O Modelo Z

Eduardo Bráulio Wanderley Netto
IC-UNICAMP, CEFET-RN
ra007219@ic.unicamp.br

Nahri Balesdent Moreano
IC-UNICAMP, DCT-UFMS
ra007220@ic.unicamp.br

Maio 2001

Resumo

Este trabalho tem como objetivo a apresentação das principais características do método formal Z para especificação de sistemas de computação, incluindo as fases de especificação e validação. Mostramos um exemplo de modelagem de uma máquina de bebidas e finalmente apontamos um conjunto de ferramentas automatizadas que oferecem suporte ao especificador seja na simples apresentação do modelo ou até na prova de teoremas utilizados no processo de validação. O Z se mostrou bastante versátil mas apresenta algumas restrições de uso, especialmente na modelagem de sistemas temporizados.

Abstract

This paper introduces the main features of the Z formal method for specification of computer systems, including the specification and validation stages. We show the modeling of a drink machine as an example and finally we present a set of automated tools which offer support to the specifier including simple viewers of the model, as well as, theorems proof systems which are used in the validation phase. The Z model is versatile but has some constraints of usage, specially in time-dependent systems modeling.

1. Introdução

O alto custo de desenvolvimento e manutenção do *software* tem levado a uma crescente preocupação com a qualidade do *software* e com a qualidade do processo de seu desenvolvimento. Desejamos averiguar a qualidade de um sistema em todas as etapas do processo de desenvolvimento, e encontrar erros o mais cedo possível neste processo.

Um dos primeiros estágios no desenvolvimento do *software*, a engenharia de requisitos, tem papel fundamental no processo de desenvolvimento. Neste estágio elaboramos a especificação dos requisitos do sistema, que servirá de documento base para as etapas seguintes do desenvolvimento.

Erros na especificação de requisitos trazem em conseqüência erros nas etapas seguintes do desenvolvimento. Portanto, a validação da especificação é de vital importância para a qualidade do *software*. Ou seja, é necessário analisar se a especificação dos requisitos está completa, precisa, sem ambigüidades, etc.

Diversos modelos e notações vêm sendo propostos e utilizados para a especificação de requisitos de um sistema, como por exemplo, diagramas gráficos e textos em linguagem natural. Modelos formais, baseados principalmente em conceitos matemáticos, também foram propostos como notações para a especificação de requisitos de sistemas. A utilização destes modelos produz, em geral, especificações precisas e sem ambigüidades (principalmente se comparados à linguagem natural). Além disso, permitem a utilização de ferramentas automatizadas para a validação da especificação.

O objetivo deste trabalho é a apresentação do modelo formal Z para especificação de sistemas de computação, incluindo as fases de especificação e validação, e a avaliação de um conjunto de ferramentas que dão suporte ao modelo.

Este texto está organizado em cinco seções. A segunda seção apresenta o modelo Z, descreve os principais conceitos matemáticos nos quais o modelo se baseia, oriundos da teoria de conjuntos e da lógica de predicados, e mostra como especificar os requisitos de um sistema usando o modelo.

A quarta seção trata da validação de especificações escritas em Z. Várias checagens que devem ser feitas na especificação, com o objetivo de averiguar sua consistência, são descritas.

A seção 5 apresenta algumas ferramentas disponíveis para se trabalhar com especificações em Z. As facilidades oferecidas por cada ferramenta são descritas e uma análise comparativa entre elas é feita. A ferramenta Z/EVES é apresentada com maiores detalhes, em função da sua utilização para a validação da especificação de um sistema de uma máquina de bebidas, usado como exemplo.

A última seção apresenta as conclusões do trabalho, onde fazemos uma avaliação do modelo Z, da forma de validação de especificações no modelo, e das ferramentas disponíveis.

Um apêndice apresenta a especificação de um sistema em particular, o controle de uma máquina de bebidas, usando o modelo Z. O objetivo é exemplificar de forma clara a utilização do modelo.

2. O Modelo Formal Z

O modelo formal Z foi criado na Universidade de Oxford/UK [9] no fim da década de 70, com o objetivo de servir como notação para a especificação formal de sistemas. O nome do modelo é uma homenagem ao matemático alemão Ernst Zermelo. Seu formalismo matemático é baseado na teoria de conjuntos e na lógica de predicados de primeira ordem. Conceitos da teoria de conjuntos são utilizados para representar as informações do sistema sendo especificado e operações entre elas. A lógica de predicados é usada para representar o comportamento ou propriedades do sistema. Justamente por ter este forte embasamento matemático, o modelo Z é classificado como um modelo formal.

Além de oferecer uma linguagem de especificação matematicamente precisa, isto é, com sintaxe e semântica bem definidas, o modelo oferece um mecanismo de estruturação poderoso, denominado esquema. Através do uso de esquemas, a especificação de um sistema pode ser dividida em módulos, permitindo o encapsulamento e o reuso dos mesmos.

Na construção de uma especificação de requisitos de um sistema utilizando o modelo Z, podemos descrever as variáveis que representam o estado, mudanças de estado, suas pré- e pós-condições e propriedades do sistema (por exemplo, invariante de estado).

Um importante aspecto do modelo Z diz respeito ao processo de refinamentos. Nós podemos construir uma especificação abstrata de um sistema, usando os objetos matemáticos e modelando suas propriedades. É possível então construir uma segunda especificação mais detalhada, oriunda da primeira, com a definição de objetos mais próximos da estrutura dos dados do sistema, mas ainda respeitando as propriedades modeladas na especificação abstrata.

