

7

Estudo de Caso: Controle de Tráfego Aéreo

A principal função do controle de tráfego aéreo (CTA) é monitorar e regular o tráfego aéreo de um aeroporto de origem até um aeroporto de destino (Ndovie 1994). Este tipo de sistema é frequentemente caracterizado como de larga escala, com alta complexidade e dinamismo. Tipicamente, sistemas desta natureza precisam controlar e monitorar milhares de aeronaves em um cenário onde as condições do ambiente estão em constante mudança e diante de situações imprevistas. Os dois principais atores de um CTA são o piloto e o controlador. Os controladores precisam lidar com situações muitas vezes complexas em intervalos de tempo bastante rígidos. Além disso, tanto as ações dos pilotos quanto dos controladores precisam estar em conformidade com as regras definidas pelas agências reguladoras. No Brasil, a agência responsável pelo CTA é a Agência Nacional de Aviação Civil (ANAC).

Como consequência deste cenário onde decisões complexas precisam ser tomadas com restrições de tempo bastante rígidas, os pilotos e controladores são claramente um ponto de risco no sistema e cuja falha pode acarretar consequências desastrosas. O grande número de variáveis do domínio de CTA aumenta a probabilidade de falhas. A ocorrência de falhas pode ser desastrosa dada a criticidade do sistema. Torna-se, portanto, necessário a aplicação de técnicas que auxiliem a construção de sistemas de CTA fidedignos.

O principal objetivo deste estudo de caso é ilustrar como a abordagem de governança proposta nesta tese pode ser aplicada em um domínio complexo para melhorar na sua fidedignidade.

7.1. Leis em CTAs

Alguns dos empecilhos na melhoria da fidedignidade dos sistemas de CTA atuais residem na dificuldade de verificar se as aplicações comerciais “*off-the-shelf*” se comportam de acordo com o esperado e na dificuldade em traduzir o conhecimento técnico do domínio de CTA para o projeto do software (Matthews

2002). A abordagem de governança proposta nesta tese auxilia na diminuição destas dificuldades da seguinte forma:

- “...dificuldade de verificar se as aplicações comerciais *off-the-shelf*...” – A estrutura de monitoramento implementada no middleware M-Law assume que os agentes são tratados como caixas-pretas. Ou seja, não é feita nenhuma suposição sobre os detalhes internos de implementação ou arquitetura dos agentes. Assume-se apenas que os agentes se comunicam através de troca de mensagens mediadas pelo M-Law. Desta forma, as aplicações comerciais *off-the-shelf* podem ser vistas como agentes de software e, portanto, teriam o seu comportamento verificado de acordo com a especificação das leis.
- “... dificuldade em traduzir o conhecimento técnico do domínio de CTA para o projeto do software ... ” – Conforme pode ser visto no modelo conceitual e na comparação com as abordagens de governança relacionadas, o XMLaw possui abstrações que alto nível que são usadas para a especificação das leis. Estas abstrações tratam de conceitos que estão bem mais próximos do mundo real para especificação do comportamento esperado. Conceitos como cena, obrigação, proibição, permissão e mensagem permitem que os especialistas no domínio de CTA foquem na especificação das regras enquanto os especialistas em TI (Tecnologia da Informação) se preocupem em construir o sistema em conformidade com as especificações de leis e com os seus requisitos funcionais e não-funcionais.

7.2.Fontes de Problemas em Sistemas de CTA

De forma geral os problemas que ocorrem em um sistema de CTA e que acarretam em acidentes podem ser causados pelas seguintes fontes de problemas:

- Falta de conhecimento técnico – a ignorância de fatores técnicos tais como estruturas, materiais e aerodinâmica foi responsável por um grande número de acidentes (Matthews 2002). Entretanto, conforme o conhecimento científico e tecnológico avança, acidentes com esta causa são cada vez mais raros.

- Software – faltas que não tratadas adequadamente induzem a falhas no sistema. Embora não tenha sido encontrado nenhum artigo sobre percentual de falhas de software no total das causas dos acidentes aéreos, o artigo de Rahman et al. (Rahman, Beznosov et al. 2006) mostra que no período de 12 anos entre 1994 e 2005, 36% das falhas apresentadas em infraestruturas consideradas críticas, dentre elas, transporte aéreo, fornecimento de água e transportes ferroviários, foram causadas por software. O segundo maior percentual foram falhas de hardware com 21%, seguido por falhas humanas com 7%.
- Hardware – em sistemas de CTA exemplos de falhas de hardware podem ser um radar que deixou de funcionar, ou o link da comunicação entre o piloto e o controlador que não conseguiu ser estabelecido.
- Problemas durante o voo – esta categoria de problemas agrupa as situações como terrorismo, passageiros sob efeito de álcool e brigas, dentre outras.
- Falha Humana – com o avanço do conhecimento técnico, falhas nos aviões tem se tornado cada vez mais raras. Isto tem levado a exposição das falhas humanas. Em 2004, nos Estados Unidos, a falha de pilotos foi considerada como a principal causa de 78,6% dos acidentes fatais e 75,5% de todos os acidentes ocorridos na aviação civil (Krey 2006).
- Condições do tempo – geralmente esta categoria é classificada como uma subcategoria de falha humana. Embora os acidentes causados por condições adversas de tempo, como temporais, ocorram com frequência relativamente baixa (4,5% do total de acidentes por falha humana), eles correspondem por 19,7% dos acidentes fatais causados por falha humana (Krey 2006).

Devido à importância das categorias de software e falha humana, optou-se por abordar predominantemente estas duas categorias neste estudo de caso. Para isto, analisou-se o domínio de ATC tanto sob a perspectiva tradicional de governança quanto sob a perspectiva de fidedignidade.

7.3. Engenharia do Sistema

O sistema de controle de tráfego aéreo será utilizado por dois tipos de usuários: controladores e pilotos. A principal função dos controladores é manter uma separação lateral, vertical e longitudinal entre as aeronaves. Além disso, os controladores também devem manter um fluxo contínuo de aeronaves no espaço aéreo de forma segura e manter os pilotos informados sobre eventos relevantes (informações do tempo, por exemplo). Para os pilotos, o sistema de CTA deve fornecer canais de comunicação confiáveis com os controladores e informações que auxiliem na condução da aeronave.

As aeronaves voam no espaço aéreo nas chamadas aerovias. O espaço aéreo é dividido em setores. Cada setor é composto de uma ou mais interseções. As interseções são conectadas através das linhas imaginárias chamadas aerovias. Em geral, existe pelo menos um centro de controle para cada setor, e cada centro de controle é formado por uma equipe de controladores. A Figura 31 ilustra como o espaço aéreo pode ser modelado.

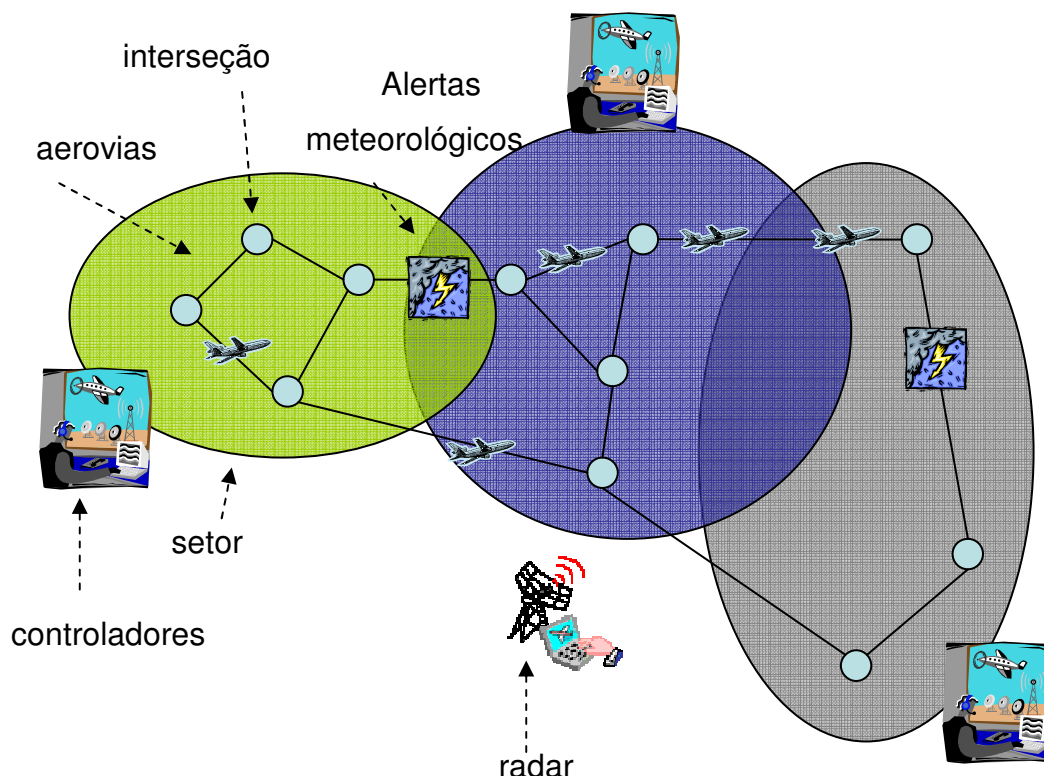


Figura 31 – Modelagem do Espaço Aéreo em um CTA

Tipicamente, existem quatro fases durante o movimento de uma aeronave: (i) controle de pista; (ii) decolagem; (iii) voo e (iv) aterrissagem. Cada uma destas etapas é descrita em mais detalhes a seguir.

- i) Controle de pista – esta é a primeira etapa de voo e, nela, o piloto se comunica com os controladores que informa como a aeronave deve se locomover em pista para efetuar a decolagem. Esta etapa é muito importante para evitar acidentes em solo, como o ocorrido em Tenerife – Espanha em 1977 (Wikipedia 2007). Nesta ocasião, uma grande neblina e a não obediência dos pilotos às ordens do controlador provocaram o maior acidente em número de mortos da história, através do choque entre uma aeronave tentando decolar, com outra aeronave ainda em solo, na mesma pista usada para a decolagem. Nesta etapa, as principais atividades são:
 - a. piloto apresenta plano de voo;
 - b. controlador aprova ou rejeita o plano de voo;
 - c. caso o plano de voo tenha sido aprovado, o piloto informa a intenção de decolar;
 - d. controlador informa as etapas para o posicionamento da aeronave;
- ii) Decolagem – após a fase anterior, a aeronave estará posicionada para a decolagem. Nesta etapa o (a) piloto solicita autorização para decolar e (b) o controlador concede ou rejeita a autorização para decolagem.
- iii) Voo – durante o voo, as aeronaves são monitoradas por radares que disponibilizam informações como posicionamento, altitude e a velocidade. Além disso, a própria aeronave se comunica diretamente com os controladores enviando informações de sensores na aeronave. As aeronaves voam nas aerovias sob o monitoramento do controlador responsável pelo setor da aerovia. A medida que a aeronave segue o seu trajeto, ela pode cruzar vários setores. Entre setores vizinhos existe sempre uma área de interseção. Logo, para mudar de setor, a aeronave entra em um modo de operação chamado *hand-over*. Este modo de operação estabelece protocolos de comportamento diferenciados em relação ao voo normal.

- iv) Aterrissagem – nesta etapa, os controladores precisam se certificar de que os pousos obedecerão aos requisitos de distância mínima e de intervalo mínimo entre pousos consecutivos. Além disso, os pilotos precisam cumprir determinados procedimentos tal como o circuito de tráfego padrão, que estabelece um caminho específico para a aproximação das aeronaves na pista, de maneira que toda aeronave deve respeitar este caminho.

Em cada uma destas etapas existe um conjunto de regulamentações que os pilotos e controladores devem seguir (Ndovie 1994; Rogério 2007). Neste estudo de caso, foi selecionado um subconjunto destas regulamentações classificadas como relevantes sob a ótica de governança e fidedignidade.

7.4.Casos de Uso

Os casos de uso abaixo descrevem as funcionalidades principais de um sistema de CTA (SCTA). Os casos de uso não descrevem os fluxos de eventos, pois o objetivo não é construir um SCTA totalmente funcional, mas identificar as principais interações entre os agentes que compõem o sistema e então identificar os aspectos de governança e de fidedignidade. Desta forma, os casos de uso identificados se assemelham a uma lista de requisitos, que, se refinada, pode gerar o fluxo de eventos e outras características normalmente encontradas em casos de uso.

