View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Repositório da Universidade Nova de Lisboa

Universidade Nova de Lisboa Faculdade de Ciências e Tecnologia

Departamento de Informática

Dissertação de Mestrado em Engenharia Informática

2º Semestre, 2008/2009

Verificação de protocolos de e-voting

Aluno nº29721 Maria de Fátima Rodrigues Reis

Orientador

Prof. Doutora Carla Ferreira

Dissertação apresentada na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa para obtenção do grau de Mestre em Engenharia Informática.

Lisboa, 02 de Novembro de 2009

Nº do aluno: 29721

Nome: Maria de Fátima Rodrigues Reis

Título da dissertação: Verificação de protocolos de e-voting

Palavras-Chave:

- Verificação por modelos
- Votação electrónica
- Sistemas distribuídos
- Segurança
- Democracia
- Privacidade

Resumo

Os sistemas de votação electrónica, designados também por *e-voting*, são sistemas informáticos que permitem aos eleitores não só registarem-se para poder exercer o seu direito de voto, como também expressarem-no de forma electrónica e com o consequente apuramento por parte das autoridades competentes do resultado das eleições. Dada a sua relevância a todos os níveis da sociedade é crucial que todos os elementos envolvidos num sistema de votação electrónica tenham confiança no sistema utilizado. No final devem ter a certeza que o sistema proporcionou um bom escrutínio e que reflecte exactamente o que era esperado dele. Para tal, é necessário que se adoptem as medidas que permitem assegurar a segurança a diversos níveis, nomeadamente: privacidade, democracia, possibilidade de verificação e precisão, entre outras.

Através da verificação formal de protocolos e utilizando ferramentas de verificação de modelos, pode-se caminhar para atingir a confiança necessária neste tipo de sistemas. Estas ferramentas permitem a modelação e validação de propriedades de um protocolo, avaliando a sua correcção e identificando problemas na sua especificação.

Pretende-se contribuir para que o sistema de votação electrónica passe a ser uma realidade e assim facilitando o papel de todos os intervenientes nos processos eleitorais. Os sistemas de votação electrónica, poderão ajudar no combate à abstenção, proporcionar melhor acesso a deficientes motores e melhorar privacidade para invisuais.

Neste trabalho foi analisado o protocolo REVS (Robust Electronic Voting System) vocacionado para operar em sistemas distribuídos como a Internet. Utilizaram-se duas ferramentas de verificação de modelos na verificação de propriedades relevantes. Como resultado, identificaram-se problemas já conhecidos nos protocolos de votação electrónica e levantaram-se outros tipos de questões em relação à sua implementação que podem ser alvo de futuros estudos.

Abstract

Electronic voting systems, also called e-voting systems, are electronic systems that enable voters to register and cast their vote. At the end of the election tallying authoritites are able to count and publish election results. E-voting systems are highly relevant to the society in general. For this reason, it is crucial that all intervenients trust the e-voting system. They must be sure that the system provided a reliable poll, which reflects what was expected from it.
In order to achieve those features we will need to adopt the necessary measures to enable security at all levels, namely: privacy, democracy, verification, and precision, among others.

By using formal protocol verification through model checking tools, it is possible to contribute to reach the aimed trust on e-voting systems. These tools enable protocol modeling and properties verification by evaluating its accuracy and help identifying problems in its specification.

We aim to contribute to the electronic voting system becoming a reality and thus facilitating the role of all players in the electoral process. E-voting systems will help in abstention reduction and to improve voting access to physically and visually impaired people.

In this document we analyze REVS (Robust Electronic Voting System) protocol wich was designed to operate on distributed systems like Internet. We use two model checking tools for protocol modeling and properties verification. As a result, we have identified some already known issues in these protocols. We rose up other issues related to its implementation that might be studied in future works.

Prefácio

O presente documento reúne o trabalho de investigação, desenvolvimento, apresentação e conclusões referente à "Dissertação de Mestrado em Engenharia Informática" da Faculdade de Ciências e Tecnologias da Universidade Nova de Lisboa.

Os meus agradecimentos à professora orientadora Carla Ferreira, pela sua disponibilidade, paciência, acessibilidade e principalmente contributos ao nível de motivação e condução no trabalho realizado.

Obrigada à minha amiga, Eunice Silva e aos meus colegas de trabalho Fiona Joosab e João Azinhais, pelas valiosas contribuições nas revisões do texto.

Um reconhecimento muito especial para todos os familiares e amigos que durante o tempo de trabalho nesta dissertação me apoiaram e deram força para que conseguisse atingir este objectivo.

Dedico este trabalho aos meus afilhados, Carla Reis e Ricardo Reis, para que sejam sempre persistentes, superem as dificuldades e não desistam.

Maria de Fátima Rodrigues Reis
02 de Novembro de 2009

Índice de Matérias

1 Introdução ... 11

2 Protocolos votação electrónica e suas características ... 15

2.1 Intervenientes do processo de votação ... 15

2.2 Fases do processo de votação ... 16

2.3 Propriedades de um sistema de votação ... 18

2.4 REVS – Robust Electronic Voting System ... 20

3 Verificação ... 27

3.1 SPIN ... 35

3.2 UPPAAL ... 41

4 Verificação de protocolos de votação electrónica com SPIN ... 47

4.1 Modelação ... 49

4.1.1 Opções tomadas ... 49

4.1.2 Representação do protocolo e principais estruturas utilizadas ... 50

4.1.3 Descrição dos Processos ... 53

4.2 Simulação ... 57

4.3 Verificação de propriedades genéricas ... 63

4.4 Verificação de propriedades de votação electrónica ... 65

4.4.1 Privacidade ... 68

4.4.2 Democracia ... 68

4.4.3 Precisão ... 69

4.4.4 Fiabilidade ... 69

4.4.5 REVS – Assinaturas para o voto ser válido ... 69

4.4.6 REVS – Eleitor termina processo de votação ... 69

4.4.7 REVS – Eleitor pode não terminar processo de votação ... 70

4.4.8 REVS - Ordem dos votos 70
4.4.9 Apenas verificável em SPIN – Linear Temporal Logic 71
5 Verificação de protocolos de votação electrónica com UPPAAL 72
5.1 Modelação 76
5.1.1 Opções tomadas 76
5.1.2 Representação do protocolo e principais estruturas utilizadas 77
5.1.3 Descrição dos Processos 81
5.2 Simulação 86
5.3 Verificação de propriedades genéricas 87
5.4 Verificação de propriedades de sistemas de votação electrónica 88
5.4.1 Privacidade 88
5.4.2 Democracia 89
5.4.3 Precisão 89
5.4.4 Fiabilidade 89
5.4.5 REVS – Assinaturas para o voto ser válido 89
5.4.6 REVS – Eleitor termina processo de votação 90
5.4.7 REVS – Eleitor pode não terminar processo de votação 90
5.4.8 REVS e UPPAAL – Ordem dos votos com tempo 90
5.4.9 REVS – Lógica Temporal Ramificada 91
6 Comparação de SPIN e UPPAAL na verificação de protocolos de votação electrónica .. 93
7 Intrusos 96
8 Conclusões e trabalho futuro 99
Referências 101
A. Modelo SPIN 104
A.1. Propriedades SPIN 107
B. Modelo UPPAAL - XML 118
B.1. Propriedades UPPAAL 124
C. Modelo SPIN – Intruso 125
C.1. Propriedades SPIN – intruso 129

Índice de Figuras

Figura 2.1 Processo Genérico de votação electrónica [6] 17
Figura 2.2 Arquitectura REVS 23
Figura 3.1 A verificação de propriedades em SPIN 36
Figura 3.2 Invariância 39
Figura 3.3 Resposta 39
Figura 3.4 Precedência 40
Figura 3.5 Objectivo 40
Figura 3.4 Verificação de propriedades em UPPAAL 42
Figura 3.5 E<> p – Existe eventualmente p 43
Figura 3.6 E[] p – p existe globalmente 43
Figura 3.7 A[] p 44
Figura 3.8 A<> p – "Sempre eventualmente p" 44
Figura 3.9 q->p – "q leva sempre a p" 45
Figura 4.1 Comunicação entre os tipos de objectos em SPIN 48
Figura 4.2 Validação de sintaxe em SPIN 58
Figura 4.3 Validação de redundâncias no modelo 58
Figura 4.4 Janela de simulação do SPIN 59
Figura 4.5 Resultado de uma simulação 59
Figura 4.6 Valores das variáveis e canais numa simulação 60
Figura 4.7 Visualização gráfica da simulação aleatória 1 61
Figura 4.8 Visualização gráfica da simulação aleatória 2 62
Figura 4.9 Opções de verificação 63
Figura 4.10 Resultado de uma verificação 64
Figura 4.11 Output de uma verificação de propriedade 66
Figura 5.1 Subsistemas do UPPAAL 72
Figura 5.2 Configuração de um arco na modelação de um processo 73
Figura 5.3 Configuração de um nó na modelação de um processo 74

Figura 5.4 Processo ModuloVoto 82
Figura 5.5 Processo Distribuidor 83
Figura 5.6 Processo Administrador 84
Figura 5.7 Processo Anonimizador 84
Figura 5.8 Processo Totalizador 85
Figura 5.9 Processo Temporizador 86
Figura 5.10 Janela de simulação do SPIN 86
Figura 5.11 Resultado de uma verificação 88

Índice de Quadros

Tabela 3-1 Resumo de trabalhos em verificação 29

Tabela 4-2 SPIN - Processos do modelo e suas instanciações de execução 50

Tabela 5-2 UPPAAL - Processos do modelo e suas instanciações de execução 77

Tabela 6-1 Comparação das ferramentas SPIN e UPPAAL 94

Tabela 6-2 Comparação das verificações em SPIN e UPPAAL 95

1 Introdução

A comunidade científica tem desenvolvido um vasto trabalho na área de segurança de protocolos de votação electrónica no sentido de mostrar de que forma estes podem ser utilizados. Deste modo contribui para que os mesmos possam ser usados com todas as garantias dos sistemas de votação não electrónicos, ou seja, convencionais.

Quando é utilizado voto em papel, o eleitor pode facilmente perceber e ficar confortável com o facto de o seu voto ser correctamente efectuado. Ao chegar à mesa de eleição, o eleitor é identificado, recebe um boletim de voto para efectuar a sua escolha e deposita o mesmo já preenchido com o resultado da sua escolha na urna, na presença das autoridades eleitorais que fiscalizam o acto. O eleitor confirma que já votou através de assinatura para que não possa votar novamente. No final do período de votação as urnas são abertas na presença das autoridades competentes e os votos são contados. Estes já não têm qualquer ligação ao eleitor que os efectuou. A contagem é feita na presença de entidades reconhecidas pela comissão eleitoral que assegura a idoneidade do processo. Após essa fase todos os resultados das contagens são combinados e os resultados da eleição oficialmente apresentados. O eleitor tem a certeza que votou em privacidade e que ninguém irá poder votar utilizando a sua identidade ou associar um dos votos nas urnas ao seu próprio voto. Para além do processo descrito, existem sempre outras organizações responsáveis por monitorizar e avaliar se o acto eleitoral decorreu de acordo com as regras.

Quando o voto é efectuado de forma electrónica, existe a possibilidade de fraude. Sendo grande parte do processo efectuado de forma electrónica, os sistemas de votação electrónica estão sujeitos a ataques informáticos que podem por em risco a fiabilidade das eleições. Para os eleitores e outras entidades envolvidas terem confiança no processo de votação deverá ser garantido, em todas as fases, que não ocorrem interferências que possam alterar o resultado do mesmo. O eleitor pretende votar em privacidade. A comissão eleitoral e os candidatos pretendem uma eleição fidedigna que

reflicta exactamente a realidade em termos, por exemplo, de número de eleitores, número de votos, integridade dos mesmos e garantia de privacidade.

Neste contexto, a garantia de propriedades como democracia e privacidade são essenciais à utilização de um sistema de votação electrónica. Ataques a protocolos de segurança são difíceis de prevenir tendo em conta o ambiente distribuído onde estes sistemas operam e a grande variedade de ataques a que estes protocolos estão sujeitos.

A análise e verificação formal disponibilizam uma abordagem rigorosa que complementa a execução de testes. Esta não é suficiente por si só para determinar que um sistema é fiável, mas contribui para tal e é portanto uma vertente que deve ser explorada.

Neste âmbito, foi efectuado o levantamento detalhado de um dos protocolos existentes que implementa um processo de votação electrónica, o REVS (Robust Electronic Voting System) [1]. O REVS foi escolhido por ser um protocolo desenhado para ser utilizado através da Internet, indo assim de encontro à melhoria de acessibilidade aos cidadãos para a realização dos seus deveres cívicos. Por outro lado, o REVS é baseado noutros protocolos já existentes, como por exemplo o protocolo com múltiplos administradores desenvolvido por DuRette's [2]. O protocolo com múltiplos administradores era por sua vez uma melhoria proposta para o sistema EVOX por Herschberg [3]. O protocolo REVS melhora assim algumas vertentes, como por exemplo a disponibilidade e tolerância a falhas, proporcionando assim uma solução mais fiável.

Identificaram-se ainda as propriedades implementadas pelo REVS e algumas das formas de verificação das mesmas, nomeadamente através das ferramentas para modelação e verificação de protocolos SPIN (**S**imple **P**romela **In**terpreter) [4] e UPPAAL (**UPP**SALA e **AAL**BORG) [5]. Foi ainda efectuado um estudo comparativo das propriedades verificadas nas duas ferramentas estudadas.

A verificação formal é uma técnica complementar aos testes e à simulação que permite aferir a correcção de um sistema ou protocolo, ou seja, garantir que ele está conforme as especificações de desenho e propriedades definidas. Através da verificação poder-se-á garantir, por exemplo, que de acordo com a especificação, um voto inválido não será contado pelo sistema de escrutínio [6].

Ambas as ferramentas, SPIN [4] e UPPAAL [5], permitem a verificação automática de sistemas. Para verificar um sistema é necessário construir um modelo do mesmo. Este é criado baseado na sua especificação. Com base no modelo, a ferramenta efectua a

verificação, permitindo validar se os requisitos definidos na especificação são cumpridos.
A verificação não é a única forma de garantir a fiabilidade de um sistema. É apenas uma das formas de contribuir para esse objectivo. São alvos de verificação, por exemplo, que o sistema de voto:

- É consistente;
- Está de acordo com as normas;
- Utiliza técnicas de confiança e boas práticas;
- Escolhe as funções correctas e da forma correcta;
- Está de acordo com os requisitos de correcção e de completude, entre outros, para todos os passos da votação.

O presente trabalho foca-se essencialmente neste último ponto utilizando para tal as ferramentas de verificação estudadas, nomeadamente o SPIN [4] e o UPPAL [5].
Ambas permitem modelar os requisitos de um protocolo através de linguagens e estruturas próprias que permitem uma abstracção em relação aos detalhes de uma especificação. Permitem ainda efectuar a verificação de sintaxe dos sistemas descritos no modelo, a simulação da sua execução e a verificação de propriedades.
Efectuou-se um estudo generalizado do processo de votação convencional e electrónico e utilizou-se o protocolo REVS, que implementa um sistema de votação electrónica, como olução base para estudo.
O presente documento apresenta nas secções seguintes uma descrição mais generalizada dos protocolos de votação electrónica e suas características, nomeadamente:

- Intervenientes
- Fases do processo de votação
- Propriedades de um sistema de votação

Na Secção 2 faz-se uma introdução sobre os protocolos de votação electrónica e convencional, caracterizando-os. Apresentam-se também os intervenientes, as propriedades e detalha-se o protocolo REVS.

Na secção 3 apresentam-se as ferramentas de verificação SPIN E UPPAAL. São apresentadas as linguagens que são utilizadas para formalização de verificação de propriedades. Nas Secções 4 e 5 detalha-se a modelação, simulação e verificação com as ferramentas SPIN e UPPAAL. A modelação do protocolo REVS é descrita, em termos de opções tomadas e estruturas de dados utilizadas. Na Secção 6 é apresentada uma análise comparativa entre estas ferramentas, e as propriedades verificadas por elas. Na secção 7 é apresentado um modelo de intruso que tenta subverter o sistema de votação duplicando votos e mostra-se através da verificação que o modelo do REVS o evita. Na secção 8 apresentam-se as conclusões deste trabalho e direcções futuras de melhoria ou exploração de outras vertentes de estudo na área de verificação de protocolos de votação electrónica.

2 Protocolos votação electrónica e suas características

Votação electrónica refere-se, de uma forma generalizada, à possibilidade dos indivíduos votarem electronicamente numa eleição.
O processo de votação convencional baseado em papel e nas entidades reconhecidas e autorizadas para executar, monitorizar e garantir uma votação é aceite pela comunidade.
Um protocolo de votação electrónica deve assegurar que todas as fases do processo de votação são atingidas e concluídas devidamente, garantindo assim a fiabilidade do processo de votação. Muita da desconfiança em relação a estes sistemas assenta sobretudo na não garantia de fiabilidade no processo de votação.
De seguida apresenta-se os intervenientes e fases de um sistema de votação electrónica que implementam os processos de uma votação convencional [6].

2.1 Intervenientes do processo de votação

Eleitor

Tem o direito de voto e vota na eleição.

Entidades de registo

Registam os eleitores elegíveis para votarem nas eleições, previamente ao início das mesmas. Estas entidades asseguram que apenas os eleitores registados podem votar e que os mesmos o podem fazer apenas uma vez no decorrer do acto eleitoral. As entidades de registo têm como função registar os eleitores, autorizar e validar os mesmos, distribuir os boletins de voto e gerar chaves de segurança para as funções que disponibilizam.

Entidades de cálculo

Reúnem os votos efectuados pelos eleitores e calculam o resultado das eleições.

2.2 Fases do processo de votação

Existem seis fases que indicam como é que cada uma destas entidades interage entre si. Estas fases descrevem de uma forma genérica o processo eleitoral [1, 6].

Pré-registo/Registo

Esta fase precede o acto eleitoral. Os eleitores que querem votar inscrevem-se perante as entidades de registo para que a lista de eleitores fique disponível antes do acto eleitoral. Os boletins de voto são preparados para que estejam finalizados no momento da eleição. Nem todos os protocolos de votação electrónica implementam esta fase.

Validação de autenticação e autorização

No dia da eleição, os eleitores registados requerem um boletim de voto e autorização para votar às entidades de registo. As entidades de registo verificam as credenciais de acesso do eleitor e caso este esteja registado, é aceite para o acto eleitoral.

Votar

O eleitor exerce o seu direito de voto. O eleitor apenas pode votar uma vez por eleição.

Verificação

É verificada a validade dos votos para que apenas os votos válidos sejam contados na fase final de apuramento de contagem de votos.

Cálculo

Esta fase envolve a contagem dos votos e respectiva divulgação de resultados.

Reclamações

Após o acto eleitoral, qualquer entidade pode reclamar eventuais más práticas que deverão ser averiguadas e validadas antes da publicação e homologação dos resultados finais da votação.

Nem todos os protocolos de votação electrónica observam as fases apresentadas. Por exemplo, o REVS não considera a fase de Reclamações apesar de disponibilizar uma forma de um eleitor saber se o seu voto foi ou não contado e dessa forma permitir que o eleitor faça uma reclamação e que a mesma seja verificada.

A imagem seguinte representa de uma forma genérica a interacção entre os intervenientes num processo de votação [6] proposto por Cetinkaya [7], Cranor [8] e Fujioka [26].

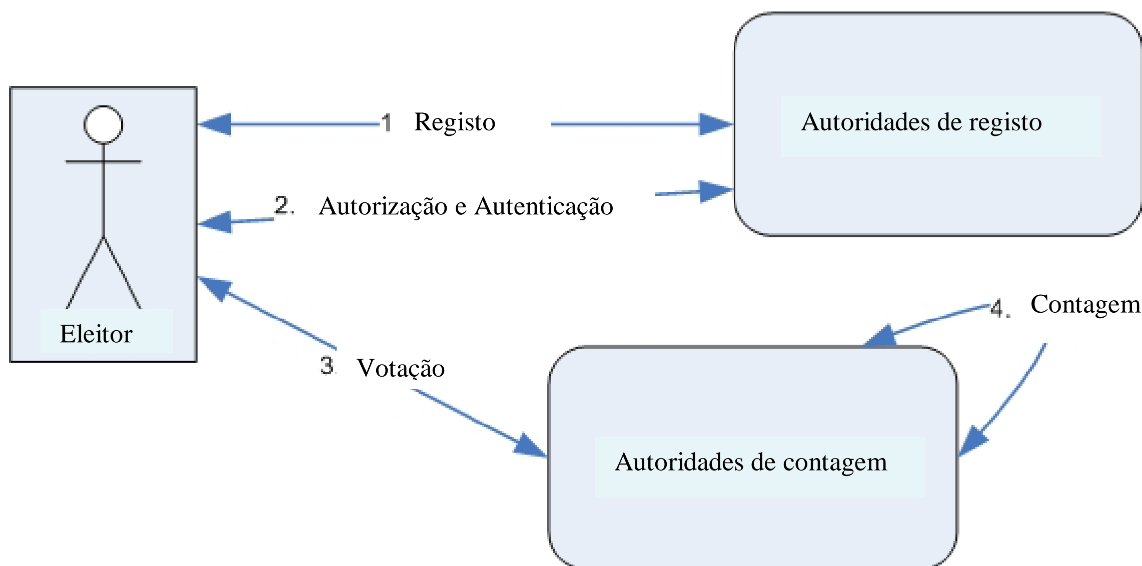

Figura 2.1 Processo Genérico de votação electrónica [6]

Em primeiro lugar e previamente à eleição, os eleitores registam-se perante as autoridades competentes.

De seguida, já no período definido para a eleição, cada eleitor contacta as autoridades perante as quais se registou, autentica-se e é-lhe dada autorização para votar. Após escolha da sua opção de voto, o eleitor submete o seu voto.

Após o término do período da eleição, as entidades de contagem apuram o resultado da votação e publicam os resultados.

2.3 Propriedades de um sistema de votação

De seguida apresentam-se as propriedades básicas de segurança que um sistema de votação electrónica deverá assegurar propostas por Cranor e Cytron [8]. Cada propriedade foi dividida em vários requisitos, referidos por Cetinkaya [10], Cranor [8] e Fujioka [9], seguindo uma estrutura similar aos trabalhos [1, 6, 11].

Democracia

A democracia é caracterizada por dois requisitos:

Elegibilidade - Apenas os eleitores elegíveis votam no escrutínio.

Unicidade – Os eleitores elegíveis apenas podem votar uma única vez.

Privacidade

A privacidade pode ser caracterizada por três requisitos:

Anonimato - Não deve ser possível estabelecer uma ligação entre o eleitor e o seu voto, quer pelas entidades eleitorais, como por quaisquer outras entidades. Esta propriedade deverá ser assegurada durante o processo eleitoral e por um período longo após as eleições de modo a garantir a não coercibilidade.

Não Coercibilidade - Assegura que ninguém pode ser forçado a votar de determinada forma. Os eleitores devem votar de modo totalmente livre.

Ausência de Recibo de Voto - Nenhum eleitor pode provar que votou de determinada forma. Esta característica evita possíveis burlas que envolvam compra ou venda de votos.

Verificação de voto

Qualquer entidade deverá poder verificar todos os votos que foram efectuados e se foram contados correctamente. Significa portanto que ninguém pode falsificar o resultado e uma votação.

Precisão

A precisão pode ser caracterizada por dois requisitos:

Precisão - Um voto tem que reflectir a escolha do eleitor e não pode em ponto algum do sistema ser adulterado. Se um voto for inválido, este não deverá ser incluído no escrutínio final. Qualquer ataque aos votos deverá ser identificado e invalidado. Se um voto for efectuado com sucesso pelo eleitor e for válido, este terá que ser obrigatoriamente contado no escrutínio final.

Unicidade - Apenas pode ser contada um voto por eleitor.

Apresentam-se de seguida ainda outras propriedades que deverão ser asseguradas para garantir a fiabilidade do sistema.

Justiça

Não deve ser possível consultar ou publicar contagens parciais no decorrer do processo eleitoral para assegurar que todos os candidatos estão em igualdade de circunstâncias.

Transparência

Deve ser disponibilizada informação sobre o processo eleitoral para que os eleitores estejam informados do acto eleitoral. A segurança e fiabilidade do sistema de votação electrónica não podem depender exclusivamente da privacidade da rede, visto esta não poder ser assegurada.

Robustez

Ninguém poderá influenciar qualquer parte do sistema de voto. A robustez deverá ser assegurada a diversos níveis, tais como:

- Não permitir que as autoridades atribuam permissões de voto a quem não as tem;
- Não permitir que indivíduos votem sob outra identidade;
- Garantir que urnas e máquinas de contagem de votos sejam fiáveis;
- Garantir que o sistema funciona devidamente durante o processo eleitoral;

- Garantir que todos os eleitores têm acesso ao sistema durante o período do acto eleitoral;
- O processo de voto deve poder ser interrompido e depois retomado pelo eleitor.

A resistência à fraude é normalmente medida como o número de elementos que são necessários para conspirar para introduzir outros votos ou evitar que eleitores votem.
Nem todas as características apontadas são verificadas por todos os protocolos de votação electrónica.
Algumas das propriedades apresentadas são contraditórias e esse é um dos problemas dos sistemas de votação electrónica. Por exemplo, o eleitor deverá poder no final de uma votação verificar que o seu voto foi contado. Essa possibilidade poderá também permitir-lhe provar em quem votou e como tal poderá ser uma porta aberta para a coercibilidade, precisamente por poder associar o eleitor ao voto final.
De seguida apresenta-se o Protocolo REVS e identificam-se as suas propriedades.

2.4 REVS – Robust Electronic Voting System

Este sistema de voto electrónico foi concebido para funcionar em ambientes distribuídos e não tolerantes a falhas, como por exemplo a Internet.
O sistema REVS foi desenhado para ser robusto e assegura muitas das características dos sistemas de voto electrónico referidas no capítulo anterior, mesmo quando ocorrem falhas nas máquinas envolvidas ou nas comunicações.
Como características adicionais de robustez, o REVS assegura que, por exemplo, quando ocorrem falhas nas máquinas que originam interrupções no protocolo este continua a comportar-se correctamente já que outras máquinas assumem a mesma função. Cada eleitor mantém um estado local que lhe permite continuar a votação mais tarde, caso ocorra algum problema. Cada servidor mantém um estado distinto para cada eleitor registado no sistema, que indica o estado da votação do eleitor no sistema e permite que o eleitor receba a mesma resposta de cada servidor. Cada servidor individualmente não pode agir como um eleitor ou dar respostas falsas sem que seja detectado. Por exemplo, se um servidor do tipo Administrador tentar fazer passar-se por um eleitor, tal não será possível já que não basta a sua própria assinatura para que o voto seja válido. Por outro lado, cada Administrador não tem a palavra-chave dos outros

Administradores para poder obter todas as assinaturas necessárias para um voto ser válido.

O sistema de voto electrónico REVS [6, 12] é baseado em *blind signature* [1] [13], permitindo um bom comportamento num ambiente distribuído, como por exemplo a Internet. É baseado no trabalho de DuRette's [2], que melhorou o sistema EVOX de Herschberg [3]. DuRette concebeu uma versão melhorada do EVOX a fim de eliminar entidades que pudessem sozinhas corromper a eleição, como, por exemplo o Administrador. Além disso através da arquitectura de múltiplos servidores para as mesmas funções, o REVS não é sensível a falhas de comunicação entre estes. Melhorou também alguns pontos de autenticação de utilizadores.

Fujioka, Okamoto e Otha propuseram um protocolo de *blind signature* conhecido como FOO [9], que é referência em protocolos deste tipo. Por sua vez, Cranor e Cytron [8] com o Sensus e Herschberg com o EVOX, propuseram os seus protocolos de votação electrónica baseados no FOO. No entanto, nenhum dos dois protocolos controlam o poder de um Administrador. Em 1999, DuRette propôs o protocolo EVOX Managed Administrator. Este é uma evolução do EVOX, reduzindo o poder do Administrador e que o REVS também utiliza introduzindo tolerância a falhas e disponibilidade.

O REVS inclui os seguintes tipos de servidores e a comunicação entre os mesmos é toda efectuada utilizando sessões seguras via SSL (**S**ecure **S**ocket **L**ayer) sendo portanto conhecida a chave pública de todos os servidores que fazem parte do sistema.

Módulo de Voto

É uma aplicação cliente que o eleitor utiliza para votar. Através dela o eleitor pode obter a eleição, o boletim de voto para a mesma e validar ou submeter o seu voto. Esta aplicação é disponibilizada pelo Comissário Eleitoral e assinada pelo mesmo.

Comissário Eleitoral

O supervisor da eleição recebe reclamações de qualquer eleitor ou servidor eleitoral. Poderá promover investigações se houver suspeitas de fraude do acto eleitoral.

---

[1] *Blind signature* é uma assinatura electrónica concebida por David Chaum com a particularidade de o conteúdo a assinar ser disfarçado, através da combinação de um factor aleatório com a mensagem, antes de a assinatura ser efectuada. A mensagem é assim ocultada para ser enviada para assinatura, é assinada e devolvida ao originador que pode voltar a obter a mensagem original. Quem assina não tem acesso ao conteúdo da mensagem que assina permitindo neste caso a privacidade da escolha de voto do eleitor.

É também responsável por preparar a eleição, gerar e manter as chaves privadas da eleição, responder a questões e definir a configuração operacional da eleição. Esta é constituída, por exemplo, pelos endereços e chaves públicas dos servidores do tipo Administrador, número de assinaturas requeridas para um voto ser válido, etc.

Distribuidor de Votos

Responsável pela distribuição dos elementos da eleição. Toda a informação disseminada pelo distribuidor de votos deve ser assinada pelo Comissário Eleitoral em quem todos os eleitores confiam. Este módulo é bastante exigente em termos de troca de informação e poderá ser replicado em vários servidores para garantir tempos de resposta adequados durante o acto eleitoral.

Administrador

Tem a responsabilidade de validar o boletim de voto de um eleitor. Para ser válido e contado na contagem final, o voto tem que estar assinado por um conjunto de Administradores distinto e superior a metade do número total de Administradores. Deste modo um eleitor não pode ter dois votos válidos. Um eleitor utiliza um conjunto Utilizador/Palavra-Chave diferente para cada Administrador e contacta o servidor. O voto que é assinado pelo Administrador está ocultado através de “blind signature” pelo que o Administrador não pode ver o voto do Eleitor, sendo assegurada a privacidade.

Anonimizador

Permite que o eleitor fique anónimo no acto eleitoral. O eleitor pode escolher um ou mais nós Anonimizadores e enviar o seu voto através dos mesmos. O Anonimizador retira a identificação da máquina do eleitor e provoca atrasos aleatórios antes de enviar o voto para o Totalizador. Desta forma, o Totalizador não pode assim associar o voto ao eleitor. Foi encontrada ainda uma proposta de arquitectura, definida em [14], para os anonimizadores do REVS. Essa proposta é uma alternativa à original, na medida em que o eleitor pode receber uma confirmação de submissão do voto, o que não acontece na arquitectura original. Neste caso, deixaria de haver anonimizadores e existiriam apenas totalizadores. No entanto devido à maior complexidade, não foi esta a solução escolhida para modelação.

Totalizador

Verifica a validade dos votos recebidos através da confirmação da presença de todas as assinaturas necessárias, retira duplicações caso existam e conta os votos válidos. As duplicações podem ocorrer porque o próprio protocolo não permite a confirmação de recepção do voto, permite que um mesmo eleitor submeta diversas vezes o mesmo voto para anonimizadores diferentes ou até para o mesmo. Os totalizadores apenas podem analisar os votos no final do acto eleitoral, após o Comissário eleitoral publicar a chave privada da eleição.

Os eleitores enviam os seus votos finais para os contadores através dos anonimizadores, encriptados com a chave pública da eleição, não permitindo que os anonimizadores e totalizadores tenham acesso aos votos durante a votação. Só no final da eleição e quando a chave privada é publicada conseguem ter acesso aos mesmos.

O REVS permite vários anonimizadores e vários contadores, resultando uma arquitectura com *no single point of failure*[2].

Como os eleitores podem enviar os votos através de vários anonimizadores, os totalizadores terão que eliminar os votos duplicados antes de efectuar a contagem final.

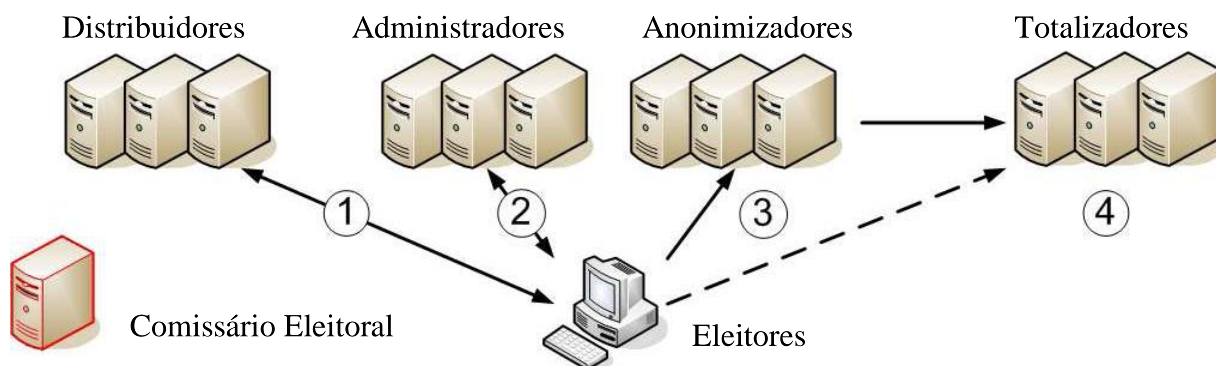

Figura 2.2 Arquitectura REVS

Este protocolo pressupõe que os eleitores já se encontram registados para o acto eleitoral, função que é assegurada pelo Comissário Eleitoral. Segue as etapas apresentadas a seguir, as três primeiras referentes ao ponto de vista do eleitor e a última relativa ao processo de votação em geral [6, 12].

[2] no single point of failure significa que o sistema é tolerante a falhas e que se uma parte do sistema falha, há outra que assegura a mesma função.

1. **Distribuição de boletins de voto**
    a. O Módulo do Eleitor contacta um módulo Distribuidor de Voto para obter as eleições em que poderá participar.
    b. O Distribuidor de Voto devolve a lista de eleições em que o eleitor pode participar.
    c. De seguida, o Módulo do Eleitor contacta um módulo Distribuidor de Voto para obter uma eleição para participar.
    d. O Distribuidor de Voto devolve o boletim de voto, a chave pública da eleição e outros elementos específicos da eleição, tais como identificação dos servidores de chaves e respectivas localizações.

As informações anteriores devem ser validadas e assinadas pelo Comissário Eleitoral. A própria aplicação cliente que o eleitor utiliza foi assinada pelo Comissário Eleitoral com a sua chave privada sendo a chave pública conhecida de todos os eleitores.

2. **Assinatura do voto**

Após o Eleitor expressar a sua intenção de voto, o Módulo de Voto confirma o voto com uma *random bit string*[3] através de um *bit commitment*[4]. O voto é anonimizado com um *random blind factor*[5] e enviado para o Administrador, para assinatura. O Módulo de voto guarda a selecção do voto, a *random bit string* e o *random blind factor*. A mensagem é enviada do Módulo de Voto para um número de Administradores superior a metade do total de Administradores. A mensagem é autenticada com o código de Utilizador/Palavra-Chave para cada administrador de modo a garantir a autenticação do eleitor. Cada Administrador, ao receber um pedido autenticado de assinatura, verifica se já assinou algum pedido para o eleitor em questão. Se não assinou, assina o voto com a sua chave privada e guarda essa assinatura, Se já assinou, retorna ao eleitor a informação prévia já assinada, ou seja, o voto do eleitor assinado com a chave do Administrador.

Após recepção da assinatura, o Módulo de Voto remove o *random blind factor* e verifica a sua correcção utilizando para tal a chave pública do Administrador.

---

[3] R*andom bit string* é um número aleatório utilizado para cifrar uma mensagem utilizando um algoritmo de cifração.

[4] B*it commitment* é um método que permite utilizar um valor mantendo-o invisível e inacessível para terceiros.

[5] R*andom blind factor* é um número aleatório utilizado para ocultar uma mensagem num sistema *blind signature*

O processo é repetido até que seja atingido um número *t* de assinaturas de Administradores. O número *t* deve sempre ser superior ou igual a *N/2* onde *N* é o número total de Administradores. Desta forma evita-se que um eleitor tenha mais do que um voto válido.
Os administradores recebem um voto que foi anonimizado pelo que não conseguem saber o conteúdo do mesmo, logo a privacidade é assegurada. Por outro lado se o eleitor repetir a operação, ele receberá a assinatura que já tinha sido devolvida na primeira vez que tinha requisitado a assinatura ao Administrador em questão.