Por ser um modelo formal, existem ferramentas (*softwares*) que auxiliam na validação de uma especificação em Z. Esta validação em geral consiste da checagem sintática e semântica e da prova de determinados teoremas.

Segundo [4], o modelo Z já foi utilizado para especificação formal de diversos sistemas reais. Um exemplo de sua aplicação é o sistema CICS/ESA v.3 (*Customer Information Control System*), um sistema de processamento de transações que provê acessos a dados e serviços de comunicação, integridade e segurança, desenvolvido pela IBM/UK em 1989.

2.1. Lógica de Predicados

Nesta seção apresentamos de maneira sucinta os principais conceitos da lógica de predicados que são usados no modelo Z para representar o comportamento do sistema sendo especificado. Não cobriremos todos os aspectos do modelo, mas apenas aqueles essenciais para o entendimento da especificação da máquina de bebidas.

Um predicado expressa propriedades ou restrições sobre objetos. A partir destes predicados, proposições podem ser construídas e avaliadas como verdadeiras ou falsas. Em um predicado, os objetos podem ser comparados através de operadores relacionais ($=, \neq, >, <, \geq, \leq$), como por exemplo, no predicado abaixo. Como não sabemos o valor de x, não é possível ainda estabelecer se o predicado é verdadeiro ou falso.

$$x > 5$$

É possível construir predicados a partir da combinação de outros mais simples, utilizando-se conectivos lógicos de negação ($\neg$), conjunção ($\wedge$), disjunção ($\vee$), implicação ($\Rightarrow$) e equivalência ($\Leftrightarrow$), como no exemplo a seguir.

$$(y \geq 0 \wedge x > y) \Rightarrow x > 0$$

Quantificadores podem ser usados em predicados para expressar propriedades universais ($\forall$) ou existenciais ($\exists$) de objetos. No primeiro predicado abaixo a variável x é declarada como sendo um número natural ($\mathbb{N}$) e a seguinte propriedade é representada: "existe um número natural x, tal que x é maior que 5". Este predicado é uma proposição verdadeira, enquanto que o predicado seguinte é uma proposição falsa, pois nem todo número natural é maior que 5.

$$\exists x: \mathbb{N} \bullet x > 5$$
$$\forall x: \mathbb{N} \bullet x > 5$$

Proposições são representadas no modelo Z usando o seguinte formato geral,

$$Q\ x: a \mid p \bullet q$$

onde:

- Q é um quantificador ($\exists$ ou $\forall$);
- x é uma variável que representa um objeto;
- a é o tipo da variável x e define o conjunto de valores válidos para o objeto;
- p é um predicado que especifica uma restrição sobre x, isto é, restringe o conjunto de valores de x a serem considerados. Apenas aqueles valores em a que satisfazem p são considerados;
- q é um predicado que expressa uma propriedade da variável.

Pode haver mais de uma variável em uma proposição, cada qual com o seu quantificador, embora não necessariamente haverá alguma restrição na proposição.

A proposição seguinte tem o seguinte significado: "para todo x em a que satisfaz p, q é verdadeiro". Já a segunda proposição significa: "existe um x em a que satisfaz p, tal que q é verdadeiro".

$$\forall x: a \mid p \bullet q$$
$$\exists x: a \mid p \bullet q$$

2.2. Teoria de Conjuntos

O modelo Z usa a teoria de conjuntos para representar objetos que modelam as informações do sistema sendo especificado e operações entre elas. Um conjunto é uma coleção bem definida de objetos de um mesmo tipo. Existem operadores que relacionam um objeto e um conjunto ($\in$, $\notin$), que relacionam dois conjuntos ($\subset$, $\supset$, $\subseteq$, $\supseteq$) e que operam dois conjuntos, obtendo um terceiro resultante ($\cup$, $\cap$, $\setminus$).

Por exemplo, dados os conjuntos a e b definidos abaixo, podemos construir proposições verdadeiras como as mostradas em seguida.

$$a == \{1, 2, 3, 4\}$$
$$b == \{2, 4\}$$
$$\forall x: \mathbb{N} \bullet x \in b \Rightarrow x \in a$$
$$\exists x: \mathbb{N} \mid x \in a \bullet x \notin b$$
$$a \cap b \neq \varnothing$$
$$b \subset a$$
$$a \setminus b = \{1, 3\}$$

É possível ainda associar objetos de diferentes conjuntos. Para isso, usamos o produto cartesiano que é um conjunto de tuplas, onde cada tupla é uma associação entre objetos, um de cada conjunto. O produto cartesiano dos conjuntos a e b é representado por $a \times b$.

$$a \times b = \{(1, 2), (1, 4), (2, 2), (2, 4), (3, 2), (3, 4), (4, 2), (4, 4)\}$$

Podemos também criar conjuntos de conjuntos, usando o conceito de conjunto potência ($\mathbb{P}$). Por exemplo, dados os conjuntos a e b definidos anteriormente, as seguintes proposições são verdadeiras:

$$\mathbb{P}\, b = \{\varnothing, \{2\}, \{4\}, \{2, 4\}\}$$
$$\{2\} \in \mathbb{P}\, b$$
$$\{(1, 2), (1, 4)\} \in \mathbb{P}\, a \times b$$

2.3. Definição de Objetos

Para representar as informações do sistema sendo especificado, podemos definir tipos e declarar variáveis no modelo Z. Um tipo é um conjunto maximal de valores possíveis para variáveis daquele tipo. Uma variável é declarada como sendo de um determinado tipo T e representa um objeto que possui um dos valores possíveis definidos por T.

