sid.inpe.br/mtc-m21c/2021/02.13.01.58-TDI

PLANEJAMENTO AUTOMÁTICO DE OPERAÇÃO DE SATÉLITES COM PREDIÇÃO DE ESTADOS INVÁLIDOS

Caio Gustavo Rodrigues da Cruz

Dissertação de Mestrado do Curso de Pós-Graduação em Engenharia e Tecnologia Espaciais/Engenharia e Gerenciamento de Sistemas Espaciais, orientada pelos Drs. Maurício Gonçalves Vieira Ferreira, e Rodrigo Rocha Silva, aprovada em 04 de março de 2021.

URL do documento original:
<http://urlib.net/8JMKD3MGP3W34R/446GAE8>

INPE
São José dos Campos
2021

PUBLICADO POR:

Instituto Nacional de Pesquisas Espaciais - INPE
Coordenação de Ensino, Pesquisa e Extensão (COEPE)
Divisão de Biblioteca (DIBIB)
CEP 12.227-010
São José dos Campos - SP - Brasil
Tel.:(012) 3208-6923/7348
E-mail: pubtc@inpe.br

sid.inpe.br/mtc-m21c/2021/02.13.01.58-TDI

PLANEJAMENTO AUTOMÁTICO DE OPERAÇÃO DE SATÉLITES COM PREDIÇÃO DE ESTADOS INVÁLIDOS

Caio Gustavo Rodrigues da Cruz

Dissertação de Mestrado do Curso de Pós-Graduação em Engenharia e Tecnologia Espaciais/Engenharia e Gerenciamento de Sistemas Espaciais, orientada pelos Drs. Maurício Gonçalves Vieira Ferreira, e Rodrigo Rocha Silva, aprovada em 04 de março de 2021.

URL do documento original:
<http://urlib.net/8JMKD3MGP3W34R/446GAE8>

INPE
São José dos Campos
2021

Dados Internacionais de Catalogação na Publicação (CIP)

Cruz, Caio Gustavo Rodrigues da.

C889p Planejamento automático de operação de satélites com predição de estados inválidos / Caio Gustavo Rodrigues da Cruz. – São José dos Campos : INPE, 2021.

xxii + 85 p. ; (sid.inpe.br/mtc-m21c/2021/02.13.01.58-TDI)

Dissertação (Mestrado em Engenharia e Tecnologia Espaciais/Engenharia e Gerenciamento de Sistemas Espaciais) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2021.

Orientadores : Drs. Maurício Gonçalves Vieira Ferreira, e Rodrigo Rocha Silva.

1. Planejamento automático. 2. Estados inválidos. 3. Classificação de dados. 4. PDDL. 5. Sistemas de informação. I.Título.

CDU 629.7.05:629.78

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÕES

INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

Serviço de Pós-Graduação

DEFESA FINAL DE DISSERTAÇÃO DE CAIO GUSTAVO RODRIGUES DA CRUZ

BANCA Nº 010/2021, REGISTRO 141267/2018

No dia 04 de março de 2021, às 09h, por videoconferência, o aluno mencionado acima defendeu seu trabalho final (apresentação oral seguida de arguição) perante uma Banca Examinadora, cujos membros estão listados abaixo. O aluno foi APROVADO pela Banca Examinadora, por UNANIMIDADE, em cumprimento ao requisito exigido para obtenção do Título de Mestre em Engenharia e Tecnologia Espaciais/Eng. Gerenc. de Sistemas Espaciais. O trabalho precisa da incorporação das correções sugeridas pela Banca Examinadora e revisão final pelos ORIENTADORES.

Título Novo: "PLANEJAMENTO AUTOMÁTICO DE OPERAÇÃO DE SATÉLITES COM PREDIÇÃO DE ESTADOS INVÁLIDOS"

Eu, Walter Abrahão dos Santos, como Presidente da Banca Examinadora, assino esta ATA em nome de todos os membros.

Membros da Banca

Dr. Walter Abrahão dos Santos - Presidente - INPE
Dr. Maurício Gonçalves Vieira Ferreira - Orientador - INPE
Dr. Rodrigo Rocha Silva - Orientador - FATEC
Dra. Ana Maria Ambrosio - Membro da banca - INPE
Dr. Jorge Fernandes Rodrigues Bernardino - Convidado - Instituto Politécnico de Coimbra

---

Documento assinado eletronicamente por **Walter Abrahão dos Santos**, **Tecnologista**, em 11/03/2021, às 15:17 (horário oficial de Brasília), com fundamento no art. 6º do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.mctic.gov.br/verifica.html, informando o código verificador **6570386** e o código CRC **2740111D**.

"A simplicidade é o último grau de sofisticação.".

Leonardo da Vinci

A meus pais ***Ivair*** *e* ***Eliana*** *e à minha irmã* ***Rayane****.*

AGRADECIMENTOS

Agradeço primeiramente a Deus por ter me dado saúde, força e colocado pessoas incríveis no meu caminho que tanto me ajudaram a seguir nessa difícil jornada. Em especial meu grande amigo Rodrigo Rocha, que sempre me incentivou, agradeço imensamente todo o seu apoio, orientação, ensinamento e oportunidade que tem me dado desde a graduação e que foram fundamentais para que eu conseguisse realizar este trabalho.

Agradeço ao meu amigo e orientador Mauricio Ferreira pela grande ajuda e oportunidade, por todo conhecimento e orientação que me passou durante a minha jornada no INPE.

Agradeço especialmente a minha mãe Eliana, meu pai Ivair e minha irmã Rayane por me incentivarem nas minhas escolhas e fornecerem todo apoio necessário.

Agradeço a todos que tiveram paciência e boa vontade, que colaboraram de alguma forma. Meus amigos do INPE por grande ajuda durante as disciplinas.

Agradeço as contribuições do professor Jorge Bernardino no desenvolvimento dos artigos.

Agradeço particularmente ao fundamental suporte que a Muralis Assessoria e Tecnologia me deu durante minha jornada no mestrado e pelo apoio na publicação de artigos.

RESUMO

O aumento do uso de sistemas automatizados em missões espaciais fez com que aspectos como segurança e confiabilidade das operações espaciais sejam tratados com maior cautela devido à habitual degradação de um satélite durante seu tempo de operação. Em diversos trabalhos na literatura que têm como objetivo o planejamento automático, observa-se que planos são gerados com estados inválidos. O estado inválido pode ser compreendido como um cenário proibido ou que coloque em risco a operação que demanda o planejamento automático. Nesse contexto, este trabalho tem como objetivo apresentar uma nova arquitetura de planejamento com restrições. Para que planejadores automáticos não gerem planos com estados inválidos, foi proposta a adição de um método responsável por validar estados no software planejador. O método de validação proposto tem como base uma nova representação de restrições, que é configurada por meio de um processo de aprendizagem de máquina. Desta maneira, o planejamento automático para a área espacial pode ser beneficiado pela arquitetura apresentada neste trabalho, já que a base de dados da operação de um satélite pode ser utilizada para configurar restrições, gerar planos com maior qualidade e aumentar a segurança da operação. Através do estudo de caso realizado, observa-se que a validação dos estados inválidos possibilita que o planejador cumpra as restrições e garanta que o objetivo do problema seja alcançado com apenas estados permitidos. Por fim, é possível concluir que os planos gerados com base na utilização da arquitetura apresentada nesta dissertação contribuem para que as restrições emergentes do domínio da área espacial, como a queima de um subsistema, sejam representadas e cumpridas no planejamento automático.

Palavras-chave: Planejamento Automático. Estados Inválidos. Classificação de Dados. PDDL. Sistemas de Informação.

AUTOMATIC SATELLITE OPERATION PLANNING WITH PREDICTION OF INVALID STATES

ABSTRACT

The increase in the use of automated systems in space missions has made that aspects such as safety and reliability of space operations are treated with greater caution due to the natural degradation of a satellite during its operating time. In several works in literature that focus on automatic planning, it is observed that plans are generated with invalid states. The invalid state can be understood as a prohibited scenario or one that jeopardizes the operation that requires automatic planning. In this context, this work aims to present a new planning architecture with restrictions. In order for automatic planners to not generate plans with invalid states, it was proposed to add a method that is responsible for the validation of states in the planner software. The proposed validation method is based on a new representation of restrictions, which is configured through a machine learning process. In this way, the automatic planning for space can be benefited by the architecture presented in this work, since the knowledge base of operation of a satellite can be used to configure restrictions, generate plans with higher quality and increase the security of the operation. Through the experiments carried out, it is observed that the validation of invalid states enables the planner to comply with the restrictions and ensure that the objective of the problem is achieved with only allowed states. Finally, it is possible to conclude that the plans generated based on the use of the architecture presented in this thesis, contributes so that the restrictions emerging from the domain of the space area, such as the burning of a subsystem, are represented and fulfilled in the automatic planning.

Keywords: Automated Planning. Invalid States. Machine Learning. PDDL. Information Systems.

LISTA DE FIGURAS

Pág.

LISTA DE TABELAS

Pág.

LISTA DE ABREVIATURAS E SIGLAS

AI	–	*Artificial Intelligence* (Inteligência artificial)
CCS	–	Centro de Controle de Satélites
HTN	–	*Hierarchical Task Network* (Rede de Tarefas Hierárquicas)
MDP	–	*Markov Decision Process* (Processo de decisão de Markov)
OBC	–	*Onboard Computer* (Computador de bordo)
PBP	–	*Preference-based Planning* (Planejamento baseado em preferências)
PDDL	–	*Planning Domain Definition Language* (Linguagem de definição de domínio de planejamento)
PSP	–	*Partial Satisfaction Planning* (Planejamento de satisfação parcial)
SDATF	–	Sistema de Determinação de Atitude com Tolerância a Falhas
SLP	–	*Scientific Langmuir Proble payload* (Carga útil da Sonda Langmuir Científica)
SMDH	–	Santa Maria Design House
STCC	–	*Satellite Tracking and Control Center* (Centro de controle e rastreamento de satélites)
STRIPS	–	*Stanford Research Institute Problem Solver* (Solucionador de problemas do Stanford Research Institute)

SUMÁRIO

Pág.

1 INTRODUÇÃO

O Centro de Rastreio e Controle de Satélites (CRC) no INPE, realiza as operações de controle de satélites através de algumas atividades executadas manualmente, como a geração do plano de voo do satélite (TOMINAGA et al., 2011). A geração automática de planos de operação de voo tem sido alvo de inúmeras pesquisas com aplicação de sistemas e planejadores automáticos, para futuras automações na operação de satélites (RIBEIRO, 2013); (CARDOSO et al., 2006); (KUCINSKIS et al., 2007); (SILVA et al., 2010); (AMIGONI et al., 2010); (TOMINAGA et al., 2011).

O tópico de planejamento automático é aplicado em diversos domínios e problemas de diferentes áreas. Na área espacial, a geração de planos de voo para o controle de satélites artificiais é um exemplo desta aplicação. No entanto, por se tratar de um domínio complexo, a validação de planos é necessária (SOUZA, 2012). A precaução em gerar planos condizentes com a situação atual do domínio tem incentivado o desenvolvimento de aplicações responsáveis por diagnosticar e validar planos gerados automaticamente. Além dos quesitos de segurança e confiabilidade das operações com satélites terem sido intensificados com o crescente número de sistemas automatizados nessa área de domínio (TOMINAGA et al., 2011).

A etapa de validação de planejamento exige que todos os estados pelo qual o domínio percorrerá para alcançar o objetivo do problema, sejam verificados no plano e havendo uma ou mais situações inválidas, que comprometam o domínio e não possam ser executados, o plano é considerado rejeitado (SOUZA, 2012). Planos inválidos podem ser gerados porque a real situação do domínio pode não ser considerada no planejamento, devido a fatores como, por exemplo, a degradação física que os satélites estão sujeitos (CARDOSO et al., 2006).

Diversos domínios de planejamento possuem o problema de ter um grande conjunto de soluções ou um conjunto de metas que não podem ser completamente alcançadas. Ainda há cenários onde podem ser gerados planos para soluções esperadas que contenham estados inválidos dadas características do domínio. Nesses casos, o modo como o objetivo do problema é encontrado pode ser mais importante do que alcançar o objetivo propriamente dito. Por esse motivo, é importante gerar planos de melhor qualidade, atingindo todas as metas do problema, quando possível, ou algum subconjunto delas (BAIER et al., 2008).

Essa deficiência abre caminhos para os temas de muitas pesquisas em planejamento de Inteligência Artificial (IA), por exemplo, o planejamento com restrições, com pre-

ferências dos usuários, de problemas complexos e o sobre incerteza. Esses temas são encontrados na literatura em trabalhos que criam linguagens de planejamento, técnicas ou fazem a implementação de planejadores para atender restrições específicas do problema de planejamento (GEREVINI; LONG, 2005).

Uma abordagem que elimine estados errôneos da solução planejada de forma automática é a motivação deste trabalho. Com os resultados obtidos com o desenvolvimento desta proposta, a partir da implementação de um validador de estados inválidos no planejador, foi possível comprovar que a geração de planos é capaz de seguir as restrições impostas ao domínio. Assim, o planejamento automático voltado para a área espacial poderá se beneficiar desta abordagem.

1.1 Objetivo

O objetivo deste trabalho é propor uma abordagem para geração automática de planos com validação de estados inválidos, para permitir que as restrições do domínio sejam consideradas. Nesta abordagem o método validador de estados faz uso de uma nova representação para validar os estados que não devem compor a solução planejada. Neste caso, planos válidos são gerados respeitando as mudanças que ocorrem no estado de operação do domínio por meio de um processo de aprendizagem e configuração de restrições.

Assim, apresenta-se a criação de uma abordagem para planejadores baseada no Solucionador de Problemas do Stanford Research Institute (STRIPS), que permite validar os estados do plano em tempo de planejamento.

Com o objetivo de validar a arquitetura proposta, utilizou-se a especificação de um domínio da área espacial para a criação de um estudo de caso. Onde foram configurados grupos de estados inválidos, classificados no processo de aprendizagem do status de operação do domínio. A validação foi realizada a partir da comparação do resultado do planejamento realizado com cada grupo configurado.

Para o estudo de caso, foi considerada a análise e criação de estados inválidos relacionados com o subsistema de suprimento de energia de um satélite, porque de acordo com um estudo feito por Mak Tafazoli (TAFAZOLI, 2009), conclui-se que 27% das falhas em espaçonaves são devido ao subsistema de suprimento de energia. A mineração de dados tem sido aplicada para detecção dessas anomalias em telemetrias de satélite (AZEVEDO et al., 2012) (GALAL et al., 2019) (ABDELGHAFAR et al., 2019).

1.2 Contribuição

Este trabalho contribui com uma nova maneira de validar os estados inválidos contidos em domínios de planejamento. Permitindo que essa abordagem seja aplicada para validação de restrições emergentes não apenas no domínio de planejamento com satélites, mas em domínios de planejamento em que estados sejam modificados com frequência e novas restrições possam surgir, como por exemplo, no domínio da logística de containers que foi utilizado nos experimentos realizados durante o desenvolvimento desta dissertação.

No domínio de planejamento podem ocorrer mudanças com o passar do tempo, com surgimento de novas regras ou determinadas situações podem não estar disponíveis em alguns momentos, tornando-se um grande problema em gerar planos inválidos. Assim como em outras abordagens de planejamento, a mineração de dados é empregada para melhorar o processo, aprendendo novas regras. Este trabalho contribui também com o emprego de novas tecnologias que permitem realizar a configurar de estados inválidos emergentes no domínio para aprimorar o processo de planejamento automatizado.

1.3 Organização do trabalho

Os capítulos restantes desta proposta estão organizados da seguinte maneira:

- **Capítulo 2**: Este capítulo apresenta os conceitos e fundamentos teóricos deste trabalho, como os conceitos relevantes de operação de satélites, *Planning*, *Planning with preferences* e *Planning Domain Definition Language* (Linguagem de Definição de Domínio de Planejamento) PDDL.

- **Capítulo 3**: Neste capítulo, os trabalhos correlatos de planejamento com preferências são apresentados.

- **Capítulo 4**: Neste capítulo, a arquitetura é apresentada e seus conceitos principais explicados.

- **Capítulo 5**: Visa apresentar o desenvolvimento e implementações, demonstrando as validações preliminares realizados em um planejador.

- **Capítulo 6**: Neste capítulo, é apresentado o estudo de caso realizado, configurando estados inválidos em um domínio para área espacial.

- **Capítulo 7**: Com base nos resultados alcançados, esse capítulo apresenta as conclusões obtidas, contribuições, pontos favoráveis e desfavoráveis, publicações e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os conceitos fundamentais relacionados a essa proposta, começando pela operação dos satélites na Seção 2.1, apresentando a definição de *Planning* na Seção 2.2, e o conceito da Linguagem de Definição de Domínio de Planejamento (PDDL) na Seção 2.3.

2.1 Operação de satélites

Um sistema espacial é formado por três segmentos: o segmento solo, o lançador e o segmento espacial. O segmento espacial é composto por um ou mais satélites e o segmento solo é formado pelo centro de controle de satélites, centro de missão espacial e estação terrena. O controle, monitoramento e comunicação com os satélites é realizado através de um sistema de telemetrias e telecomandos, os quais transportam as informações entre os segmentos do sistema espacial (RABENAU et al., 2002).

As telemetrias transmitem diversas informações como: o estado do satélite, as medições realizadas nos sensores, o status da execução de comandos, mensagens de erro e alertas sobre anomalias detectadas. De outro modo, os telecomandos são gerados e validados no Centro de Controle de Satélites (CCS), enviados para o satélite, onde são agendados para execução (KUCINSKIS et al., 2007).

As tarefas realizadas na operação de satélites são baseadas no planejamento de missão espacial, onde o desenvolvimento de um plano é feito para as atividades executadas durante a operação de satélites que devem estabelecer como alcançar os objetivos da missão de maneira possível, eficiente e segura (RABENAU et al., 2002).

A operação de uma missão espacial consiste em uma atividade sob responsabilidade do segmento solo, que tem como objetivo (ISO, 2004):

- Garantir a provisão da missão e dos serviços e produtos científicos;
- Executar as tarefas de operações de rotina;
- Recuperar as contingências de bordo;
- Gerenciar os recursos de bordo de forma a maximizar a vida útil da missão e as provisões de serviços e produtos.

O plano de operação de voo tem como objetivo estabelecer os comandos necessários para manter o satélite operando de modo que a sua missão seja alcançada. Então

as informações contidas no plano devem atender aos procedimentos de controle de voo, procedimentos de recuperação de contingências, regras, planos e cronogramas (SOUZA, 2012).

A operação de satélites é realizada por diferentes grupos, sendo: grupo de controle de voo, grupo de dinâmica de voo, grupo de operações de solo e o grupo de exploração de missão (ECSS-E-70 A, 2000). As diversas tarefas da operação de missões espaciais são preparadas e executadas pelas equipes. Para que seja executada a operação, uma infraestrutura contendo recursos de hardware e software devem estar disponíveis para que a operação seja realizada.

Na operação de rotina das missões espaciais, as tarefas de preparação, execução e avaliação, conduzem a execução das operações contidas no plano de voo. O planejamento de atividades de operações futuras e de atividades de manutenção é realizado com as tarefas de preparação, que produzem cronogramas detalhados a partir de pedidos dos usuários. A implementação das ações contidas no plano é coberta por tarefas de execução, incluindo a execução de ações corretivas necessárias no caso de anomalias (SOUZA, 2012).

2.2 Planejamento automático

O planejamento automatizado é uma área da IA focada na solução de problemas de um domínio. Um domínio de planejamento é composto por um conjunto finito de estados possíveis $S = \{s_1, s_2...s_k\}$, e um conjunto finito de ações $A = \{a_1, a_2...a_k\}$, aplicável aos estados do domínio (GHALLAB et al., 2004). Um problema de planejamento é originado quando há necessidade de transformar um estado inicial s_i em um conjunto de estados finais Sg. A sequência de ação $< a_1, a_2, ...a_k >$, quando aplicada nos resultados da ordem no estado final, é chamada de plano. A seguir, uma descrição do sistema de transição de estados usado no planejamento automático é mostrada:

$$s_1 = \gamma(s_0, a_1),\ s_2 = \gamma(s_1, a_2), ...,\ s_k = (s_{k-1}, a_k)$$
$$and\ s_k \in S_g$$

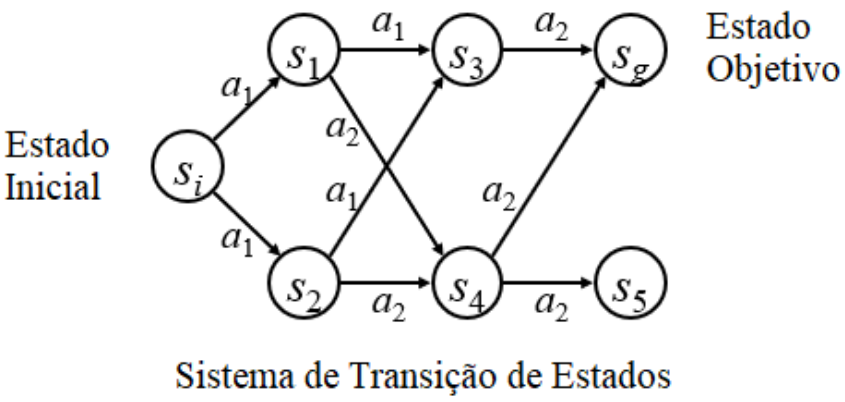
A função $\gamma(s, a)$ aplica uma ação a um estado, resultando em um sistema de transição de estado. Para cada estado gerado, uma ação é aplicada até que o estado gerado corresponda ao estado final. Um problema pode ser resolvido de inúmeras maneiras, ou seja, conjuntos de etapas infinitas podem transformar o estado inicial no estado final (GHALLAB et al., 2004).

O planejamento automatizado é realizado usando um software de planejamento (planejador) que encontra a sequência de etapas que transforma o estado inicial no estado final (MCDERMOTT et al., 1998). Normalmente um planejador utiliza duas representações para resolver um problema: a representação de um domínio, que define as ações aplicáveis, e a representação de um problema, que define os estados inicial e final.

A técnica usada para encontrar soluções de um domínio e um problemas é o STRIPS, que passa por todos os estados possíveis após aplicar as ações do domínio até encontrar o estado final (FIKES; NILSSON, 1971). A linguagem mais comum no planejamento automatizado é a PDDL, introduzida em 1998 por Drew McDermott (MCDERMOTT et al., 1998).

Um problema de planejamento pode ser compreendido a partir da seguinte representação: $P = (\Sigma, s_i, S_g)$, onde Σ é o sistema de transição de estados, s_i é o estado inicial e S_g é um conjunto de estados do objetivo. E no conceito de planejamento clássico, um plano é não determinístico, podendo haver caminhos diferentes para encontrar a sequência de ações que transformem o estado inicial S_i no estado objetivo S_g, como pode se observar na Figura 2.1.

Figura 2.1 - Sistema de transição de estados não determinísticos.



Fonte: Produção do autor.

A PDDL foi estabelecida como uma linguagem padrão comumente utilizada em planejamento automatizado. O PDDL abre caminho para uma colaboração mais forte, fornecendo uma plataforma para abordagens de benchmarking, trocando diferentes ferramentas, bem como técnicas e problemas, graças a uma série de competições internacionais de planejamento que foram associadas e estendidas em várias etapas para capturar variantes mais expressivas (VALLATI et al., 2015).

Com o tempo, as abordagens de planejamento evoluíram para resolver problemas mais complexos com novas técnicas. Eles abraçam a inclusão de preferências, pré-processamento usando macros e aplicação de aprendizagem usando mineração de dados em diferentes contextos de planejamento, que são abordados no capítulo 3.

2.3 Linguagens de planejamento

A seguir apresenta-se algumas linguagens de planejamento em IA encontradas na literatura, bem como os recursos, técnicas utilizadas e um comparativo.

