Simulação de um Sistema de Bombeamento de Insulina

Universidade do Porto

Faculdade de Engenharia

FEUP

Mestrado Integrado em Engenharia Informática e Computação

Sistemas Críticos 2015

FEUP-SCRI, Grupo 1
João Trindade - ei11118@fe.up.pt
Leonel Araújo - ei11130@fe.up.pt
Pavel Alexeenko - ei11155@fe.up.pt

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

Conteúdo

1 Introdução

1.1 Âmbito

No âmbito da unidade curricular Sistemas Críticos, pertencente ao Mestrado de Engenharia Informática e Computação da Faculdade de Engenharia da Universidade do Porto, foi proposto ao grupo a elaboração de um sistema tolerante a falhas que simulasse o bombeamento de insulina no sangue de um paciente consoante a leitura de glicose obtida a partir de sensores no corpo deste.

O presente relatório apresenta não só a arquitetura do sistema elaborada pelo grupo, mas também as suposições tomadas, os mecanismos de tolerância a falhas implementados e o processo de decisão.

1.2 Descrição do Problema

O problema a solucionar consiste na monitorização dos níveis de glucose de um paciente através de dois sensores e, como resposta a alterações nestes valores, bombeamento de insulina no sangue do doente.

O nível de glucose é medido de minuto em minuto, e a decisão sobre a injeção de insulina é feita a cada três minutos. É necessária ainda uma conversão dos valores dos sensores para uma unidade utilizável nos cálculos a aplicar.

O sistema deve ser tolerante a falhas.

2 Análise do Problema

Dada a situação proposta, podemos dizer que, trantando-se de tratamento médico constante e com intervalos pequenos, é necessário que o sistema tenha não só uma elevada disponibilidade, mas também uma elevada fiabilidade, não comprometendo a saúde do paciente.

Para além da situação descrita acima, temos de ter em atenção a função de conversão dos valores de entrada lidos pelo sensor para valores de glucose. Esta análise é essencial para o sistema, pois devemos impedir que grandes flutuações de dados de entrada (que podem ser causados por falta de precisão dos sensores) causem resultados errados ao ponto de comprometer a vida do paciente. De seguida, mostramos todas as conclusões obtidas a partir da análise da função.

$$g = -3.4 + 1.354 * x + 1.545 * \tan\left(\sqrt[4]{x}\right)$$

Figura 1: Expressão Analítica da Função de Conversão.

Em primeiro lugar, analisando a expressão (Figura 1), podemos facilmente identificar duas principais restrições no domínio dos valores de entrada (x). Por um lado, não podem ser inferiores a zero, pois há uma restrição do domínio em $\sqrt[4]{x}$ e, por outro, a tangente aplicada sobre a raiz faz com que $\sqrt[4]{x}$ só possa tomar valores no intervalo $\mathbb{R} \setminus \{\frac{\pi}{2} + k\pi\}$. Assim, ao calcularmos os zeros da função, concluímos que o sinal da função muda em $x \approx 0.8654$ (primeiro zero da função). A função têm ainda um segundo zero da função em $x \approx 9.1746$.

Dado o contexto do problema, o programa não deve aceitar valores negativos de glucose, e dado que o sinal é positivo a seguir ao primeiro zero, podemos