O modelo Z oferece dois tipos básicos: o conjunto de todos os números inteiros ($\mathbb{Z}$) e o conjunto de todos os números naturais ($\mathbb{N}$). Novos tipos podem ser definidos de várias maneiras. Por exemplo, na construção abaixo, definimos $Bebidas$ como um tipo básico.

$$[Bebidas]$$

Podemos definir um tipo usando outro já existente e declarando que os dois são o mesmo tipo. Para isso, usamos a definição por abreviação. Por exemplo, na abreviação abaixo definimos $Valores$ como sendo o mesmo tipo que $\mathbb{N}$.

$$Valores == \mathbb{N}$$

Na definição abaixo, declaramos um tipo *Mensagens* e seus quatro valores válidos. Esta construção é um tipo "livre", isto é, um conjunto com informações explícitas sobre sua formação.

$$Mensagens ::= VendaEfetuada \mid BebidaNaoDisponivel \mid DinheiroInsuficiente \mid TrocoNaoDisponivel$$

Uma variável é declarada como sendo de um determinado tipo. Por exemplo, temos abaixo várias declarações de variáveis. A primeira declaração cria uma nova variável b do tipo $Bebidas$ definido anteriormente. A segunda declaração cria uma variável $BebidasDisponiveis$ cujo tipo é o conjunto potência de $Bebidas$. Isto é, a variável pode assumir um valor que é um subconjunto de $Bebidas$. Na terceira declaração criamos uma variável do tipo $Valores$ definido anteriormente.

$$b : Bebidas$$
$$BebidasDisponiveis : \mathbb{P} Bebidas$$
$$DinheiroDisponivel : Valores$$

2.4. Relações e Funções

No modelo Z as associações entre objetos podem ser representadas usando relações matemáticas. Uma relação (binária) estabelece ligação entre pares de objetos, isto é, uma relação entre dois conjuntos a e b ($a \leftrightarrow b$) é um conjunto de pares ordenados, onde o primeiro elemento de cada par pertence a a e o segundo a b. Ou seja uma relação é um subconjunto do produto cartesiano dos dois conjuntos.

O operador *dom*, quando aplicado a uma relação representa o domínio da relação. Isto é, $dom\ a \leftrightarrow b$ representa o conjunto de elementos de a que estão relacionados a algum elemento de b.

Uma função é um tipo especial de relação, onde cada objeto de um conjunto está relacionado a no máximo um objeto do outro conjunto. A função de um conjunto a para um conjunto b é denominada parcial (representada por $a \pfun b$) se podem existir elementos de a que não estão relacionados a nenhum elemento de b. Caso contrário, a função é total (representada por $a \fun b$).

No exemplo abaixo, declaramos a variável $PrecoBebida$ como uma função parcial entre os conjuntos representados pelos tipos $Bebidas$ e $Valores$. No predicado em seguida, a variável $PrecoBebida$ é utilizada com uma notação similar à notação de funções

matemáticas.

$$PrecoBebida: Bebidas \pfun Valores$$
$$\forall Beb: Bebidas \mid Beb \in dom\ PrecoBebida \bullet PrecoBebida(Beb) \geq 0$$

2.5. Esquemas

As construções matemáticas do modelo Z vistas até aqui descrevem objetos e suas propriedades em uma especificação. Esquemas podem ser usados para estruturar e combinar estas descrições, agrupando partes da especificação, encapsulando-as e nomeando-as para reuso.

Um esquema é um agrupamento de objetos matemáticos e suas propriedades e consiste de duas partes:

- parte declarativa: contém a declaração de variáveis;
- parte assertiva: contém predicados estabelecendo restrições sobre as variáveis.

Um esquema possui um nome, através do qual ele pode ser referenciado em outros pontos da especificação, e é descrito da forma abaixo:

Nome

Parte declarativa

Parte assertiva

a) Esquema Declarativo

Na especificação de um sistema usando o modelo Z, descrevemos as variáveis que representam o estado do sistema com um esquema declarativo. Este esquema possui na parte declarativa as variáveis de estado, e na parte assertiva, predicados que representam a invariante de estado do sistema.

Por exemplo, mostramos abaixo o esquema declarativo da máquina de bebidas.

MaquinaBebidas

$BebidasDisponiveis: \mathbb{P} Bebidas$
$DinheiroDisponivel: Valores$
$PrecoBebida: Bebidas \pfun Valores$
$NBebidasVendidas: Bebidas \pfun \mathbb{N}$

$DinheiroDisponivel \geq 0$
$\forall Beb: Bebidas \mid Beb \in dom\ PrecoBebida \bullet PrecoBebida(Beb) \geq 0$
$\forall Beb: Bebidas \mid Beb \in dom\ PrecoBebida \land Beb \in BebidasDisponiveis \bullet$
$PrecoBebida(Beb) > 0$

b) Variáveis de Entrada e Saída

Alguns esquemas podem precisar de informações de entrada e/ou produzir informações como saída. Estas informações são especificadas como variáveis de entrada e saída, respectivamente. Variáveis de entrada e saída são declaradas da mesma forma que as demais variáveis, porém possuem um símbolo (*decoration*) ? ou !, respectivamente, no final de seu nome.

Por exemplo, o esquema a seguir possui uma variável de entrada $BebidaEscolhida?$.