CASO DE USO	
Identificação	CDU001_Manter_Separação_Lateral_Vertical_Longitudinal
Descrição Sucinta	
<p>O controlador de vôo deve possuir ferramentas visuais para a identificação da distância lateral, vertical e longitudinal entre as aeronaves.</p> <p>De posse da visualização o sistema deve fornecer ferramentas que permitam que o controlador interaja com os pilotos para informar as instruções de posicionamento da aeronave.</p> <p>A interação deve ser baseada em protocolos padrão bem definidos com o intuito de evitar ambigüidades.</p>	
Atores	
1. Controlador de vôo	

CASO DE USO	
Identificação	CDU002_Receber_Informações_Estações_Metereológicas
Descrição Sucinta	
O sistema deve ser capaz de receber dados das diversas estações meteorológicas disponíveis. Estes dados devem poder ser apresentados ao controlador.	
Atores	
1. Controlador de voo 2. Estação Meteorológica (sistema externo)	

CASO DE USO	
Identificação	CDU003_Informar_Condições_Tempo_Ao_Piloto
Descrição Sucinta	
Os dados recebidos de estações meteorológicas devem ser apresentados aos pilotos na medida em que eles precisarem da informação. Um piloto precisa de uma determinada informação meteorológica se a informação se relaciona com o seu plano de voo ou quando o piloto solicita explicitamente a informação.	
Atores	
1. Piloto 2. Estação Meteorológica	

CASO DE USO	
Identificação	CDU004_Receber_Informações_Posicionamento_Velocidade
Descrição Sucinta	
Os radares fornecem informações sobre posicionamento e velocidade das aeronaves dentro do seu perímetro de alcance. O sistema deve ser capaz de receber informações destes radares.	
Atores	
1. Controlador de Voo 2. Radar (sistema externo)	

CASO DE USO	
Identificação	CDU005_Informar_Posicionamento_Velocidade
Descrição Sucinta	
Os controladores necessitam de informação precisa sobre posicionamento e velocidade das aeronaves. Esta informação é originada, principalmente, através dos radares. Desta forma, o sistema deve ser capaz de exibir esta informação tanto aos controladores quanto aos pilotos.	
Atores	
1. Controlador de Vôo 2. Piloto	

CASO DE USO	
Identificação	CDU006_Receber_Plano_Vôo
Descrição Sucinta	
<p>Durante a etapa de <i>controle de pista</i> descrita na Seção 7.3, o piloto envia o plano de vôo para que o controlador aprove. A aprovação do plano de vôo precisa respeitar a regra da autonomia mínima. Autonomia é o tempo total que um aeronave é capaz de voar, em velocidade de cruzeiro, baseado na quantidade de combustível que ela possui. Para a realização de um vôo, a autonomia mínima será:</p> <p>Da decolagem ao destino mais o tempo entre o destino e a alternativa, mais 45 minutos de reserva. Ou seja:</p> <p>A -> B -> C + 45 min.</p> <p>(ORI) (DEST) (ALT) (reserva)</p> <p>Desta forma, o sistema deve ser capaz de receber os planos de vôo dos pilotos e disponibilizá-los para que os controladores os aprovem ou rejeitem.</p>	
Atores	
1. Piloto 2. Controlador de vôo	

CASO DE USO	
Identificação	CDU007_Permitir_Aprovacao_Planos_Voo
Descrição Sucinta	
O sistema deve fornecer uma interface que permita que o controlador analise o plano de vôo e emita a aprovação ou rejeição do plano submetido.	
Atores	
1. Controlador de vôo	

CASO DE USO	
Identificação	CDU008_Garantir_Segurança_Aterrissagem
Descrição Sucinta	
Existem várias regras de segurança que precisam ser seguidas para melhorar o nível de segurança nas aterrissagens. O sistema deve monitorá-las de forma a verificar se elas estão sendo seguidas. Tanto os controladores quanto os pilotos devem ser informados em caso de não cumprimento das regras.	
Atores	
1. Piloto 2. Controlador de vôo	

CASO DE USO	
Identificação	CDU009_Informar_Proativamente_Mudança_Controlador
Descrição Sucinta	
Ao mudar de setor, as aeronaves também passam a ser controladas por controladores diferentes. Ou seja, existe uma troca de controladores. O sistema deve proativamente informar aos pilotos e controladores envolvidos a respeito da nova configuração. Os pilotos são informados de quem é o novo controlador, o controlador anterior passa a não controlar mais a aeronave e o novo controlador recebe a informação da aeronave que estará sob o seu controle.	
Atores	
1. Piloto 2. Controlador de vôo	

CASO DE USO	
Identificação	CDU010_Monitorar_Ações_Piloto
Descrição Sucinta	
<p>O sistema deve ser capaz de monitorar e armazenar todas as ações do piloto em relação a pilotagem.</p> <p>Esta informação pode ser utilizada para auditoria ou mesmo para identificação de falhas em tempo de execução.</p>	
Atores	
1. Piloto	

CASO DE USO	
Identificação	CDU011_Informar_Pilotos_Ações
Descrição Sucinta	
<p>De posse das informações armazenadas com o monitoramento, o sistema eventualmente pode sugerir ao piloto quais ações são mais apropriadas dado que ele executou alguma ação fora do procedimento.</p> <p>Desta forma, o sistema além de exibir alertas, exibe também sugestões de procedimentos a serem adotados.</p>	
Atores	
1. Piloto	

CASO DE USO	
Identificação	CDU012_Prover_Comunicação_Não_Ambígua
Descrição Sucinta	
<p>Um dos grandes riscos de problemas em SCTA é o problema da comunicação. Dificuldade de entendimento entre pilotos e controladores que não possuem o idioma inglês como primeira língua resulta na adoção de procedimentos errados. Além disso, ambigüidades também são geradas devido a não utilização de comunicações padrão.</p> <p>Desta forma, o sistema deve prover o maior número possível de protocolos de comunicação bem definidos que estabeleçam as formas de interação entre controladores e pilotos.</p>	

Atores
1. Controlador de voo
2. Piloto

CASO DE USO	
Identificação	CDU013_Garantir_Segurança_Decolagem
Descrição Sucinta	
<p>É na etapa de decolagem que ocorrem aproximadamente 15,7% dos acidentes com causas não mecânicas (Krey 2006). Existe um conjunto de regras de segurança que devem ser seguidas por pilotos e controladores.</p> <p>O sistema deve verificar se estas regras estão sendo cumpridas e intervir em caso de não cumprimento.</p>	
Atores	
1. Piloto	
2. Controlador de voo	

CASO DE USO	
Identificação	CDU014_Garantir_Segurança_Voo
Descrição Sucinta	
<p>Aproximadamente 9,7% dos acidentes aéreos com causas não mecânicas ocorrem durante o voo (Krey 2006). Porém, os acidentes durante o voo são responsáveis por 22,8% do total de acidentes fatais.</p> <p>Esta estatística indica que garantir a segurança durante esta fase é crítico para diminuir o número de acidentes fatais. Portanto, o sistema deve continuamente monitorar as ações dos pilotos, as informações da aeronave e as instruções dos controladores e intervir quando alguma situação não ocorrer conforme o especificado.</p>	
Atores	
1. Controlador de voo	
2. Piloto	

CASO DE USO	
Identificação	CDU015_Exibir_Interface_Gráfica
Descrição Sucinta	
O sistema deve se comunicar com o controlador e com o piloto através de interfaces gráficas.	
Atores	
1. Controlador de voo 2. Piloto	

7.5.Arquitetura

Um SCTA é composto de vários subsistemas que precisam cooperar para atingir o objetivo do sistema. Cada subsistema possui um objetivo específico para atingir. Por exemplo, o subsistema TACS possui como objetivo evitar colisões entre as aeronaves em voo. Além disso, os subsistemas também possuem autonomia de decisão, muito embora esta autonomia possa ser na maioria dos casos supervisionada por atores humanos que podem interferir no processo. Além disso, muitos destes subsistemas estão geograficamente distribuídos e precisam se comunicar via protocolos de rede. Neste trabalho, optou-se por representar cada um destes subsistemas por um agente de software conforme pode ser visto na arquitetura apresentada Figura 32. Os elementos desta arquitetura são descritos a seguir:

- Estação Meteorológica – fornece informações sobre as condições do tempo atuais e futuras.
- Radar – monitora o posicionamento e velocidade das aeronaves no espaço aéreo.
- Piloto – um agente de software que representa o piloto humano.
- Controlador – agente de software que representa o controlador humano.
- ATC GUI – Interface gráfica que permite com que o controlador tenha acesso a todas as informações para basear a tomada de decisões.
- ATC Façade – Agente que atua como um “broker” da comunicação entre os sistemas descritos acima e os agentes que fazem parte da organização de agentes composta pelos agentes descritos a seguir. De

fato, este agente é uma simplificação do sistema para efeitos de prototipação.

- TCAS – agente que implementa um sistema de alertas de possíveis rotas de colisão entre aeronaves.
- CKPT – agente responsável pelas informações do cockpit da aeronave.
- ADMG – gerencia as etapas de decolagem e pouso.
- MSAW – verifica constantemente violações de segurança relacionadas à altitude permitida.
- SYSC – auxilia controladores e pilotos na etapa de “hand-over”, ou seja, no momento em que uma aeronave muda de setor.

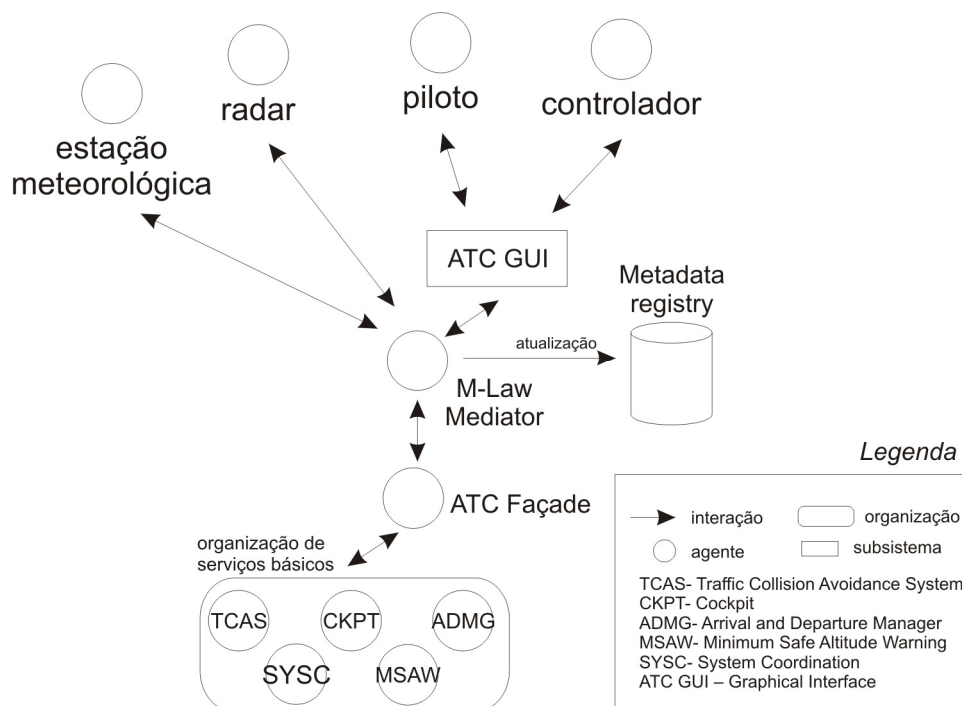


Figura 32 – Arquitetura do Estudo de Caso

Nas seções 7.6 e 7.7, o estudo de caso é analisado sob as óticas de governança e de fidedignidade. Utiliza-se a abordagem desta tese para representar as leis sob estes dois pontos de vista.

7.6. Análise da Governança

Para a realização da análise de forma sistemática, o problema será modelado através dos passos descritos na Tabela 12.

Passo 1: identificar as cenas de interação
Passo 2: identificar os protocolos de interação de cada cena
Passo 3: identificação dos outros elementos de leis

Tabela 12 – Guia de Análise de Governança

7.6.1. Passo 1

Conforme identificado na Seção 7.3, existem 4 fases principais em um voo: controle de pista, decolagem, voo e aterrissagem. Cada uma destas fases possui protocolos de comunicação e regras bem delimitadas. Estas fases são modeladas como cenas.

7.6.2. Passo 2 e 3

A identificação dos protocolos é realizada a partir da análise das interações entre os agentes em cada cena.

Cena 1: controle de pista. Nesta etapa, o piloto apresenta o plano de voo e o controlador aprova ou rejeita o plano de voo. Caso o plano de voo tenha sido aprovado, o piloto informa a intenção de decolar e por fim, o controlador informa as etapas para o posicionamento da aeronave. A Figura 33 mostra o protocolo de interação destas atividades. A representação deste protocolo em XMLaw é ilustrada na listagem de Código 38.

