - Brasil
bosco@inf.ufsc.br / leao@coe.ufrj.br / aloysio@coe.ufrj.br

Resumo *Este artigo apresenta uma proposta de linguagem para a descrição formal de sistemas concorrentes/distribuidos que integra uma abordagem comportamental baseada no CCS de Milner e as abordagens axiomáticas da linguagem Z e de uma Lógica Temporal. O objetivo é estabelecer uma estrutura formal na qual uma abordagem construtiva é integrada à abordagens axiomáticas de modo que sistemas possam ser descritos através da interação com seu ambiente, alguma forma de estruturação interna pode ser assumida, e também verificação formal dos modelos de sistemas construidos seja possível. Nós podemos descrever propriedades em Z e Lógica Temporal, e assim usar o poder de raciocínio dessas linguagens, para provar propriedades de segurança, justiça e vivacidade da especificação resultante. A linguagem utiliza o sistema de tipos de Z e permite descrever o funcionamneto interno de agentes, através de um ou mais tipos abstratos de dados baseados em estados, onde as operações internas aos agentes são descritas em Z.* Um exemplo de aplicação da linguagem é mostrado para uma parte do protocolo ABRACADABRA ISO/CCITT.

Abstract *Objective of this paper is to present a first proposal of a language for describing communication protocol or general concurrent/distributed systems. The language integrates a behavioral approach based on Milner's CCS with the Z and Temporal Logic axiomatic approaches. The intention is to establish a formal framework in which a constructive approach is joined with property-oriented approaches so that systems should be characterized by their ability to interact with their environment, some form of internal structure have to be assumed and also formal verification of the constructed protocol models is possible. We can describe properties in Z and temporal logic and so use the reasoning power of these languages to prove safety, justice and lively properties of the result specification. The language exploites the mathematical data types of Z and allows to model the internal behaviour of agents by using state-based abstract data types, in which the internal operations of the agents are described in Z. CCS and Z are suitable for development strategies based upon refinement of models. The language is applied to specify a part of the ISO/CCITT ABRACADABRA protocol.*

1 Introdução

O esforço de padronização para especificar protocolos de comunicação e sistemas distribuidos (possivelmente abertos) objetiva o uso de técnicas de descrição formal (TDF) para descrever tais sistemas.

*INE/UFSC Florianópolis - O trabalho deste autor foi realizado no programa da CAPES/PICD, através da Universidade Federal de Santa Catarina - SC

Atualmente, as entidades de padronização reconhecem três linguagens de especificação para descrever sistemas abertos: SDL [4], ESTELLE, [20] e LOTOS [2]. A linguagem SDL tem evoluido desde seu aspecto gráfico até sua representação textual. Tipos de dado (ACT-ONE) foram adicionados em SDL88 e a última versão, SDL92, contém a descrição de "Object SDL" [1]. SDL tem uma comunidade de usuários centrada em ISDN e protocolos de sinalização CCITT. LOTOS combina um cálculo de processos com tipos de dados algébricos (ACT- ONE), tem as características de linguagem declarativa e tem sido mais aplicada a protocolos com comportamentos complexos. ESTELLE é derivada de PASCAL e, portanto, apresentando algumas peculiaridades de linguagems imperativas.

Os requisitos gerais de TDF's para especificar protocolos e sistemas distribuidos abertos (ODP-ISO/IEC) são classificados em quatro grupos: Poder de Expressividade, Propriedades e Restrições, Abstração e Estruturação e mais Semântica Formal [21]. Outros aspectos que devem ser considerados são os conceitos relativos à modelagem: os conceitos básicos, os conceitos de especificação, os conceitos arquiteturiais e algum método de estruturar especificações (abordagem orientada para objetos ou baseada em estilo).

Alguns requisitos gerais para TDF's e conceitos de modelagem previstos no modelo de referência básico ODP, já estão consolidados. Entretanto, nem todas as questões relacionadas a adequação das linguagens já aceitas e a capacidade delas de representar certos conceitos de modelagem estão resolvidas. Claramente, nem todos os conceitos de modelagem são representados em uma única TDF (e é difícil que sejam), e assim TDF's distintas podem ser usadas em conjunção, observando-se os pontos fortes de cada TDF.

Uma questão importante não abordada nas técnicas SDL, ESTELLE e LOTOS, é a expressão de propriedades correspondendo ao comportamento do sistema. Consequentemente, tornando a linguagem, distante de um sistema dedutivo para verificação de propriedades. Um outra questão, por exemplo, é a adequação dessas linguagens para especificar entidades do mundo real abstraidas no contexto de objetos.

Uma outra TDF, a linguagem Z [18], tem recebido atenção por projetistas de protocolos nos últimos anos. Embora Z não tenha sido projetada para especificar sistemas distribuidos, a linguagem tem sido usada para especificar e verificar certos aspectos do Sistema de Sinalização No. 7 do CCITT [14]; [6] trata um nível de enlace, e um nível de rede é abordado em [8].

Z não é a linguagem de especificação tradicionalmente considerada no contexto da engenharia de comunicações. Entretanto, pelo alto nível de abstração que a linguagem apresenta, e dadas as características de Z para especificar dados e operações (tipos), nós contemplamos o uso da linguagem Z, já considerada pelo projeto ISO Open Distributed Processing (ODP).

Para tornar Z uma TDF adequada para sistemas concorrentes/distribuidos, uma nova alternativa de linguagem, que comporte a linguagem Z, deve permitir a representação de alguns conceitos básicos para modelagem de sistemas distribuidos.

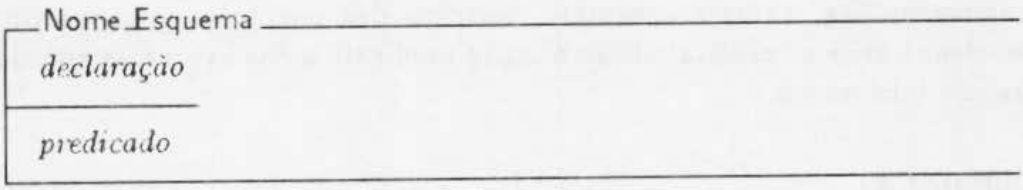
Este trabalho propõe uma linguagem declarativa em alto nível de abstração, chamada Zag (acrônimo de *Z com agentes*) que integra uma extensão da linguagem básica do Cálculo de Agentes de Milner [12], a linguagem Z, e a linguagem de uma lógica temporal. Denominamos essa extensão de CCS de linguagem CA. A abordagem comportamental de CA é adequada para se raciocinar sobre concorrência e comunicação. Z é uma linguagem adequada para descrever os estados e condições lógicas para sensibilização das operações de um sistema, e juntamente com a lógica temporal introduz lógica formal propiciando poder de expressar propriedades.