2.3.1 A linguagem de definição de domínio de planejamento

Com base na sintaxe Lisp (STEELE, 1990), a linguagem PDDL (MCDERMOTT et al., 1998) usa uma estrutura baseada nas variantes amplamente usadas de notações de tiras. O estabelecimento de uma linguagem padrão comum teve um impacto semelhante em pesquisas sobre planejamento, como a introdução de padrões em outras áreas da pesquisa, abrindo caminho para uma colaboração mais forte, o intercâmbio de ferramentas, técnicas, problemas diferentes e fornecendo uma plataforma de avaliação comparativa de abordagens.

A PDDL foi estendida em vários estágios para capturar variantes mais expressivas. Houve várias explorações do poder expressivo das diferentes variantes do PDDL. Resultados recentes incluem uma demonstração de que características temporais podem ser compiladas em trabalho polinomial, sujeitas a certas restrições nas formas de simultaneidade que podem aparecer no problema (RINTANEN et al., 2007), enquanto outros examinaram a compilabilidade de efeitos condicionais, literais iniciais cronometrados e axiomas de domínio (FOX et al., 2005), (NEBEL, 2000), (THIÉBAUX et al., 2005).

2.3.2 A extensão da PDDL

Nos trabalhos de (GEREVINI; LONG, 2005) e (GEREVINI; LONG, 2006), PDDL foi estendida para incluir dois novos recursos importantes. A capacidade de expressar

objetivos que se aplicam não apenas ao estado final da trajetória dos estados visitados por um plano, mas também aos estados intermediários. Esses objetivos assumem a forma de restrições de trajetória, familiares no trabalho de lógica temporal. Outra extensão é a capacidade de expressar restrições ou leves preferências.

Na busca por atingir mais objetivos de um problema e gerar planos de qualidade, melhorando a quantidade de critérios expressados em planos complexos, a linguagem PDDL foi estendida para uma *Preference-based Planning* PBP (Planejamento Baseado em Preferências). A PDDL3 usa Rede de Tarefas Hierárquicas (HTN) para incluir até três tipos de preferências, aumentando o poder expressivo sobre a especificação de qualidade do plano. (JORGE et al., 2008).

A Figura 2.2 mostra como são definidas as preferências na linguagem PDDL3. No exemplo foram incluídas três preferências sobre: *break*, *social* e *reviewing*. E no final, o método atribui os valores quando cada preferência for violada após a execução do plano. Sendo o tempo total de planejamento o maior fator.

Figura 2.2 - Exemplo de codificação PDDL3.

```
(:goal (and (finished job1)
            (finished job2)
            (finished job3)) )

(:constraints (and (preference break (sometime (at coffee-room)))
                   (preference social (sometime (and (at coffee-room)
                                                     (coffee-time))))
                   (preference reviewing (reviewed paper1))) )

(:plan-metric minimize (+  (* 5 (total-time))
                           (* 4 (is-violated social))
                           (* 2 (is-violated break))
                           (is-violated reviewing)) )
```

Fonte: Gerevini e Long (2005).

A Figura 2.3 mostra a comparação final de três planos executados com valores que atendem as preferências de forma diferente. Neste caso quanto menor o valor obtido, o plano possui maior qualidade, pois o fator definido é multiplicado pela quantidade de preferências violadas.

Figura 2.3 - Exemplo comparação de planos PDDL3.

Plan	Quality
π_1	5*4 + 4*1 + 2*1 + 1 = 27
π_2	5*8 + 4*0 + 2*0 + 0 = 40
π_3	5*6 + 4*1 + 2*0 + 0 = 34

Fonte: Gerevini e Long (2005).

Ambas as extensões da linguagem são motivadas pelo desejo de ver o planejamento preencher a lacuna entre pesquisa e aplicação. Muitos problemas reais exigem a especificação de objetivos mais complexos do que facilmente expressos em versões anteriores do PDDL. Isso inclui restrições nos estados (ou estado inválido) que um plano visita, bem como no estado em que é concluído. Também pode ser importante especificar os benefícios relativos de diferentes condições desejáveis, talvez conflitantes, que um plano deve satisfazer, para que um plano possa ser construído para avaliar esses benefícios em relação aos custos de alcançá-los.

3 TRABALHOS CORRELATOS

Neste capítulo são apresentados os trabalhos correlatos em abordagens de planejamento e abordagens de planejamento aplicadas à área espacial. Na Seção 3.1 as abordagens de planejamento com preferências e aprendizagem em planejamento são apresentadas. Na Seção 3.2 são apresentadas diferentes aplicações em planejamento automático para a área espacial. Na Seção 3.3 é apresentada uma discussão sobre as limitações e técnicas usadas nas abordagens.

3.1 Abordagens em planejamento

Existem diferentes métodos de planejamento automatizado encontrados na literatura. Em (NEBEL, 2000), (BRIEL et al., 2004), (GEREVINI; LONG, 2006), (RINTANEN et al., 2007), (BAIER et al., 2008) os autores incluem preferências de planejamento. Outros autores como (COLES; SMITH, 2004), (DOMSHLAK et al., 2010), (ALHOSSAINI; BECK, 2013), (ARECES et al., 2014) usam técnicas de aprendizagem. No entanto, eles não implementam uma solução de previsão de estado inválida.

3.1.1 Planejamento com preferências

Na literatura encontram-se diferentes trabalhos sobre planejamento em IA que propõem resolver problemas de planejamento mais complexos. Algumas das questões abordadas são as seguintes: geração de planos de qualidade; problemas de planejamento complexos; planejamento com incerteza; preferências do usuário e aprendizagem de preferências. A maioria desses trabalhos estão relacionados com o uso de preferências, um tópico interdisciplinar e que tem sido bastante estudado nos últimos anos, desenvolvendo diferentes técnicas e formas de planejamento (DOMSHLAK et al., 2011).

Para melhorar a quantidade dos critérios expressados em planos complexos, a linguagem PDDL foi estendida para uma linguagem de PBP na versão PDDL3. A HTN (Rede de Tarefas Hierárquicas) foi utilizada para aumentar o poder expressivo sobre a especificação de qualidade do plano. (JORGE et al., 2008). Dois novos recursos importantes foram incluídos permitindo que o usuário expresse restrições fortes e suaves sobre a estrutura dos planos desejados, bem como objetivos de problemas fortes e suaves. A primeira é a capacidade de expressar objetivos que se aplicam não apenas ao estado final da trajetória dos estados visitados por um plano, mas também aos estados intermediários. Esses objetivos assumem a forma de restrições de trajetória, familiares no trabalho de lógica temporal. A segunda extensão é a capacidade

de expressar restrições ou leves preferências. Embora os autores deixem claro que as extensões são úteis para expressar restrições que governam a qualidade do plano, em vez de controlar o conhecimento em si (GEREVINI; LONG, 2005; GEREVINI; LONG, 2006).

O PBP tem o objetivo de encontrar planos que atendam o maior número de preferências em uma instância de planejamento. Critérios são fornecidos para determinar quando um plano possui qualidade superir a outro. A busca pela construção de planos ideais é um tema abordado em (BOUTILIER et al., 1999), que examina vários tipos de representações para problemas de planejamento. O planejamento teórico de decisão usa o MDP (Processo de Decisão de Markov) para explorar a construção de políticas ótimas para expressar problemas de planejamento. O MDP associa uma função de recompensa a cada transição de estados, definindo dessa forma as preferências dos usuários. Todos os estados possíveis são classificados quantitativamente, onde a política ótima escolhida retorna uma ação considerando o histórico de ações executadas pelo agente. (BOUTILIER et al., 1999). Nesta abordagem obstáculos podem ser encontrados quanto a combinação de métodos que explorem diferenças de estruturas ao mesmo tempo, tornando-se um desafio fazer integração ou desenvolver ferramentas adicionais (BOUTILIER et al., 1999).

Outra abordagem baseada em técnicas de heurísticas fora criada através da preocupação com qualidade na geração de planos. A PSP (Planejamento de Satisfação Parcial) oferece recursos para resolver problemas parcialmente, podendo atender um subconjunto de objetivos (BRIEL et al., 2004). Nesse tipo de abordagem, o tempo para encontrar as melhores soluções pode ser muito alto, pois a busca por melhores planos acaba gerando um número muito maior de nós de pesquisa. Outra questão muito importante é sobre a utilização da função de avaliação usada para podar estados não promissores, que pode inevitavelmente perder um plano ótimo quando removido um estado do espaço de pesquisa (BRIEL et al., 2004).

Existem trabalhos que realizam a compilação da instância de planejamento criando um método de procedimento de controle representada em PDDL. (BAIER et al., 2008) permitiu representar no domínio de planejamento o procedimento como um autômato de estados finitos, adicionando-se um predicado para modificação de efeitos e condições prévias das ações, permitindo que o procedimento seja cumprido (BAIER et al., 2008). Embora seja uma abordagem geral, a maior vantagem desta técnica é a possibilidade de lidar com uma variedade de domínios e linguagens de especificação de controle (BAIER et al., 2008).

3.1.2 Remodelagem de domínio e aprendizagem em planejamento

Para aprimorar a eficiência dos planejadores e fazê-los solucionar um número maior de problemas, foram criadas abordagens modulares. A utilização da remodelagem de domínios para identificar ações e operadores irrelevantes que podem ser removidos ou substituídos por outras ações do plano tem contribuído para a melhoria de domínios. O planejamento com conjuntos de operadores reduzidos realiza uma etapa de pré-processamento para remover ações que são geradas por uma sequência de outras ações que não a contém (HASLUM; JONSSON, 2000). Este processo reduz o número de estados explorados, melhorando o tempo de busca. Outra técnica aplicada ao planejamento para remodelagem de domínio é a utilização de macro operadores, que atuam como atalhos para estados mais profundos nas ramificações do espaço de busca do planejamento. Por exemplo, (BOTEA et al., 2005) adiciona operadores macro encontrados ao domínio, incrementando-o para solucionar as instâncias de problema como um novo operador normal.

A aprendizagem de máquina tem sido usada em diferentes contextos para aprimorar o processo de planejamento. (COLES; SMITH, 2004) propõe o planejador Marvin, que evoluiu o planejador FF (Fast Forward) aplicando aprendizagem de macros. No processo de busca de planos, (DOMSHLAK et al., 2010) cria um preditor para escolher a melhor heurística. (ALHOSSAINI; BECK, 2013) comprovou que a remodelagem específica da instância usando aprendizagem pode melhorar o desempenho da geração de planos e há também a implementação de um processo de divisão automática de ações, para decompor os operadores mais complexos em mais simples, usando o tipo de aprendizado específico de domínio proposto por (ARECES et al., 2014).

Embora as macros funcionem como atalhos no espaço de pesquisa, elas aumentam o fator de ramificação e, potencialmente, tornam alguns operadores redundantes. Remover os operadores redundantes ou irrelevantes, por outro lado, diminui o fator de ramificação, mas é difícil encontrar operadores que sejam seguros para remoção de todas as instâncias em um domínio. Portanto, segundo estudo comparativo apresentado por Alhossaini e Beck (2013) a combinação entre a adição de macro e remoção de operador é uma direção valiosa que pode alcançar uma aceleração significativa em planejamento (ALHOSSAINI; BECK, 2013).

Uma oportunidade de se aplicar a mineração de dados é a produção de regras ou mapeamentos de regras para medidas de qualidade do plano (FRANK, 2007). Em discussão apresentada por Frank (2007), a mineração de dados pode ser usada para melhorar a qualidade do planejamento. Um validador pode ser visto como uma

função para acelerar o planejador ou uma abordagem semelhante à aprendizagem de *"nogoods"*, podendo aprender sobre restrições no espaço de solução que levam a um retrocesso precoce, o planejador adquire e minimiza os itens inúteis para solução dos problemas (FRANK, 2007).

3.2 Planejamento na área espacial

Nesta seção são apresentadas diferentes abordagens de planejamento automático que envolvem a geração de planos para a operação de voos em satélites.

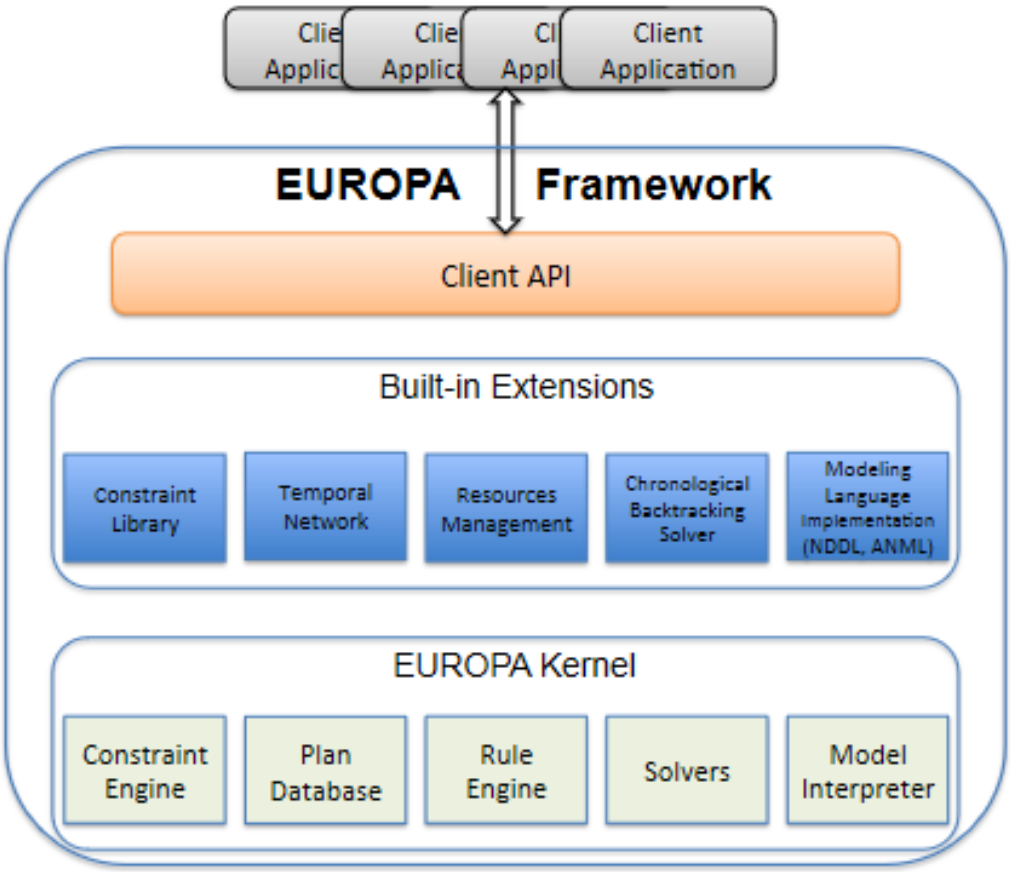
3.2.1 Arquitetura de planejamento de operações remoto

A *Extensible Universal Remote Operations Planning Architecture* (EUROPA) é um conjunto de ferramentas com o objetivo criar um sistema de planejamento, que possa ser facilmente integrável a outros sistemas para encapsular a lógica de planejamento e escalonamento. Usada apenas nos planejadores que implementam o paradigma de Planejamento Temporal e baseado em restrição. O planejamento temporal e escalonamento se baseia em um conceito claro de tempo e com a fórmula baseada nas restrições dos problemas de planejamento (FRANK; JÓNSSON, 2003).

Segundo Barreiro et al. (2012) os principais pontos fortes do EUROPA como ferramenta de Engenharia do Conhecimento de planejamento e escalonamento é possuir componentes de suporte aos processos de modelagem e análise de planos (BARREIRO et al., 2012). O EUROPA tem sido utilizado para uma variedade de missões, investigação orientada para a missão e demonstrações em aplicações da *National Aeronautics and Space Administration* (NASA), incluindo:

- Planejamento de atividades táticas MER: EUROPA é a tecnologia de planejamento central por trás do MAPGEN, uma ferramenta de suporte à decisão para gerar planos de atividades detalhados diariamente para a missão robótica MER a Marte.
- Planejamento a bordo e execução do plano. EUROPA foi a tecnologia de planejamento central para o planejamento deliberativo e reativo a bordo de uma variedade de robôs móveis.
- Missão MSL: Suporte para planejamento e programação para as Operações Científicas do Laboratório da Mars Science
- Projeto de pesquisa de planejamento de tripulação sobre planejamento e programação de missões espaciais

Figura 3.1 - Arquitetura EUROPA.



Fonte: Barreiro et al. (2012).

A Figura 3.1 mostra os principais componentes da arquitetura EUROPA, que são organizados de acordo com suas funções estruturais relativamente estáticas (determinadas por suas responsabilidades). A camada mais profunda na Figura 3.1 representa o kernel do EUROPA, que oferece recursos fundamentais de representação, raciocínio, modelagem e pesquisa que podem ser configurados, estendidos e personalizados para criar aplicativos de planejamento e programação (BARREIRO et al., 2012).

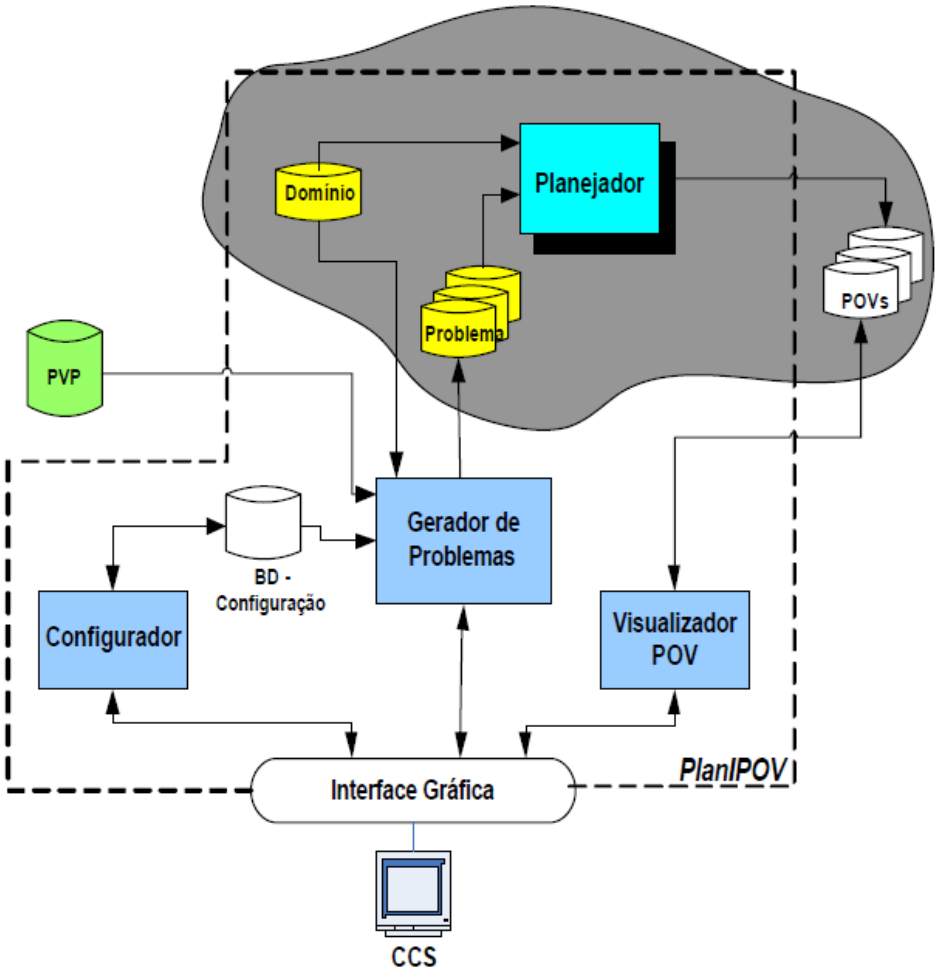
A arquitetura EUROPA utiliza-se de uma linguagem específica para a descrição de domínios e problemas de planejamento, a New Domain Definition Language (NDDL) que também permite incluir a descrição dos estados. Embora seja sistema de geração de planos que usa linguagem de representação que suporta a modelagem restrita de

mudanças contínuas e tempo métrico, ela é restrita com relação a semântica, sendo necessário a escrita de algoritmos de busca específicos para construir planos.

3.2.2 Agentes de planejamento em operações de satélites

No trabalho de Cardoso (2006), foi desenvolvida uma arquitetura para a aplicação da tecnologia de agentes de planejamento em operações de satélites, como mostra a Figura 3.2.

Figura 3.2 - Arquitetura do sistema PlanIPOV.



Fonte: Cardoso et al. (2006).

A arquitetura proposta tem o objetivo de aplicar a tecnologia de planejamento da IA na geração de Planos de Operação de Voo (POVs). O sistema *PlanIPOV* utiliza

a linguagem PDDL na modelagem da base de conhecimento do domínio de rastreio de satélites e no módulo Gerador de Problemas, responsável por gerar arquivos com os problemas de planejamento escritos em PDDL.

A abordagem *PlanIPOV* se apresenta como uma solução viável para o planejamento automático de planos, porém possui uma grande dependência do planejador utilizado com o sistema gerador de problemas. As interfaces definidas para comunicação entre esses módulos da arquitetura não possibilitam a substituição do software planejador.

Outra limitação presente nesta arquitetura é a necessidade de que as alterações ocorridas na representação de domínio também sejam refletidas no código. Isto se dá pelo fato da codificação implementada no gerador de problemas ser totalmente dependente do arquivo de domínio.

3.2.3 Replanejamento automatizado a bordo de satélites

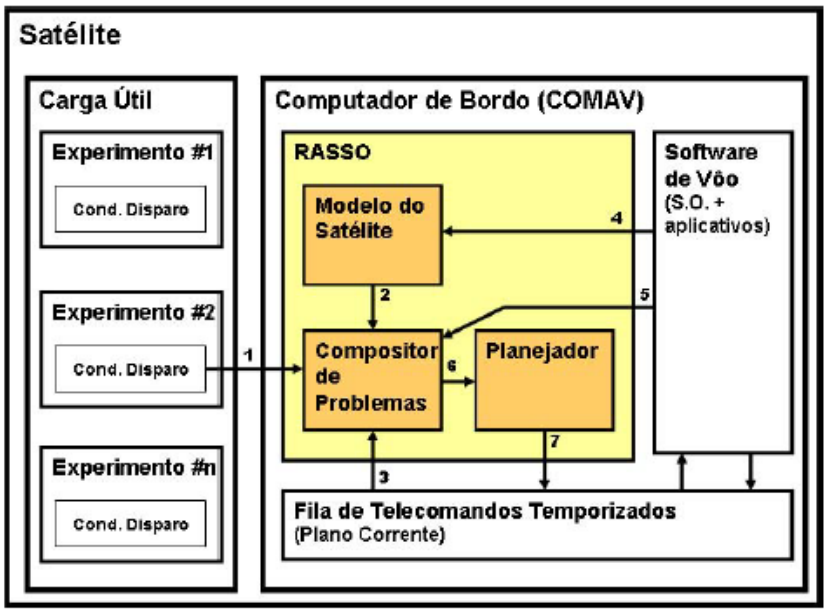
No trabalho de Kucinskis et al. (2007), foi desenvolvida uma arquitetura de alocação dinâmica de recursos computacionais para experimentos científicos com replanejamento automatizado a bordo de satélites, como mostra a Figura 3.3. *Allocation Service for Scientific Opportunities* (RASSO)

O planejador embarcado deve considerar a conversão do modelo antes de compilar o aplicativo ou adicionar o analisador ao software do satélite. Tendo em vista que um dos objetivos da RASSO é buscar a integração entre o programa embarcado e o restante do software do satélite, soluções mais complexas devem ser evitadas, como a conversão de modelos antes de compilar o software (KUCINSKIS et al., 2007).

Há algumas limitações que dificultam o uso em domínios complexos como da área espacial, dentre as quais se destacam:

- Falta de representação de recursos e consumo;
- A forma limitada de descrição de pré-condições e efeitos – não há estruturas de loops ou seleções de casos, e nem todas as linguagens baseadas em predicados possuem condicionais (ifs);
- O fato de que modelos descritos nestas linguagens devem ser interpretados por um analisador (parser) e convertidos em estruturas de dados, antes de serem utilizados pelo planejador.

Figura 3.3 - A Arquitetura do RASSO.