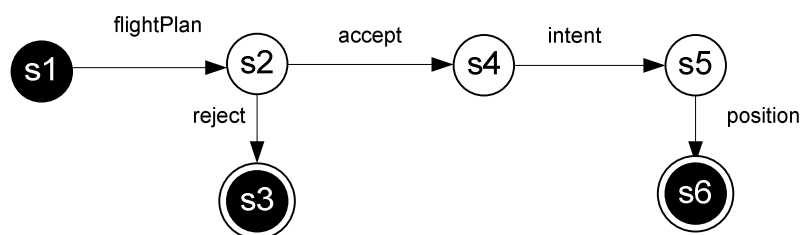


Figura 33 – Protocolo de Interação da Cena Controle de Pista

```

groundControl{ //nome da cena

// Mensagens
flightPlan{pilot, controller,propose(flightPlan($content))}
reject{controller, pilot, reject-proposal}
accept{controller, pilot, accept-proposal}
intent{pilot, controller, request(go-position)}
position{controller, pilot, inform(pos, $instructions)}

// Estados especiais
s1{initial}
s3{failure}
s6{success}

// Transições
t1{s1->s2, flightPlan}
t2{s2->s3, reject}
t3{s2->s4, accept}
t4{s4->s5, intent}
t5{s5->s6, position}
}

```

Código 38 – Protocolo de interação em XMLaw da cena *groundControl*

Cena 2: decolagem – o protocolo de interação desta etapa é bastante simples. O piloto solicita autorização para decolar e o controlador concede ou rejeita a autorização para decolagem.

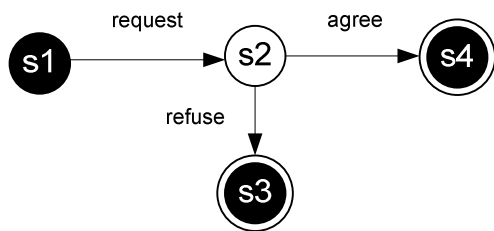


Figura 34 - Protocolo de Interação da Cena Decolagem

```

take-off{ //nome da cena

// Mensagens
request{pilot, controller, request(take-off)}
refuse{controller, pilot, refuse }
agree{controller, pilot, agree }

// Estados especiais
s1{initial}
s3{failure}
s4{success}

// Transições
t1{s1->s2, request}
t2{s2->s3, refuse}
t3{s2->s4, agree}
}

```

Código 39 – Protocolo de interação em XMLaw da cena *take-off*

Cena 3: vôo – nesta etapa, existem 2 tipos de interações principais: monitoramento da aeronave e o processo de mudança de setor (*hand-over*). A Figura 35 mostra o protocolo de interação. A partir do estado inicial, duas mensagens podem ser enviadas: *progressR* e *progressA*. A mensagem *progressR* representa uma mensagem enviada do radar para o controlador informando o posicionamento e a velocidade da aeronave. Algumas aeronaves são equipadas com dispositivos que também permitem que a própria aeronave envie uma mensagem para o controlador informando o posicionamento e a velocidade da aeronave. Esta mensagem é representada pela mensagem *progressA*. No estado *s2*, existem 3 mensagens possíveis: *switch*, *progressR* e *progressA*. Elas representam o vôo da aeronave, onde constantemente existem mensagens de monitoramento e no momento que a aeronave for realizar o *hand-over* ocorrerá a mensagem *switch*. Finalmente, a cena de vôo se encerra quando o piloto enviar uma mensagem de intenção de pouso (*landing*) para o controlador. O Código 40 mostra a representação deste protocolo em XMLaw.

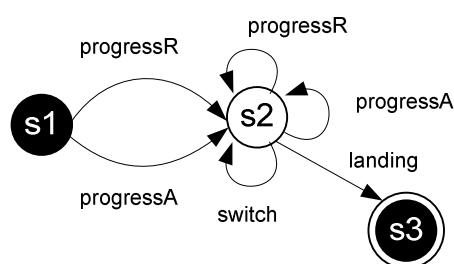


Figura 35 - Protocolo de Interação da Cena Vôo

```

flight{ //nome da cena

// Mensagens
  progressR{radar, controller, inform(strip,
$flightProgressStrip)}
  progressA{pilot, controller, inform(strip,
$flightProgressStrip)}
  switch{controller, pilot, inform(switch, $newController)}
  landing{pilot, controller, inform(landIntention)}

// Estados especiais
  s1{initial}
  s3{success}

// Transições
  t1{s1->s2, progressR}
  t2{s1->s2, progressA}
  t3{s2->s2, progressR}
  t4{s2->s2, progressA}

```



```

t5{s2->s2, switch}
t6{s2->s3, landing}
}

```

Código 40 – Protocolo de Interação em XMLaw da cena *flight*

Cena 4: aterrissagem – o protocolo de interação desta etapa consiste no piloto solicitar permissão para pouso e aguardar a permissão do controlador. A permissão pode ser concedida imediatamente ou após algum tempo de espera.

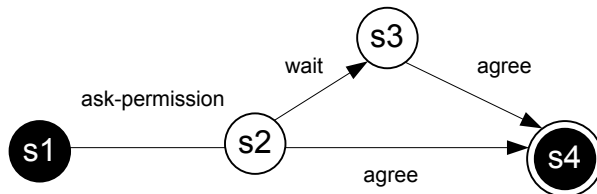


Figura 36 - Protocolo de Interação da Cena Aterrissagem

```

landing{
  ask-permission{pilot, controller,
request(landingPermission)}
  wait{controller, pilot, inform(wait)}
  agree{controller, pilot, agree(permission)}

  s1{initial}
  s4{success}

  t1{s1->s2, ask-permission}
  t2{s2->s3, wait}
  t3{s3->s4, agree}
  t4{s2->s4, agree}
}

```

Durante a identificação dos protocolos acima, foi possível perceber quais eram os agentes que interagem em cada cena. São eles:

- Cena *groundControl*: Pilot e Controller
- Cena *take-off*: Pilot e Controller
- Cena *flight*: Pilot, Radar e Controller
- Cena *landing*: Pilot e Controller

7.7. Análise da Fidedignidade

A análise da fidedignidade envolve a análise das ameaças, a identificação de estratégias de mitigação destas ameaças, a introdução de um agente detector de falhas e a especificação de leis que implementam as estratégias de mitigação.

7.7.1. Análise das ameaças

Nesta etapa, procura-se identificar, priorizar e relacionar as ameaças que, se efetivadas, poderiam prejudicar a o objetivo do sistema. A cada ameaça atribui-se um identificador único, uma lista de identificadores de casos de uso, uma descrição e a consequência da sua efetivação. O identificador é utilizado para fazer um rastreamento da ameaça no restante da documentação. Os identificadores de casos de uso são utilizados para identificar os casos de uso que a ameaça pode afetar. A descrição fornece um breve contexto para a ameaça e, por fim, a consequência identifica o que acontece em uma situação que a ameaça se concretiza. O objetivo deste campo é identificar a criticalidade que é definida em uma escala de 1 a 5.

Embora, a Tabela 13 utilizada para a elicitación das ameaças utilizadas neste trabalho seja bastante simples, ela atende as necessidades deste estudo de caso. Caso o problema a ser resolvido demande por uma análise de ameaças mais refinada, é possível utilizar-se uma das várias abordagens existentes na literatura para elicitación, análise e gerenciamento de riscos (Boehm 1981; Charette 1989; Karolak 1996).

Id	Ids CDUs	Descrição	Consequências
CDL01	CDU01	Aeronave se aproxima demais de outra aeronave	4 Colisão
CDL02	CDU14	Aeronave voa em uma altitude diferente da planejada	4 Colisão
CDL03	CDU13	Piloto não ativa o "flap" no momento da decolagem	4 Queda
CDL04	CDU12	Ocorre falha no entendimento da comunicação entre o piloto e a torre de controle	3 Colisão, informação inconsistente
CDL05	CDU13	O piloto não ativa o sistema descongelador	3 Congelamento da turbina

CDL06	CDU14	Piloto desativa inapropriadamente um instrumento	2 Informação inconsistente, queda, dificuldade de auditoria
CDL07	CDU02	Falha na comunicação com a estação meteorológica	2 Turbulência, queda
CDL08	CDU04	Falha na comunicação com os radares	5 Colisão
CDL09	CDU07	Controlador aprova plano de vôo fora das especificações de segurança	3 Acidente
CDL10	CDU07	Controlador concede permissão de vôo que acarretará em uma distância de autonomia mínima menor que o especificado. A autonomia mínima é definida como sendo o tempo total que um aeronave é capaz de voar, em velocidade de cruzeiro, baseado na quantidade de combustível que ela possui. Para a realização de um vôo, a autonomia mínima será: Da decolagem ao destino mais o tempo entre o destino e a alternativa, mais 45 minutos de reserva. Ou seja: A -> B -> C + 45 min. (DEP) (ARR) (ALT) (reserva)	2 Falta de combustível
CDL11	CDU09	Controlador perde o canal de comunicação	5 Acidentes
CDL12	CDU08	Para que uma aeronave possa operar sem restrições em uma determinada	1 Prejuízos na pista

		pista, o ACN da aeronave deverá ser menor ou igual que o PCN da pista (ACN \leq PCN). A ameaça consiste em a aeronave solicitar pouso com ACN > PCN	
CDL13	CDU08	<p>O piloto não executa a operação padrão de circuito de tráfego ao pousar. A altura padrão para as aeronaves realizarem o circuito de tráfego é:</p> <ul style="list-style-type: none"> • 1500ft (pés) para aeronaves a jato; • 1000ft (pés) para aeronaves a hélice. <p>Todas as curvas são feitas para a esquerda</p>	2 Colisão
CDL14	CDU14	Exceto em procedimentos de pouso e decolagem, as aeronaves não poderão voar sobre cidades, povoados, lugares habitados ou grupo de pessoas ao ar livre a uma altura inferior a 1000 pés (300M) acima do obstáculo mais alto existente num raio de 600M em torno da acft; A ameaça consiste no piloto descumprir estas restrições.	3 Colisão em prédios, paraquedistas, etc.
CDL15	CDU14	Exceto em procedimentos de pouso e decolagem, as aeronaves não poderão voar em lugares desabitados em altura inferior a 500 pés (150M) sobre o solo ou água. A ameaça consiste no piloto voar a uma altura inferior a 500 pés.	3 Colisão com formações geológicas

Tabela 13 – Elicitação das Ameaças

Ao se concretizarem, as ameaças afetam os atributos de fidedignidade do sistema. Na Seção 7.7.2, mostra-se como o XMLaw e o M-Law são utilizados para especificar e implementar estratégias de mitigação destas ameaças.

7.7.2.Mitigação das Ameaças

Ameaça	Mitigação
CDL01	<ul style="list-style-type: none"> - solicitar plano de ação ao agente TCAS (<i>traffic collision avoidance system</i>); - informar plano de ação ao piloto; - informar plano de ação ao controlador.
CDL02	<ul style="list-style-type: none"> - solicitar plano de ação ao agente MSAW (<i>minimum safe altitude warning</i>); - informar plano de ação ao piloto; - informar plano de ação ao controlador.
CDL03	- emitir alerta ao piloto.
CDL04	- utilizar protocolos de comunicação bem definidos e mensagens padronizadas.
CDL05	- emitir alerta ao piloto.
CDL06	- Alertar o piloto das conseqüências de se permanecer com o instrumento desligado.
CDL07	- Solicitar ao agente <i>ATC Façade</i> outra estação meteorológica que possa prover os dados.
CDL08	- Alertar administradores do sistema através de email.
CDL09	<ul style="list-style-type: none"> - Solicitar ao ADMG (<i>arrival and departure manager</i>) que alerte o controlador, informe onde está o erro e explique as conseqüências. - Solicitar confirmação de ação.
CDL10	<ul style="list-style-type: none"> - Solicitar ao ADMG (<i>arrival and departure manager</i>) que alerte o controlador, informe onde está o erro e explique as conseqüências. - Solicitar confirmação de ação
CDL11	- Solicitar ao ATC Façade que avise aos pilotos quais são os controladores disponíveis.
CDL12	- Solicitar ao ADMG (<i>arrival and departure manager</i>) que alerte o controlador, informe onde está o erro e explique as conseqüências.

	- Solicitar confirmação de ação.
CDL13	- Alertar o controlador e o piloto sobre o descumprimento do circuito de tráfego padrão.
CDL14	- solicitar plano de ação ao agente MSAW (<i>minimum safe altitude warning</i>); - informar plano de ação ao piloto; - informar plano de ação ao controlador
CDL15	- solicitar plano de ação ao agente MSAW (<i>minimum safe altitude warning</i>); - informar plano de ação ao piloto; - informar plano de ação ao controlador

Tabela 14 – Estratégias de Mitigação das Ameaças

Uma vez identificadas as estratégias de mitigação é preciso alterar a especificação das leis definidas na Seção 7.6 para contemplar a implementação destas estratégias. Para que as estratégias de mitigação possam ser implementadas é necessário que se possa identificar a ocorrência da ameaça durante a execução do sistema. A abordagem de governança proposta nesta tese faz isso através da especificação da lei para detectar a ameaça e através de um agente especial intitulado “detector de falhas”. Este agente identifica os agentes que estão indisponíveis no sistema. Esta indisponibilidade pode ter sido provocada pelo excesso de processamento, pelo não funcionamento do agente ou até mesmo por problemas no link de comunicação. O objetivo deste agente é reduzir o número de clocks com a finalidade de identificar agentes indisponíveis. Ao detectar um agente indisponível, o *detector de falhas* emitirá um evento no XMLaw *agent_unavailable* que pode ser utilizado para ativar algum outro elemento do XMLaw.