Este artigo está organizado como segue: a seção 2 contém uma breve apreciação da linguagem Z; a seção 3 resume a linguagem CA; a seção 4 apresenta a linguagem Zag; a seção 5 mostra como se descrever propriedades através de lógica formal; na seção 6 é mostrado o enfoque semantico adotado para Zag; um exemplo de aplicação da linguagem está na seção 7, e na secão 8 contém considerações finais e perspectivas.

2 A Linguagem Z

Z é uma linguagem de especificação para software sequencial, baseada nos conceitos e notações da teoria dos conjuntos, lógica clássica de primeira ordem e o Cálculo Lambda [18].

A linguagem Z tem seu método formal baseado em um modelo de transição de estados, mas também pode ser considerado como um método formal orientado a propriedades. Em Z, um sistema pode ser descrito tendo um espaço de estados e operações que produzem mudanças de estado. Estas operações são definidas por relacionar os estados antes e após cada transição. Uma decoração " ' " é aplicada para significar variáveis em um estado após à mudança de estado. Restrições e propriedades são definidas por predicados que devem ser verdadeiros, e representam relações entre componentes de estado. Uma especificação na linguagem Z é modularizada através de uma unidade sintática chamada *esquema*, na qual se declara e restringe variáveis. Um esquema pode ser utilizado como um tipo de Z. Um esquema é usualmente utilizado em sua forma gráfica como segue:



Z possui um *Cálculo de Esquemas* que contém operadores para combinar ou compor novos esquemas. Podemos considerar os elementos em uma especificação Z como: definições de tipo, definição de relações e funções, esquemas de estado, esquema de inicialização e esquemas de operações.

Algumas desvantagens em Z para especificar SDs são notadas:

1. Não existe em Z nenhuma noção de agente ou comunicação, ou tipos primitivos correspondentes ;

2. Em Z, sequências de operações simples, não são imediatamente óbvias. Seu modelo de estados não prove ao usuário da linguagem um sentimento intuitivo para simples sequências de operações . Uma especificação Z é baseada sobre estados permissíveis, descreve as transições de estados quando elas existem, mas não explicita a ordem em que operações são realizadas.

3. O conceito de ponto de interação, tão importante para sistemas distribuidos, é inexistente em Z [7]. Portanto, quando especificando operações em Z, pode não ser claro quais operações são internas ao sistema, e quais são visíveis (invocadas pelo) ao ambiente.

4. Conceitos tais como ações, eventos, comunicação e concorrência podem ser representados diretamente em Z, ou compatíveis com a semântica de Z, entretanto, tornando ausente o aspecto de simplicidade da especificação.

3 A Linguagem CA

Consideramos uma extensão da linguagem básica do cálculo de Milner (CCS), aqui denominada CA. A linguagem CA é uma versão adaptada à lingua portuguêsa, que se pretende integrar à linguagem Z [18] no sentido que esta linguagem se torne também apropriada para especificar sistemas concorrentes/distribuidos.

Em [17] é mostrada uma definição sistemática da linguagem, com seu vocabulário, sintaxe e semantica formal. Na sintaxe de CA definimos os *termos*, as expressões de valores, as expressões

booleanas, as *ações*, e as *expressões de agentes*. Dentre as extensões consideradas em CA, estão: a definição do agente de término bem sucedido **Done**, a composição sequencial de agentes, algumas formas de desabilitação de agentes herdadas de [5] e sincronização múltipla como está em [15].

Definição (*Expressões de Agentes*)

1. Se A é um símbolo de agente com aridade $n > 0$, e exp_i são expressões de valores ou booleanas, então $A[exp_1, \cdots, exp_n]$ é uma expressão de agente.

   Se $n = 0$, A é uma expressão de agente.

   O e **Done** são expressões de agentes.

2. Toda *variável de agente* é uma expressão de agente.

3. As expressões de agentes da linguagem CA podem ser vistas como aquelas apresentadas para a linguagem Zag, na seção seguinte, restritas das operações internas Op_Ref, da variável booleana *bvar* e substituindo-se a parte predicatica das expresões guardadas por uma expressão booleana b.

Definição (*Agente* CA)
Um *agente* é uma expressão de agente sem variáveis de agente, na qual nenhuma variável de termo é livre. Isto é, as únicas variáveis permitidas são variáveis de termo que ocorrem *ligadas*.

Algumas características de CA que podem ser mencionadas são: (1) cada especificação CA pode ser interpretada como um sistema de transições rotuladas (LTS) que é um tipo de autômato, mas a noção de estado não é explicitamente considerada; (2) os elementos da linguagem que contém valor (as variáveis) são elementos abstratos que não fazem referência à memória física de um computador, sendo o conceito de varíavel em CA no sentido da linguagem de lógica formal; (3) a linguagem descreve a escolha entre diferentes comportamentos, nas duas formas que esta escolha pode tomar: externa, feita em cooperação com o ambiente, e interna (ações τ), que não pode ser afetada pelo ambiente; (5) comunicação é modelada por "rendez-vous"; (6) no sentido de facilitar especificação de sistemas OSI, CA tem as três formas de desabilitar agentes como proposta em [5]; (7) sincronização múltipla é definida com semântica dada em [15].

Uma desvantagem de CA é notada para especificar sistemas concorrentes/ distribuidos: CA define as possíveis sequências de ações de um agente , mas não existe uma maneira direta de expressar traços. Consequentemente, não é uma boa linguagem para expressar propriedades de vivacidade.

4 Z com Agentes: Zag

Zag é uma linguagem que integra as linguagens CA, Z e Lógica Temporal, se utilizando do mesmo modelo-base de transições que estas linguagens possuem. A integração ocorre primeiramente entre CA e Z, sobre o modelo formal de um sistema de transições rotuladas definido por:

Definição (*Um Sistema de Transições Rotuladas* Zag)
Definimos um sistema de transições rotuladas LTS_{Zag} para a linguagem Zag através da seguinte estrutura:

$$LTS_{Zag} = (\mathcal{L}, \Sigma, p_{init}, P)$$

onde:

- $\mathcal{L}$ representa o conjunto de rótulos ("labels") de ações observáveis de CA ;
- Σ é um conjunto não-vazio de estados, onde um estado qualquer é representado pela metavariável σ ;
- p_{init} é um predicado de iniciação ;
- P é um conjunto de predicados correspondendo às ações definidas em CA:

$$P = (\{ (a, p_a) \mid a \in \mathcal{L} \} \cup \{ (\tau, p_\tau) \mid \tau \in Act \})$$

onde p_a é um predicado definido sobre $\Sigma \times \Sigma$, o qual descreve o efeito da execução de uma ação observável a de CA, sobre os estados σ e σ', respectivamente antes e após a ação a ocorrer ; p_a é tal que $p_a \subseteq (\Sigma \times \Sigma)$.

p_τ denota um predicado referente a uma operação interna de um agente descrito em Zag, decorrente de uma ação interna τ em CA, caso exista na especificação CA.