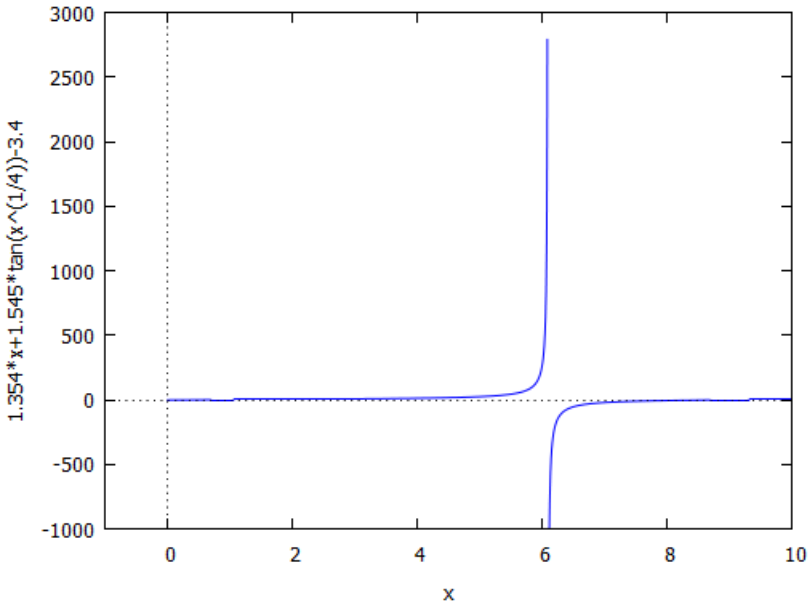

Figura 2: Gráfico da função com x a tomar valores de 0 a 10.

concluir que os dados de entrada devem obedecer, pelo menos, ao conjunto $x > 0.8654 \wedge x < 9.1746$.

A partir da Figura 2, podemos verificar a existência de uma assímptota que reflecte uma súbita mudança no sinal da função em $x = \frac{\pi^4}{16} \approx 6.088$. O que refuta a segunda parte da conclusão obtida anteriormente, sendo necessário uma maior restrição ao nível do limite superior do domínio dos valores de entrada.

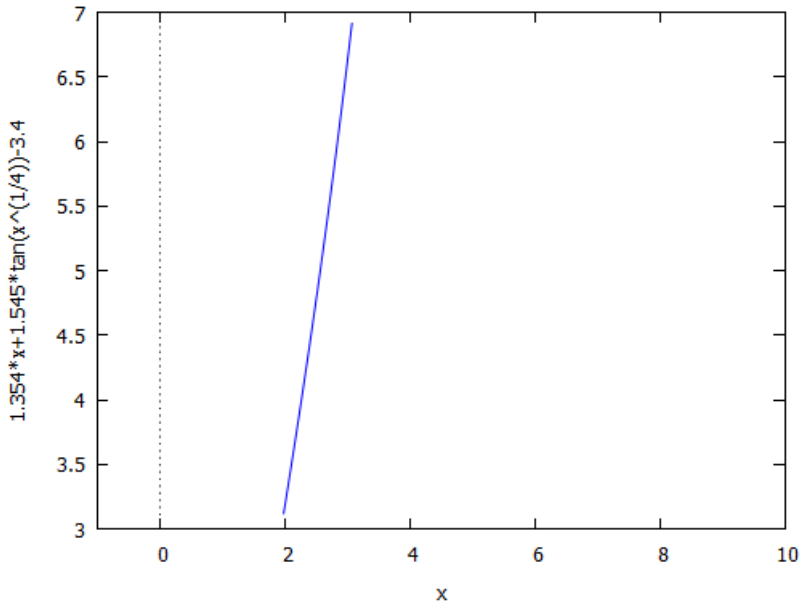

Figura 3: Gráfico da função com contra-domínio de 3 a 7.

Como os valores normais de glucose estão entre 3.9 mmol/l e 6.0 mmol/l (gama de valores que função de conversão deve devolver mais vezes), optamos por reavaliar a função quando o contradomínio varia entre 3 e 7. Como podemos verificar na Figura 3 que os valores que sistema vai receber na maioria dos casos vão estar numa gama reduzida de valores.

O domínio dos valores de entrada que aplicação deve aceitar pode ser representado pelo conjunto $x > 0.8654 \wedge x < 6.088$.

Com esta análise, o grupo considera que não há necessidade de recorrer a reexpressão dos dados de entrada, visto que possíveis erros nestes são facilmente descartados se restringirmos o domínio com as condições supra-mencionadas e visto que não ocorre nenhum súbito crescimento na função tradutora nos dados fornecidos pelos sensores (dados próximos dos limites normais de glucose no corpo de um ser humano).