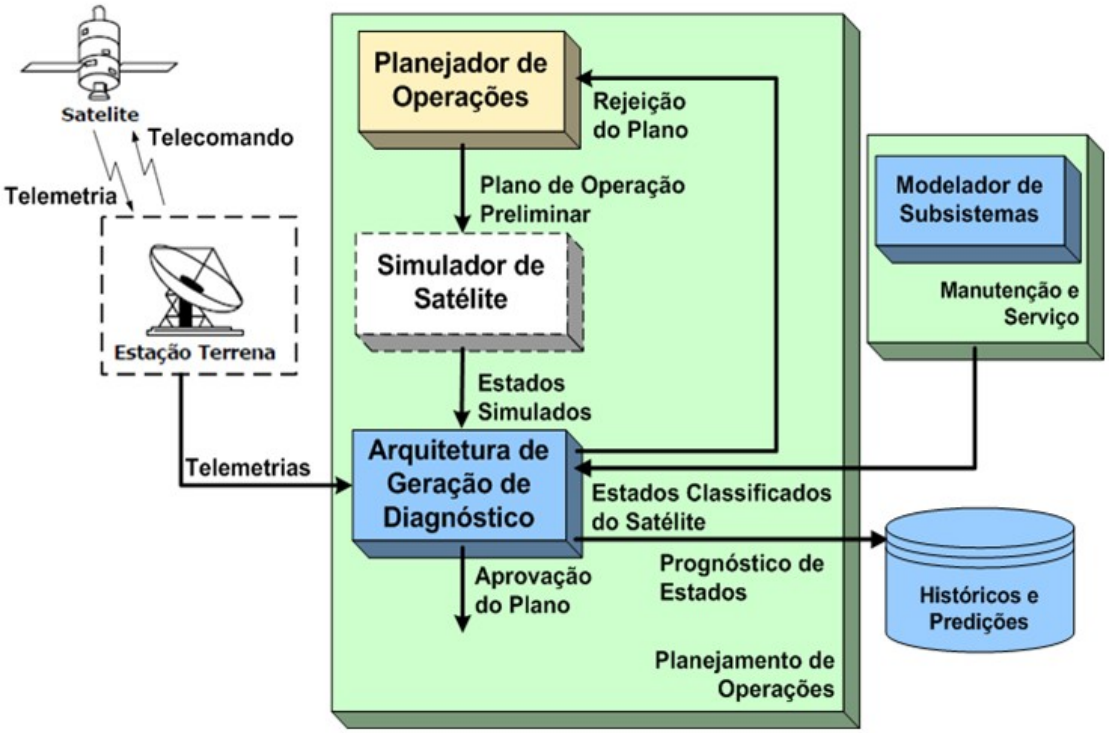
Fonte: Kucinskis et al. (2007).

3.2.4 Gerador de diagnósticos

No trabalho de Souza (2012), foi desenvolvida uma estratégia para validar plano de operação de voo a partir de predições de estados dos satélites do INPE, como mostra a arquitetura da Figura 3.4.

No contexto da operação de satélites, onde a degradação consequente do uso é um fator problema que causa mudança nos estados de um domínio, impedir que algumas etapas sejam geradas no plano é um desafio que esta arquitetura tenta solucionar. O gerador de diagnóstico recebe o estado de operação do satélite simulado executado pelas ações planejadas geradas pelo planejador da operação, diagnostica e prediz o estado do satélite por meio de classificação de parâmetros e telemetria simulada, e indica o estado do satélite. A evolução do estado do satélite depende das ações planejadas e da aprovação ou rejeição do plano proposto. A rejeição de um plano implica a geração de um novo plano, com etapas diferentes (SOUZA, 2012).

Figura 3.4 - Arquitetura de validação do plano de operação de voo.



Fonte: Souza (2012).

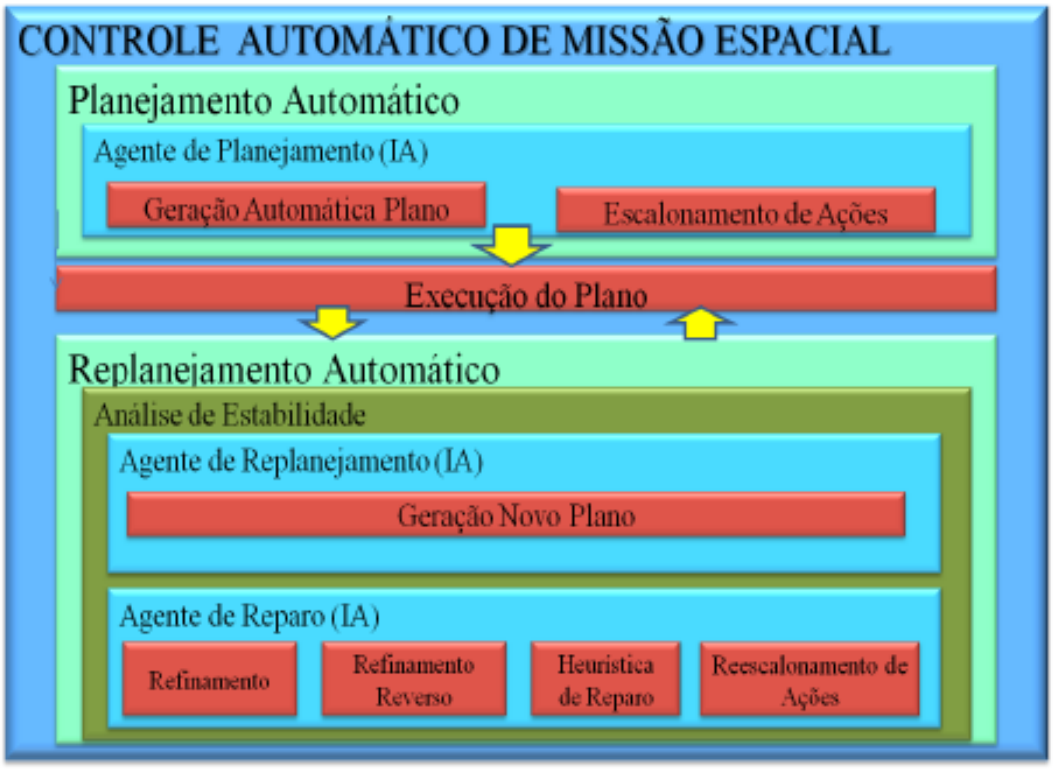
A utilização do módulo simulador de satélite na arquitetura implica que um longo trabalho de análise seja realizado para estabelecer a modelagem para a simulação do satélite, tornando-se uma limitação. Outra limitação também relacionada à necessidade de alteração manual da base de dados, utilizada no componente gerador de diagnóstico, é causada por não haver um mecanismo que permita importar o modelo de um satélite e que a base de dados seja gerada automaticamente.

3.2.5 Agentes de replanejamento

No trabalho de Ribeiro (2015), foi desenvolvida uma arquitetura de agente de replanejamento em tempo real para plano de voo em missões espaciais, como mostra a Figura 3.5. O modelo de replanejamento trata o modelo geral como dois módulos, um para as fases do preparo e a outro para o reparo do plano, que não se integram dinamicamente, ou seja, o planejamento será realizado off-line, isto é fora de passagem e o replanejamento on-line, ou seja, durante uma passagem.

Nesta arquitetura há dificuldade de representar todas as combinações possíveis de conjuntos de falhas e medidas corretivas, sem que haja decisões humanas. Outra limitação é encontrada no processo de reparo do plano, pois em tempo real o processo de reparo não dispõe de tempo suficiente, para preparar as representação dos problemas e gerar o novo plano. (RIBEIRO, 2015).

Figura 3.5 - Arquitetura geral do agente de planejamento e replanejamento.



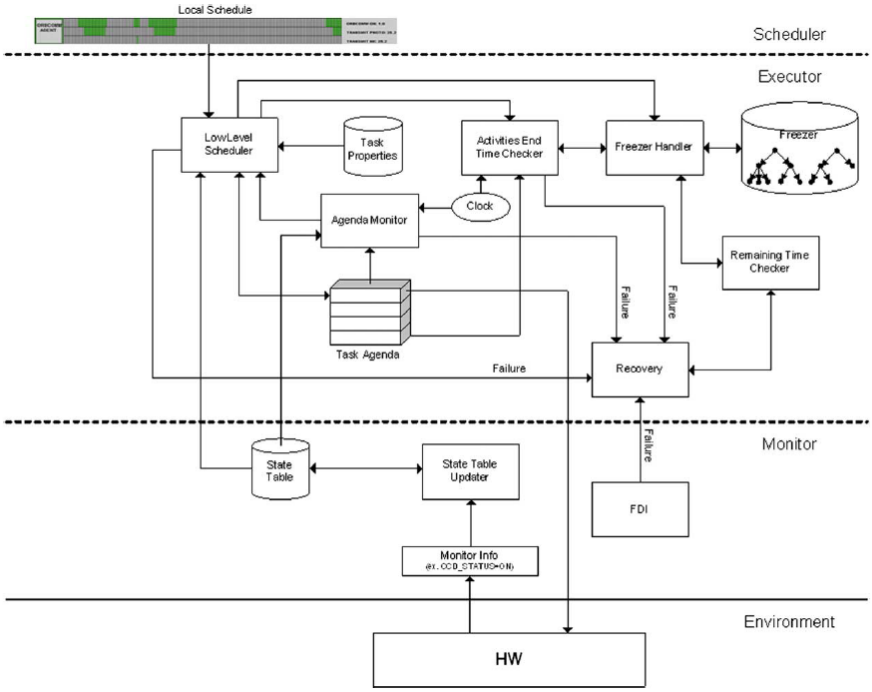
Fonte: Ribeiro (2015).

3.2.6 Planejamento distribuído com multiagentes

O planejamento distribuído por meio de uma arquitetura multiagentes é investigado utilizando planejamento com Rede de Tarefas Hierárquicas (HTN). No estudo de caso voltado a satélites de sensoriamento remoto, onde as tarefas relacionadas ao planejamento de operação são consideradas fundamentais para alcançar a maior autonomia de veículos espaciais (AMIGONI et al., 2010).

A Figura 3.6 mostra uma arquitetura multiagente destinada a funcionar a bordo de um satélite espacial, que desempenha o gerenciamento autônomo de atividades, com o agente associado a um subsistema do satélite *Palamede* para gerenciar as atividades. Cada agente é dividido em três módulos ou camadas: planejador, executor e monitor. Essas camadas juntas constituem um planejador distribuído, um planejador e um programa distribuído entre os agentes (AMIGONI et al., 2010).

Figura 3.6 - Arquitetura multiagente do satélite *Palamede*.



Fonte: Amigoni et al. (2010).

A arquitetura multiagente, possibilita explorar a robustez, facilidade de reutilização de agentes e concentrar em apenas um pequeno número de problemas por vez (AMIGONI et al., 2010). Ao provar que o desempenho do escalonador distribuído é equivalente ao do escalonador centralizado, a verificação experimental do sistema multiagente mostrou-se satisfatória e com a modularidade fornecida pela arquitetura multiagente. No entanto, as possíveis vantagens de usar vários agentes para gerenciar atividades em um único sistema espacial são amplamente inexploradas (AMIGONI et al., 2010).

Como limitação constatada neste tipo de abordagem estão as diferenças semânticas

dos modelos que representam o mesmo sistema, possibilitando erros de projeto, que além de dificultar os testes, também são difíceis de detectar. Isso abre a possibilidade de falha entre os comportamentos previstos por cada modelo (AMIGONI et al., 2010). O planejamento distribuído objetiva melhorar o desempenho e reduzir o espaço de busca, com atividades ocorrendo em paralelo (AMIGONI et al., 2010). No entanto, isso traz maior complexidade para coordenar as tarefas de planejamento e resolver quaisquer conflitos no plano.

3.3 Comparativo e discussão

Na literatura, encontramos diferentes abordagens de planejamento, que foram propostas para solucionar problemas distintos. A disparidade de uma abordagem para outra está no uso de técnicas diferentes, como, por exemplo, a inclusão de preferências, remodelagem de domínios, utilização de macros, utilização de aprendizagem de máquina, entre outros recursos que foram incorporados ao planejamento. A Tabela 3.1 mostra uma comparação do diferencial entre as abordagens de planejamento apresentadas na Seção 3.1, considerando os principais recursos utilizados por elas. Entende-se que cada abordagem inclui recursos ao planejamento de forma diferente e para solucionar problemas específicos.

Tabela 3.1 - Tabela comparativa das abordagens com preferências

Abordagem	**Técnica**	**Modelagem de Preferências**
PDDL3	Rede Hierárquica de Tarefas	Métrica de preferências violadas
MDP	Função de recompensa	Classificação baseada no histórico de ações executadas
PSP	Heurísticas	Planejamento com subconjunto de objetivos
PCK	Lógica Linear Temporal e Autômatos	Predicado adicional nos efeitos das ações

A Tabela 3.2 apresenta uma comparação entre as abordagens propostas para o planejamento automático na área espacial, onde são comparadas suas limitações e discutidas sugestões. Nota-se que poucas das abordagens referenciadas fazem uso de técnicas de planejamento com preferências, seja para adicionar restrições ou validar os planos gerados automaticamente. Apenas a abordagem proposta por Amigoni et al. (2010) na Seção (3.2.6), propôs a utilização de HTN para um planejamento distribuído.

Tabela 3.2 - Tabela comparativa das arquiteturas

Arquitetura	**Diferença e Limitações**	**Comparativo e discussão**
Arquitetura de planejamento de operações remoto (FRANK; JÓNSSON, 2003) Seção (3.2.1)	A principal diferença é a maior flexibilidade na integração com os aplicativos do cliente, embora limita-se na utilização de uma linguagem específica exigindo planejadores apropriados	A limitação presente nesta arquitetura exige que algoritmos específicos de busca sejam implementados. Esta limitação poderia ser minimizada se fosse utilizada uma extensão de uma linguagem mais utilizada entre os planejadores
Arquitetura do sistema PlanI-POV (CARDOSO et al., 2006) Seção (3.2.2)	A principal diferença é fornecer uma interface gráfica para configuração de problemas, embora não permita a atualização de mudanças de domínio e substituição do planejador	As mudanças do ambiente em cada passagem do satélite representam uma grande limitação para o Gerador de Problemas. Falta uma representação genérica do domínio e problema de planejamento para que o acoplamento entre os módulos da arquitetura seja minimizado
Arquitetura RASSO (KUCINSKIS et al., 2007) Seção (3.2.3)	A principal diferença é disponibilizar o replanejamento a bordo, embora haja falta de recursos de hardware para suportar a solução de planejamento	O replanejamento embarcado pode exigir que o modelo de planejamento seja convertido antes de ser compilado e a conversão pode ter um alto custo de processamento. Exige que uma representação de recursos e consumo seja desenvolvida
Arquitetura de validação do plano de operação de voo (SOUZA, 2012) Seção (3.2.4)	A principal diferença é predizer o estado de operação permitindo aprovar ou rejeitar planos, embora utiliza grande esforço e tempo para criação do processo de planejamento com validação de planos feita em um processo à parte	Atrelar a geração da base de conhecimento a um simulador de satélites causa aumento do trabalho e do tempo total do processo de planejamento, por validar os planos após o planejador de operações gerar o plano. O gerador de diagnósticos poderia fazer parte do processo de planejamento e não ser um processo à parte. A alteração manual da base supervisionada também impacta no desempenho da solução, pois poderia ser minimizado a partir da utilização de uma representação genérica para importar novos modelos de dados

Arquitetura	Diferença e Limitações	Comparativo e discussão
Arquitetura agente de planejamento e replanejamento (RIBEIRO, 2015) Seção (3.2.5)	A principal diferença é a execução do planejamento em tempo real. Decisões humanas são necessárias para a representação de combinações de medidas para as falhas	O replanejamento neste caso que exige ser realizado on-line durante a passagem do satélite, se mostra muito desafiador quando o processo de reparo de plano for necessário, devido à falta de tempo hábil para realização do reparo. Para que esse problema seja minimizado, um processo de predição de falhas para auxiliar o processo de planejamento off-line seria um caminho viável
Arquitetura multiagente (AMIGONI et al., 2010) Seção (3.2.6)	A principal diferença é permitir explorar a combinação de multiplos agentes, embora tenha um aumento da complexidade das tarefas de planejamento e resolução de conflitos no plano	O uso de multiagentes aumenta a possibilidade de falhas, devido a utilização de semânticas diferentes para modelos iguais, além aumentar a complexidade no caso de resolver conflitos em planos

Entre as abordagens comparadas, apenas a arquitetura de agentes de replanejamento (RIBEIRO, 2015) na Seção 3.2.5 e a Arquitetura de validação do plano de operação de voo (SOUZA, 2012) na Seção 3.2.4 apresentam soluções para uma melhoria da qualidade dos planos ou reparo. Na literatura, foi observado que, nos problemas de planejamento encontrados nas agências espaciais, é necessário verificar o plano antes de executá-lo, devido a linguagem de planejamento não poder capturar a descrição completa do domínio (FRANK, 2007).

Frank (2007) discute a oportunidade de aplicar aprendizagem de máquina e outras técnicas para gerar regras ou mapeamentos de regras para métricas de qualidade do plano, uma vez que o planejamento automático pode incorporar simulações realizadas durante o processo de verificação de planejamento no processo de geração de novos planos (FRANK, 2007).

4 ARQUITETURA DE PREDIÇÃO DE ESTADOS INVÁLIDOS

Nesta seção, descreve-se a contribuição deste trabalho por meio da apresentação de uma arquitetura para o planejamento automático de operação de voos em satélites com predição de estados inválidos.

4.1 Concepção

Dada a hipótese de que planos válidos podem ser gerados caso estados inválidos forem validados no planejador em tempo de planejamento, o planejador encontrará o estado final usando uma sequência de etapas que transita apenas por estados válidos do domínio, a concepção de uma arquitetura utilizando o método de validação contribuirá para evitar que planos sejam rejeitados.

O planejador aplica no estado inicial as ações definidas no domínio e cria uma árvore de estados enquanto as ações são aplicadas. No entanto, em tempo de planejamento, quando uma ação é aplicada no estado atual, o planejador sabe qual será o estado que será adicionado à árvore de estados, desta maneira o novo estado gerado pode ser validado para não compor o plano para atender o objetivo do problema de planejamento definido.

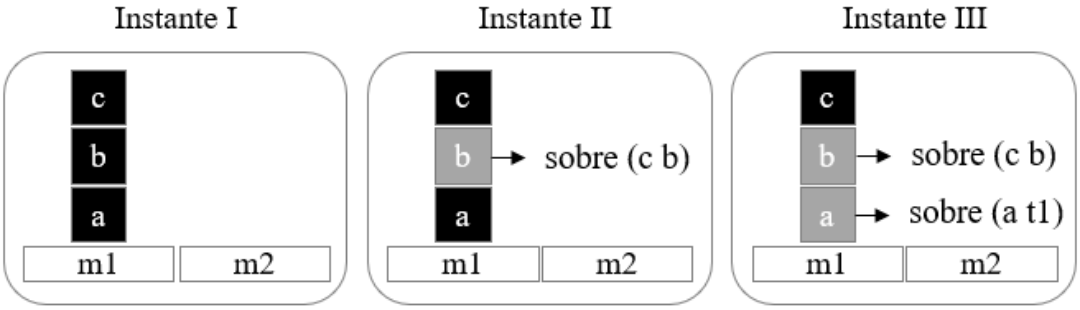
Um sistema de transição de estados não determinístico permite que um problema de planejamento seja resolvido através de diversas soluções, desta maneira, estados podem ser ignorados e ainda assim o estado objetivo ser alcançado. Assim, neste trabalho é proposto um processo de verificação de estados inválidos para que o planejador não considere estes estados na solução final.

4.2 Definição de estado inválido

Um estado inválido é formado por situações correntes de um cenário do domínio. Para ilustrar o entendimento da formulação dos estados inválidos, a Figura 4.1 mostra um exemplo do clássico problema de planejamento do mundo dos blocos, onde são apresentados três momentos (Instante I, II, III) diferentes usando o mesmo cenário do domínio, com três blocos empilhados sobre uma mesa. Neste exemplo, nota-se que, com a variação do tempo, os estados do domínio de planejamento sofreram alteração, pois surgiram determinadas situações inválidas com o passar do tempo.

O primeiro instante representa o cenário onde não há nenhuma situação inválida. No segundo instante uma determinada situação tornou-se inválida, supondo que o *bloco b* foi acometido por uma anomalia que o impede de sustentar o *bloco c*. Neste

Figura 4.1 - Ilustração de estado inválido.



Fonte: Produção do autor.

caso, dizemos que a situação *on (c b)* tornou-se inválida. No terceiro instante, outro estado inválido é formado por duas situações inválidas, supondo que o elemento *bloco a* não pode ser empilhado na mesa *mesa m1*, então a situação *on (a m1)* também se tornou inválida.

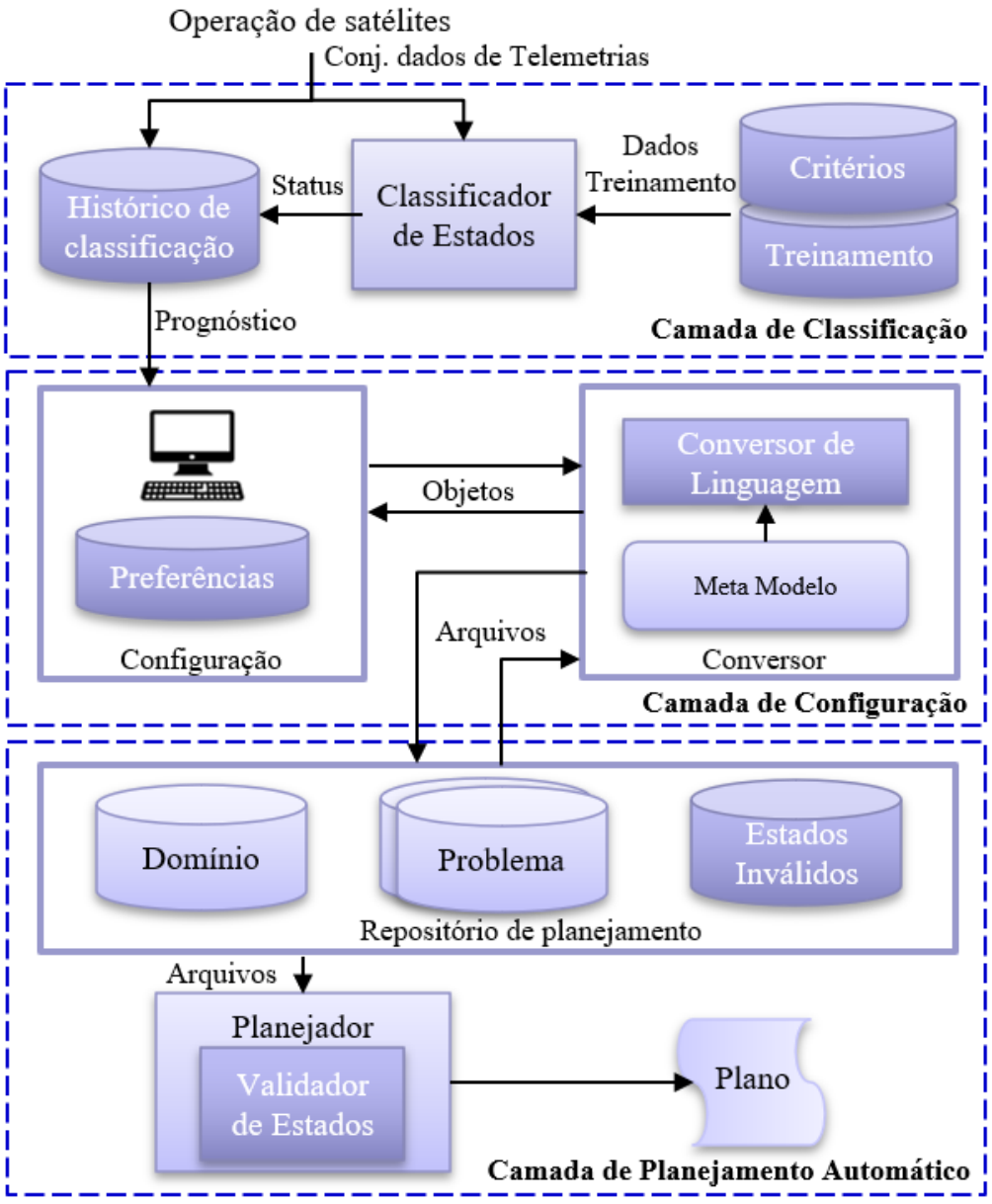
No cenário apresentado no instante III, as duas situações inválidas - um objeto empilhado sobre outro - são semelhantes, porém relacionam elementos com tipos diferentes. Na situação *on (c b)* há um relacionamento entre dois elementos do tipo bloco, já na situação *on (a m1)* um elemento é do tipo bloco e outro elemento é do tipo mesa. No entanto, para identificar qual ação do domínio gera a situação inválida, é necessário observar quais elementos são relacionados no efeito da ação. Por exemplo, neste domínio, a ação empilhar dois blocos tem um efeito com a definição *on (bloco1 bloco2)*, indicando que o efeito da aplicação da ação empilhar gera a situação *on (c b)*. A situação inválida *on (a m1)* foi gerada pela ação mover, que possui a definição *on (bloco mesa)* em seu efeito. Em resumo, o efeito de cada ação definida no domínio de planejamento é composto por um conjunto de situações. Os parâmetros de cada situação assumem os valores dos objetos descritos na definição do problema de planejamento, dessa forma podemos formular os estados inválidos.