7.7.3. Agente Detector de Falhas

Neste trabalho, optou-se por implementar o agente detector de falhas através da utilização de um agente único que implementa um algoritmo de *heartbeat*. Este algoritmo consiste em enviar mensagens de controle periodicamente para os agentes do sistema esperando que os agentes respondam a esta mensagem. Devido

ao grande número de variáveis e a simplicidade da estratégia de implementação do *heartbeat*, podem ocorrer falsos negativos, ou seja, a demora da resposta pode ser interpretada como indisponibilidade do agente. Sendo assim, para minimizar este problema, o agente tentará pelo menos 3 vezes antes de indicar que o agente está indisponível. Se ainda assim, esta estratégia não for suficiente para um determinado domínio de aplicação, então o agente detector poderia ser substituído por implementações mais sofisticadas, como por exemplo o *Globus Heartbeat Monitor* (Stelling, DeMatteis et al. 1999). Na Figura 37, mostra-se a arquitetura do estudo de caso modificada para a inclusão do agente detector de falhas.

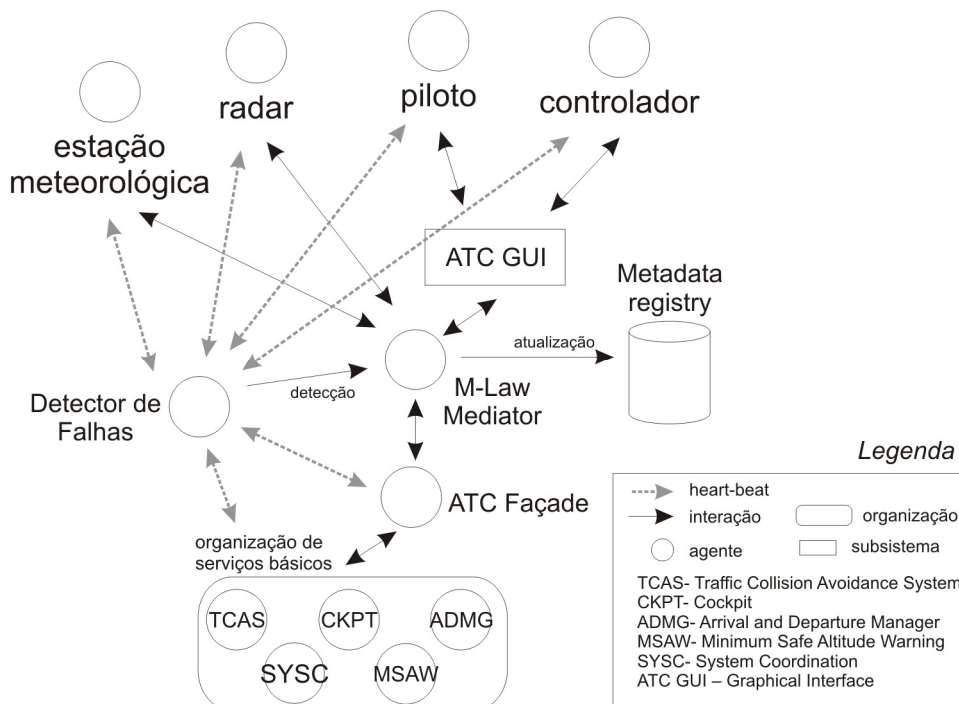


Figura 37 – Arquitetura do Estudo de Caso com a Inclusão do Agente Detector de Falhas

7.7.4.Modificação das Leis para Implementar as Ameaças

Nesta seção, as estratégias de mitigação de cada ameaça serão especificadas utilizando o XMLaw. Para isto, apresenta-se o identificador da ameaça, o contexto de lei no qual a ameaça afeta, a estratégia de detecção da ameaça e finalmente, quando for necessário, o exemplo de código XMLaw da implementação.

Ameaça: CDL01
Contexto: Cena de Voo
Estratégia de detecção: para detectar se as aeronaves estão próximas, é preciso

saber os seus posicionamentos. O posicionamento é informado pelas mensagens *progressR* ou *progressA*. Baseado nestas mensagens, a lei deve executar um algoritmo de cálculo de distância. Se este algoritmo indicar uma distância menor do que o permitido, então o sistema de prevenção de colisão deve ser alertado. O algoritmo de cálculo e verificação de distância é executado por uma *constraint*. Esta *constraint* retorna *true* quando a distância é menor que o determinado. Quando uma *constraint* retorna *true*, o evento *constraint_not_satisfied* é gerado. Este evento deve ativar uma *action* que avisa o TCAS. Além disso, de acordo com a estratégia de mitigação especificada na Tabela 14, o piloto e o controlador também devem ser informados. O Código 41 apresentado a seguir mostra a cena de voo modificada para a inclusão da implementação desta estratégia de mitigação, e no Código 42 apresenta-se a implementação da *constraint checkDistance*.

```

flight{    //nome da cena

// Mensagens
  progressR{radar, controller, inform(strip,
$flightProgressStrip)}
  progressA{pilot, controller, inform(strip,
$flightProgressStrip)}
  switch{controller, pilot, inform(switch, $newController)}
  landing{pilot, controller, inform(landIntention)}

// Estados especiais
  s1{initial}
  s3{success}

// Transições
  t1{s1->s2, progressR, [checkDistance]}
  t2{s1->s2, progressA, [checkDistance]}
  t3{s2->s2, progressR, [checkDistance]}
  t4{s2->s2, progressA, [checkDistance]}
  t5{s2->s2, switch}
  t6{s2->s3, landing}

// Constraints
  checkDistance{br.les.CheckDistance}

// Actions
  warnTcas{ ((checkDistance,constraint_not_satisfied)),
br.les.WarnTCAS}
  warnPilot{ ((checkDistance,constraint_not_satisfied)),
br.les.WarnPilot}
  warnController{
((checkDistance,constraint_not_satisfied)),
br.les.WarnController}

```

Código 41 – XMLaw da cena de voo com a estratégia de mitigação da

ameaça CDL01

```

class CheckDistance implements IConstraint{
    private void init(){
        ...
    }
    public boolean constrain(ReadonlyContext ctx){
        String id = ctx.get("flightProgressStrip.airplaneId");
        String x = ctx.get("flightProgressStrip.posX");
        String y = ctx.get("flightProgressStrip.posY");
        String z = ctx.get("flightProgressStrip.posZ");

        saveCurrentPosition(id,x,y,z);

        if ( tooClose(id) ) {
            return true;
        }
    }
}

```

Código 42 – Implementação da Constraint CheckDistance**Ameaça: CDL02****Contexto:** Cena de Vôo

Estratégia de detecção: utilizar as mensagens *progressR* ou *progressA* como informações para uma constraint que verifica a altura segura. Esta *constraint* retorna *true* quando a altura é menor que o determinado. A constraint ativa uma *action* que avisa o MSAW. Além disso, o piloto e o controlador também devem ser informados. O Código 43 apresentado a seguir mostra a cena de vôo modificada para a inclusão da implementação desta estratégia de mitigação. As setas apontando para retângulos tracejados indicam onde foram feitas as modificações. A implementação da *constraint* foi omitida, pois é bastante similar a *constraint* exibida no Código 42.

```

flight{    //nome da cena

// Mensagens
    progressR{radar, controller, inform(strip,
$flightProgressStrip)}
    progressA{pilot, controller, inform(strip,
$flightProgressStrip)}
    switch{controller, pilot, inform(switch, $newController)}
    landing{pilot, controller, inform(landIntention)}

// Estados especiais

```

```

s1{initial}
s3{success}

// Transições
t1{s1->s2, progressR, [checkDistance, checkAltitude]}
t2{s1->s2, progressA, [checkDistance, checkAltitude]}
t3{s2->s2, progressR, [checkDistance, checkAltitude]}
t4{s2->s2, progressA, [checkDistance, checkAltitude]}
t5{s2->s2, switch}
t6{s2->s3, landing}

// Constraints
checkDistance{br.les.CheckDistance}
checkAltitude{br.les.CheckAltitude}

// Actions
warnTcas{ ((checkDistance,constraint_not_satisfied)),
br.les.WarnTCAS}

warnMsaw{ ( (checkAltitude,constraint_not_satisfied) ),
br.les.WarnMSAW}

warnPilot{
(
    (checkDistance,constraint_not_satisfied),
    (checkAltitude,constraint_not_satisfied)
), br.les.WarnPilot}

warnController{
(
    (checkDistance,constraint_not_satisfied),
    (checkAltitude,constraint_not_satisfied)
), br.les.WarnController}
}

```

Código 43 – XMLaw da Cena de Voo com a Estratégia de Mitigação da Ameaça CDL02

Ameaça: CDL03

Contexto: Cena de decolagem

Estratégia de detecção: a informação de que o piloto ativou ou não o *flap* está disponível somente na própria aeronave. Desta forma, para monitorar esta informação, acrescenta-se uma mensagem de status da aeronave enquanto o piloto estiver decolando. Esta mensagem deve ser enviada continuamente até que o piloto inicie a etapa de voo. Desta forma, uma *action* pode ser ativada a cada novo reporte de status. Esta *action* verifica se o *flap* foi ativado e caso não tenha sido, emite um alerta para o piloto.

```

take-off{ //nome da cena

// Mensagens
request{pilot, controller, request(take-off)}
refuse{controller, pilot, refuse }
agree{controller, pilot, agree }
status{pilot, controller, inform($status)}
flight{pilot, controller, inform(flightPhase)}

// Estados especiais
s1{initial}
s3{failure}
s5{success}

// Transições
t1{s1->s2, request}
t2{s2->s3, refuse}
t3{s2->s4, agree}
t4{s4->s4, status}
t5{s4->s5, flight}

// Actions
warnPilot{ (t4), br.les.WarnPilot}
}

```

Código 44 – XMLaw da Cena de Decolagem com a Estratégia de Mitigação da Ameaça CDL03

Ameaça: CDL04

Contexto: Todas as cenas

Estratégia de detecção: as falhas de comunicação podem ser consideravelmente reduzidas quando se utilizam protocolos de interação bem definidos. Neste caso, a estratégia é utilizar os protocolos de interação do próprio XMLaw para definir as comunicações que são válidas. Os protocolos das cenas já foram apresentados no decorrer deste capítulo.

Ameaça: CDL05

Contexto: Cena de voo

Estratégia de detecção: para a detecção de que o piloto não ativou o sistema descongelador utilizou-se uma abordagem similar a estratégia utilizada para a detecção da ameaça CDL03. A *action warnDefroster* é ativada a cada novo reporte de status. Esta *action* verifica se o descongelador foi ativado e caso não tenha sido, emite um alerta para o piloto. Nota-se que esta estratégia foi diferente da utilizada para identificar a distância e altitude. Nestas duas, utilizou-se uma *constraint*. Entretanto, nada impede que se tivesse utilizado a estratégia de *action* proposta nessa ameaça.

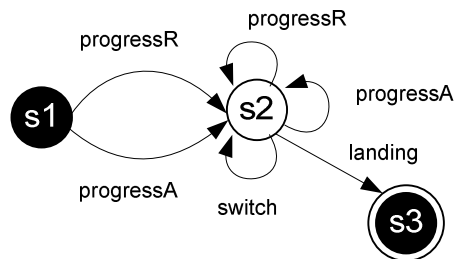


Figura 38 – Protocolo de Interação da cena de voo

```

flight{    //nome da cena

// Mensagens
  progressR{radar, controller, inform(strip,
$flightProgressStrip)}
  progressA{pilot, controller, inform(strip,
$flightProgressStrip)}
  switch{controller, pilot, inform(switch, $newController)}
  landing{pilot, controller, inform(landIntention)}

// Estados especiais
  s1{initial}
  s3{success}

// Transições
  t1{s1->s2, progressR, [checkDistance, checkAltitude]}
  t2{s1->s2, progressA, [checkDistance, checkAltitude]}
  t3{s2->s2, progressR, [checkDistance, checkAltitude]}
  t4{s2->s2, progressA, [checkDistance, checkAltitude]}
  t5{s2->s2, switch}
  t6{s2->s3, landing}

// Constraints
  checkDistance{br.les.CheckDistance}
  checkAltitude{br.les.CheckAltitude}

```

```

// Actions
warnTcas{ ((checkDistance,constraint_not_satisfied)),
br.les.WarnTCAS}

warnMsaw{ ( (checkAltitude,constraint_not_satisfied) ),
br.les.WarnMSAW}

warnPilot{
(
    (checkDistance,constraint_not_satisfied),
    (checkAltitude,constraint_not_satisfied)
), br.les.WarnPilot}

warnController{
(
    (checkDistance,constraint_not_satisfied),
    (checkAltitude,constraint_not_satisfied)
), br.les.WarnController}

warnDefroster{ (t1,t2,t3,t4), br.les.WarnDefroster}
}

```



Código 45 – XMLaw da Cena de Vôo com a Estratégia de Mitigação da Ameaça CDL05

Ameaça: CDL06

Contexto: Cena de vôo

Estratégia de detecção: A *action checkInstruments* é ativada a cada nova comunicação de status. Esta *action* verifica se o algum instrumento foi desativado durante o vôo e caso tenha sido, informa ao piloto as consequências de ter o instrumento desligado.