BebidaOK

$\Xi MaquinaBebidas$
$BebidaEscolhida?: Bebidas$

$BebidaEscolhida? \in BebidasDisponiveis$

c) Esquemas Representando Operações

Na especificação de um sistema usando o modelo Z, descrevemos as mudanças de estado através de esquemas que representam operações sobre as variáveis de estado do sistema. Estes esquemas possuem na sua parte declarativa uma referência ao esquema declarativo (que define as variáveis de estado e a invariante de estado), além de, possivelmente, outras variáveis. Na parte assertiva, estes esquemas possuem predicados que representam pré- e pós-condições das operações que eles especificam.

Em um esquema que representa uma operação, nós referenciamos as variáveis de estado através do nome do esquema declarativo, e podemos representar os estados do sistema antes e após a operação. Para diferenciá-los, o estado do sistema após a operação possui o símbolo ′ (outro *decoration*) no final de seu nome.

Se um esquema representa uma operação que pode causar uma mudança no estado do sistema, este esquema deve possuir em sua parte declarativa referências aos estados anterior e posterior do sistema. Para isso, utilizamos o símbolo Δ antes do nome do esquema declarativo.

Se um esquema representa uma operação que nunca causa mudança no estado do sistema, utilizamos o símbolo Ξ antes do nome do esquema declarativo.

Por exemplo, o primeiro esquema abaixo representa uma operação que não altera o estado do sistema, enquanto que o segundo representa uma operação que causa uma mudança no mesmo.

DinheiroOK

Ξ*MaquinaBebidas*
BebidaEscolhida?: Bebidas
DinheiroFornecido?: Valores

BebidaEscolhida? $\in$ *dom PrecoBebida*
DinheiroFornecido? $\geq$ *PrecoBebida(BebidaEscolhida?)*

DinheiroDisponivelOK

Δ*MaquinaBebidas*
BebidaEscolhida?: Bebidas

BebidaEscolhida? $\in$ dom *PrecoBebida*
DinheiroDisponivel' = DinheiroDisponivel + PrecoBebida(BebidaEscolhida?)

d) Esquemas em Predicados

Como mencionado anteriormente, esquemas são utilizados para modularizar a especificação e permitir a utilização de uma mesma construção em vários pontos da especificação. Assim, um esquema E_1 pode ser usado como um predicado dentro de um outro esquema E_2. O efeito é introduzir em E_2 uma restrição equivalente à parte assertiva de E_1.

Por exemplo no esquema *VendaOK* a seguir, vários outros esquemas são referenciados na parte assertiva. Estes esquemas estão atuando como predicados dentro do esquema que os engloba, isto é, são restrições impostas às variáveis.

VendaOK

Δ*MaquinaBebidas*
BebidaEscolhida?: Bebidas
DinheiroFornecido?: Valores
Resultado!: Mensagens

BebidaOK
DinheiroOK
TrocoOK
DinheiroDisponivelOK
Historico
Resultado != VendaEfetuada

Uma outra forma de definir esquemas é mostrada abaixo, na construção do esquema $Venda$.

$$Venda \mathrel{\widehat=} VendaOK \lor VendaNaoOK$$

2.6. Outros Conceitos

O modelo Z oferece uma vasta gama de construções, além das apresentadas aqui. Por exemplo, é possível a especificação de seqüências, isto é, conjuntos ordenados de objetos. Existem de operadores para este tipo de dado, tais como concatenação, seleção de elementos, referência aos primeiro e último elementos, etc.

Uma outra construção oferecida é a *bag*, um conjunto não ordenado de objetos, em que pode haver repetições de elementos. Novamente, diversos operadores permitem a manipulação deste tipo, como por exemplo, união e diferença de *bags* e a contagem do número de repetições é um determinado elemento dentro do conjunto. Essas e outras construções estão descritas em detalhes em [3, 4].

3. Validação de uma Especificação

A especificação de um sistema usando um modelo formal não garante que ela está correta ou completa. Podem existir vários tipos de erros na especificação. Porém, especificações formais podem ser analisadas e checadas em relação a algumas propriedades. Mais ainda, estas checagens podem ser feitas com o auxílio de ferramentas automatizadas.

Assim, uma vez desenvolvida a especificação, ela deve ser validada, isto é, deve-se analisar se a especificação apresenta todos os requerimentos do sistema sendo modelado. A checagem da consistência de uma especificação em Z é baseada em técnicas de análise estática. A validação consiste em checar a correção sintática e semântica (de tipos) da especificação, garantir que as operações respeitam o modelo de estados do sistema, checar se a especificação contém as funcionalidades e propriedades desejadas e analisar se há informações inconsistentes ou faltosas na mesma [1].

3.1. Checagem Sintática e Semântica

Dado que o modelo Z possui uma linguagem com sintaxe e semântica precisas, é possível checar se a especificação inteira está sintática e semanticamente correta. A checagem sintática verifica se todas as construções estão no “formato” correto, enquanto que a checagem semântica verifica se os dados operados e os operadores são de tipos compatíveis.