3. Submissão do voto

O Módulo de Voto junta o voto encriptado com o *bit commitment*, as assinaturas recebidas dos administradores e o próprio *bit commitment* que permitirá ter acesso ao voto no final do escrutínio e encripta tudo com a chave pública da eleição. Depois envia tudo para o Totalizador, através do Anonimizador. O eleitor pode enviar o seu voto para vários Totalizadores e tantas vezes quantas ache necessário para que o seu voto seja contado. O Anonimizador não tem acesso ao pacote que contém o voto encriptado, o *bit commitment* e as assinaturas dos administradores, porque o pacote vai encriptado com a chave pública da eleição. A chave privada da eleição que permitiria acesso a essa informação só será disponibilizada no final da eleição.

4. Cálculo final da votação

Após o término da eleição, os Totalizadores fazem a contagem de todos os pacotes submetidos na votação e publicam os resultados. A contagem dos votos é efectuada da seguinte forma:

- Decifra cada pacote recebido com a chave pública da eleição;
- Verifica que cada pacote tem as assinaturas requeridas pelos administradores;
- Remove os votos repetidos, ou seja os que têm o mesmo *bit commitment*;
- Conta os votos que passaram os passos anteriores;
- Se existirem vários Totalizadores, um deles é o *master* que vai juntar os valores determinados por cada totalizador, retirar as repetições e calcular o resultado final da eleição.

Os Totalizadores devem publicar os pacotes submetidos e recebidos, enquanto os Administradores publicam os boletins anonimizados. Deste modo poderão ser verificadas se as assinaturas dos votos são ou não válidas e se coincidem com as que foram efectuadas pelos respectivos administradores. Após isto, o eleitor com a informação de voto que submeteu pode verificar se o seu voto foi contado ou não utilizando para isso o *bit commitment*. Se verificar que o seu voto não consta na lista de votos contados, pode reclamar junto das entidades eleitorais.

O Comissário eleitoral é quem publica a chave privada da eleição e qualquer pessoa pode validar a contagem dos votos, ou seja desencriptar os pacotes submetidos utilizando a chave pública da eleição, descartar eventuais repetições, validar as assinaturas dos votos restantes e efectuar a contagem oficial com os votos válidos. O eleitor pode voltar a submeter o seu voto se o mesmo foi perdido.

3 Verificação

A verificação formal é o acto de provar a correcção de um sistema, de acordo com uma especificação formal ou propriedade, utilizando métodos formais. Estes podem ser linguagens matemáticas, técnicas e ferramentas para especificar e verificar esses sistemas. Apesar de a utilização de métodos formais não garantir a correcção de um sistema, poderá permitir identificar incorrecções, ambiguidades e identificação incompleta de requisitos [15].

Ferramentas de verificação

Através das ferramentas de verificação pode concluir-se a consistência lógica das regras de um protocolo e aferir a correcção dos requisitos. Estas utilizam técnicas de verificação de modelos que exploram todos os possíveis estados do modelo. Deste modo poderá ser provado que o mesmo satisfaz as propriedades especificadas [16].

Como as sequências de eventos nem sempre são previsíveis em ambientes distribuídos, as ferramentas de verificação de protocolos permitem percorrer todos os estados de um modelo, de uma forma exaustiva e tirar conclusões a partir daí. No entanto, para que tal seja possível, é necessário que seja efectuada uma abstracção focada apenas no essencial que se pretende verificar. Caso isso não aconteça, facilmente o espaço disponível de memória poderá ser esgotado pela verificação do modelo, na geração de todos os estados possíveis do mesmo.

Como o foco destas ferramentas é nas regras e procedimentos, as especificações utilizadas são apenas representações parciais dos protocolos. Um modelo de validação corresponde a uma abstracção de uma implementação e representa apenas as interacções entre processos num sistema distribuído. O fundamental desta validação é garantir que o protocolo representa um conjunto completo e consistente de regras que controlam os processos neste tipo de sistema [17].

Verificação de protocolos

Foram encontrados no decorrer desta investigação alguns trabalhos realizados na área de verificação de protocolos de votação electrónica. Uns utilizando ferramentas de verificação baseadas em modelos tais como *PaMoChSA* em [18], onde são efectuadas as verificações respectivamente da não possibilidade de introdução de votos não válidos baseados na abstenção. Outros trabalhos referem as propriedades dos protocolos de votação electrónica mas utilizam linguagens formais tais como *event-B* em [19] onde é analisado o armazenamento dos votos num sistema de votos electrónico, *applied pi-calculus* em [20] onde são analisadas propriedades de privacidade destes sistemas, em [21] onde são verificadas propriedades Justiça, Elegibilidade, e Privacidade e em [22] onde é verificada a Resistência à Coação e Ausência de Recibo de Voto. Estes trabalhos são importantes do ponto de vista da análise que é efectuada às propriedades estudadas e que complementa as características de cada uma delas e o que deverá ser esperado delas quando se verifica um modelo de um protocolo de votação electrónica.

Por outro lado existem alguns trabalhos que utilizam ferramentas de verificação baseadas em modelos com o SPIN, com o UPPAAL ou com ambos, sendo neste último caso efectuada uma comparação entre alguns aspectos de ambas as ferramentas. No entanto são trabalhos realizados noutras áreas que não protocolos de votação electrónica. Destacam-se, por exemplo, [23] onde é efectuada a simulação e verificação do protocolo "Collision Avoidance Protocol" com o SPIN e UPPAAL; [24] onde é utilizada apenas a ferramenta SPIN para verificação do protocolo "NetBill Protocol"; [25, 26] onde é utilizado o UPPAAL para verificação respectivamente do protocolo "Zeroconf" e "Distributed real-time network protocol RTnet"; [27] onde é verificado com UPPAAL o desenho de um sistema distribuído de elevadores. Além dos trabalhos aqui referidos nomeadamente para as ferramentas SPIN e UPPALL, existem muitos outros não referidos aqui.

Os trabalhos indicados revelaram-se importantes na medida em que exemplificam ou comparam as ferramentas escolhidas para verificação no decorrer deste trabalho.

A tabela seguinte resume a documentação e respectivas áreas de trabalho, enquadrando também o presente trabalho que está assinalado na última linha.

Nome	Tipo	Área de verificação	Propriedades verificadas	Referência
PaMoChSA	Ferramenta	Votação electrónica	Impossibilidade de inclusão de votos não válidos baseados em abstenção.	[18]
event-B	Linguagem Formal	Votação electrónica	Armazenamento de votos num sistema de votação electrónica	[19]
pi-calculus	Linguagem Formal	Votação electrónica	Privacidade	[20]
pi-calculus	Linguagem Formal	Votação electrónica	Justiça, Elegibilidade, e Privacidade	[21]
pi-calculus	Linguagem Formal	Votação electrónica	Resistência à Coação e Ausência de Recibo de Voto	[22]
SPIN e UPPAAL	Ferramenta	Outros protocolos	Propriedades do protocolo "Collision Avoidance Protocol"	[23]
SPIN	Ferramenta	Outros protocolos	Propriedades do protocolo "NetBill Protocol"	[24]
UPPAAL	Ferramenta	Outros protocolos	Propriedades do protocolo "Zeroconf"	[25]
UPPAAL	Ferramenta	Outros protocolos	Propriedades do protocolo "Distributed real-time network protocol RTnet"	[26]
UPPAAL	Ferramenta	Outros protocolos	Verificação de uma solução de um sistema de elevadores de veículos.	[27]
SPIN e UPPAAL	Ferramenta	Votação electrónica	Propriedades genéricas de modelos e de votação electrónica, estas últimas referente ao protocolo REVS.	Documento corrente

Tabela 3-1 Resumo de trabalhos em verificação

SPIN E UPPAAL

O SPIN e o UPPAAL são duas ferramentas de verificação de modelos. Uma das principais diferenças entre elas é o facto de o UPPAAL incluir uma noção de tempo, o

que não é suportado pelo SPIN. O SPIN suporta apenas uma relação temporal entre os eventos e pode saber-se que um ocorre a seguir a outro. Por outro lado, o UPPAAL permite por exemplo saber que existe um tempo específico entre dois eventos.

Ambas as linguagens apresentam a possibilidade de desenhar modelos, efectuar simulações e verificações.

Em termos de utilização as ferramentas apresentam capacidades distintas. O SPIN é menos visual do que o UPPAAL. Com o SPIN cria-se uma especificação de um modelo em modo texto e só com a simulação ou verificação é possível ter uma representação gráfica. Por outro lado, com o UPPAAL pode definir-se o modelo do sistema de um modo gráfico.

A representação gráfica facilita a visualização das interacções entre os elementos do modelo e permite efectuar simulações do modelo acompanhando as transições também de modo gráfico.

Os modelos descrevem os comportamentos do sistema a analisar. Depois, para efectuar uma verificação, as propriedades são descritas de forma rigorosa usando lógica temporal. Em seguida, de forma automatizada é verificado para todos os estados do modelo se a propriedade a verificar é válida ou não. A verificação de uma propriedade tem três possíveis resultados: válida, não válida ou sem resultados.

Se a propriedade é válida, pode partir-se para outras verificações baseadas nessa propriedade. Se a propriedade verificada não for válida, então deveremos analisar o problema que poderá ter várias causas: o modelo não reflecte o sistema, existe um erro de desenho do sistema modelado ou existe um erro da especificação da propriedade que estamos a verificar. Em qualquer dos casos deve ser identificado o problema. Caso seja alterado, deverá ser revalidado se as verificações que já tinham sido efectuadas previamente se mantêm válidas. O modelo pode ter que ser corrigido. Se o modelo é demasiado complexo, e não há memória disponível para efectuar a verificação, então há duas alternativas para ultrapassar esse problema. Uma alternativa é simplificar o modelo, abstraindo detalhes do sistema. Outra alternativa é usar outros algoritmos disponíveis nas ferramentas e que permitem efectuar algumas optimizações nas verificações, por exemplo limitando o tipo de verificação que se vai fazer, consoante a propriedade. Exemplos mais concretos serão utilizados no decorrer do trabalho.

Linear Temporal Logic e Computar Tree Logic

Na verificação de modelos é utilizada uma linguagem de formalização para especificar propriedades. O SPIN e o UPPAAL disponibilizam uma linguagem de especificação de propriedades. A verificação dessas propriedades em SPIN é especificada utilizando Linear Temporal Logic (LTL) e no UPPAAL utilizando Computar Tree Logic (CTL) [16].

A lógica temporal é baseada na lógica proposicional, mas alargada para modalidades temporais que permitem avaliar o comportamento de sistemas. Através da lógica temporal é possível definir, por exemplo, que alguma coisa acontecerá eventualmente no futuro. A natureza das lógicas temporais pode ser *linear* ou *ramificada*. A linear é a LTL, utilizada no SPIN e a ramificada é a CTL que é utilizada no UPPAAL. O que as distingue, de uma forma mais generalizada, é que na linear em cada momento existe apenas um sucessor no tempo mas no CTL existe um ramo em que o tempo pode dividir-se em vários caminhos alternativos. A interpretação LTL é baseada em caminhos, ou seja sequências de estados que vão do presente até ao futuro. Por sua vez os caminhos podem ser ramificações que geram alternativas de caminhos. A CTL é baseada em ramos e apenas baseados em estados do futuro. Enquanto com o LTL se parte de um estado inicial e se verifica o que pretendemos que aconteça no futuro, ou seja, refere-se ao caminho, no CTL refere-se aos estados de um ramo algures no futuro. Com CTL podem-se verificar propriedades que não são possíveis de verificar com LTL e vice-versa.

Um exemplo de propriedades verificáveis em CTL e não verificáveis em LTL são as propriedades relativas a reinicialização, ou seja, para todos os estados existe pelo menos uma sequência que permite voltar ao estado inicial. Por outro lado em LTL pode-se ter *justiça (fairness)* na verificação, ou seja saber se qualquer execução cíclica percorre, ou não determinados estados infinitamente.

Existem vantagens e desvantagens nas utilizações destes tipos de lógicas. Mas tipicamente não permitem verificar o mesmo tipo de propriedades. Existem, no entanto, propriedades que podem ser verificadas por ambas as lógicas.

A definição de um protocolo inclui os seguintes elementos:

1. Serviço disponibilizado pelo protocolo (o serviço de voto electrónico).
2. Ambiente em que o protocolo opera, no caso do REVS a Internet.

3. Vocabulário das mensagens utilizadas no protocolo.
4. Método de codificação das mensagens.
5. As regras para a troca de mensagens.

Neste trabalho, na modelação vamos focar-nos no aspecto descrito no ponto 5, abstraindo-nos dos restantes aspectos. Desta forma, os modelos SPIN e UPPAAL irão descrever parcialmente o protocolo REVS. Essa descrição parcial do protocolo define as interacções entre os vários processos do sistema de votação electrónica. As razões que se prendem com esta abstracção estão relacionadas com uma das limitações das ferramentas de verificação de modelos que é o facto de necessitarem de muita memória para representar os estados possíveis dos modelos.
O computador utilizado neste trabalho tem 3GB de memória física tendo que ser controlado o número de processos de modo a poder efectuar verificações sem problemas. No entanto as reduções não foram em termos dos processos principais mas sim em termos do número de instâncias de cada processo para conseguir obter resultados satisfatórios.
A verificação será depois efectuada sobre a modelação indicada.

Análise dos REVS

Nesta secção faremos uma análise do protocolo REVS tendo já como ponto de vista a implementação do modelo em SPIN e UPPAAL. Detalhar-se-á para cada uma das ferramentas o necessário que apenas seja referente a cada uma delas.
Analisando o protocolo REVS, consideram-se as trocas de mensagens tal como é a seguir indicado:

1 - Mensagens relativas à distribuição de boletins de eleição e de voto entre o Módulo de Voto e o Distribuidor de Voto

PedidoBoletimEleicao
EnvioBoletimEleicao
PedidoBoletimVoto
EnvioBoletimVoto

Nestas mensagens o eleitor é identificado perante o Distribuidor de Voto. É assumido que o eleitor já está previamente registado no sistema. O eleitor envia os seus pedidos para um dos Distribuidores de Voto disponíveis no sistema.

2 – Mensagens relativas ao pedido e envio de assinatura de voto entre o Módulo de Voto e os Administradores

PedidoAssinaturaVoto a cada Administrador
EnvioAssinaturaVoto de cada Administrador

Para cada Administrador o eleitor contacta administradores diferentes até ter um número de assinaturas suficientes para poder submeter o seu voto e o mesmo ser válido em termos do número de assinaturas.

3 – Mensagens relativas à submissão de voto entre o Módulo de Voto e o Anonimizador
EnvioVotoAnonimizador
EnvioVotoTotalizador

Apenas a mensagem enviada ao Anonimizador é enviada pelo eleitor. A mensagem enviada ao Totalizador é enviada pelo Anonimizador, após mascarar a mensagem recebida e com eventual atraso, para que a ordem da submissão do voto não seja a mesma da recepção no Anonimizador.

4 – Mensagens entre o Módulo de Voto e o Totalizador para a contagem e validação de voto

PedidoConsultaVoto
EnvioConsultaVoto
PedidoRe-submissãoVoto

Optou-se por não representar no modelo a contagem e validação de voto, isto porque não traz nenhum valor acrescentado. Se necessitarmos de verificar, por exemplo, que o voto foi contado no escrutínio final, é possível fazê-lo com a informação disponível no modelo actual. Em relação à possibilidade de voltar a submeter um voto, num cenário

onde ela ocorresse, significaria que o voto seria então contado posteriormente, à semelhança do que já acontece no modelo proposto.

As mensagens indicadas são trocadas entre os processos do sistema de votação. Consoante as mensagens enviadas ou recebidas por cada processo este vai alterando o seu conjunto de estados, sendo importante a ordem em que as mensagens são enviadas, recebidas ou tratadas por cada processo. No decorrer do trabalho, obtiveram-se frequentemente situações de *deadlock* nas simulações porque mensagens esperadas ou não eram recebidas ou eram recebidas na ordem incorrecta fazendo com que os processos não prosseguissem a execução esperada.

Após o estudo prévio da ferramenta e do protocolo realizado na primeira fase deste trabalho, definiu-se nesta fase um modelo mais alargado em que o Módulo de Voto permite simular diversos eleitores, vários processos Distribuidor, Administrador, Anonimizador e Totalizador. Não foi modelada a fase inicial de pré-registo dos eleitores, assumindo-se que estes já se encontrariam registados. Não foi modelada a fase final em que o eleitor verifica que o seu voto foi contado no escrutínio final. De qualquer forma, existe informação no modelo que permitirá tirar algumas conclusões sobre este assunto.

De uma forma genérica teremos:

- Cada eleitor pede o boletim de eleição a um dos Distribuidores;
- Cada um dos Distribuidores envia o boletim de eleição para o eleitor que o solicitou;
- Cada eleitor pede o boletim de voto ao Distribuidor que contactou inicialmente;
- Cada um dos distribuidores envia o boletim de voto para o eleitor que o solicitou;
- Cada eleitor pede as diversas assinaturas de voto aos Administradores disponíveis até obter um número necessário de assinaturas para tornar o seu voto válido quando recebido;
- Cada Administrador envia a assinatura de voto a cada eleitor que a solicite;
- Uma vez recebidas todas as assinaturas necessárias, o eleitor submete o seu voto ao Anonimizador;

- O Anonimizador atrasa o voto recebido, oculta-o e reenvia-o para o Totalizador;
- Após o término da eleição é calculado o escrutínio.

3.1 SPIN

SPIN é uma ferramenta que suporta a verificação de sistemas distribuídos desenvolvida pelos laboratórios Bell, no grupo de métodos formais de verificação. Ou seja, permite verificar a correcção das interacções entre sistemas distribuídos, nomeadamente em ambientes complexos com uma multiplicidade de estados e transições possíveis. O SPIN está focado na comunicação entre os processos e nos problemas de sincronização entre processos de um sistema distribuído [4, 17].

Os sistemas distribuídos são normalmente de grandes dimensões e comportam uma grande diversidade de informação que muitas vezes não é necessária para a verificação formal. O que é essencial na verificação é a comunicação entre processos e o acesso concorrente aos recursos. Assim, a primeira etapa para a verificação formal de um protocolo é desenvolver uma especificação que retire o que é supérfluo. O modelo do sistema deve focar apenas os aspectos relevantes para a verificação.

Após a especificação do modelo, o *SPIN* permite verificar a sua consistência lógica, a existência de *livelocks*[6], *deadlocks*[7], estados não esperados e pontos de controlo incompletos que podem resultar em terminações abruptas que implicam que o protocolo não cumpra aquilo a que se propõe.

O SPIN suporta uma linguagem de alto nível para especificar sistemas, denominada PROMELA (**PRO**TOCOL/**PRO**CESS **ME**TA **LA**NGUAGE). PROMELA é uma linguagem não determinística. Possui primitivas para especificar comunicação assíncrona de mensagens com um número de parâmetros variável, utilizando um *buffer*. Por outro lado, permite também a especificação de mensagens síncronas.

Utilizando o SPIN, especifica-se um modelo do sistema e os requisitos de correcção para essa especificação, tendo em atenção dois princípios básicos:

- O modelo é fechado
- O modelo é de estado finito

---

[6] *Live lock* acontece quando um determinado processo não progride, apesar da alteração do seus estados. Pode ser consequência de uma recuperação de *dead lock* se dois processos retomarem em simultâneo.

[7] *Deadlock* acontece quando duas ou mais acções concorrentes esperam cada uma que a outra termine, ficando ambas indefinidamente à espera. No caso da verificação significa que não existe um processo que satisfaça as condições para passar ao próximo estado.

Um sistema é fechado se nenhuma parte do mesmo fica por definir, ou seja, se tudo o que é necessário estiver definido.

Um sistema é de estado finito se tem um número finito de estados potenciais. Um estado de um sistema é o conjunto de todos os seus valores e pontos de controlo. Se isto acontecer, todos os estados poderão ser percorridos e verificados em conformidade com a especificação.

Uma das vantagens do SPIN é o facto de a ferramenta verificar automaticamente propriedades do sistema. Isto faz com que seja assegurado que tudo o que foi especificado é satisfeito pelo modelo. Esta técnica de validação por modelos é muito vantajosa uma vez que faz uma validação exaustiva de todos os estados, o que é praticamente impossível noutras situações de *debug*. Apresenta a desvantagem de ser necessária uma boa especificação, com um bom nível de abstracção, pois caso contrário facilmente se tem problemas de memória ao efectuar uma verificação. O SPIN efectua a verificação de propriedades baseado na lógica temporal LTL (ver secção 3).

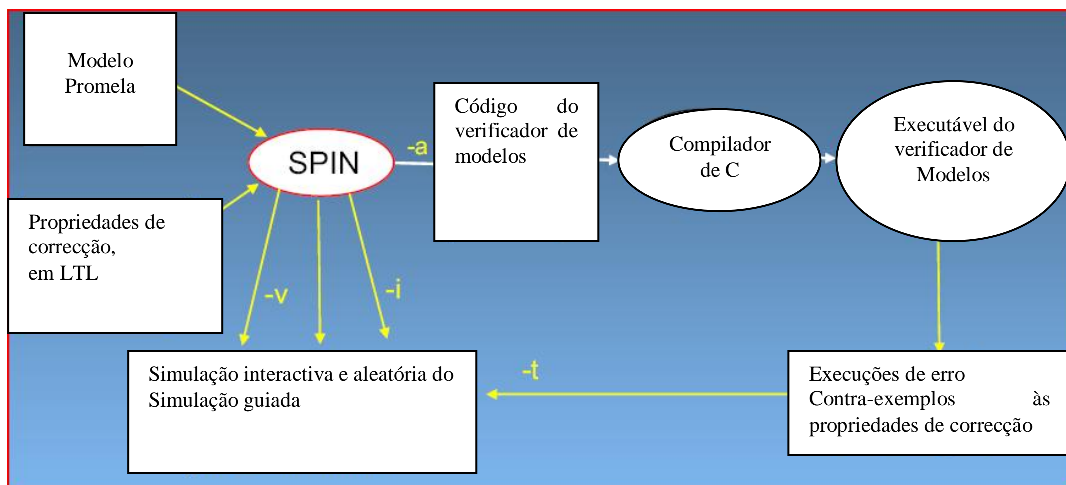

Figura 3.1 A verificação de propriedades em SPIN

Quando é efectuada uma verificação de uma propriedade que não é válida, permite guardar essa execução para utilizar, por exemplo, em modo de *debug* e permite corrigir/validar o modelo/propriedade.

Na criação do modelo de um sistema, representa-se o seu comportamento num ambiente distribuído com foco no desenho do mesmo. Ao modelo definido designamos modelo de validação. De forma a poder efectuar a validação de um modelo é necessário poder

dizer com precisão o que significa um desenho estar correcto. Alguns dos critérios a que um modelo correcto deve atender são:

- ausência de *deadlocks*
- ausência de *livelocks*
- ausência de terminações abruptas

No entanto estas características apesar de serem importantes não são suficientes para provar que o modelo está correcto.

Há outras propriedades inerentes aos protocolos que deverão ser alvo de prova. No caso deste trabalho serão as propriedades referentes aos sistemas de votação electrónica.

No entanto, a fim de ser possível verificá-las será necessário reduzir a complexidade do modelo focando a sua especificação no que pretendemos verificar do protocolo. Para tal utiliza-se uma linguagem para especificar os requisitos de correcção que permita analisar o modelo.

De forma a reduzir a complexidade do modelo devemos escolher cuidadosamente o conjunto de critérios de correcção que se pretende exprimir. O método para essa escolha não abrange todas as situações possíveis. Vários níveis de complexidade são suportados.

A verificação de *deadlocks* é computacionalmente mais simples e existem diversos algoritmos que facilmente detectam esta propriedade nos sistemas.

Outros tipos de requisitos, tais como ausência de *livelocks* são especificados de outra forma e têm um peso computacional mais elevado. Quanto mais sofisticados forem os requisitos a verificar, mais recursos utilizará a verificação automática.

Em Promela podemos especificar as propriedades referidas acima. Para verificar um modelo, devemos especificar critérios de correcção, como *afirmação* (*claim)* sobre o seu comportamento.

Podemos ter dois tipos de afirmações: *inevitável* ou *impossível*.

Como o número de comportamentos possíveis de um modelo Promela é finito, uma *afirmação* de qualquer tipo define uma afirmação complementar e equivalente à primeira.

Assim, para indicar que determinado comportamento é inevitável, basta indicarmos que todos os comportamentos diferentes desse são impossíveis. Similarmente, se uma asserção de correcção diz que uma condição é invariavelmente verdadeira, a afirmação de correcção indicará que é impossível que essa asserção seja violada, independentemente do comportamento do sistema.

O comportamento de um modelo é definido como o conjunto de todas as sequências de execução do mesmo. Uma sequência de execução é um conjunto finito de estados. Um estado é definido pela especificação de todas as variáveis globais e locais e todos os pontos de controlo de um processo, bem como ainda o conteúdo de todos os canais de mensagens. Um modelo pode ser colocado num determinado estado através da atribuição de valores a variáveis, pontos de controlo e fluxo de canais.

Um conjunto de estados em Promela é válido se satisfaz duas condições:

- O primeiro estado da sequência, isto é, o estado de ordem 1, é o estado inicial do sistema M com todas as variáveis inicializadas a zero, todos os canais de mensagens vazios, apenas com o processo *init* activo e no seu estado inicial.
- Se M é colocado no estado com ordem i, então existe pelo menos uma sequência de execução que o transforma no estado com número ordinal $i+1$.

Existem dois tipos de sequências de execução:

- Sequências de terminação que ocorrem quando um estado não se verifica mais do que uma vez numa sequência de execução. O modelo M não contém instruções executáveis quando colocado no último estado da sequência.
- Sequência cíclica em que todos os estados, excepto o último, são distintos e o último estado da sequência é igual a um dos estados anteriores.

Execuções cíclicas definem potencialmente execuções infinitas. Todas as sequências de terminação e cíclicas podem ser geradas executando um modelo Promela. Ao conjunto de todos os estados incluídos no comportamento do modelo chama-se conjunto dos estados atingíveis do modelo.
As afirmações de correcção em Promela podem ser efectuadas sobre proposições simples onde a proposição é uma condição booleana num sistema. A proposição pode referir-se a todos os estados das variáveis locais, globais, conteúdos de canais de mensagens e pontos de controlo de fluxo. As proposições definem implicitamente uma rotulagem dos estados do modelo. Num determinado estado uma proposição será verdadeira ou falsa. Os critérios de correcção poderão ser definidos em termos de estados, por exemplo especificando em que passo uma proposição deverá aguardar. Alguns desses requisitos podem ser definidos dentro de código Promela, por exemplo,

com *asserções* (*assert*). No entanto, se é necessário mais do que uma proposição, poderão ser necessários critérios de ordem indicando, por exemplo, quais as propriedades que devem aguardar para obter a veracidade de uma outra.

Afirmações temporais

Nesta formalização especifica-se uma ordem das proposições. Numa afirmação temporal, uma ordem sequencial de duas proposições indica uma sequência imediata.

Os tipos de requisitos de correcção que são efectuados são diferentes para sequências de terminação e sequências cíclicas. Um requisito importante para as sequências de terminação é a ausência de *deadlock*. Nem todas as sequências de terminação correspondem no entanto a *deadlocks*. Teremos então de indicar quais as propriedades dos estados finais de modo a indicar os que são considerados *deadlocks*. Para as sequências cíclicas devemos indicar condições relativas a ausência de *livelocks*.

O SPIN permite verificar os seguintes tipos de propriedades:

Invariância

[] p

Durante uma execução, todos os estados satisfazem p.

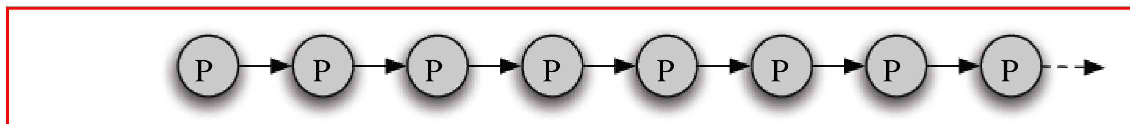

Figura 3.2 Invariância

Resposta

p -> <> q

Todo o estado que satisfaz p será seguido eventualmente por um que satisfaz *q*.

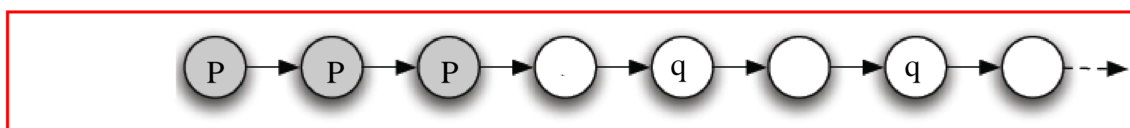

Figura 3.3 Resposta

Precedência

p -> (q U r)

Todo o estado que satisfaz *p* é seguido por uma sequência de estados que satisfaz *q* e essa sequência é terminada por um estado que satisfaz *r*.

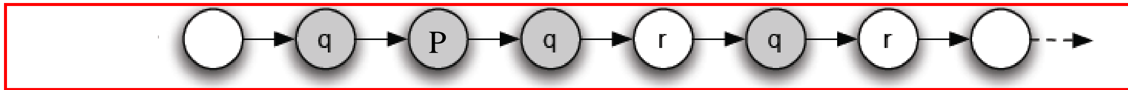

Figura 3.4 Precedência

Objectivo

p -> <> (q || r)

Toda a sequência que satisfaz p é precedida por uma sequência de estados que satisfaz q ou r.

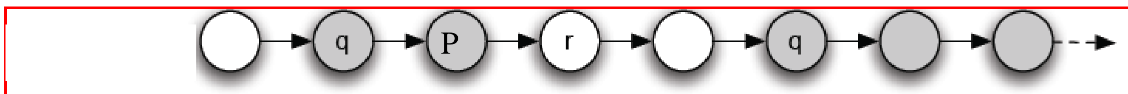

Figura 3.5 Objectivo

Deadlock

O modelo do sistema está num estado global onde nenhuma transição é possível.

Afirmações violadas

Apresenta a possibilidade de fazer afirmações sobre o modelo e verificar se são verdadeiras ou falsas. Em SPIN é utilizado o *assert*.

Código não executado

Apresenta a possibilidade de verificação que permite validar se existe código não atingido ou seja código que nunca é executado.

A Secção 4 detalha a modelação do protocolo REVS em SPIN, onde serão validadas as propriedades básicas de um modelo e os princípios atrás descritos, bem como propriedades específicas inerentes aos sistemas de votação electrónica.

3.2 UPPAAL

É uma ferramenta integrada para modelação, validação e verificação de sistemas em tempo real. É desenvolvida através de cooperação entre o Departamento de Tecnologias da Informação da Universidade de **UPP**SALA, na Suécia, e o Departamento de Ciências da Computação da Universidade de **AAL**BORG, na Dinamarca.

O modelo de um sistema é definido como um autómato, que utiliza tipos de dados determinísticos e comunica através de estruturas de dados partilhadas e canais de comunicação. A ferramenta UPPAAL é adequada para sistemas em tempo real ou para protocolos de comunicação, ou seja, aqueles em que o momento em que os eventos ocorrem é relevante [5, 28, 29].

Nos sistemas em tempo real, a sua correcção depende não só da ordem de execução mas também do momento de execução. A importância de um serviço ser disponibilizado no momento certo é em muitos dos sistemas de uma relevância extrema. Os sistemas de votação electrónica não são excepção; por exemplo, é muito importante que as fases do protocolo se interliguem adequadamente, de forma a garantir um sistema de votação robusto e fiável.

O UPPAAL inclui uma linguagem de modelação, um simulador e um verificador automático de modelos. Através da linguagem pode desenhar-se o modelo do sistema. O simulador permite a análise das diversas execuções dinâmicas do sistema modelado. O verificador de modelos permite uma verificação exaustiva e dinâmica do comportamento do sistema. No entanto, para que isso seja possível, as especificações deverão ser correctamente formuladas numa linguagem que seja reconhecida pela ferramenta. No caso do UPPAAL é utilizada uma versão simplificada do CTL (ver secção 3).

O UPPAAL utiliza uma arquitectura cliente-servidor subdividindo-se numa interface gráfica para modelação e simulação de sistemas e num sistema de verificação automática.

Um modelo de um sistema em UPPAAL é constituído por uma rede de processos descrita como autómatos temporais. A descrição do modelo é efectuada em três partes, através de um editor:

- As declarações globais e locais;
- Os *templates* dos autómatos;
- A definição do sistema.

O simulador é uma ferramenta que possibilita a validação de todas as execuções dinâmicas durante a modelação. Poderá também ser utilizado para visualizar pormenores de execuções do verificador. Por exemplo ao verificar uma propriedade que falha, poderá ser guardada a execução da simulação que origina essa falha de propriedade a assim pode-se efectuar o *debug* e corrigir o modelo.

O UPPAAL disponibiliza ainda um editor onde se pode fazer a especificação de requisitos de propriedades para utilizar no verificador de modelos.

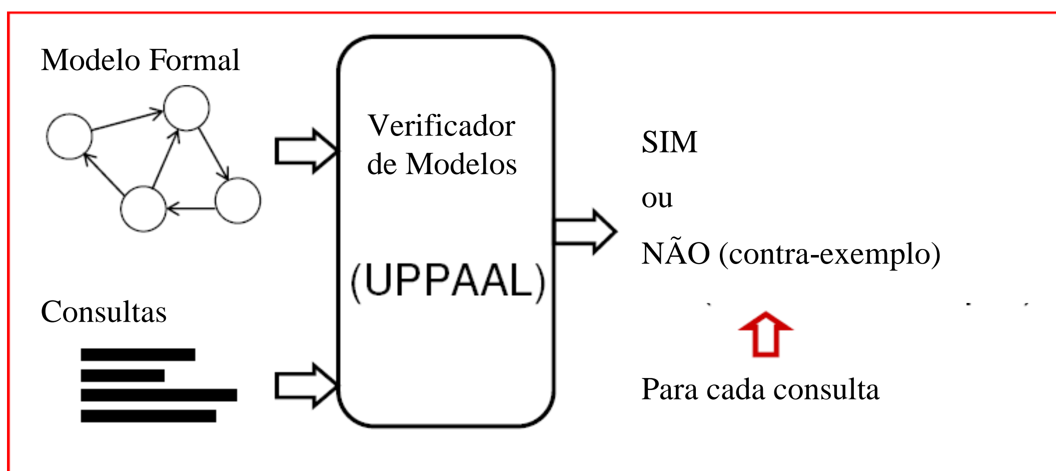

Figura 3.4 Verificação de propriedades em UPPAAL

Após a criação do modelo um UPPAAL é necessário indicar como verificar as propriedades pretendidas.

O UPPAAL permite verificar quatro tipos de propriedades:

Acessibilidade

Uma determinada condição é verificada em algum ponto do modelo. Graficamente poderemos representar tal como na figura seguinte:

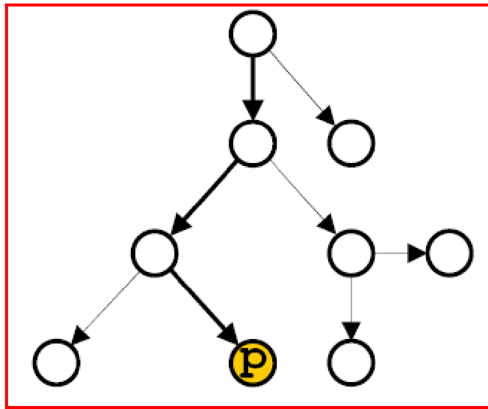

Figura 3.5 E<> p – Existe eventualmente p

Existe um caminho de execução em que *p* ocorre em algum estado desse caminho.

Segurança

Uma determinada condição é verdadeira em todos os estados, num caminho de execução. Esta propriedade está representada graficamente nas figuras seguintes:

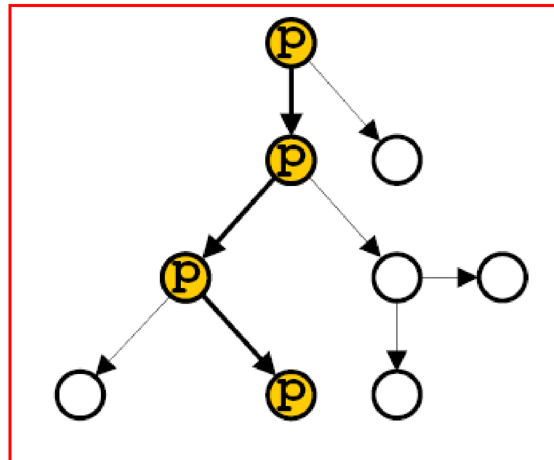

Figura 3.6 E[] p – p existe globalmente

Existe um caminho de execução onde *p* é verificado para todos os estados no caminho. Outra possibilidade, representada na figura seguinte, é quando para todos os caminhos de execução, *p* é verdadeiro para todos os estados.