Zag tem o propósito de usar ações, variáveis e dados com operações (tipos) e suportando também concorrência e comunicação. A idéia básica que norteia uma especificação Zag é revelar os estados implícitos de um agente descrito em CA e descrever os efeitos das execuções das ações possíveis (observáveis e internas) de CA sobre esses estados. Tais efeitos são materializados por esquemas de operações de Z. Zag mostra os aspectos internos de um agente, construindo um ou mais *tipos abstratos de dados baseados em estado* [11] [19] inerentes ao funcionamento interno de um agente CA. O sistema de tipos de Z é usado para declarar tipos não considerados no nível de abstração de CA e a linguagem Zag define funções usando Z. Todos os símbolos operadores definidos para CA são herdados por Zag, no sentido de se preservar a semântica de uma especificação CA quando se transforma (refinamento) um agente de CA para Zag.

Definição (*Agentes* Zag)

1. Se A é um símbolo de agente de aridade $n > 0$, e exp_i são expressões de valores ou booleanas, então $A[exp_1, \cdots, exp_n]$ é um agente.

   Se $n = 0$, A é um agente.

   O e **Done** são agentes.

2. Se E, E_i são agentes Zag, A um símbolo de agente Zag, exp é uma expressão de valor ou booleana, $bvar$ é uma variável booleana, $x_1, \cdots, x_n$ são variáveis de termo, c é uma constante de termo, l_i e m_i são símbolos de portas de entrada ou de saída, $l!(x_1, \cdots, x_n)$, $l!(e_1, \cdots, e_n)$ são ações observáveis, e w é um inteiro não negativo, então as seguintes cadeias são *agentes* na linguagem Zag:

   - $l?(x_1, \cdots, x_n)$; [OpRef] ; E (Prefixação)
     $l!(e_1, \cdots, e_n)$; [OpRef] ; E
   - $E_1 + E_2 + \cdots + E_n$ (Escolha Não- Determinística)
   - $E_1 \mid E_2$ (Composição Paralela)
   - $E \setminus \{ l_1, \cdots, l_n \}$ (Restrição de portas)
   - $E_1 \parallel E_2$ (Composição Paralela restrita) $(E_1 \mid E_2)$ {...}
   - $E[l_1/m_1, \cdots, l_n/m_n]$ (Rerrotulação de portas)
   - OpRef . $bvar \rightarrow E$ ou $f(x_1, \cdots, x_n) = c \rightarrow E$ (Guarda)

- $E_1 \,|||\, E_2$ (Intercalação)
- $E_1 \gg E_2$ (Composição Sequencial)
- $E_1 \gg [x_1, \cdots x_n] E_2$ (Comp. Seq. com passagem de valores)
- $E_1 [> E_2$ (Desabilitação)
- $E_1 [>_{|||} E_2$ (Desabilitação por Intercalação)
- $E_1 [>_L E_2$ (Desabilitação por Porta)
- $A [exp_1, \cdots, exp_n]$ (Instanciação de Agente)
- $(\alpha_1 \, \alpha_2 \, \cdots \, \alpha_{n+1})^{\omega}, \quad \alpha \in \mathcal{L}$ (Simultaneidade de ações e Repetição)
- $(E_1 \&_L E_2) \cdots \&_L E_n$ (Sincronização Múltipla)

onde OpRef representa uma operação interna do agente, descrita por um esquema de Z, e f corresponde a um símbolo funcional.

Definição (*Equação de Definição de Agente em* Zag)

- Se A é um símbolo de agente Zag e E é um agente Zag, então $A \triangleq E$ é uma equação de definição de agente em Zag.

Definição (*Estrutura Sintática de um Agente* Zag)

A integração sintática que define uma típica definição de um *agente ordinário* na linguagem Zag é definida na figura 1. Em Zag é utilizada uma outra estrutura sintática para a definição de um *agente-sistema* representado por **System specification** e **End specification**, como mostra o exemplo na seção 7.

A estrutura sintática de um *agente ordinário* Zag introduz as Definições Locais, que resumem as declarações de tipo (básico ou livre), as descrições axiomáticas (constantes, variáveis e funções), abreviações, e referência a agentes.

Semanticamente, um agente Zag é definido considerando o conceito de agente da linguagem CA e mais a especificação de um ou mais tipos abstratos de dados inerentes ao comportamento interno do agente, cujas operações são referenciadas na equação de definição do agente.

Sendo a estrutura de agentes Zag definida por componentes sintáticos opcionais, o não uso dos componentes inerentes à linguagem Z, faz com que tenhamos a especificação dos agentes de um sistema como na linguagem CA.

5 Descrevendo Propriedades

Propriedades do sistema ou de agentes componentes do sistema podem ser especificadas usando Z ou uma *lógica temporal linear*. Objetivamos utilizar um sistema dedutivo para verificar propriedades. Neste caso, cada execução individual dos agentes de um sistema produz uma *história*: qualquer caminho no espaço de estados consistindo de uma sequência infinita de estados e operações (transições). Na abordagem adotada neste trabalho, o comportamento de um agente Zag é estabelecido pelo conjunto de suas histórias possíveis (nem toda história é possível), que são as trajetórias legais do agente.

Ao invés de se enumerar um conjunto de histórias, usamos uma fórmula de lógica temporal para caracterizar tal conjunto. Uma tal fórmula denota uma propriedade, que é uma fórmula satisfazendo a todas as histórias de um agente. Especificações de um comportamento requerido são fórmulas que satisfazem algum subconjunto das histórias possíveis, isto é, uma especificação