No predicado abaixo por exemplo, há um erro sintático, dado que está faltando um operador relacionando a variável Beb com o domínio de $PrecoBebida$. O predicado corrigido é mostrado em seguida.

$$\forall Beb : Bebidas \mid Beb\ dom\ PrecoBebida \bullet PrecoBebida(Beb) = 0$$
$$\forall Beb : Bebidas \mid Beb \in dom\ PrecoBebida \bullet PrecoBebida(Beb) = 0$$

O predicado abaixo está sintaticamente correto, mas contém um erro de semântica. A variável $DinheiroDisponivel$ foi declarada como do tipo $Valores$ (que corresponde ao tipo $\mathbb{N}$) e está sendo comparada a um conjunto. O predicado corrigido é mostrado em seguida.

$$DinheiroDisponivel = \varnothing$$
$$DinheiroDisponivel = 0$$

As checagens sintática e semântica são feitas automaticamente por ferramentas que dão suporte ao modelo Z, entre elas o Z/EVES, utilizada para validar a especificação da máquina de bebidas.

3.2. Checagem de Domínio

Uma construção escrita em Z pode estar sintática e semanticamente correta, mas não ter seu significado completamente definido. Por exemplo, quando uma função ou um operador são aplicados a valores fora de seu domínio, há um erro de domínio na especificação. Isto é, não há valores definidos para o resultado da função ou do operador.

O predicado abaixo possui um erro de domínio, dado que o operador *div* (divisão inteira) não tem valor definido quando o segundo operando é 0. O predicado corrigido é mostrado em seguida.

$$\forall x, y: \mathbb{N} \bullet x > 0 \Rightarrow x \; div \; y > 0$$
$$\forall x, y: \mathbb{N} \mid y > 0 \bullet x > 0 \Rightarrow x \; div \; y > 0$$

No esquema abaixo também há um erro de domínio, pois não é garantido que a variável $BebidaEscolhida$ pertence ao domínio da função $PrecoBebida$. Este erro pode ser corrigido acrescentando-se um predicado que estabeleça esta restrição, como mostrado em seguida.

DinheiroOK

$\Xi MaquinaBebidas$
BebidaEscolhida?: Bebidas
DinheiroFornecido?: Valores

$DinheiroFornecido? \geq PrecoBebida(BebidaEscolhida?)$

DinheiroOK

. . .

$BebidaEscolhida? \in \text{dom } PrecoBebida$
$DinheiroFornecido? \geq PrecoBebida(BebidaEscolhida?)$

O Z-EVES, entre outras ferramentas, também realiza a checagem de domínio, gerando automaticamente um teorema apropriado e tentando prová-lo.

3.3. Checagem de Consistência

A consistência da especificação é estabelecida em duas etapas. Inicialmente, temos a validação do estado inicial. É necessário mostrar que existe um estado inicial válido que satisfaz a invariante de estado do sistema. Para isso, cabe ao especificador escrever um teorema apropriado. Uma ferramenta que possua um provador de teoremas (por exemplo o Z/EVES) pode ser utilizada para fazer a prova do mesmo.

Por exemplo, para a máquina de bebidas, o teorema a seguir precisa ser provado. $MaquinaBebidas'$ é uma referência ao esquema declarativo e $MaquinaBebidasInit$ é uma referência ao esquema de iniciação das variáveis de estado (que descreve propriedades do estado inicial do sistema). O teorema estabelece que deve haver um estado que atenda à invariante de estado do sistema e à operação de iniciação do mesmo.

$$\exists\, MaquinaBebidas' \bullet\ MaquinaBebidasInit$$

Em uma segunda etapa temos a validação de cada operação. Isto é, precisamos provar que cada operação respeita a invariante de estado do sistema. Mais uma vez, o especificador precisa escrever um teorema apropriado e utilizar uma ferramenta que possua um provador de teoremas para fazer a prova do mesmo.

Por exemplo, para a máquina de bebidas, o teorema a seguir precisa ser provado. $MaquinaBebidas$ é uma referência ao esquema declarativo e $DinheiroDisponivelOK$ é uma referência a um esquema representa uma operação. O teorema estabelece que, se o estado anterior à operação atende à invariante de estado do sistema e a operação pode ser aplicada, então o estado após a operação deve também atender à invariante..

$$MaquinaBebidas \wedge DinheiroDisponivelOK \Rightarrow MaquinaBebidas'$$

Um teorema similar a este deve ser construído e provado para cada esquema que causa uma mudança de estado no sistema.

3.4. Validação de Refinamentos

O modelo Z pode ser utilizado para construir uma especificação mais refinada (denominada concreta), a partir de uma especificação mais alto nível (denominada abstrata) desenvolvida no próprio modelo.

Podemos validar esse refinamento, mostrando que as duas especificações são equivalentes. Isto equivale a mostrar que:

- existe uma relação entre os estados iniciais abstrato e concreto;
- se uma operação abstrata pode ser aplicada, a operação concreta correspondente também pode (e vice-versa);
- uma operação abstrata produz um resultado consistente com a operação concreta correspondente (e vice-versa).

Para isso, cabe ao especificador escrever os teoremas que garantam as propriedades acima e utilizar uma ferramenta para prová-los.

4. Ferramentas

Uma notação precisa e fundamentada na matemática possibilita a criação de ferramentas automatizadas de auxílio à criação e validação de documentos que sigam esta notação. Com a popularidade crescente do Z muitas destas ferramentas foram desenvolvidas para a notação enriquecendo a gama de opções para aqueles que utilizam este modelo. Diversas dessas ferramentas estão catalogadas no *site* [9], disponiblilizado pela Universidade de Oxford

De uma forma genérica, ferramentas para checagens sintática, semântica (de tipos), de domínio e provadores de teoremas estão disponíveis, assim como simples visualizadores da notação. O formato do documento de especificação aceito por estas ferramentas é baseado em LaTeX *markup*, uma extensão do LaTeX. Não é preciso, entretanto, compilar o código. Desde que suas marcações estejam corretas, as ferramentas conseguem interpretar o arquivo ASCII contendo a especificação.