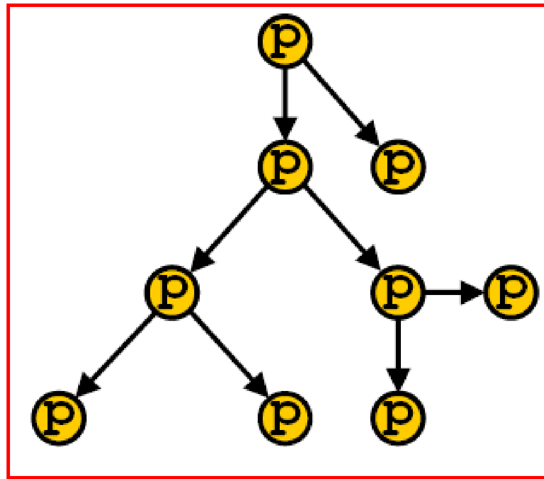

Figura 3.7 A[] p

Liveness

Uma determinada condição é verificada eventualmente algures no modelo.

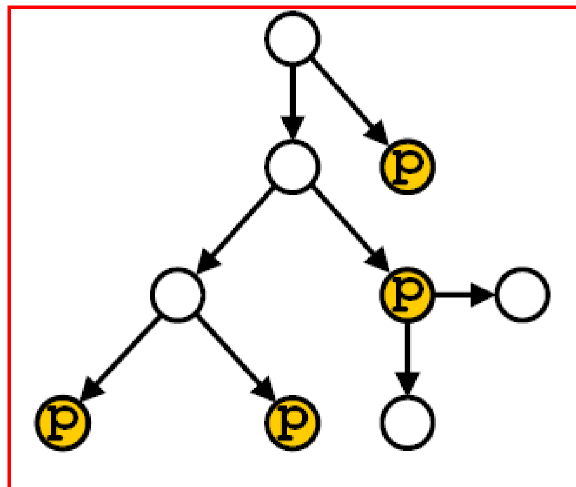

Figura 3.8 A<> p – “Sempre eventualmente p”

A figura anterior mostra que para todos os caminhos de execução, *p* é verdadeiro, pelo menos para um estado do modelo.

A figura seguinte mostra uma situação em que um caminho de execução que inicie com um estado onde *q* se verifique alcança posteriormente um estado onde *p* se verifica também.

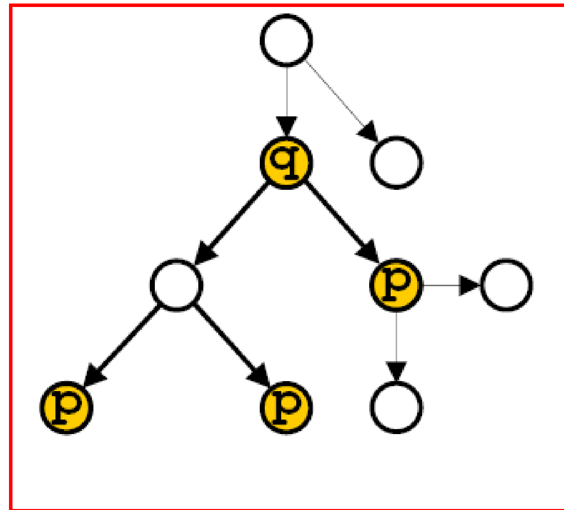

Figura 3.9 q->p – “q leva sempre a p”

Deadlock

Indica se é possível *deadlock* ou não no modelo.

Por exemplo, podem-se definir as seguintes propriedades a serem verificadas:

E<> deadlock indica “Existe *deadlock*”

A[] not deadlock indica “Não existe *deadlock*”

Nas expressões utilizadas para a verificação em UPPAAL o primeiro elemento, refere-se a uma quantificação em relação aos caminhos possíveis do modelo: “A” representa todos os caminhos e “E” algum caminho. O segundo elemento representa uma quantificação mas em relação aos estados dos caminhos considerados; “[]” representa todos os estados e “<>” representa algum estado.

À semelhança do que acontece em SPIN, nem sempre um *deadlock* representa um erro no modelo do sistema. Pode ser suficiente mostrar que o *deadlock* ocorre apenas quando é esperado. Este tipo de verificação será mostrado no decorrer do trabalho realizado.

Semanticamente, em UPPAAL a verificação é efectuada da seguinte forma:

O UPPAAL usa um sistema temporal de transições (S, $s0$, $\rightarrow$) representando uma rede de autómatos temporais. S representa o conjunto de todos os estados, $s0$ o estado inicial e $\rightarrow$ as transições possíveis de estado. No estado $s0$, todos os processos estão no seu

estado inicial. Ou seja, variáveis no seu estado inicial, relógios iguais a zero. Os dois tipos de transições possíveis são: transições de acção e transições de espera.
Quando é efectuada uma avaliação, se uma expressão for inválida, a mesma é abortada. Numa transição de espera, a mudança de um estado para o outro é efectuada por passagem de tempo. É esta uma das características em que o UPPAAL difere do SPIN.
Numa transição de acção, a mesma ocorre por sincronização dos nós e pode ser de três tipos: transição interna que é aquela se ocorre dentro dum processo, sincronização binária que é aquela que ocorre entre dois processos ou sincronização em *broadcast* que é aquela que ocorre entre um emissor e vários receptores. No decorrer do trabalho será novamente referenciada este tipo de sincronização associada ao UPPAAL.

4 Verificação de protocolos de votação electrónica com SPIN

O SPIN é uma ferramenta utilizada na verificação automática de sistemas distribuídos. Esta ferramenta apresenta uma interface gráfica que permite de uma forma intuitiva especificar e validar o modelo do sistema em estudo.

O SPIN permite efectuar uma simulação do modelo utilizando vários parâmetros, por exemplo, indicando valores que se vão repercutir na representação gráfica da simulação: espaçamentos e legendas a apresentar; escolha da visualização dos valores das variáveis locais, globais e canais de comunicação existentes.

Por outro lado, a simulação pode ser automática e aleatória, permitindo fazer *debug* do modelo. Neste tipo de simulação todas as decisões não determinísticas são resolvidas aleatoriamente. Pode efectuar-se simulações aleatórias baseadas num valor (*seed*) que, se for mantido, permite reproduzir sempre a mesma execução. Neste documento será apresentado um exemplo desta opção.

É possível também efectuar simulações nas quais se pode forçar uma determinada execução. Neste caso, para todos os pontos de escolha não determinística é requerida a intervenção do utilizador para a escolha de um dos ramos.

Além das opções anteriores, pode ainda efectuar-se uma simulação guiada. Esta opção permite seguir um erro que se tenha obtido previamente. Se a simulação tiver muitos passos, é possível omitir vários passos e avançar para passos posteriores.

Além das simulações é possível efectuar verificações e ainda gerar uma visualização do modelo de autómatos que o SPIN utiliza para validar o sistema. No SPIN cada processo é representado por um autómato.

Define-se um modelo de um sistema utilizando três tipos de objectos:

- Processos;
- Canais de mensagens;
- Variáveis de estado.

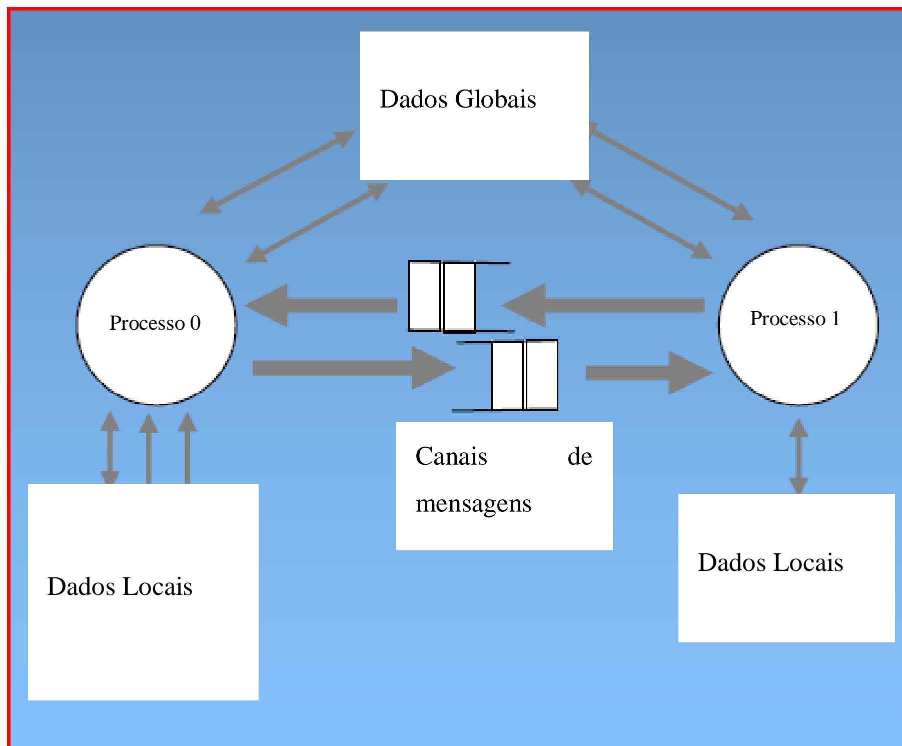

Figura 4.1 Comunicação entre os tipos de objectos em SPIN

Todos os processos são por defeito objectos globais e os canais ou variáveis podem ser locais ou globais.

A estrutura de um modelo em SPIN é muito idêntica à da linguagem C.

Os processos são definidos através de declarações *proctype* onde é definido o seu comportamento. Para os instanciar e executar é utilizado o processo *init* que é equivalente ao *main* da linguagem C. Este processo pode inicializar variáveis globais, criar canais de mensagens e instanciar processos.

Para a verificação, o SPIN tem um componente gestor de fórmulas LTL que permite expressar um comportamento positivo ou negativo consoante o pretendido. Uma propriedade positiva é negada automaticamente pelo SPIN para ser convertida numa cláusula *never* que representa a respectiva propriedade negativa. Esta propriedade representa o comportamento que não se pretende e que se afirma nunca ocorrer.

4.1 Modelação

4.1.1 Opções tomadas

Para modelar o sistema em SPIN foram tomadas as seguintes opções:

- Os eleitores registados são os correspondentes aos constantes no *array* `gEleitor[3]` onde o índice do *array* representa o eleitor em questão. Para cada eleitor, além de ser conhecida a sua identificação que corresponde ao índice indicado, é guardada informação sobre as assinaturas dos administradores já obtidas e o total de assinaturas válidas, bem como o estado do Módulo de Voto, ou seja a indicação do momento da votação em que cada eleitor se encontra. Por questões que se prendem com a memória disponível para efectuar verificações, são considerados apenas 2 eleitores.
- São assumidos por defeito 3 Administradores, sendo portanto de acordo com o protocolo necessárias duas assinaturas para que um voto seja válido.
- Na comunicação entre Anonimizador e o Totalizador, a identificação do eleitor é dissimulada para não obter uma associação directa entre o *id* do eleitor que efectua o voto e o que chega ao final do processo de votação. No entanto neste sistema essa associação será sempre possível e será uma das propriedades do sistema de votação electrónica que não será verificada.
- É permitido no máximo a recepção de 5 votos no Anonimizador e no Totalizador. Assim poderão existir repetições de votos.
- Abstrai-se completamente a troca de chaves, encriptações para comunicação segura entre servidores.
- O temporizador representará o momento do final da eleição, quando é publicada a chave privada da eleição e é possível validar se os votos que chegam ao final estão correctos e contá-los. Este processo foi criado para permitir simular o decorrer de algum tempo, a partir do qual se considera que a eleição terminou e é calculado o escrutino final.
- O modelo é constituído por 7 processos sendo o processo *init* que lança todos os outros

4.1.2 Representação do protocolo e principais estruturas utilizadas

O REVS foi representado por 7 processos que em tempo de execução corresponderão a 13 instâncias distribuídas da seguinte forma:

PROCESSO	# INSTÂNCIAS
ModuloVoto	2
Distribuidor	2
Administrador	3
Anonimizador	2
Totalizador	2
Temporizador	1
Init	1

Tabela 4-2 SPIN - Processos do modelo e suas instanciações de execução

Utilizando a linguagem do SPIN, o PROMELA, elaborou-se o modelo deste sistema que é apresentado e explicado nesta secção. A totalidade do código do modelo é incluída no Anexo A.

- O SPIN apenas permite fazer verificações sobre variáveis/canais globais pelo que todos os elementos do modelo importantes para a verificação terão que ser globais ao modelo. São definidas como constantes as mensagens trocadas entre o Módulo de Voto e o Distribuidor e que estão ligadas com os estados do Módulo de Voto. Estas declarações são efectuadas na área global, ou seja, são comuns a todos os processos

```
#define PedidoBoletimEleicao 1; /*Mensagens trocadas entre o Modulo de Voto e
Distribuidor*/
#define EnvioBoletimEleicao 2;
#define PedidoBoletimVoto 3;
#define EnvioBoletimVoto 4;
#define PedidoAssinaturaVoto 5;
#define EnvioAssinaturaVoto 6;
#define EnvioSubmissaoVoto 7;
```

- Além disso os estados do Módulo de Voto são também guardados. Assim em cada momento sabe-se em que ponto da votação o eleitor se encontra.

```
#define INICIO 0;
#define PedidoBoletimEleicaoEnviado 1;
#define PedidoBoletimVotoEnviado 2;
```

```
#define PedidoAssinaturaVotoEnviado 3;
#define TotalAssinaturasObtido 4;
#define FIM 5;
```

- O número de processo do modelo é parametrizado, pelo que é possível fazer variar o número de processos com o mínimo de impacto no modelo.

```
#define NumDistribuidores 2;
#define NumAdministradores 3;
#define NumEleitores 2;
#define NumAnonimizadores 2;
#define NumTotalizadores 2;
#define CapacidadeEscrutinio 5;
#define NumTotalAssinaturas 2;
#define DELTA 10;
```

- Nesta área estão também definidas as selecções dos Distribuidores, Administradores e Totalizadores, já que temos mais do que um de cada e não contactamos sempre o mesmo. É permitida assim simulação de um modelo distribuído.

```
int selVoto=0, selAnonimizador=0, selDistribuidor=0;

#define SelDistribuidor() selDistribuidor = (selDistribuidor+1) % NumDistribuidores
#define SelAnonimizador() selAnonimizador = (selAnonimizador+1) % NumAnonimizadores
#define SelAdministrador() admin = (admin+1) % NumAdministradores
```

- Existe ainda uma área de valores definidos para simular o temporizador com valores diferentes.

```
#define randomNumber()       if  \
                                              :: count = 5 \
                                              :: count = 15 \
                                              :: count = 30 \
                                              :: count = 90 \
                                              fi
```

- As variáveis seguintes indicam respectivamente quando o tempo termina e o escrutínio é iniciado e quando a contagem dos votos termina.

```
int iniciarEscrutinio=0, fimEscrutinio = 0;
```

- Variáveis de índice correspondentes a estruturas globais.

```
int indexA=0, indexR=0;
```

- Votos podem ficar perdidos se o temporizador terminar sem que todos os eleitores tenham terminado a votação.

```
int  lostMsg = 0;
```

- Para a comunicação entre os processos, utilizaram-se canais que também foram declarados globais, pelas razões já indicadas. São utilizados canais distintos para comunicação entre Módulo de Voto e Distribuidor, Administrador e Distribuidor. Depois existem ainda mais dois canais: um para comunicação entre Anonimizador e Totalizador e outro que permite provocar um atraso aleatório na ordem dos votos.

```
chan gEleitorDistribuidor = [10] of {byte, byte, byte};

chan gEleitorAdministrador = [10] of {byte, byte,byte};

chan gEleitorAnonimizador = [10] of {byte, byte, byte};

chan gAnonimizadorTotalizador = [10] of {byte, byte, byte};

chan gDelayAnonimizadorTotalizador = [10] of {byte, byte, byte};
```

Os canais seguem uma filosofia de FIFO (First In, First Out), e podem ser síncronos ou assíncronos. Os síncronos têm uma dimensão do vector igual a zero, os assíncronos tem dimensão diferente de zero. Neste trabalho, utilizaram-se canais assíncronos, ou seja um processo lê ou escreve várias mensagens no canal e não fica à espera de outra operação.

- Finalmente existem ainda estruturas que permitem guardar informação sobre os estados do eleitor em cada momento, que assinaturas possui e de que administradores e o número total de assinaturas.

```
typedef  recEleitor {
        int Estado;
        bool AssinaturaAdministrador[3];
        int NumAssinaturas;
};

typedef  recVoto {
        int IdVoto;
        int NumAssinaturas
};
```

- As estruturas seguintes permitem ter informação sobre todos os eleitores bem como todos os votos recebidos no Anonimizador e contados efectivamente no Totalizador.

```
recEleitor gEleitor[3];
recVoto gVotoRecebido[5];
recVoto gVotoApurado[5];
```

4.1.3 Descrição dos Processos

Módulo de Voto

O processo do Módulo de Voto recebe como parâmetro o eleitor. Este processo tem uma variável local que indica o estado em que está o processo Módulo de Voto. Começa no estado inicial e de seguida solicita um boletim de eleição, alterando na mesma operação e de forma atómica o seu estado. De seguida, aguarda no mesmo canal a recepção do Boletim de Eleição. Os passos repetem-se até o eleitor ter o seu boletim de voto. Após isso, solicita as assinaturas aos Administradores até obter o total de assinaturas necessário para que o seu voto seja validado. Assim sendo assumiu-se que cada eleitor apenas submete o seu voto após ter um número de assinaturas de Administradores diferentes e necessário para que o seu voto seja válido no final. Quando obtém todas as assinaturas, submete o seu voto ao Anonimizador. Note-se que esta operação não tem qualquer retorno de mensagem, pelo que não se poderá saber efectivamente se o voto chegou ou não ao seu destino. Por isso mesmo o protocolo permite que o eleitor submeta várias vezes o seu voto. No modelo base definido o eleitor apenas submete uma única vez.

```
proctype ModuloVoto(int pEleitor)
{
        int lDistribuidor;
        int admin = 0;

        do
        :: gEleitor[pEleitor].Estado == INICIO ->
              atomic {
                     SelDistribuidor();
                     gEleitorDistribuidor!pEleitor, selDistribuidor,
                                                  PedidoBoletimEleicao;
                     gEleitor[pEleitor].Estado = PedidoBoletimEleicaoEnviado;
                     }
        :: gEleitor[pEleitor].Estado == PedidoBoletimEleicaoEnviado ->
              gEleitorDistribuidor?eval(pEleitor), lDistribuidor,
                                                  EnvioBoletimEleicao ->
              atomic {
                     gEleitorDistribuidor!pEleitor, lDistribuidor, PedidoBoletimVoto;
                     gEleitor[pEleitor].Estado = PedidoBoletimVotoEnviado
              }
        :: gEleitor[pEleitor].Estado == PedidoBoletimVotoEnviado ->
              gEleitorDistribuidor?eval(pEleitor), lDistribuidor, EnvioBoletimVoto ->
              atomic {
                     SelAdministrador();
                     gEleitorAdministrador!pEleitor, admin, PedidoAssinaturaVoto;
                     gEleitor[pEleitor].Estado = PedidoAssinaturaVotoEnviado;
                     }
        :: gEleitor[pEleitor].Estado == PedidoAssinaturaVotoEnviado  ->
              do
              :: gEleitor[pEleitor].NumAssinaturas < NumTotalAssinaturas ->
                     gEleitorAdministrador?eval(pEleitor), eval(admin),
                                                          EnvioAssinaturaVoto;
                     if
                     :: gEleitor[pEleitor].AssinaturaAdministrador[admin] == 0 ->
                            atomic {
                            gEleitor[pEleitor].AssinaturaAdministrador[admin] = 1;
                            gEleitor[pEleitor].NumAssinaturas++;
                            }
```

```
                        fi;
                        if
                        :: gEleitor[pEleitor].NumAssinaturas < NumTotalAssinaturas ->
                                atomic {
                                        SelAdministrador();
                                        gEleitorAdministrador!pEleitor,admin,
                                                                PedidoAssinaturaVoto;
                                }
                        :: gEleitor[pEleitor].NumAssinaturas == NumTotalAssinaturas ->
                                gEleitor[pEleitor].Estado = TotalAssinaturasObtido;
                                break;
                        fi;
                od;
        :: gEleitor[pEleitor].Estado == TotalAssinaturasObtido ->
                atomic {
                        SelAnonimizador();
                        gEleitorAnonimizador!pEleitor,gEleitor[pEleitor].NumAssinaturas,
                                                                selAnonimizador;
                        gEleitor[pEleitor].Estado = FIM;
                }
        :: gEleitor[pEleitor].Estado == FIM -> break;
        od;
}
```

Distribuidor

O processo Distribuidor encontra-se continuamente em ciclo e consoante receba um Pedido de Boletim de Eleição ou Pedido de Boletim de Voto devolve o respectivo Boletim de Eleição ou Boletim de Voto. Este processo originará uma sequência cíclica que não termina, mas que poderá ser definida como válida pois é o funcionamento pretendido.

```
proctype Distribuidor(byte pDistribuidor)
{
        int Eleitor=0;

        end:
        do
        :: gEleitorDistribuidor?Eleitor, eval(pDistribuidor), PedidoBoletimEleicao ->
                gEleitorDistribuidor!Eleitor, pDistribuidor, EnvioBoletimEleicao
        :: gEleitorDistribuidor?Eleitor, eval(pDistribuidor), PedidoBoletimVoto ->
                gEleitorDistribuidor!Eleitor, pDistribuidor, EnvioBoletimVoto
        od
}
```

Administrador

O processo Administrador, à semelhança do processo Distribuidor, encontra-se continuamente em ciclo e caso receba um Pedido de Assinatura de Voto, simplesmente envia a assinatura para o respectivo canal.

```
proctype Administrador(int pAdmin)
{
        int Eleitor = 0;

        end:
        do
```

```
        :: gEleitorAdministrador?Eleitor,eval(pAdmin),PedidoAssinaturaVoto ->
              gEleitorAdministrador!Eleitor,pAdmin,EnvioAssinaturaVoto
        od
}
```

Anonimizador

Este processo recebe os votos vindos do Módulo de Voto e provoca um atraso aleatório nos votos, bem como altera a sua identidade, para que o totalizador não possa saber de quem é o voto.

```
proctype Anonimizador(int pAnonimizador)
{
        int Eleitor, lidEleitor, lTotalAssinaturas = 0;

        endAnonimizador: do
        :: gEleitorAnonimizador??lidEleitor, lTotalAssinaturas, eval(pAnonimizador) ->
              gPrimeiroEleitor == 0 -> gPrimeiroEleitor = lidEleitor+DELTA;
              gDelayAnonimizadorTotalizador!lidEleitor, lTotalAssinaturas,
                                              pAnonimizador
        :: gDelayAnonimizadorTotalizador??lidEleitor, lTotalAssinaturas,
eval(pAnonimizador) ->
              atomic { Eleitor = lidEleitor+DELTA; gAnonimizadorTotalizador!Eleitor,
                                              lTotalAssinaturas, pAnonimizador; Eleitor=0}
        od;
}
```

Totalizador

O Totalizador recebe ciclicamente mensagens do Anonimizador enquanto não tiver começado o escrutínio. Após o início do escrutínio, o que é determinado através da variável `iniciarEscrutinio`, um dos Totalizadores iniciará o processo de escrutínio apurando os votos que são válidos transferindo a informação do *array* `gVotoRecebido` para `gVotoApurado`. Neste apuramento só serão considerados os votos que não tenham duplicações e que tenham o número de assinaturas correcto.

```
proctype Totalizador(int pTotalizador)
{
        int x,y,z;
        do
        :: iniciarEscrutinio == 0 && selVoto < CapacidadeEscrutinio ->
              if
              :: atomic{
                    gAnonimizadorTotalizador??gVotoRecebido[selVoto].IdVoto,
                          gVotoRecebido[selVoto].NumAssinaturas, eval(pTotalizador);
                    selVoto++;
                    };
              :: empty(gAnonimizadorTotalizador) -> skip;
              fi;
        :: iniciarEscrutinio  == 1 -> break;
        :: else -> skip;
        od;
        if
        :: pTotalizador == 0 ->
              do
              :: indexR < selVoto ->
                    if
```

```
                    :: gVotoApurado[indexA].IdVoto == 0 &&
                    gVotoRecebido[indexR].NumAssinaturas == NumTotalAssinaturas ->
                            atomic {
                                    gVotoApurado[indexA].IdVoto =
                                            gVotoRecebido[indexR].IdVoto;
                                    gVotoApurado[indexA].NumAssinaturas =
                                            gVotoRecebido[indexR].NumAssinaturas;
                                    indexR++;
                                    indexA++;
                            }
                    :: gVotoApurado[indexA].IdVoto != gVotoRecebido[indexR].IdVoto &&
                                    gVotoApurado[indexA].IdVoto != 0 -> indexA++;
                    :: else-> indexR++;
                    fi;
            :: else -> fimEscrutinio = 1; break;
            od;
        :: else  -> skip;
        fi;

        end:
        do
        :: gAnonimizadorTotalizador??x,y,z -> lostMsg++;
        od;
}
```

Temporizador

O processo temporizador permite que, após algum tempo decorrido, seja actualizada a variável correspondente ao momento final da eleição, `iniciarEscrutinio`. Quando essa variável tiver o valor 1, significa que deverá ser efectuado o escrutínio, ou seja, num dos processos Totalizador são retiradas as repetições de votos recebidos, verificados os votos que têm o número de assinaturas válido e são guardados para contagem final. Iremos depois definir propriedades sobre estes repositório de votos final, `gVotoApurado[5]`. O momento do escrutínio representa na especificação o momento da publicação da chave privada da eleição e o acesso aos dados da votação. Como o SPIN não permite o conceito de tempo, este foi implementado através de um contador que é incrementado até atingir aleatoriamente determinados valores. Assim, quando esse tempo é atingido, é despoletado o escrutínio.

```
proctype Temporizador()
{
        int count;
        int i = 0;

        randomNumber();
        do
        :: i < count -> i++;
        :: i == count -> break;
        od;

        iniciarEscrutinio = 1;
}
```

init

O processo principal faz a instanciação dos vários processos do modelo.

```
init
{
        int id = 0;

        /* Lancar um numero variavel de Eleitores */
        do
        :: id < NumEleitores -> atomic { run ModuloVoto(id); id++ }
        :: id == NumEleitores -> id=0; break
        od;

        /* Lancar um numero variavel de Distribuidores */

        do
        :: id < NumDistribuidores -> atomic { run Distribuidor(id); id++ }
        :: id == NumDistribuidores -> id=0; break
        od;

        /* Lancar um numero variavel de Administradores */
        do
        :: id < NumAdministradores -> atomic { run Administrador(id); id++ }
        :: id == NumAdministradores -> id=0; break
        od ;

        /* Lancar um numero variavel de Anonimizadores */
        do
        :: id < NumAnonimizadores -> atomic { run Anonimizador(id); id++ }
        :: id == NumAnonimizadores -> id=0; break
        od;

        /* Lancar um numero variavel de Totalizadores */
        do
        :: id < NumTotalizadores -> atomic { run Totalizador(id); id++ }
        :: id == NumTotalizadores -> id=0; break
        od;

        run Temporizador();
}
```

4.2 Simulação

Após a modelação, é necessário efectuar uma verificação de sintaxe sobre o modelo (ver Figura 4.2).

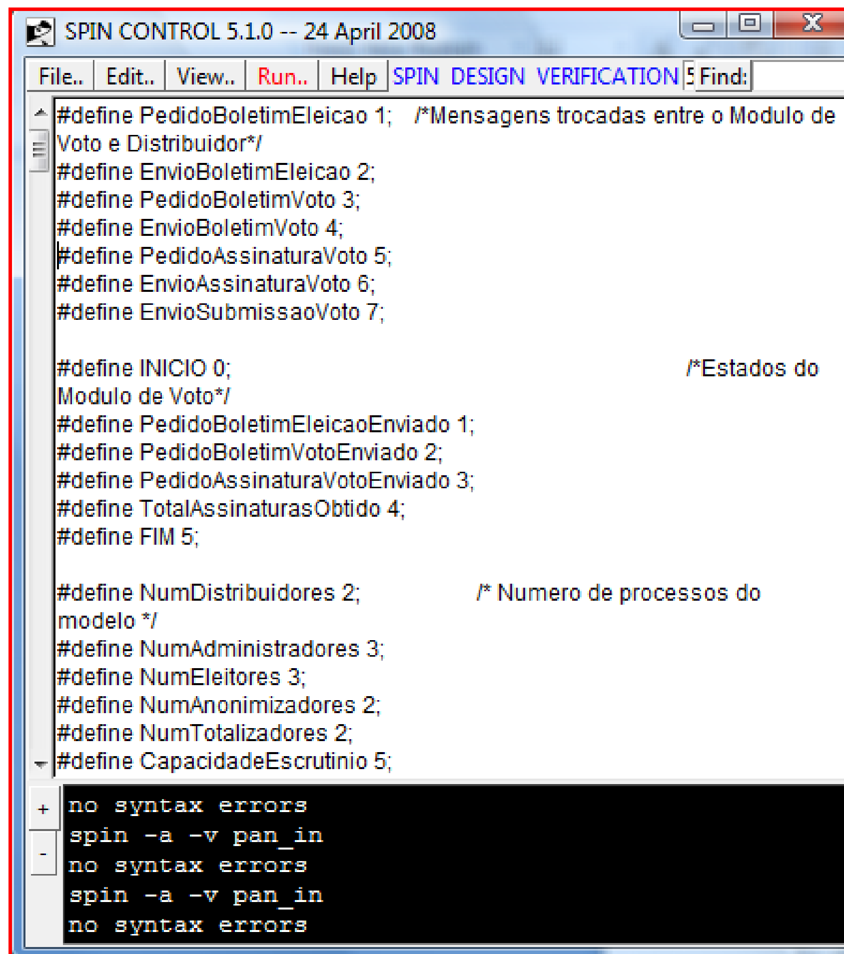

Figura 4.2 Validação de sintaxe em SPIN

Pode também ser efectuada uma validação Run Slicing Algoritm (ver Figura 4.3). Esta validação permite identificar se existem redundâncias desnecessárias no modelo, o que o tornará complexo e poderá causar problemas de memória nas verificações.

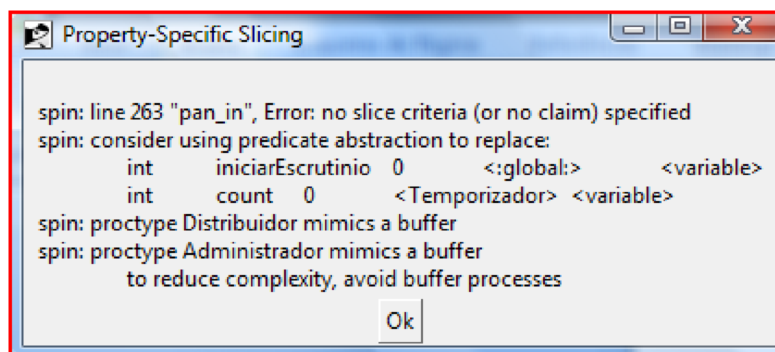

Figura 4.3 Validação de redundâncias no modelo

Pode efectuar-se uma simulação aleatória não indicando nenhum valor em *Seed Value*, visualizando as variáveis globais/locais ou canais de comunicação, tal como seleccionado na figura seguinte:

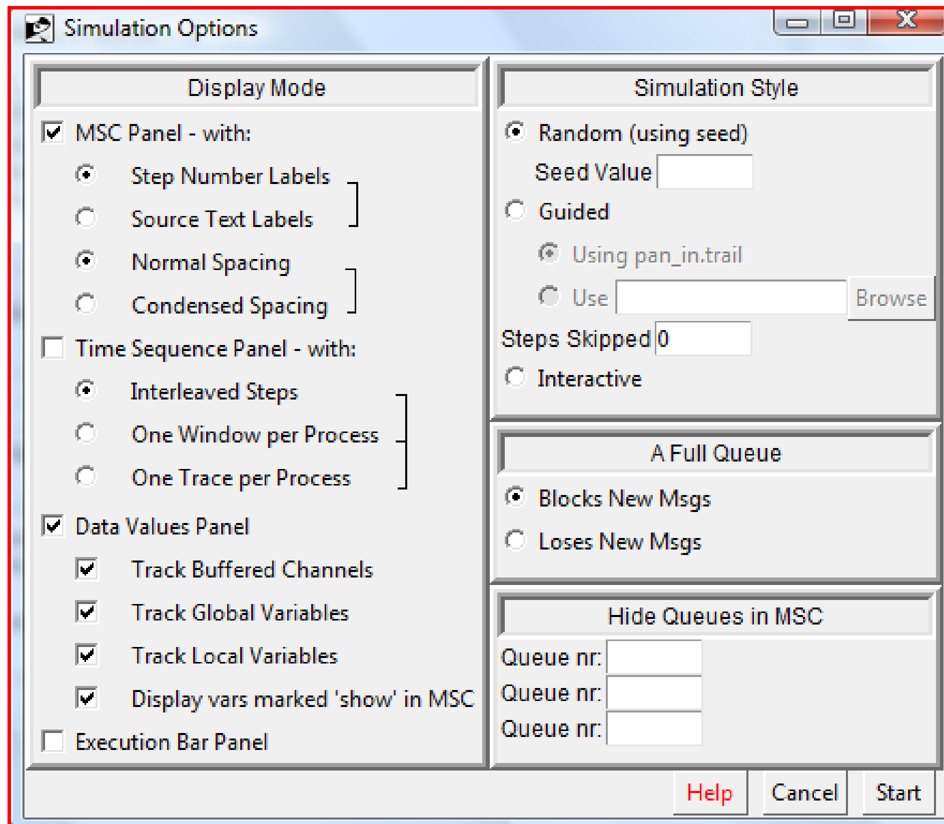

Figura 4.4 Janela de simulação do SPIN

O resultado da simulação é o seguinte:

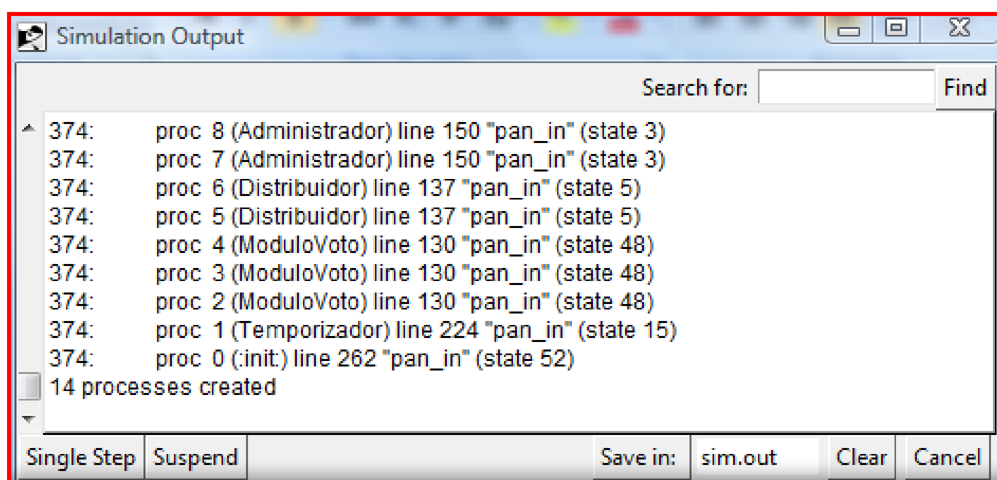

Figura 4.5 Resultado de uma simulação

Na figura anterior é mostrada parte da simulação do modelo. Este detalhe é muito importante para fazer *debug*, já que permite ver linha a linha a execução simulada.

Na imagem seguinte é visualizado um detalhe das variáveis após uma das execuções. Em simulações guiadas também se pode visualizar passo a passo a evolução destes valores.