Por fim, para garantirmos que o paciente é tratado correctamente, mesmo perante sensores em estado de erro, o sistema deve apresentar mensagens de erro a avisar que determinado sensor pode estar estragado e deve contactar o seu médico e/ou um técnico.

3 Arquitetura Utilizada

Dada a situação proposta, o grupo assume que é essencial que o programa tenha elevada disponibilidade, pois é necessário que o paciente seja constantemente tratado. O grupo considera ainda que ao oferecer grande diversidade à implementação com três variantes e ao seleccionar uma arquitetura com verificações dos dados, garantimos suficiente fidelidade que uma aplicação do género necessita.

Tal como mencionado anteriormente e tal como sugerido no enunciado, o grupo considerou que um modelo baseado no desenvolvimento de três variantes e execução paralela - baseado no modelo N-Version Programing - se enquadra no contexto do problema. Assim, a implementação seguiu o modelo *Acceptance Voting*.

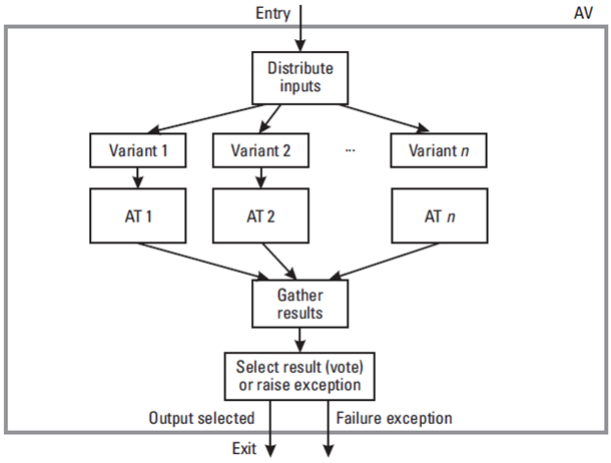

Figura 4: Exemplo de Arquitetura Acceptance Voting para N Variantes.

Tal como outros modelos que oferecem *Design Diversity*, o *Acceptance Voting* aumenta a probabilidade de detectarmos erros na execução e a probabilidade

de cada variante falhar em diferentes situações.

Como podemos ver na Figura 4, o Acceptance Voting difere do N-Version Programming ao implementar testes de aceitação ao nível de cada variante. Posteriormente os resultados são aglomerados e analisados por um votador dinâmico. Trata-se portanto de um modelo com *Forward Recovery*.

A escolha pelo Acceptance Voting baseou-se essencialmente na preferência do grupo pela disponibilidade do sistema. Esta é garantida pelo Acceptance Voting, na medida em que mesmo que uma Variante falhe, o votador dinâmico poderá continuar a efetuar a eleição do valor final, evitando a suspensão do serviço. Para além disto, garante um valor de overhead baixo, o que é muito importante para aplicações como esta que necessitam de respostas atempadas. Por fim, o facto de não termos muitas vantagens em aplicarmos técnicas de *Data Diversity* (DRA) também nos direccionou para esta arquitetura, caso contrário o grupo teria optado pelo modelo *Two-Pass Adjudicator*.

Como referido anteriormente, foram implementadas três variantes. Numa tentativa de oferecer maior diversidade à aplicação e, consequentemente, maior tolerância a falhas. Cada variante foi desenvolvida numa linguagem diferente. Assim, o nosso programa possui uma variante desenvolvida em Java, outra em C++ e outra em ADA.