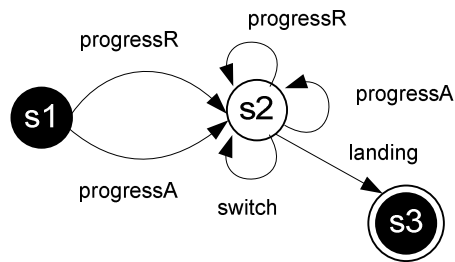


Figura 39 – Protocolo de Interação da Cena de Vôo

```

flight{ //nome da cena

// Mensagens
  progressR{radar, controller, inform(strip,
$flightProgressStrip)}
  progressA{pilot, controller, inform(strip,
$flightProgressStrip)}
  switch{controller, pilot, inform(switch, $newController)}
  landing{pilot, controller, inform(landIntention)}

// Estados especiais
  s1{initial}
  s3{success}

// Transições
  t1{s1->s2, progressR, [checkDistance, checkAltitude]}
  t2{s1->s2, progressA, [checkDistance, checkAltitude]}
  t3{s2->s2, progressR, [checkDistance, checkAltitude]}
  t4{s2->s2, progressA, [checkDistance, checkAltitude]}
  t5{s2->s2, switch}
  t6{s2->s3, landing}

// Constraints
  checkDistance{br.les.CheckDistance}
  checkAltitude{br.les.CheckAltitude}

// Actions
  warnTcas{ ((checkDistance,constraint_not_satisfied)),
br.les.WarnTCAS}

  warnMsaw{ ( (checkAltitude,constraint_not_satisfied) ),
br.les.WarnMSAW}

```


```

warnPilot{
  (
    (checkDistance,constraint_not_satisfied),
    (checkAltitude,constraint_not_satisfied)
  ), br.les.WarnPilot}

warnController{
  (
    (checkDistance,constraint_not_satisfied),
    (checkAltitude,constraint_not_satisfied)
  ), br.les.WarnController}

warnDefroster{ (t1,t2,t3,t4), br.les.WarnDefroster}
checkInstruments{( t1,t2,t3,t4), br.les.CheckInstruments}
}

```



Código 46 – XMLaw da Cena de Voo com a Estratégia de Mitigação da Ameaça CDL06

Ameaça: CDL07

Contexto: Todas as cenas

Estratégia de detecção: a falha de comunicação com a estação meteorológica poderia ser detectada de duas formas: através do uso combinado do protocolo de interação e de um *clock* ou através do agente detector de falhas. Optou-se pelo uso do agente que informará caso ele se depare com 3 tentativas mal-sucedidas de comunicação com a estação meteorológica. Neste caso, o agente gera o evento *agent_unavailable* que é captado por uma *action*, que informa o agente ATC para utilizar outra estação meteorológica. Neste caso, a *action* é declarada em um escopo global. Desta forma, ela é válida para todas as cenas.

```


atc-law{
  // global_actions
  switchStation{(failure_detector, agent_unavailable),
  br.les.SwitchStation}

  //cena
  groundControl {
    ...
  }

  //cena
  take-off{
    ...
  }

  //cena
  flight{

```



```

    ...
}

//cena
landing{
    ...
}

} //end law

```

Código 47 – XMLaw Utilizando o Agente Detector de Falhas

Ameaça: CDL08

Contexto: Todas as cenas

Estratégia de detecção: no caso de falha de comunicação com os radares, a estratégia de mitigação adotada consiste em enviar um email para os administradores do sistema. A falha é detectada através do agente detector de falhas, que através do evento *agent_unavailable* ativa uma *action* de envio de email.

```

atc-law{
  // global actions
  switchStation{(failure_detector, agent_unavailable),
br.les.SwitchStation}

  radarDown{(failure_detector, agent_unavailable),
br.les.RadarDown}

  //cena
  groundControl {
    ...
  }

  //cena
  take-off{
    ...
  }

  //cena
  flight{
    ...
  }

  //cena
  landing{
    ...
  }

} //end law

```


Código 48 – Utilização do Agente Detector de Falhas para Reportar Falhas de Comunicação com os Radares.
--

Ameaça: CDL09

Contexto: Cena de controle de pista
--

<p>Estratégia de detecção: foram utilizados vários elementos do XMLaw de forma combinada (Código 49). Declara-se uma <i>constraint</i> na transição <i>t3</i>. Essa <i>constraint</i> verifica se o plano de voo está de acordo com as regras. Caso não esteja, a <i>constraint</i> gera o evento <i>constraint_not_satisfied</i> e evita que a transição <i>t3</i> dispare. Porém, o evento de <i>constraint_not_satisfied</i> faz com que a transição <i>t6</i> dispare. A transição <i>t6</i> muda o estado do protocolo de <i>s2</i> para <i>s7</i>. Por sua vez, disparo da transição <i>t6</i> faz com que a ação <i>askConfirmation</i> seja ativada. Esta ação envia uma mensagem para o controlador solicitando a confirmação da aprovação do plano de voo, visto que algo fora dos padrões fora encontrado. Neste ponto o protocolo está no estado <i>s7</i> e só existem duas transições de saída. A transição <i>t8</i> dispara quando o controlador rejeita o plano e a transição <i>t7</i> dispara quando o controlador confirma a aprovação do plano de voo.</p>
--

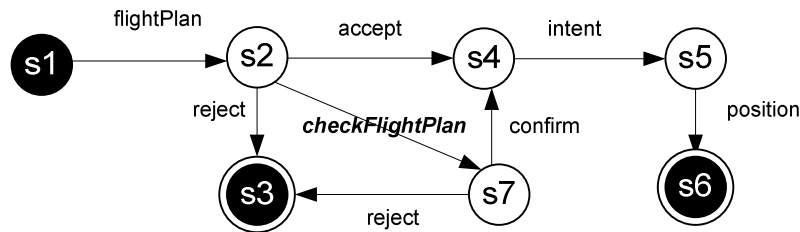


Figura 40 – Protocolo de Interação da Cena de Controle de Pista

```

groundControl{ //nome da cena

// Mensagens
  flightPlan{pilot,
controller,propose(flightPlan($content))}
  reject{controller, pilot, reject-proposal}
  accept{controller, pilot, accept-proposal}
  intent{pilot, controller, request(go-position)}
  position{controller, pilot, inform(pos, $instructions)}
  confirm{controller, pilot, confirm(accept-proposal)}

// Estados especiais
  s1{initial}
  s3{failure}
  s6{success}

// Transições
  t1{s1->s2, flightPlan}
  t2{s2->s3, reject}
  t3{s2->s4, accept, [checkFlightPlan]}
  t4{s4->s5, intent}
  t5{s5->s6, position}
  t6{s2->s7, (checkFlightPlan, constraint_not_satisfied)}
  t7{s7->s4, confirm}
  t8{s7->s3, reject}

  // constraints
  checkFlightPlan{br.les.CheckFlightPlan}

  // action
  askConfirmation{t6, br.les. AskConfirmation}

}
  
```

Código 49 – XMLaw de Mitigação da Ameaça CDL09.

Ameaça: CDL10

Contexto: Cena de controle de pista

Estratégia de detecção: a estratégia de mitigação da ameaça CDL09 inseriu uma constraint chamada *checkflightPlan* que verificava quebra nos padrões de

segurança no plano de vôo. Para que a autonomia mínima também seja verificada, basta modificar a implementação desta *constraint* e verificar se o plano de vôo contempla a autonomia mínima. Desta forma, se não contemplar, o protocolo segue da forma como foi especificado, solicitando uma confirmação e assim por diante.

Ameaça: CDL11

Contexto: Todas as cenas

Estratégia de detecção: a ameaça consiste em o controlador perder o canal de comunicação e a estratégia de mitigação é solicitar ao agente ATC Façade que informe a todos os pilotos quais são os controladores disponíveis. Assim, o piloto pode se comunicar com outro controlador. A detecção de que um controlador perdeu o canal de comunicação é feita pelo agente detector de falhas. Este agente gera o evento *agent_unavailable* passando como parâmetro o controlador que não está conseguindo se comunicar. Quando este evento é detectado, ativa-se uma action que envia uma mensagem para o agente ATC Façade informado da indisponibilidade do controlador. A action diferencia entre os eventos de *agent_unavailable* através dos parâmetros. Isso permite que ela não mande mensagens ao ATC quando o que ficou indisponível foi um radar, por exemplo. Ao receber a mensagem, o ATC Façade envia uma mensagem a todos os pilotos informando os controladores disponíveis.

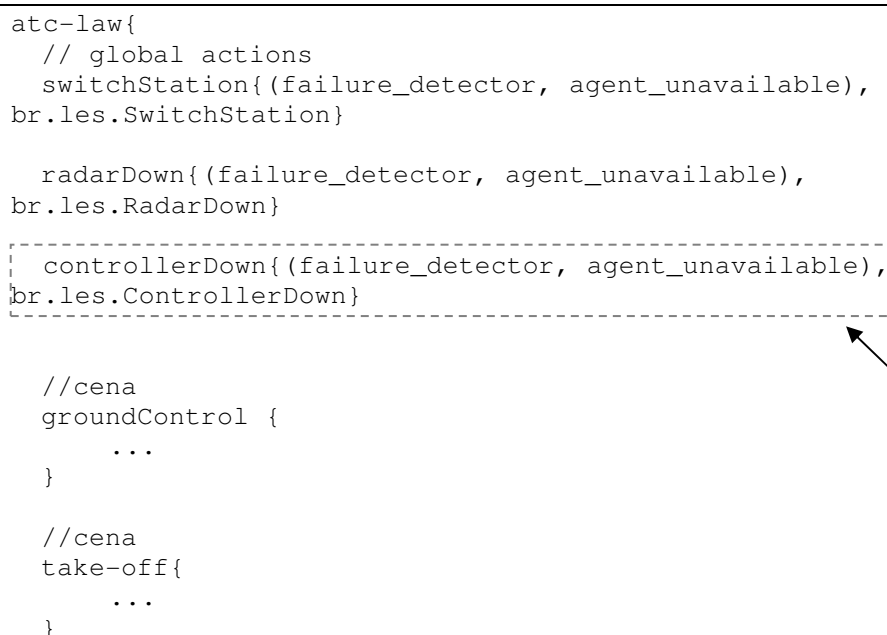
```
atc-law{
  // global actions
  switchStation{(failure_detector, agent_unavailable),
br.les.SwitchStation}

  radarDown{(failure_detector, agent_unavailable),
br.les.RadarDown}

  controllerDown{(failure_detector, agent_unavailable),
br.les.ControllerDown}

  //cena
  groundControl {
    ...
  }

  //cena
  take-off{
    ...
  }
}
```



```

//cena
flight{
    ...
}

//cena
landing{
    ...
}

} //end law

```

Código 50 – XMLaw de Mitigação da Ameaça CDL11

Ameaça: CDL12

Contexto: Cena aterrissagem

Estratégia de detecção: a implementação desta estratégia ocorre de forma análoga a utilizada para a ameaça CDL09. Declara-se uma *constraint* nas transições *t3* e *t4*. Essa *constraint* verifica se o ACN da aeronave é menor ou igual ao PCN da pista. Caso não seja, a *constraint* gera o evento *constraint_not_satisfied* e evita que a transição *t3* ou *t4* dispare. Porém, o evento de *constraint_not_satisfied* faz com que as transições *t5* ou *t6* disparem. A transição *t5* muda o estado do protocolo de *s2* para *s5*. A transição *t6* muda o estado do protocolo de *s3* para *s5*. O disparo das transições *t5* ou *t6* faz com que a ação *askConfirmation* seja ativada. Esta ação envia uma mensagem para o controlador solicitando a confirmação da autorização de pouso, visto que algo fora dos padrões fora encontrado. Neste ponto o protocolo está no estado *s5* e só existem duas transições de saída. A transição *t8* dispara quando o controlador rejeita o pouso e a transição *t7* dispara quando o controlador confirma a pouso.