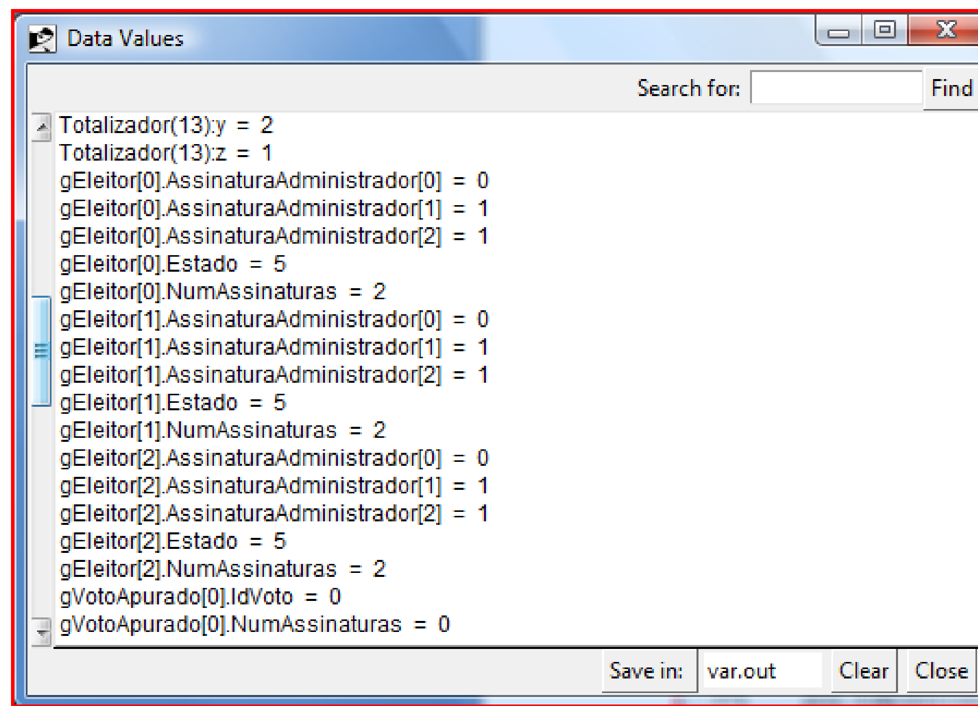

Figura 4.6 Valores das variáveis e canais numa simulação

As Figuras 4.7 e 4.8 apresentam uma parte de uma simulação do modelo, mostrando algumas trocas de mensagens entre os diferentes processos.

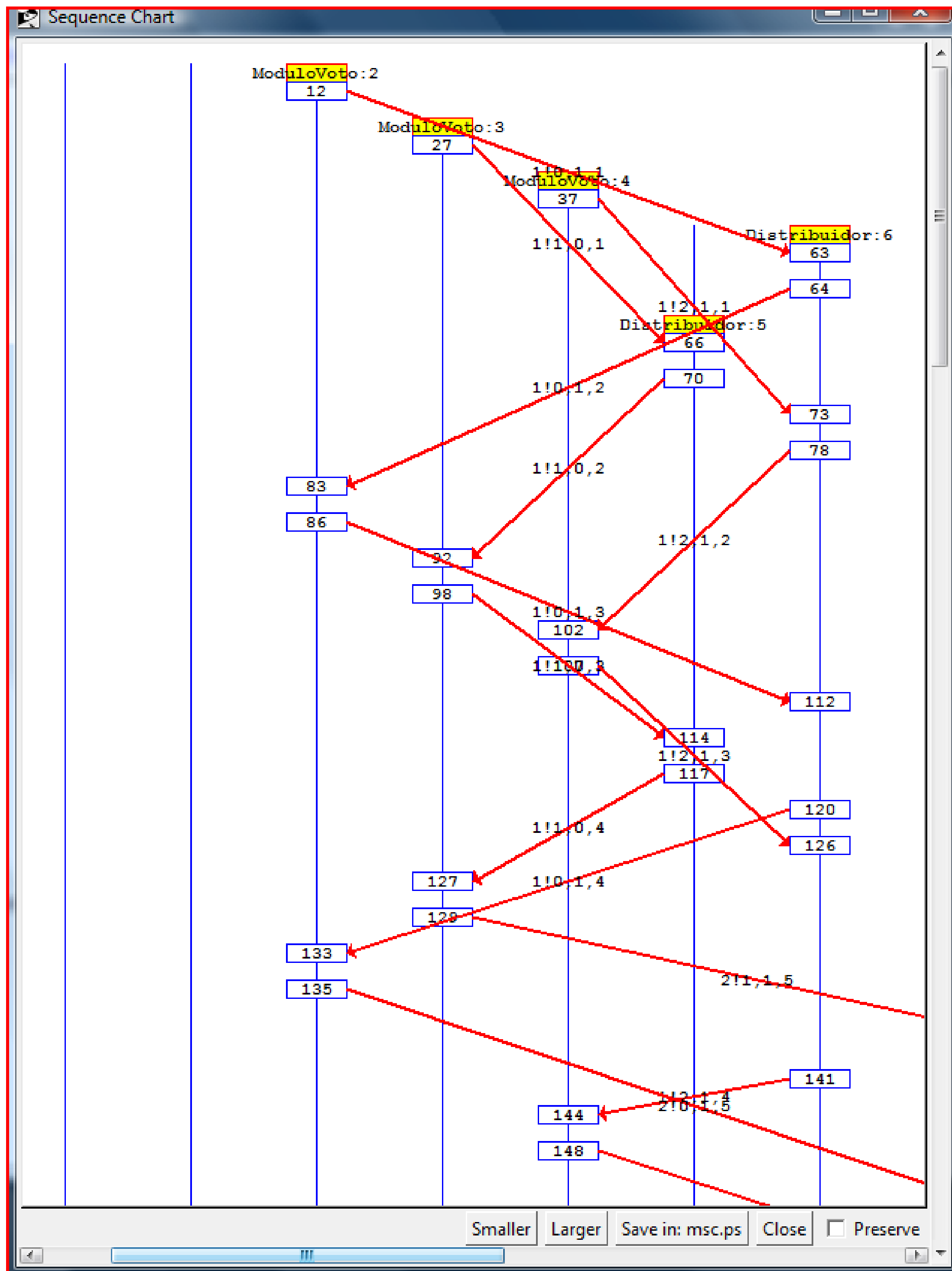

Figura 4.7 Visualização gráfica da simulação aleatória 1

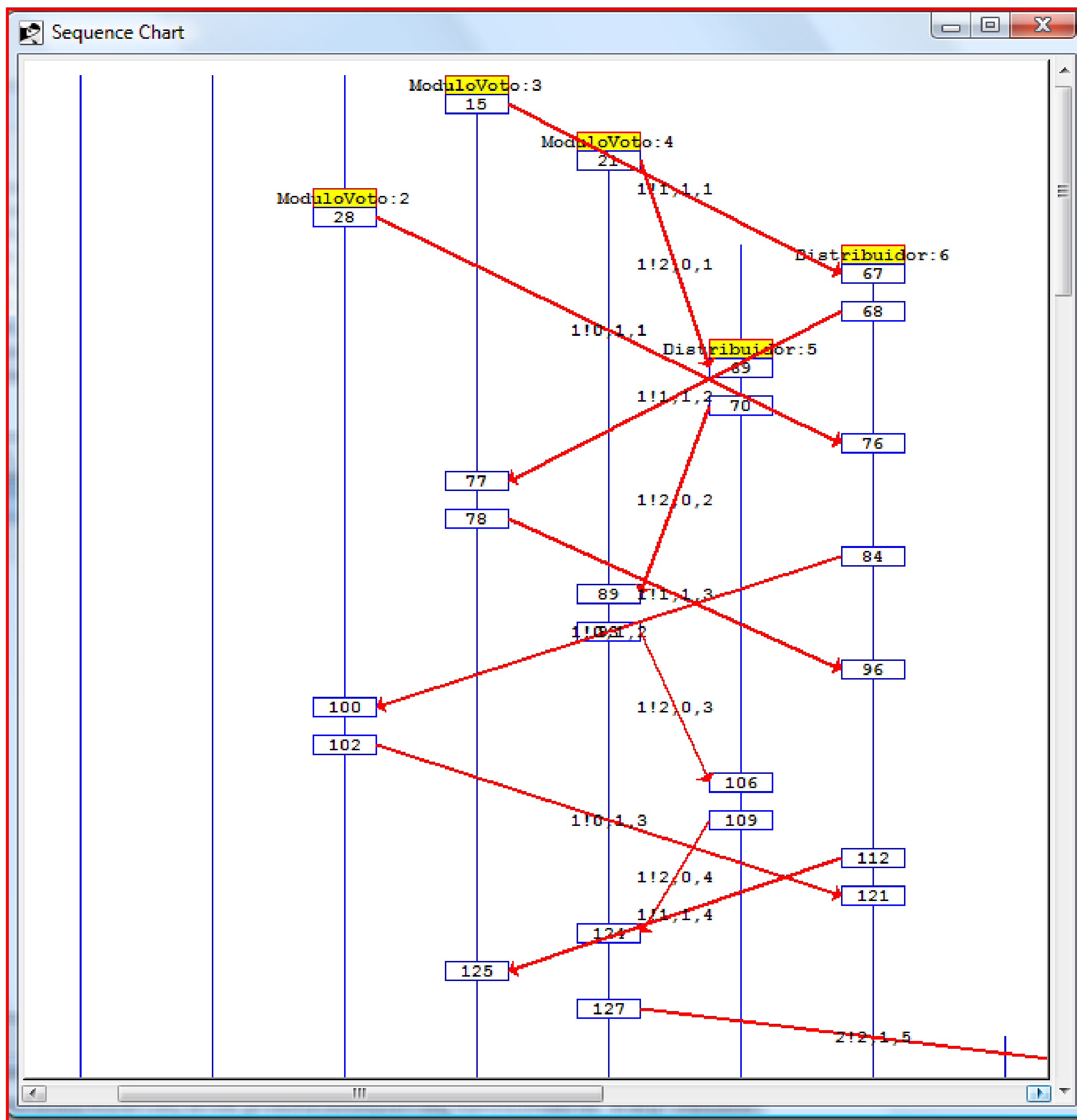

Figura 4.8 Visualização gráfica da simulação aleatória 2

Na secção seguinte serão apresentadas algumas propriedades genéricas e específicas de votação electrónica.

4.3 Verificação de propriedades genéricas

O SPIN permite verificar vários tipos de propriedades: propriedades genéricas e aplicadas a qualquer modelo; outras propriedades específicas e relacionadas com a modelação em causa. Numa primeira verificação, pode ver-se se o modelo tem estados inválidos ou código que nunca é executado, utilizando para tal as opções a seguir indicadas na Figura 4.9:

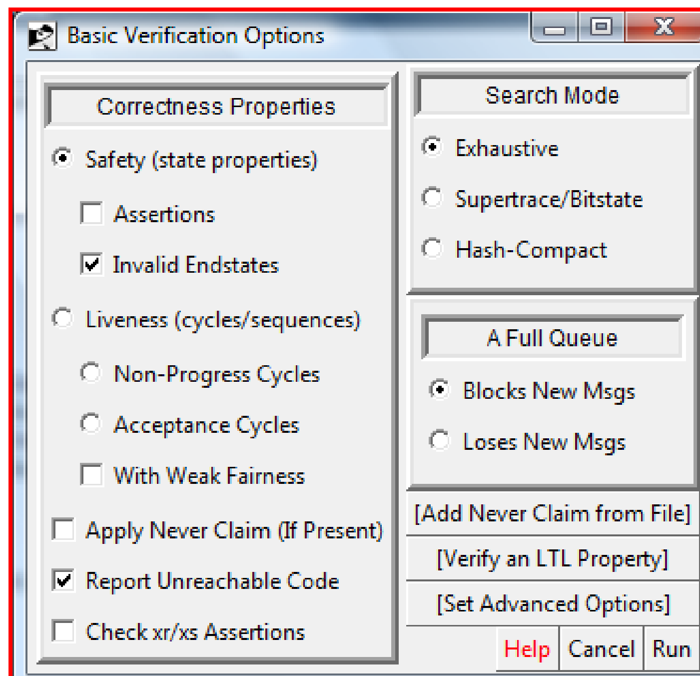

Figura 4.9 Opções de verificação

O resultado da verificação é a seguir apresentado na Figura 4.10.

```
(Spin Version 5.1.6 -- 9 May 2008)
        + Partial Order Reduction

Full statespace search for:
        never claim         - (not selected)
        assertion violations - (disabled by -A flag)
        cycle checks        - (disabled by -DSAFETY)
        invalid end states +

State-vector 496 byte, depth reached 712, errors: 0
  489574 states, stored
  667523 states, matched
 1157097 transitions (= stored+matched)
   47793 atomic steps
hash conflicts:    118690 (resolved)

 236.192  memory usage (Mbyte)

unreached in proctype ModuloVoto
        (0 of 48 states)
unreached in proctype Distribuidor
        line 148, "pan_in", state 8, "-end-"
        (1 of 8 states)
unreached in proctype Administrador
        line 159, "pan_in", state 6, "-end-"
        (1 of 6 states)
unreached in proctype Anonimizador
        line 171, "pan_in", state 11, "-end-"
        (1 of 11 states)
unreached in proctype Totalizador
        line 200, "pan_in", state 25, "indexA = (indexA+1)"
        line 201, "pan_in", state 27, "indexR = (indexR+1)"
        line 213, "pan_in", state 44, "-end-"
        (3 of 44 states)
unreached in proctype Temporizador
        (0 of 15 states)
unreached in proctype :init:
        (0 of 52 states)

pan: elapsed time 2.87 seconds
pan: rate 170880.98 states/second
pan: avg transition delay 2.476e-006 usec
```

Figura 4.10 Resultado de uma verificação

Na figura anterior vemos que são indicados o total de estados e o total de estados não atingidos, por processo, não existindo erros. Para os processos ModuloVoto, Temporizador e *init* não existe código não atingido. Para os processos Distribuidor, Administrador, Anonimizador e Totalizador existe, em cada um, uma linha de código não atingida. Essa linha corresponde aos ciclos infinitos já que estes processos estão em ciclo a tratar de pedidos. Estes estados são portanto esperados. No processo Totalizador

existem ainda mais duas linhas não atingidas, porque não temos votos repetidos e portanto a parte do algoritmo identificada nunca é atingida neste caso.

Será sempre efectuada a verificação exaustiva. O modelo foi construído com utilização de constantes que indicam o número de processos do sistema de modo a que seja possível parametrizar com os valores pretendidos em cada momento. Assim, em função dos recursos de memória podemos facilmente aumentar ou diminuir o número de processos do modelo.

4.4 Verificação de propriedades de votação electrónica

Para verificar as propriedades relacionadas com o modelo, utilizaremos o "Linear Time Temporal Logic Formulae" que permite especificar propriedades em LTL (ver Figura 4.11). Esta interface permite ler propriedades já especificadas e guardadas em ficheiro, bem como especificar novas propriedades. Quando especificamos uma propriedade, ela será transformada numa cláusula *never* com o comportamento inverso do especificado. O que significa que o SPIN vai tentar encontrar um estado em que aconteça o inverso do indicado. Se o encontrar significa que a propriedade não é satisfeita. Se não, significa que a propriedade é satisfeita.

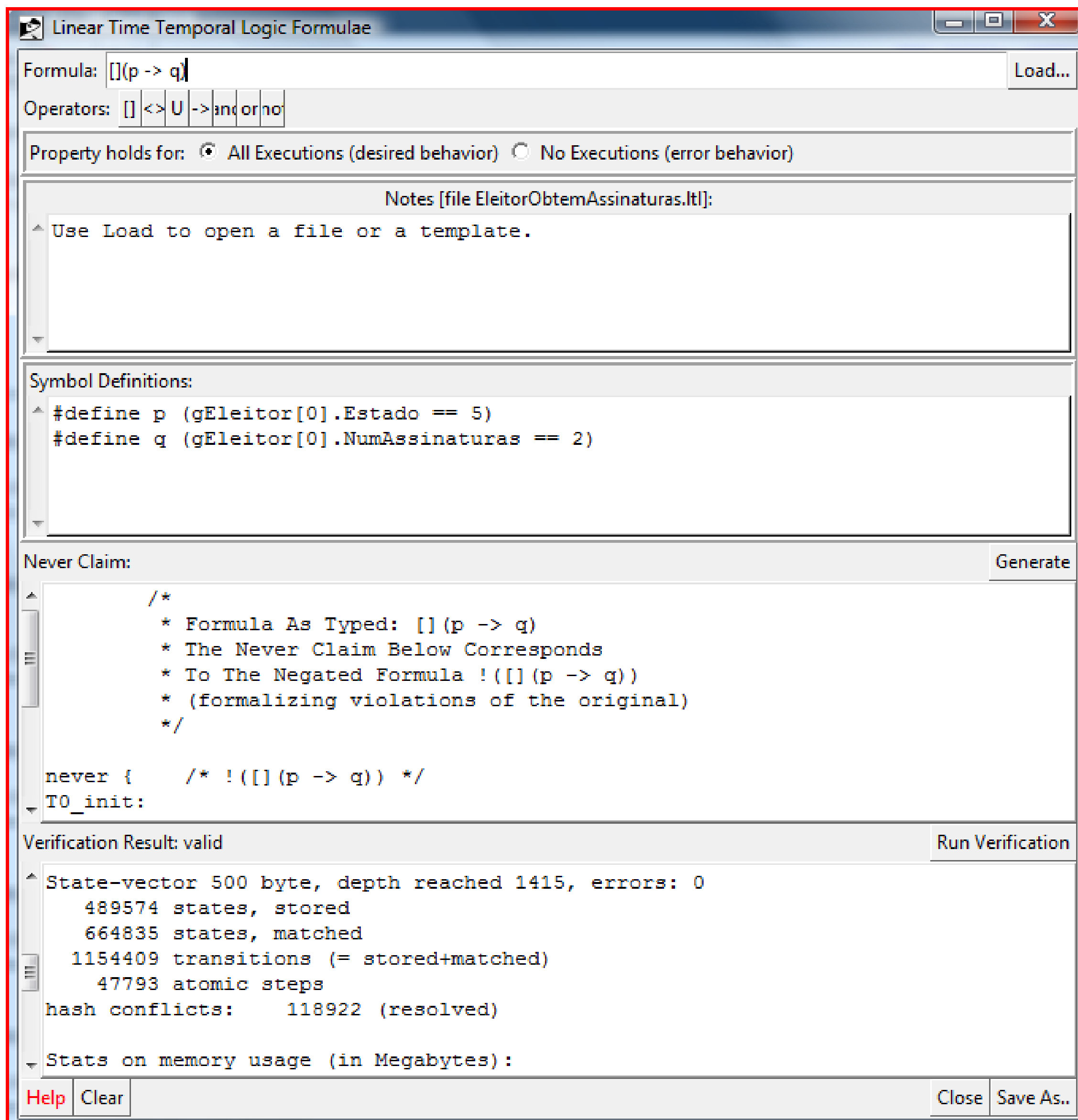

Figura 4.11 Output de uma verificação de propriedade

De seguida será utilizado o LTL para a especificação e verificação de propriedades inerentes aos sistemas de votação electrónica, para o modelo construído e já apresentado. Ver Secção 3.1 para notação utilizada em LTL.

Após uma verificação, o resultado será sempre idêntico ao apresentado a seguir e gerado automaticamente pelo SPIN. Para cada propriedade especificada e verificada, o seu detalhe será apresentado no anexo A.

```
/*
 * Formula As Typed: [](p -> q)
 * The Never Claim Below Corresponds
 * To The Negated Formula !([](p -> q))
 * (formalizing violations of the original)
 */
```

```
never {    /* !([](p -> q)) */
T0_init:
       if
       :: (! ((q)) && (p)) -> goto accept_all
       :: (1) -> goto T0_init
       fi;
accept_all:
       skip
}

#ifdef NOTES
Use Load to open a file or a template.

#endif
#ifdef RESULT
warning: for p.o. reduction to be valid the never claim must be stutter-invariant
(never claims generated from LTL formulae are stutter-invariant)
depth 0: Claim reached state 5 (line 258)
Depth=     265 States=    1e+06 Transitions= 2.79e+06 Memory=     8.598   t=   18.6 R=
5e+04

(Spin Version 5.1.6 -- 9 May 2008)
       + Partial Order Reduction

Bit statespace search for:
       never claim          +
       assertion violations  + (if within scope of claim)
       acceptance   cycles   + (fairness disabled)
       invalid end states    - (disabled by never claim)

State-vector 660 byte, depth reached 265, errors: 0
  1089566 states, stored
  1972458 states, matched
  3062024 transitions (= stored+matched)
   207741 atomic steps

hash factor: 61.5923 (best if > 100.)

bits set per state: 3 (-k3)

Stats on memory usage (in Megabytes):
  702.426      equivalent memory usage for states (stored*(State-vector + overhead))
    8.000      memory used for hash array (-w26)
    0.038      memory used for bit stack
    0.305      memory used for DFS stack (-m10000)
    8.598      total actual memory usage


pan: elapsed time 20.4 seconds
pan: rate 53305.577 states/second
pan: avg transition delay 6.6753e-06 usec
       20.45 real        20.20 user         0.07 sys

#endif
```

As cláusulas *never* devem ser invariantes. Por exemplo, não se pode utilizar em LTL no SPIN o valor de uma constante. É ainda visível caso ocorram erros na verificação, a memória utilizada e o tempo utilizados para a mesma.

Para as propriedades não verificadas após o resultado da verificação é possível efectuar uma simulação guiada pelo contra-exemplo encontrado e confirmar o resultado ou se é um problema do modelo.

Nas propriedades seguintes apenas será mostrado, para cada uma das propriedades enunciadas e especificadas, se a mesma é ou não satisfeita.

4.4.1 Privacidade

Definição da propriedade

"Não é possível estabelecer uma ligação entre o eleitor e o seu voto."

Esta propriedade está em contradição com a propriedade de precisão, tal como referido em [30], definida na Secção 4.4.3: "Se um voto for efectuado pelo eleitor, ele deverá ser contado no escrutínio final".

Resultado da verificação

A verificação da propriedade na Secção 4.4.3 implica que o voto seja contado no escrutínio final e para tal o voto terá que ser distinguido em função do seu *bit commitment* (representado pelo valor de DELTA).

O eleitor envia o *bit commitment* juntamente com o seu voto, encriptado com a chave pública da eleição. Devido à encriptação a informação não é visível pelo Anonimizador. Após a publicação da chave privada da eleição, o eleitor pode verificar se o seu voto foi contado através do *bit commitment*. Logo, as propriedades de anonimato, não coercibilidade e ausência de recebido de voto podem ser violadas, pois o *bit commitment* pode associar de forma única o eleitor a um determinado voto.

4.4.2 Democracia

Definição da propriedade

"Só os eleitores elegíveis votam e uma única vez."

```
#define p ((gVotoApurado[0].IdVoto == 10) | (gVotoApurado[1].IdVoto == 10) |
                (gVotoApurado[2].IdVoto == 10))
        && ((gVotoApurado[0].IdVoto == 11) | (gVotoApurado[1].IdVoto == 11) |
                (gVotoApurado[2].IdVoto == 11))
        && ((gVotoApurado[0].IdVoto == 12) | (gVotoApurado[1].IdVoto == 12) |
                (gVotoApurado[2].IdVoto == 12))

#define q (indexR == selVoto && indexR == 3)

[] (q -> p)
```

Resultado da verificação

A propriedade é satisfeita pelo modelo. Foi efectuada a verificação para os três eleitores que estão definidos no modelo. É incluída uma condição `indexR == selVoto && indexR == 3)` para considerar apenas a situação em que chegam os três votos. São assim excluídos os estados em que o voto não é enviado antes do fecho das eleições.

4.4.3 Precisão

Definição da propriedade

“Se um voto for efectuado pelo eleitor, ele deverá ser contado no escrutínio final”.

```
#define p (gVotoApurado[0].IdVoto == 10) | (gVotoApurado[1].IdVoto == 10) |
          (gVotoApurado[2].IdVoto == 10)
#define q (gVotoRecebido[0].IdVoto == 10) | (gVotoRecebido[1].IdVoto == 10) |
          (gVotoRecebido[2].IdVoto  == 10)

[]  p-> q
```

Resultado da verificação

A propriedade é satisfeita pelo modelo.

4.4.4 Fiabilidade

“Não é possível contar ou publicar contagens intermédias”

Esta propriedade está dependente das questões relacionadas com a encriptação, já que os votos vão para o Anonimizador encriptados coma chave pública da eleição. A desencriptação só pode acontecer após o fim da eleição e consequente publicação da chave privada da eleição. É possível saber quantos votos foram submetidos a partir de determinada máquina mas não se sabe de quem são os mesmos ou em quem o eleitor votou. Esta propriedade não será portanto confirmada através do modelo corrente.

4.4.5 REVS – Assinaturas para o voto ser válido

“Um eleitor que termine o seu processo de voto garantidamente obteve um número de assinaturas suficiente para o voto ser válido.”

Definição da propriedade

```
#define p (gEleitor[0].Estado == 5)
#define q (gEleitor[0].NumAssinaturas == 2)

[](p -> q)
```

Resultado da verificação

A propriedade é satisfeita pelo modelo.

4.4.6 REVS – Eleitor termina processo de votação

“O Eleitor eventualmente termina o seu processo de votação.”

Definição da propriedade

```
#define p (gEleitor[0].Estado == 5) && (gEleitor[1].Estado == 5)
                                     && (gEleitor[2].Estado == 5)
#define q  iniciarEscrutinio==1

<>    (q -> p)
```

Resultado da verificação

A propriedade é satisfeita pelo modelo.

4.4.7 REVS – Eleitor pode não terminar processo de votação

"O Eleitor pode não terminar o seu processo de votação."

Definição da propriedade

```
#define p (gEleitor[0].Estado == 5) && (gEleitor[1].Estado == 5)
                                     && (gEleitor[2].Estado == 5)
#define q  iniciarEscrutinio==1

[] (q -> p)
```

Resultado da verificação

A propriedade não é satisfeita pelo modelo.

Esta propriedade é mais forte que a anterior, mas já não é satisfeita pelo modelo. Não há portanto garantia de que todos os eleitores terminem o seu voto no modelo implementado. Esta situação deve-se ao facto do modelo incluir um processo temporizador que foi implementado para simular o término da eleição e a publicação da chave privada da mesma. Quando é efectuada uma verificação sobre todo o espaço de estados do modelo, ocorre sempre alguma simulação em que o temporizador termina antes que o Eleitor tenha finalizado o seu voto. Esta é uma situação perfeitamente possível num sistema real, principalmente no REVS onde não há garantia de que o voto chegue ao final.

4.4.8 REVS - Ordem dos votos

"A ordem de recepção do voto no Anonimizador não é sempre igual à ordem de votos recebidos."

Definição da propriedade

```
#define p (iniciarEscrutinio == 1 && selVoto >=2 && gPrimeiroEleitor == (0 + 10))
#define q (gVotoRecebido[0].IdVoto == (0 + 10))

[] (p -> q)
```

Resultado da verificação

A propriedade não é satisfeita pelo modelo.

A ordem dos votos que chegam ao Anonimizador não é a mesma pela qual os votos são enviados para o Totalizador. Em SPIN essa funcionalidade foi obtida através da passagem das mensagens recebidas no Anonimizador por um canal intermédio antes de serem enviadas para o canal onde o totalizador as deverá ler. As mensagens são retiradas de forma aleatória do canal origem e colocadas no canal destino, para serem retiradas novamente por ordem aleatória do canal destino e para serem escritas no canal onde o Totalizador lerá as mesmas. Se a mesma propriedade for verificada com:
<> (p->q), a mesma será verificada.

4.4.9 Apenas verificável em SPIN – Linear Temporal Logic

Nesta secção exemplificar-se-ão as principais possibilidades de verificação em LTL e CTL e que não são verificáveis em CTL. As fórmulas LTL são precedidas sempre pelo quantificador “∀”, que indica que são considerados sempre todos os caminhos, antes das respectivas fórmulas. Devido a esta característica, as fórmulas com quantificadores existenciais, “∃”, não são verificáveis em SPIN. Por outro lado, em SPIN podem efectuar-se verificações com fórmulas compostas que seleccionam, por exemplo, grupos de caminhos com propriedades. Então, para a propriedade:

"Sempre que o eleitor eventualmente termina o seu voto, então sempre eventualmente terá obtido todas as assinaturas necessárias para o mesmo ser válido "

Definição da propriedade

```
#define p (gEleitor[0].Estado == 5)
#define q (gEleitor[0].NumAssinaturas == 2)

[] <> p -> []<>  q
```

Resultado da verificação

Esta propriedade não é verificável em UPPALL pelas razões já descritas.

5 Verificação de protocolos de votação electrónica com UPPAAL

A ferramenta UPPAAL permite a construção de modelos abstractos para sistemas de tempo real, simular o seu comportamento e especificar e verificar propriedades dos referidos modelos [23].

A ferramenta é constituída por três partes: um editor com uma linguagem de descrição; um simulador; e um verificador de modelos para definir e validar propriedades do modelo construído.

Utilizando o editor, a linguagem permite definir o modelo do sistema como um conjunto de autómatos estendido com variáveis e que podem contem noções de tempo.

A descrição de um sistema é composta por um conjunto de *templates* de processos eventualmente com declarações locais, globais e definição do sistema.

O editor da ferramenta é usado para criar e definir o sistema a ser analisado (ver Figura 5.1).

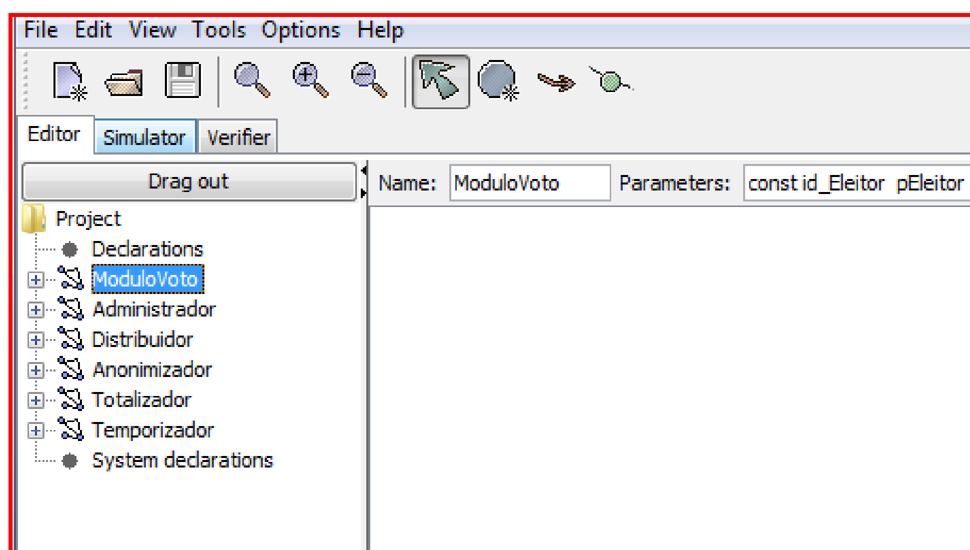

Figura 5.1 Subsistemas do UPPAAL

Na área *Declarations* declaram-se as constantes, variáveis ou estruturas de dados globais a todos os processos. Exemplos de constantes são as correspondentes aos estados do Módulo de Voto que permitem indicar o momento em que cada eleitor se encontra no seu processo de votação.

Comunicam através de variáveis globais quando ocorre a sincronização entre processos através dos canais definidos na área global de declarações. Neste caso os canais são síncronos.

Os processos são identificados por um nome e podem ter na sua declaração parâmetros. Nos processos do modelo apresentado, todos os processos excepto o temporizador têm parâmetros que irão corresponder ao número de instâncias pretendidas. Para o temporizador, é utilizado um parâmetro do tipo inteiro que é instanciado com um valor correspondente ao tempo total máximo.

A ferramenta UPPAAL permite-nos desenhar os processos de uma forma bastante visual por "drag and drop" dos elementos que compõe o desenho do processo, nomeadamente:

- Nó ou *Location*
- Arco ou *Edge*

Para cada arco existe um conjunto de configurações (ver Figura 5.2), onde se pode definir parametrizações de *selecção*, *guarda*, *sincronização* e *actualização*.

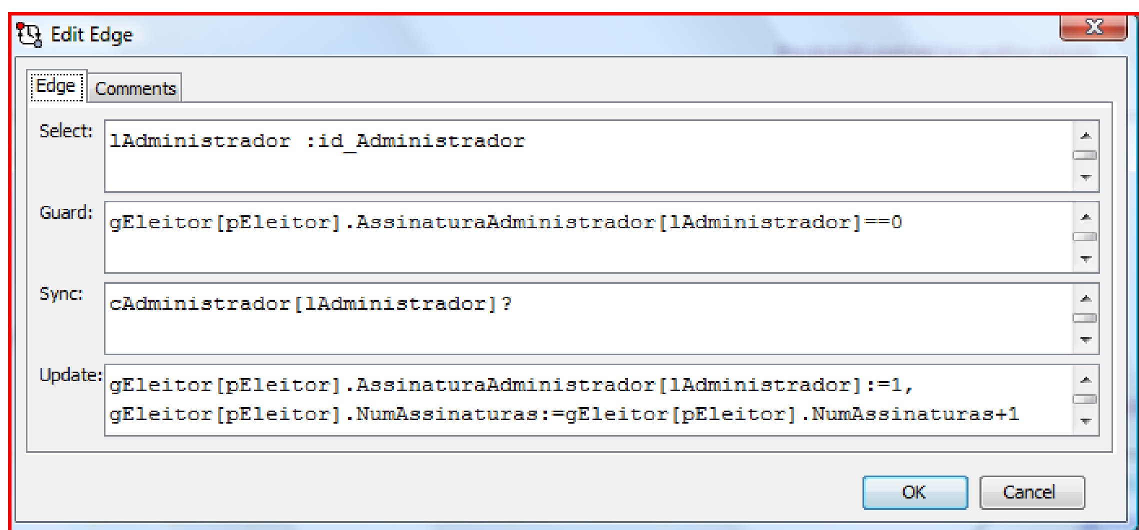

Figura 5.2 Configuração de um arco na modelação de um processo

Na figura anterior visualizamos a definição de um arco com todos os tipos de configuração:

Select

É definido o tipo de variável que vai ser utilizada neste arco. Esta variável é local ao arco e não é reconhecida noutros elementos do *template*.

Guard

É definida a guarda para o arco, ou seja, a condição que terá que ser satisfeita para que possa ocorrer a transição para o nó seguinte através deste arco.

Sync

Indica o canal em que este arco vai sincronizar e que permite a transição para o nó seguinte, caso se verifique a guarda.

Update

Caso ocorra a sincronização e se verifique a guarda, transição para o nó destino é despoletada e ocorrem as actualizações declaradas nesta área.

Para cada nó, também é possível efectuar configurações, como indica a Figura 5.3, nomeadamente para definir o tipo de nó e uma condição invariante.

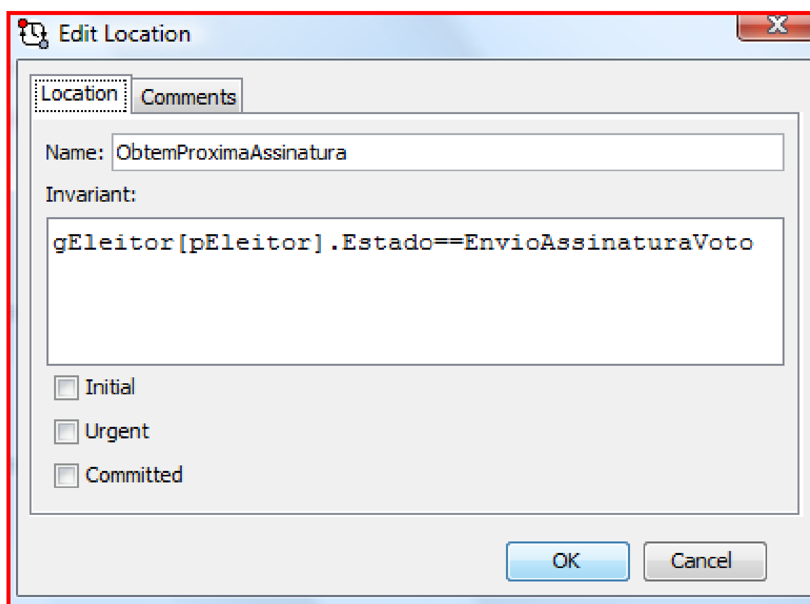

Figura 5.3 Configuração de um nó na modelação de um processo

É possível ainda incluir comentários quer nos arcos, quer nos nós.

No UPPAAL os processos sincronizam num canal e trocam valores através de variáveis partilhadas.
Olhando ainda para as figuras 5.2 e 5.3, temos uma sequência de passos constituída pela interface referente a um arco representado na figura 5.4, que permite transitar do nó anterior para o nó seguinte. Neste caso, os processos vão sincronizar lendo no canal `cAdminstrador[i]`, onde i corresponde ao índice do Administrador em questão. Como é uma operação de leitura, o valor da variável global, `gEleitor[pEleitor]`, não está disponível no momento de avaliação deste arco pelo UPPAAL, quando é efectuada a simulação. Então, a solução é colocar a condição de progressão pretendida no "invariante" do nó seguinte (figura 5.3) em vez de na guarda do arco da (figura 5.2). O resultado pretendido é o seguinte:

- Se ao sincronizar no canal `cAdminstrador[i]` a variável `gEleitor[pEleitor].Estado` tiver um valor correspondente a EnvioAssinaturaVoto, então ocorre uma transição para o nó da figura 5.3.

