Utilizando o J-SDL para a geração automática de casos de testes a partir de especificações em SDL

Júlio Resende Ribeiro[1], Nandamudi L. Vijaykumar[1], Valdivino Alexandre de Santiago Júnior[2], Fabiana Fraga Ferreira[3], Alessandro Oliveira Arantes[4]

[1]Laboratório Associado de Computação e Matemática Aplicada (LAC) – Instituto Nacional de Pesquisas Espaciais (INPE) São José dos Campos, SP

[2]Ciências Espaciais Atmosféricas (CEA) – Instituto Nacional de Pesquisas Espaciais (INPE) São José dos Campos, SP

[3]Departamento de Ciência da Computação – Faculdade de Administração e Informática de Santa Rita do Sapucaí (FAI) Santa Rita do Sapucaí, MG

[4]Instituto de Estudos Avançados (IEAv), Comando Geral de Tecnologia Aeroespacial (CTA), São José dos Campos, SP

{julio@jrrsistemas.com.br, vijay@lac.inpe.br, valdivino@das.inpe.br, fabiana_fraga@yahoo.com.br, aarantes@ieav.cta.br}

Abstract. *Testing exposes defects before the software is launched in the market. Several techniques are available to automatically generate test sequences from formal specifications. The objective of the work described in this paper is to present the feasibility of automatically generating test sequences from systems specified formally in SDL. A framework, J-SDL, has been developed for this purpose that simulates the behavior of the specified system by stimulating events to change system states to generate test cases, based on algorithms such as T, UIO, DS, W and Switch Cover, in a straightforward manner without the necessity of generating a Finite State Machine.*

Resumo. *A finalidade dos testes de software é expor defeitos latentes em um sistema antes que o mesmo seja colocado em produção. Existem técnicas que permitem a geração automática de casos de teste a partir de especificações formais. O objetivo deste trabalho é apresentar a viabilidade de uma abordagem para geração automática de casos de teste a partir de especificações em SDL, utilizando um framework denominado J-SDL, que permite a simulação do comportamento de sistemas, em termos de mudança de estados e a derivação de casos de teste de forma direta (dispensando a transformação para maquina de estados finitos), a partir de algoritmos de exploração de estados como T, UIO, DS, W, Switch Cover, entre outros.*

1. Introdução

Softwares para computadores de bordo de satélites são reativos por natureza, pois respondem a estímulos ou eventos. Este tipo de *software* é também complexo devido a sua forte interação com o *hardware* do computador, sensores, atuadores e outros

dispositivos presentes em satélites. Além disso, é de difícil substituição em caso de falhas devido ao aspecto não pilotado da missão [Santiago et at. 2006]. Desta forma, o processo de verificação e validação assume papel importante durante todo o ciclo de vida desse tipo de *software*.

O processo de verificação e validação tem como objetivo assegurar que o *software* cumpra com suas especificações e atenda às necessidades para o qual está sendo desenvolvido. Este processo deve estar ativo durante todo o ciclo de vida do *software*, começando com as revisões de requisitos e continuando com as revisões de projeto e as inspeções de código até chegar aos testes [Sommerville 2003].

A finalidade dos testes de *software* é expor defeitos latentes em um sistema antes que o mesmo seja colocado em produção. A idéia básica da atividade de teste envolve a execução do *software* e a observação do seu comportamento e de suas saídas. Caso uma falha seja observada, o contexto da execução é armazenado e analisado para que seja possível encontrar e corrigir os defeitos que causaram a falha. Um teste é considerado bem sucedido se consegue expor um defeito do sistema, fazendo que o mesmo opere incorretamente. É importante destacar que os testes são capazes apenas de indicar a presença e nunca a ausência de defeitos em *software*.

Os testes funcionais (conhecidos como testes de caixa preta) consistem em uma abordagem na qual, os testes são derivados da especificação do *software*. A funcionalidade foco do teste é encarada como uma "caixa-preta" cujo comportamento só pode ser avaliado por meio da comparação das saídas obtidas em relação às saídas desejadas para entradas conhecidas [Sommerville 2003].

Testes estruturais (conhecidos como testes de caixa branca) são derivados do conhecimento da estrutura e da implementação do programa. De maneira geral, esse tipo de teste é aplicado à unidades de programa relativamente pequenas, como métodos ou funções. O testador nesse caso utiliza os conhecimentos sobre a estrutura da implementação para derivar os dados para o teste [Sommerville 2003].