4.3 Visão geral da arquitetura de predição de estados inválidos

Nesta seção, é apresentada na Figura 4.2 a arquitetura que consiste em agrupar os principais componentes que envolvem a classificação de estados de domínio e processos de planejamento automatizado. Os componentes foram agrupados em três camadas principais: a camada de classificação de estados, de configuração, e de

planejamento. A arquitetura proposta visa gerar planos de acordo com as mudanças e situações atuais do domínio atual, permitindo a previsão de estados inválidos. A partir da configuração de preferências incluídas na camada de configuração por um especialista do domínio, é possível predizer os estados inválidos no processo de planejamento automatizado. Nas próximas seções, será explicada cada uma das três camadas.

Figura 4.2 - Visão geral da arquitetura de predição de estados inválidos.



Fonte: Produção do autor.

4.3.1 Camada de classificação

A camada de classificação encapsula o processo de aprendizagem do domínio de planejamento. Os dados de entrada desta camada são obtidos por meio da operação de rotina e o conhecimento de especialistas no domínio. Os critérios de classificação são condições, intervalos de valores ou a combinação de informações que definem todas as possíveis classes, no qual o domínio pode ser classificado. Como por exemplo, um critério para criar uma classe representando a operação *normal* de um domínio pode ser definido com a condição dos dados estarem dentro de um intervalo estabelecido. A formação desses critérios depende exclusivamente do domínio, pois é o conjunto de características que configuram cada classe assumida na operação.

O componente Classificador de Estados representa o modelo de aprendizagem, que utiliza os critérios para o treinamento dos algoritmos de classificação, armazenando os dados classificados no componente Histórico de Classificação, que fornece os prognósticos para a camada de configuração.

A partir de novas observações sobre os dados de telemetrias recebidos das operações de satélite, a camada de classificação irá prever o estado atual do domínio com base nos critérios de classificação. O status de classificação atual é usado na camada de configuração para obter o grupo de estados inválidos associados ao domínio.

4.3.2 Camada de configuração

A camada de configuração é a parte mais relevante e que tem o objetivo de integrar os dois processos fundamentais para a geração de planos válidos: processo de planejamento e processo de classificação. Para que um plano seja gerado respeitando as situações inválidas a partir do status do domínio apresentado no momento do planejamento, as mudanças ocorridas devem estar adequadamente representadas no domínio de planejamento. A validação dos estados inválidos na geração do plano é feita a partir da configuração de um conjunto de estados inválidos, correspondentes a cada um dos status que o domínio pode assumir ao longo do tempo e de acordo com o status classificado do domínio, um conjunto diferente de estados inválidos são validados. A associação dos estados inválidos com os status do domínio são armazenados na base de dados do componente de Configuração que foi nomeada como Preferências.

O Conversor é o principal componente da camada de configuração, sendo responsável por obter o domínio e o problema utilizados pela camada de planejamento

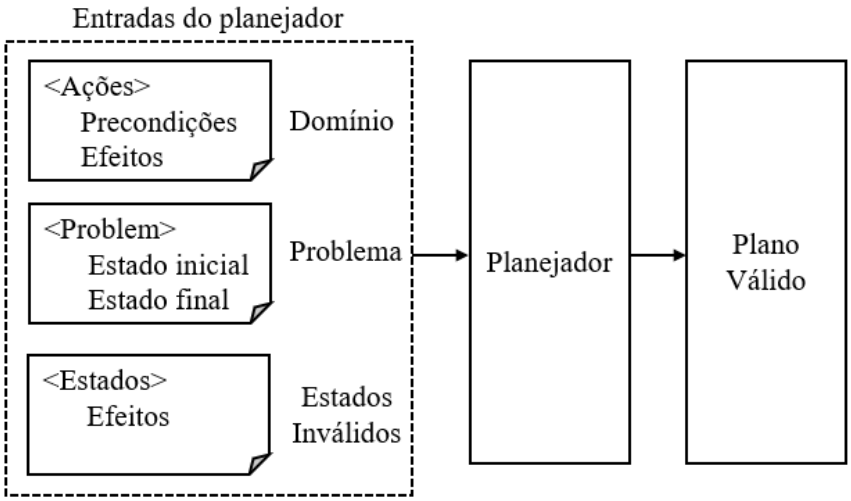
automático e realizar a conversão do domínio escrito na linguagem de planejamento, para uma representação orientada à objetos. O elemento responsável por fornecer a estrutura de representação de planejamento é o *KPlanOO* - Um Meta-modelo orientado a objetos para descrição de domínios de planejamento (SILVA et al., 2010), usado para garantir que a representação do conhecimento do domínio de planejamento seja corretamente mapeada entre as camadas da arquitetura.

O componente Conversor de Linguagem gera a representação dos estados inválidos que é entregue para a camada de planejamento automático, a partir da escolha dos estados inválidos e configuração realizada através da interface de configuração.

4.3.3 Camada de planejamento automatizado

A camada de planejamento automatizado representa o processo de planejamento com predição de estados inválidos. O componente Planejador representa um software planejador existente que foi alterado a partir da inclusão do componente Validador de Estados, que é responsável por permitir que os estados inválidos sejam validados em tempo de planejamento. O planejador, além de receber os arquivos de domínio e problema, recebe o arquivo de estados inválidos, que contém as definições dos estados que não poderão compor os planos (CRUZ et al., 2020).

Figura 4.3 - Representação da estratégia.



Fonte: Produção do autor.

Os estados inválidos definidos como uma nova entrada serão utilizados para validar os estados gerados no tempo de planejamento; consequentemente, quando um novo estado for gerado, ele será comparado à lista de estados inválidos e o planejador descartará o estado da solução do plano, como mostra a Figura 4.3.

Para formalizar a implementação da estratégia proposta, modificou-se o conceito de problema definido anteriormente como $P = (\Sigma, s_i, S_g)$, para utilizar a partir de então a função $V(s_k, I)$ responsável por validar e filtrar os estados em tempo de planejamento, onde s_k representa o estado atual e I representa a instância de estados inválidos previamente configurados como entrada para o planejador. A modificação transforma a função padrão da seguinte forma $P = (\Sigma, s_i, S_g, V(s_k, I))$.

Para descrever os estados inválidos e representá-los em memória, a mesma linguagem de planejamento utilizada pelo planejador foi reutilizada. O estado inválido é configurado como uma ação comum, usando as mesmas notações de um arquivo de domínio. A Listagem 4.1 apresenta o exemplo da definição dos estados inválidos ilustrados no Instante III da Figura 4.1. Neste exemplo, a ação *invalidState* define duas variáveis como parâmetros, e nas precondições associa-se o tipo a cada parâmetro, sendo dois blocos. O posicionamento e a ordem dos blocos empilhados são definidos nos efeitos, tal como: *(and (on ?c ?b)*.

Listagem 4.1 - Configuração dos estados inválidos

```
(define (domain invalid-state-settings)
  (:requirements :strips)
    (:action invalidState
     :parameters (?c ?b)
     :precondition (and (block ?c) (block ?b))
     :effect (and (on ?c ?b))
    )
)
```

A função que aplica as ações gerando novos estados filhos, implementada do planejador, foi adaptada para fazer a validação dos estados. A implementação da função *States Validator* no planejador, foi necessária para validar os estados durante o planejamento, após a geração de cada novo estado, a função é chamada para validar se o estado gerado corresponde a um estado inválido.

Listagem 4.2 - Função validador de estados

```
1  function StateValidator(newState, invalidStates)
2      var equalState
3      for each invalidStates.defs do
4          actions <- invalidStates.defs.effect
5          for each actions do
6              equalState <- false
7              for each newState.defs do
8                  if newState.defs.operation != not and
9                     newState.defs.action == actions.action and
10                    Equal(newState.defs.params, actions.params)
11                 then
12                     equalState <- true
13     return equalState;
14 End
```

A Listagem 4.2 mostra a definição do algoritmo *State Validator*. A função recebe como parâmetro o novo estado gerado e a lista de estados inválidos. A comparação dos estados é feita a partir de três condições: (1) se a *operation* não for negativa - pois uma situação com o operador negativo indica que a situação não existe naquele estado e por isso são comparadas apenas situações existentes); (2) se a *action* de ambas as partes for a mesma; (3) se os *parameters* das partes forem iguais. A função *Equal* compara os parâmetros dos estados, verificando se o tamanho e os valores são idênticos.

A validação é feita percorrendo e comparando todas as definições da lista *invalidStates*. Caso algum estado inválido tenha todas suas partes encontradas no estado novo, a função retorna *true* e o estado gerado é descartado do planejamento. O planejador continua gerando e validando outros estados, até encontrar o estado objetivo do problema.

A implementação e validação da arquitetura é apresentada no capítulo 5. A aplicação do processo de classificação de dados e configuração dos estados inválidos são apresentados por meio de um estudo de caso no capítulo 6.

4.4 Visão dinâmica da arquitetura de predição de estados inválidos

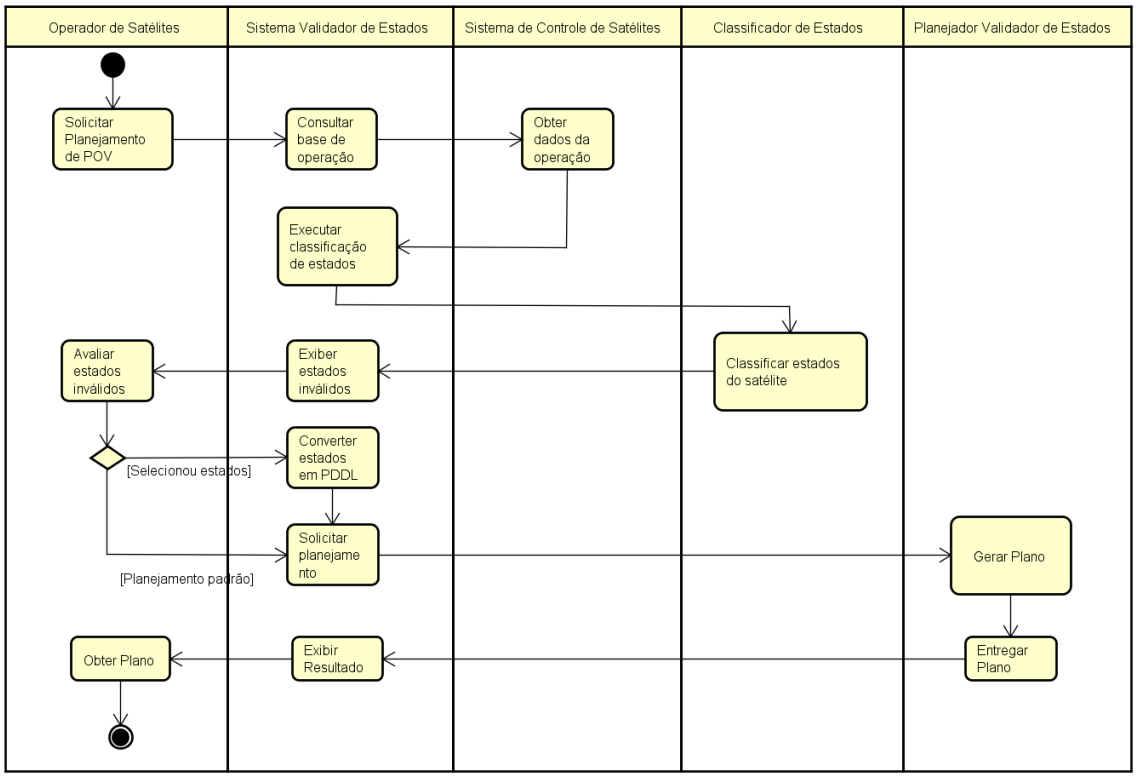
Nesta seção, é apresentada na Figura 4.4 a visão dinâmica da arquitetura de predição de estados inválidos. Por meio do diagrama de atividades, descreve-se a sistemática

do funcionamento da arquitetura.

A listagem a seguir descreve as cinco raias verticais apresentadas no diagrama de atividades:

- **Operador de Satélites** - representa um engenheiro especialista que realiza a operação de satélites;
- **Sistema Validador de Estados** - representa a camada de configuração da arquitetura de predição de estados inválidos;
- **Sistema de Controle de Satélites** - representa o sistema que fornece os dados de operação do satélite;
- **Classificador de Estados** - representa a camada de classificação;
- **Planejador Validador de Estados** - representa a camada de planejamento automático.

Figura 4.4 - Visão dinâmica da arquitetura de predição de estados inválidos.



Fonte: Produção do autor.

A sequência a seguir descreve o fluxo de execução do planejamento realizado a partir da arquitetura de predição de estados inválidos:

- O processo de planejamento automático se inicia por parte do operador de satélite, a partir de uma solicitação do planejamento para o sistema validador de estados por meio de uma interface;
- O sistema validador de estados, ao receber a solicitação de um novo planejamento, obtém os últimos dados de operação recebidos do sistema de controle de satélites e inicia uma nova atividade, solicitando a classificação dos dados de operação para o classificador de estados;
- O classificador de estados utiliza a base de treinamento classificada a partir dos critérios definidos e o modelo de predição para classificar o estado atual do domínio. Com base no prognóstico da classificação recebida, o sistema validador de estados exibe os estados inválidos por meio de uma interface

para o operador de satélites;

- O operador de satélites pode realizar a configuração a partir da visualização do estado atual do domínio e do conjunto de estados inválidos que foram associados, para selecionar os estados inválidos desejados ou manter o planejamento automático com a configuração padrão configurada. Caso sejam escolhidos estados inválidos, o conversor de estados inválidos converterá os estados na linguagem de representação utilizada no planejador. O sistema validador de estados realiza a solicitação do planejamento para o planejador validador de estados enviando a definição dos estados inválidos configurados pelo operador de satélites;

- O planejador validador de estados inválidos recebe a solicitação para gerar o plano e utiliza as definições de domínio, problema e estados inválidos informados pelo sistema validador de estados. O resultado do planejamento é a entrega de um plano para o sistema validador de estados, que apresenta o resultado para o operador de satélites, representando o final das atividades executadas no processo de planejamento da arquitetura de predição de estados inválidos.

4.5 Diferenciais da arquitetura de predição de estados inválidos

Nesta subseção será abordada uma comparação entre a arquitetura desta dissertação com as abordagens de planejamento apresentadas na seção de trabalhos correlatos da área espacial 3.2.

Neste trabalho, similarmente às abordagens com aprendizagem descritas na Seção 3.1, como, por exemplo, em PDDL3, que foi proposta a criação de uma extensão da linguagem PDDL; no MDP, que desenvolveu um processo de classificação baseado no histórico de execução; ou no PCK, que incluiu novas definições na representação de planejamento, a criação do conceito de estados inválidos, no qual considera-se uma forma de incluir preferências dentro do processo de planejamento. Modificando o processo de planejamento automatizado por meio da validação do plano, criando assim, um processo diferente dos trabalhos correlatos da área espacial apresentados e que contribui para gerar planos de maior qualidade. As principais diferenças são destacadas a seguir:

- Diferente das abordagens de arquitetura de planejamento de operações remoto 3.2.1 e planejamento distribuído 3.2.6, a arquitetura de predição de

estados inválidos propõe uma reutilização da linguagem utilizada no planejador para validar as restrições, sem que uma outra linguagem de representação diferente da usada no planejador seja necessária para representar as restrições;

- Diferente das abordagens de agentes de planejamento 3.2.2 e replanejamento a bordo de satélites 3.2.3, a arquitetura de predição de estados inválidos faz uso de um meta modelo orientado à objetos para representar os domínios e problemas, permitindo que os módulos da arquitetura sejam mais independentes e evitando fortes acoplamentos com a linguagem;

- Diferente da abordagem do gerador de diagnósticos 3.2.4, a arquitetura de predição de estados inválidos realiza um processo de aprendizagem para classificar o estado de operação antes do processo de planejamento e inclui a validação do plano ao planejador, minimizando assim, o tempo total do planejamento e gerando planos validados durante a etapa de planejamento;

- Diferente da abordagem de agentes de replanejamento 3.2.5, a arquitetura de predição de estados inválidos realiza um processo de predição de estados inválidos para corrigir o plano quando o planejador realiza o planejamento, evitando que o plano seja reparado posteriormente.

5 IMPLEMENTAÇÃO E VALIDAÇÃO

Este capítulo apresenta as implementações realizadas para aplicação da arquitetura e também as validações preliminares executadas com o software planejador.

5.1 Implementação da camada de planejamento automático

Nesta seção, apresenta-se a implementação do validador e os resultados obtidos com os testes da validação de estados em um planejador baseado em STRIPS. Os testes e validações da arquitetura foram executados utilizando inicialmente o domínio do mundo dos blocos (GUPTA et al., 1991) e o robô trabalhador portuário (GHALLAB et al., 2004), que são domínios didáticos de planejamento escolhidos para realizar as validações preliminares desta arquitetura.

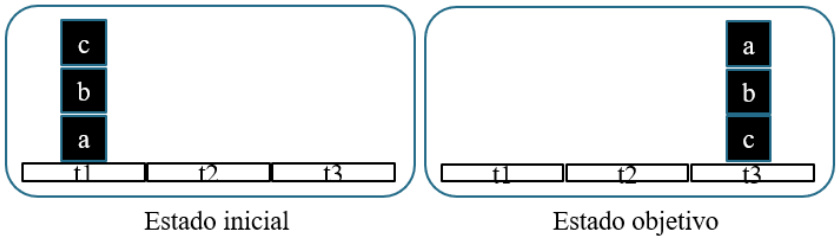
5.1.1 Escolha do problema e planejador

Para validar a proposta de criação do método validador de estados, escolheu-se o planejador automático AI Planning com STRIPS (BECKER, 2015). Este planejador é um projeto de demonstração para a biblioteca de planejadores automatizados do STRIPS, escrita em NodeJS (BECKER, 2015).

Inicialmente, o problema do mundo dos blocos foi utilizado para formular um exemplo de utilização da estratégia para validar os estados. Este problema é um problema clássico de planejamento, que consiste em empilhar blocos em cima de mesas em alguma ordem arbitrária (GUPTA et al., 1991). Algumas ações desse domínio são: mover um bloco de uma mesa para outra, empilhar dois blocos e desempilhar dois blocos. Após a implementação do método validador, outro problema de planejamento foi escolhido para validar um cenário com número maior de estados.

O problema do mundo dos blocos escolhido é composto por seis objetos, sendo três blocos e três mesas. O estado inicial tem o bloco *a*, *b* e *c*, onde o bloco *c* sobre o bloco *b*, bloco *b* sobre o bloco *a* e o bloco *a* empilhado sobre a mesa *t1* e outras duas mesas, *t2* e *t3* vazias. O estado objetivo é composto por empilhar os blocos de forma reversa sobre a mesa *t3*. Como ilustra a Figura 5.1.

Figura 5.1 - Problema de planejamento.

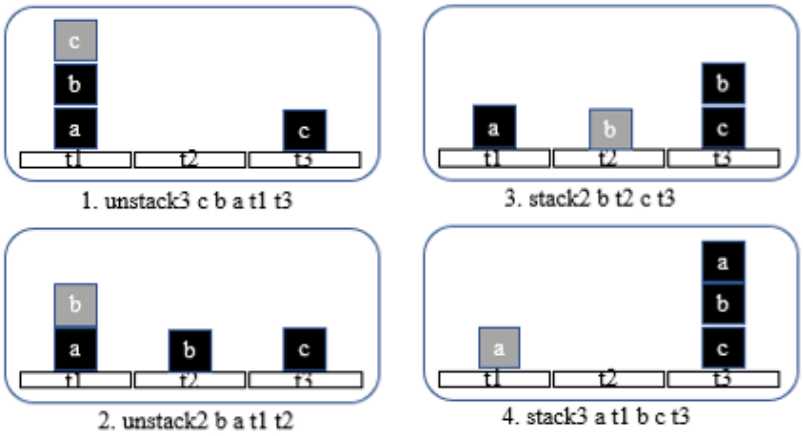


Fonte: Produção do autor.

O planejador deve encontrar uma sequência de ações capaz de transformar o estado inicial no estado objetivo usando apenas as ações definidas no domínio do problema, por exemplo: *move* - move um bloco de uma mesa para outra; *stack2* - empilha um bloco sobre outro bloco em outra mesa; *stack3* - empilha um bloco sobre outros dois blocos sobre outra mesa; *unstack2* - desempilha dois blocos, colocando o bloco não empilhado sobre outra mesa vazia; *unstack3* - desempilha três blocos, colocando o bloco não empilhado sobre outra mesa vazia.

Este problema foi executado no planejador e um plano foi obtido. O plano contém a seguinte transição de estados mostrada na Figura 5.2. Pode ser vista a sequência de ações do domínio que foram aplicadas aos estados até que o objetivo fosse alcançado. A primeira ação aplicada foi a *unstack3*, responsável por desempilhar três blocos, resultando em desempilhar o bloco *c* do bloco *b* para a mesa *t3*. Em seguida o bloco *b* é desempilhado para a mesa *t2*, efeito da ação *unstack2* responsável por desempilhar dois blocos. No terceiro estado, a ação *stack2* foi executada para empilhar o bloco *b* sobre o bloco *c*. Por fim, na a última ação, *stack3* é executada para empilhar o bloco *a* da mesa *t1* para o bloco *b*.

Figura 5.2 - Transição de estados.



Fonte: Produção do autor.

5.1.2 Geração do arquivo de estados inválidos

Após representar a execução do plano, escolheu-se aleatoriamente um dos estados percorridos para representar um estado inválido. O estado escolhido foi o gerado através da aplicação da ação *Unstack2*. O estado escolhido foi configurado em um novo arquivo escrito em PDDL como mostra a Listagem 5.1.

Listagem 5.1 - Arquivo de estado inválido escrito em PDDL

```
1  (define (domain invalidstates)
2      (:requirements :strips)
3      (:action invalid1
4          :parameters (?a ?b ?c ?t1 ?t2 ?t3)
5          :precondition (and (block ?a) (block ?b)
6                  (block ?c) (table ?t1) (table ?t2)
7                  (table ?t3))
8          :effect (and (on ?a ?t1) (clear ?a)
9                  (on ?b ?t2) (clear ?b) (on ?c ?t3)
10                 (clear ?c))
11     )
12 )
```

O estado foi configurado como uma ação comum, usando as mesmas definições do arquivo de domínio em PDDL. No *invalid1* (Listagem 5.1, linha 3), define-se os

parâmetros como variáveis, sendo: três blocos e três mesas. Nas precondições associa-se o tipo a cada parâmetro de entrada. O posicionamento e a ordem dos blocos empilhados é definido nos efeitos, tal como: bloco *a* sobre a mesa *t1 - (on ?a ?t1)*, bloco *a* disponível *(clear a)*, bloco *b* sobre a mesa *t2- (on ?b ?t2)*, bloco *b* disponível *(clear b)*, bloco *c* sobre a mesa *t3 - (on ?c ?t3)* e bloco *c* disponível *(clear c)*.