System Declarations

Nesta área são colocadas as instruções de instanciação do modelo.

```
ProcessoTemporizador = Temporizador(gTempoTotalEleicao);
system
ModuloVoto, Administrador, Distribuidor, Anonimizador,Totalizador, ProcessoTemporizador;
```

O processo Temporizador tem um parâmetro que corresponde a um ciclo desse temporizador, que é atribuído nesta área. Para os restantes processos apenas são indicados os seus nomes. Em tempo de execução será criada cada instância de acordo com os tipos definidos em cada parâmetro de cada processo. Para a solução implementada correspondem ao número de eleitores, distribuidores, Administradores, Anonimizadores e Totalizadores.
A secção seguinte detalha o modelo construído em UPPAAL para efectuar verificações, à semelhança do que já foi efectuado para o SPIN.

5.1 Modelação

5.1.1 Opções tomadas

Para modelar o sistema em UPPALL foram tomadas as seguintes opções que são assumidas ao longo de toda a modelação e que são muito semelhantes às já descritas para o SPIN, na Secção 4.4.1

- Os eleitores são representados como autómatos cuja instanciação pode ir desde o valor 0 até ao máximo de eleitores pretendido. São considerados normalmente apenas dois eleitores.
- Os processos são instanciados em tempo de execução. A informação que é guardada em relação a cada eleitor e ao momento da votação em que se encontra.
- São assumidos por defeito 3 Administradores. De acordo com o protocolo REVS serão portanto necessárias duas assinaturas para que um voto seja válido. No modelo UPPAAL, à semelhança do que acontece para os eleitores, foi definida uma constante com o número de Administradores e um tipo baseado nesse valor. Através da instanciação dos processos em tempo de execução são criados o número de eleitores correspondente ao tipo definido.
- Na comunicação entre Anonimizador e o Totalizador, a identificação do eleitor é dissimulada para não obter uma associação directa entre o eleitor e o voto. No entanto verificar-se-á que neste sistema essa associação será sempre possível e será uma das propriedades do sistema de votação electrónica que não será verificada.
- É permitido no máximo a recepção de 5 votos no Anonimizador e no Totalizador. Assim, poderão existir repetições de votos.
- Abstrai-se completamente a troca de chaves e encriptações para comunicação segura entre servidores.
- O temporizador representa o momento do final da eleição, isto é, quando é publicada a chave privada da eleição, sendo possível validar e contar os votos que chegam ao final. Este processo foi criado para permitir simular o decorrer de um intervalo de tempo, ao fim do qual se considera que a eleição terminou e se inicia o escrutínio final. Existem simulações onde o Temporizador termina sem que tenham sido finalizados todos processos do Módulo de Voto. Esta situação

representa um cenário onde alguém tenta submeter ou comunicar com o sistema após o fecho da eleição.

- O modelo é constituído por 6 *templates* que em tempo de execução são instanciados em 12 processos.

5.1.2 Representação do protocolo e principais estruturas utilizadas

O REVS foi representado pelos *templates* que corresponderão ao número de instâncias indicadas na tabela seguinte:

PROCESSO	# INSTÂNCIAS
ModuloVoto	2
Distribuidor	2
Administrador	3
Anonimizador	2
Totalizador	2
Temporizador	1

Tabela 5-2 UPPAAL - Processos do modelo e suas instanciações de execução

De notar que em UPPAAL existe menos uma instância do que em SPIN, já que não existe o equivalente ao processo *init* do SPIN.

Utilizando o editor do UPPAAL, elaborou-se o modelo deste sistema que é apresentado e explicado nesta secção. A totalidade do código do modelo, em formato XML, é incluída no Anexo B.

O UPPAAL permite fazer verificações sobre variáveis/canais globais e locais pelo que se poderão efectuar verificações diferentes com o SPIN e com o UPPAAL. No entanto, mesmo podendo guardar o estado de alguns elementos e efectuar verificações sobre eles localmente, optou-se por utilizar uma estrutura idêntica à já utilizada com o SPIN. Assim, nem todos os elementos do modelo importantes para a verificação são globais ao modelo. Os estados do Módulo de Voto são definidos como constantes tendo em conta as mensagens trocadas entre o Módulo de Voto e os Distribuidor ou Administradores. Estas declarações são efectuadas na área de declarações, ou seja, são comuns a todos os processos.

- A seguir são mostradas as constantes e variáveis do modelo, acompanhadas do respectivo comentário indicativo da sua função. Destacam-se as constantes que definem o estado em que o eleitor se encontra a cada momento e os tipos de dados utilizados para definir o número de instâncias para cada processo do modelo, excepto para o temporizador que tem uma instância única.

```
const int INICIO:=0;
const int PedidoBoletimEleicao := 1;
const int EnvioBoletimEleicao := 2;
const int PedidoBoletimVoto := 3;
const int EnvioBoletimVoto := 4;
const int PedidoAssinaturaVoto := 5;
const int EnvioAssinaturaVoto := 6;
const int EnvioSubmissaoVoto := 7;
```

Estes estados não são comuns aos utilizados em SPIN. Em UPPAAL reflectem o envio e a recepção de mensagens através da mudança de valores de variáveis. Em SPIN isso não é necessário, já que ao sincronizar é escrita ou recebida de imediato uma mensagem.

Através destes estados, em cada momento sabe-se em que ponto da votação o eleitor se encontra.

- O número de processos do modelo é definido através dos parâmetros dos *templates*. Cada tipo de dados representa o número máximo de processos que é instanciado automaticamente em tempo de execução, pelo que é possível fazer variar o número de processos com o mínimo de impacto no modelo:

```
const int gMaximoEleitores := 2;
typedef int[0,gMaximoEleitores-1] id_Eleitor;

const int gMaximoAdministradores := 3;
typedef int[0,gMaximoAdministradores-1] id_Administrador;

const int gMaximoDistribuidores := 2;
typedef int[0,gMaximoDistribuidores-1] id_Distribuidor;

const int gMaximoTotalizadoresAnonimizadores := 2;
typedef int[0,gMaximoTotalizadoresAnonimizadores-1] id_TotalizadorAnonimizador;
```

Como existem vários Distribuidores, Administradores e Totalizadores é efectuada uma selecção aleatória em cada simulação e verificação do modelo distribuído.

Apresenta-se de seguida outras constantes ou variáveis que são necessárias para controlar as simulações e verificações.

- Constante que permite alterar a identificação do eleitor quando o voto é submetido ao totalizador

```
const int DELTA:=10;
```

- Constante que permite definir o número máximo de votos que podem ser recebidos. Deverá ser sempre superior ao número de eleitores, já que cada eleitor pode submeter mais do que uma vez o seu voto.

```
const int gMaximoEscrutinio := 5;
typedef int[0,gMaximoEscrutinio-1] id_MaximoEscrutinio;
```

- No Anonimizador, quando é recebido um voto, este não é enviado automaticamente, podendo um voto recebido posteriormente ter sido enviado primeiro. Assim os votos podem chegar ao final com ordem diferente da ordem da submissão ao Anonimizador. O atraso dos votos no anonimizador é efectuado através de uma variável local do tipo *clock* que permite definir um tempo de espera e enviar o voto só após esse tempo ter decorrido. Nas simulações manuais nem sempre é possível ver esse atraso, mas este acontece efectivamente para algumas simulações e nas verificações.

```
const int gAtrasovotos :=5;
```

- Tempo base de um ciclo do temporizador e número de vezes que este será reiniciado. Cada vez que o temporizador é reiniciado, o tempo que decorre corresponde a `gTempoTotalEleicao`. Decorrerão vários ciclos, contabilizados pelo valor de `gNPeriodosTempoDecorrer`, de forma a permitir que pelo menos em alguns casos a votação termine após terem sido recebidos todos os votos. É incluído um valor máximo parametrizável correspondente a `gNPeriodosTempoDecorrer` que indica quantas vezes o Temporizador será reiniciado. O tempo corrente contabilizado pela variável `gNPeriodosTempoCorrente` será incrementado na execução dos processos. Além disso em cada momento pode verificar-se se o temporizador está activo ou não. Caso não esteja, é reiniciado até que tenham decorrido o número máximo de períodos de tempo.

```
const int gTempoTotalEleicao := 500;
```

```
const int gNPeriodosTempoDecorrer :=20;

int gNPeriodosTempoCorrente :=0;

bool Activo:=false;
```

- Variável que indica o início do escrutínio.

```
bool gFimVotacao := false;
```

- O cálculo de escrutínio é incluído como uma função no processo Totalizador, sendo executado apenas para um dos totalizadores considerado o *Master*.

```
/*Fazer a contagem dos votos recebidos, contando apenas os válidos e eliminando
repetições*/
void escrutinio()
{
int indexR, indexA := 0;
while ( indexR<  gMaximoEscrutinio)
       {if     ((gVotoApurado[indexA].IdVoto==0) and (gVotoRecebido[indexR].IdVoto !=
0))
              {gVotoApurado[indexA].IdVoto := gVotoRecebido[indexR].IdVoto;
              gVotoApurado[indexA].NumAssinaturas =
gVotoRecebido[indexR].NumAssinaturas;
              indexA:=0;
              indexR++;}
       if      ((gVotoApurado[indexA].IdVoto !=0) and (gVotoRecebido[indexR].IdVoto !=
0) and (gVotoRecebido[indexR].IdVoto) != (gVotoApurado[indexA].IdVoto ))
              indexA++;
       if      (gVotoRecebido[indexR].IdVoto==0)
              indexR++;
}
}
```

- Além disso, estão também definidas outras estruturas que permitem guardar outros tipos de informação relevantes para o modelo, tais como, a informação do primeiro eleitor, número de assinaturas que cada eleitor já tem e a que administradores correspondem. Guardam também informação sobre o estado em que se encontra o processo de votação e os votos recebidos/apurados consoante seja antes ou após o término da eleição:

```
int gPrimeiroEleitor;

int gNumAssAdministradores := gMaximoAdministradores /2+1;

int iVotosRecebidos;

typedef struct {
int Estado;
int NumAssinaturas;
bool AssinaturaAdministrador[3];
}recEleitor;

typedef  struct{
int IdVoto;
int NumAssinaturas;
```

```
}recVoto;
recEleitor gEleitor[gMaximoEleitores];
recVoto gVotoRecebido[id_MaximoEscrutinio];
recVoto gVotoApurado[id_MaximoEscrutinio];
```

- Para a comunicação entre os processos, utilizaram-se *arrays* globais de canais, um para cada eleitor.

```
chan cEleitor[gMaximoEleitores];
chan cDistribuidor[gMaximoDistribuidores];
chan cAdministrador[gMaximoAdministradores];
chan cAnonimizador[gMaximoEleitores];
chan cTotalizador[gMaximoTotalizadoresAnonimizadores];
chan Atribuir;
```

Os canais em UPPAAL permitem que os processos envolvidos avaliem os estados dos nós e das guardas para transitar para o estado seguinte.

5.1.3 Descrição dos Processos

Módulo de Voto

O processo do Módulo de Voto é instanciado para cada eleitor do modelo. Este processo inicia a execução do sistema. Começa no estado INICIO que corresponde ao primeiro nó e de seguida solicita um boletim de eleição, alterando na mesma operação de forma atómica o seu estado. O sistema é iniciado escrevendo no seu canal de comunicação um pedido de sincronização e indicado no seu estado que pretende um boletim de eleição. De seguida, aguarda no canal de um dos distribuidores a recepção do Boletim de Eleição. Os passos de comunicação repetem-se até o eleitor obter o seu boletim de voto. Após isso, solicita então as assinaturas aos Administradores até obter o total de assinaturas necessário para que o seu voto seja validado. Quando obtém todas as assinaturas, submete o seu voto ao Anonimizador sincronizando para escrita no canal de um dos Anonimizadores. Note-se que após esta operação o processo do Módulo de Voto chega ao seu estado final, pelo que não se poderá saber efectivamente se o voto chegou ou não ao seu destino. Por isso mesmo o protocolo permite que o eleitor submeta várias vezes o seu voto.

O modelo completo do *template* é apresentado na figura seguinte:

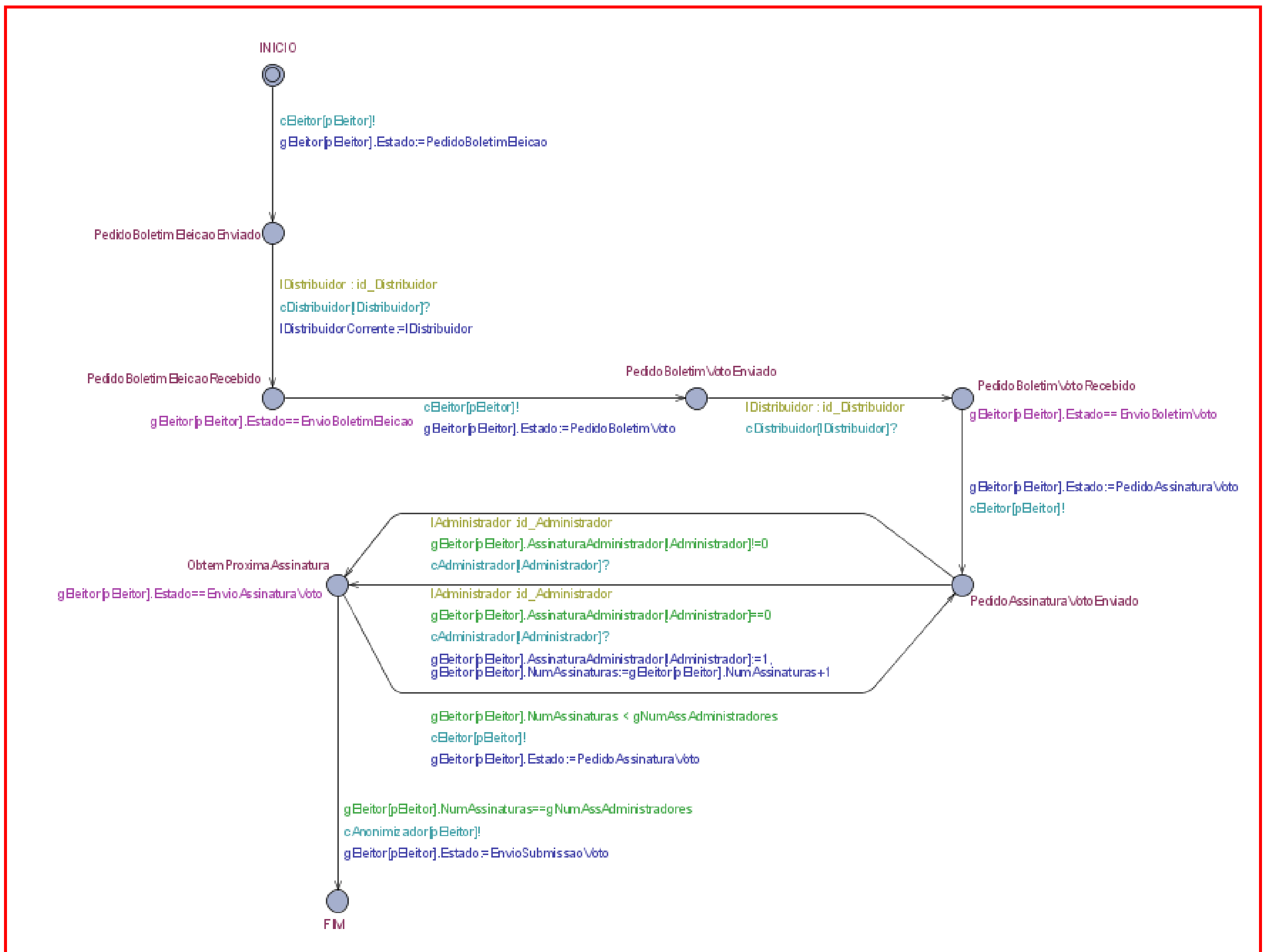

Figura 5.4 Processo ModuloVoto

Distribuidor

O processo Distribuidor encontra-se continuamente em ciclo e sincroniza com qualquer um dos eleitores. Consoante receba um Pedido de Boletim de Eleição ou Pedido de Boletim de Voto, devolve o respectivo Boletim de Eleição ou Boletim de Voto. Este processo originará uma sequência cíclica que não termina, ocorrendo um *deadlock*, mas que poderá ser considerada como válida pois corresponde ao funcionamento pretendido no modelo em causa. Este processo aguarda ciclicamente contactos do processo ModuloVoto para responder com um boletim de eleição ou com um boletim de voto.

O processo Distribuidor é mostrado na imagem seguinte:

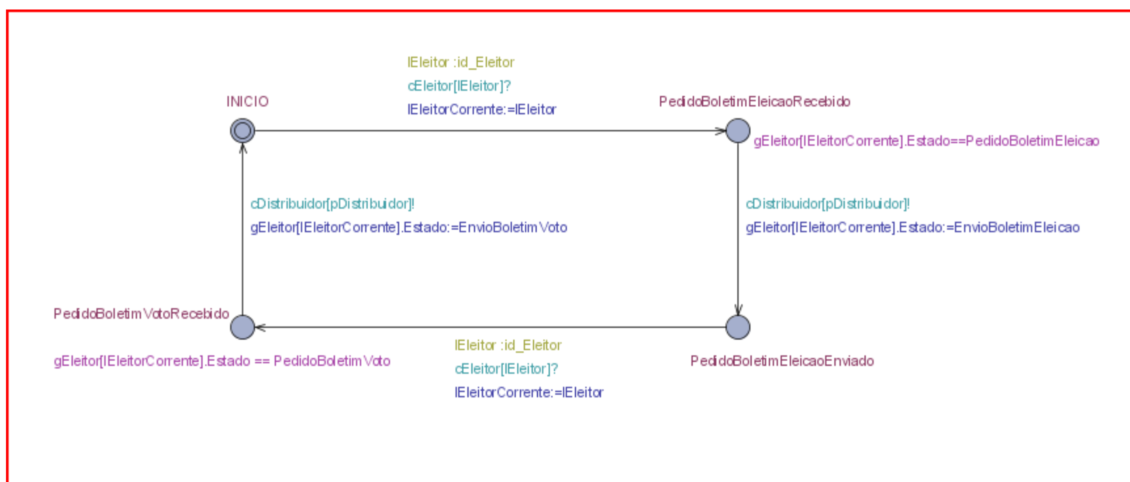

Figura 5.5 Processo Distribuidor

Administrador

O processo Administrador, à semelhança do processo Distribuidor, encontra-se continuamente em ciclo e caso receba um Pedido de Assinatura de Voto, simplesmente sincroniza no canal respectivo e actualiza a variável global correspondente com o novo estado do eleitor. Este processo originará uma sequência cíclica que não termina, originando um *deadlock,* mas que é considerada válida. A imagem seguinte detalha o autómato definido para o Administrador.

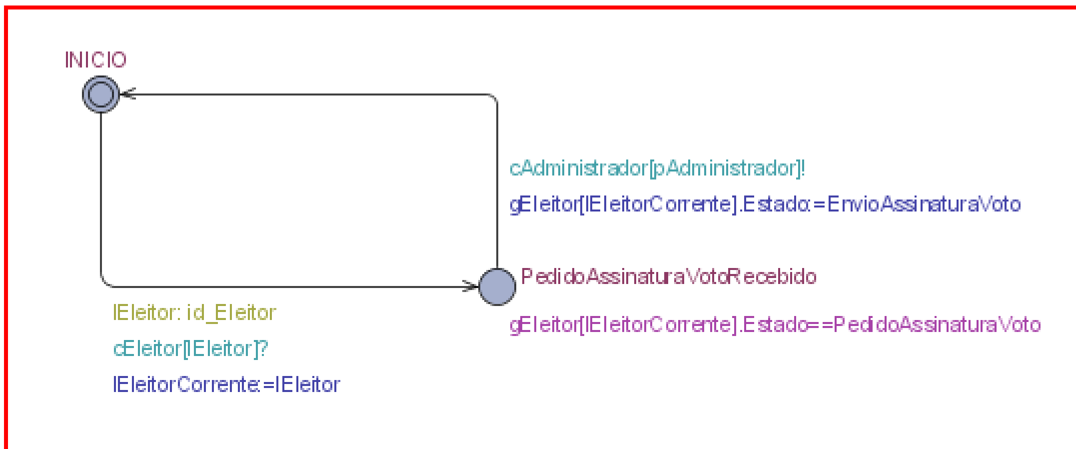

Figura 5.6 Processo Administrador

Anonimizador

Este processo recebe os votos vindos do Módulo de Voto e provoca um atraso aleatório nos votos, bem como altera a identidade do voto, para que o Totalizador não possa identificar o eleitor.

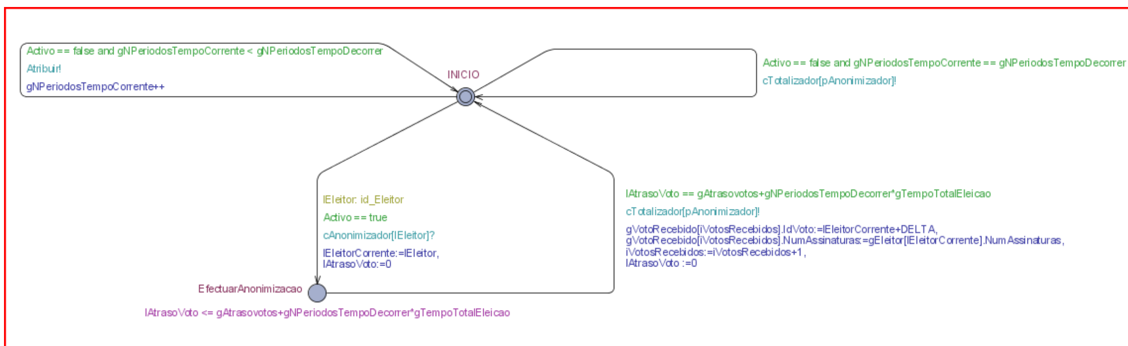

Figura 5.7 Processo Anonimizador

O Totalizador

O Totalizador receberá ciclicamente mensagens do Anonimizador enquanto não tiver começado o escrutínio. Quando o escrutínio se iniciar, o que é determinado através da variável `iniciarEscrutinio`, o Totalizador em questão que pode ser qualquer um dos disponíveis. Iniciará então o processo de escrutínio apurando os votos que são válidos e transferindo a informação do *array* `gVotoRecebido` para `gVotoApurado`. Neste apuramento só serão considerados os votos que não tenham duplicações e que tenham o número de assinaturas correcto.

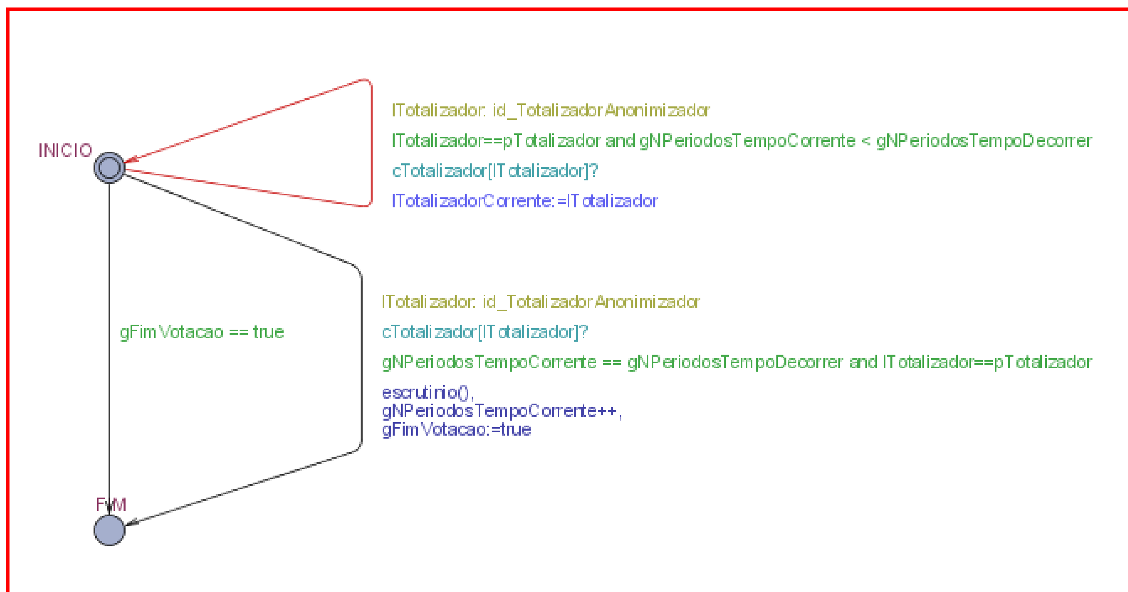

Figura 5.8 Processo Totalizador

Temporizador

O processo temporizador permite que, após algum tempo decorrido, seja actualizada a variável correspondente ao momento final da eleição denominada `iniciarEscrutinio`. Quando essa variável tiver o valor 1, significa que deverá ser efectuado o escrutínio, ou seja, num dos processos Totalizador escolhido para o efeito são retiradas as repetições de votos recebidos, `gVotoRecebido[5]`, verificados os votos que têm o número de assinaturas válido e são guardados para contagem final. Iremos depois definir propriedades sobre este repositório de votos, `gVotoApurado[5]`. O momento de início da contagem dos votos representa na especificação o momento da publicação da chave privada da eleição e o acesso aos dados da votação. Apesar de o UPPAAL permitir o conceito de tempo, apenas foi utilizado um processo Temporizador que consoante as simulações poderá ter maior prioridade que os outros processos e que poderá terminar sem que tenham sido efectuadas todas as votações. No entanto esse é um cenário possível, mesmo num sistema real.

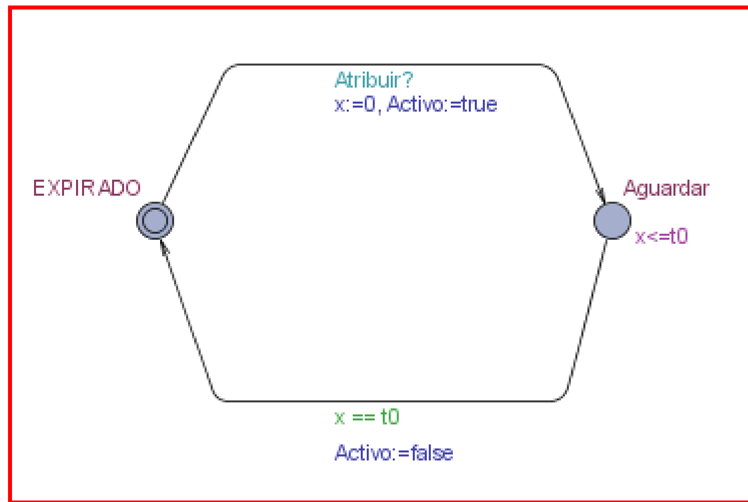

Figura 5.9 Processo Temporizador

5.2 Simulação

Após modelação, ao passar para a área de verificação é efectuada a validação de sintaxe, após a qual surge o ecrã de simulação.

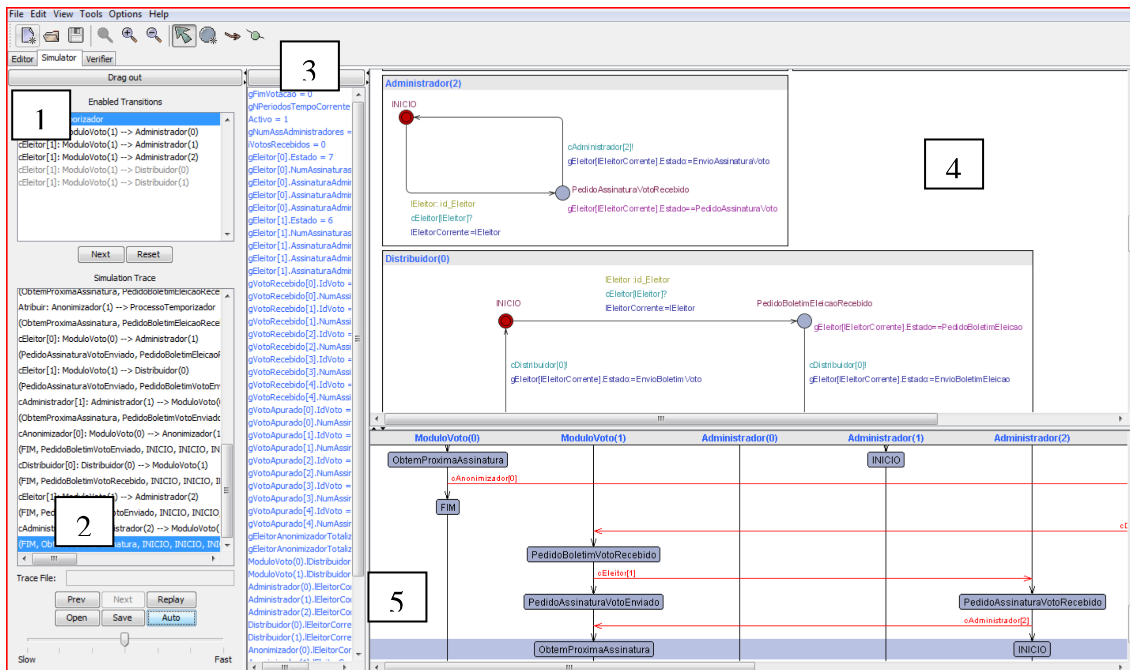

Figura 5.10 Janela de simulação do SPIN

Na simulação, existem 5 áreas:

1. Área onde se encontram os processos e as transições entre eles. A tonalidade mais escura são mostrados os que têm transições possíveis, em tonalidade mais clara os que não têm transições possíveis nesse momento.

2. Quando ocorre uma transição, é visível nesta área o estado anterior e o estado final após a transição.
3. Nesta área pode-se visualizar o estado de todas as variáveis globais e locais do modelo.
4. Nesta área visualizam-se todas as instâncias dos processos em execução.
5. Nesta área visualizam-se os fluxos de controlo de processos, ou seja o esquema de mensagens correspondentes, já que em cada sincronização a informação é trocada entre os processos através das suas variáveis.

De notar que quando se executa uma simulação, pode visualizar-se em simultâneo:

- A alteração dos valores das variáveis.
- Qual foi o processo que sincronizou.
- Qual o seu estado antes da transição e após a transição.
- Visualizar a transição na instância do processo visualizada na área 4.

À semelhança do que acontece no SPIN, simulações diferentes têm transições diferentes, por ordens diferentes. Este ecrã também pode ser utilizado quando se obtém uma verificação que falha, para seguir o rasto da execução e ver em que condições isso acontece.
Na secção seguinte serão apresentadas em linguagem corrente algumas propriedades.

5.3 Verificação de propriedades genéricas

O UPPAL permite verificar vários tipos de propriedades, algumas mais genéricas e aplicadas a qualquer modelo de u sistema votação electrónica, outras específicas e relacionadas com protocolo em causa.
A figura 5.11 mostra algumas propriedades já definidas. Quando uma propriedade é verificada com sucesso, esta fica com um indicador a verde visualizando-se na área de *status* a mensagem “Property is satisfied”. Quando falha, fica vermelho, visualizando-se na área de *Status* a mensagem “Property is not satisfied”. A propriedade marcada na página seguinte indica que o sistema está numa situação de *deadlock* apenas quando o apuramento da votação já iniciou, ou seja, terminou o tempo disponível para realizar a eleição. Neste caso, não existem mais transições possíveis para o estado actual do

sistema, precisamente porque o temporizador já terminou. É portanto uma situação de *deadlock* já esperada.

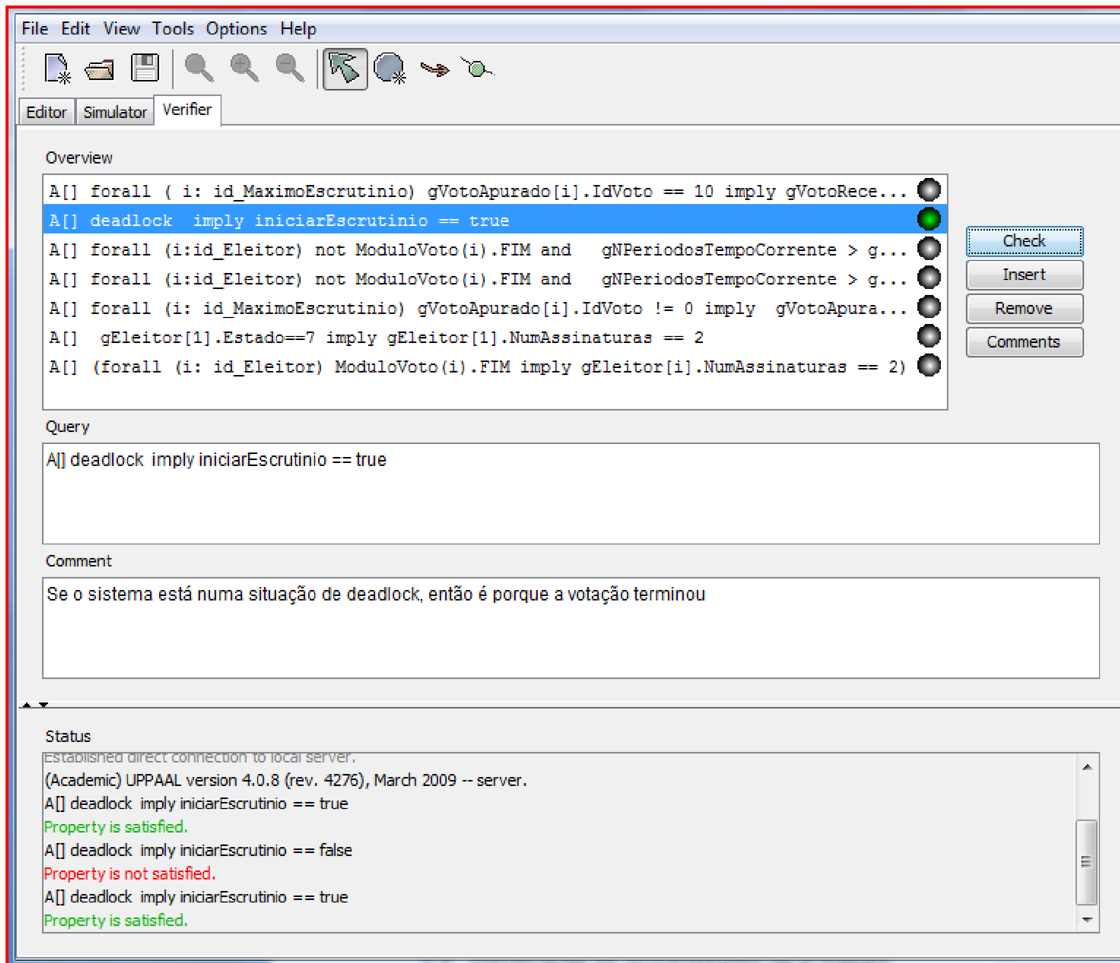

Figura 5.11 Resultado de uma verificação

Pode gravar-se as propriedades para mais tarde se voltar a utilizar as mesmas. Ver Secção 3.2 para a notação utilizada em CTL.

5.4 Verificação de propriedades de sistemas de votação electrónica

5.4.1 Privacidade

"Não é possível estabelecer uma ligação entre o eleitor e o seu voto."

Há uma propriedade que será verificada a seguir na Secção 5.4.3 que tem a ver com a precisão e que acaba por ser contraditória com as propriedades de privacidade:

```
A[] forall (i: id_MaximoEscrutinio) gVotoApurado[i].IdVoto != 0
imply  forall (j: id_MaximoEscrutinio) gVotoRecebido[i].IdVoto == 0+DELTA or
gVotoRecebido[i].IdVoto  == 1+DELTA
```

Resultado da verificação

A propriedade não é satisfeita pelo modelo. Ver Secção 4.4.1.

5.4.2 Democracia

Definição da propriedade

"Só os eleitores elegíveis votam e uma única vez."

```
A[]forall (i:id_MaximoEscrutinio) gVotoApurado[i].IdVoto == 1 + DELTA  or
gVotoApurado[i].IdVoto == 1+ DELTA   or gVotoApurado[i].IdVoto == 1+ DELTA  or
gVotoApurado[i].IdVoto or gVotoApurado[i].IdVoto == 0
```

Resultado da verificação

A propriedade é satisfeita pelo modelo. Foi efectuada a verificação para os três eleitores que estão definidos no modelo. É incluída uma condição para excluir os estados em que o voto só seja efectuado após o término da eleição.

5.4.3 Precisão

Definição da propriedade

"Se um voto for efectuado pelo eleitor, ele deverá ser contado no escrutínio final".

```
A[] forall (i: id_MaximoEscrutinio) gVotoApurado[i].IdVoto != 0
imply  forall (j: id_MaximoEscrutinio) gVotoRecebido[i].IdVoto == 0+DELTA
or gVotoRecebido[i].IdVoto  == 1+DELTA
```

Resultado da verificação

A propriedade é satisfeita pelo modelo.