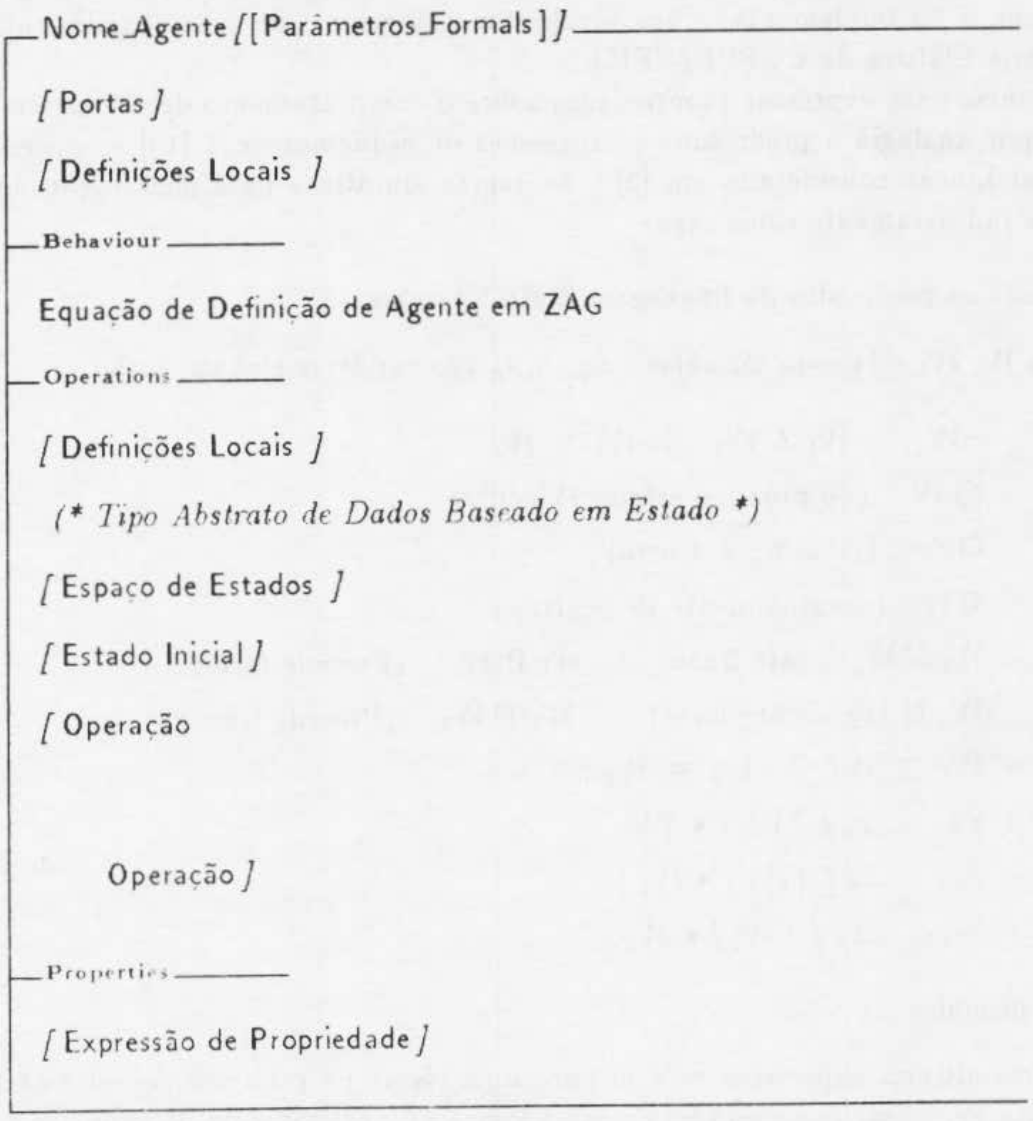


Figura 1: Estrutura sintática de um agente Zag

de um comportamento requerido impõe restrições adicionais sobre o comportamento permitido. Por conseguinte investigam-se propriedades de histórias, e a lógica temporal adequada é a lógica temporal linear, em cuja semântica cada instante possui um único futuro possível. Assim, uma fórmula temporal ρ será uma propriedade válida de um agente A se cada história do conjunto satisfizer ρ.

Uma variedade de lógicas temporais existem na literatura. Dada a condição de ortogonalidade existente entre os formalismos utilizados em Zag, o usuário pode mesmo escolher uma lógica temporal que se adapte melhor a sua aplicação. Propomos que a lógica temporal a ser integrada a Zag seja a lógica temporal linear que está em [3] e é utilizada para verificar programas concorrentes. Essa lógica tem seu sistema dedutivo baseado num procedimento de decisão de *tableau*, e foi implementado um verificador automático de propriedades, no Programa de Engenharia Elétrica da COPPE/UFRJ.

Fórmulas para expressar propriedades sobre o comportamento de um sistema, foram construidas por analogia a predicados e expressões de esquemas de Z [19] e expressões da Lógica Temporal Linear considerada em [3]. As regras sintáticas para construção de *fórmulas* são definidas indutivamente como segue:

- Todo os predicados da linguagem Z são fórmulas ;
- Se $\mathcal{W}$, $\mathcal{W}_1$ e $\mathcal{W}_2$ são fórmulas e $x_1, ..., x_n$ são variáveis globais, então:
  - $\neg\mathcal{W}$ $\quad \mathcal{W}_1 \wedge \mathcal{W}_2$ $\quad \mathcal{W}_1 \vee \mathcal{W}_2$;
  - $\bigcirc \mathcal{W}$ (no próximo estado $\mathcal{W}$ ocorre) ;
  - $\Box \mathcal{W}$ ($\mathcal{W}$ sempre ocorre) ;
  - $\Diamond \mathcal{W}$ (eventualmente $\mathcal{W}$ ocorre) ;
  - $\mathcal{W}_1 \, \mathcal{U} \, \mathcal{W}_2$ (até fraco) $\quad \mathcal{W}_1 \, \mathbf{P} \, \mathcal{W}_2$ (Precede forte) ;
  - $\mathcal{W}_1 \, \mathbf{U} \, \mathcal{W}_2$ (Até forte) $\quad \mathcal{W}_1 \, \mathcal{P} \, \mathcal{W}_2$ (Precede fraco) ;
  - $\mathcal{W}_1 \supset \mathcal{W}_2$ $\quad \mathcal{W}_1 \equiv \mathcal{W}_2$;
  - $\forall x_1, ..., x_n \, [\, | \mathcal{W}_1] \bullet \mathcal{W}_2$;
  - $\exists x_1, ..., x_n \, [\, | \mathcal{W}_1] \bullet \mathcal{W}_2$;
  - $\exists_1 x_1, ..., x_n \, [\, | \mathcal{W}_1] \bullet \mathcal{W}_2$

  são fórmulas.

As três últimas expressões servem para uma lógica de primeira ordem e as demais podem servir para expressar propriedades em uma lógica proposicional ou de primeira ordem.

6 Semântica Formal de Zag

Esta seção resume o enfoque semantico adotado para Zag. A definição semantica é decorrente das semânticas formais de CA [17] e a semântica formal estabelecida para Z em [18]

A semântica de Zag consiste de duas partes:

- um modelo que define a semântica de um agente ;
- o modelo semântico apropriado para combinar agentes, definido com base na semantica operacional de CA e na definição do sistema de transições rotuladas LTS_{Zag} da seção 4.

Modelo Semântico de Agentes

Informalmente, a definição de um agente Zag tem quatro componentes como mostrado na figura 1.

1. declaração do preâmbulo: portas, tipos básicos, variáveis e funções ;
2. parte comportamental que descreve ações e referencia as operações internas ;
3. definição do tipo abstrato de dados baseado em estado ;
4. especificação de propriedades ;

Para dar a semântica formal de um agente Zag, nós tratamos três partes:

- a semântica do preâmbulo ;
- a semântica da parte comportamental ;
- a semântica de definições locais, o tipo abstrato de dados baseado em estado (estados e operações) e propriedades.

Semântica do Preâmbulo

As portas de um agente são descritas através de descrições axiomáticas sem a parte predicatica. Os tipos básicos seguem a semântica dada em Z. Declarações de variáveis seguem a semântica de descrições axiomáticas em Z, e funções tem sua semântica ou por meio de suas descrições axiomáticas ou por meio de expressões-Z do Cáculo Lambda.