4.1. A Ferramenta Z/EVES

O Z/EVES é uma dessas ferramentas, versátil o suficiente para aglutinar todas as características supracitadas. Disponibilizada gratuitamente pela ORA Canadá [7], sua versão 2.1 apresenta dois modos de operação (linha de comando – prompt – e interface gráfica – GUI). A portabilidade é outro fator interessante nesta ferramenta, que pode ser utilizada nas plataformas Solaris, Linux e MSWindows. Seu modo de operação GUI, descrito em nosso trabalho, está baseado em Python [8] o que representa um requisito a mais para quem deseja utilizá-la.

Outra característica marcante do Z/EVES-GUI [2] é sua capacidade de guardar contextos de checagens. Isto nos possibilita interromper um trabalho de validação de uma especificação e retomá-lo, do exato local onde paramos. Também é permitido no Z/EVES-GUI a edição e criação de parágrafos, diretamente na ferramenta, sem ser necessário fazer uso do LaTeX *markup*, embora os parágrafos editados desta maneira (*on-the-fly)* já não possam ser salvos no formato de marcadores LaTeX. Arquivos *Rich-Text-Format (rtf)* e *PostScript (ps)* podem ser gerados contendo a notação usual do Z para futuras impressões.

O processo de instalação do Z/EVES no MSWindows é simples, embora no Linux possa exigir um pouco mais de habilidade com o sistema operacional

4.2. Outras Ferramentas

O CADiℤ foi desenvolvido pela Universidade de York [6] e também contempla todas as características gerais de uma ferramenta para Z. Adicionalmente um sistema de navegação hypertexto entre os parágrafos foi disponibilizado. As mensagens de erro de sintaxe e semântica são amigáveis, apontando exatamente onde se encontram os problemas e também permitem navegação. O CADiℤ 3.13 está disponível para Linux, IRIX e Solaris, mas o seu processo de instalação exige um certo grau de experiência do

usuário. Seu arquivo de entrada padrão tem extensão .z embora seja exatamente igual ao .tex do Z/EVES. A saída padrão pode ser feita em LaTeX e *troff*.

O ZBrowser [7] é uma ferramenta apenas de visualização, mas apresenta características didaticamente interessantes como coloração da especificação de acordo com a sintaxe, navegação entre parágrafos e um sistema de ajuda em dois níveis (*hints* e *help*) acionados a partir dos parágrafos. Um simples passar do mouse sobre um operador válido em Z aciona o *hint* (pequena dica de uso) e se clicarmos sobre o mesmo o *help* é ativado com uma descrição completa e exemplo de uso. Uma outra característica importante é que o ZBrowser suporta comandos DDE (um sistema de troca de mensagens entre aplicações do MSWindows), permitindo integração com ferramentas que não possuem visualizador gráfico, como o Z/EVES–prompt.

A ferramenta Fuzz é apenas um checador de sintaxe e semântica mas chama a atenção por suas mensagens que apresentam exatamente onde se encontra o erro, inclusive sugerindo como corrigi-lo. Infelizmente, não possui interface gráfica. Uma restrição importante é que não é permitida a checagem dos parágrafos fora de ordem. Ou seja, para checar um parágrafo precisamos checar todos os anteriores.

Finalmente, apresentamos uma ferramenta que ajuda desenvolvedores de extensões para Z, principalmente se desejam conhecer como se escreve um analisador sintático para o modelo. O ZAST [7] mostra apenas a árvore de sintaxe de uma especificação, não recaindo no modelo comum de ferramentas de auxílio a especificadores.

O quadro a seguir contém um resumo das principais características das ferramentas supracitadas, mostrando um comparativo entre elas e destacando suas melhores características do ponto de vista do usuário do *software*.

	Z/EVES	Fuzz	CADiZ	ZBrowser
Checagem fora de ordem	Sim	Não	Sim	-
Mensagens de erro	★★★	★★★★★	★★★★	-
Interface gráfica	★★★★	-	★★★★	★★★★★
Navegabilidade	Não	Não	Sim	Sim
Edição *on-the-fly*	Sim	Não	Não	Não
Manual do software	★★★★	★★★	★★★★★	-
Plataformas (S.O.)	MSWindows Linux Solaris	Linux SunOS	Linux IRIX Solaris	MSWindows
Instalação	Médio	Médio	Difícil	Fácil
Entrada/Saída	tex, zev/ zev, rtf, ps	tex/ tex, *troff*	tex/-	tex/-
Salva contexto	Sim	Não	Não	Não
Tipos de checagens	Sintaxe Semântica Domínio Teoremas	Sintaxe Semântica	Sintaxe Semântica Domínio Teoremas	-
Outros	Assistência ao Usuário	-	-	Porta DDE

Tabela 1: Quadro comparativo das ferramentas utilizadas (baseado em [5]). A pontuação em estrelas é meramente ilustrativa sendo o valor máximo 5 estrelas.

5. Conclusões

Neste artigo apresentamos o modelo formal Z para especificação de sistemas de computação. Mostramos que existem diversas ferramentas de auxílio à criação e validação da especificação mas ainda há muito trabalho intelectual, inclusive na criação dos teoremas, onde, embora seguindo diretrizes, o especificador deve interagir com a ferramenta. Muito do trabalho de especificação depende do grau de abstração requerido, mas o próprio modelo Z oferece suporte para refinamentos sucessivos.