É conhecido o fato de que a realização exaustiva de testes de *software* em sistemas complexos, como os de aplicação espacial, é impraticável, pois as atividades de testes consumiriam muito tempo, devido à complexidade desse tipo de *software* [Santiago et al 2006]. Dessa forma, torna-se interessante a utilização de técnicas que permitam a geração automática de casos de teste a partir de especificações formais que possam ser manipuladas por computador, segundo algum critério de cobertura que permita a redução da quantidade de casos de testes gerados a um número que seja possível de ser executado, tentando manter o mais elevada possível a possibilidade de encontrar erros através do conjunto de casos de testes gerados.

O uso de máquinas de estados finitos como técnica formal de descrição, para esse fim é bem popular, devido ao fato dessa técnica lidar naturalmente com a representação de sistemas reativos, que consistem de estados e transições entre estados através de estímulos também conhecidos como eventos.

Embora, máquinas de estados finitos sejam usadas tradicionalmente para representação de sistemas reativos, falta nessa técnica, facilidades para representar características comuns dos sistemas complexos modernos, tais como hierarquia e paralelismo. Por esse motivo torna-se interessante o uso de técnicas de mais alto nível

que ofereçam essas facilidades e possuam também o formalismo necessário para possibilitar a manipulação de especificações de sistemas reativos pelo computador, com o objetivo de geração de casos de teste. Entre essas técnicas de mais alto nível, podemos citar: *Statecharts* [Amaral 2005], [Santiago et at. 2006], Redes de Petri [Desel et at. 2007], *SDL* [Wong 2003], entre outras.

Para gerar automaticamente casos de teste a partir de uma especificação formal, faz-se necessária a utilização de algoritmos que utilizem critérios de cobertura para a exploração do espaço de estados. Entre os algoritmos mais utilizados para esta finalidade, podemos citar: T, UIO, DS, W, Switch Cover, entre outros [Lee et at. 1996], [Martins et at. 2000], [Myers 1979] e [Pimont et at.1979]. De maneira geral para gerar casos de teste utilizando essas técnicas, faz-se necessário a representação do sistema reativo como uma máquina de estados finitos, dessa forma, caso a especificação esteja em uma linguagem de mais alto nível como as mencionadas anteriormente, torna-se necessária a transformação da mesma para máquina de estados finitos.

Este artigo tem como objetivo abordar uma técnica de geração de casos de teste de caixa preta, a partir de especificações em *SDL*, através de um *framework* denominado J-SDL. Este *framework* possibilita a simulação do comportamento de sistemas especificados em *SDL*, em termos de mudança de estado, e também a geração automática e direta (dispensando a transformação para maquina de estados finitos) de casos de teste, a partir de algoritmos de exploração de estados que possam implementar ou não algum critério de cobertura.

2. SDL – Specification and Description Language

A *SDL* (*Specification and Description Language*) é uma linguagem formal para a especificação e descrição de sistemas complexos, que são reativos, mantida pela *ITU* (*International Telecommunication Union*) [ITU 2002] sob a denominação de Recomendação Z.100. Sistemas reativos são aqueles que respondem a algum estímulo também conhecido como sinal ou evento. Esta linguagem possibilita a representação formal de sistemas quanto aos aspectos de comportamento, estrutura, comunicação e descrição de dados.

A primeira versão da Recomendação Z.100 foi disponibilizada pela *CCITT* (*Comitê Consultatif International Télégraphique et Téléphonique*, atual *ITU*) em 1976 apresentando apenas uma abordagem rudimentar de representação de comportamento. Em 1984 foi lançada uma nova versão da linguagem, abordando também representação de estrutura e de comunicação. Apenas em 1988, a *SDL* recebeu embasamento necessário para atingir o status de técnica formal de descrição. Os primeiros conceitos de orientação a objetos foram incluídos na linguagem em 1992 [Ellsberger et at. 1997]. A versão atual da *SDL* foi disponibilizada pela *ITU* em 2002, apresentando suporte aprimorado para modelagem orientada a objetos e recursos adicionais para geração de código-fonte.

No escopo da *SDL*, os termos especificação e descrição possuem significados distintos. O termo especificação é utilizado para representar o comportamento esperado de um sistema, enquanto o termo descrição é utilizado para representar o comportamento real de um sistema. Como a forma de utilização da *SDL* para especificar

ou para descrever é a mesma, é comum o uso do termo especificação de forma genérica para referenciar tanto especificações quanto descrições [ITU 2002].