5.1.3 Implementação do validador de estados inválidos

O planejador possuí o método de leitura do arquivo em PDDL, que recebe o caminho do arquivo e monta o objeto em memória. Utilizou-se a implementação existente no planejador para ler o arquivo de estados inválidos. O método *StripsManager.loadDomain()* foi reutilizado para carregar o arquivo de estados inválidos. A (Listagem 5.2, linha 4) mostra a alteração feita no método que carrega os arquivos PDDL no planejador, onde o parâmetro *invalidStatePath* foi adicionado ao método *load* que passa a receber três parâmetros, o *domainPath*, *problemPath* e *invalidStatePath*.

Listagem 5.2 - Método de carregamento do arquivo

```
1 load: function(domainPath, problemPath,
2          invalidStatePath, indice, callback, isCode) {
3
4     StripsManager.loadDomain(invalidStatePath,
5                                 function(invalidState)) {}
6
7 }
```

Após carregar o estado inválido em memória, o objeto contendo as definições dos estados inválidos foi adicionado à uma nova propriedade *invalidState* criada no objeto *domain*. O objeto domain é o principal parâmetro utilizado pelos métodos do planejador, pois contém a lista de ações usadas no planejamento.

O trecho de código onde o planejador aplica as ações, gerando novos estados, é apresentado na Listagem 5.3. O método *getChildStates* tem a função de aplicar as ações do domínio no estado atual do planejamento, gerando novos estados filhos. O método recebe como parâmetro a definição de domínio e o estado atual do planejamento.

Listagem 5.3 - Método de carregamento do arquivo

```
1  getChildStates: function(domain, state) {
2      var children = [];
3      var actions =
4          StripsManager.applicableActions(domain, state);
5      for (var i in actions) {
6          var action = actions[i];
7
8          var newState = { state: StripsManager
9                           .applyAction(action, state),
10                          action: action };
11
12         if(!StripsManager.stateValidator(
13           domain.invalidState, newState)){
14           children.push(newState);
15         }
16     }
17     return children;
18 }
```

O método *applicableActions* (Listagem 5.3, linha 4) que implementa a aplicação da ação no estado atual, são verificadas todas as ações do domínio que a pré-condição corresponde ao estado atual passado como parâmetro. O número de ações aplicáveis será o mesmo de novos estados gerados. A cada ação iterada, um novo estado é gerado pelo método *applyAction* (Listagem 5.3). A aplicação de uma ação em um estado é feita a partir dos parâmetros de efeito da ação definida no domínio.

O método itera sobre a lista de efeitos e modifica o estado atual adicionando e removendo partes do estado de acordo com a operação, como por exemplo, o estado onde *(on a t1) (on b a) (on c t3) (clear t2)* for o estado atual e a ação *unstack2* estiver sendo aplicada, o efeito de desempilhar dois blocos será aplicado da seguinte forma: cada parte do estado atual será comparada com as partes de efeito definidas na ação *unstack2*.

A Listagem 5.2 mostra a estrutura dos dados da definição do efeito da ação unstack2. Portanto, o método verifica cada item do efeito e se a propriedade *operation* for *"and"*, a parte será adicionada no estado atual, caso a *operation* seja *"not"*, essa parte será removida do estado. O resultado dessa aplicação transforma o estado

atual no seguinte estado *(on a t1) (on b t2) (on c t3) (clear a)*.

Listagem 5.4 - Representação da estrutura de dados de um efeito

```
{
    action: "on",
    operation: "not",
    parameters: ["b", "a"]
},
{
    action: "on",
    operation: "and",
    parameters: ["b", "t2"]
},
{
    action: "clear",
    operation: "not",
    parameters: ["t2"]
},
{
    action: "clear",
    operation: "and",
    parameters: ["a"]
}
```

Após gerar um novo estado, o método *stateValidator* foi incluído na implementação para validar se o novo estado gerado pelo método *applyAction* é um estado inválido. O método *stateValidator* recebe como parâmetro a definição *invalidState* que se inclui no parâmetro de domain e o *newState* gerado. O validador retornará *true*, caso o novo estado seja idêntico à algum estado inválido configurado. No entanto, deve ser incluído na lista de estados filhos somente os estados diferentes dos estados inválidos.

A implementação do método validador de estado inválidos mostrado na Listagem 5.5 realiza a comparação entre os estados configurados no arquivo de estados inválidos com os novos estados gerados em tempo de planejamento para compor a solução do planejamento.

O método validador itera em três laços de repetição para validar os estados. O primeiro laço de repetição percorre a lista de estados inválidos configurados no arquivo,

pois podem haver mais de um estado inválido configurado. Para cada estado inválido, será iterada cada uma de suas partes no segundo laço de repetição, como por exemplo, o estado *(on a t1) (on b t2) (on c t3)* é composto por três partes, e na primeira iteração a parte *(on a t1)* será comparada. O terceiro laço de repetição itera sobre o conjunto de partes do novo estado e realiza a comparação.

Listagem 5.5 - Método validador de estados

```
1  stateValidator: function(invalidState, newState) {
2      var egualState;
3      for (var h in invalidState.actions){
4          actions = invalidState.actions[h].effect;
5          for(var i in actions){
6              egualState = false;
7              for (var j in newState.state.actions) {
8                  if (newState.state.actions[j].operation != "not"
9                          && actions[i].action ==
10                         newState.state.actions[j].action
11                         && StripsManager.arraysIdentical(
12                         actions[i].parameters,
13                         newState.state.actions[j].parameters)){
14                     egualState = true;
15                     break;
16                 }
17             }
18             if (!egualState)
19                 break;
20         }
21         if (egualState)
22             break;
23     }
24     return egualState;
25 }
```

A estrutura de dados que representa uma parte do estado é composta de três propriedades. A propriedade *action* (Listagem 5.5, linha 9) representa o relacionamento entre os parâmetros definidos na propriedade *parameters* (Listagem 5.5, linha 12) e a propriedade *operation* (Listagem 5.5, linha 8) indica se o relacionamento existe ou não no estado. Neste caso, quando a *operation* da parte do estado novo apresentar

o valor *"not"*, significa que o relacionamento não existe e não há necessidade de comparar com parte do estado inválido.

A comparação dos estados é feita a partir de três condições: se a *operation* não é negativa; se a *action* de ambas as partes for a mesma; e se os *parameters* das partes forem idênticos. O método *arraysIdentical* é um método existente na implementação do planejador que compara os parameters dos estados, verificando se o tamanho e os valores dos parâmetros são iguais. A Listagem 5.5 mostra a implementação completa do método *stateValidator*.

No segundo loop na (Listagem 5.5, linha 6), a variável *equalState* recebe o valor false, indicando que a parte ainda não foi encontrada no estado novo. Assim, caso o terceiro laço de repetição não encontrar uma parte igual e ao sair do loop a variável *equalState* (Listagem 5.5, linha 18) permanecer com o valor false, o comando *break* (Listagem 5.5, linha 19) é chamado, pois não há necessidade de continuar comparando as partes daquele estado inválido se o método não encontrar uma parte igual. Quando a condição é satisfeita, indicando que foi encontrada uma parte do estado inválido no estado novo, a variável *equalState* (Listagem 5.5, linha 14) é marcada com true e o comando *break* (Listagem 5.5, linha 15) é chamado para sair do laço de repetição e iterar sobre mais uma parte do estado inválido. Caso não haja mais partes para serem iteradas e a variável *equalState* (Listagem 5.5, linha 21) for *true* outro comando *break* (Listagem 5.5, linha 22) é chamado pois não há necessidade de continuar procurando na lista de estados inválidos caso algum estado inválido tenha todas suas partes encontradas no estado novo e o validador retorna a variável *equalState* (Listagem 5.5, linha 24).

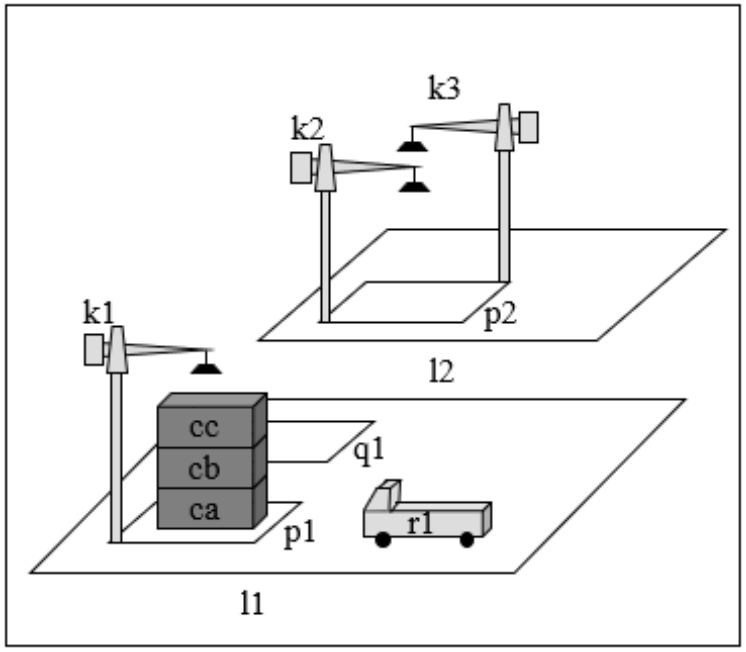
5.2 Validação da implementação com o problema de planejamento do robô trabalhador portuário

Escolheu-se o problema do robô trabalhador portuário (GHALLAB et al., 2004) para validar a implementação por ser um problema com mais ações do que o problema do mundo dos blocos e usar mais objetos no domínio de planejamento. O problema consiste em mover três containers entre dois locais diferentes, utilizando guindastes e um robô para transportá-los como mostra a Figura 5.3.

O problema possui, no estado inicial, dois locais, sendo *l1* e *l2*. No local *l1*, há duas pilhas *p1* e *q1*, um guindaste *k1* e três containers *ca*, *cb* e *cc* empilhados sobre a pilha *p1*. O local *l2* possui uma pilha *p2* e dois guindastes *k2* e *k3*. A configuração do estado inicial e disposição dos objetos do problema foram organizados para possibilitar a

inclusão de estados inválidos para o teste.

Figura 5.3 - Problema dos containers.



Fonte: Produção do autor.

O objetivo do problema é mover os containers para o local l2. O arquivo de domínio inclui as seguintes funções: mover - move um robô entre dois locais adjacentes; carregar - carrega um robô vazio com um container preso por um guindaste próximo; descarregar - descarrega um robô segurando um container com um guindaste próximo; pegar - pega um recipiente de uma pilha com um guindaste e colocar - coloca um container preso por um guindaste em uma pilha próxima.

5.2.1 Execução do plano

O problema foi executado no planejador e um plano com dezessete passos foi gerado para solucionar o problema. A Tabela 5.1 mostra a transição do estado inicial *Si* até o estado objetivo *Sg*, contendo a sequência de ações necessárias para mover os containers do local *l1* para o local *l2*, contidas no plano I.

Tabela 5.1 - Representação dos estados do plano I.

Ação	Estado	Representação
Inicial	Si	Containers empilhados no local *l1*
1. take *k1 l1 cc cb p1*	S1	O guindaste *k1* pegou o container *cc*
2. load *k1 l1 cc r1*	S2	O robô *r1* foi carregado com container *cc*
3. move *r1 l1 l2*	S3	O robô foi movido para o local *l2*
4. unload *k2 l2 cc r1*	S4	O guindaste *k2* descarregou o container *cc*
5. move *r1 l2 l1*	S5	O robô *r1* foi movido para o local *l1*
6. take *k1 l1 cb ca p1*	S6	O guindaste *k1* pegou o container *cb*
7. load *k1 l1 cb r1*	S7	O robô *r1* foi carregado com container *cb*
8. move *r1 l1 l2*	S8	O robô *r1* foi movido para o local *l2*
9. unload *k3 l2 cb r1*	S9	O guindaste *k3* descarregou o container *cb*
10. move *r1 l2 l1*	S10	O robô foi movido para o local *l1*
11. take *k1 l1 ca pallet p1*	S11	O guindaste *k1* pegou o container *ca*
12. load *k1 l1 ca r1*	S12	O robô *r1* foi carregado com container *ca*
13. move *r1 l1 l2*	S13	O robô foi movido para o local *l2*
14. put *k3 l2 cb pallet p2*	S14	O guindaste *k3* colou o container *cb* sobre a pilha *p2*
15. unload *k3 l2 ca r1*	S15	O guindaste *k3* descarregou o container *ca*
16. put *k3 l2 ca cb p2*	S16	O guindaste *k3* colou o container *ca* sobre *cb*
17. put *k2 l2 cc ca p2*	S17	O guindaste *k3* colou o container *cc* sobre *ca*

O teste consiste em escolher estados gerados no plano I para representar um grupo de estados inválidos, escolhidos aleatoriamente, simulando por exemplo, uma situação inválida dentro do cenário de planejamento. Por exemplo, nota-se no plano I que durante a transição de estados, o container *ca* é descarregado pelo guindaste *k3* no local *l2*. Suponha-se que há uma regra no domínio de transporte desses containers, que impeça o guindaste *k3* de carregar o container *ca*, por uma limitação de peso. Então determinou-se que o primeiro estado inválido desse domínio é "O guindaste *k3* descarregando o container *ca*".

Observa-se na Tabela 5.1 que o estado *S15 - O guindaste k3 descarregou o container ca sobre cb* é o estado que foi escolhido como inválido. Isso significa que a ação *15 – unload k3 l2 ca r1* contém o efeito responsável por gerar o estado inválido. Então, para criar o arquivo de estados inválidos em PDDL, é necessário utilizar a definição do efeito da ação *unload* definido no arquivo de domínio de planejamento. O efeito é representado da seguinte forma: *effect (and (unloaded ?r) (holding ?k ?c).*

A definição do efeito será utilizada para mapear o estado inválido. Os parâmetros do efeito são alterados para os nomes utilizados no problema. Neste caso, definiu-se o efeito como: *effect (and (unloaded ?r1) (holding ?k3 ?ca)*, indicando que o *k3* está segurando o container *ca* como mostra a Listagem 5.6.

Listagem 5.6 - Definição do primeiro estado inválido

```
(:action invalid1
    :parameters (?r1 ?k3 ?ca )
    :precondition (and (robot ?r1) (crane ?k3)
                       (container ?ca))
    :effect (and (unloaded ?r1) (holding ?k3 ?ca))
)
```

Outros estados inválidos foram escolhidos, incluindo mais quatro restrições para o domínio. A Listagem 5.7 mostra a configuração dos cinco estados inválidos configurados para este problema. No estado inválido *invalid1*, *invalid2* e *invalid3*, adicionou-se restrições nos guindastes do local *l2*. Onde o *invalid1* restringe o guindaste *k3* de descarregar o container *ca*, o *invalid2* restringe o guindaste *k2* de descarregar o container *cc* e o *invalid3* restringe o guindaste *k2* de descarregar o container *cb*. O estado inválido *invalid4* adiciona uma condição diferente, que restringe que o container *ca* esteja sobre o container *cc* e o *invalid5* restringe que o container *cc* esteja no topo da pilha de containers.

Listagem 5.7 - Definição dos estados inválidos do problema

```
(:action invalid1
  :parameters (?r1 ?k3 ?ca )
  :precondition (and (robot ?r1) (crane ?k3)
                (container ?ca))
  :effect (and (unloaded ?r1) (holding ?k3 ?ca))
)
(:action invalid2
  :parameters (?r1 ?k2 ?cc )
  :precondition (and (robot ?r1) (crane ?k2)
                (container ?cc))
  :effect (and (unloaded ?r1) (holding ?k2 ?cc))
)
(:action invalid3
  :parameters (?r1 ?k2 ?cb )
  :precondition (and (robot ?r1) (crane ?k2)
                (container ?cb))
  :effect (and (unloaded ?r1) (holding ?k2 ?cb))
)
(:action invalid4
  :parameters (?ca ?cc )
  :precondition (and (container ?ca)
        (container ?cc))
  :effect (and (on ?ca ?cc))
)
(:action invalid5
  :parameters (?pallet ?cc ?p2)
  :precondition (and (container ?pallet)
                (container ?cc) (pile ?p2))
  :effect (and (in ?cc ?p2) (top ?cc ?p2)
        (on ?cc ?pallet))
)
```

O problema foi executado novamente no planejador, dessa vez utilizando o método validador de estados inválidos no planejador. O plano II foi gerado com ações diferentes para encontrar o estado objetivo, cumprindo, assim, as restrições adicionadas. A Tabela 5.2 mostra que a ação que gerava o *invalid1* não está mais presente no

plano II: (1) A ação *11. unload k2 l2 ca r1* evidencia que o container *ca* agora é descarregado pelo guindaste *k2*; (2) A ação *4. unload k3 l2 cc r1* evidencia que o container *cc* agora é descarregado pelo guindaste *k3*; (3) A ação *18. unload k3 l2 cb r1* evidencia que o container *cb* agora é descarregado pelo guindaste *k3*; (4) A ação *16. put k2 l2 ca pallet p2* evidencia que o container *ca* agora é colocado sobre o pallet *p2*; (5) A ação *19. put k3 l2 cb cc p2* evidencia que o container *cc* agora não está ao topo da pilha *p2* pois o container *cb* foi empilhado sobre ele.

A relação entre as ações do plano e as transições de estado geradas é mostrada na Tabela 5.2. Os resultados obtidos com a função do validador de estado foram bem-sucedidos, pois os estados mapeados não estão presentes no plano resultante.

O planejador considerou o estado inválido em tempo de planejamento ao gerar o novo plano, mostrando que o problema de planejamento pode ser resolvido de maneiras diferentes, usando etapas diferentes. Por exemplo. o problema do mundo dos blocos pode ser resolvido em uma permutação de pilha quase infinita. Com o resultado do plano obtido na Tabela 5.2, observa-se que todos os estados inválidos descritos na Listagem 5.7 foram cumpridos nos testes preliminares.

Tabela 5.2 - Resultado da execução do plano.

Ação	Estado	Representação
Inicial	Si	Containers empilhados no local *l1*
1. take *k1 l1 cc cb p1*	S1	O guindaste *k1* pegou o container *cc*
2. load *k1 l1 cc r1*	S2	O robô *r1* foi carregado com o container *cc*
3. move *r1 l1 l2*	S3	O robô foi movido para o local *l2*
4. unload *k3 l2 cc r1*	S4	O guindaste descarregou o container *cc*
5. move *r1 l2 l1*	S5	O robô *r1* foi movido para o local *l1*
6. take *k1 l1 cb ca p1*	S6	O guindaste *k1* pegou o container *cb*
7. put *k1 l1 cb pallet q1*	S7	O guindaste *k1* colocou o container *cb* na pilha *q1*
8. take *k1 l1 ca pallet p1*	S8	O guindaste *k1* pegou o container *ca*
9. load *k1 l1 ca r1*	S9	O robô *r1* foi carregado com o container *ca*
10. move *r1 l1 l2*	S10	O robô foi movido para o local *l2*
11. unload *k2 l2 ca r1*	S11	O guindaste descarregou o container *ca*
12. move *r1 l2 l1*	S12	O robô foi movido para o local *l1*
13. take *k1 l1 cb pallet q1*	S13	O guindaste *k1* pegou o container *cb*
14. load *k1 l1 cb r1*	S14	O robô *r1* foi carregado com o container *cb*
15. move *r1 l1 l2*	S15	O robô foi movido para o local *l2*
16. put *k2 l2 ca pallet p2*	S16	O guindaste *k2* colocou o container *ca* na pilha *p2*
17. put *k3 l2 cc ca p2*	S17	O guindaste *k3* colocado container *cc* em *ca*
18. unload *k3 l2 cb r1*	S18	O guindaste *k3* descarregou o container *cb*
19. put *k3 l2 cb cc p2*	S19	O guindaste *k3* colocou o container *cb* em *cc*

5.3 Implementação da camada de configuração

Esta seção descreve a implementação do conversor de linguagem responsável por gerar as representações de estados inválidos para a camada de configuração, apresentada na Seção 4.3.2 da arquitetura.

Para garantir que a representação do conhecimento do domínio de planejamento seja corretamente mapeada entre as camadas da abordagem, foi utilizado o KPlanOO - Um Meta-modelo orientado a objetos para descrição de domínios de planejamento (SILVA et al., 2010). A ontologia aplicada na representação do domínio do KPlanOO, permite criar a configuração e integração dos processos de forma flexível e padronizada. Além de ser um modelo genérico, permitindo o reuso da solução sem depender da linguagem de planejamento usada no processo.

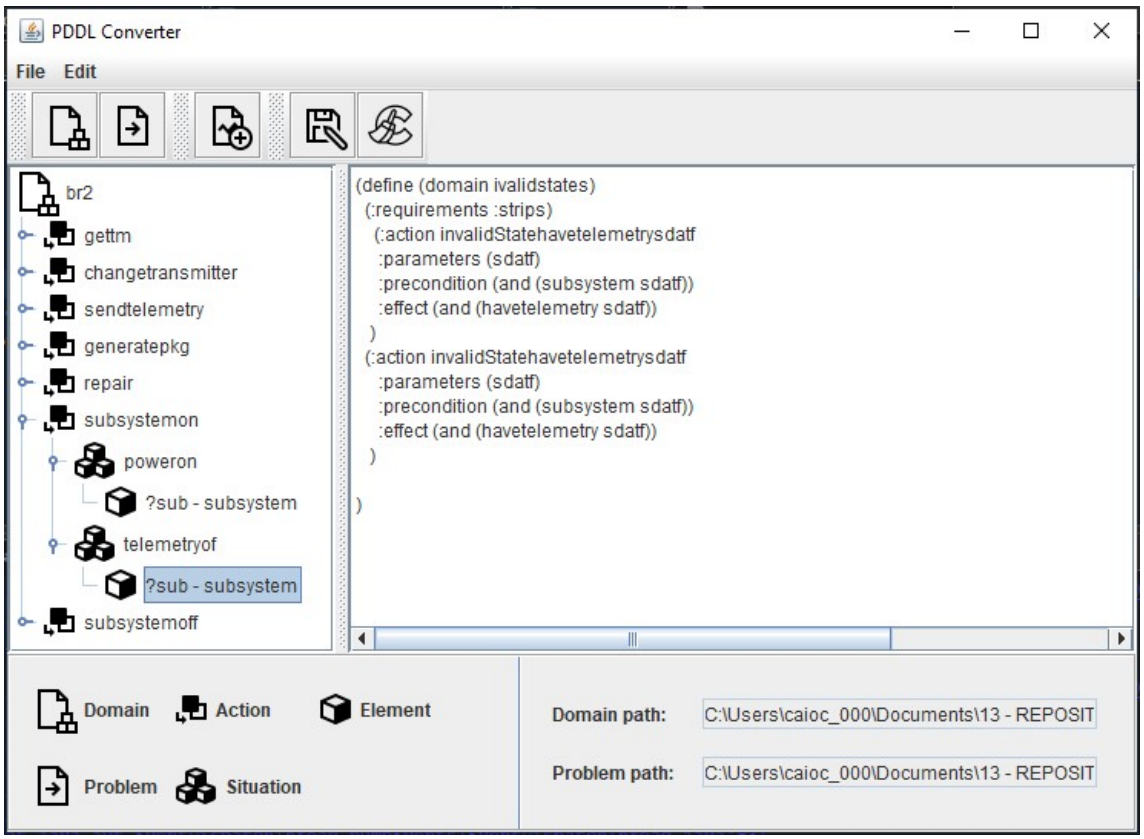
A seguir é apresentada uma breve descrição de alguns objetos de classe definidos na estrutura de representação do KPlanOO que são utilizados na implementação do conversor. Segundo Silva et al. (2010):

- **Domain (domínio)**: representa a descrição do domínio de planejamento, composta de um conjunto de ações, que possuem um conjunto de estados prováveis para o problema com tipos definidos.
- **Problem (problema)**: seu objetivo é representar a descrição do problema para um determinado domínio. É composto de uma instância da *classe Init*, uma instância da *classe Goal*, e também uma lista de elementos que irão representar os objetos a serem usados pelo planejador.
- **Action (ação)**: é uma abstração de ações do domínio, que contempla a especificação de uma "Super Ação", por meio de uma auto-relação. Cada ação possui um efeito e uma pré-condição, que são objetos do tipo cenário.
- **Scenario (cenário)**: entidade que representa um cenário composto pelos estados definidos no problema em questão. Sendo estes tipos declarados no domínio.
- **Situation (situação)**: entidade que representa as situações do objeto que serão manipuladas no planejamento. Estas situações são o conjunto formado pelo estado dos objetos e as condições vinculadas a este estado.
- **Element (elemento)**: abstrai os objetos que farão parte do problema. O elemento tem um tipo e pode ter valores (*Value*).