A notação utilizada é precisa mas não é de fácil entendimento sem a fundamentação matemática envolvida. Um bom processo de adequação ao modelo pode ser demorado dependendo do grau de formação matemática da equipe de especificadores e também dos desenvolvedores que serão usuários do modelo validado. Uma boa ferramenta pode ajudar a diminuir esta dificuldade inicial.

O modelo Z mostrou-se versátil na modelagem de sistemas contando com um conjunto sofisticado de construções capaz de representar boa parte dos requisitos de sistemas. Salientamos, entretanto que comportamentos concorrentes e temporização de ações não são tratados pelo modelo, bem como requisitos não funcionais (facilidade de uso, desempenho, etc).

Ainda existe muito a ser feito tratando-se da utilização do modelo Z. Entre as ferramentas, destacamos o Z/EVES por ter uma boa portabilidade e facilidade de operação, mas entendemos que sua interface gráfica pode ser consideravelmente melhorada, aproximando-se do ZBrowser, e suas mensagens de erro podem ser mais amigáveis como no Fuzz. Em relação ao modelo, seria apreciável poder contar com extensões do Z que abordassem comportamentos dependentes de tempo, o que estimamos representar um longo caminho a ser percorrido.

Um próximo passo a ser seguido aborda como seria possível a elaboração de testes a partir de uma especificação em Z.

6. Referências

[1] Alagar, V.S. & Periyasamy, K., *Specification of Software Systems*, Springer-Verlag, 1998.

[2] Saaltink, M., *The Z/EVES 2.0 User's Guide*, 1999. Disponível em http://www.ora.on.ca/, maio 2001.

[3] Spivey, J.M., *The Z Notation: A Reference Manual*, 2nd edition, 1998. Disponível em http://spivey.oriel.ox.ac.uk/~mike/zrm/, abril 2001.

[4] Woodcock, J. & Davies, J., *Using Z: Specification, Refinement, and Proof*. Disponível em http://softeng.comlab.ox.ac.uk/usingz/, abril 2001.

[5] Wu, B., *A Tool Survey on Z Specification and Refinement*, Technical Report, Department of Computing/University of Bradford, 1999. Disponível em http://www.personal.comp.brad.ac.uk/~bwu/research/toolsurvey.html , maio 2001.

[6] CadiZ web page, http://www.cs.york.ac.uk/~ian/cadiz/, abril 2001.

[7] ORA Canadá web page, http://www.ora.on.ca/, maio 2001.

[8] Pyton web page, http://www.python.org/, abril 2001.

[9] Z web page, http://www.comlab.ox.ac.uk/archive/z.html, abril 2001.

Apêndice: Especificação em Z da Máquina de Bebidas

O objetivo desta seção é apresentar um exemplo de modelagem de uma máquina automatizada de fornecimento de bebidas, capaz de interagir com clientes e operadores (técnicos e gerentes). As características desta máquina compõem o conjunto de requisitos da especificação: o cliente solicita uma bebida disponível para máquina e efetua o pagamento. A máquina, em contrapartida, entrega a bebida solicitada e eventualmente um troco. A propósito, condições de erros como dinheiro insuficiente devem ser tratadas. Outra fase de funcionamento da máquina refere-se ao processo de manutenção, onde técnico e/ou gerente irão prover os insumos operativos (bebidas e cédulas e moedas para trocos) ou recolher as quantias auferidas pelas vendas efetuadas até aquele momento, inclusive recebendo o histórico de transações.

Uma vez definido o problema apresentamos uma modelagem que compreende os requisitos supracitados.

a) Declarações de Tipos

% Tipos que compõem o escopo de declarações globais da especificação

$[Bebidas]$

$Valores == \mathbb{N}$

$Mensagens ::= VendaEfetuada \mid BebidaNaoDisponivel \mid DinheiroInsuficiente \mid TrocoNaoDisponivel$

b) Esquema Declarativo

% Esquema que define as variáveis de estado e a invariante de estado do sistema

MaquinaBebidas

$$
\begin{array}{l}
BebidasDisponiveis: \mathbb{P} Bebidas \\
DinheiroDisponivel: Valores \\
PrecoBebida: Bebidas \rightarrow Valores \\
NBebidasVendidas: Bebidas \rightarrow \mathbb{N} \\
\hline
DinheiroDisponivel \geq 0 \\
\forall Beb: Bebidas \mid Beb \in \text{dom } PrecoBebida \bullet PrecoBebida(Beb) \geq 0 \\
\forall Beb: Bebidas \mid Beb \in \text{dom } PrecoBebida \wedge Beb \in BebidasDisponiveis \bullet \\
\quad PrecoBebida(Beb) > 0
\end{array}
$$

c) Esquema de Iniciação

% Esquema que descreve propriedades do estado inicial do sistema

MaquinaBebidasInit

$MaquinaBebidas'$

$BebidasDisponiveis' = \varnothing$
$DinheiroDisponivel' = 0$
$PrecoBebida' = \varnothing$
$NBebidasVendidas' = \varnothing$

d) Esquemas Operativos

% Esquemas que representam operações para realização de uma atividade na máquina

% Checa se a bebida escolhida está disponível para venda

BebidaOK

$\Xi MaquinaBebidas$
$BebidaEscolhida?: Bebidas$

$BebidaEscolhida? \in BebidasDisponiveis$

% Checa se o dinheiro fornecido pelo cliente é suficiente para compra da bebida

DinheiroOK

$\Xi MaquinaBebidas$
$BebidaEscolhida?: Bebidas$
$DinheiroFornecido?: Valores$

$BebidaEscolhida? \in \text{dom}\ PrecoBebida$
$DinheiroFornecido? \geq PrecoBebida(BebidaEscolhida?)$