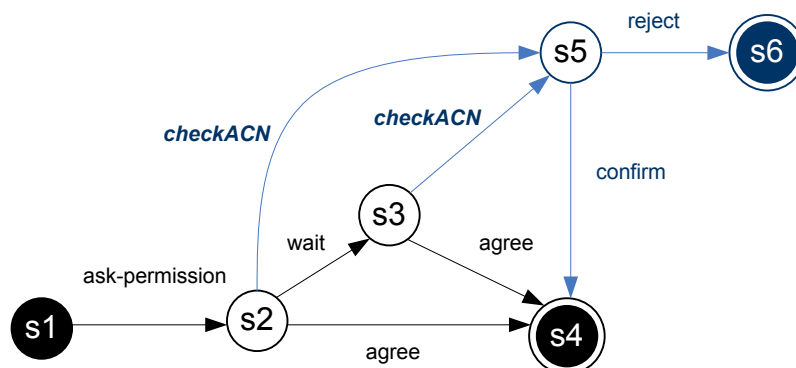


Figura 41 – Protocolo da Cena de Aterrissagem com a inclusão dos estados s5 e s6

```

landing{
  ask-permission{pilot, controller,
request(landingPermission)}
  wait{controller, pilot, inform(wait)}
  agree{controller, pilot, agree(permission)}
  confirm{controller, pilot, confirm(accept-proposal)}
  reject{controller, pilot, reject-proposal}

  s1{initial}
  s4{success}
  s6{failure}

  t1{s1->s2, ask-permission}
  t2{s2->s3, wait}
  t3{s3->s4, agree, [checkACN]}
  t4{s2->s4, agree, [checkACN]}
  t5{s2->s5, (checkACN, constraint_not_satisfied)}
  t6{s3->s5, (checkACN, constraint_not_satisfied)}
  t7{s5->s4, confirm}
  t8{s5->s6, reject}

  // constraints
  checkACN{br.les.CheckACN}

  // action
  askConfirmation{(t5,t6), br.les. AskConfirmation}
}

```

Código 51 – XMLaw de Mitigação da Ameaça CDL12

Ameaça: CDL13
Contexto: Cena aterrissagem
<p>Estratégia de detecção: para evitar que o piloto não execute o circuito de tráfego padrão ao pousar, foi adicionado ao protocolo uma etapa de monitoramento onde o piloto informa continuamente qual a sua posição e velocidade. Esta informação é transmitida através da mensagem <i>flightStrip</i>. O recebimento desta mensagem pelo mediador ativa a <i>action checkAndWarnCircuit</i>. Baseado no <i>flightStrip</i>, esta <i>action</i> verifica se o circuito está efetivamente sendo cumprido e caso não esteja avisa ao piloto e ao controlador.</p>

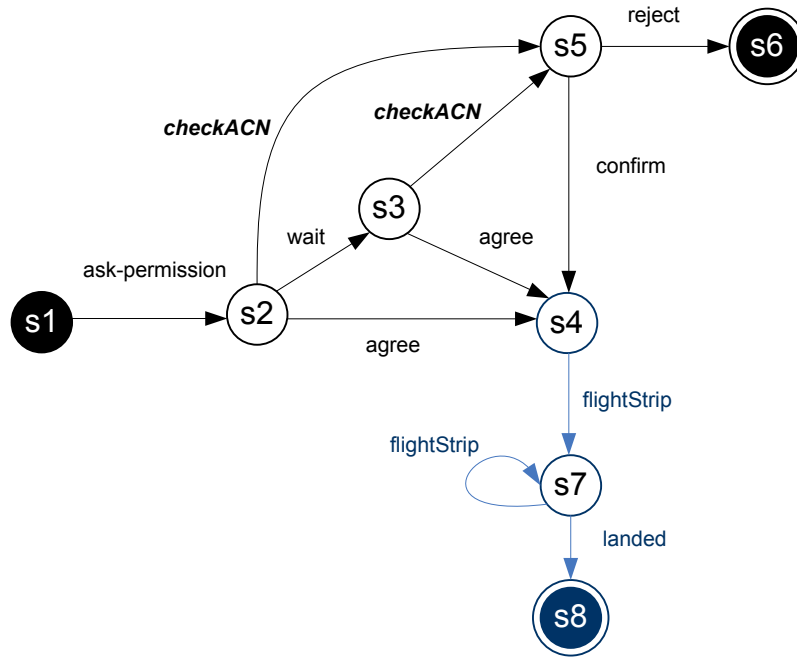


Figura 42 – Protocolo da Cena de Aterrissagem Modificado com a inclusão dos estados s7 e s8.

```

landing{
  ask-permission{pilot, controller,
request(landingPermission)}
  wait{controller, pilot, inform(wait)}
  agree{controller, pilot, agree(permission)}
  confirm{controller, pilot, confirm(accept-proposal)}
  reject{controller, pilot, reject-proposal}
  flightStrip{pilot, controller, inform($strip)}
  landed{pilot, controller, inform(landed)}

  s1{initial}
  s8{success}
  s6{failure}

  t1{s1->s2, ask-permission}
  t2{s2->s3, wait}
  t3{s3->s4, agree, [checkACN]}
  t4{s2->s4, agree, [checkACN]}
  t5{s2->s5, (checkACN, constraint_not_satisfied)}
  t6{s3->s5, (checkACN, constraint_not_satisfied)}
  t7{s5->s4, confirm}
  t8{s5->s6, reject}
  t9{s4->s7, flightStrip}
  t10{s7->s7, flightStrip}
  t11{s7->s8, landed}

  // constraints
  checkACN{br.les.CheckACN}

  // action
  askConfirmation{(t5,t6), br.les. AskConfirmation}
  checkAndWarnCircuit{(t9,t10),br.les.CheckAndWarnCircuit}

```

}

Código 52 – XMLaw de Mitigação da Ameaça CDL13**Ameaça: CDL14****Contexto:** Cena de voo

Estratégia de detecção: A mitigação desta ameaça pode ser implementada apenas modificando-se a estratégia de mitigação da ameaça CDL02. Para isto, basta mudar a implementação da *constraint checkAltitude* para verificar também se a aeronave está voando pelo menos 1000 pés acima do obstáculo mais alto num raio de 600 metros.

Ameaça: CDL15**Contexto:** Cena de voo

Estratégia de detecção: Mais uma vez, a implementação desta estratégia de mitigação consiste em modificar a *constraint* para verificar também se a aeronave está voando a uma altura superior a 500 pés em lugares habitados. Note-se, que embora tenha-se reaproveitado a mesma *constraint* para implementar as mitigações das ameaças CDL02, CLD14 e CLD15, poderia-se, ao invés disto, ter-se criado três *constraints* diferentes, sendo uma para cada ameaça.

7.8.Dependability Explicit Computing

As especificações das leis podem tratar explicitamente conceitos de fidedignidade, e auxiliar na coleta e publicação de dados sobre fidedignidade. Estes dados podem ser utilizados, por exemplo, para auxiliar na construção de aplicações guiando decisões tanto em tempo de projeto quanto em tempo de execução.

As principais vantagens da utilização de uma abordagem de leis para a especificação de preocupações de fidedignidade são: (i) definição explícita das preocupações; (ii) coleta automática de metadados usando a infra-estrutura de

mediadores presente na maioria das abordagens de leis; e (iii) habilidade de especificar estratégias para reagir a situações não-desejadas, auxiliando na prevenção de falhas de serviço. Nesta seção, ilustra-se como a abordagem proposta nesta tese foi utilizada para implementar *Dependability Explicit Computing(DepEx)* no estudo de caso de CTA.

Em uma abordagem de DepEx é importante definir quais são os metadados de fidedignidade de maneira explícita e, também o propósito do seu uso. Neste estudo de caso, utilizou-se uma abordagem de GQM (Van Solingen and Berghout 1999) para identificar estes metadados. O raciocínio é que através da utilização do mediador e da especificação das leis é possível obter informações da execução do sistema que podem auxiliar a análise da fidedignidade do sistema. O GQM auxilia na identificação de quais informações devem ser monitoradas para alcançar os objetivos da análise. Um dos propósitos do GQM é obter formas concretas de identificar se um determinado objetivo está sendo alcançado. Isto é feito identificando-se um conjunto de objetivos, questões que auxiliam na verificação do cumprimento destes objetivos e finalmente um conjunto de métricas que respondem às questões. De acordo com o GQM, um objetivo (*goal*) deve conter em sua declaração os seguintes itens:

- Objeto – o artefato que está em estudo
- Propósito – a motivação por trás do objetivo
- Foco – o atributo do objeto em estudo
- Ponto de Vista – a perspectiva do objetivo
- Ambiente – o escopo no qual o estudo se baseia

Desta forma, utilizou-se a estrutura acima como um guia para estruturar os objetivos. Estes objetivos são apresentados a seguir:

Objetivo 01 (G01)

A equipe de qualidade de voo deseja (ponto de vista)
 analisar os pilotos (objeto)
 para identificar (propósito)
 o desempenho em termos de falhas cometidas (foco)
durante a operação das aeronaves em todas as fases de voo (ambiente)

Objetivo 02 (G02)

A equipe de qualidade de voo deseja

analisar os controladores
para identificar
o desempenho ao recomendar ações fora dos padrões estabelecidos
durante a operação das aeronaves em todas as fases de voo

Objetivo 03 (G03)

A equipe de qualidade de voo deseja
analisar os radares
para identificar
a confiabilidade em termos de falhas de comunicação
durante a operação das aeronaves em todas as fases de voo

Objetivo 04 (G04)

A equipe de qualidade de voo deseja
analisar as estações meteorológicas
para identificar
a confiabilidade em termos de falhas de comunicação
durante a operação das aeronaves em todas as fases de voo

Objetivo 05 (G05)

A equipe de qualidade de voo deseja
analisar o canal de comunicação do controlador
para identificar
a confiabilidade
durante a operação das aeronaves em todas as fases de voo

A partir deste conjunto de objetivos definidos, é preciso identificar as questões que auxiliam a concluir se um determinado objetivo foi efetivamente alcançado. A estratégia seguida para identificar as questões foi reler as ameaças e entender o relacionamento dela com os objetivos. Por exemplo, a ameaça CDL01 é descrita como “aeronave se aproxima demais de outra aeronave”. Uma das causas desta aproximação indevida é por falha do piloto, que é exatamente o objetivo G01. Desta forma, a identificação desta ameaça serve como subsídio para

a coleta de dados que auxiliam na verificação do cumprimento do objetivo. A Figura 43 ilustra como o objetivo G01 pode ser verificado.

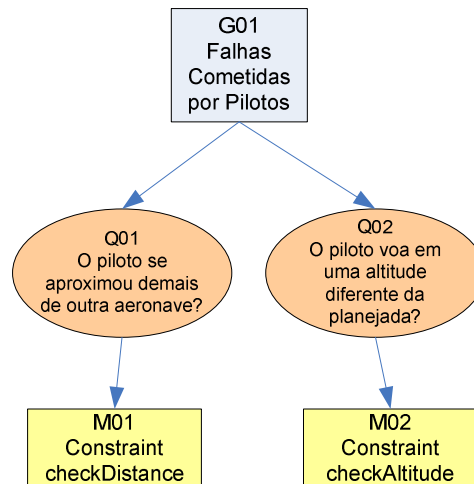


Figura 43 – GQM do Objetivo G01

As questões Q01 e Q02 são utilizadas para verificar o objetivo G01. A questão Q01 utiliza como métrica a violação da *constraint checkDistance*. Todas as vezes que esta *constraint* for violada, uma *action* é ativada como forma de atualizar o banco de dados com os metadados. O banco de dados de metadados possui a estrutura apresentada na Figura 44 e explicada na Tabela 15. Todas as vezes que a *action updateG01* é executada, ela insere no banco de dados uma tupla como exemplificado na Tabela 16.