2.1. Conceitos básicos da linguagem

Uma especificação em *SDL* é um modelo formal que define as propriedades relevantes de um sistema existente ou a ser desenvolvido. No paradigma da *SDL* o mundo é divido em duas partes: o sistema especificado e o ambiente, onde o ambiente é tudo que circunda e se comunica com o sistema em questão.

A comunicação entre ambiente e sistema ocorre através da troca assíncrona de sinais. Desta forma, se existe no ambiente um evento relevante para o sistema, este evento é representado na especificação por meio de um sinal que pode ser enviado pelo ambiente para o sistema. A troca de sinais entre ambiente e sistema ocorre de forma bidirecional.

Uma especificação, quando apropriado, pode ser realizada de forma hierárquica através do particionamento sucessivo de um sistema em níveis diferentes de abstração. Cada nível definido na especificação pode possuir a sua própria especificação, ou seja, pode continuar a ser decomposto em sub-níveis e assim sucessivamente até que sejam representados todos os aspectos desejados do sistema em questão, conforme apresentado na Figura 1.

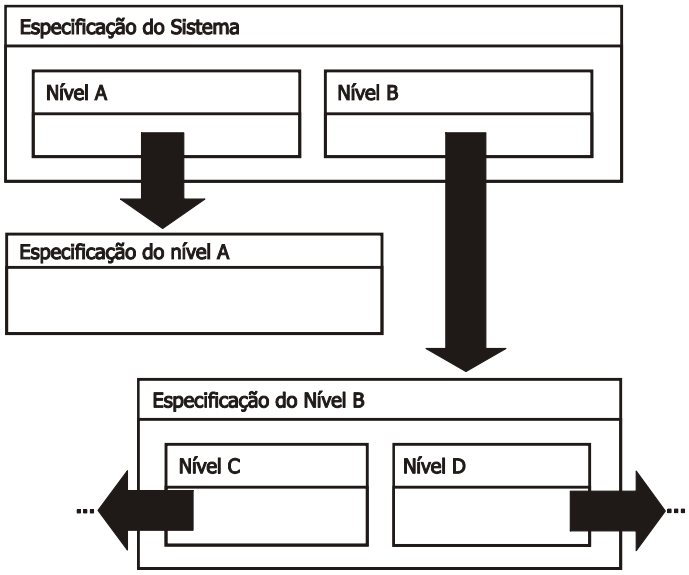

Figura 1 - Particionamento em níveis de abstração.

A *SDL* oferece duas formas de representação: uma textual (SDL-PR) e uma gráfica (SDL-GR). Ambas oferecem formas equivalentes de representação dos mesmos elementos. Os elementos da *SDL* podem ser organizados em três grupos: elementos de estrutura, elementos de comunicação e elementos de comportamento [Ellsberger et at. 1997].

Os elementos de estrutura são aqueles que permitem a representação dos três níveis de abstração existentes no paradigma da *SDL*: sistema (*system*), bloco (*block*) e processo (*process*). Toda especificação possui obrigatoriamente um e apenas um nível

de sistema. Trata-se do nível mais alto de abstração da especificação, onde são definidos os aspectos de comunicação entre sistema e ambiente. Pode-se dizer que este nível é a raiz da especificação.

O nível de bloco funciona como um *container* para outros níveis, sejam eles de bloco ou de processo. A utilização desse tipo de nível não é obrigatória, porem se torna muito útil em projetos reais, onde os diagramas tendem a ficar muito grandes e complexos. A quebra de um nível de sistema ou de bloco em níveis de blocos menores facilita significativamente a manipulação dos diagramas, além de possibilitar que essas "partes" da especificação sejam reutilizadas em outros níveis de abstração, até mesmo de outros sistemas.

O nível de processo permite a representação de porções de comportamento do sistema, por meio do encapsulamento de máquinas de estados finitos estendidas que comunicam entre si através da troca de sinais.

O principal mecanismo de comunicação utilizado dentro da *SDL* é a troca assíncrona de sinais (*signals*). É através do envio de sinais (parametrizados ou não) que os processos trocam informações e se mantêm sincronizados. É importante destacar que os sinais podem ser enviados e recebidos por processos ou pelo ambiente.

Para que um sinal seja transmitido é necessário um meio de transmissão. Na *SDL* este meio é denominado canal (*channels*). Os canais são utilizados para definir uma rota de comunicação entre dois elementos de estrutura. Para definir um canal é preciso que o elemento de estrutura possua algum ponto de conexão. O elemento que representa este ponto de conexão é chamado porta (*gate*).