Uma primeira tentativa se definir a semântica desta parte, usa o mesmo modelo formal utilizado em [18]. Definimos um esquema de Z (possivelmente sem sua parte predicatica) para descrever a semântica do preâmbulo.

Semântica da Parte Comportamental

Na linguagem CA foi definido o conjunto de símbolos operadores para agentes [17]. Para se ter preservação de semantica, este conjunto foi designado para ser o mesmo para a linguagem Zag. Nós escrevemos a parte comportamental de um agente (ou sistema), através de sua equação de definição do agente, a qual envolve operadores de Zag. Assim, os operadores Zag são investigados no sentido de prover os seus significados. Usamos a base formal do sistema de transições rotuladas LTS_{Zag} definido antes, e assim os operadores Zag podem ser formalmente descritos como segue:

Sejam R, E e F agentes Zag ; $\mathcal{L}(R)$ é um *sort* de R (see [12]) ; uma ação a é $l?(x_1, \cdots, x_n)$ ou $l!(e_1, \cdots, e_n)$; σ and σ' são estados em Σ, respectivamente antes e após a ação a ter ocorrido ; E_a e F_a denotam predicados p_a tal que $p_a \subseteq (\Sigma \times \Sigma)$:

Semântica para *Prefixação*

Seja $R \triangleq a \; ; \; [\,\mathsf{Op\_Ref}\,] \; ; \; E$, então

$$\mathcal{L}(R) \triangleq \{a\} \cup \mathcal{L}(E)$$

$$\Sigma_R \triangleq \Sigma_E$$

$$R_{init} \triangleq E_{init}$$

$$R_a \triangleq E_a \qquad \forall a \in \mathcal{L}(R)$$

Semântica para *Escolha de Agentes*

Seja $R \triangleq E + F$; então

$\mathcal{L}(R) \triangleq \mathcal{L}(E) \cup \mathcal{L}(F)$

$\Sigma_R \triangleq \Sigma_E - \Sigma_F \quad$ ou $\quad \Sigma_F - \Sigma_E$

$R_{init} \triangleq E_{init} \wedge F_{init}$

$R_a \triangleq (E_a \wedge \sigma'_F = \sigma_F) \vee (F_a \wedge \sigma'_E = \sigma_F) \qquad \forall a \in \mathcal{L}(R)$

Semântica para *Intercalação*

Seja $R \triangleq E \,|||\, F$; então

$\mathcal{L}(R) \triangleq \mathcal{L}(E)\ (= \mathcal{L}(F))$

$\Sigma_R \triangleq \Sigma_E \times \Sigma_F$

$R_{init} \triangleq E_{init} \wedge F_{init}$

$R_a \triangleq (E_a \wedge \sigma'_F = \sigma_F) \vee (F_a \wedge \sigma'_E = \sigma_F) \qquad \forall a \in \mathcal{L}(R)$

Semântica para *Composição Paralela*

Seja $R \triangleq E \mid F$, então

$\mathcal{L}(R) \triangleq \mathcal{L}(E) \cup \mathcal{L}(F)$

$\Sigma_R \triangleq \Sigma_E \times \Sigma_F$

$R_{init} \triangleq E_{init} \wedge F_{init}$

$R_a \triangleq E_a \wedge \sigma'_F = \sigma_F \qquad$ if $a \in (\mathcal{L}(E) - \mathcal{L}(F))$

$R_a \triangleq F_a \wedge \sigma'_E = \sigma_E \qquad$ if $a \in (\mathcal{L}(F) - \mathcal{L}(E))$

$R_a \triangleq E_a \wedge F_a \qquad$ if $a \in (\mathcal{L}(E) \cap \mathcal{L}(F))$

Outros operadores Zag podem ser descritos do mesmo modo.

Semântica da parte do Tipo Abstrato de Dados e Propriedades

O tipo abstrato de dados baseado em estado de um agente é definido por um conjunto de esquemas de Z, sendo os esquemas de operações do tipo, referenciados na equação de definição do agente da parte comportamental. Além disso, a descrição de propriedades de um agente segue o modelo de histórias como definido antes, na seção 5. A semântica formal do tipo abstrato de dados baseado em estado e propriedades de um agente, é dada por uma *variedade* na qual sua visão formal incorpora o modelo de histórias que é usado para dar significado a propriedades [17].

7 Aplicando Zag a Protocolos

Especificamos nesta seção uma parte do Protocolo ABRACADABRA (ISO/CCITT) definido em [16]. Tal protocolo permite a transmissão confiável de mensagens sdu's entre dois usuários *Usu*[0] e *Usu*[1], conectados e prescindindo do serviço orientado à conexão, provido por duas entidades de protocolo *Abrac*[0] e *Abrac*[1]. Essas entidades se comunicam através do serviço

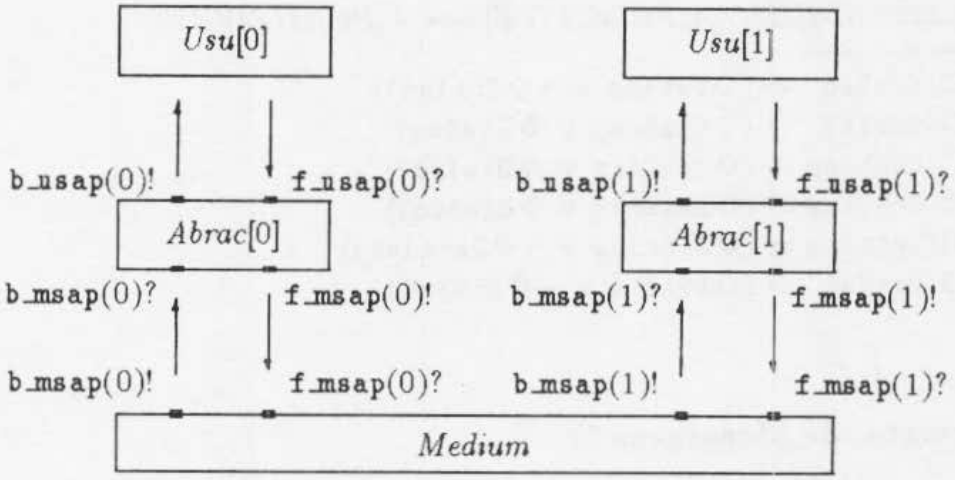


Figura 2: Abracadabra Protocol

de uma camada inferior, representada por *Medium*, que permite a transmissão bidirecional e simultânea de mensagens pdu's, não garantindo a entrega das mesmas. As portas nos agentes são rotuladas por `f_usap(0)?`, `f_msap(0)!`, `b_msap(0)?`, `b_usap(0)!`, `f_usap(1)?`, `f_msap(1)!`, `b_msap(1)?`, `b_usap(1)!`, `b_msap(0)!`, `f_msap(0)?`, `b_msap(1)!` e `f_msap(1)?`, com mostrado na figura 2. No que segue é mostrado um exemplo de aplicação da linguagem. Uma entidade *Abrac*[*i*] tem, basicamente, três fases de funcionamento: abertura de conexão, transferência de dados e liberação de conexão, mas as duas últimas fases não estão mostradas no exemplo. A especificação no seu todo é encontrada em [17].