5.4.4 Fiabilidade

Ver secção 4.4.4.

5.4.5 REVS – Assinaturas para o voto ser válido

"Um eleitor que termine o seu processo de voto garantidamente obteve um número de assinaturas suficiente para o voto ser válido."

Definição da propriedade

```
A[] (forall (i: id_Eleitor) ModuloVoto(i).FIM
       imply gEleitor[i].NumAssinaturas == gNumAssAdministradores)
```

Resultado da verificação

A propriedade é satisfeita pelo modelo.

5.4.6 REVS – Eleitor termina processo de votação

"O Eleitor eventualmente termina o seu processo de votação."

Definição da propriedade

```
A<>  iniciarEscrutinio==1 imply  forall (i:id_Eleitor) ModuloVoto(i).FIM
```

Resultado da verificação

A propriedade é satisfeita pelo modelo.

5.4.7 REVS – Eleitor pode não terminar processo de votação

"O Eleitor pode não terminar o seu processo de votação."

Definição da propriedade

```
A[]iniciarEscrutinio==1 imply  forall (i:id_Eleitor) ModuloVoto(i).FIM
```

Resultado da verificação

A propriedade não é satisfeita pelo modelo.

Ver secção 4.4.7.

5.4.8 REVS e UPPAAL – Ordem dos votos com tempo

"A ordem de recepção do voto no Anonimizador não é sempre igual à ordem de votos recebidos."

Definição da propriedade

```
A[] iniciarEscrutinio ==1 and iVotosRecebidos== gMaximoEleitores and gPrimeiroEleitor==1
+DELTA imply gVotoRecebido[0].IdVoto == 1+DELTA
```

Resultado da verificação

A propriedade não é satisfeita pelo modelo.

Esta propriedade é assegurada pelo relógio local que cada processo Anonimizador possui. Este relógio assegura que existem simulações onde a ordem dos votos é trocada comparativamente à ordem de recepção dos mesmos no Anonimizador. A variável do tipo *clock* utilizada faz com que em alguns casos os votos demorem mais a chegar ao Totalizador. Se a mesma propriedade for verificada com `A<>`, a mesma será verificada.

Neste caso significará que para todos os caminhos, existem alguns estados onde a ordem do voto não é trocada.

5.4.9 REVS – Lógica Temporal Ramificada

Nesta secção exemplificar-se-á as principais possibilidades de verificação em UPPAAL. Os exemplos 3 e 4 a seguir indicados são apenas possíveis em UPPAAL visto que utilizam o quantificador “∃” que indica que a propriedade não é verificada para todos os caminhos. São possíveis quando existe algum caminho onde a propriedade é válida, a partir do estado corrente. Por isso mesmo, as propriedades referidas não são verificáveis em SPIN.

A verificação destas propriedades efectuar-se-á por comparação, mudando os operadores e também as proposições de acordo com a propriedade.

1. "Quando o escrutínio inicia, já todos os votos chegaram ao Anonimizador e Totalizador."
2. "Quando o escrutínio inicia, eventualmente já todos os votos chegaram ao Anonimizador e Totalizador."
3. “Existe um caminho onde o voto do eleitor é sempre recebido e apurado.”
4. “É possível que exista um caminho onde o voto do eleitor é eventualmente recebido e apurado.”

Definição das propriedades

```
1. A[] iniciarEscrutinio ==1 imply forall (i:id_Eleitor) gVotoRecebido[i].IdVoto !=
   0
2. A<> iniciarEscrutinio ==1 imply forall (i:id_Eleitor) gVotoRecebido[i].IdVoto !=
   0
3. E[] forall (i:id_Eleitor) gVotoRecebido[i].IdVoto != 0 and gVotoApurado[i].IdVoto
   != 0
4. E<> forall (i:id_Eleitor) gVotoRecebido[i].IdVoto != 0 and gVotoApurado[i].IdVoto
   != 0
```

Resultado da verificação

Quando utilizamos fórmulas do tipo E<> ou E[], ou seja, quando é utilizado o quantificador sobre os caminhos a serem considerados, não são gerados contra-exemplos quando a propriedade falha mas apenas exemplos que indicam em que condições a propriedade é verificada.

1. A propriedade não é válida e é gerado um contra-exemplo em que é executado apenas o Temporizador com o Anonimizador e onde a eleição termina sem que algum eleitor tenha votado.

2. A propriedade é verificada.
3. A propriedade é verificada apenas em UPPAAL.
4. A propriedade é verificada apenas em UPPAAL.

6 Comparação de SPIN e UPPAAL na verificação de protocolos de votação electrónica

Em ambas as ferramentas, após modelação principal teve que existir um conjunto de refinamentos de modo a conseguir fazer verificações de uma forma aceitável, ou seja, de forma a termos capacidade de memória sem comprometer as conclusões tiradas.

Para o SPIN, por exemplo, quando se efectuava uma validação com 3 eleitores, obtinha-se sempre problemas de falta de memória.

As verificações foram portanto efectuadas com três ou dois eleitores, em função das capacidades de memória.

A seguir apresenta-se um quadro resumo da comparação das ferramentas SPIN e UPPAAL nas várias vertentes utilizadas no decorrer deste trabalho.

Característica	SPIN	UPPALL
Facilidade de utilização	Inferior ao UPPAAL: desenho do modelo por linguagem de programação Promela implica maior abstracção	Superior ao SPIN: desenho do modelo de forma gráfica clicando e arrastando com o rato, com visualização imediata do modelo.
Conceito de tempo	Não	Sim, mas apenas entre autómatos. Não tem conceito global de tempo para todos os autómatos.
Sincronização	Por leitura/escrita em canais de comunicação (mensagens). Pode ser síncrona ou assíncrona	Por leitura/escrita em variáveis globais (guardas de nós e/ou de arcos) quando ocorre uma sincronização. Pode ser síncrona ou assíncrona.

<table>
<tr><th>Característica</th><th>SPIN</th><th>UPPALL</th></tr>
<tr><td>Capacidade</td><td colspan="2">Possível efectuar verificações com modelo equivalente, originando problemas de memória em qualquer dos casos. Por exemplo ao aumentar um eleitor, ou seja, utilizando dois ou três eleitores.
Possível efectuar verificações com modelo equivalente, originando problemas de memória em qualquer dos casos por exemplo ao aumentar um eleitor, ou seja, utilizando dois ou três eleitores.</td></tr>
<tr><td>Verificação de propriedades (Gravar propriedades)</td><td>Sim, incluindo os resultados da verificação.</td><td>Sim, mas apenas as propriedades a verificar, não incluindo os resultados da verificação.</td></tr>
<tr><td>Verificação de propriedades (estruturas dados)</td><td>Globais</td><td>Utilizando estruturas de dados globais e locais.</td></tr>
<tr><td>Verificação de propriedades (Geração de contra-exemplo com possibilidade de debug)</td><td>Sim.</td><td>Sim, apesar de ser mais intuitivo e de mais simples utilização nesta ferramenta.</td></tr>
<tr><td>Mensagens de erro quando se utiliza a ferramenta</td><td>Mais genéricas, o que dificulta a detecção da causa do erro.</td><td>Mais precisas, permitindo identificar mais facilmente o erro em questão.</td></tr>
<tr><td>Formalização de propriedades para verificação</td><td>Não permite utilizar constantes pré-definidas.</td><td>Permite utilizar constantes pré-definidas.</td></tr>
</table>

Tabela 6-1 Comparação das ferramentas SPIN e UPPAAL

O quadro seguinte apresenta um resumo das propriedades de votação electrónica implementadas e verificadas no decorrer deste trabalho.

PROPRIEDADE	SPIN	UPPAL
Liveness	Sim, quando se utiliza o eventualmente (<>)	Sim, quando se utiliza o eventualmente (A<>)
Deadlock/ Invalid end states	Sim, *Invalid end states*	Sim, *deadlock*
Privacidade	Sim	Sim
Democracia	Sim	Sim
Precisão	Sim	Sim
Fiabilidade	Sim	Sim
Específicas REVS – Assinaturas para o voto ser válido	Sim	Sim
Específicas REVS – Eleitor termina processo de votação	Sim	Sim
Específicas REVS – Eleitor pode não terminar processo de votação	Sim	Sim
Específicas REVS – Ordem dos votos	Sim	Sim
Apenas verificável em SPIN – Linear Temporal Logic	Sim	Não

Tabela 6-2 Comparação das verificações em SPIN e UPPAAL

7 Intrusos

Um intruso é um elemento que tenta subverter os resultados de uma votação, acedendo a informação privilegiada e tentando comprometer a segurança do sistema, podendo causar danos a diversos níveis, como por exemplo, na integridade, fiabilidade ou precisão.

O protocolo REVS assenta num modelo misto de encriptação utilizando chaves simétricas e assimétricas. Nas chaves simétricas, apenas quem encripta conhece a chave. Um exemplo de chave simétrica no REVS é a chave utilizada para cifrar o voto antes da submissão do mesmo ao Anonimizador. Por outro lado, a chave pública ou assimétrica é conhecida por todos os elementos que dela necessitam para interagir com o sistema e pode ser distribuída. Para desencriptar informação encriptada com a chave pública é necessária a chave privada. Só esta pode ser utilizada para ter acesso a informação cifrada com a chave pública.

Assume-se no âmbito dos modelos definidos para o SPIN e UPPAAL que as chaves referidas são invioláveis pois a sua verificação daria por si só matéria para um estudo específico nessa área.

Uma vez que as mensagens são encriptadas com chaves simétricas ou assimétricas e partindo do princípio que a encriptação não pode ser quebrada, concentrar-nos-emos apenas noutras formas possíveis de intrusão. Para tal partimos do princípio que o intruso pode ter acesso ao sistema e que pode conhecer tudo o que os outros elementos podem conhecer, excepto as chaves privadas. Por exemplo, sendo estas conhecidas poderá ser possível construir mensagens que alterem o bom funcionamento do sistema de votação electrónica. As chaves não foram portanto representadas no modelo pois é possível a sua abstracção. O intruso poderá também interceptar mensagens que não lhe são destinadas.

O modelo Dolev-Yao é um dos modelos de referência em relação a intrusão [20, 31, 32]. O modelo de intrusão Dolev-Yao representa um atacante que consegue observar,

interceptar e alterar qualquer mensagem, não podendo corromper apenas a parte salvaguardada pela encriptação. No entanto, pode ter acesso a qualquer chave pública de encriptação e utilizá-la para encriptar ou desencriptar mensagens ou ainda gerar novas mensagens.

Baseado nos pressupostos do modelo atrás referido, construiu-se um modelo em SPIN onde um intruso tenta submeter votos repetidos utilizando as credenciais de outro eleitor.

O modelo do intruso é a seguir apresentado:

```
proctype Intruso()
{
	int eleitor, assinaturas, anonimizador;

	end:
	do
	:: iniciarEscrutinio == 0 -> gEleitorAnonimizador?<eleitor,assinaturas,
	                                          anonimizador>;
	                       gDelayAnonimizadorTotalizador!eleitor,assinaturas,
	                                          anonimizador;
	od
}
```

Assim, com este exemplo poderá ser verificada a seguinte propriedade:

Definição da propriedade

“Apenas um voto por eleitor será contado”

Quando não há repetições nos eleitores

```
#define p (gVotoApurado[2].IdVoto == 0)
#define q (indexR == selVoto)

[] (q -> p)
```

Resultado da verificação

A propriedade é satisfeita pelo modelo, quando há repetições de votos no *array* `gVotoApurado`. O índice 2 do *array* `gVotoApurado`, para uma simulação com 2 eleitores, já não deverá estar preenchido porque foram eliminadas todas as repetições.

A condição `indexR == selVoto` permite que apenas seja considerado quando já foi efectuado o escrutínio. Estão excluídas, por exemplo, as situações relacionadas com o temporizador em que não votam todos os eleitores.

Por outro lado a propriedade verificada na secção 4.4.2 continua a ser válida mas teve que ser alterada em relação ao original, com a inclusão de mais um contador `eleitoresContados`. A razão da alteração é que com a inclusão de votos repetidos, não

se sabe quantas repetições vão ocorrer, já que as mesmas são aleatórias. Através deste contador consegue saber-se quantos votos são apurados na totalidade e validar a propriedade.

É verificada portanto a propriedade:

```
#define p ((gVotoApurado[0].IdVoto == 10) ^ (gVotoApurado[1].IdVoto == 10) ^
(gVotoApurado[2].IdVoto == 10)) && ((gVotoApurado[0].IdVoto == 11) ^
(gVotoApurado[1].IdVoto == 11) ^ (gVotoApurado[2].IdVoto == 11)) &&
((gVotoApurado[0].IdVoto == 12) ^ (gVotoApurado[1].IdVoto == 12) ^
(gVotoApurado[2].IdVoto == 12))
#define q (indexR == selVoto && eleitoresContados == 3)

[] (q -> p)
```

O resultado da verificação é incluído no anexo no Anexo C.

8 Conclusões e trabalho futuro

A utilização do SPIN e UPPAAL apresenta vantagens e desvantagens. Como principais vantagens do SPIN podemos apontar a leitura e escrita nos canais de comunicação, que permite generalizar mais facilmente o modelo. Uma desvantagem do SPIN consiste no facto de só permitir verificações sobre variáveis globais no modelo e não permitir a utilização de constantes. Em relação ao UPPAALL a maior vantagem é o facto de ser uma ferramenta visual e intuitiva e facilitar a depuração de código. A sua maior desvantagem é o facto de não permitir guardar em ficheiro o resultado da verificação em conjunto com as definições das propriedades.

Um dos maiores problemas identificados no decorrer do trabalho está relacionado com a solução do Temporizador implementada para ambos os modelos. Pretendia-se uma forma de iniciar o escrutínio e esse evento teria que ser despoletado passado algum tempo. Esse conceito de tempo não existe em SPIN mas existe em UPPAAL. Apesar disso a solução encontrada acabou por ser idêntica em ambas as ferramentas. Incluir o tempo em todos os processos do UPPAAL, com conceito disponível na ferramenta, iria aumentar substancialmente a complexidade do modelo, sem grande benefício adicional.

Outra condicionante está relacionada com o que vai ser representado no modelo a verificar. Esta decisão tem impacto no que se pode verificar e no desempenho da verificação em ambas as ferramentas.

A análise efectuada neste documento parte do princípio de que existem sessões a decorrer entre o Módulo de Voto e Distribuidor e Administradores. Ou seja, na realidade representa uma sessão simplificada do REVS na medida em que o Modulo de Voto comunica com vários servidores, mas não inicia um processo num servidor e termina noutro. Um cenário onde isso ocorresse seria muito mais complexo de representar. Visto o REVS ser uma solução tolerante a falhas, este seria um modelo interessante de analisar. Compreenderia, por exemplo, analisar cenários em que o Eleitor inicia o processo de votação com um servidor, mas por qualquer motivo, o

servidor deixa de estar disponível e o processo terá que terminar com outro servidor. A coerência destas comunicações e a gestão dos estados de cada processo por si só seriam interessantes de validar com ferramentas de verificação. Esse trabalho teria que, além de manter as sessões do lado do Módulo de voto, manter o estado do voto do lado dos Distribuidores e Administradores para poder ser possível responder adequadamente em caso de falha de sessão e consequente recuperação por outro servidor.

A juntar à sugestão indicada, seria interessante incluir em cada Administrador quais os votos assinados, para cada eleitor. Em face desta representação poder-se-iam efectuar verificações adicionais com outros tipos de intrusos que, por exemplo, enviassem assinaturas inválidas, ou referentes a outros eleitores.

Constatou-se que para efectuar determinado tipo de verificações seria necessário ter diversas variantes do modelo, tal como exemplificado na Secção 7 em relação ao intruso. Na exploração destas vertentes provavelmente seria necessário abstrairmo-nos de outros detalhes do modelo para conseguir gerir a verificação com sucesso de outras propriedades.

A utilização maliciosa de um sistema pode ser validada pelas ferramentas de verificação, de forma a identificar comportamentos que podem ser problemáticos. Estes podem não ter sido pensados, nem identificados no desenho inicial das soluções normalmente direccionadas para o comportamento correcto. É possível ver o que acontece a um protocolo com tipos de utilização fora dos comportamentos inicialmente esperados.

A explosão de estados e alguns problemas de usabilidade podem ser ainda factores condicionantes de uma utilização mais alargada deste tipo de soluções, mas mesmo assim não deixam de ter um contributo muito valioso no processo de desenvolvimento de software.

Referências

[1] Rui Joaquim, André Zúquete, Paulo Ferreira. *REVS- A Robust Electronic Voting System.* In *IADIS - International Journal of WWW/Internet*, December, 2003.

[2] Brandon William DuRette. *Multiple Administrators for Electronic Voting. BsC Thesis*, MIT, Cambridge, MA,1999.

[3] Mark A. Herschberg. *Secure Electronic Voting Over the World Wide Web. Master's Thesis, Department of Electrical Engineering and Computer Science, MIT*, 1999.

[4] Website oficial da ferramenta SPIN, http://spinroot.com/spin/whatispin.html

[5] Website oficial da ferramenta UPPAAL, http://www.uppaal.com/.

[6] Orhan Cetinkaya, Deniz Cetinkaya. *Verification and Validation Issues.* In *Electronic Journal of e-Government. Volume 5 Issue* 2, 2007.

[7] Orhan Cetinkaya, Deniz Cetinkaya. *Towards Secure E-Elections in Turkey: Requirements and Principles.* In *International Workshop on Dependability and Security in e-Government (DeSeGov'07) - In Proceedings of ARES'07*, Vienna, Austria, pp. 903-907, 2007.

[8] Lorrie Faith Cranor, Ron K. Cytron. *Sensus: A Security-Conscious Electronic Polling System for the Internet.* In *Proceedings of the Hawaii International Conference on System Sciences*, 1997.

[9] Atsushi Fujioka, Tatsuaki Okamoto, Kazuo Ohta. In *A Practical Secret Voting Scheme for Large Scale Elections.* Proc. of Advances In *Cryptology – AUSCRYPT '92*, Queensland, Australia, LNCS 718, pp 244-25, 1992.

[10] Orhan Cetinkaya, Deniz Cetinkaya. *Towards Secure E-Elections in Turkey: Requirements and Principles.* In *International Workshop on Dependability and Security in e-Government*, Vienna, Austria, pp903-907, April, 2007.

[11] Filipe Simões, Pedro Antunes. *Auditoria de sistemas de votação electrónica: uma proposta de arquitectura e protótipo de simulação.* In *2º Workshop sobre Voto pela Internet.* Aveiro, Portugal, Outubro, 2006.

[12] Ricardo Lebre, Rui Joaquim, André Zúquete, Paulo Ferreira. *Internet Voting: Improving Resistance to malicious Servers in REVS*. In *International Conference on Applied Computing* (IADIS'2004), 2004 .

[13] D. Chaum. *Blind Signature for Untraceable Payments*. In *Advances in Cryptology - CRYPTO '82, pp 199-203*, 1983.

[14] André Zúquete, Filipe Almeida. *Verifiable Anonymous Vote Submission.* In Proceedings Symposium on Applied Computing, pp 2159-2166, 2008.

[15] Edmund M. Clarke, Jeannette M. Wing. *Formal Methods: State of the Art and future directions.* In *ACM Computing Surveys (CSUR),* Volume 28, Issue 4, 1996.

[16] Christel Baier, Joost-Pieter Katoen, *Principles of Model Checking*, MIT Press, 2008.

[17] Gerald J.Holzman. *Design and validation of computer protocols*. Prentice-Hall, 1990.

[18] Stefano Campanelli, Alessandro Falleni, Fabio Martinelli, Marinella Petrocchi, Anna Vaccarelli. *Mobile implementation and formal verification of an e-voting system.* In *ICIW archive Proceedings of the 2008 Third International Conference on Internet and Web Applications and Services*, 2008.

[19] Dominique Cansell, J. Paul Gibson, Dominique Méry. *Formal verification of tamper-evident storage for e-voting*. In *Fifth IEEE International Conference on Software Engineering and Formal Methods*, 2007.

[20] Stéphanie Delaune, Steve Kremer, Mark Ryan. *Verifying privacy type of electronic voting protocol.* In *Journal of Computer Security,* IOS Press, Volume 17, Number 4, pp 435-487, 2009.

[21] Steve Kremer1, Mark Ryan. *Analysis of an Electronic Voting Protocol in the Applied Pi Calculus.* In *Electronic Notes in Theoretical Computer Science,* Proceedings *of the Second Workshop on Automated Reasoning for Security Protocol Analysis,* Volume 135, Issue 1, pp 115-134, July, 2005.

[22] Stéphanie Delaune, Steve Kremer, Mark Ryan. *Coercion-Resistance and Receipt-Freeness in Electronic Voting*. In *Proceedings of the 19th IEEE workshop on Computer Security Foundations,* pp 28 – 42, 2006.

[23] Henrik Ejersbo Jensen, Kim G. Larsen, Arne Skou. *Modelling and Analysis of a Collision Avoidance Protocol using SPIN and UPPAAL*. In *Proc. 2nd SPIN Verification Workshop on Algorithms, Applications, Tool Use, Theory. American Mathematical Society*, Providence, 1996.

[24] Jose Garcia-Fanjul, Javier Tuya, Jose Antonio Corrales. *Formal Verification and Simulation of the NetBill Protocol Using SPIN.* In *4th International Workshop on Automata Theoretic Verification with the SPIN Model Checker*, París, 1998.

[25] Biniam Gebremichael, Frits Vaandrager, Miaomiao Zhang. *Analysis of the Zeroconf Protocol Using UPPAAL.* In *Proceedings of EMSOFT*, 2006.

[26] Ferdy Hanssen, Angelika Mader, Pierre G. Jansen. *Verifying the distributed real-time network protocol RTnet using Uppaal.* In *Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation,* pp 239 – 246, 2006.

[27] Jun Pang, Bart Karstens, Wan Fokkink. *Analyzing the Redesign of a Distributed Lift System in UPPAAL.* In *Fifth International Conference on Formal Engineering Methods, ICFEM'2003, Lecture Notes in Computer Science Volume 2885*, pp 504-522, 2003.

[28] Gerd Behrmann, Alexandre David, Kim G. Larsen, *A Tutorial on Uppaal*. In *Proc. of 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, number 3185 in Lecture Notes in Computer Science*, 2004.

[29] Kim G. Larsen, Paul Pettersson, Wang Yi. *UPPAAL in a Nutshell*. IN *Int.Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, October 1997.

[30] Benoît Chevallier-Mames, Pierre-Alain Fouque, David Pointcheval, Julien Stern, Jacques Traoré. *On Some Incompatible Properties of Voting Schemes*. In *Proceedings of the IAVoSS Workshop on Trustworthy Elections*, 2006.

[31] Stylianos Basagiannis, Panagiootis Katsaros, Andrew Pombortsis. *Intrusion Attack Tatics for the Model Checking of e-commerce Security Guarantees.* In *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, Volume 4680 Computer Safety, Reliability, and Security, pp238-251, 2007.

[32] Paolo Maggi, Riccardo Sisto. *Using SPIN to Verify Security Properties of Cryptographic Protocols*, In *Lecture Notes In Computer Science*; Volume 2318 Proceedings of the 9th International SPIN Workshop on Model Checking of Software, pp 187 – 204, 2002.

A. Modelo SPIN

```
#define PedidoBoletimEleicao 1;                     /*Mensagens trocadas entre o Modulo
                                                    de Voto e Distribuidor*/
#define EnvioBoletimEleicao 2;
#define PedidoBoletimVoto 3;
#define EnvioBoletimVoto 4;
#define PedidoAssinaturaVoto 5;
#define EnvioAssinaturaVoto 6;
#define EnvioSubmissaoVoto 7;

#define INICIO 0;                                           /*Estados do Modulo de Voto*/
#define PedidoBoletimEleicaoEnviado 1;
#define PedidoBoletimVotoEnviado 2;
#define PedidoAssinaturaVotoEnviado 3;
#define TotalAssinaturasObtido 4;
#define FIM 5;

#define NumDistribuidores 2;                        /* Numero de processos do modelo */
#define NumAdministradores 3;
#define NumEleitores 3;
#define NumAnonimizadores 2;
#define NumTotalizadores 2;
#define CapacidadeEscrutinio 5;
#define NumTotalAssinaturas 2;
#define DELTA 10;

int selVoto=0, selAnonimizador=0, selDistribuidor=0;

#define SelDistribuidor() selDistribuidor = (selDistribuidor+1) % NumDistribuidores
#define SelAnonimizador() selAnonimizador = (selAnonimizador+1) % NumAnonimizadores
#define SelAdministrador() admin = (admin+1) % NumAdministradores

#define randomNumber()       if  \
                                          :: count = 5 \
                                          :: count = 15 \
                                          :: count = 30 \
                                          :: count = 90 \
                                          fi


int iniciarEscrutinio=0, fimEscrutinio = 0;

int indexA=0, indexR=0;

int  lostMsg = 0;

int gPrimeiroEleitor =0;

/* Comunicacao entre Eleitor e Distribuidor */
chan gEleitorDistribuidor = [10] of {byte, byte, byte};

/* Comunicacao entre Eleitor e Administrador */
chan gEleitorAdministrador = [10] of {byte, byte,byte};

/* Comunicacao entre Eleitor e Anonimizadorr */
chan gEleitorAnonimizador = [10] of {byte, byte, byte};

/* Comunicacao entre Eleitor e Totalizador */
chan gAnonimizadorTotalizador = [10] of {byte, byte, byte};
```

```
/* Delay entre o Anonimizador e Totalizador */
chan gDelayAnonimizadorTotalizador = [10] of {byte, byte, byte};

/* Informa • ‹o sobre o Eleitor: estado, assinaturas, numero assinaturas recolhidas */
typedef  recEleitor {
        int Estado;
        bool AssinaturaAdministrador[3];
        int NumAssinaturas;
};

/* Informacao sobre o Voto: identificacaodo voto, numero de assinaturas recolhidas */
typedef  recVoto {
        int IdVoto;
        int NumAssinaturas
};

recEleitor gEleitor[3];
recVoto gVotoRecebido[5];
recVoto gVotoApurado[5];


proctype ModuloVoto(int pEleitor)
{
        int lDistribuidor;
        int admin = 0;

        do
        :: gEleitor[pEleitor].Estado == INICIO ->
                atomic {
                        SelDistribuidor();
                        gEleitorDistribuidor!pEleitor, selDistribuidor,
                                                PedidoBoletimEleicao;
                        gEleitor[pEleitor].Estado = PedidoBoletimEleicaoEnviado;
                }
        :: gEleitor[pEleitor].Estado == PedidoBoletimEleicaoEnviado ->
                gEleitorDistribuidor?eval(pEleitor), lDistribuidor,
                                                EnvioBoletimEleicao ->
                atomic {
                        gEleitorDistribuidor!pEleitor, lDistribuidor, PedidoBoletimVoto;
                        gEleitor[pEleitor].Estado = PedidoBoletimVotoEnviado
                }
        :: gEleitor[pEleitor].Estado == PedidoBoletimVotoEnviado ->
                gEleitorDistribuidor?eval(pEleitor), lDistribuidor, EnvioBoletimVoto ->
                atomic {
                        SelAdministrador();
                        gEleitorAdministrador!pEleitor, admin, PedidoAssinaturaVoto;
                        gEleitor[pEleitor].Estado = PedidoAssinaturaVotoEnviado;
                }
        :: gEleitor[pEleitor].Estado == PedidoAssinaturaVotoEnviado  ->
                do
                :: gEleitor[pEleitor].NumAssinaturas < NumTotalAssinaturas ->
                        gEleitorAdministrador?eval(pEleitor), eval(admin),
                                                EnvioAssinaturaVoto;
                        if
                        :: gEleitor[pEleitor].AssinaturaAdministrador[admin] == 0 ->
                                atomic {
                                        gEleitor[pEleitor].AssinaturaAdministrador[admin] =
1;
                                        gEleitor[pEleitor].NumAssinaturas++;
                                }
                        fi;
                        if
                        :: gEleitor[pEleitor].NumAssinaturas < NumTotalAssinaturas ->
                                atomic {
                                        SelAdministrador();
                                        gEleitorAdministrador!pEleitor,admin,
                                                        PedidoAssinaturaVoto;
                                }
                        :: gEleitor[pEleitor].NumAssinaturas == NumTotalAssinaturas ->
                                gEleitor[pEleitor].Estado = TotalAssinaturasObtido;
                                break;
                        fi;
                od;
        :: gEleitor[pEleitor].Estado == TotalAssinaturasObtido ->
                atomic {
                        SelAnonimizador();
```

```
                        gEleitorAnonimizador!pEleitor,gEleitor[pEleitor].NumAssinaturas,
                                                        selAnonimizador;
                        gEleitor[pEleitor].Estado = FIM;
                }
        :: gEleitor[pEleitor].Estado == FIM -> break;
        od;
}

proctype Distribuidor(byte pDistribuidor)
{
        int Eleitor=0;

        end:
        do
        :: gEleitorDistribuidor?Eleitor, eval(pDistribuidor), PedidoBoletimEleicao ->
                gEleitorDistribuidor!Eleitor, pDistribuidor, EnvioBoletimEleicao
        :: gEleitorDistribuidor?Eleitor, eval(pDistribuidor), PedidoBoletimVoto ->
                gEleitorDistribuidor!Eleitor, pDistribuidor, EnvioBoletimVoto
        od
}

proctype Administrador(int pAdmin)
{
        int Eleitor = 0;

        end:
        do
        :: gEleitorAdministrador?Eleitor,eval(pAdmin),PedidoAssinaturaVoto ->
                gEleitorAdministrador!Eleitor,pAdmin,EnvioAssinaturaVoto
        od
}

proctype Anonimizador(int pAnonimizador)
{
        int Eleitor, lidEleitor, lTotalAssinaturas = 0;

        endAnonimizador: do
        :: gEleitorAnonimizador??lidEleitor, lTotalAssinaturas, eval(pAnonimizador) ->
                gPrimeiroEleitor == 0 -> gPrimeiroEleitor = lidEleitor+DELTA;
                gDelayAnonimizadorTotalizador!lidEleitor, lTotalAssinaturas,
pAnonimizador
        :: gDelayAnonimizadorTotalizador??lidEleitor, lTotalAssinaturas,
eval(pAnonimizador) ->
                atomic { Eleitor = lidEleitor+DELTA; gAnonimizadorTotalizador!Eleitor,
                                        lTotalAssinaturas, pAnonimizador; Eleitor=0}
        od;
}

proctype Totalizador(int pTotalizador)
{
        int x,y,z;
        do
        :: iniciarEscrutinio == 0 && selVoto < CapacidadeEscrutinio ->
                if
                :: atomic{
                        gAnonimizadorTotalizador??gVotoRecebido[selVoto].IdVoto,
                                gVotoRecebido[selVoto].NumAssinaturas, eval(pTotalizador);
                        selVoto++;
                        };
                :: empty(gAnonimizadorTotalizador) -> skip;
                fi;
        :: iniciarEscrutinio  == 1 -> break;
        :: else -> skip;
        od;
        if
        :: pTotalizador == 0 ->
                do
                :: indexR < selVoto ->
                        if
                        :: gVotoApurado[indexA].IdVoto == 0 &&
                        gVotoRecebido[indexR].NumAssinaturas == NumTotalAssinaturas ->
                                atomic {
                                        gVotoApurado[indexA].IdVoto =
                                                gVotoRecebido[indexR].IdVoto;
                                        gVotoApurado[indexA].NumAssinaturas =
                                                gVotoRecebido[indexR].NumAssinaturas;
                                        indexR++;
```

```
                                    indexA++;
                            }
                    :: gVotoApurado[indexA].IdVoto != gVotoRecebido[indexR].IdVoto
                                    && gVotoApurado[indexA].IdVoto != 0 -> indexA++;
                    :: else-> indexR++;
                    fi;
            :: else -> fimEscrutinio = 1; break;
            od;
        :: else  -> skip;
        fi;

        end:
        do
        :: gAnonimizadorTotalizador??x,y,z -> lostMsg++;
        od;
}


proctype Temporizador()
{
        int count;
        int i = 0;

        randomNumber();
        do
        :: i < count -> i++;
        :: i == count -> break;
        od;

        iniciarEscrutinio = 1;
}

init
{
        int id = 0;

        /* Lancar um numero variavel de Eleitores */
        do
        :: id < NumEleitores -> atomic { run ModuloVoto(id); id++ }
        :: id == NumEleitores -> id=0; break
        od;

        /* Lancar um numero variavel de Distribuidores */

        do
        :: id < NumDistribuidores -> atomic { run Distribuidor(id); id++ }
        :: id == NumDistribuidores -> id=0; break
        od;

        /* Lancar um numero variavel de Administradores */
        do
        :: id < NumAdministradores -> atomic { run Administrador(id); id++ }
        :: id == NumAdministradores -> id=0; break
        od ;

        /* Lancar um numero variavel de Anonimizadores */
        do
        :: id < NumAnonimizadores -> atomic { run Anonimizador(id); id++ }
        :: id == NumAnonimizadores -> id=0; break
        od;

        /* Lancar um numero variavel de Totalizadores */
        do
        :: id < NumTotalizadores -> atomic { run Totalizador(id); id++ }
        :: id == NumTotalizadores -> id=0; break
        od;

        run Temporizador();
}
```

A.1. Propriedades SPIN

4.4.2 Democracia

```
#define p ((gVotoApurado[0].IdVoto == 10) | (gVotoApurado[1].IdVoto == 10) |
        (gVotoApurado[2].IdVoto == 10)) && ((gVotoApurado[0].IdVoto == 11)
|(gVotoApurado[1].IdVoto == 11) |(gVotoApurado[2].IdVoto == 11)) &&
((gVotoApurado[0].IdVoto == 12) | (gVotoApurado[1].IdVoto == 12) |
        (gVotoApurado[2].IdVoto == 12))

#define q (indexR == selVoto && indexR == 3)

        /*
         * Formula As Typed: []   (q -> p)
         * The Never Claim Below Corresponds
         * To The Negated Formula !([]   (q -> p))
         * (formalizing violations of the original)
         */

never {    /* !([]   (q -> p)) */
T0_init:
        if
        :: (! ((p)) && (q)) -> goto accept_all
        :: (1) -> goto T0_init
        fi;
accept_all:
        skip
}

#ifdef NOTES
"Só os eleitores elegíveis votam e uma única vez."

#endif
#ifdef RESULT
warning: for p.o. reduction to be valid the never claim must be stutter-invariant
(never claims generated from LTL formulae are stutter-invariant)
depth 0: Claim reached state 5 (line 283)
Depth=     333 States=   1e+006 Transitions= 3.86e+006 Memory=     8.598  t=   20.4 R=
5e+004
Depth=     333 States=   2e+006 Transitions= 7.74e+006 Memory=     8.598  t=   39.5 R=
5e+004
Depth=     333 States=   3e+006 Transitions= 1.16e+007 Memory=     8.598  t=   58.4 R=
5e+004
Depth=     333 States=   4e+006 Transitions= 1.55e+007 Memory=     8.598  t=   77.7 R=
5e+004
Depth=     333 States=   5e+006 Transitions= 1.93e+007 Memory=     8.598  t=   96.5 R=
5e+004
Depth=     333 States=   6e+006 Transitions= 2.34e+007 Memory=     8.598  t=    116 R=
5e+004
Depth=     333 States=   7e+006 Transitions= 2.8e+007 Memory=     8.598   t=    138 R=
5e+004
Depth=     333 States=   8e+006 Transitions= 3.27e+007 Memory=     8.598  t=    160 R=
5e+004
Depth=     333 States=   9e+006 Transitions= 3.73e+007 Memory=     8.598  t=    181 R=
5e+004
Depth=     333 States=   1e+007 Transitions= 4.2e+007 Memory=     8.598   t=    203 R=
5e+004
Depth=     333 States= 1.1e+007 Transitions= 4.66e+007 Memory=     8.598  t=    225 R=
5e+004
Depth=     333 States= 1.2e+007 Transitions= 5.13e+007 Memory=     8.598  t=    247 R=
5e+004
Depth=     333 States= 1.3e+007 Transitions= 5.59e+007 Memory=     8.598  t=    269 R=
5e+004
Depth=     333 States= 1.4e+007 Transitions= 6.06e+007 Memory=     8.598  t=    291 R=
5e+004
Depth=     333 States= 1.5e+007 Transitions= 6.5e+007 Memory=     8.598   t=    312 R=
5e+004
Depth=     333 States= 1.6e+007 Transitions= 6.8e+007 Memory=     8.598   t=    328 R=
5e+004
Depth=     333 States= 1.7e+007 Transitions= 7.09e+007 Memory=     8.598  t=    343 R=
5e+004
Depth=     333 States= 1.8e+007 Transitions= 7.39e+007 Memory=     8.598  t=    359 R=
5e+004
Depth=     333 States= 1.9e+007 Transitions= 7.69e+007 Memory=     8.598  t=    375 R=
5e+004
Depth=     333 States=   2e+007 Transitions=   8e+007 Memory=     8.598   t=    391 R=
5e+004
Depth=     333 States= 2.1e+007 Transitions= 8.31e+007 Memory=     8.598  t=    407 R=
5e+004
Depth=     333 States= 2.2e+007 Transitions= 8.59e+007 Memory=     8.598  t=    421 R=
5e+004
```

```
Depth=     333 States= 2.3e+007 Transitions= 8.85e+007 Memory=     8.598  t=    436 R=
5e+004
Depth=     333 States= 2.4e+007 Transitions= 9.12e+007 Memory=     8.598  t=    450 R=
5e+004
Depth=     333 States= 2.5e+007 Transitions= 9.4e+007 Memory=     8.598   t=    465 R=
5e+004
Depth=     333 States= 2.6e+007 Transitions= 9.69e+007 Memory=     8.598  t=    480 R=
5e+004
Depth=     333 States= 2.7e+007 Transitions=   1e+008 Memory=     8.598   t=    497 R=
5e+004
Depth=     333 States= 2.8e+007 Transitions= 1.04e+008 Memory=     8.598  t=    514 R=
5e+004
Depth=     339 States= 2.9e+007 Transitions= 1.07e+008 Memory=     8.598  t=    532 R=
5e+004
Depth=     345 States=   3e+007 Transitions= 1.1e+008 Memory=     8.598   t=    549 R=
5e+004
Depth=     551 States= 3.1e+007 Transitions= 1.14e+008 Memory=     8.696  t=    567 R=
5e+004
Depth=     575 States= 3.2e+007 Transitions= 1.18e+008 Memory=     8.696  t=    586 R=
5e+004
Depth=     575 States= 3.3e+007 Transitions= 1.21e+008 Memory=     8.696  t=    606 R=
5e+004

(Spin Version 5.1.6 -- 9 May 2008)
       + Partial Order Reduction

Bit statespace search for:
       never claim           +
       assertion violations  + (if within scope of claim)
       acceptance   cycles   + (fairness disabled)
       invalid end states    - (disabled by never claim)

State-vector 520 byte, depth reached 575, errors: 0
 33230335 states, stored
 89143012 states, matched
1.2237335e+008 transitions (= stored+matched)
  8611197 atomic steps

hash factor: 2.01951 (best if > 100.)

bits set per state: 3 (-k3)

Stats on memory usage (in Megabytes):
16986.332      equivalent memory usage for states (stored*(State-vector + overhead))
    8.000      memory used for hash array (-w26)
    0.038      memory used for bit stack
    0.305      memory used for DFS stack (-m10000)
    8.696      total actual memory usage


pan: elapsed time 611 seconds
pan: rate 54399.621 states/second
pan: avg transition delay 4.9917e-006 usec

#endif
```