Em *SDL*, os sinais são emitidos por *broadcast*, por isso é necessário delimitar por onde cada sinal deve ou não propagar, isso pode ser feito através da definição de listas de sinais (*signal list*). As listas de sinais funcionam como filtros que determinam por onde um sinal pode propagar. Caso um sinal não esteja na lista de sinais de um canal, o mesmo não será propagado.

O comportamento dinâmico em *SDL* é definido dentro dos processos. Uma instância de processo é uma maquina de estados estendida que se comunica com outras instâncias de processos ou com o ambiente, por meio da troca assíncrona de sinais. É possível a existência de mais de uma instância do mesmo processo. Cada instância de processo possui um identificador único (*PID*), o que possibilita o envio de sinais diretamente à determinada instância de um processo.

A *SDL* possui diversos elementos utilizados para descrever o comportamento de processos em termos de máquinas de estados estendida, entre eles podemos citar: estado (*state*), início (*start*), entrada (*input*), saída (*output*), registro (*save*) e decisão (*decision*).

2.2. Especificação de um sistema de manufatura em SDL

Neste tópico será apresentada a especificação em *SDL* de um sistema de manufatura, que possui internamente três componentes paralelos, sendo eles: duas máquinas que possuem a função de fabricar algum produto e um supervisor que possui a função de reparar as máquinas caso as mesmas quebrem, podendo consertar apenas uma máquina por vez.

A Figura 2 ilustra o nível de sistema dessa especificação, definido por um nível de bloco denominado *Principal*, que se comunica com o ambiente através do canal *c1* que por sua vez, conecta o sistema e o bloco *Principal* pelos dos portões *a* e *s* respectivamente. O canal *c1* possui uma lista de sinais no sentido do bloco principal que permite a propagação dos sinais *a1*, *a2*, *r1*, *r2*, *f1*, *f2*, *s1* e *s2* para o mesmo.

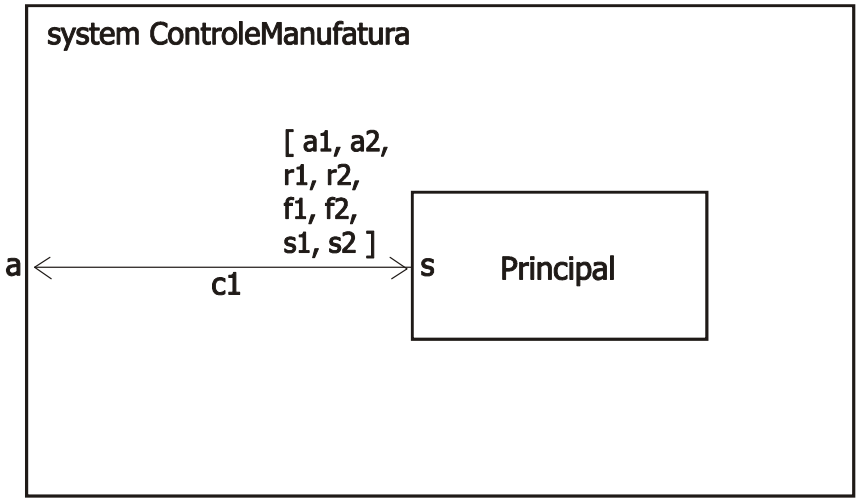

Figura 2 – Definição do nível de sistema da especificação.

Cada um dos sinais existentes dentro de uma especificação em *SDL* mapeia um evento existente no domínio do problema. Estes eventos podem ser classificados como explícitos ou implícitos. São considerados eventos explícitos aqueles que são disparados do ambiente para o sistema e implícitos aqueles que são disparados internamente ao sistema, normalmente em decorrência de uma transição de estado. A Tabela 1 apresenta todos os sinais presentes na especificação deste sistema de manufatura, assim como sua classificação e os eventos que os mesmos representam.

Tabela 1 – Sinais utilizados na especificação.

Sinal	Escopo	Classificação
a1	Início da produção na Máquina1.	Explícito
a2	Início da produção na Máquina2.	Explícito
r1	Conclusão produção na Máquina1.	Explícito
r2	Conclusão produção na Máquina2.	Explícito
f1	Quebra da Máquina1.	Explícito
f2	Quebra da Máquina2.	Explícito
c1	Início do conserto da Máquina1.	Implícito
c2	Início do conserto da Máquina2.	Implícito
s1	Conclusão do conserto da Máquina1.	Explícito
s2	Conclusão do conserto da Máquina2.	Explícito