System specification *Protocolo_Abracadabra*

(* **Variáveis** *)

$$\begin{array}{|l} i : \mathbb{N} \\ \hline i \in 0..1 \end{array}$$

(* **Tipos Básicos** *)

$[ABRAC\_ADDRESS, DATA\_TYPE]$

$CODEPDU ::= \texttt{CR} \mid \texttt{CC} \mid \texttt{DT} \mid \texttt{AK} \mid \texttt{DR} \mid \texttt{DC}$

$SEQNUM \triangleq 0..1$

(* **Tipos de Mensagens** *)

$[SDUMSG]$

$PDUMSG ::= CRpdu \mid CCpdu \mid DTpdu \mid AKpdu \mid DRpdu \mid DCpdu$

(* **Especificação do Serviço** *)

[*A_PRIMITIVE*]

ConReq, ConInd, ConResp, ConConf : *A_PRIMITIVE*
DataReq, DataInd : *A_PRIMITIVE*
DisReq, DisInd : *A_PRIMITIVE*
f_sdu : *SDUMSG*[*A_PRIMITIVE*] $\rightarrowtail$ *A_PRIMITIVE*

$\Box$(**ConReq** $\Rightarrow$ ($\Diamond$**ConInd** $\vee$ $\neg$ $\Diamond$**ConInd**))
$\Box$(**ConInd** $\Rightarrow$ ($\Diamond$**ConResp** $\vee$ $\Diamond$**DisReq**))
$\Box$(**ConResp** $\Rightarrow$ ($\Diamond$**ConConf** $\vee$ $\Diamond$**DisInd**))
$\Box$(**ConConf** $\Rightarrow$ ($\Diamond$**DataReq** $\vee$ $\Diamond$**DisReq**))
$\Box$(**DataReq** $\Rightarrow$ ($\Diamond$**DataInd** $\vee$ $\neg$ $\Diamond$**DataInd**))
$\Box$(**DisReq** $\Rightarrow$ ($\Diamond$**DisInd** $\vee$ $\neg$ $\Diamond$**DisInd**))

⋮

(* Formatos de Mensagens *)

⋮

CRpdu

---

cod : *CODEPDU*
loc_add : *ABRAC_ADDRESS*
rem_add : *ABRAC_ADDRESS*

⋮

Referenced Agents *AbracProtocol*

End specification

(* Especificando o protocolo *Abracadabra* *)

AbracProtocol

(* Composição Paralela dos agentes do protocolo ABRACADABRA *)

Behaviour

AbracProtocol $\triangleq$

(*Abrac*[0] |

Medium |

Abrac[1]) \

{**f_msap**(0), **b_msap**(0), **f_msap**(1), **b_msap**(1)}

Operations

ProtocolState $\widehat{=}$

AbracState[0] $\times$ *MediumState* $\times$ *AbracState*[1]

InitAbrac $\widehat{=}$ *InitAbrac*[0] $\wedge$ *InitMedium* $\wedge$ *InitAbrac*[1]

(* Modelando um agente que representa uma entidade ABRACADABRA *)

Abrac[*i*]

(* Ports *)

| **f_usap**?(*i*), **b_usap**!(*i*) : *PORT*[*SDUMSG*]

| **f_msap**!(*i*), **b_msap**?(*i*) : *PORT*[*PDUMSG*]

CONNECTION_STATE ::= *idle* | *waiting* | *open* | *closing*

(* Constantes *)

(* Um tempo máximo para o estabelecimento de conexão é especificado, em *tmax_A*. O protocolo estabelece um número máximo de tentativas para estabelecer uma conexão, transferir dados ou liberar conexão, através da variável *max_attempts*. *)

tmax_A : **R**	*max_attempts* : **N**
tmax_A = ···	*max_attempts* = ···

(* Variáveis *)

| *x_sdu* : *SDUMSG*;

| *x_pdu* : *PDUMSG*;

| *cond*! : *BOOL*;

| *local_address*, *remote_address* : *ABRAC_ADDRESS*

(* Funções *)

BuildCR : *CRpdu* ⟶ *PDUMSG*
BuildCR = λ *CRpdu* • (*cod* = **CR** ∧ ...)

⋮

Code : *PDUMSG* ⟶ *CODEMSG*
Code = λ *PDUMSG* • (...)

Referenced Agents *Data_Transfer*[*i*], *Connection_Closing*[*i*]

Behaviour

```
(* Fase de estabelecimento de conexão *)
Abrac[i] ≙
  (
    (* Recebe pedido de conexão do usuário *)
    f_usap(i)?(x_sdu) ;
    (CheckConReq[i].cond! → AcceptConReq[i] ;
              f_msap(i)!( BuildCR ) ;
              time!(tmax_A) ;
              A_ConConfirm[i] )

    +

    (* Recebe pedido remoto do meio *)
    b_msap(i)?(x_pdu) ;
    ( Code(x_pdu) = DR → ClearReq[i] ; Abrac[i]

      +

     Code(x_pdu) = CR → CallReq[i] ;
              b_usap(i)!(ConInd) ;
              A_ConResponse[i] )
  )

(* Recebe pedido de confirmação remota *)
A_ConConfirm[i] ≙
  (
    b_msap(i)?(x_pdu) ;
    ( Code(x_pdu) = CC → CallAcc[i] ;
              b_usap(i)!(ConConf) ;
              Data_Tranfer[i]

      +

     Code(x_pdu) = CR → CallReq[i] ;
              b_usap(i)!(ConConf) ;
              Data_Transfer[i]

      +

     Code(x_pdu) = DR → ClearReq[i] ;
              b_usap(i)!(DisInd) ;
              f_msap(i)!(BuildDC) ;
              Abrac[i]

      +

     otherwise → A_ConConfirm[i] )

    +
```

```
f_usap(i)?(x_sdu);
    (
      CheckDisReq[i].cond! →
        ClosingConn[i];
        f_msap(i)!(BuildDR); time!(tmax_F);
        Connection_Closing[i]
    )
+
⋮
```

Operations

AbracState[i]

$discon\_received : BOOL$
$retrans\_attempts : \mathbb{N}$
$state\_conn : CONNECTION\_STATE$

$retrans\_attempts \leq max\_attempts$

InitAbrac[i]

AbracState[i]

$state\_conn = idle$
$discon\_received = \mathbf{false}$
$retrans\_attempts = 0$

⋮

(* Esta operação descreve a aceitação do pedido de estabelecimento de conexão *)

AcceptConReq[i]

ΔAbracState[i]

$local\_address = x\_sdu.loc\_add$
$remote\_address = x\_sdu.rem\_add$
$state\_conn = idle \Rightarrow state\_conn' = waiting$