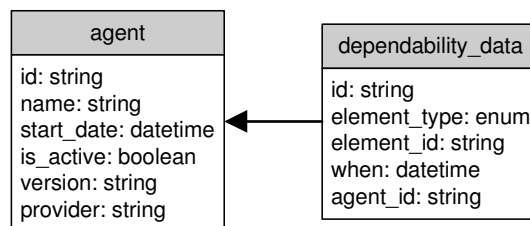


Figura 44 – Modelo do Banco de Dados

agent	dependability_data
id – identificador único do agente.	id – identificador único para o dado
name – nome do agente	element_type – tipo do elemento XMLaw (ex: obligation, clock, ...)
start_date – data de quando o agente foi adicionado ao banco de dados	element_id – id do elemento XMLaw
is_active – assume valor <i>true</i> se o agente está em	when – data da inserção do dado

execução	
version – versão do agente	agent_id – identificador do agente associado com estes dados
provider – organização que é responsável pelo agente	

Tabela 15 – Descrição dos Atributos do Banco de Dados

id	element_type	element_id	when	agent_id
1	constraint	checkDistance	2007-07-23 14:13:44	1

Tabela 16- Exemplo de Tupla Inserida no Banco de Dados

A partir do armazenamento destas informações, é possível responder as questões. Por exemplo, para saber se o piloto se aproximou demais da aeronave basta consultar quantas tuplas cujo campo *element_id* tem valor igual a *checkDistance*. O Código 53 mostra a cena de voo modificada para inserir a ação cuja implementação adiciona uma tupla no banco de dados.

```

flight{    //nome da cena

// Mensagens
  progressR{radar, controller, inform(strip,
$flightProgressStrip)}
  progressA{pilot, controller, inform(strip,
$flightProgressStrip)}
  switch{controller, pilot, inform(switch, $newController)}
  landing{pilot, controller, inform(landIntention)}

// Estados especiais
  s1{initial}
  s3{success}

// Transições
  t1{s1->s2, progressR, [checkDistance, checkAltitude]}
  t2{s1->s2, progressA, [checkDistance, checkAltitude]}
  t3{s2->s2, progressR, [checkDistance, checkAltitude]}
  t4{s2->s2, progressA, [checkDistance, checkAltitude]}
  t5{s2->s2, switch}
  t6{s2->s3, landing}

// Constraints
  checkDistance{br.les.CheckDistance}
  checkAltitude{br.les.CheckAltitude}

// Actions
  warnTcas{ ((checkDistance,constraint_not_satisfied)),
br.les.WarnTCAS}

  warnMsaw{ ( (checkAltitude,constraint_not_satisfied) ),
br.les.WarnMSAW}

```

```

warnPilot{
    (
        (checkDistance,constraint_not_satisfied),
        (checkAltitude,constraint_not_satisfied)
    ), br.les.WarnPilot}

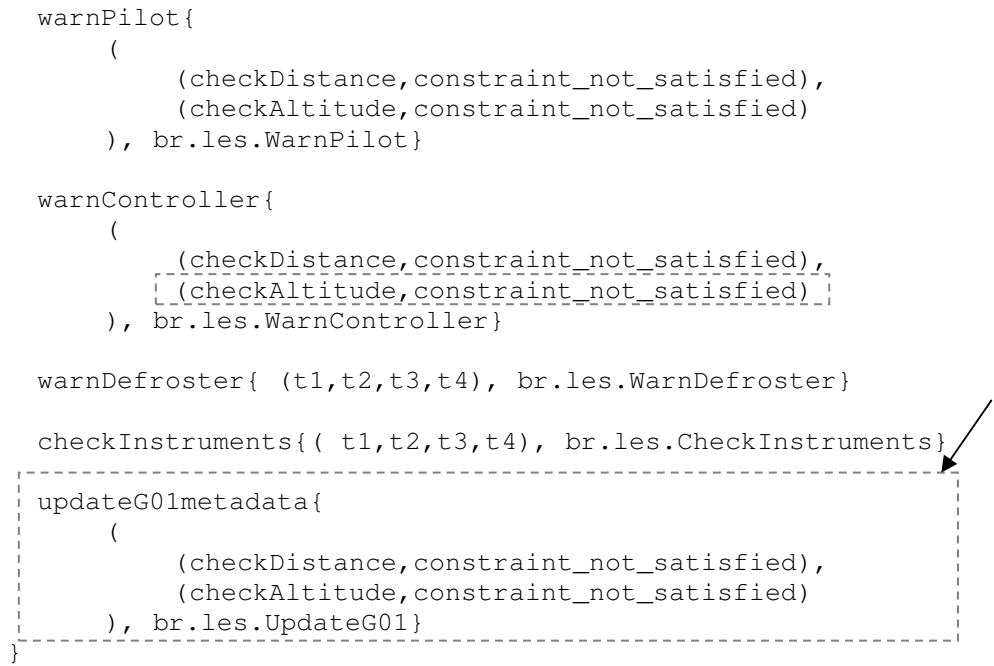
warnController{
    (
        (checkDistance,constraint_not_satisfied),
        [(checkAltitude,constraint_not_satisfied)]
    ), br.les.WarnController}

warnDefroster{ (t1,t2,t3,t4), br.les.WarnDefroster}

checkInstruments{( t1,t2,t3,t4), br.les.CheckInstruments}

updateG01metadata{
    (
        (checkDistance,constraint_not_satisfied),
        (checkAltitude,constraint_not_satisfied)
    ), br.les.UpdateG01}
}

```



Código 53 – Cena *flight* Modificada com a Inserção da *action* para Armazenar os Dados no Banco de Dados

Os outros objetivos são resolvidos de forma análoga ao G01. A Figura 45 mostra a aplicação do GQM para a identificação dos metadados que irão popular o banco de dados.

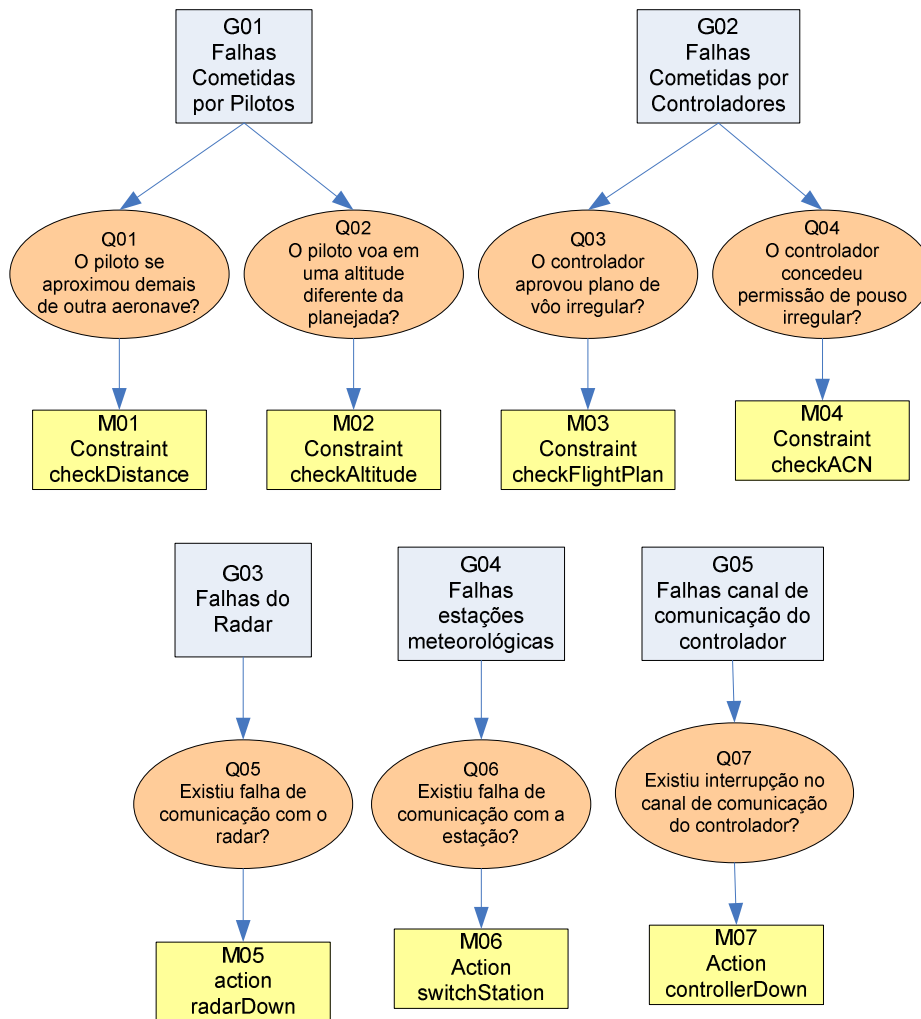


Figura 45 – GQM do Estudo de Caso para a Identificação dos Metadados

7.9. Resumo dos Artefatos de Lei Gerados no Estudo de Caso

Neste capítulo, apresentou-se como o estudo de caso foi construído. Para auxiliar a consolidar o resultado das discussões das sessões anteriores, esta seção apresenta os principais artefatos relacionados a governança gerados no estudo de caso.

7.9.1.Arquitetura

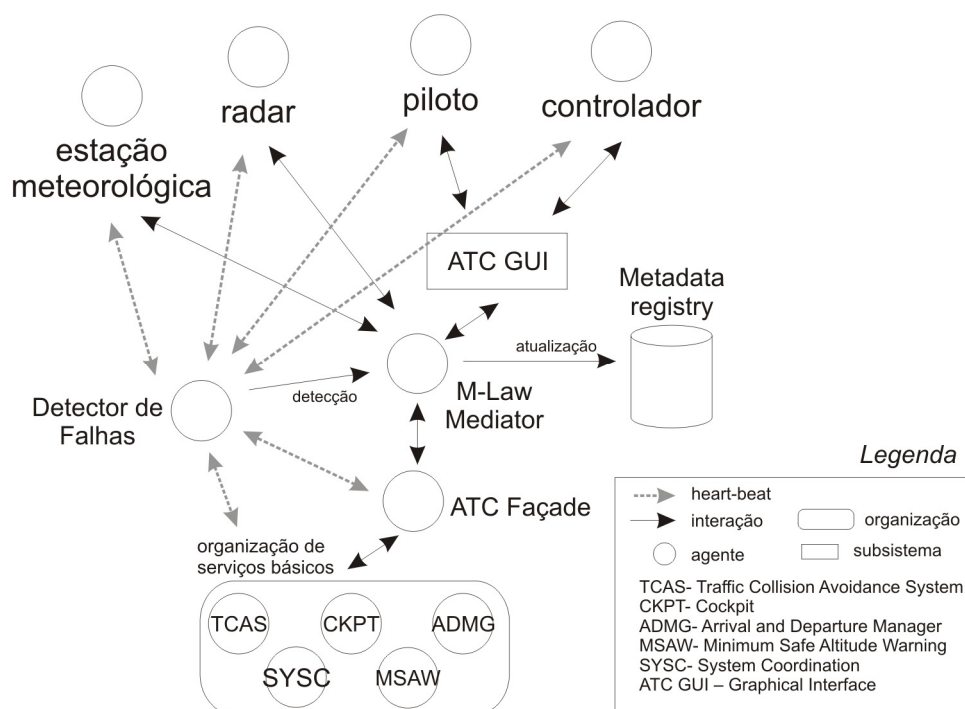


Figura 46 – Arquitetura do Estudo de Caso do Controle de Tráfego Aéreo

7.9.2.Lei

```

atc-law{
  // global actions
  switchStation{(failure_detector, agent_unavailable),
br.les.SwitchStation}

  radarDown{(failure_detector, agent_unavailable),
br.les.RadarDown}

  controllerDown{(failure_detector, agent_unavailable),
br.les.ControllerDown}

  //cena
  // #####
  groundControl{ //nome da cena

  // Mensagens
  flightPlan{pilot, controller,propose(flightPlan($content))}
  reject{controller, pilot, reject-proposal}
  accept{controller, pilot, accept-proposal}
  intent{pilot, controller, request(go-position)}
  position{controller, pilot, inform(pos, $instructions)}
  confirm{controller, pilot, confirm(accept-proposal)}

  // Estados especiais

```

```

s1{initial}
s3{failure}
s6{success}

// Transições
t1{s1->s2, flightPlan}
t2{s2->s3, reject}
t3{s2->s4, accept, [checkFlightPlan]}
t4{s4->s5, intent}
t5{s5->s6, position}
t6{s2->s7, (checkFlightPlan, constraint_not_satisfied)}
t7{s7->s4, confirm}
t8{s7->s3, reject}

// constraints
checkFlightPlan{br.les.CheckFlightPlan}

// action
askConfirmation{t6, br.les. AskConfirmation}
}

//cena
// #####
take-off{ //nome da cena

// Mensagens
request{pilot, controller, request(take-off)}
refuse{controller, pilot, refuse }
agree{controller, pilot, agree }
status{pilot, controller, inform($status)}
flight{pilot, controller, inform(flightPhase)}

// Estados especiais
s1{initial}
s3{failure}
s5{success}

// Transições
t1{s1->s2, request}
t2{s2->s3, refuse}
t3{s2->s4, agree}
t4{s4->s4, status}
t5{s4->s5, flight}

// Actions
warnPilot{ (t4), br.les.WarnPilot}
}

//cena
// #####
flight{ //nome da cena

// Mensagens
progressR{radar, controller, inform(strip,
$flightProgressStrip)}
progressA{pilot, controller, inform(strip,
$flightProgressStrip)}
switch{controller, pilot, inform(switch, $newController)}
landing{pilot, controller, inform(landIntention)}