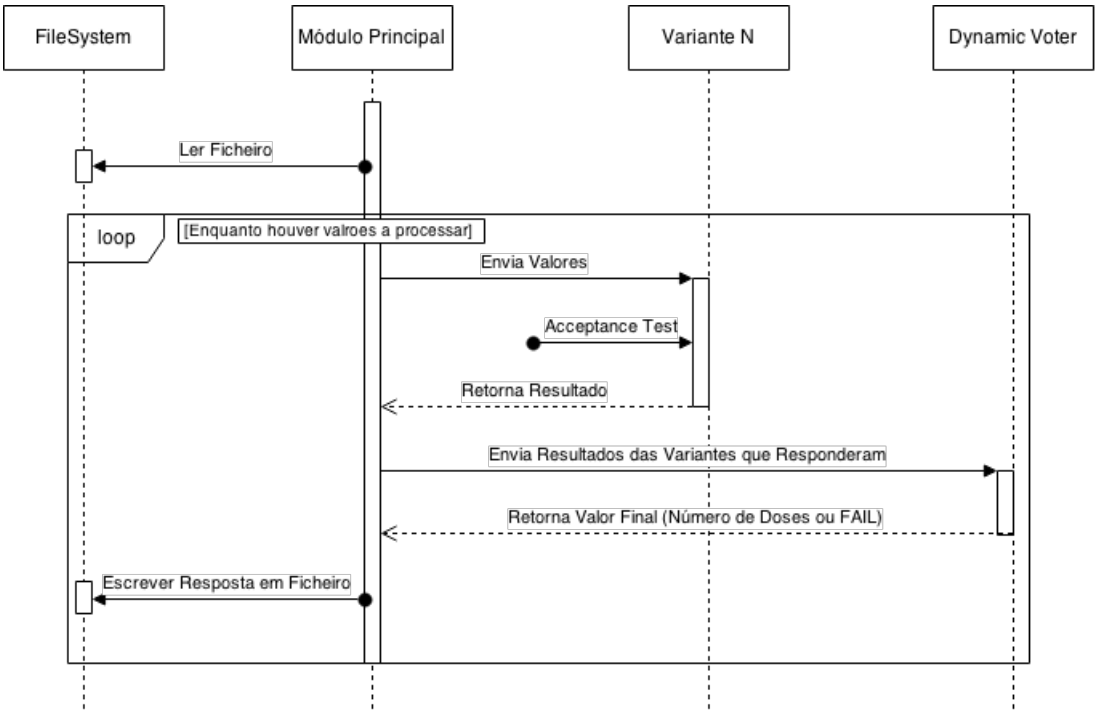

Figura 5: Diagrama de Sequência da Aplicação Desenvovlida.

Na Figura 5 podemos ver a sequência que a aplicação toma quando é executada. De notar que o *Dynamic Voter* é apenas um módulo do programa "Módulo Principal", enquanto que cada Variante N corresponde a um processo diferente.

Como já referido anteriormente, o grupo assume a necessidade de oferecer à aplicação uma grande disponibilidade, pelo que optou pela implemntação de um *Formal Majority Voter*. Esta decisão baseia-se em dois pontos essenciais. Em

primeiro lugar, este votador oferece dinamismo à aplicação, fazendo com que o facto de uma variante falhar não quebre o sistema, garantindo a continuidade do tratamento do paciente, dependendo, todavia, em apenas duas variantes. Em segundo lugar, embora os números de doses sejam valores inteiros, os cálculos associados são efectuados sobre valores com várias casas decimais, logo alguns arredondamentos feitos nas diferentes variantes podem resultar em resultados finais diferentes, como por exemplo: a Variante 1 assume que devem ser injectadas 3 doses e a Variante 2 assume que devem ser injectadas 4 doses quando o resultado de ambas foi 3.5. Nestes casos, o grupo assume que não há uma única resposta correcta, e portanto o paciente não será prejudicado se for injectado com, no máximo, uma dose a mais, daí a implementação do *Formal Majority Voter* que introduz o conceito de *Feasible Set*, onde para o exemplo dado, ambos os valores entrariam.