⋮

(* A operação seguinte trata a chegada de uma "pdu" CR, e muda o estado da conexão *)

CallReq[i]

ΔAbracState[i]

$local\_address = CRpdu.data(0)$
$remote\_address = CRpdu.data(1)$
$state\_conn = idle \Rightarrow state\_conn' = waiting$
$state\_conn = waiting \Rightarrow state\_conn' = open$
$discon\_received = \mathbf{false}$

```
    :
(* Um pedido de liberação de conexão é tratado
após a chegada de uma "pdu" DR. *)
ClearReq[i]
ΔAbracState[i]
---
discon_received' = true
state_conn = idle ⇒ state_conn' = state_conn
state_conn = waiting ⇒ state_conn' = idle
```

(* Especificando o meio de comunicação *Medium* *)

Medium

(* Ports *)

| f_msap(0)?, b_msap(1)! : *PORT*[*PDUMSG*]

| f_msap(1)?, b_msap(0)! : *PORT*[*PDUMSG*]

(* Variables *)

| x, y, w, z : *PDUMSG*;

| *lost_01*, *lost_10* : *PDUMSG*;

Referenced Agents *Channel_01*, *Channel_10*, *LostPdu*

Behaviour

```
Medium ≙ Channel_01 | Channel_10 | LostPdu

Channel_01 ≙
    (
      f_msap(0)?(x); MRecPdu_01;
        ( MSendPdu_01; b_msap(1)!(y);
            Medium
             +
            α!(x); LosePdu_01; Medium )
    )

Channel_10 ≙ ( ··· )
```

LostPdu $\triangleq$

($\alpha?(lost\_01)$; *Medium* + $\beta?(lost\_10)$; *Medium*)

Operations

MediumState

s : *PDUMSG*;
t : *PDUMSG*

⋮

(* O agente *Channel_01* recebe uma "pdu" e muda seu estado *)

MRecPdu_01

ΔMediumState

$s' = x$

(* O agente *Channel_01* envia uma "pdu" e não muda seu estado *)

MSendPdu_01

ΞMediumState

$y = s$
$s' = s$

(* O agente *Channel_01* perde uma "pdu". *)

LosePdu_01

ΔMediumState

$s = x$
$s' \neq x$

⋮

Properties

(* Uma pdu não é perdida indefinidamente. *)

$\neg \Diamond \Box (\bigcirc$ **LosePdu_01** $)$ ...

Algumas observações importantes podem ser colocadas sobre uma especificação Zag:

Tudo o que é para ser especificado é parte da estrutura sintática que inicia em **System specification** < *nome_sistema* > e finaliza em **End specification**. Tal estrutura define um *agente*, no nível mais alto da hierarquia de uma especificação Zag. Nela, a visão mais abstrata do sistema é mostrada. O sistema pode ser representado como um único agente, o *agente-sistema*. Essa estrutura sintática está basicamente, dividida em duas partes: uma parte diz respeito, às *declarações do sistema*, envolvendo as declarações globais, e uma segunda parte, que define a *arquitetura do sistema*, por referenciar um agente cujo comportamento especifica os diversos subagentes que interagem entre si, e os tipos de associações existentes entre os subagentes identificados como componentes do sistema. Assim, podemos dizer que um sistema é uma hierarquia de definições de agentes.

As definições de tipos de dados globais fornecem os tipos reconhecidos pelo ambiente e pelo

sistema. Esses tipos são vistos pelo ambiente (usuários) e pelo sistema (protocolo), e assim estão disponíveis para a comunicação entre o sistema e seu ambiente. Os tipos definidos dentro da definição de um agente, só são visíveis dentro do agente, e por conseguinte não servem para a comunicação entre o sistema e seu ambiente.

Os elementos da linguagem que contém *valor* (constantes, variáveis e funções) são elementos abstratos que não fazem referência à memória física de um computador, sendo o conceito de variável em Zag, no sentido da linguagem de lógica formal, não havendo separação desses elementos em estruturas sintáticas distintas (apenas comentários foram colocados entre (* e *).

A especificação descreve o serviço oferecido por uma entidade *Abrac*[*i*], como visto pela camada superior representada pelos usuários *Usu*. Uma entidade *Abrac*[*i*] especifica um serviço orientado à conexão, cujos estados da mesma são especificados no tipo *CONNECTION_STATE*. Intuitivamente, o protocolo realiza a transmissão de uma sequência de mensagens de uma entidade *Usu* a uma entidade *Usu* remota, através das primitivas de serviço. O serviço ABRACADABRA é dado pela decrição de um *tipo livre*, denominado *A_PRIMITIVE*, no qual é permitido a parte axiomática especificar propriedades em lógica temporal, podendo-se assim especificar certas restrições do sistema. Isto estende a definição de um tipo livre de Z, com os predicados temporais definidos na seção 5.

O tipo de letra não tem nenhuma importância no que diz respeito à sintaxe da linguagem. Entretanto, por questão de legibilidade, uma boa regra, se possível, é seguir os tipos de letra usados nos identificadores, como mostra o exemplo.

8 Considerações Finais e Perspectivas

Apresentamos a linguagem Zag que estende o poder de expressão da linguagem Z com os aspectos de concorrência e comunicação baseados em CCS de Milner (CA) e introduz lógica formal em contraste às linguagens SDL, ESTELLE e LOTOS. Zag reúne a abordagem comportamental e algébrica de CA com a abordagem axiomática de Z e da Lógica Temporal.

Zag tem método formal baseado em modelo, mais precisamente o modelo de transições de estado proporcionado pela definição de um sistema de transições rotuladas apropriado à linguagem Zag, mas tem seu método formal também orientado para propriedades.

Tendo suporte de CA, Zag segue o modelo fundamental para sistemas distribuidos: aquele de uma coleção de entidades comunicantes (possivelmente autonomas), onde os conceitos de concorrência e comunicação são estabelecidos. Os conceitos básicos de modelagem envolvidos para descrever o domínio do problema, que estão presentes em Zag são *ações*, *agentes* e *comunicação* como em CA. Comunicação por "rendez-vous" é adotada, por conseguinte independente de todos os mecanismos de comunicação existentes.

Zag permite especificar num nível de abstração envolvendo ações, variáveis e dados com operações (tipos). O sistema de tipos de Z é totalmente explorado e uma ampla coleção de tipos de dados pode ser construida. Especificações de comportamento em Zag focalizam as ações possíveis de CA e permite as operações internas de um agente serem referenciadas. Essas operações descrevem os efeitos das execuções das ações de CA sobre os estados do agente.

Propriedades de sistemas distribuidos (ordenação de ações e de restrições, conceitos de segurança, vivacidade e justiça) e restrições sobre o comportamento de um sistema são descritas através da linguagem de lógica de primeira ordem de Z ou da lógica temporal.