A Figura 3 apresenta a definição do bloco *Principal*, que é composto por três processos paralelos chamados: *Máquina1*, *Máquina2* e *Supervisor*. Esses três processos são conectados ao bloco *Principal* através da porta *s* pelos canais *c2*, *c4* e *c3* respectivamente. Cada um desses canais possui uma lista de sinais no sentido de

propagação dos processos, o que restringe a propagação dos sinais recebidos pelo bloco *Principal* para seus componentes. Por exemplo, quando um sinal *a1* for recebido pelo sistema através da porta *a* ele será propagado para a porta *s* do bloco *Principal*, através do canal *c1* que possui o sinal s em sua lista de sinais no sentido do bloco *Principal*. Dentro do bloco *Principal*, o sinal *a1* será propagado apenas para o processo *Máquina1*, pois este é o único que possui um canal com uma lista de sinais que contenha o sinal *a1* no seu sentido de propagação.

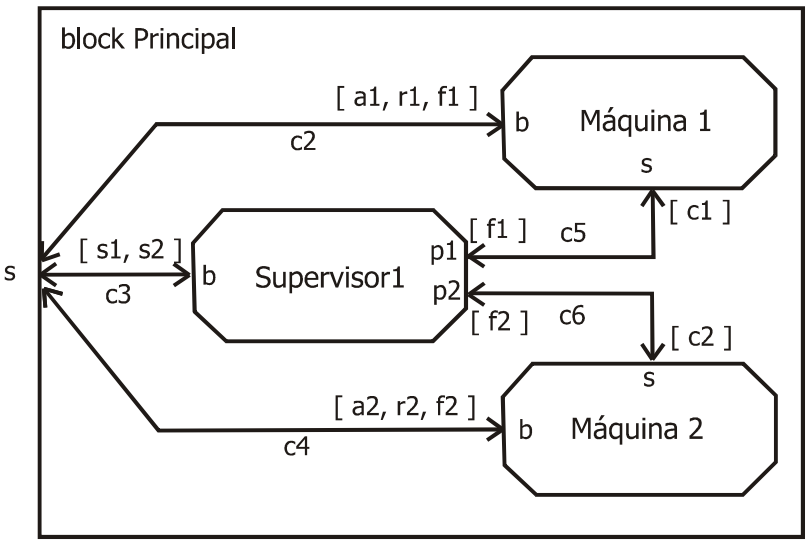

Figura 3 – Definição do bloco Principal.

Os processos *Máquina1* e *Máquina2* trocam sinais com o processo *Supervisor* através dos canais *c5* e *c6* respectivamente. Pela propagação dos sinais *f1* e *f2*, os processos *Maquina1* e *Máquina2* informam ao processo *Supervisor* quando estão quebrados. Pela propagação dos sinais c1 e c2, o processo *Supervisor* informa aos processos *Maquina1* e *Máquina2* que os mesmos foram consertados.

A Figura 4 apresenta a definição do comportamento dos processos *Máquina1* e *Máquina2* que são idênticos em termos de comportamento.

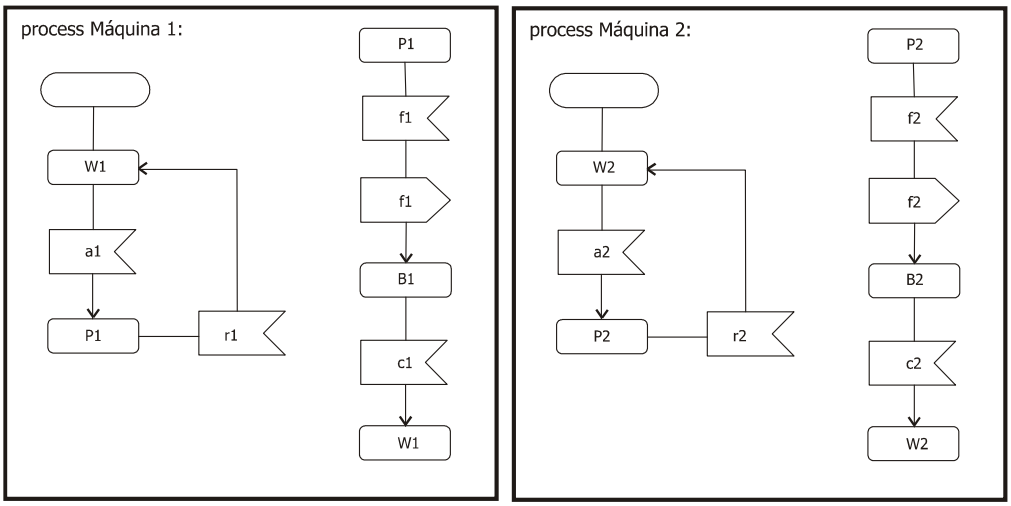

Figura 4 – Definição dos processos Máquina1 e Máquina2.