4.4.3 Precisão

```
#define p (gVotoApurado[0].IdVoto == 10) | (gVotoApurado[1].IdVoto == 10) |
(gVotoApurado[2].IdVoto == 10)
#define q (gVotoRecebido[0].IdVoto == 10) | (gVotoRecebido[1].IdVoto == 10) |
(gVotoRecebido[2].IdVoto  == 10)

       /*
        * Formula As Typed: []  p-> q
        * The Never Claim Below Corresponds
        * To The Negated Formula !([]  p-> q)
        * (formalizing violations of the original)
        */

never {    /* !([]  p-> q) */
accept_init:
T0_init:
       if
       :: (! ((q)) && (p)) -> goto accept_S3
```

```
        fi;
accept_S3:
T0_S3:
        if
        :: ((p)) -> goto accept_S3
        fi;
}

#ifdef NOTES
"Se um voto for efectuado pelo eleitor, ele deverá ser contado no escrutínio final"

#endif
#ifdef RESULT
warning: for p.o. reduction to be valid the never claim must be stutter-invariant
(never claims generated from LTL formulae are stutter-invariant)
depth 0: Claim reached state 3 (line 283)

(Spin Version 5.1.6 -- 9 May 2008)
        + Partial Order Reduction

Bit statespace search for:
        never claim             +
        assertion violations    + (if within scope of claim)
        acceptance   cycles     + (fairness disabled)
        invalid end states      - (disabled by never claim)

State-vector 328 byte, depth reached 0, errors: 0
        1 states, stored
        0 states, matched
        1 transitions (= stored+matched)
        0 atomic steps

hash factor: 6.71089e+007 (best if > 100.)

bits set per state: 3 (-k3)

Stats on memory usage (in Megabytes):
    0.000       equivalent memory usage for states (stored*(State-vector + overhead))
    8.000       memory used for hash array (-w26)
    0.038       memory used for bit stack
    0.305       memory used for DFS stack (-m10000)
    8.501       total actual memory usage


pan: elapsed time 0 seconds

#endif
```

4.4.5 REVS – Assinaturas para o voto ser válido

```
#define p (gEleitor[0].Estado == 5)
#define q (gEleitor[0].NumAssinaturas == 2)

        /*
         * Formula As Typed: [](p -> q)
         * The Never Claim Below Corresponds
         * To The Negated Formula !([](p -> q))
         * (formalizing violations of the original)
         */

never {    /* !([](p -> q)) */
T0_init:
        if
        :: (! ((q)) && (p)) -> goto accept_all
        :: (1) -> goto T0_init
        fi;
accept_all:
        skip
}

#ifdef NOTES
"Um eleitor que termine o seu processo de voto garantidamente obteve um número de
assinaturas suficiente para o voto ser válido."
```

```
#endif
#ifdef RESULT
warning: for p.o. reduction to be valid the never claim must be stutter-invariant
(never claims generated from LTL formulae are stutter-invariant)
depth 0: Claim reached state 5 (line 282)
Depth=     333 States=   1e+006 Transitions= 3.86e+006 Memory=     8.903  t=   20.9 R=
5e+004
Depth=     333 States=   2e+006 Transitions= 7.74e+006 Memory=     8.903  t=   40.1 R=
5e+004
Depth=     333 States=   3e+006 Transitions= 1.16e+007 Memory=     8.903  t=   59.9 R=
5e+004
Depth=     333 States=   4e+006 Transitions= 1.55e+007 Memory=     8.903  t=   79.5 R=
5e+004
Depth=     333 States=   5e+006 Transitions= 1.93e+007 Memory=     8.903  t=   98.5 R=
5e+004
Depth=     333 States=   6e+006 Transitions= 2.34e+007 Memory=     8.903  t=    118 R=
5e+004
Depth=     333 States=   7e+006 Transitions= 2.8e+007 Memory=     8.903   t=    140 R=
5e+004
Depth=     333 States=   8e+006 Transitions= 3.27e+007 Memory=     8.903  t=    162 R=
5e+004
Depth=     333 States=   9e+006 Transitions= 3.73e+007 Memory=     8.903  t=    184 R=
5e+004
Depth=     333 States=   1e+007 Transitions= 4.2e+007 Memory=     8.903   t=    205 R=
5e+004
Depth=     333 States= 1.1e+007 Transitions= 4.66e+007 Memory=     8.903  t=    227 R=
5e+004
Depth=     333 States= 1.2e+007 Transitions= 5.13e+007 Memory=     8.903  t=    249 R=
5e+004
Depth=     333 States= 1.3e+007 Transitions= 5.59e+007 Memory=     8.903  t=    271 R=
5e+004
Depth=     333 States= 1.4e+007 Transitions= 6.06e+007 Memory=     8.903  t=    293 R=
5e+004
Depth=     333 States= 1.5e+007 Transitions= 6.5e+007 Memory=     8.903   t=    314 R=
5e+004
Depth=     333 States= 1.6e+007 Transitions= 6.8e+007 Memory=     8.903   t=    329 R=
5e+004
Depth=     333 States= 1.7e+007 Transitions= 7.09e+007 Memory=     8.903  t=    347 R=
5e+004
Depth=     333 States= 1.8e+007 Transitions= 7.39e+007 Memory=     8.903  t=    365 R=
5e+004
Depth=     333 States= 1.9e+007 Transitions= 7.69e+007 Memory=     8.903  t=    383 R=
5e+004
Depth=     333 States=   2e+007 Transitions=   8e+007 Memory=     8.903   t=    400 R=
5e+004
Depth=     333 States= 2.1e+007 Transitions= 8.31e+007 Memory=     8.903  t=    416 R=
5e+004
Depth=     333 States= 2.2e+007 Transitions= 8.59e+007 Memory=     8.903  t=    431 R=
5e+004
Depth=     333 States= 2.3e+007 Transitions= 8.85e+007 Memory=     8.903  t=    445 R=
5e+004
Depth=     333 States= 2.4e+007 Transitions= 9.12e+007 Memory=     8.903  t=    459 R=
5e+004
Depth=     333 States= 2.5e+007 Transitions= 9.4e+007 Memory=     8.903   t=    473 R=
5e+004
Depth=     333 States= 2.6e+007 Transitions= 9.69e+007 Memory=     8.903  t=    489 R=
5e+004
Depth=     333 States= 2.7e+007 Transitions=   1e+008 Memory=     8.903   t=    505 R=
5e+004
Depth=     333 States= 2.8e+007 Transitions= 1.04e+008 Memory=     8.903  t=    522 R=
5e+004
Depth=     339 States= 2.9e+007 Transitions= 1.07e+008 Memory=     8.903  t=    539 R=
5e+004
Depth=     345 States=   3e+007 Transitions= 1.1e+008 Memory=     8.903   t=    556 R=
5e+004
Depth=     551 States= 3.1e+007 Transitions= 1.14e+008 Memory=     9.001  t=    574 R=
5e+004
Depth=     575 States= 3.2e+007 Transitions= 1.18e+008 Memory=     9.001  t=    593 R=
5e+004
Depth=     575 States= 3.3e+007 Transitions= 1.21e+008 Memory=     9.001  t=    613 R=
5e+004

(Spin Version 5.1.6 -- 9 May 2008)
       + Partial Order Reduction

Bit statespace search for:
```

```
        never claim           +
        assertion violations  + (if within scope of claim)
        acceptance   cycles   + (fairness disabled)
        invalid end states    - (disabled by never claim)

State-vector 520 byte, depth reached 575, errors: 0
 33230335 states, stored
 89143012 states, matched
1.2237335e+008 transitions (= stored+matched)
  8611197 atomic steps

hash factor: 2.01951 (best if > 100.)

bits set per state: 3 (-k3)

Stats on memory usage (in Megabytes):
16986.332      equivalent memory usage for states (stored*(State-vector + overhead))
    8.000      memory used for hash array (-w26)
    0.076      memory used for bit stack
    0.610      memory used for DFS stack (-m20000)
    9.001      total actual memory usage


pan: elapsed time 617 seconds
pan: rate 53844.827 states/second
pan: avg transition delay 5.0432e-006 usec

#endif
```

4.4.6 REVS – Eleitor termina processo de votação

```
#define p (gEleitor[0].Estado == 5) && (gEleitor[1].Estado == 5) && (gEleitor[2].Estado
== 5)
#define q  iniciarEscrutinio==1

        /*
         * Formula As Typed: <>    (q -> p)
         * The Never Claim Below Corresponds
         * To The Negated Formula !(<>    (q -> p))
         * (formalizing violations of the original)
         */

never {    /* !(<>    (q -> p)) */
accept_init:
T0_init:
        if
        :: (! ((p)) && (q)) -> goto T0_init
        fi;
}

#ifdef NOTES
"O Eleitor eventualmente termina o seu processo de votação."


#endif
#ifdef RESULT
warning: for p.o. reduction to be valid the never claim must be stutter-invariant
(never claims generated from LTL formulae are stutter-invariant)
depth 0: Claim reached state 3 (line 283)

(Spin Version 5.1.6 -- 9 May 2008)
        + Partial Order Reduction

Bit statespace search for:
        never claim           +
        assertion violations  + (if within scope of claim)
        acceptance   cycles   + (fairness disabled)
        invalid end states    - (disabled by never claim)

State-vector 328 byte, depth reached 0, errors: 0
        1 states, stored
        0 states, matched
        1 transitions (= stored+matched)
        0 atomic steps
```

```
hash factor: 6.71089e+007 (best if > 100.)

bits set per state: 3 (-k3)

Stats on memory usage (in Megabytes):
    0.000      equivalent memory usage for states (stored*(State-vector + overhead))
    8.000      memory used for hash array (-w26)
    0.076      memory used for bit stack
    0.610      memory used for DFS stack (-m20000)
    8.806      total actual memory usage


pan: elapsed time 0 seconds

#endif
```

4.4.7 REVS – Eleitor pode não terminar processo de votação

```
#define p (gEleitor[0].Estado == 5) && (gEleitor[1].Estado == 5) && (gEleitor[2].Estado
== 5)
#define q  iniciarEscrutinio==1

       /*
        * Formula As Typed: []    (q -> p)
        * The Never Claim Below Corresponds
        * To The Negated Formula !([]    (q -> p))
        * (formalizing violations of the original)
        */

never {    /* !([]    (q -> p)) */
T0_init:
       if
       :: (! ((p)) && (q)) -> goto accept_all
       :: (1) -> goto T0_init
       fi;
accept_all:
       skip
}

#ifdef NOTES
"O Eleitor pode não terminar o seu processo de votação."

#endif
#ifdef RESULT
warning: for p.o. reduction to be valid the never claim must be stutter-invariant
(never claims generated from LTL formulae are stutter-invariant)
depth 0: Claim reached state 5 (line 282)
pan: claim violated! (at depth 79)
pan: wrote pan_in.trail

(Spin Version 5.1.6 -- 9 May 2008)
Warning: Search not completed
       + Partial Order Reduction

Bit statespace search for:
       never claim           +
       assertion violations  + (if within scope of claim)
       acceptance   cycles   + (fairness disabled)
       invalid end states    - (disabled by never claim)

State-vector 520 byte, depth reached 79, errors: 1
       40 states, stored
        0 states, matched
       40 transitions (= stored+matched)
        0 atomic steps

hash factor: 1.67772e+006 (best if > 100.)

bits set per state: 3 (-k3)

Stats on memory usage (in Megabytes):
    0.020      equivalent memory usage for states (stored*(State-vector + overhead))
    8.000      memory used for hash array (-w26)
    0.076      memory used for bit stack
```

```
    0.610      memory used for DFS stack (-m20000)
    8.806      total actual memory usage


pan: elapsed time 0.004 seconds

#endif
```

4.4.8 REVS – Ordem dos votos

```
#define p (iniciarEscrutinio == 1 && selVoto >=2 && gPrimeiroEleitor == (0 + 10))
#define q (gVotoRecebido[0].IdVoto == (0 + 10))

        /*
         * Formula As Typed: []  (p -> q)
         * The Never Claim Below Corresponds
         * To The Negated Formula !([]  (p -> q))
         * (formalizing violations of the original)
         */

never {    /* !([]  (p -> q)) */
T0_init:
        if
        :: (! ((q)) && (p)) -> goto accept_all
        :: (1) -> goto T0_init
        fi;
accept_all:
        skip
}

#ifdef NOTES
"A ordem de recepção do voto no Anonimizador não é sempre igual à ordem de votos
recebidos."

#endif
#ifdef RESULT
warning: for p.o. reduction to be valid the never claim must be stutter-invariant
(never claims generated from LTL formulae are stutter-invariant)
depth 0: Claim reached state 5 (line 282)
Depth=     333 States=   1e+006 Transitions= 3.86e+006 Memory=     8.903  t=   19.9 R=
5e+004
Depth=     333 States=   2e+006 Transitions= 7.74e+006 Memory=     8.903  t=   38.8 R=
5e+004
Depth=     333 States=   3e+006 Transitions= 1.16e+007 Memory=     8.903  t=   58.2 R=
5e+004
Depth=     333 States=   4e+006 Transitions= 1.55e+007 Memory=     8.903  t=   77.6 R=
5e+004
Depth=     333 States=   5e+006 Transitions= 1.93e+007 Memory=     8.903  t=   96.6 R=
5e+004
Depth=     333 States=   6e+006 Transitions= 2.34e+007 Memory=     8.903  t=    117 R=
5e+004
Depth=     333 States=   7e+006 Transitions= 2.8e+007 Memory=     8.903   t=    139 R=
5e+004
Depth=     333 States=   8e+006 Transitions= 3.27e+007 Memory=     8.903  t=    161 R=
5e+004
Depth=     333 States=   9e+006 Transitions= 3.73e+007 Memory=     8.903  t=    183 R=
5e+004
Depth=     333 States=   1e+007 Transitions= 4.2e+007 Memory=     8.903   t=    205 R=
5e+004
Depth=     333 States= 1.1e+007 Transitions= 4.66e+007 Memory=     8.903  t=    227 R=
5e+004
Depth=     333 States= 1.2e+007 Transitions= 5.13e+007 Memory=     8.903  t=    249 R=
5e+004
Depth=     333 States= 1.3e+007 Transitions= 5.59e+007 Memory=     8.903  t=    271 R=
5e+004
Depth=     333 States= 1.4e+007 Transitions= 6.06e+007 Memory=     8.903  t=    293 R=
5e+004
Depth=     333 States= 1.5e+007 Transitions= 6.5e+007 Memory=     8.903   t=    314 R=
5e+004
Depth=     333 States= 1.6e+007 Transitions= 6.8e+007 Memory=     8.903   t=    329 R=
5e+004
Depth=     333 States= 1.7e+007 Transitions= 7.09e+007 Memory=     8.903  t=    344 R=
5e+004
```

```
Depth=     333 States= 1.8e+007 Transitions= 7.39e+007 Memory=     8.903  t=    359 R=
5e+004
Depth=     333 States= 1.9e+007 Transitions= 7.69e+007 Memory=     8.903  t=    375 R=
5e+004
Depth=     333 States=   2e+007 Transitions=   8e+007 Memory=     8.903  t=    391 R=
5e+004
Depth=     333 States= 2.1e+007 Transitions= 8.31e+007 Memory=     8.903  t=    406 R=
5e+004
Depth=     333 States= 2.2e+007 Transitions= 8.59e+007 Memory=     8.903  t=    421 R=
5e+004
Depth=     333 States= 2.3e+007 Transitions= 8.85e+007 Memory=     8.903  t=    435 R=
5e+004
Depth=     333 States= 2.4e+007 Transitions= 9.12e+007 Memory=     8.903  t=    449 R=
5e+004
Depth=     333 States= 2.5e+007 Transitions= 9.4e+007 Memory=     8.903  t=    464 R=
5e+004
Depth=     333 States= 2.6e+007 Transitions= 9.69e+007 Memory=     8.903  t=    479 R=
5e+004
Depth=     333 States= 2.7e+007 Transitions=   1e+008 Memory=     8.903  t=    495 R=
5e+004
Depth=     333 States= 2.8e+007 Transitions= 1.04e+008 Memory=     8.903  t=    512 R=
5e+004
Depth=     339 States= 2.9e+007 Transitions= 1.07e+008 Memory=     8.903  t=    529 R=
5e+004
Depth=     345 States=   3e+007 Transitions= 1.1e+008 Memory=     8.903  t=    546 R=
5e+004
pan: claim violated! (at depth 477)
pan: wrote pan_in.trail

(Spin Version 5.1.6 -- 9 May 2008)
Warning: Search not completed
       + Partial Order Reduction

Bit statespace search for:
       never claim          +
       assertion violations  + (if within scope of claim)
       acceptance   cycles   + (fairness disabled)
       invalid end states    - (disabled by never claim)

State-vector 520 byte, depth reached 499, errors: 1
 30513909 states, stored
 81702924 states, matched
1.1221683e+008 transitions (= stored+matched)
  7970299 atomic steps

hash factor: 2.19929 (best if > 100.)

bits set per state: 3 (-k3)

Stats on memory usage (in Megabytes):
15597.778      equivalent memory usage for states (stored*(State-vector + overhead))
    8.000      memory used for hash array (-w26)
    0.076      memory used for bit stack
    0.610      memory used for DFS stack (-m20000)
    9.001      total actual memory usage


pan: elapsed time 555 seconds
pan: rate  54995.97 states/second
pan: avg transition delay 4.9443e-006 usec

#endif
```

4.4.9 Apenas verificável em SPIN – Linear Temporal Logoc

```
#define p (gEleitor[0].Estado == 5)
#define q (gEleitor[0].NumAssinaturas == 2)

       /*
        * Formula As Typed: []  (<>  p )-> []  (<>  q)
        * The Never Claim Below Corresponds
        * To The Negated Formula !([]  (<>  p )-> []  (<>  q))
        * (formalizing violations of the original)
        */

never {    /* !([]  (<>  p )-> []  (<>  q)) */
```

```
T0_init:
	if
	:: (! ((q)) && (p)) -> goto accept_S69
	:: (! ((q))) -> goto T0_S69
	:: (1) -> goto T0_init
	fi;
accept_S69:
	if
	:: (! ((q))) -> goto T0_S69
	fi;
T0_S69:
	if
	:: (! ((q)) && (p)) -> goto accept_S69
	:: (! ((q))) -> goto T0_S69
	fi;
}

#ifdef NOTES
"Sempre que o eleitor eventualmente termina o seu voto, então sempre eventualmente terá
obtido todas as assinaturas necessárias para o mesmo ser válido "




#endif
#ifdef RESULT
warning: for p.o. reduction to be valid the never claim must be stutter-invariant
(never claims generated from LTL formulae are stutter-invariant)
depth 0: Claim reached state 7 (line 282)
depth 0: Claim reached state 7 (line 283)
depth 2: Claim reached state 17 (line 292)
depth 2: Claim reached state 17 (line 293)
depth 302: Claim reached state 17 (line 291)
Depth=     319 States=   1e+006 Transitions= 3.59e+006 Memory=     8.903  t=     20 R=
5e+004
Depth=     319 States=   2e+006 Transitions= 7.19e+006 Memory=     8.903  t=   38.5 R=
5e+004
Depth=     319 States=   3e+006 Transitions= 1.14e+007 Memory=     8.903  t=   61.5 R=
5e+004
Depth=     319 States=   4e+006 Transitions= 1.58e+007 Memory=     8.903  t=   85.2 R=
5e+004
Depth=     319 States=   5e+006 Transitions= 2.02e+007 Memory=     8.903  t=    107 R=
5e+004
Depth=     319 States=   6e+006 Transitions= 2.43e+007 Memory=     8.903  t=    128 R=
5e+004
Depth=     319 States=   7e+006 Transitions= 2.7e+007 Memory=     8.903   t=    143 R=
5e+004
Depth=     319 States=   8e+006 Transitions= 2.98e+007 Memory=     8.903  t=    159 R=
5e+004
Depth=     319 States=   9e+006 Transitions= 3.26e+007 Memory=     8.903  t=    174 R=
5e+004
Depth=     319 States=   1e+007 Transitions= 3.49e+007 Memory=     8.903  t=    188 R=
5e+004
Depth=     319 States= 1.1e+007 Transitions= 3.72e+007 Memory=     8.903  t=    201 R=
5e+004
Depth=     319 States= 1.2e+007 Transitions= 3.94e+007 Memory=     8.903  t=    214 R=
6e+004
Depth=     319 States= 1.3e+007 Transitions= 4.17e+007 Memory=     8.903  t=    227 R=
6e+004
Depth=     319 States= 1.4e+007 Transitions= 4.4e+007 Memory=     8.903   t=    240 R=
6e+004
Depth=     319 States= 1.5e+007 Transitions= 4.67e+007 Memory=     8.903  t=    255 R=
6e+004
Depth=     319 States= 1.6e+007 Transitions= 4.94e+007 Memory=     8.903  t=    270 R=
6e+004
Depth=     319 States= 1.7e+007 Transitions= 5.21e+007 Memory=     8.903  t=    285 R=
6e+004
Depth=     319 States= 1.8e+007 Transitions= 5.48e+007 Memory=     8.903  t=    300 R=
6e+004
Depth=     319 States= 1.9e+007 Transitions= 5.76e+007 Memory=     8.903  t=    315 R=
6e+004
Depth=     319 States=   2e+007 Transitions= 6.02e+007 Memory=     8.903  t=    330 R=
6e+004
Depth=     319 States= 2.1e+007 Transitions= 6.27e+007 Memory=     8.903  t=    344 R=
6e+004
Depth=     319 States= 2.2e+007 Transitions= 6.52e+007 Memory=     8.903  t=    358 R=
6e+004
```

```
Depth=     319 States= 2.3e+007 Transitions= 6.82e+007 Memory=     8.903  t=    374 R=
6e+004
Depth=     325 States= 2.4e+007 Transitions= 7.17e+007 Memory=     8.903  t=    392 R=
6e+004
Depth=     325 States= 2.5e+007 Transitions= 7.5e+007 Memory=     8.903   t=    410 R=
6e+004
Depth=     325 States= 2.6e+007 Transitions= 7.83e+007 Memory=     8.903  t=    429 R=
6e+004
Depth=     325 States= 2.7e+007 Transitions= 8.17e+007 Memory=     8.903  t=    448 R=
6e+004
Depth=     349 States= 2.8e+007 Transitions= 8.48e+007 Memory=     8.903  t=    465 R=
6e+004
Depth=     678 States= 2.9e+007 Transitions= 8.84e+007 Memory=     9.099  t=    484 R=
6e+004
Depth=     678 States=   3e+007 Transitions= 9.2e+007 Memory=     9.099   t=    505 R=
6e+004
Depth=     678 States= 3.1e+007 Transitions= 9.57e+007 Memory=     9.099  t=    525 R=
6e+004

(Spin Version 5.1.6 -- 9 May 2008)
       + Partial Order Reduction

Bit statespace search for:
       never claim           +
       assertion violations  + (if within scope of claim)
       acceptance   cycles   + (fairness disabled)
       invalid end states    - (disabled by never claim)

State-vector 520 byte, depth reached 678, errors: 0
 31939009 states, stored
 67304821 states, matched
 99243830 transitions (= stored+matched)
  5271672 atomic steps

hash factor: 2.10116 (best if > 100.)

bits set per state: 3 (-k3)

Stats on memory usage (in Megabytes):
16326.245      equivalent memory usage for states (stored*(State-vector + overhead))
    8.000      memory used for hash array (-w26)
    0.076      memory used for bit stack
    0.610      memory used for DFS stack (-m20000)
    9.099      total actual memory usage


pan: elapsed time 546 seconds
pan: rate 58448.396 states/second
pan: avg transition delay 5.5061e-006 usec

#endif
```

B. Modelo UPPAAL - XML


```
<?xml version="1.0" encoding="utf-8" ?>
  <!DOCTYPE nta (View Source for full doctype...)>
- <nta>
  <declaration>
       // Estados do Módulo de Voto
       const int INICIO:=0;
       const int PedidoBoletimEleicao := 1;
              const int EnvioBoletimEleicao := 2;
       const int PedidoBoletimVoto := 3;
       const int EnvioBoletimVoto := 4;
       const int PedidoAssinaturaVoto := 5;
       const int EnvioAssinaturaVoto := 6;
       const int EnvioSubmissaoVoto := 7;

       //Numero Total de Eleitores
       const int gMaximoEleitores := 2;
       typedef int[0,gMaximoEleitores-1] id_Eleitor;

       // Numero Total de Administradores
       const int gMaximoAdministradores := 3;
       typedef int[0,gMaximoAdministradores-1] id_Administrador;

       // Numero de Distribuidores
       const int gMaximoDistribuidores := 2;
       typedef int[0,gMaximoDistribuidores-1] id_Distribuidor;

       //Numero de Anonimizadores e respectivsamente Totalizadores - Igual para
       Totalizadores e Anonimizadores
       const int gMaximoTotalizadoresAnonimizadores := 2;
       typedef int[0,gMaximoTotalizadoresAnonimizadores-1] id_TotalizadorAnonimizador;

       // Valor para esconder o boletim de voto
       const int DELTA:=10;

       // Numero maximo de votos no escrutinio
       const int gMaximoEscrutinio := 5;
       typedef int[0,gMaximoEscrutinio-1] id_MaximoEscrutinio;

       // Tempo de atraso nos votos
       const int gAtrasovotos :=5;

       // Tempo utilizado no temporizador
       const int gTempoTotalEleicao := 500;
       // Numero de vezes que o temporizador e reiniciado
       const int gNPeriodosTempoDecorrer :=20;

       // Numero corrente de periodos de tempo decorridos - Temporizador
       int gNPeriodosTempoCorrente :=0;

       // Indica se Temporizador esta activo
       bool Activo:=false;

       // Verdadeiro quando decorrido o tempo definido para a votação
       bool iniciarEscrutinio := false;

       // Indica o primeiro eleitor a chegar ao anonimizador
       int gPrimeiroEleitor;

       // Numero de Administradores, necessario para validar um voto
       int gNumAssAdministradores := gMaximoAdministradores /2+1;
```

```
        // Indice dos votos recebidos
        int iVotosRecebidos;

        // Para comunicacao entre Eleitor e Distribuidor
        chan cEleitor[gMaximoEleitores];

        // Para comunicacao entre Eleitor e Distribuidor
        chan cDistribuidor[gMaximoDistribuidores];

        // Para comunicacao entre Eleitor e Administrador
        chan cAdministrador[gMaximoAdministradores];

        // Para comunicacao entre Eleitor e Anonimizador
        chan cAnonimizador[gMaximoEleitores];

        // Para comunicacao entre Anonimizador e Totalizador
        chan cTotalizador[gMaximoTotalizadoresAnonimizadores];

        //Para activar o Temporizador
        chan Atribuir;

        //Informacao relevante do eleitor
        typedef struct { int Estado;
        int NumAssinaturas;
        bool AssinaturaAdministrador[3]; }recEleitor;

        //Guarda informacao sobre o Voto, nomeadamente: Identificacaodo voto, Numero de
        assinaturas recolhidas
        typedef struct{ int IdVoto;
        int NumAssinaturas; }recVoto;

        //Todos os eleitores validos
        recEleitor gEleitor[gMaximoEleitores];

        //Guardar todos os votos efectuados pelo Eleitor, incluindo repetidos
        recVoto gVotoRecebido[id_MaximoEscrutinio];

        //Guardar apenas os votos correctos, eliminando repetições e erros recVoto
        gVotoApurado[id_MaximoEscrutinio];</declaration>
- <template>
  <name x="5" y="5">ModuloVoto</name>
  <parameter>const id_Eleitor pEleitor</parameter>
  <declaration>id_Distribuidor lDistribuidorCorrente;</declaration>
- <location id="id0" x="-1992" y="32">
  <name x="-2008" y="48">FIM</name>
  </location>
- <location id="id1" x="-2064" y="-888">
  <name x="-2080" y="-928">INICIO</name>
  </location>
- <location id="id2" x="-1992" y="-320">
  <name x="-2160" y="-352">ObtemProximaAssinatura</name>
  <label kind="invariant" x="-2304" y="-
320">gEleitor[pEleitor].Estado==EnvioAssinaturaVoto</label>
  </location>
- <location id="id3" x="-1296" y="-320">
  <name x="-1288" y="-312">PedidoAssinaturaVotoEnviado</name>
  </location>
- <location id="id4" x="-1296" y="-528">
  <name x="-1280" y="-552">PedidoBoletimVotoRecebido</name>
  <label kind="invariant" x="-1288" y="-
520">gEleitor[pEleitor].Estado==EnvioBoletimVoto</label>
  </location>
- <location id="id5" x="-1592" y="-528">
  <name x="-1672" y="-568">PedidoBoletimVotoEnviado</name>
  </location>
- <location id="id6" x="-2064" y="-712">
  <name x="-2264" y="-720">PedidoBoletimEleicaoEnviado</name>
  </location>
- <location id="id7" x="-2064" y="-528">
  <name x="-2272" y="-560">PedidoBoletimEleicaoRecebido</name>
  <label kind="invariant" x="-2200" y="-
512">gEleitor[pEleitor].Estado==EnvioBoletimEleicao</label>
  </location>
  <init ref="id1" />
- <transition>
  <source ref="id2" />
```

```
  <target ref="id0" />
  <label kind="guard" x="-1984" y="-
80">gEleitor[pEleitor].NumAssinaturas==gNumAssAdministradores</label>
  <label kind="synchronisation" x="-1984" y="-56">cAnonimizador[pEleitor]!</label>
  <label kind="assignment" x="-1984" y="-
32">gEleitor[pEleitor].Estado:=EnvioSubmissaoVoto</label>
  </transition>
- <transition>
  <source ref="id3" />
  <target ref="id2" />
  <label kind="select" x="-1888" y="-400">lAdministrador :id_Administrador</label>
  <label kind="guard" x="-1888" y="-
376">gEleitor[pEleitor].AssinaturaAdministrador[lAdministrador]!=0</label>
  <label kind="synchronisation" x="-1888" y="-
352">cAdministrador[lAdministrador]?</label>
  <nail x="-1400" y="-400" />
  <nail x="-1928" y="-400" />
  </transition>
- <transition>
  <source ref="id2" />
  <target ref="id3" />
  <label kind="guard" x="-1888" y="-184">gEleitor[pEleitor].NumAssinaturas <
gNumAssAdministradores</label>
  <label kind="synchronisation" x="-1888" y="-160">cEleitor[pEleitor]!</label>
  <label kind="assignment" x="-1888" y="-
136">gEleitor[pEleitor].Estado:=PedidoAssinaturaVoto</label>
  <nail x="-1928" y="-200" />
  <nail x="-1400" y="-200" />
  </transition>
- <transition>
  <source ref="id1" />
  <target ref="id6" />
  <label kind="synchronisation" x="-2056" y="-848">cEleitor[pEleitor]!</label>
  <label kind="assignment" x="-2056" y="-
824">gEleitor[pEleitor].Estado:=PedidoBoletimEleicao</label>
  </transition>
- <transition>
  <source ref="id3" />
  <target ref="id2" />
  <label kind="select" x="-1888" y="-320">lAdministrador :id_Administrador</label>
  <label kind="guard" x="-1888" y="-
296">gEleitor[pEleitor].AssinaturaAdministrador[lAdministrador]==0</label>
  <label kind="synchronisation" x="-1888" y="-
272">cAdministrador[lAdministrador]?</label>
  <label kind="assignment" x="-1888" y="-
248">gEleitor[pEleitor].AssinaturaAdministrador[lAdministrador]:=1,
gEleitor[pEleitor].NumAssinaturas:=gEleitor[pEleitor].NumAssinaturas+1</label>
  </transition>
- <transition>
  <source ref="id4" />
  <target ref="id3" />
  <label kind="synchronisation" x="-1288" y="-416">cEleitor[pEleitor]!</label>
  <label kind="assignment" x="-1288" y="-
440">gEleitor[pEleitor].Estado:=PedidoAssinaturaVoto</label>
  </transition>
- <transition>
  <source ref="id5" />
  <target ref="id4" />
  <label kind="select" x="-1536" y="-528">lDistribuidor : id_Distribuidor</label>
  <label kind="synchronisation" x="-1536" y="-504">cDistribuidor[lDistribuidor]?</label>
  </transition>
- <transition>
  <source ref="id7" />
  <target ref="id5" />
  <label kind="synchronisation" x="-1896" y="-528">cEleitor[pEleitor]!</label>
  <label kind="assignment" x="-1896" y="-
504">gEleitor[pEleitor].Estado:=PedidoBoletimVoto</label>
  </transition>
- <transition>
  <source ref="id6" />
  <target ref="id7" />
  <label kind="select" x="-2056" y="-664">lDistribuidor : id_Distribuidor</label>
  <label kind="synchronisation" x="-2056" y="-640">cDistribuidor[lDistribuidor]?</label>
  <label kind="assignment" x="-2056" y="-
616">lDistribuidorCorrente:=lDistribuidor</label>
  </transition>
  </template>
```

```
- <template>
  <name>Administrador</name>
  <parameter>const id_Administrador pAdministrador</parameter>
  <declaration>id_Eleitor lEleitorCorrente;</declaration>
- <location id="id8" x="-1288" y="-440">
  <name x="-1272" y="-456">PedidoAssinaturaVotoRecebido</name>
  <label kind="invariant" x="-1280" y="-
424">gEleitor[lEleitorCorrente].Estado==PedidoAssinaturaVoto</label>
  </location>
- <location id="id9" x="-1552" y="-568">
  <name x="-1576" y="-600">INICIO</name>
  </location>
  <init ref="id9" />
- <transition>
  <source ref="id8" />
  <target ref="id9" />
  <label kind="synchronisation" x="-1280" y="-
528">cAdministrador[pAdministrador]!</label>
  <label kind="assignment" x="-1280" y="-
504">gEleitor[lEleitorCorrente].Estado:=EnvioAssinaturaVoto</label>
  <nail x="-1288" y="-568" />
  </transition>
- <transition>
  <source ref="id9" />
  <target ref="id8" />
  <label kind="select" x="-1544" y="-432">lEleitor: id_Eleitor</label>
  <label kind="synchronisation" x="-1544" y="-408">cEleitor[lEleitor]?</label>
  <label kind="assignment" x="-1544" y="-384">lEleitorCorrente:=lEleitor</label>
  <nail x="-1552" y="-440" />
  </transition>
  </template>
- <template>
  <name>Distribuidor</name>
  <parameter>const id_Distribuidor pDistribuidor</parameter>
  <declaration>id_Eleitor lEleitorCorrente;</declaration>
- <location id="id10" x="-2608" y="-1008">
  <name x="-2800" y="-1032">PedidoBoletimVotoRecebido</name>
  <label kind="invariant" x="-2800" y="-984">gEleitor[lEleitorCorrente].Estado ==
PedidoBoletimVoto</label>
  </location>
- <location id="id11" x="-2104" y="-1008">
  <name x="-2152" y="-984">PedidoBoletimEleicaoEnviado</name>
  </location>
- <location id="id12" x="-2104" y="-1208">
  <name x="-2184" y="-1248">PedidoBoletimEleicaoRecebido</name>
  <label kind="invariant" x="-2088" y="-
1208">gEleitor[lEleitorCorrente].Estado==PedidoBoletimEleicao</label>
  </location>
- <location id="id13" x="-2608" y="-1208">
  <name x="-2624" y="-1248">INICIO</name>
  </location>
  <init ref="id13" />
- <transition>
  <source ref="id13" />
  <target ref="id12" />
  <label kind="select" x="-2440" y="-1288">lEleitor :id_Eleitor</label>
  <label kind="synchronisation" x="-2440" y="-1264">cEleitor[lEleitor]?</label>
  <label kind="assignment" x="-2440" y="-1240">lEleitorCorrente:=lEleitor</label>
  </transition>
- <transition>
  <source ref="id10" />
  <target ref="id13" />
  <label kind="synchronisation" x="-2600" y="-
1144">cDistribuidor[pDistribuidor]!</label>
  <label kind="assignment" x="-2600" y="-
1120">gEleitor[lEleitorCorrente].Estado:=EnvioBoletimVoto</label>
  </transition>
- <transition>
  <source ref="id11" />
  <target ref="id10" />
  <label kind="select" x="-2392" y="-1000">lEleitor :id_Eleitor</label>
  <label kind="synchronisation" x="-2392" y="-976">cEleitor[lEleitor]?</label>
  <label kind="assignment" x="-2392" y="-952">lEleitorCorrente:=lEleitor</label>
  </transition>
- <transition>
  <source ref="id12" />
  <target ref="id11" />
```

```
  <label kind="synchronisation" x="-2096" y="-
1144">cDistribuidor[pDistribuidor]!</label>
  <label kind="assignment" x="-2096" y="-
1120">gEleitor[lEleitorCorrente].Estado:=EnvioBoletimEleicao</label>
  </transition>
  </template>
- <template>
  <name>Anonimizador</name>
  <parameter>const id_TotalizadorAnonimizador pAnonimizador</parameter>
  <declaration>int lEleitorCorrente; clock lAtrasoVoto; void PrimeiroEleitor() { if
(gPrimeiroEleitor == 0) gPrimeiroEleitor:=lEleitorCorrente + DELTA; }</declaration>
- <location id="id14" x="-952" y="104">
  <name x="-1112" y="88">EfectuarAnonimizacao</name>
  <label kind="invariant" x="-1184" y="120">lAtrasoVoto <=
gAtrasovotos+gNPeriodosTempoDecorrer*gTempoTotalEleicao</label>
  </location>
- <location id="id15" x="-752" y="-160">
  <name x="-776" y="-200">INICIO</name>
  </location>
  <init ref="id15" />
- <transition>
  <source ref="id15" />
  <target ref="id15" />
  <label kind="guard" x="-352" y="-216">Activo == false and gNPeriodosTempoCorrente ==
gNPeriodosTempoDecorrer</label>
  <label kind="synchronisation" x="-352" y="-192">cTotalizador[pAnonimizador]!</label>
  <nail x="-640" y="-224" />
  <nail x="-360" y="-224" />
  <nail x="-360" y="-160" />
  </transition>
- <transition>
  <source ref="id15" />
  <target ref="id15" />
  <label kind="guard" x="-1344" y="-232">Activo == false and gNPeriodosTempoCorrente <
gNPeriodosTempoDecorrer</label>
  <label kind="synchronisation" x="-1344" y="-208">Atribuir!</label>
  <label kind="assignment" x="-1344" y="-184">gNPeriodosTempoCorrente++</label>
  <nail x="-1352" y="-160" />
  <nail x="-1352" y="-232" />
  <nail x="-864" y="-232" />
  </transition>
- <transition>
  <source ref="id15" />
  <target ref="id14" />
  <label kind="select" x="-944" y="-32">lEleitor: id_Eleitor</label>
  <label kind="guard" x="-944" y="-8">Activo == true</label>
  <label kind="synchronisation" x="-944" y="16">cAnonimizador[lEleitor]?</label>
  <label kind="assignment" x="-944" y="40">lEleitorCorrente:=lEleitor, lAtrasoVoto:=0,
PrimeiroEleitor()</label>
  <nail x="-952" y="-56" />
  </transition>
- <transition>
  <source ref="id14" />
  <target ref="id15" />
  <label kind="guard" x="-536" y="-40">lAtrasoVoto ==
gAtrasovotos+gNPeriodosTempoDecorrer*gTempoTotalEleicao</label>
  <label kind="synchronisation" x="-536" y="-16">cTotalizador[pAnonimizador]!</label>
  <label kind="assignment" x="-536"
y="8">gVotoRecebido[iVotosRecebidos].IdVoto:=lEleitorCorrente+DELTA,
gVotoRecebido[iVotosRecebidos].NumAssinaturas:=gEleitor[lEleitorCorrente].NumAssinaturas
, iVotosRecebidos:=iVotosRecebidos+1, lAtrasoVoto:=0</label>
  <nail x="-552" y="104" />
  <nail x="-552" y="-56" />
  </transition>
  </template>
- <template>
  <name>Totalizador</name>
  <parameter>const id_TotalizadorAnonimizador pTotalizador</parameter>
  <declaration>
       id_TotalizadorAnonimizador lTotalizadorCorrente;
       /*Fazer a contagem dos votos recebidos, contando apenas os válidos e eliminando r
       epetições*/
       void escrutinio()
       { int indexR, indexA := 0;
       while ( indexR< gMaximoEscrutinio)
       {if ((gVotoApurado[indexA].IdVoto==0) and (gVotoRecebido[indexR].IdVoto != 0))
```

```
              {gVotoApurado[indexA].IdVoto := gVotoRecebido[indexR].IdVoto;
              gVotoApurado[indexA].NumAssinaturas =
              gVotoRecebido[indexR].NumAssinaturas; indexA:=0; indexR++;}
       if ((gVotoApurado[indexA].IdVoto !=0) and (gVotoRecebido[indexR].IdVoto != 0) and
       (gVotoRecebido[indexR].IdVoto) != (gVotoApurado[indexA].IdVoto ))
              indexA++;
       if (gVotoRecebido[indexR].IdVoto==0) indexR++; } }
</declaration>
- <location id="id16" x="168" y="320">
  <name x="158" y="290">FIM</name>
  </location>
- <location id="id17" x="168" y="32">
  <name x="112" y="8">INICIO</name>
  </location>
  <init ref="id17" />
- <transition>
  <source ref="id17" />
  <target ref="id16" />
  <label kind="guard" x="176" y="152">iniciarEscrutinio == true</label>
  </transition>
- <transition>
  <source ref="id17" />
  <target ref="id16" />
  <label kind="select" x="384" y="128">lTotalizador: id_TotalizadorAnonimizador</label>
  <label kind="guard" x="384" y="176">gNPeriodosTempoCorrente == gNPeriodosTempoDecorrer
and lTotalizador==pTotalizador</label>
  <label kind="synchronisation" x="384" y="152">cTotalizador[lTotalizador]?</label>
  <label kind="assignment" x="384" y="200">escrutinio(), gNPeriodosTempoCorrente++,
iniciarEscrutinio:=true</label>
  <nail x="368" y="112" />
  <nail x="368" y="256" />
  </transition>
- <transition>
  <source ref="id17" />
  <target ref="id17" />
  <label kind="select" x="392" y="-24">lTotalizador: id_TotalizadorAnonimizador</label>
  <label kind="guard" x="392" y="0">lTotalizador==pTotalizador and
gNPeriodosTempoCorrente < gNPeriodosTempoDecorrer</label>
  <label kind="synchronisation" x="392" y="24">cTotalizador[lTotalizador]?</label>
  <label kind="assignment" x="392" y="48">lTotalizadorCorrente:=lTotalizador</label>
  <nail x="376" y="64" />
  <nail x="376" y="-40" />
  </transition>
  </template>
- <template>
  <name>Temporizador</name>
  <parameter>int t0</parameter>
  <declaration>clock x;</declaration>
- <location id="id18" x="312" y="8">
  <name x="320" y="-24">Aguardar</name>
  <label kind="invariant" x="328" y="8">x<=t0</label>
  </location>
- <location id="id19" x="8" y="8">
  <name x="-72" y="-24">EXPIRADO</name>
  </location>
  <init ref="id19" />
- <transition>
  <source ref="id18" />
  <target ref="id19" />
  <label kind="guard" x="128" y="128">x == t0</label>
  <label kind="assignment" x="128" y="152">Activo:=false</label>
  <nail x="280" y="128" />
  <nail x="48" y="128" />
  </transition>
- <transition>
  <source ref="id19" />
  <target ref="id18" />
  <label kind="synchronisation" x="128" y="-96">Atribuir?</label>
  <label kind="assignment" x="128" y="-80">x:=0, Activo:=true</label>
  <nail x="56" y="-96" />
  <nail x="272" y="-96" />
  </transition>
  </template>
  <system>
// Processos que constituem o sistema e que sao instanciados com os valores possiveis
para os respectivos parametros
ProcessoTemporizador= Temporizador(gTempoTotalEleicao);
```

```
system
ModuloVoto, Administrador, Distribuidor, Anonimizador,Totalizador, ProcessoTemporizador;
</system>
  </nta>
```