- **Type (tipo)**: é a abstração dos tipos que podem ser criados em um domínio. Por exemplo: satélite, subsistema, órbita, etc...

A Figura 5.4 mostra a interface gráfica implementada para o conversor de estados. Na visualização em árvore que apresenta os itens de planejamento, são exibidas as situações definidas no efeito de cada ação do domínio e listados conforme seu tipo, sendo: ação; situação ou elemento.

Figura 5.4 - Interface gráfica do coversor de estados.



Fonte: Produção do autor.

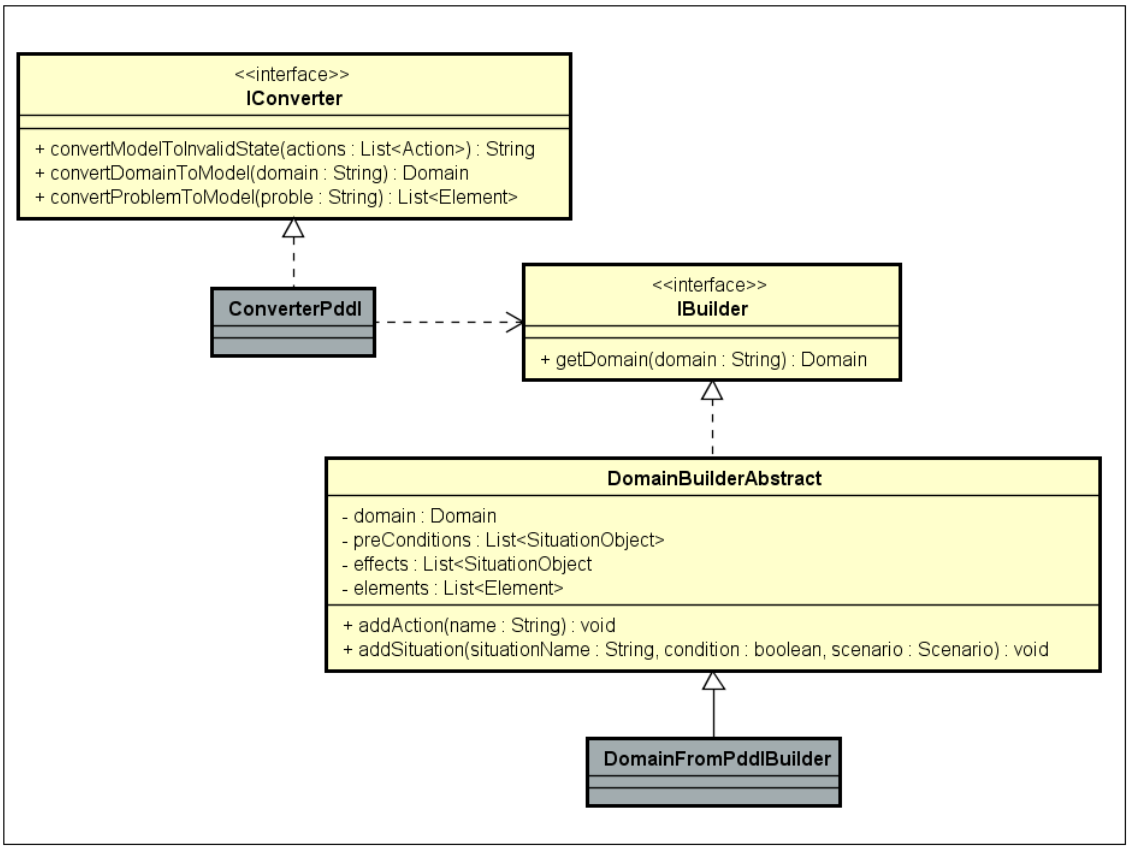
O conversor de linguagem lê os arquivos de domínio e problema de planejamento, e usando a estrutura de representação de domínio do KPlanOO (SILVA et al., 2010), carrega os objetos do domínio por meio de uma interface gráfica, que o especialista informará as preferências dos estados inválidos.

A confecção do arquivo de estados inválidos é feita a partir da escolha de situa-

ções associadas aos elementos do problema e que são geradas a partir dos efeitos definidos nas ações do domínio. Para montar a definição do arquivo, um especialista visualiza os objetos carregados e escolhe dentro de uma ação qual situação será invalidada. Conforme descrito na Seção 4.2, após a escolha de uma situação, para cada elemento deverá ser escolhido um objeto definido no problema. Então o conjunto de situações escolhidas será convertido na representação de estados inválidos no formato da linguagem utilizada no planejador.

A Figura 5.5 apresenta, através do diagrama de classes, a estrutura do componente conversor de estados. Nas interfaces são utilizadas as classes definidas no metamodelo KPlannOO para permitir que a representação do domínio de planejamento seja convertida em objetos. A interface *IConverte* foi criada para definir os seguintes métodos de conversão:

Figura 5.5 - Diagrama de classes do conversor.



Fonte: Produção do autor.

- **ConvertModelToInvalidState:** é responsável por construir o arquivo de estados inválidos em uma linguagem de planejamento. O parâmetro recebido depende de uma lista de objetos do tipo *Action*, que são usados para construir os estados inválidos e retornar à definição em um objeto do tipo *String*.

- **ConvertDomainToModel:** é responsável por carregar o arquivo de domínio de planejamento de um objeto do tipo *String*. Os dados do arquivo são lidos, instanciados e retornados em um objeto do tipo *Domínio*.

- **ConvertProblemToModel:** é responsável por carregar o arquivo do problema de planejamento de um objeto do tipo *String*. Uma lista de objetos do tipo *Elemento* é retornada.

As classes em destaque no diagrama representam as implementações específicas necessárias para a conversão da linguagem de planejamento. O conversor pode ser implementado usando outras linguagens, se novas classes implementarem as interfaces *IConverter* e *IBuilder*, convertendo o domínio para a respectiva linguagem utilizada. A classe que implementa a interface *IConverter* tem relacionamento com uma classe construtora de objetos de domínio. A partir da herança da classe *DomainBuilderAbstract*, que encapsula a construção do domínio por meio de um grupo de métodos responsáveis por facilitar a conversão de cada objeto em código da linguagem, ao utilizar os métodos *addAction* e *addSituation* é possível construir os arquivos na linguagem de planejamento.

Por fim, um usuário irá associar um arquivo de estados inválidos a cada classe definida na camada de classificação de acordo com a necessidade do domínio, podendo configurar grupos de situações inválidas, como no exemplo apresentado na Seção 4.2, cada instante pode ser considerado uma classe do domínio, onde um grupo diferente de situações são inválidas em cada classe.

5.4 Implementação da camada de classificação

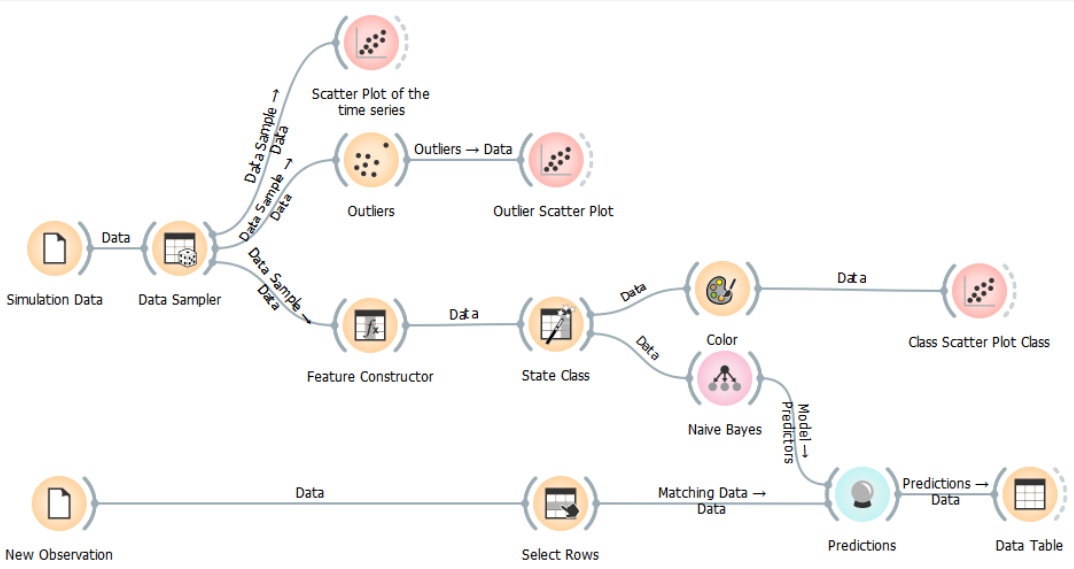
Nesta seção descreve a implementação da camada de Classificação de Dados.

Para classificar as instâncias dos dados recebidos da operação de satélites, foi utilizada a ferramenta Orange Data Mining (DEMŠAR et al., 2013), que permite criar fluxos de trabalho para carregar dados, visualizar, processar, classificar e entre outras operações possíveis no processo de mineração de dados (MINING, 2020).

A organização do fluxo é dada a partir da conexão dos *widgets*, que possuem quatro funções diferentes sendo identificados por cores: os *widgets* de cor laranja são responsáveis por manipular os dados; os *widgets* de cor vermelho são responsáveis por visualizar os dados; os *widgets* de cor lilás representam um modelo de predição de dados e os *widgets* de cor azul representam o processo de avaliação os dados classificados.

A Figura 5.6 mostra o fluxo de classificação de dados implementado, definindo a sequência de passos genéricos necessários para realizar a classificação de dados do domínio e a partir dos critérios de classificação realizando a predição do estado do sistema.

Figura 5.6 - *Work flow* do Orange.



Fonte: Produção do autor.

O fluxo é iniciado com o *widget file*, responsável por carregar a base de dados. Na sequência o *widget* responsável por separar amostras dos dados foi conectado, fornecendo uma amostra dos dados para os próximos *widgets* conectados. A primeira análise realizada nesse fluxo é a utilização do *widget outlier*, responsável por identificar anormalidades em uma série temporal. Na sequência a partir do *widget constructor*, onde os critérios de classificação são informados, os dados ganham uma nova *feature* onde é preenchido o valor correspondente a classe gerada pelo *feature constructor*. No *widget* de modelo o algoritmo de classificação de dados um algoritmo

de classificação de dados é associado ao *widget* de predição de dados. Para classificar as novas instâncias dos dados, no fluxo de trabalho é incluído com *widget new observation*, uma nova entrada de dados representando novas observações para serem classificas, fornecendo os dados de entrada para o widget predictions, que classifica as instâncias com base no modelo de classificação definido.

6 ESTUDO DE CASO

Neste capítulo, será demonstrada a aplicação da arquitetura de planejamento automatizado com predição de estados inválidos em um estudo de caso, usando um domínio da área espacial. Os dados de telemetrias de um satélite, obtidos por meio de simulação, foram utilizados para desenvolver um modelo de classificação, configurar estados inválidos e integrar um processo de planejamento. O estudo de caso foi realizado para atender os seguintes objetivos:

- Criar um cenário no domínio de satélite para aplicar um processo de classificação de estados inválidos;
- Apresentar o resultado da integração dos processos de classificação e planejamento por meio dos planos gerados automaticamente;
- Validar se os estados inválidos configurados foram atendidos e correspondem a classificação apresentada pelo estado atual do domínio.

6.1 Descrição do estudo de caso

Nesse experimento foi definido um estudo de caso para criar um processo de classificação de estados inválidos e realizada a simulação da operação de um satélite para obter os dados de telemetria.

Os dados da camada de classificação foram obtidos através de uma ferramenta para desenvolvimento e simulação de autômatos finitos, o simulador Atom SysVAP (Sistema de validação de autômatos finitos e planos de execução) (IVO, 2013). A ferramenta permite representação de máquinas de estado e é adequada para simular os estados de domínio do experimento. A Figura 6.1 mostra a versão da ferramenta utilizada no estudo de caso.

O domínio escolhido foi o satélite **NanoSat BR2** presente nos modelos de exemplo do simulador Atom SysVAP (IVO, 2013). Este domínio é composto por cinco subsistemas, sendo três de carga útil, o OBC - computador de bordo e o transmissor. Este domínio possuí o seguinte conjunto de telecomandos, que podem ser executados em um plano no simulador: **get_tm** - gera o pacote de telemetrias dos subsistemas; **repair** - repara o estado de alerta do subsistema de controle de bordo; **com_t_on** - liga o modo de transmissão do transmissor; **com_t_off** - desliga o modo de transmissão do transmissor; **sub_on** - liga um subsistema **sub_off** - desliga um subsistema (IVO, 2013).

Figura 6.1 - Versão do simulador Atom SysVAP.

Fonte: Ivo (2013).

A Tabela 6.1 lista as características do conjunto de dados enviados no pacote de telemetrias. A telemetria de bateria é um valor numérico que indica o nível de bateria do satélite (o nível varia entre 13% e 19%). O *Total drop* é um índice do total de descarga de energia da bateria, calculado a través da somatória do consumo de energia de cada subsistema pelo tempo de operação (o valor varia entre 0 e 1). A telemetria de órbita indica a leitura do ambiente, *Sun* - quando a bateria do satélite está sendo carregada pelo sol e *Ecl* - quando o satélite está em eclipse e a bateria não está sendo carregada. A telemetria do OBC indica se o sistema está no estado normal ou em alerta. A telemetria *Communication* indica o modo de operação do transmissor, *Re/TR* - indicando que o modo de transmissão e recebimento está ativo ou *Receiver* - apenas o modo de receptor ativo. As demais telemetrias correspondem as cargas úteis e indicam quando estão ligadas ou desligadas.

A simulação consistiu em submeter o satélite em uma situação de extrema descarga da bateria, através da simulação da operação com todos os subsistemas ligados ao mesmo tempo, causando maior consumo da bateria. O processo de análise dos dados e criação do modelo de classificação de dados foi realizado utilizando a ferramenta Orange Data Mining (DEMŠAR et al., 2013). Ao executar a simulação de operação do satélite, os dados de telemetrias gerados foram carregados e analisados para a

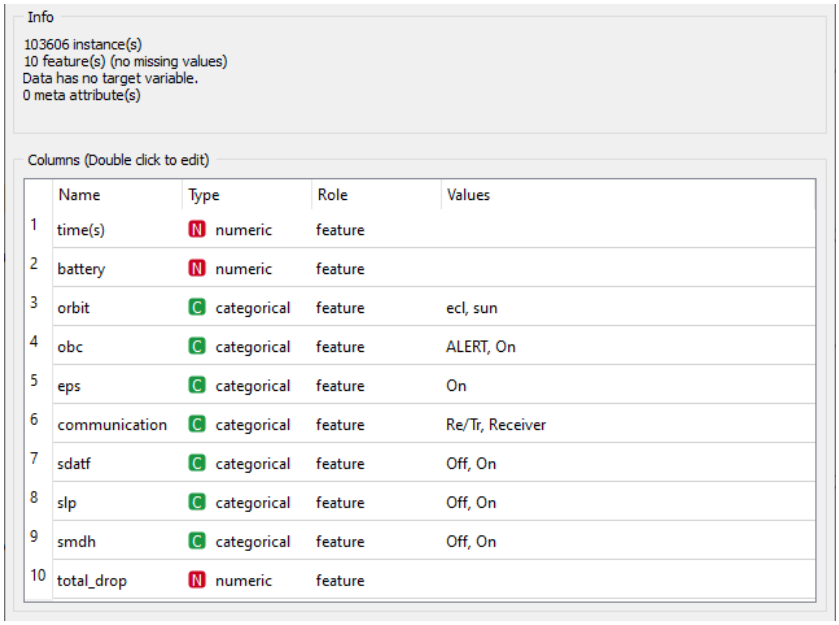
formação dos critérios de classificação e treinamento dos algoritmos.

Tabela 6.1 - Telemetry data

Name	Type	Values
Battery	numérico	13% - 19%
Total drop	numérico	0 - 1
Orbit	categórico	Sun, Ecl
OBC	categórico	Alert, On
Communication	categórico	Re/Tr, Receiver
SDATF	categórico	On, Off
SLP	categórico	On, Off
SMDH	categórico	On, Off
Time(s)	numérico	-

O pacote de dados de telemetria gerado na simulação com o experimento da bateria foi exportado no formato *.csv* e carregado na ferramenta Orange, no fluxo de trabalho definido pela Figura 5.6 na Seção 5.4. A Figura 6.2 mostra as características dos dados obtidos na simulação, sendo três *features* do tipo numérico e outras sete *features* do tipo categóricas. O número de total é de 103606 instâncias.

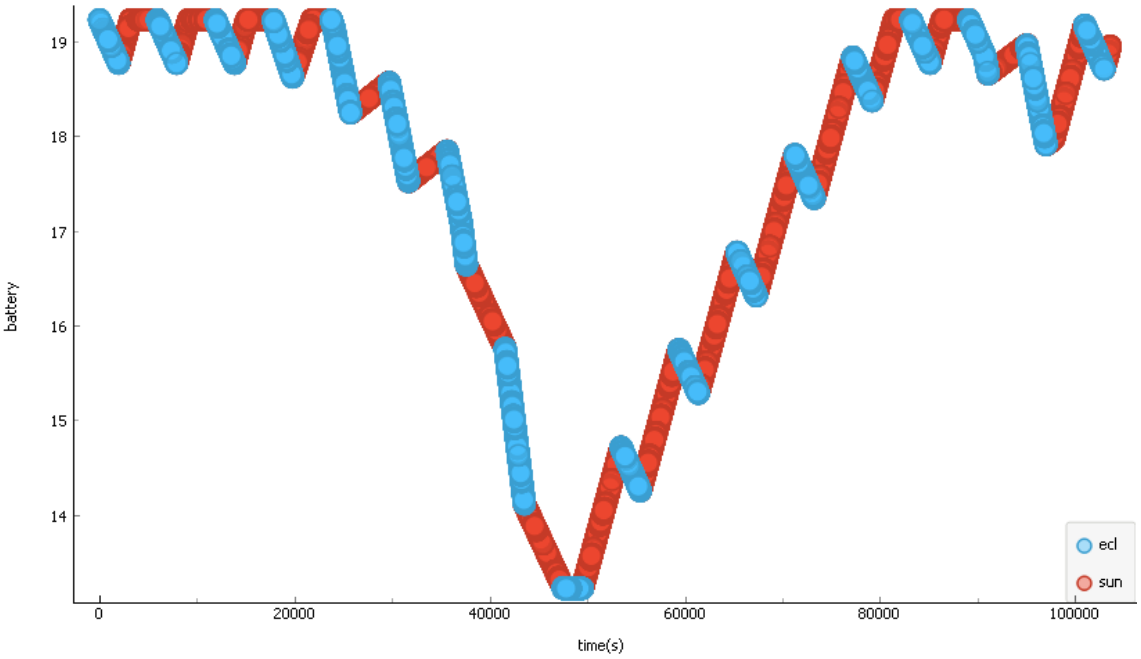
Figura 6.2 - Dados da simulação.



Fonte: Produção do autor.

A Figura 6.3 mostra o gráfico de dispersão da variação do nível de bateria durante a simulação. Observa-se que no instante 25.000, o nível de bateria começa a abaixar gradativamente até que no instante 40.000, mesmo com a presença do sol (representado pela cor vermelho), a bateria continua descarregando. O cenário de descarga da bateria ocorre até o instante 50.000, quando o subsistema OBC entra em estado de alerta, desligando os demais subsistemas do satélite, para garantir a bateria não seja descarregada totalmente. Após este momento a bateria começa a ser carregada novamente.

Figura 6.3 - Gráfico de dispersão do nível de bateria.



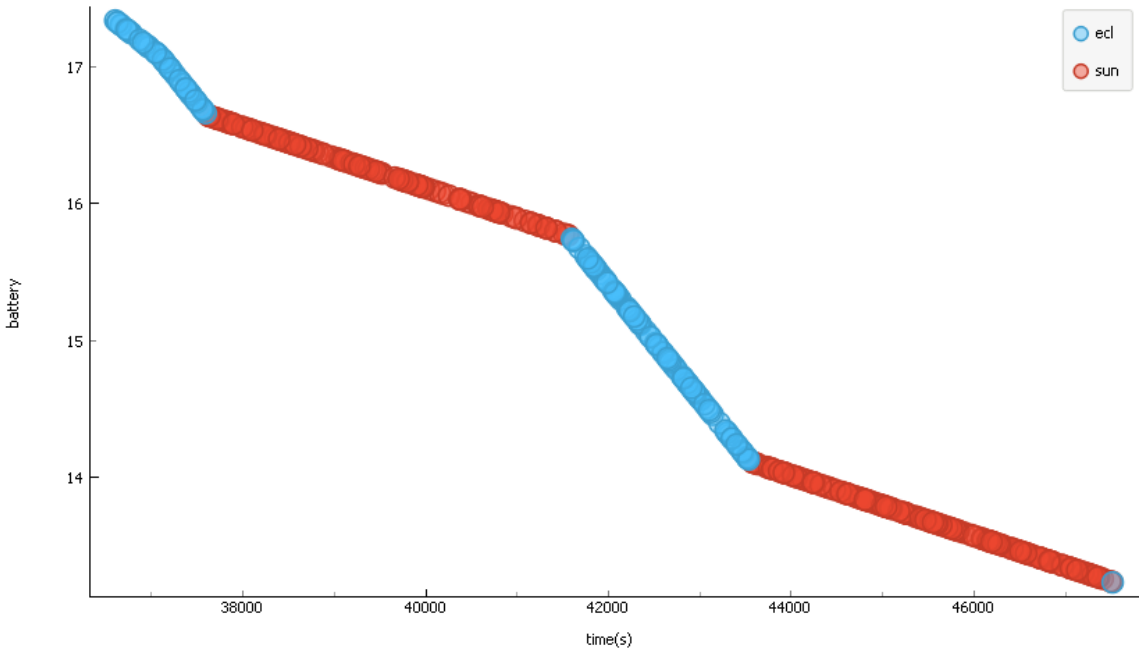
Fonte: Produção do autor.

A seguir são descritas a classificação dos estados do sistema, a configuração dos estados inválidos e a definição do domínio e problema de planejamento: na etapa de classificação, foi realizada a análise dos dados, desenvolvido o critério de classificação e a escolha de um algoritmo para classificar os dados; em configuração, foi realizada a escolha dos grupos de estados inválidos que foram configurados; em planejamento, foi mostrado o resultado dos planos gerados usando a configuração do experimento.

6.1.1 Classificação dos estados do sistema

Uma das maneiras de identificar automaticamente as anormalidades em um conjunto de dados é através da utilização de algoritmos identificadores de Outliers (MINING, 2020). Foi utilizado o *widget Outlier Detection* do Orange (DEMŠAR et al., 2013) com o método estimador de covariância para visualizar as anormalidades presentes nos dados da simulação. A Figura 6.4 mostra o resultado dos outliers identificados na série temporal.

Figura 6.4 - Gráfico de dispersão da análise de *outlier*.



Fonte: Produção do autor.