O processo *Máquina1* possui os estados: *W1*, *P1* e *B1*, sendo *W1* seu estado inicial. O estado *W1* possui uma entrada para o sinal *a1*. Uma entrada em *SDL* define a possibilidade de transição de estados mediante ao recebimento de um sinal, por exemplo, quando o processo *Máquina1* esta no estado *W1* e recebe um sinal *a1*, ocorre uma transição do estado *W1* para o estado *P1*. O estado *P1* possui entradas para os sinais: *r1* e *f1* e o estado *B1* para o sinal *c1*. É interessante destacar que a transição do estado *P1* para o estado *B1* causa o envio do sinal implícito *f1* que irá informar ao processo *Supervisor* que a *Máquina1* esta quebrada. A Tabela 2 apresenta a descrição de todos os estados possíveis para cada um dos processos dessa especificação.

Tabela 2 – Estados da especificação.

Estado	Processo	Significado
W1	Máquina 1	Pronto
P1	Máquina 1	Produzindo
B1	Máquina 1	Quebrado
W2	Máquina 2	Pronto
P2	Máquina 2	Produzindo
B2	Máquina 2	Quebrado
WS	Supervisor	Pronto
C1	Supervisor	Consertando Máquina1
C2	Supervisor	Consertando Máquina2

A Figura 5 apresenta a definição do processo *Supervisor*, que possui os estados *WS*, *C1* e *C2*, sendo *WS* seu estado inicial. O estado *WS* possui entradas para os sinais *f1* e *f2*. O estado *C1* possui entrada para o sinal *s1* e registro para o sinal *f2*. O estado *C2* possui entrada para o sinal *s2* e registro para o sinal *f1*. Em *SDL* um registro tem a função de armazenar um sinal recebido até a próxima transição de estados, por exemplo, quando o processo Supervisor esta no estado *C1* e recebe um sinal *f2*, o mesmo é armazenado, caso o processo receba em seguida o sinal s1, uma transição para o estado *WS* será disparada. Quando a transição para o estado *WS* for concluída o processo Supervisor irá receber naquele momento, o sinal *f2* que foi armazenado anteriormente, causando, nesta situação, a transição do estado *WS* para *C2*.

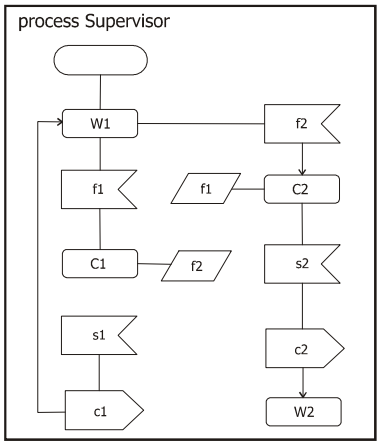

Figura 5 – Definição do processo Supervisor.

3. O framework J-SDL

Para permitir a manipulação de especificações em *SDL*, tendo como objetivo principal auxiliar as atividades do processo de verificação e validação, foi criado um *framework* em Java, denominado J-SDL.

O J-SDL possui internamente um meta-modelo orientado a objetos, que mapeia os principais elementos da linguagem *SDL*, permitindo que especificações nessa linguagem sejam carregadas em memória de forma dinâmica. Além de permitir a representação de especificações em memória o J-SDL possui recursos para a simulação do comportamento de sistemas, em termos de mudança de estado.

Quando uma especificação é carregada pelo J-SDL é criada em memória uma estrutura de dados capaz de reagir a estímulos da mesma forma com que um sistema, que tenha sido construído seguindo a especificação em questão, reagiria. Além disso, é possível obter informações a respeito do estado atual da simulação da especificação e sobre o histórico de transições de estados ocorridas desde que a especificação foi carregada em memória.

O J-SDL atua como uma valiosa ferramenta tanto para a verificação quando para a validação de sistemas. Visando as atividades de verificação é possível utilizar sua API (*Application Program Interface*) para implementar algoritmos com a finalidade de geração automática de casos de testes.

Tendo como foco as atividades de validação, é possível utilizar a API do J-SDL para observar o comportamento que o sistema especificado apresentará caso seja construída em conformidade com a sua especificação. Isso permite que seja realizada uma avaliação para verificar se a especificação realmente corresponde ao sistema que se pretende desenvolver, auxiliando na redução dos riscos de não aceitação do sistema após sua conclusão.