Um agente em Zag é fundamentalmente construido de uma parte comportamental, e um ou mais tipos abstratos de dados *baseados em estado*, inerentes ao funcionamento interno de um agente. Tais tipos são especificados seguindo o estilo de uma especificação Z.

A noção de um tipo abstrato de dados *baseado em estado* difere da noção de um tipo abstrato de dados *aplicativo* onde as operações do sistema são descritas por funções, que são inerentemente sem estado. A semântica algébrica de tipos aplivativos conduz ao problema de especificar

operações parciais (somente operações totais podem ser expressas). Este problema não existe quando se utiliza a caracterização de tipo abstrato de dados baseado em estado que permite operações parciais e também não-determinismo como definido em [18]. Tipos abstratos de dados baseados em estado permitem ir de uma forma direta ao modelo orientado para objetos. Tais tipos são mais apropriados para implementação em uma linguagem imperativa, que os tipos abstratos aplicativos usados em SDL e LOTOS.

O alto nível de abstração de CA e Z, e o poder de estruturação da linguagem CA permitem reduzir a complexidade de projeto. Modularidade, instanciação de agentes, e composição são requisitos de TDF's presentes em Zag advindos da linguagem CA. Relações entre agentes são expressas usando os operadores de Zag, cujo conjunto de símbolos operadores foi herdado de CA.

Pretende-se que Zag seja inserida numa metodologia de desenvolvimento de sistemas. Uma especificação Zag é um refinamento de uma especificação CA, que começa num dado nível de abstração imediatamente após o último nível de abstração usando a linguagem CA.

A partir de uma especificação usando a linguagem CA, refinamentos de agentes CA para agentes Zag são realizados considerando-se o efeito das execuções das ações atômicas de comunicação e das ações internas (τ), sobre o estado (interno) dos agentes. Tais efeitos são descritos pelas operações de um tipo abstrato de dados baseado em estado que reflete o funcionamento interno do agente.

Na transformação de CA para Zag, determinados parâmetros nos agentes CA se transformam em variáveis de estado na especificação Zag. Uma descrição em Zag, portanto, "implementa" uma descrição CA por extensão, incluindo funcionalidades (as operações internas) à especificação CA. No presente momento em que este artigo é escrito está sendo estudado um processo de desenvolvimento de programas a partir de uma especificação Zag, tomando-se como referência uma base formal que explore a junção de descrições baseadas em estados (como em Z) e descrições baseadas em ações (como em CA) [10] [22].

Nós contemplamos um modelo de pré-implementação baseado na integração de uma linguagem paralela utilizando comandos guardados de Dijkstra [9] com a linguagem de especificação usada por C. Morgan em [13], acrescentada de expressões para descrever o sequenciamento de refinamentos de ações e operações, o não-determinismo e a recursividade proveniente de agentes CA. A linguagem do modelo de pré-implementação permitirá refinar agentes Zag, seguindo a estrutura de um modelo de transição de estados. A metodologia se utiliza de regras de transformação de Z para refinamento de dados e operações do tipo abstrato de dados de um agente Zag, e é baseada na preservação da semântica entre uma especificação Zag e uma pré-implementação. Uma implementação de uma pré-implementação pode então ser realizada utilizando-se uma linguagem de programação imperativa.

Referências

[1] CCITT Com X-R 17-E. *Rec Z100 Functional Specification and Description Language (SDL).* Technical Report, CCITT, 1992.

[2] ISO IS 8807. *LOTOS - a Formal Description Technique based on the temporal ordering of observational behaviour.* Technical Report, ISO, 1989.

[3] M. Averbuch. *Aplicação do Método de Tableau para Lógica Temporal 'a Verificação de Programas Concorrentes.* Master's thesis, Programa de Engenharia Elétrica - COPPE/UFRJ, 1994.

[4] Ferenc Belina, Dieter Hogrefe, and Amardeo Sarma. *SDL with applications from Protocol Specification.* Prentice-Hall, 1991.

[5] V. Carchiolo, Stefano A. DI, Alberto Faro, and Giuseppe Pappalardo. Eccs and lips: two languages for osi systems specification and verification. *ACM Transations on Programming Languages and Systems*, 11(2):284–329, 1989.

[6] R. Duke, I. Hayes, P. King, and G. Rose. Protocol specification and verification using z. In S. Aggarwall and K. Sabnani, editors, *IFIP - Protocol Specification, Testing and Verification VIII*, pages 33–46, Elsevier Science Publishers B. V.(North-Holland), 1988.

[7] R. Gotzhein. Specifying open distributed systems with z. In *Proceedings of VDM'90: VDM and Z - Formal Methods in Software Development*, pages 319–339, April 1990.

[8] I. J. Hayes, M. Mowbray, and G. A. Rose. Signalling system no. 7: the network layer. In *in Protocol Specification, Testing, and Verification IX , Proc.,*, 1989.

[9] M. Hennessy. *The Semantcs of Programming Languages.* Wiley, 1990.

[10] He Jifeng. Various simulations and refinements. In J. W. de Bakker, W. P. de Roever, and G. Rozemberg, editors, *Stepwise Refinement of Distributed System*, pages 340–360, Springer-Verlag, May-June 1989.

[11] Cliff Jones. *Systematic Software Development using VDM.* Prentice Hall International - Series in Computer Science - C. A. R. Hoare Editor, 1990.

[12] R. Milner. *Communication and Concurrency.* Prentice-Hall, first edition edition, 1989.

[13] Carroll Morgan. Programming from Specifications. Prentice Hall International Series in Computer Science, C. A. R. Hoare Series Editor, 1990.

[14] CCITT Rec Q.703. *Specification of Signalling Link No. 7 vol.VI fascicle VI.7, Geneva.* Technical Report, CCITT, 1985.

[15] B. G. Riso. *Uma Abordagem para o Design de Sistemas Distribuidos e Protocolos de Comunicação.* PhD thesis, Departamento de Engenharia Elétrica - UFPB - Campina Grande, Novembro 1991.

[16] ISO TC97 SC21-N1533. *Status and Applicability of Formal Description Techniques.* Technical Report, ISO, 1989.

[17] J. B. M. Sobral. *Integrando CCS e Z.* Technical Report, COPPE - Programa de Engenharia Elétrica - UFRJ, 1994.

[18] J. M. Spivey. *Understanding Z: a specification language and its formal semantcs.* Cambridge University Press, 1988.

[19] J. M. Spivey. *The Z Notation - A Reference Manual.* Academic Press, New York, 1989.

[20] International Organization Standadization. *ESTELLE: A Formal Description Technique Based on the Extended State Transition Model.* Technical Report IS 9074, ISO, 1989.

[21] P. Stocks, K. Raymond, D. Carrington, and A. Lister. Modelling open distributed systems in z. *Computer Communication*, 15(2):103–113, March 1992.

[22] J. C. P. Woodcock and C. Morgan. Refinement of state-based concurrent systems. In *Proceedings of VDM'90: VDM and Z - Formal Methods in Software Development*, pages 340–351, April 1990.