4 Mecanismos de Tolerância a Falhas Adotados

Para além da diversidade que o sistema têm ao implementarmos um modelo baseado em *N-Version Programming*, é importante que a aplicação consiga evitar entrar em estado de erro e para o caso de entrar conseguir resolver a falha e/ou descartar erros parecidos futuros.

Tal como mencionado anteriormente, para evitar que a função da glucose devolva valores impossíveis (glucose negativa ou valores sem nexo) são efectuadas verificações por intervalo dos dados de entrada. Estas verificações são bastante simples e resultam da análise da função, permitindo detectar erros nos sensores. Quando um sensor submete 5 valores errados as variantes assumem posições diferentes relativamente aquele sensor (enquanto que uma descarta todos os valores futuros, outra continua a usá-los avisando apenas que o sensor pode estar em estado de erro).

Outro mecanismo aplicado, intrínseco ao *Acceptance Voting*, são testes de aceitação ao nível de cada variante. Os testes deste género devem ser simples e facilmente implementáveis, para se facilmente detetarem erros nestes. Assim, os testes efectuados têm apenas em consideração a capacidade da bomba (o grupo assumiu um máximo de 400, é utilizado por uma das variantes), e o número de doses máximo por iteração (o grupo assumiu um máximo de 10, restição usada por outra variante).

Finalmente, o grupo achou que era necessário tomar considerações relativamente à comunicação, de forma a conferir uma maior fidelidade a esta. A comunicação entre as variantes e o programa principal é feita por sockets e existem dois tipos de mensagens:

putdata

- Mensagem enviada pelo Módulo Principal a cada uma das variantes
- Contém informação que uma variante precisa para efectuar os cálculos de uma iteração.
- São enviados o número da iteração, um *timestamp* e 6 valores lidos pelos 2 sensores (correspondentes aos 3 minutos)

putresult

- Mensagem enviada por cada variante ao módulo principal.
- Contém o resultado que a variante obteve.

- São enviados o número da iteração (It), um *timestamp* (TS) e o valor inteiro do número de doses.

Dada a falta de fidelidade de dados que a utilização de sockets traz, o grupou optou por implementar um CRC no final de cada mensagem. A Hash resultante é anexada no final de cada mensagem. Tanto o Módulo Principal como as variantes verificam a Hash da mensagem recebida antes de utilizar os valores para votação (no caso do Módulo Principal) ou cálculos (no caso das variantes).

Exemplos de mensagens:

```
1 // putdata It TS Minuto1Sensor1 M1S2 M2S1 M2S2 M3S2 MS2 Hash
2 putdata 0 1432219608371 2.03 2.05 2.04 3.01 2.89 2.94
      9cf759229a4173de4792695ed1353362
3
4 // putresult It TS Response Hash
5 putresult 0 1432219608371 2 e66563e0dc7472d619a23057056ab9df
```

Por fim, de notar a utilização de um mecanismo detetor de erros do tipo "stuckat"numa das variantes. Quando é notado o constante retorno do mesmo valor por parte de um sensor durante pelo menos 5 min, a variante deixa de utilizar os valores desse sensor durante essa iteração. Enquanto que o valor dado por esse sensor não mudar, a variante assume que esse sensor está em estado de erro e não utiliza os seus valores. Optamos por deixar o programa a voltar a utilizar os valores deste sensor mal os valores passados voltem a mudar, pois o grupo considera que pode haver uma regularidade nos níveis de glucose do paciente e, portanto, poderíamos estar a considerar um erro quando este não existe.

5 Conclusão

O grupo considera que abordou todos os conceitos relevantes de sistemas com tolerância a falhas e implementou com sucessou a arquitetura necessária para garantir um sistema funcional com elevada fiabilidade e com ainda maior enfase na disponibilidade.

6 Recursos Utilizados

Pullum, Laura L.; Software fault tolerance techniques and implementation. ISBN: 1-58053-137-7

Slides fornecidos na página da unidade curricular Sistemas Críticos - "Software Tolerance Techniques"