O J-SDL foi construído no formato de *framework* para possibilitar que suas funcionalidades fossem estendidas e aplicadas em implementações com finalidades variadas. Dessa forma, alem da utilização básica da sua API, é possível estender seus recursos, adicionando suporte a outros elementos da *SDL* e ou novas funcionalidades.

4. Gerando casos de teste com o J-SDL

Com o objetivo de verificar a viabilidade de utilização do J-SDL para a geração automática de casos de teste, foi desenvolvido um aplicativo capaz de realizar essa tarefa, através da exploração do espaço de estados de uma especificação em *SDL* pelo algoritmo Transition Tour [Lee & Yannakakis 1996].

O Transition Tour, conhecido também como Método T, consiste em produzir uma seqüência de estímulos, na qual todas as transições de estado previstas pela especificação de um sistema reativo, ocorram pelo menos uma vez. No contexto da *SDL*, uma transição ocorre toda vez que um processo se encontra em um estado e uma entrada desse estado é acionada pelo recebimento de um sinal. Pode-se considerar que para aplicar o Transition Tour em uma especificação em *SDL* é necessário que todas as entradas da especificação sejam ativadas pelo menos uma vez.

A geração de casos de testes a partir do J-SDL é possível através da observação do comportamento de uma simulação que tenha sido carregada em memória a partir de uma especificação em *SDL*. Uma vez que a simulação é iniciada, o *framework* permite que se interaja com a mesma das seguintes formas: estímulo de sinais, obtenção de informações sobre a estrutura da especificação, obtenção de informações sobre o estado da simulação e consulta ao histórico de transições ocorridas desde o início da simulação. São essas informações oferecidas pelo *framework* em tempo de simulação que permitem a geração de casos de teste sem a necessidade de gerar máquinas de estados finitos, que são a modelagem mais natural para a geração de casos de testes.

O aplicativo utiliza as informações oferecidas pelo *framework* sobre a simulação de comportamento para decidir qual sinal estimular, tendo como objetivo conseguir ativar todas as entradas existentes. A seqüência de sinais gerada, associada às mudanças de estado causadas pelo estímulo de cada sinal compõe uma seqüência de teste que pode ser aplicado para verificação de sistemas que tenham sido construídos a partir da especificação em questão. A Tabela 3 ilustra a seqüência de passos utilizados para implementar o Transition Tour utilizando os recursos da API do J-SDL.

Tabela 3 – Passos da implementação do Transition Tour utilizando o J-SDL

Passo	Descrição
1	Carregar a especificação em memória.
2	Iniciar a simulação de comportamento da especificação.
3	Enquanto existir na especificação alguma entrada que não tenha sido ativada, pelo menos uma vez durante a simulação, realizar os passos: 4, 5 e 6, caso contrário ir para o passo 7.
4	Obter uma coleção com todas as entradas da especificação ordenadas de forma crescente pelo número de vezes que foram ativadas, desde o inicio da simulação.
5	Percorrer a coleção estimulando os sinais capazes de ativar as entradas até que uma delas seja ativada, causando uma transição de estado.
6	Retornar ao passo 3.
7	Fim

A Figura 6 apresenta a seqüência de testes gerada pelo programa a partir da especificação do sistema de manufatura. Nessa figura é possível observar as mudanças de estado do sistema a partir do estímulo dos sinais gerados através do J-SDL. A idéia é aplicar essa seqüência de sinais em um sistema real construído segundo a mesma especificação e comparar o comportamento observado ao comportamento previsto.

Máquina1	Máquina2	Supervisor	Sinal
W1	W1	WS	a1
P1	W2	WS	r1
W1	W2	WS	a2
W1	P2	WS	r2
W1	W2	WS	a1
P1	W2	WS	f1
B1	W2	C1	s1
W1	W2	WS	a2
W1	P2	WS	f2
W1	B2	C2	s2
W1	W2	WS	

Figura 6 - Sequência de testes do sistema de manufatura

5. Conclusão e trabalhos futuros

A linguagem *SDL* é uma técnica formal de descrição, muito utilizada no setor de telecomunicações. Por apresentar facilidades para a representação de características típicas de sistemas reativos, torna-se interessante o uso dessa linguagem para a especificação de sistemas de aplicação espacial, principalmente, quando existe a necessidade de manipular especificações por computador, por exemplo, com o propósito de simulação ou geração de casos.