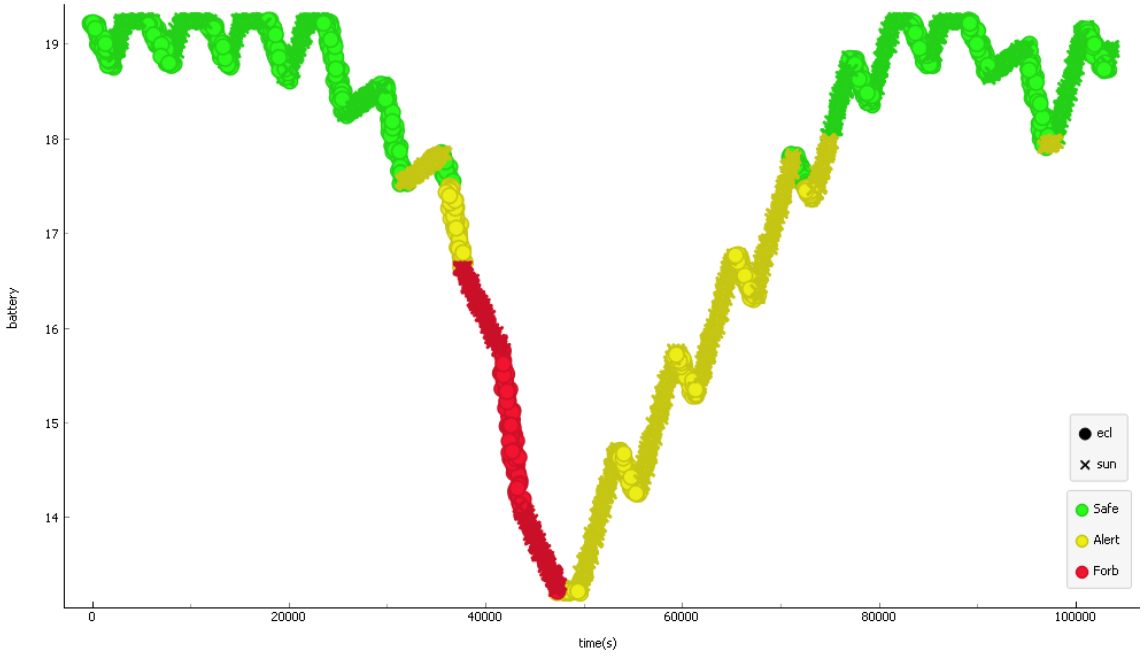
Com base na análise do comportamento anormal identificado nos outliers e no estudo de classificação de estados de bateria apresentado por Primavera Souza (SOUZA et al., 2010), foram desenvolvidos os seguintes critérios de classificação de estados apresentados na Tabela 6.2. Considerando o nível de bateria, a órbita e o total de descarga, foram criadas três classes. O estado *Forbidden* representa a situação em que a bateria não é mais carregada quando iluminada pelo sol. O estado de *Alert* representa a situação onde o nível da bateria é inferior a 18 no sol e a 17,5 em eclipse. O estado *Safe* representa a situação comum quando o nível de bateria se apresenta superior os valores de 18 em sol e 17,5 em eclipse.

Tabela 6.2 - Tabela de definição dos criatérios de classificação

Battery	Total Drop	Orbit	Operating State
$> 17.5\%$	< 0.000533	ECL	SAFE
$> 18\%$	< 0.000533	SUN	SAFE
$< 17.5\%$	< 0.000533	ECL	ALERT
$< 18\%$	< 0.000533	SUN	ALERT
$< 16.65\%$	> 0.000533	ECL/SUN	FORBIDDEN

A ferramenta Orange disponibiliza o *widget* Feature Constructor para criação de novas features no conjunto de dados. Os parâmetros de classificação definidos na Tabela 6.2 foram inseridos no *widget* para criar uma nova feature com o valor da classe estabelecida a partir da característica dos dados. A Figura 6.5 mostra, através do gráfico de dispersão, o resultado da classificação, onde as classes são identificadas por cores.

Figura 6.5 - Gráfico de dispersão dos dados classificados.



Fonte: Produção do autor.

A partir dos dados classificados, um modelo de predição de dados foi criado, para identificar o status de operação do sistema. Utilizando o widget *Predictions* conec-

tado ao modelo do classificador *Naive Bayes* do Orange (DEMŠAR et al., 2013). Porque segundo o estudo apresentado por Galal et al. (2019), o classificador *Naive Bayesian* apresentou um bom resultado como classificador de detecção de falhas em baterias de satélites. O modelo utiliza os dados classificados para realizar as predições em novas observações de telemetrias recebidas.

6.1.2 Definição do domínio de planejamento e configuração dos estados inválidos

As definições de domínio e problema de planejamento para serem usadas na validação da camada de planejamento automatizado, foram criados com base nos comandos presentes na simulação, conforme descritos na Seção 6.1. O objetivo do problema de planejamento é encontrar um plano que contenha os comando necessários para enviar as telemetrias obtidas dos subsistemas do domínio NanoSatC BR2. No entanto, o subsistema OBC deve obter as telemetrias dos subsistemas de carga útil e gerar um pacote para ser enviado pelo transmissor. A Listagem 6.1 mostra a definição do domínio de planejamento e a Listagem 6.2 mostra a definição do problema de planejamento.

Listagem 6.1 - Domínio de planejamento

```
(define (domain BR2)
 (:requirements :strips :typing )
 (:types
  satellite
  subsystem
  instrument
  mode)

;;Obtem telemetrias
(:action gettm
    :parameters (?sat - satellite ?obc - subsystem
                  ?sub - subsystem)
    :precondition
        (and (obcsub ?obc) (alertoff ?obc)
        (onboard ?obc ?sat) (onboard ?sub ?sat)
        (telemetryof ?sub))
    :effect
        (and (havetelemetry ?sub) (not (telemetryof ?sub)))
)
```

```
;; Liga transponder
(:action changetransmitter
        :parameters (?trans - instrument ?mod - mode)
        :precondition (and (support ?trans ?mod)
        (receiveron ?trans) (transmissionmode ?trans ?mod))
        :effect (and (transmitteron ?trans)
        (not (receiveron ?trans)))
)

;; Envia telemetrias
(:action sendtelemetry
        :parameters (?trans - instrument ?mod - mode
                ?sub - subsystem)
        :precondition (and (support ?trans ?mod)
        (transmitteron ?trans) (fullpkg ?sub))
        :effect (and (sendtelemetry ?trans)
        (receiveron ?trans)
        (not (transmitteron ?trans)))
)

;; Gera pacote telemetiras
(:action generatepkg
        :parameters (?obc - subsystem ?sub1 - subsystem
            ?sub2 - subsystem ?sub3 - subsystem)
        :precondition (and (obcsub ?obc)
        (havetelemetry ?sub1) (havetelemetry ?sub2)
        (havetelemetry ?sub3)
        (poweravail ?sub1) (poweravail ?sub2)
        (poweravail ?sub3))
        :effect (and (fullpkg ?obc))
)

;; Repara o OBC do estado de Alerta
(:action repair
    :parameters (?obc - subsystem ?sat - satellite)
    :precondition
```

```
        (and (obcsub ?obc) (onboard ?obc ?sat)
        (alerton ?obc))
    :effect
        (and (alertoff ?obc) (not (alerton ?obc)))
)

;; Liga um subsistema
(:action subsystemon
    :parameters (?sub - subsystem)
    :precondition
        (and (poweravail ?sub))
    :effect
        (and (poweron ?sub) (telemetryof ?sub)
        (not (poweravail ?sub)))
)

;; Desliga um subsistema
(:action subsystemoff
    :parameters (?sub - subsystem)
    :precondition
        (and (poweron ?sub))
    :effect
        (and (poweravail ?sub) (not (poweron ?sub))))
)
```

Listagem 6.2 - Problema de planejamento

```
(define (problem stripsproblem)
(:domain BR2)
(:objects
        sat - satellite
        obc sdatf slp smdh - subsystem
    transmissor - instrument
    receiver rectr - mode
)
(:init
        (obcsub obc)
        (onboard obc sat)
        (onboard transmissor sat)
```

```
        (onboard sdatf sat)
        (onboard slp sat)
        (onboard smdh sat)
        (poweravail transmissor)
        (poweravail sdatf)
        (poweravail slp)
        (poweravail smdh)
        (alertoff obc)
        (support transmissor rectr)
        (support transmissor receiver)
        (transmissionmode transmissor rectr)
        (receiveron transmissor)
)
(:goal (and
        (sendtelemetry transmissor)))
)
```

Nas ações do domínio foram definidas as seguintes precondições para alcançar o objetivo do problema:

- O transmissor deve estar configurado no modo Re/Tr para enviar as telemetrias;
- O subsistema deve estar ligado para que o OBC obtenha a telemetria;
- O pacote deve ser gerado com um par de telemetrias correspondentes a no máximo dois subsistemas de carga útil.

O estado inicial do problema de planejamento foi definido com todos os subsistemas de carga útil desligados e o transmissor configurado no mode Receiver. O objetivo do problema é enviar as telemetrias do satélite.

Para configurar os estados inválidos no processo de planejamento automatizado, para cada estado de operação definido na Tabela 6.2 foi atribuído um grupo de estados inválidos, como mostra a Tabela 6.3. No primeiro grupo foi definido que no estado de operação SAFE, o cenário onde o subsistema SLP estiver ligado representa uma situação inválida. No segundo grupo, no estado de operação ALERT, o cenário onde os subsistemas SLP e SMDH estiverem ligados simultaneamente é inválido e

por último, no terceiro grupo, onde ambos os subsistemas SLP, SMDH ou SDATF estiverem ligados individualmente ou simultaneamente é inválido.

Tabela 6.3 - Tabela de configuração de estado inválida

Group	Operating State	Invalid Situations
G1	SAFE	ON - SLP
G2	ALERT	ON - SLP and SDATF
G3	FORBIDDEN	ON - (SLP and SDATF and SMDH) and (SLP or SDATF or SMDH)

O arquivo de estado inválido foi gerado utilizando a interface da camada de configuração correspondente, onde os arquivos gerados em cada um dos grupos configurados neste experimento estão listados abaixo. A Listagem 6.3 mostra o arquivo de estados inválidos configurado para o grupo G1. A situação inválida é configurada a partir da definição do efeito da ação *subsystemon*, onde a expressão *(poweron ?slp)* representa a situação onde o subsistema SLP está ligado.

Listagem 6.3 - Definindo estados inválidos do grupo G1

```
(define (domain invalidstatesG1)
  (:requirements :strips)
    (:action invalidState
     :parameters (?slp)
     :precondition (and (subsystem ?slp))
     :effect (and (poweron ?slp))
    )
)
```

A Listagem 6.4 mostra a definição do arquivo de estados inválidos do grupo G2. Para representar a ligação simultânea de dois subsistemas, foi configurado o efeito como mostra a expressão *(poweron ?slp) (poweron ?sdatf)*, combinando os subsistemas SLP e SMDH ligados simultaneamente.

Listagem 6.4 - Definindo estados inválidos do grupo G2

```
(define (domain invalidstatesG2)
  (:requirements :strips)
    (:action invalidState
     :parameters (?slp ?sdatf)
```

```
    :precondition (and (subsystem ?slp) (subsystem ?sdatf))
    :effect (and (poweron ?slp) (poweron ?sdatf))
  )
)
```

A configuração do grupo G3 de estados inválidos é apresentada na Listagem 6.5. Cada subsistema é definido em uma ação diferente, representando que os subsistemas não poderão estar ligados individualmente nem simultaneamente.

Listagem 6.5 - Definindo estados inválidos do grupo G3

```
(define (domain invalidstatesG3)
  (:requirements :strips)
    (:action invalidState
     :parameters (?slp ?sdatf ?smdh)
     :precondition (and (subsystem ?slp) (subsystem ?sdatf)
        (subsystem ?smdh))
     :effect (and (poweron ?slp) (poweron ?sdatf)
        (poweron ?smdh))
    )
    (:action invalidSlp
     :parameters (?slp)
     :precondition (and (subsystem ?slp))
     :effect (and (poweron ?slp))
    )
    (:action invalidSdatf
     :parameters (?sdatf)
     :precondition (and (subsystem ?sdatf))
     :effect (and (poweron ?sdatf))
    )
    (:action invalidSmdh
     :parameters (?smdh)
     :precondition (and (subsystem ?smdh))
     :effect (and (poweron ?smdh))
    )
)
```

Com o modelo de classificação definido e os estados inválidos do domínio de planejamento configurados de acordo com a definição da arquitetura na Seção 4.3, o

planejador validador de estados inválidos proposto por Cruz et al. (2020) foi utilizado para executar o problema de planejamento, usando o mesmo domínio e problema com os diferentes grupos de estados inválidos.

6.2 Resultado do experimento

Esta seção apresenta o resultado da execução do processo de planejamento automático, com base no estudo de caso definido. Para cada grupo de estados inválidos foi executado um planejamento. A Listagem 6.6 mostra o plano gerado usando o arquivo de estados inválidos do grupo G1.

Listagem 6.6 - Plano executado com o grupo G1

```
1. changetransmitter transmissor rectr
2. subsystemon sdatf
3. gettm sat obc sdatf
4. subsystemon smdh
5. gettm sat obc smdh
6. subsystemoff sdatf
7. subsystemoff smdh
8. generatepkg obc sdatf smdh
9. sendtelemetry transmissor rectr obc

- Solution found in 9 steps!
Depth: 9, 134883 child states.
Runtime: 476579.710ms
```

Como mostrado na Listagem 6.6 no passo 1 - o transmissor foi alterado para o modo de Re/Tr; no passo 2 - o subsistema SDATF foi ligado; no passo 3 - o OBC obteve a telemetria do SDATF; no passo 4 - o subsistem SMDH foi ligado; no passo 5 - o OBC obteve a telemetria do SMDH; no passo 6 e 7 - os subsistemas foram desligados; no passo 8 - o pacote de telemetrias foi gerado; e no passo 9 - o transmissor envia o pacote de telemetrias gerado pelo OBC. Nota-se que o subsistema do SLP, definido no grupo G1, não foi ligado pela ação *subsystemon* durante os passos do plano gerado.

A Listagem 6.7 mostra o plano gerado usando o arquivo de estados inválidos do grupo G2. A restrição imposta no grupo G2 definiu que os subsistemas SLP e SDATF não estavam ligados simultaneamente, então no plano, após obter a telemetria do subsistema SDATF no passo 3, o subsistema foi desligado no passo 4 antes que o subsis-

tema SLP fosse ligado, respeitando assim a restrição imposta nos estados inválidos. As restrições foram seguidas pelo planejador em ambos os grupos, onde podemos observar que os subsistemas foram ligados respeitando as regras configuradas.

Em termos de desempenho da geração dos planos, embora a profundidade da árvore de busca seja a mesma, observa-se que com o aumento de situações inválidas do grupo G2, o número de estados filhos gerados no espaço de busca é 50 vezes menor, assim como o tempo de execução, que é 25 vezes menor quando comparado ao plano do grupo G1. Devido ao fato de o planejador ignorar os estados inválidos e gerar menos ramos.

Listagem 6.7 - Plano executado com o grupo G2

```
1. changetransmitter transmissor rectr
2. subsystemon sdatf
3. gettm sat obc sdatf
4. subsystemoff sdatf
5. subsystemon slp
6. gettm sat obc slp
7. subsystemoff slp
8. generatepkg obc sdatf slp
9. sendtelemetry transmissor rectr obc

- Solution found in 9 steps!
Depth: 9, 2388 child states.
Runtime: 18887.102ms
```

No planejamento do grupo G3 não foi gerado um plano, pois o planejador não encontrou uma solução para enviar as telemetrias do satélite, porque no mínimo dois subsistemas de carga útil devem ser ligados, conforme descrito na Seção 6.1.2. Conclui-se que ambos os três grupos de estados inválidos foram atendidos nos planejamentos executados e por tanto os objetivos do estudo de caso que foram definidos no início desta seção foram alcançados.

Por fim, o estudo de caso permitiu que fosse aplicado o processo de classificação a partir dos critérios dos estados simulados da operação do domínio. O resultado da integração dos processos foi comprovado através dos planos gerados de forma automatizada e por meio da verificação das etapas diferentes geradas em cada um dos planos, respeitando estados inválidos configurados em cada grupo. No exemplo foi utilizada a situação da bateria do satélite para definir quando os subsistemas

podem ser ligados, prevenindo, dessa forma, que o satélite entre em estado de alerta e a sua bateria seja descarregada completamente.

7 CONCLUSÃO

A validação de planos não é uma tarefa trivial no planejamento automatizado de domínios complexos, pois os planos gerados podem oferecer riscos à segurança de um domínio se alguma etapa de validação não existir. O propósito desse trabalho foi alcançado com êxito por meio da configuração de estados inválidos, garantindo o atendimento das restrições e gerando planos válidos. Com a validação de estados sendo feita em tempo de planejamento, extinguiu-se a necessidade de executar uma etapa de validação de planos após o processo de planejamento, contribuindo para a redução do tempo total de planejamento e evitando replanejamentos.

Usar um planejador modificado para validar estados inválidos, incluindo as restrições de planejamento obtidas a partir da classificação de domínio, foi possível a partir da utilização do meta modelo de planejamento orientado à objetos, que ofereceu uma forma padronizada de unir os processos de planejamento e classificação. A implementação da arquitetura permite que a solução seja reutilizada e que a configuração seja facilmente implementada com outras linguagens de planejamento.

Conclui-se que a implementação da arquitetura permitiu gerar os planos de forma mais independente, porque as mudanças no comportamento do domínio puderam ser consideradas no planejamento automatizado, sem que houvesse a modificação das definições de domínio e problema de planejamento.

7.1 Contribuições

A contribuição que este trabalho deixa para o planejamento das missões espaciais do INPE é uma arquitetura capaz de gerar planos com maior qualidade, uma vez que sempre um subsistema no satélite pode parar de funcionar e isso deve ser refletido no planejamento. A arquitetura apresentada contribui para atualizar o planejamento das mudanças que podem ocorrer no domínio, porque identificar as mudanças ocorrentes em alguns domínios pode levar bastante tempo. No entanto, problemas como execução de planos inválidos podem ser eliminados com um processo que identifique mudanças imediatamente.

Este trabalho deixa também como contribuição acadêmica, uma nova forma de planejar, uma vez que consegue melhorar o planejamento validando estados inválidos e o planejador gerar planos mais próximos da realidade. Como em outros métodos de planejamento, técnicas inovadoras são usadas para melhorar os processos, aprendendo novas regras, este trabalho também emprega a classificação para configurar

grupos de estados inválidos e integrar soluções viáveis ao processo de planejamento automatizado.

7.2 Pontos favoráveis

- A arquitetura de predição de estados inválidos é uma abordagem viável para o INPE realizar planejamento automático de operação de voos em satélites, para garantir a segurança em executar planos válidos, devido o tempo de duração da operação dos satélites ser muito longe e a degradação do satélites ser inevitável no ambiente espacial;

- A predição de estados inválidos através do processo de classificação possibilita validar quais os estados do domínio são ou estão inválidos no instante de planejamento e permite que os planos inválidos sejam previstos antes de sua geração;

- A arquitetura foi desenvolvida levando em conta que existem inúmeras técnicas e linguagens de planejamento para solucionar problemas específicos, então foi utilizado um meta-modelo para representação dos dados, que permite que a implementação seja independente de tecnologia e também que a tecnologia de planejamento aproveite algoritmos cada vez mais eficientes;

- No estudo de caso utilizou-se a linguagem PDDL para validar a arquitetura e demonstrar o estudo de caso, por ser uma linguagem de modelagem mais simplificada e por ser utilizada por inúmeros planejadores. A reutilização desta linguagem, para representar os estados inválidos, demonstra que outras linguagens também podem ser reutilizadas, desde que, o validador de estados seja implementado no planejador;

7.3 Pontos desfavoráveis

- Embora seja reutilizada a linguagem usada no planejador para representar os estados inválidos, modificar a implementação do planejador para incluir o método validador de estados, pode demandar uma codificação mais complexa dependendo da complexidade ou técnica utilizada;

- À medida que as restrições aumentam, o número de planos não encontrados também pode aumentar, conforme mostrado no resultado do último experimento deste trabalho, quando o estado inválido impede o planejador de encontrar a solução para o problema. Quando esse tipo de situação

ocorre, soluções paliativas como operar o domínio com suas funções reduzidas ou a criação de planos de contingência, tomando ações para alcançar as missões de outra maneira;

- A validação dos estados inválidos no planejamento é realizada nos estados conhecidos dentro da instância do domínio de planejamento utilizado. Ou seja, o domínio deve contemplar o estado inválido em sua instância, para que seja possível de ser validado, no entanto, qualquer estado inválido que possa surgir no domínio deve estar contido na definição de domínio;

7.4 Publicações

A aceitação de artigos oriundos deste trabalho em conferência internacional mostra o interesse da comunidade da área de planejamento nas ideias aqui implementadas, principalmente a contribuição para a segurança das atividades de controle dos satélites em órbita.

Conferências:

- **WETE 2018** (Workshop em Engenharia e Tecnologia Espaciais). Foi apresentado e publicado o artigo intitulado - Sistema baseado em conhecimento para definição de restrições para construção de domínios de planejamento para área espacial;

- **ICEIS 2020** (*International Conference on Enterprise Information Systems*) - QUALIS B2 . Foi apresentado e publicado o artigo intitulado - *State Validation in Automated Planning*, onde foi apresentado a abordagem de validação dos estados inválidos em tempo de planejamento.

Submissão:

- **IEEE SYSTEMS JOURNAL** - QUALIS A2, ENGENHARIAS III. Foi submetido o artigo *Automated Planning With Invalid States Prediction*, com o resultado final deste trabalho.

7.5 Trabalhos futuros

Como trabalho futuro, pode-se usar técnicas para atender aos objetivos parciais do plano, combinadas com a validação de estados inválidos. Usar o recurso de estados inválidos para melhorar o desempenho e o tempo de planejamento, bem como o aprendizado, pode contribuir para melhorar o desempenho, pois o planejador elimina caminhos de pesquisa ao encontrar estados inválidos.

Implementação da arquitetura com diferentes linguagens e técnicas de planejamento para avaliar qual abordagem apresenta o melhor desempenho para predição de estados inválidos e consequentemente ter ganho de mais recursos com implantação desse conceito em outras linguagens mais poderosas.

Tendo em vista que nos domínios reais existem uma grande taxa de mudanças e alterações nas definições de domínio e problema de planejamento, os estados inválidos podem ser usados também para dimensionar o escopo de domínios, já que podem limitar ou abranger mais caminhos na solução dos planos.