% Checa se a máquina tem condições de fornecer troco para compra desejada

TrocoOK

$\Xi MaquinaBebidas$
$BebidaEscolhida?: Bebidas$
$DinheiroFornecido?: Valores$

$BebidaEscolhida? \in \text{dom}\ PrecoBebida$
$\exists Troco: Valores \bullet (Troco = DinheiroFornecido? - PrecoBebida(BebidaEscolhida?)) \wedge$
$(Troco \leq DinheiroDisponivel)$

% Certifica que, após a compra, o valor disponível na máquina seja a soma do que
% havia anteriormente com o valor pago pelo cliente

DinheiroDisponivelOK

Δ*MaquinaBebidas*
BebidaEscolhida?: Bebidas

BebidaEscolhida? $\in$ dom *PrecoBebida*
DinheiroDisponivel' = DinheiroDisponivel + PrecoBebida(BebidaEscolhida?)

% Adiciona ao histórico de transações a venda efetuada

Historico

Δ*MaquinaBebidas*
BebidaEscolhida?: Bebidas

$(\exists$ *Quant:* $\mathbb{N}$ $\bullet$ *(BebidaEscolhida?, Quant)* $\in$ *NBebidasVendidas*
$\Rightarrow$ *(BebidaEscolhida?, Quant + 1)* $\in$ *NBebidasVendidas')* $\vee$
($\forall$ *Quant:* $\mathbb{N}$ $\bullet$ *(BebidaEscolhida?, Quant)* $\notin$ *NBebidasVendidas*
$\Rightarrow$ *NBebidasVendidas' = NBebidasVendidas* $\cup$ *{(BebidaEscolhida?, 1)})*

% Representa uma venda bem sucedida (requer que os esquemas operativos acima
% tenham sucesso)

VendaOK

Δ*MaquinaBebidas*
BebidaEscolhida?: Bebidas
DinheiroFornecido?: Valores
Resultado!: Mensagens

BebidaOK
DinheiroOK
TrocoOK
DinheiroDisponivelOK
Historico
Resultado != VendaEfetuada

% Representa uma tentativa de venda que não foi efetuada por um dos motivos que
% serão indicados em Resultado!

VendaNaoOK

$\Delta MaquinaBebidas$
BebidaEscolhida?: Bebidas
DinheiroFornecido?: Valores
Resultado!: Mensagens

$(\neg BebidaOK \land Resultado! = BebidaNaoDisponivel) \lor$
$(\neg DinheiroOK \land Resultado! = DinheiroInsuficiente) \lor$
$(\neg TrocoOK \land Resultado! = TrocoNaoDisponivel)$

% Esquema que indica que em um processo de venda poderemos ou não ter sucesso
$Venda \mathrel{\widehat{=}} VendaOK \lor VendaNaoOK$

% Indica a colocação de bebidas na máquina

ReposicaoBebida

$\Delta MaquinaBebidas$
BebidaReposta?: Bebidas

$BebidasDisponiveis' = BebidasDisponiveis \cup \{BebidaReposta?\}$

% Indica a colocação de quantias na máquina

ReposicaoDinheiro

$\Delta MaquinaBebidas$
ValorReposto?: Valores

$ValorReposto? > 0$
$DinheiroDisponivel' = DinheiroDisponivel + ValorReposto?$

% Representa a retirada de quantias da máquina

ColetaDinheiro

$\Delta MaquinaBebidas$
ValorColetado?: Valores

$ValorColetado? > 0$
$DinheiroDisponivel \geq ValorColetado?$
$DinheiroDisponivel' = DinheiroDisponivel - ValorColetado?$

% Representa a retirada de um histórico de vendas da máquina. O valor da
% variável NBebidasVendidas volta para sua condição inicial

ColetaHistorico

$MaquinaBebidas'$

$NBebidasVendidas' = \varnothing$

% Certifica que as bebidas disponíveis possuem um valor associado

SetaPreco

$\Delta MaquinaBebidas$
$BebidaS?: Bebidas$
$PrecoS?: Valores$

$BebidaS? \in \text{dom}\ PrecoBebida'$
$PrecoBebida'(BebidaS?) = PrecoS?$

% Representa um processo de Manutenção que pode realizar uma ou mais
% tarefas mostradas acima

$Manutenção \mathrel{\widehat=} ReposicaoBebida \lor ReposicaoDinheiro \lor ColetaDinheiro \lor ColetaHistorico \lor SetaPreco$

e) Exemplo de Refinamento

Apresentamos uma possibilidade de refinamento para o tipo $Valores$. Tipicamente, os valores envolvidos em uma transação são materializados em cédulas e moedas, com suas respectivas quantidades. Uma possível abordagem para representar esta definição é a construção de uma bag. O exemplo abaixo define a variável DinheiroDisponivel como uma bag contendo cédulas e moedas com suas respectivas quantias.

$Dinheiro ::= \{Moeda25, Moeda50, Moeda100, Cedula100\}$
$DinheiroDisponivel = \{Moeda25 \mapsto 15, Moeda50 \mapsto 28, Cedula100 \mapsto 4\}$