O J-SDL se mostrou uma ferramenta útil para o processo de verificação e validação de sistemas reativos, apresentando duas contribuições principais: a possibilidade de gerar casos de teste de forma direta (dispensando a transformação para maquina de estados finitos) e a possibilidade de integração com outros aplicativos, por se tratar de um *framework*.

Como continuação do trabalho será adicionado ao J-SDL suporte para outros elementos da *SDL*, com o intuito de torná-lo mais abrangente e também será realizada a integração do mesmo com uma ferramenta *web* colaborativa para a geração automática de casos de teste [Arantes et at. 2008].

Referências

Amaral, A. S. (2005). Geração de casos de teste para sistemas especificados em statecharts. Dissertação de Mestrado em Computação Aplicada - INPE, São José dos Campos, p. 164.

Arantes, A. O., Vijaykumar, N. L. , Santiago, V. A. , Carvalho, A. R. (2008) Automatic Test Case Generation through a Collaborative Web Application. In IASTED-EuroIMSA, 2008, Innsbruck. Proceedings of the IASTED International Conference Internet & Multimedia Systems & Applications. Calgary : IASTED, p. 27-32.

Desel, J., Oberweis, A., Zimmer, T. (1997). A test case generator for the validation of high-level Petri nets. In *6th International Conference on Emerging Technologies and Factory Automation Proceedings*, p. 327 – 332.

Ellsberger, I., Hogrefe, D., Sarma, A.(1997), "SDL: Formal Object-oriented Language for Communicating System", Upper Saddle River: Prentice Hall Europe, 2ª Edição. 312 p. ISBN 0-13-632886-5.

Henniger, O., Hasan, U. (2000), Test generation based on control and data dependencies within multi-process SDL specifications. In *2nd Workshop of the SDL Forum Society on SDL and MSC* (SAM 2000), Grenoble, França.

ITU (2002) "ITU-T recomendation Z100: Specification and Description Language SDL", http://www.itu.int/ITU-T/studygroups/com17/languages/Z100.pdf, Março.

Lee, D. e Yannakakis, M. (1996) "Principles and Methods of Testing Finite State Machines – A Survey". In *Proceedings of the IEEE*, p. 84-88.

Martins, E., Sabião, S. B. e Ambrósio, A. M. (2000) "ConData: a tool for automating specification-based test case generation for communication systems". In *33rd Hawaii International Conference* on System Sciences.

Myers, G. J. (1979) “The art of software testing”, John Wiley &Sons, 1th edition, New York, USA .

Pressman, R. S. (2000),”Software engineering: a practitioner's approach, McGraw-Hill International Editions”, 5th edition.

Pimont, S. e Rault, J.C. (1979). An approach towards reliable Software. In 4th International Conference on Software Engineering, Munich, Germany, p. 220-230.

Santiago, V., Amaral, A. S. M., Vijaykumar, N.L., Mattiello-Francisco, M. F., Martins, E., Lopes, O.C. (2006), A Practical Approach for Automated Test Case Generation using Statecharts. In *Second International Workshop on Testing and Quality Assurance for Component-Based Systems* (TQACBS 2006) in the IEEE International Computer Software and Applications Conference (COMPSAC 2006), Vol. II, p.183-188, Chicago, EUA.

Schimitt, M., Ek, A., Grabowskij., Hogrete, D. e Kock, B. (1998) Autolink – Putting SDL-based test generation into practice. In *IFIP TC6 11th International Workshop* on Testing Communicating Systems, p.227-244.

Sommerville, I. (2003), “Software engineering”, Harlow: Addison Wesley, 6ª Edição, p. 592.

Wong, W. E., Sugeta T., Qi Y. e Maldonado J. C. (2005). Smart debugging software architectural design in SDL, *Journal of Systems and Software*, Volume 76, Number 1, pp. 15-28.

Wong, W. E., Sugeta T., Li J. J. e Maldonado J. C.(2003).Coverage testing software architectural design in SDL, *Journal of Computer Networks*, Volume 42, Issue 3, p. 359-374.

Wong, W. E., Sugeta T. e Maldonado J. C. (2005). Structural and Mutation Testing for SDL Specifications: A Case Study. In *Proceedings of The 6th IEEE Latin-American Test Workshop* (LATW), Salvador, Bahia, Brasil.

Wong, W. E., Sugeta T. e Maldonado J. C. (2004). Mutation Testing Applied to Validate SDL Specifications. In *Proceedings of The 16th IFIP International Conference on Testing of Communicating Systems* (TestCom), Oxford, United Kingdom, p.193-208.