B.1. Propriedades UPPAAL

```
//This file was generated from (Academic) UPPAAL 4.0.8 (rev. 4276), March 2009

/*
5.4.2 - Democracia: So os eleitores elegiveis votam e uma unica vez
*/
A[] forall (i:id_MaximoEscrutinio) gVotoApurado[i].IdVoto == 1 + DELTA  or
gVotoApurado[i].IdVoto == 1+ DELTA   or gVotoApurado[i].IdVoto == 1+ DELTA  or
gVotoApurado[i].IdVoto or gVotoApurado[i].IdVoto == 0

/*
5.4.3  Precisao
Se um voto for contado no escrutinio final, ele foi efectuado por um eleitor valido
*/
A[] forall (i: id_MaximoEscrutinio) gVotoApurado[i].IdVoto != 0 imply  forall (j:
id_MaximoEscrutinio) gVotoRecebido[i].IdVoto == 0+DELTA or gVotoRecebido[i].IdVoto  ==
1+DELTA

/*
5.4.5  REVS \u2013 Assinaturas para o voto ser valido
"Um eleitor que termine o seu processo de voto garantidamente obteve um n\u00famero de
assinaturas suficiente para o voto ser valido"
*/
A[] forall (i: id_Eleitor) ModuloVoto(i).FIM imply gEleitor[i].NumAssinaturas ==
gNumAssAdministradores

/*
5.4.6  REVS \u2013 Eleitor termina processo de votacao
"O Eleitor possivelmente termina o seu processo de votacao."
*/
A<> iniciarEscrutinio==1 imply forall (i:id_Eleitor) ModuloVoto(i).FIM

/*
5.4.7  REVS \u2013 Eleitor pode nao terminar processo de votacao
\u201cO Eleitor pode nao terminar o seu processo de votacao.\u201d

*/
A[] iniciarEscrutinio == 1 imply  forall (i:id_Eleitor) ModuloVoto(i).FIM

/*
5.4.8  REVS \u2013 Ordem dos votos
A ordem de recepcao do voto no Anonimizador nao e sempre igual a ordem de votos
recebidos.
*/
A[] iniciarEscrutinio ==1 and iVotosRecebidos== gMaximoEleitores and gPrimeiroEleitor==1
+DELTA imply gVotoRecebido[0].IdVoto == 1+DELTA

/*
5.4.9  Apenas verificavel em UPPAAL \u2013 Logica Temporal Ramificada
1 - "Quando o escrutinio inicia, ja todos os votos chegaram ao Anonimizador e
Totalizador."
*/
A[] iniciarEscrutinio == 1 imply forall (i:id_Eleitor) gVotoRecebido[i].IdVoto != 0 and
gVotoApurado[i].IdVoto != 0

/*
5.4.9  Apenas verificavel em UPPAAL \u2013 Logica Temporal Ramificada
2 - "Quando o escrutinio inicia, eventualmente ja todos os votos chegaram ao
Anonimizador e Totalizador."
*/
A<> iniciarEscrutinio == 1 imply forall (i:id_Eleitor) gVotoRecebido[i].IdVoto != 0 and
gVotoApurado[i].IdVoto != 0

/*
5.4.9  Apenas verificavel em UPPAAL \u2013 Logica Temporal Ramificada
3 - \u201cExiste um caminho onde o voto do eleitor e sempre recebido e apurado.\u201d
*/
E[] forall (i:id_Eleitor) gVotoRecebido[i].IdVoto != 0 and gVotoApurado[i].IdVoto != 0
```

```
/*
5.4.9  Apenas verificavel em UPPAAL \u2013 Logica Temporal Ramificada
4 -  \u201cE possivel que exista um caminho onde o voto do eleitor e eventualmente
recebido e apurado.\u201d
```

C. Modelo SPIN – Intruso

```
#define PedidoBoletimEleicao 1;             /*Mensagens trocadas entre o Modulo de Voto
e Distribuidor*/
#define EnvioBoletimEleicao 2;
#define PedidoBoletimVoto 3;
#define EnvioBoletimVoto 4;
#define PedidoAssinaturaVoto 5;
#define EnvioAssinaturaVoto 6;
#define EnvioSubmissaoVoto 7;

#define INICIO 0;                                              /*Estados do Modulo de Voto*/
#define PedidoBoletimEleicaoEnviado 1;
#define PedidoBoletimVotoEnviado 2;
#define PedidoAssinaturaVotoEnviado 3;
#define TotalAssinaturasObtido 4;
#define FIM 5;

#define NumDistribuidores 2;                        /* Numero de processos do modelo */
#define NumAdministradores 3;
#define NumEleitores 3;
#define NumAnonimizadores 2;
#define NumTotalizadores 2;
#define CapacidadeEscrutinio 6;
#define NumTotalAssinaturas 2;
#define DELTA 10;

int selVoto=0, selAnonimizador=0, selDistribuidor=0;
int iniciarEscrutinio = 0;

int indexA=0, indexR=0;

int  lostMsg = 0, eleitoresContados = 0;

#define SelDistribuidor() selDistribuidor = (selDistribuidor+1) % NumDistribuidores
#define SelAnonimizador() selAnonimizador = (selAnonimizador+1) % NumAnonimizadores
#define SelAdministrador() admin = (admin+1) % NumAdministradores

#define randomNumber()       if  \
                                            :: count = 15 \
                                            :: count = 30 \
                                            :: count = 45 \
                                            :: count = 90 \
                                            fi


/* Comunicacao entre Eleitor e Distribuidor */
chan gEleitorDistribuidor = [10] of {byte, byte, byte};

/* Comunicacao entre Eleitor e Administrador */
chan gEleitorAdministrador = [10] of {byte, byte,byte};

/* Comunicacao entre Eleitor e Anonimizadorr */
chan gEleitorAnonimizador = [10] of {byte, byte, byte};

/* Comunicacao entre Eleitor e Totalizador */
chan gAnonimizadorTotalizador = [10] of {byte, byte, byte};

/* Delay entre o Anonimizador e Totalizador */
chan gDelayAnonimizadorTotalizador = [10] of {byte, byte, byte};

/* Informacao sobre o Eleitor: estado, assinaturas, numero assinaturas recolhidas */
typedef  recEleitor {
       int Estado;
       bool AssinaturaAdministrador[3];
       int NumAssinaturas;
};
```

```
/* Informacao sobre o Voto: identificacaodo voto, numero de assinaturas recolhidas */
typedef  recVoto {
        int IdVoto;
        int NumAssinaturas
};

recEleitor gEleitor[3];
recVoto gVotoRecebido[9];
recVoto gVotoApurado[9];

proctype Intruso()
{
        int eleitor, assinaturas, anonimizador;

        end:
        do
        :: iniciarEscrutinio == 0 -> gEleitorAnonimizador?<eleitor,assinaturas,
anonimizador>;
                              gDelayAnonimizadorTotalizador!eleitor,assinaturas,
anonimizador;
        od
}


proctype ModuloVoto(int pEleitor)
{
        int lDistribuidor;
        int admin = 0;

        do
        :: gEleitor[pEleitor].Estado == INICIO ->
                atomic {
                        SelDistribuidor();
                        gEleitorDistribuidor!pEleitor, selDistribuidor,
PedidoBoletimEleicao;
                        gEleitor[pEleitor].Estado = PedidoBoletimEleicaoEnviado;
                }
        :: gEleitor[pEleitor].Estado == PedidoBoletimEleicaoEnviado ->
                gEleitorDistribuidor?eval(pEleitor), lDistribuidor, EnvioBoletimEleicao -
>
                atomic {
                        gEleitorDistribuidor!pEleitor, lDistribuidor, PedidoBoletimVoto;
                        gEleitor[pEleitor].Estado = PedidoBoletimVotoEnviado
                }
        :: gEleitor[pEleitor].Estado == PedidoBoletimVotoEnviado ->
                gEleitorDistribuidor?eval(pEleitor), lDistribuidor, EnvioBoletimVoto ->
                atomic {
                        SelAdministrador();
                        gEleitorAdministrador!pEleitor, admin, PedidoAssinaturaVoto;
                        gEleitor[pEleitor].Estado = PedidoAssinaturaVotoEnviado;
                }
        :: gEleitor[pEleitor].Estado == PedidoAssinaturaVotoEnviado  ->
                do
                :: gEleitor[pEleitor].NumAssinaturas < NumTotalAssinaturas ->
                        gEleitorAdministrador?eval(pEleitor), eval(admin),
EnvioAssinaturaVoto;
                        if
                        :: gEleitor[pEleitor].AssinaturaAdministrador[admin] == 0 ->
                                atomic {
                                        gEleitor[pEleitor].AssinaturaAdministrador[admin] =
1;
                                        gEleitor[pEleitor].NumAssinaturas++;
                                }
                        fi;
                        if
                        :: gEleitor[pEleitor].NumAssinaturas < NumTotalAssinaturas ->
                                atomic {
                                        SelAdministrador();
                                        gEleitorAdministrador!pEleitor,admin,
PedidoAssinaturaVoto;
                                }
                        :: gEleitor[pEleitor].NumAssinaturas == NumTotalAssinaturas ->
                                gEleitor[pEleitor].Estado = TotalAssinaturasObtido;
                                break;
                        fi;
                od;
        :: gEleitor[pEleitor].Estado == TotalAssinaturasObtido ->
```

```
		atomic {
			SelAnonimizador();
			gEleitorAnonimizador!pEleitor,gEleitor[pEleitor].NumAssinaturas,
selAnonimizador;
			gEleitor[pEleitor].Estado = FIM;
		}
	:: gEleitor[pEleitor].Estado == FIM -> break;
	od;
}

proctype Distribuidor(byte pDistribuidor)
{
	int Eleitor=0;

	end:
	do
	:: gEleitorDistribuidor?Eleitor, eval(pDistribuidor), PedidoBoletimEleicao ->
		gEleitorDistribuidor!Eleitor, pDistribuidor, EnvioBoletimEleicao
	:: gEleitorDistribuidor?Eleitor, eval(pDistribuidor), PedidoBoletimVoto ->
		gEleitorDistribuidor!Eleitor, pDistribuidor, EnvioBoletimVoto
	od
}

proctype Administrador(int pAdmin)
{
	int Eleitor = 0;

	end:
	do
	:: gEleitorAdministrador?Eleitor,eval(pAdmin),PedidoAssinaturaVoto ->
		gEleitorAdministrador!Eleitor,pAdmin,EnvioAssinaturaVoto
	od
}

proctype Anonimizador(int pAnonimizador)
{
	int Eleitor, lidEleitor, lTotalAssinaturas = 0;

	endAnonimizador: do
	:: gEleitorAnonimizador??lidEleitor, lTotalAssinaturas, eval(pAnonimizador) ->
		gDelayAnonimizadorTotalizador!lidEleitor, lTotalAssinaturas,
pAnonimizador;
	:: gDelayAnonimizadorTotalizador??lidEleitor, lTotalAssinaturas,
eval(pAnonimizador) ->
		atomic { Eleitor =lidEleitor+DELTA; gAnonimizadorTotalizador!Eleitor,
lTotalAssinaturas, pAnonimizador;}
	od;
}

proctype Totalizador(int pTotalizador)
{
	int x,y,z;
	do
	:: iniciarEscrutinio == 0 && selVoto < CapacidadeEscrutinio ->
		if
		:: atomic{

	gAnonimizadorTotalizador??gVotoRecebido[selVoto].IdVoto,gVotoRecebido[selVoto].Nu
mAssinaturas, eval(pTotalizador);
			selVoto++;
			};
		:: empty(gAnonimizadorTotalizador) -> skip;
		fi;
	:: iniciarEscrutinio  == 1 -> break;
	:: else -> skip;
	od;
	if
	:: pTotalizador == 0 ->
		do
		:: indexR < selVoto ->
			if
			:: ((gVotoApurado[indexA].IdVoto != gVotoRecebido[indexR].IdVoto)
&& (gVotoApurado[indexA].IdVoto == 0)) ->
				atomic {
					gVotoApurado[indexA].IdVoto =
gVotoRecebido[indexR].IdVoto;
```

```
                                    gVotoApurado[indexA].NumAssinaturas =
gVotoRecebido[indexR].NumAssinaturas;
                                    indexR++;
                                    indexA=0; eleitoresContados++;
                             }
                      :: ((gVotoApurado[indexA].IdVoto != gVotoRecebido[indexR].IdVoto)
&& (gVotoApurado[indexA].IdVoto != 0)) -> indexA++;
                      :: ((gVotoApurado[indexA].IdVoto == gVotoRecebido[indexR].IdVoto)
&& (gVotoApurado[indexA].IdVoto != 0)) -> indexR++; indexA=0;
                      :: else -> indexR++;
                      fi
               :: else -> break;
               od;
       :: else  -> skip;
       fi;

       end:
       do
       :: len(gAnonimizadorTotalizador) > 0 ->
              gAnonimizadorTotalizador??x,y,z -> lostMsg++;
       :: len(gAnonimizadorTotalizador) == 0 -> break
       od;

       end1:
       do
       :: len(gDelayAnonimizadorTotalizador)  > 0 ->
                                    gDelayAnonimizadorTotalizador??x,y,z -> lostMsg++;
       len(gDelayAnonimizadorTotalizador) == 0 -> break
       od;
}

proctype Temporizador()
{
       int count;
       int i = 0;

       randomNumber();
       do
       :: i < count -> i++;
       :: i == count -> break;
       od;

       iniciarEscrutinio = 1;
}

init
{
       int id = 0;

       run Intruso();

       /* Lancar um numero variavel de Eleitores */
       do
       :: id < NumEleitores -> atomic { run ModuloVoto(id); id++ }
       :: id == NumEleitores -> id=0; break
       od;

       /* Lancar um numero variavel de Distribuidores */

       do
       :: id < NumDistribuidores -> atomic { run Distribuidor(id); id++ }
       :: id == NumDistribuidores -> id=0; break
       od;

       /* Lancar um numero variavel de Administradores */
       do
       :: id < NumAdministradores -> atomic { run Administrador(id); id++ }
       :: id == NumAdministradores -> id=0; break
       od ;

       /* Lancar um numero variavel de Anonimizadores */
       do
       :: id < NumAnonimizadores -> atomic { run Anonimizador(id); id++ }
       :: id == NumAnonimizadores -> id=0; break
       od;

       /* Lancar um numero variavel de Totalizadores */
```

```
        do
        :: id < NumTotalizadores -> atomic { run Totalizador(id); id++ }
        :: id == NumTotalizadores -> id=0; break
        od;

        run Temporizador();
}
```

C.1. Propriedades SPIN – intruso

4.4.2 Democracia

```
#define p ((gVotoApurado[0].IdVoto == 10) | (gVotoApurado[1].IdVoto == 10) |
(gVotoApurado[2].IdVoto == 10)) && ((gVotoApurado[0].IdVoto == 11) |
(gVotoApurado[1].IdVoto == 11) | (gVotoApurado[2].IdVoto == 11)) &&
((gVotoApurado[0].IdVoto == 12) | (gVotoApurado[1].IdVoto == 12) |
(gVotoApurado[2].IdVoto == 12))
#define q (indexR == selVoto && eleitoresContados == 3)

        /*
         * Formula As Typed: [] (q -> p)
         * The Never Claim Below Corresponds
         * To The Negated Formula !([] (q -> p) )
         * (formalizing violations of the original)
         */

never {    /* !([] (q -> p) ) */
T0_init:
        if
        :: (! ((p)) && (q)) -> goto accept_all
        :: (1) -> goto T0_init
        fi;
accept_all:
        skip
}

#ifdef NOTES
"Só os eleitores elegíveis votam e uma única vez."

#endif
#ifdef RESULT
warning: for p.o. reduction to be valid the never claim must be stutter-invariant
(never claims generated from LTL formulae are stutter-invariant)
depth 0: Claim reached state 5 (line 301)
Depth=     358 States=   1e+006 Transitions= 3.7e+006 Memory=     9.001   t=   21.4 R=
5e+004
Depth=     358 States=   2e+006 Transitions= 7.39e+006 Memory=     9.001  t=   42.5 R=
5e+004
Depth=     358 States=   3e+006 Transitions= 1.11e+007 Memory=     9.001  t=   63.9 R=
5e+004
Depth=     358 States=   4e+006 Transitions= 1.48e+007 Memory=     9.001  t=   86.8 R=
5e+004
Depth=     358 States=   5e+006 Transitions= 1.85e+007 Memory=     9.001  t=    110 R=
5e+004
Depth=     358 States=   6e+006 Transitions= 2.22e+007 Memory=     9.001  t=    135 R=
4e+004
Depth=     358 States=   7e+006 Transitions= 2.59e+007 Memory=     9.001  t=    156 R=
4e+004
Depth=     358 States=   8e+006 Transitions= 2.96e+007 Memory=     9.001  t=    177 R=
5e+004
Depth=     358 States=   9e+006 Transitions= 3.33e+007 Memory=     9.001  t=    198 R=
5e+004
Depth=     358 States=   1e+007 Transitions= 3.7e+007 Memory=     9.001   t=    219 R=
5e+004
Depth=     358 States= 1.1e+007 Transitions= 4.07e+007 Memory=     9.001  t=    240 R=
5e+004
Depth=     358 States= 1.2e+007 Transitions= 4.45e+007 Memory=     9.001  t=    261 R=
5e+004
Depth=     368 States= 1.3e+007 Transitions= 4.83e+007 Memory=     9.001  t=    283 R=
5e+004
Depth=     368 States= 1.4e+007 Transitions= 5.21e+007 Memory=     9.001  t=    304 R=
5e+004
Depth=     368 States= 1.5e+007 Transitions= 5.6e+007 Memory=     9.001   t=    327 R=
5e+004
```

```
Depth=     368 States= 1.6e+007 Transitions= 5.98e+007 Memory=     9.001  t=    349 R=
5e+004
Depth=     368 States= 1.7e+007 Transitions= 6.36e+007 Memory=     9.001  t=    370 R=
5e+004
Depth=     368 States= 1.8e+007 Transitions= 6.75e+007 Memory=     9.001  t=    392 R=
5e+004
Depth=     368 States= 1.9e+007 Transitions= 7.14e+007 Memory=     9.001  t=    413 R=
5e+004
Depth=     368 States=   2e+007 Transitions= 7.52e+007 Memory=     9.001  t=    435 R=
5e+004
Depth=     368 States= 2.1e+007 Transitions= 7.91e+007 Memory=     9.001  t=    457 R=
5e+004
Depth=     368 States= 2.2e+007 Transitions= 8.31e+007 Memory=     9.001  t=    479 R=
5e+004
Depth=     368 States= 2.3e+007 Transitions= 8.76e+007 Memory=     9.001  t=    504 R=
5e+004
Depth=     368 States= 2.4e+007 Transitions= 9.21e+007 Memory=     9.001  t=    528 R=
5e+004
Depth=     368 States= 2.5e+007 Transitions= 9.57e+007 Memory=     9.001  t=    549 R=
5e+004
Depth=     368 States= 2.6e+007 Transitions= 9.91e+007 Memory=     9.001  t=    568 R=
5e+004
Depth=     368 States= 2.7e+007 Transitions= 1.03e+008 Memory=     9.001  t=    588 R=
5e+004
Depth=     368 States= 2.8e+007 Transitions= 1.07e+008 Memory=     9.001  t=    612 R=
5e+004
Depth=     368 States= 2.9e+007 Transitions= 1.11e+008 Memory=     9.001  t=    636 R=
5e+004
Depth=     368 States=   3e+007 Transitions= 1.15e+008 Memory=     9.001  t=    659 R=
5e+004
Depth=     375 States= 3.1e+007 Transitions= 1.19e+008 Memory=     9.001  t=    680 R=
5e+004
Depth=     445 States= 3.2e+007 Transitions= 1.23e+008 Memory=     9.001  t=    702 R=
5e+004
Depth=     873 States= 3.3e+007 Transitions= 1.27e+008 Memory=     9.196  t=    725 R=
5e+004
Depth=     873 States= 3.4e+007 Transitions= 1.31e+008 Memory=     9.196  t=    747 R=
5e+004

(Spin Version 5.1.6 -- 9 May 2008)
       + Partial Order Reduction

Bit statespace search for:
       never claim          +
       assertion violations  + (if within scope of claim)
       acceptance   cycles   + (fairness disabled)
       invalid end states    - (disabled by never claim)

State-vector 600 byte, depth reached 873, errors: 0
 34891797 states, stored
 99837536 states, matched
1.3472933e+008 transitions (= stored+matched)
 11731390 atomic steps

hash factor: 1.92334 (best if > 100.)

bits set per state: 3 (-k3)

Stats on memory usage (in Megabytes):
20497.653      equivalent memory usage for states (stored*(State-vector + overhead))
    8.000      memory used for hash array (-w26)
    0.076      memory used for bit stack
    0.610      memory used for DFS stack (-m20000)
    9.196      total actual memory usage


pan: elapsed time 769 seconds
pan: rate 45351.953 states/second
pan: avg transition delay 5.7104e-006 usec

#endif
```

7 Intruso

```
#define p (gVotoApurado[2].IdVoto == 0)
```

```
#define q (indexR == selVoto)

        /*
         * Formula As Typed: []     (q   -> p)
         * The Never Claim Below Corresponds
         * To The Negated Formula !([]     (q   -> p))
         * (formalizing violations of the original)
         */

never {    /* !([]     (q   -> p)) */
T0_init:
        if
        :: (! ((p)) && (q)) -> goto accept_all
        :: (1) -> goto T0_init
        fi;
accept_all:
        skip
}

#ifdef NOTES
“Apenas um voto por eleitor será contado”

#endif
#ifdef RESULT
warning: for p.o. reduction to be valid the never claim must be stutter-invariant
(never claims generated from LTL formulae are stutter-invariant)
depth 0: Claim reached state 5 (line 301)
Depth=     358 States=   1e+006 Transitions= 3.7e+006 Memory=    9.001   t=   23.6 R=
4e+004
Depth=     358 States=   2e+006 Transitions= 7.39e+006 Memory=    9.001  t=   45.1 R=
4e+004
Depth=     358 States=   3e+006 Transitions= 1.11e+007 Memory=    9.001  t=   65.7 R=
5e+004
Depth=     358 States=   4e+006 Transitions= 1.48e+007 Memory=    9.001  t=     87 R=
5e+004
Depth=     358 States=   5e+006 Transitions= 1.85e+007 Memory=    9.001  t=    108 R=
5e+004
Depth=     358 States=   6e+006 Transitions= 2.22e+007 Memory=    9.001  t=    129 R=
5e+004
Depth=     358 States=   7e+006 Transitions= 2.59e+007 Memory=    9.001  t=    151 R=
5e+004
Depth=     358 States=   8e+006 Transitions= 2.96e+007 Memory=    9.001  t=    171 R=
5e+004
Depth=     358 States=   9e+006 Transitions= 3.32e+007 Memory=    9.001  t=    192 R=
5e+004
Depth=     358 States=   1e+007 Transitions= 3.7e+007 Memory=    9.001   t=    214 R=
5e+004
Depth=     358 States= 1.1e+007 Transitions= 4.07e+007 Memory=    9.001  t=    235 R=
5e+004
Depth=     358 States= 1.2e+007 Transitions= 4.45e+007 Memory=    9.001  t=    256 R=
5e+004
Depth=     368 States= 1.3e+007 Transitions= 4.83e+007 Memory=    9.001  t=    278 R=
5e+004
Depth=     368 States= 1.4e+007 Transitions= 5.21e+007 Memory=    9.001  t=    299 R=
5e+004
Depth=     368 States= 1.5e+007 Transitions= 5.59e+007 Memory=    9.001  t=    321 R=
5e+004
Depth=     368 States= 1.6e+007 Transitions= 5.98e+007 Memory=    9.001  t=    343 R=
5e+004
Depth=     368 States= 1.7e+007 Transitions= 6.36e+007 Memory=    9.001  t=    364 R=
5e+004
Depth=     368 States= 1.8e+007 Transitions= 6.75e+007 Memory=    9.001  t=    386 R=
5e+004
Depth=     368 States= 1.9e+007 Transitions= 7.14e+007 Memory=    9.001  t=    408 R=
5e+004
Depth=     368 States=   2e+007 Transitions= 7.52e+007 Memory=    9.001  t=    430 R=
5e+004
Depth=     368 States= 2.1e+007 Transitions= 7.9e+007 Memory=    9.001   t=    451 R=
5e+004
Depth=     368 States= 2.2e+007 Transitions= 8.31e+007 Memory=    9.001  t=    475 R=
5e+004
Depth=     368 States= 2.3e+007 Transitions= 8.75e+007 Memory=    9.001  t=    499 R=
5e+004
Depth=     368 States= 2.4e+007 Transitions= 9.2e+007 Memory=    9.001   t=    523 R=
5e+004
Depth=     368 States= 2.5e+007 Transitions= 9.58e+007 Memory=    9.001  t=    545 R=
5e+004
```

```
Depth=     368 States= 2.6e+007 Transitions= 9.91e+007 Memory=     9.001  t=    564 R=
5e+004
Depth=     368 States= 2.7e+007 Transitions= 1.02e+008 Memory=     9.001  t=    583 R=
5e+004
Depth=     368 States= 2.8e+007 Transitions= 1.07e+008 Memory=     9.001  t=    606 R=
5e+004
Depth=     398 States= 2.9e+007 Transitions= 1.11e+008 Memory=     9.001  t=    631 R=
5e+004
Depth=     402 States=   3e+007 Transitions= 1.15e+008 Memory=     9.001  t=    652 R=
5e+004
Depth=     433 States= 3.1e+007 Transitions= 1.19e+008 Memory=     9.001  t=    675 R=
5e+004
Depth=     433 States= 3.2e+007 Transitions= 1.23e+008 Memory=     9.001  t=    699 R=
5e+004
Depth=     713 States= 3.3e+007 Transitions= 1.27e+008 Memory=     9.196  t=    725 R=
5e+004
Depth=     745 States= 3.4e+007 Transitions= 1.31e+008 Memory=     9.196  t=    750 R=
5e+004

(Spin Version 5.1.6 -- 9 May 2008)
       + Partial Order Reduction

Bit statespace search for:
       never claim           +
       assertion violations  + (if within scope of claim)
       acceptance   cycles   + (fairness disabled)
       invalid end states    - (disabled by never claim)

State-vector 596 byte, depth reached 745, errors: 0
 34862247 states, stored
 99851098 states, matched
1.3471335e+008 transitions (= stored+matched)
 11728320 atomic steps

hash factor: 1.92497 (best if > 100.)

bits set per state: 3 (-k3)

Stats on memory usage (in Megabytes):
20347.305      equivalent memory usage for states (stored*(State-vector + overhead))
    8.000      memory used for hash array (-w26)
    0.076      memory used for bit stack
    0.610      memory used for DFS stack (-m20000)
    9.196      total actual memory usage


pan: elapsed time 771 seconds
pan: rate 45205.782 states/second
pan: avg transition delay 5.7247e-006 usec

#endif
```