```

```

// Estados especiais
s1{initial}
s3{success}

// Transições
t1{s1->s2, progressR, [checkDistance, checkAltitude]}
t2{s1->s2, progressA, [checkDistance, checkAltitude]}
t3{s2->s2, progressR, [checkDistance, checkAltitude]}
t4{s2->s2, progressA, [checkDistance, checkAltitude]}
t5{s2->s2, switch}
t6{s2->s3, landing}

// Constraints
checkDistance{br.les.CheckDistance}
checkAltitude{br.les.CheckAltitude}

// Actions
warnTcas{ ((checkDistance,constraint_not_satisfied)),
br.les.WarnTCAS}

warnMsaw{ ( (checkAltitude,constraint_not_satisfied) ),
br.les.WarnMSAW}

warnPilot{
(
    (checkDistance,constraint_not_satisfied),
    (checkAltitude,constraint_not_satisfied)
), br.les.WarnPilot}

warnController{
(
    (checkDistance,constraint_not_satisfied),
    (checkAltitude,constraint_not_satisfied)
), br.les.WarnController}

warnDefroster{ (t1,t2,t3,t4), br.les.WarnDefroster}

checkInstruments{( t1,t2,t3,t4), br.les.CheckInstruments}

updateG01metadata{
(
    (checkDistance,constraint_not_satisfied),
    (checkAltitude,constraint_not_satisfied)
), br.les.UpdateG01}
}

//cena
// #####
landing{
ask-permission{pilot, controller,
request(landingPermission)}
wait{controller, pilot, inform(wait)}
agree{controller, pilot, agree(permission)}
confirm{controller, pilot, confirm(accept-proposal)}
reject{controller, pilot, reject-proposal}
flightStrip{pilot, controller, inform($strip)}
landed{pilot, controller, inform(landed)}

s1{initial}
s8{success}

```



```

s6{failure}

t1{s1->s2, ask-permission}
t2{s2->s3, wait}
t3{s3->s4, agree, [checkACN]}
t4{s2->s4, agree, [checkACN]}
t5{s2->s5, (checkACN, constraint_not_satisfied)}
t6{s3->s5, (checkACN, constraint_not_satisfied)}
t7{s5->s4, confirm}
t8{s5->s6, reject}
t9{s4->s7, flightStrip}
t10{s7->s7, flightStrip}
t11{s7->s8, landed}

// constraints
checkACN{br.les.CheckACN}

// action
askConfirmation{(t5,t6), br.les. AskConfirmation}
checkAndWarnCircuit{(t9,t10),br.les.CheckAndWarnCircuit}
}

} //end law

```

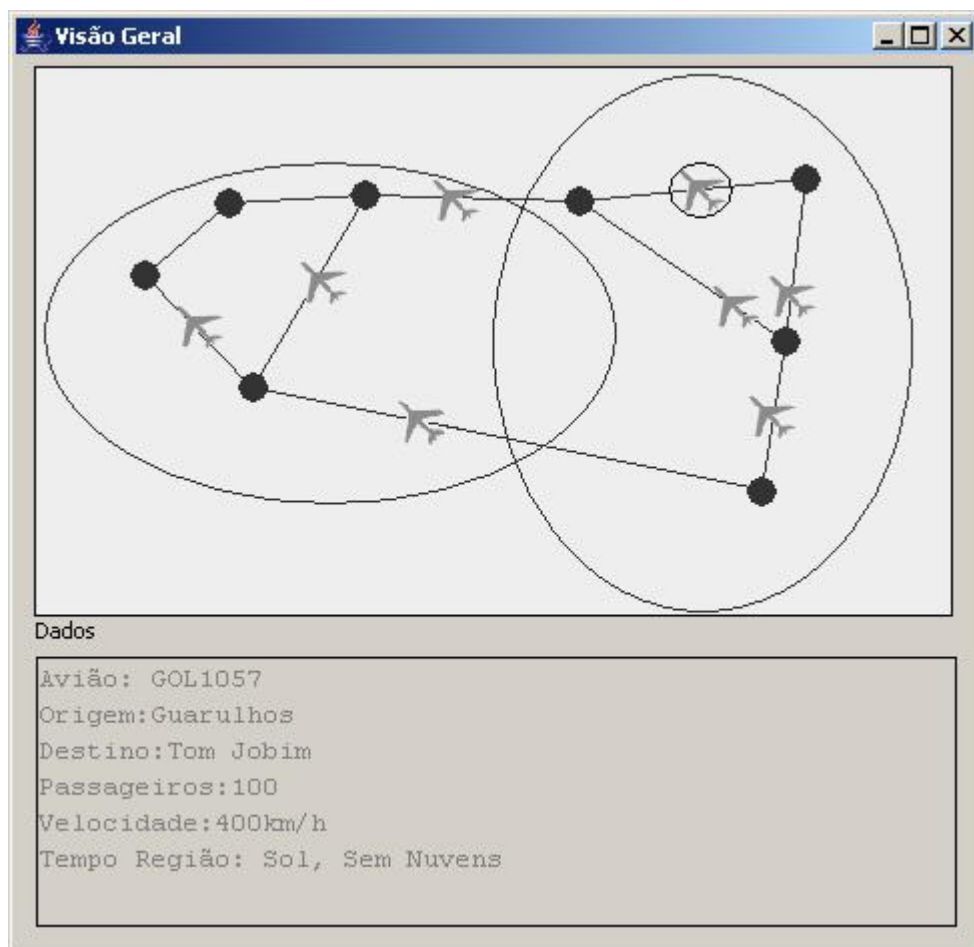


Figura 47 – Tela da Visão Geral do Sistema

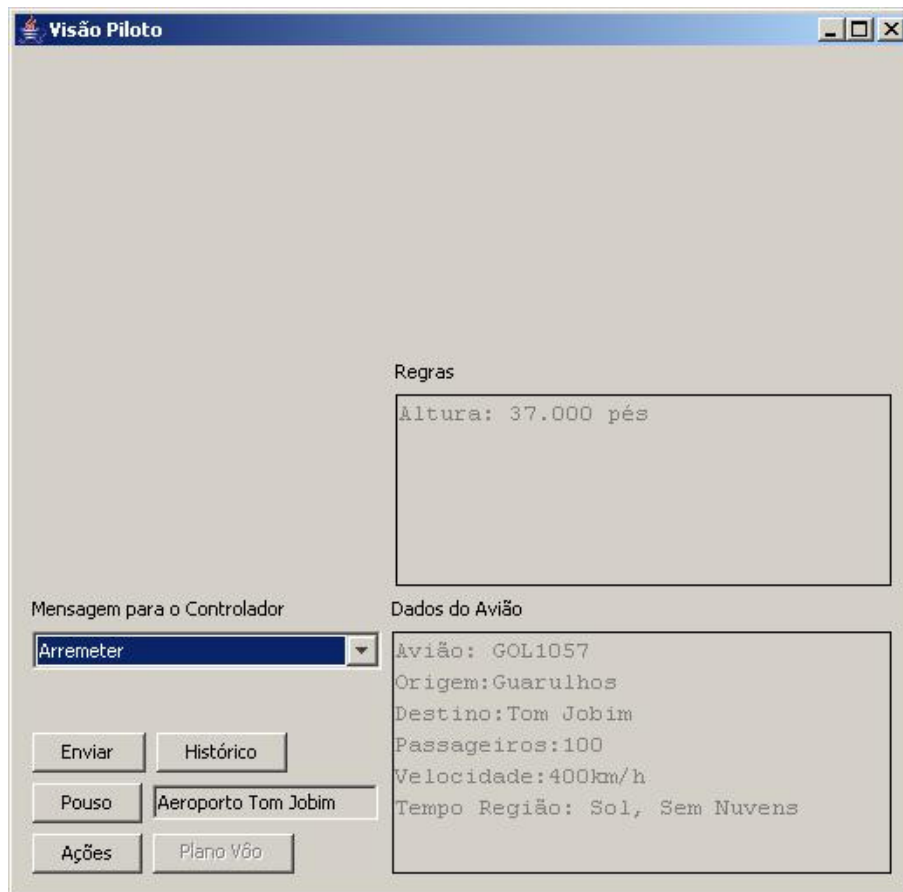


Figura 48 – Tela Representando o Agente Piloto

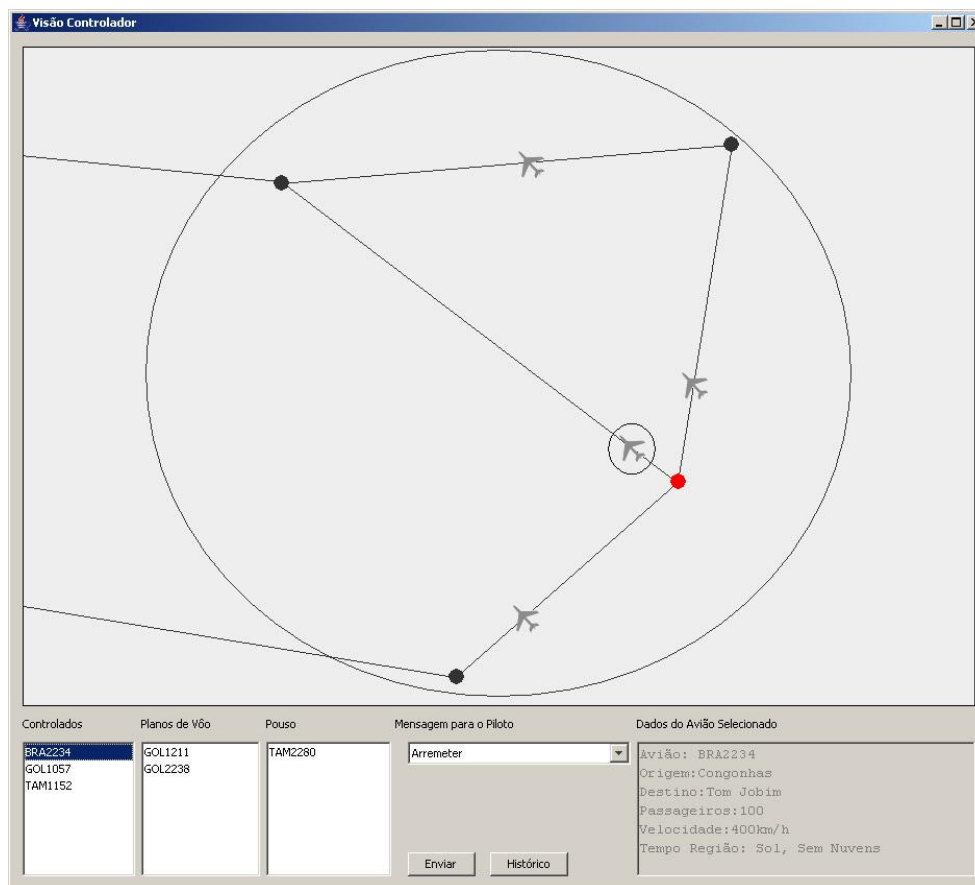


Figura 49 – Tela Representando a Visão do Agente Controlador

7.10.Considerações Finais

O estudo de caso foi implementada utilizando o XMLaw e o M-Law e os agentes foram implementados em Java. O estudo de caso de CTA possui elementos de incerteza, criticalidade, distribuição e complexidade de interações que certamente o tornam adequado para a experimentação da abordagem proposta. Através do XMLaw foi possível representar vários aspectos de governança. Mas o principal aspecto a ser destacado é que a utilização de uma abordagem de governança pode efetivamente incorporar aspectos de fidedignidade. Primeiramente, o domínio do problema foi apresentado e vários casos de uso foram definidos. Uma arquitetura baseada em agentes foi proposta para representar os atores que fazem parte do sistema. A partir daí, foi realizada uma análise detalhada de várias ameaças que podem impactar o bom funcionamento do sistema. Estas ameaças foram todas mitigadas através da utilização dos mecanismos propostos nesta tese.

O modelo conceitual do XMLaw permitiu expressar as situações onde se verifica a existência de uma ameaça. Além disso, também foi possível expressar as estratégias de mitigação utilizando a própria linguagem.

Foram simuladas situações em que a ameaça se tornava concreta. O middleware M-Law foi capaz de perceber estas situações, através da especificação das leis e do monitoramento da execução do sistema, e agir de acordo com as estratégias de implementação especificada.

De forma geral, pode-se concluir que a abordagem proposta nesta tese foi capaz de representar estratégias de mitigação apresentadas e o middleware foi capaz de monitorar e agir de acordo com as especificações. Desta forma, mostra-se que preocupações de fidedignidade puderam ser tratadas com a abordagem proposta.