REFERÊNCIAS BIBLIOGRÁFICAS

ABDELGHAFAR, S.; DARWISH, A.; HASSANIEN, A. E.; YAHIA, M.; ZAGHROUT, A. Anomaly detection of satellite telemetry based on optimized extreme learning machine. **Journal of Space Safety Engineering**, v. 6, n. 4, p. 291–298, 2019. 2

ALHOSSAINI, M.; BECK, J. C. Instance-specific remodelling of planning domains by adding macros and removing operators. In: TENTH SYMPOSIUM OF ABSTRACTION, REFORMULATION, AND APPROXIMATION. **Proceedings...** [S.l.], 2013. 11, 13

AMIGONI, F.; GUALANDI, S.; MENOTTI, D.; SANGIOVANNI, G. A multiagent architecture for controlling the palamede satellite. **Web Intelligence and Agent Systems: An International Journal**, v. 8, n. 3, p. 269–289, 2010. 1, 20, 21, 22, 23, 25

ARECES, C.; BUSTOS, F.; DOMÍNGUEZ, M. A.; HOFFMANN, J. Optimizing planning domains by automatic action schema splitting. In: INTERNATIONAL CONFERENCE ON AUTOMATED PLANNING AND SCHEDULING. **Proceedings...** [S.l.], 2014. 11, 13

AZEVEDO, D. R.; AMBRÓSIO, A. M.; VIEIRA, M. Applying data mining for detecting anomalies in satellites. In: IEEE. **2012 Ninth European Dependable Computing Conference**. [S.l.], 2012. p. 212–217. 2

BAIER, J. A.; FRITZ, C.; BIENVENU, M.; MCILRAITH, S. A. Beyond classical planning: procedural control knowledge and preferences in state-of-the-art planners. In: AAAI CONFERENCE ON ARTIFICIAL INTELLIGENCE. **Proceedings...** [S.l.], 2008. p. 1509–1512. 1, 11, 12

BARREIRO, J.; BOYCE, M.; DO, M.; FRANK, J.; IATAURO, M.; KICHKAYLO, T.; MORRIS, P.; ONG, J.; REMOLINA, E.; SMITH, T. et al. Europa: A platform for ai planning, scheduling, constraint programming, and optimization. **4th International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS)**, 2012. 14, 15

BECKER, K. **AI Planning with STRIPS Copyright (c) Koly Becker**. 2015. Disponível em: <https://github.com/primaryobjects/strips>. 39

BOTEA, A.; ENZENBERGER, M.; MÜLLER, M.; SCHAEFFER, J. Macro-ff: improving ai planning with automatically learned macro-operators. **Journal of Artificial Intelligence Research**, v. 24, p. 581–621, 2005. 13

BOUTILIER, C.; DEAN, T.; HANKS, S. Decision-theoretic planning: structural assumptions and computational leverage. **Journal of Artificial Intelligence Research**, v. 11, p. 1–94, 1999. 12

BRIEL, M. V. D.; SANCHEZ, R.; DO, M. B.; KAMBHAMPATI, S. Effective approaches for partial satisfaction (over-subscription) planning. In: PROCEEDINGS OF THE 19TH NATIONAL CONFERENCE ON ARTIFICAL INTELLIGENCE. **Proceedings...** [S.l.], 2004. p. 562–569. 11, 12

CARDOSO, L.; FERREIRA, M.; ORLANDO, V. An intelligent system for generation of automatic flight operation plans for the satellite control activities at inpe. In: SPACEOPS CONFERENCE. **Proceedings...** [S.l.], 2006. p. 5575. 1, 16, 24

COLES, A.; SMITH, A. **Marvin: Macro-actions from reduced versions of the instance**. University of Strathclyde, 2004. 24–26 p. Disponível em: <https://nms.kcl.ac.uk/andrew.coles/publications/publication84.pdf>. 11, 13

CRUZ, C. G. R.; FERREIRA, M. G. V.; SILVA, R. R. State validation in automated planning. In: INTERNATIONAL CONFERENCE ON ENTERPRISE INFORMATION SYSTEMS. **Proceedings...** [S.l.], 2020. p. 396–406. 31, 71

DEMŠAR, J. et al. Orange: data mining toolbox in python. **the Journal of Machine Learning Research**, v. 14, n. 1, p. 2349–2353, 2013. 56, 60, 63, 65

DOMSHLAK, C.; HÜLLERMEIER, E.; KACI, S.; PRADE, H. **Preferences in AI: an overview**. [S.l.]: Elsevier, 2011. 11

DOMSHLAK, C.; KARPAS, E.; MARKOVITCH, S. To max or not to max: online learning for speeding up optimal planning. In: TWENTY-FOURTH AAAI CONFERENCE ON ARTIFICIAL INTELLIGENCE. **Proceedings...** [S.l.], 2010. 11, 13

EUROPEAN COOPERATION FOR SPACE STANDARDIZATION. **ECSS-E-70: Space Engineering - Ground Systems and Operations - Part 1: Principles and Requirements, ESA-ESTEC, Requirements & Standards Division**. Noordwijk, The Netherlands, abr. 2000. 6

FIKES, R. E.; NILSSON, N. J. Strips: a new approach to the application of theorem proving to problem solving. **Artificial Intelligence**, v. 2, n. 3-4, p. 189–208, 1971. 7

FOX, M.; HOWEY, R.; LONG, D. Validating plans in the context of processes and exogenous events. In: AAAI CONFERENCE ON ARTIFICIAL INTELLIGENCE. **Proceedings...** [S.l.], 2005. v. 5, p. 1151–1156. 8

FRANK, J. Using data mining to enhance automated planning and scheduling. In: IEEE. **2007 IEEE Symposium on Computational Intelligence and Data Mining**. [S.l.], 2007. p. 251–260. 13, 14, 25

FRANK, J.; JÓNSSON, A. Constraint-based attribute and interval planning. **Constraints**, v. 8, n. 4, p. 339–364, 2003. 14, 24

GALAL, M. A.; HUSSEIN, W. M.; KAWY, E. E.; SAYED, M. M. Satellite battery fault detection using naïve bayesian classifier. In: IEEE AEROSPACE CONFERENCE. **Proceedings...** [S.l.], 2019. p. 1–11. 2, 65

GEREVINI, A.; LONG, D. **Plan constraints and preferences in PDDL3**. [S.l.: s.n.], 2005. 2, 8, 9, 10, 12

_____. Preferences and soft constraints in pddl3. In: ICAPS WORKSHOP ON PLANNING WITH PREFERENCES AND SOFT CONSTRAINTS. **Proceedings...** [S.l.], 2006. p. 46–53. 8, 11, 12

GHALLAB, M.; NAU, D.; TRAVERSO, P. **Automated planning: theory and practice**. [S.l.]: Elsevier, 2004. 6, 39, 46

GUPTA, N. et al. Complexity results for blocks-world planning. In: AAAI CONFERENCE. **Proceedings...** [S.l.], 1991. v. 91, p. 629–633. 39

HASLUM, P.; JONSSON, P. Planning with reduced operator sets. In: AIPS CONFERENCE. **Proceedings...** [S.l.], 2000. p. 150–158. 13

IVO, A. A. S. **A tool for evaluating satellite flight plans using state models**. Thesis (MSc in Engineering and Management of Space Systems) — National Institute of Space Research (INPE), São José dos Campos, 2013. Disponível em: <http://urlib.net/rep/8JMKD3MGP7W/3FB2RJE?ibiurl.language=pt-BR>. 59, 60

JORGE, A. et al. Planning with preferences. **AI Magazine**, v. 29, n. 4, p. 25–25, 2008. 9, 11

KUCINSKIS, F. de N.; FERREIRA, M. G. V.; ARIAS, R. Increasing the autonomy of scientific satellites to deal with short-duration phenomena. In: IEEE AEROSPACE CONFERENCE. **Proceedings...** [S.l.], 2007. p. 1–12. 1, 5, 17, 18, 24

MCDERMOTT, D.; GHALLAB, M.; HOWE, A.; KNOBLOCK, C.; RAM, A.; VELOSO, M.; WELD, D.; WILKINS, D. **PDDL–the planning domain definition language–version 1.2**. 1998. Disponível em: <https://homepages.inf.ed.ac.uk/mfourman/tools/propplan/pddl.pdf>. 7, 8

MINING, O. D. **Orange Visual Programming Documentation**. 2020. Disponível em: <https://orange3.readthedocs.io/en/latest/>. 56, 63

NEBEL, B. On the compilability and expressive power of propositional planning formalisms. **Journal of Artificial Intelligence Research**, v. 12, p. 271–315, 2000. 8, 11

RABENAU, E.; DENIS, M.; JAYARAMAN, P. Implementation of a mission planning system for an interplanetary mission. In: SPACEOPS CONFERENCE. **Proceedings...** [S.l.], 2002. p. 07. 5

RIBEIRO, E. A proposal for an architecture to space mission planning in order to solve the problem of flight plan replanning using the technique of planning and scheduling in area of artificial intelligence. In: SPACEOPS CONFERENCE. **Proceedings...** [S.l.], 2013. p. 1275852. 1

RIBEIRO, E. A. **Uma Arquitetura de Agente de Replanejamento em Tempo Real para Plano de Voo em Missões Espaciais**. 2015. 224 p. Tese (Doutorado em Engenharia e Tecnologia Espaciais/Gerenciamento de Sistemas Espaciais) — Instituto Nacional de Pesquisas Espaciais (INPE), 2015. 19, 20, 25

RINTANEN, J. et al. Complexity of concurrent temporal planning. In: INTERNATIONAL CONFERENCE ON AUTOMATED PLANNING AND SCHEDULING. **Proceedings...** [S.l.], 2007. v. 7, p. 280–287. 8, 11

SILVA, R.; FERREIRA, M. G.; VIJAYKUMAR, N. An object-oriented meta-model as ontology for describing domains and problems for planning space

applications planning. In: SPACEOPS CONFERENCE. **Proceedings...** [S.l.], 2010. p. 2172. 1, 31, 53, 54

SOUZA, P. A mathematical model to predict operating states of satellites. In: SPACEOPS CONFERENCE. **Proceedings...** [S.l.], 2012. p. 1272995. 1, 6, 18, 19, 24, 25

SOUZA, P. B. de; FERREIRA, M. G. V.; SILVA, J. D. S. da. A tool for prediction of satellite future states. **Journal of Aerospace Computing, Information, and Communication**, v. 7, n. 12, p. 406–414, 2010. 63

STEELE, G. **Common LISP: the language**. [S.l.]: Elsevier, 1990. 8

TAFAZOLI, M. A study of on-orbit spacecraft failures. **Acta Astronautica**, v. 64, n. 2-3, p. 195–205, 2009. 2

THIÉBAUX, S.; HOFFMANN, J.; NEBEL, B. In defense of pddl axioms. **Artificial Intelligence**, v. 168, n. 1-2, p. 38–69, 2005. 8

TOMINAGA, J.; FERREIRA, M.; SILVA, J. A rule-based satellite simulator for use in flight operations planning. **Journal of Computational Interdisciplinary Sciences**, v. 2, n. 2, p. 111–121, 2011. 1

VALLATI, M. et al. The 2014 international planning competition: progress and trends. **Ai Magazine**, v. 36, n. 3, p. 90–98, 2015. 8

ANEXO A - PUBLICAÇÕES

A tabela A.1 mostra as publicações obtidas com este trabalho.

Tabela A.1 - Publicações

Nome	Qualis	Notas
WETE 2018	Conferência	Publicado
ICEIS 2020	Conferência	Publicado
IEEE ACCESS	B1	Aceito

9º Workshop em Engenharia e Tecnologia Espaciais
15 e 16 de Agosto de 2018

Sistema baseado em conhecimento para definição de restrições para construção de domínios de planejamento para área espacial

CRUZ, C. G. R.[1 2], **FERREIRA, M. G. V.**[1], **SILVA, R. R.**[3 4]

[1] Instituto Nacional de Pesquisas Espaciais, São José dos Campos, SP, Brasil

[2] Mestrando em Engenharia e Gerenciamento de Sistemas Espaciais - CSE

[3] FATEC Mogi das Cruzes, Faculdade de Tecnologia do Estado de São Paulo, SP, Brasil

[4] CISUC – Centro de Informática e Sistemas da Universidade de Coimbra, Coimbra, Portugal

caio.cruz@inpe.br

Resumo. *O crescente aumento de sistemas automatizados nas missões espaciais tem gerado preocupação com segurança e confiabilidade nas operações realizadas com satélites. Para prevenir a geração de estados inseguros às operações, estratégias foram criadas para validar planos de operação de voo gerados a partir de planejadores automáticos. As pesquisas demonstraram que estados inseguros podem ser gerados. Verificar se é possível remover estados inseguros de um plano a partir de mudanças na definição de domínio de planejamento é o objetivo desta pesquisa. Neste trabalho foi realizado uma análise que comprovou ser possível impedir que um plano contenha um estado inseguro, a partir da modificação de precondições e efeitos das ações do domínio. Assim, após definir a hipótese de que é possível modificar domínios que possibilitem que planejadores automáticos gerem planos sem estados inseguros, foi proposto uma estratégia para modificar domínios de planejamento utilizando-se de uma base de conhecimento de operações de voo.*

Palavras-chave: Planejamento Automático; Definição de Domínios; PDDL.

1. Introdução

O Centro de Rastreio e Controle de Satélites (CRC) no INPE, realiza as operações de controle de satélites através de atividades executadas na maioria dos casos de forma manual [TOMINAGA, 2011]. Encontrar soluções para automatização das operações de voo é um desafio em aberto.

Os trabalhos desenvolvidos entorno de planejadores automáticos limitam-se à criação de Plano de Operação de Voo, porém, sem fazer uso de informações históricas sobre as operações anteriormente realizadas nos satélites, deixando a cargo dos simuladores e geradores de diagnósticos a função de validarem se de fato o plano gerado automaticamente corresponde à realidade que o satélite está sujeito, no qual em muitos casos é rejeitado [SOUZA, 2011]

Os planejadores automáticos são usados para encontrar a sequência de passos que transitam um estado inicial no estado objetivo a partir de ações definidas no domínio de planejamento [KNOBLOCK, 1998]. É importante que a definição de domínio contenha apenas ações que gerem estados válidos, seguros e aplicáveis ao contexto em que o domínio faz parte, portanto, um domínio pode ser falho quando ocasionar a transição por um estado infactível.

Existe grande preocupação com a utilização de sistemas automatizados na operação de satélites, como garantir a segurança e confiabilidade nas operações [TOMINAGA, 2011]. No trabalho de SOUZA, foi proposto uma estratégia para validar planos de operação de voo, que considera um plano rejeitado caso um estado classificado como inseguro à missão compor o plano. Apesar de o planejador encontrar o estado objetivo, um novo plano deverá ser gerado utilizando-se de ações diferentes para obter o estado objetivo com outros passos sem a passagem por estados inseguros.

A validação de planos demonstra a importância da geração de planos válidos para as operações [SOUZA, 2011]. Neste contexto, prevenir que estados inseguros não sejam gerados no plano pode depender da definição de domínios de planejamento mais adequados. Dessa forma, o objetivo desta pesquisa é verificar se alterações feitas no domínio de planejamento podem contribuir para a criação de planos com maior validade, substituindo estados inválidos da sequência de passos planejados automaticamente.

2. Metodologia

O planejador automático AI Planning with STRIPS [BECKER, 2015] foi utilizado na execução do planejamento realizado neste trabalho. O problema do mundo dos blocos foi escolhido por ser um problema comumente utilizado em pesquisas entorno de planejamento, que consiste em empilhar blocos sobre uma mesa [KNOBLOCK, 1998].

A Listagem 1 mostra a definição do problema do mundo dos blocos no formato da Linguagem para Definição de Domínios de Planejamento (PDDL). O problema é composto de um estado inicial contendo três blocos empilhados sobre a mesa *t1* e outras duas mesas *t2* e *t3* respectivamente. O estado objetivo é a disposição da pilha de blocos em cima da mesa *t3*, seguindo a ordem do bloco *a* sobre o bloco *b*, bloco *b* sobre o bloco *c* e bloco *c* sobre a mesa *t3*.

```
1  (define (problem stack-blocks-stacked-cba-from-table1-to-stacked-abc-table3-onepilepertable)
2      (:domain blocksworld)
3    (:init (and (block a) (block b) (block c) (table t1) (table t2) (table t3)
4           (on a t1) (on b a) (on c b) (clear c) (clear t2) (clear t3)))
5    (:goal (and (on a b) (on b c) (on c t3)))
6  )
```

Listagem 1 - Definição do problema do mundo dos blocos

A Listagem 2 apresenta a definição de domínio com as ações possíveis para gerar o plano. As ações definidas no domínio são: *move* – move um bloco de uma mesa para outra, *stack2* – empilha dois blocos sobre uma mesa, *stack3* – empilha três blocos sobre uma mesa, *unstack2* – desempilha dois blocos e *unstack3* – desempilha três blocos.

```
(define (domain blocksworld)
  (:requirements :strips)
  (:action move
     :parameters (?b ?x ?y)
     :precondition (and (block ?b) (table ?x) (table ?y) (on ?b ?x) (clear ?b)
     (clear ?y))
     :effect (not (on ?b ?x) (on ?b ?y) (clear ?x) not (clear ?y))
     )
  (:action stack2
     :parameters (?a ?x ?b ?y)
     :precondition (and (block ?a) (block ?b) (table ?x) (table ?y) (clear ?a)
     (clear ?b) (on ?a ?x) (on ?b ?y))
     :effect (and (on ?a ?b) not (on ?a ?x) not (clear ?b) (clear ?x))
     )
  (:action stack3
     :parameters (?a ?x ?b ?c ?y)
     :precondition (and (block ?a) (block ?b) (block ?c) (table ?x) (table ?y)
     (clear ?a) (clear ?b) (on ?a ?x) (on ?b ?c) (on ?c ?y))
     :effect (and (on ?a ?b) not (on ?a ?x) not (clear ?b) (clear ?x))
     )
  (:action unstack2
     :parameters (?a ?b ?x ?y)
     :precondition (and (block ?a) (block ?b) (table ?x) (table ?y) (on ?b ?x)
     (on ?a ?b) (clear ?a) (clear ?y))
     :effect (and (on ?a ?y) not (on ?a ?b) (clear ?b) (clear ?a) not
     (clear ?y)))
  (:action unstack3
     :parameters (?a ?b ?c ?x ?y)
     :precondition (and (block ?a) (block ?b) (block ?c) (table ?x) (table ?y)
     (on ?c ?x) (on ?b ?c) (on ?a ?b) (clear ?a) (clear ?y))
     :effect (and (on ?a ?y) not (on ?a ?b) (clear ?b) (clear ?a) not
     (clear ?y)))
)
```

Listagem 2 - Domínio do mundo dos blocos

Após realizar o planejamento do problema com a definição de domínio do mundo dos blocos foi escolhido um estado entre os estados de transição do plano gerado para representar um estado inseguro. Desse modo, o estado em que o bloco *a* está sobre o bloco *b* e o bloco *b* está sobre a mesa *t2*, deve ser removido dos estados de transição. A ação que gera este estado é a *stack2,* onde o efeito define que dois blocos serão empilhados.

Afim de alterar a definição de domínio para impedir que o estado inválido faça parte do plano foi modificada a ação *stack2*. Os parâmetros, precondições e efeitos foram alterados de modo que o bloco da primeira mesa não seja empilhado sobre o bloco da segunda mesa.

A Listagem 3 apresenta uma nova definição de domínio contendo a ação *stack2* modificada, a nova precondição para aplica-la é o estado contendo três mesas, três blocos, um bloco sobre cada uma das mesas e o novo efeito gerado é de empilhar o bloco da mesa central sobre o bloco da terceira mesa.

```
(define (domain blocksworld)
  (:requirements :strips)
  (:action move
     :parameters (?b ?x ?y)
     :precondition (and (block ?b) (table ?x) (table ?y) (on ?b ?x) (clear ?b)
     (clear ?y))
     :effect (not (on ?b ?x) (on ?b ?y) (clear ?x) not (clear ?y))
     )
  (:action stack2
     :parameters (?a ?x ?b ?y ?c ?z)
     :precondition (and (block ?a) (block ?b) (block ?c) (table ?x) (table ?y)
     (table ?z) (clear ?a) (clear ?b) (clear ?c) (on ?a ?x) (on ?b ?y) (on ?c ?z))
     :effect (and (on ?b ?c) not (on ?b ?y) not (clear ?c) (clear ?y))
     )
  (:action stack3
     :parameters (?a ?x ?b ?c ?y)
     :precondition (and (block ?a) (block ?b) (block ?c) (table ?x) (table ?y)
     (clear ?a) (clear ?b) (on ?a ?x) (on ?b ?c) (on ?c ?y))
     :effect (and (on ?a ?b) not (on ?a ?x) not (clear ?b) (clear ?x))
     )
  (:action unstack2
     :parameters (?a ?b ?x ?y)
     :precondition (and (block ?a) (block ?b) (table ?x) (table ?y) (on ?b ?x)
     (on ?a ?b) (clear ?a) (clear ?y))
     :effect (and (on ?a ?y) not (on ?a ?b) (clear ?b) (clear ?a) not (clear ?y))
     )
  (:action unstack3
     :parameters (?a ?b ?c ?x ?y)
     :precondition (and (block ?a) (block ?b) (block ?c) (table ?x) (table ?y)
     (on ?c ?x) (on ?b ?c) (on ?a ?b) (clear ?a) (clear ?y))
     :effect (and (on ?a ?y) not (on ?a ?b) (clear ?b) (clear ?a) not (clear ?y))
     )
)
```

Listagem 3 - Domínio do mundo dos blocos modificado

A alteração feita na definição da ação *stack2* gerou um novo domínio no qual foi replanejado. Os resultados do planejamento entre o domínio original e o domínio modifico foram comparados e discutidos.

3. Resultados e Discussão

A seguir é apresentado o resultado da execução do planejamento para o domínio do mundo dos blocos. A execução do planejamento com a definição do domínio original, encontrou o estado objetivo a partir de um plano gerado com quinze ações. Os estados gerados através de cada ação do plano foram representados graficamente na Figura 1.

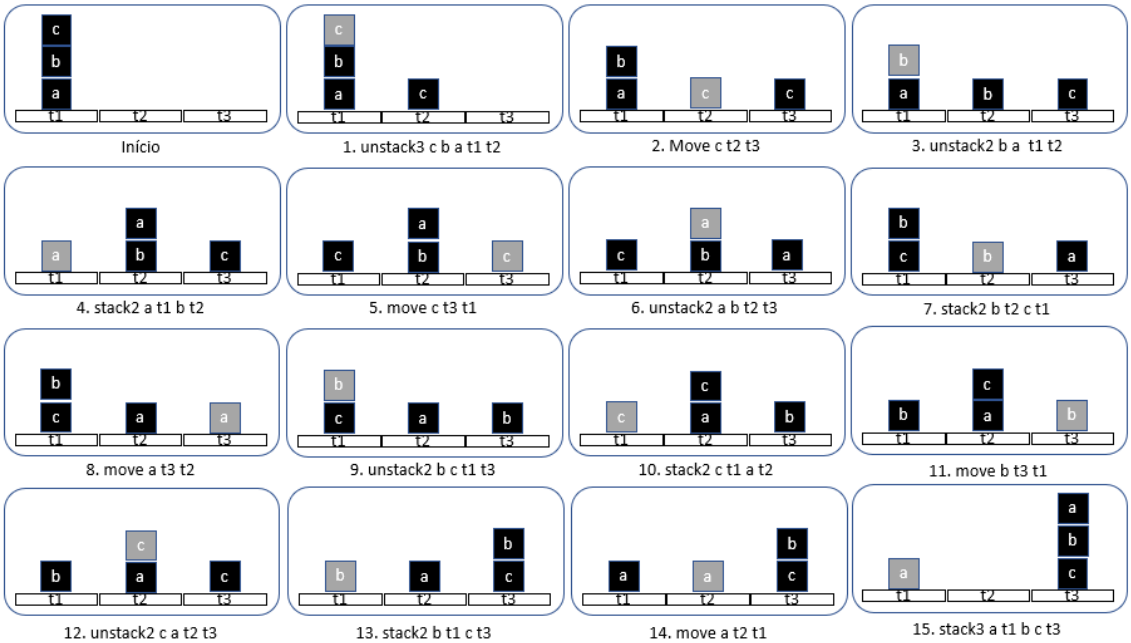


Figura 1 – Passo-a-passo da execução do plano 1

A Figura 2 apresenta o estado gerado a partir da execução da ação *stack2* que foi escolhido ao acaso (apenas como exemplo didático neste artigo) para representar um estado inválido. Assim supondo que para chegar ao estado objetivo é definido como restrição do problema do mundo dos blocos utilizado no trabalho, o estado objetivo do plano seja atingido sem passar por um estado em que o bloco *a* esteja sobre o bloco *b* e o bloco *b* sobre a mesa *t2*.

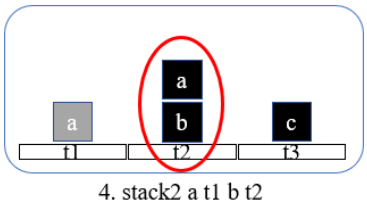


Figura 2 - Estado inválido

A execução do planejamento após a alteração da ação *stack2* (conforme apresentado na Listagem 3) é apresentado graficamente na Figura 3. O estado objetivo foi encontrado a partir de seis ações. As modificações demonstraram que o plano gerado encontrou o estado objetivo sem passar pelo estado definido como inválido.

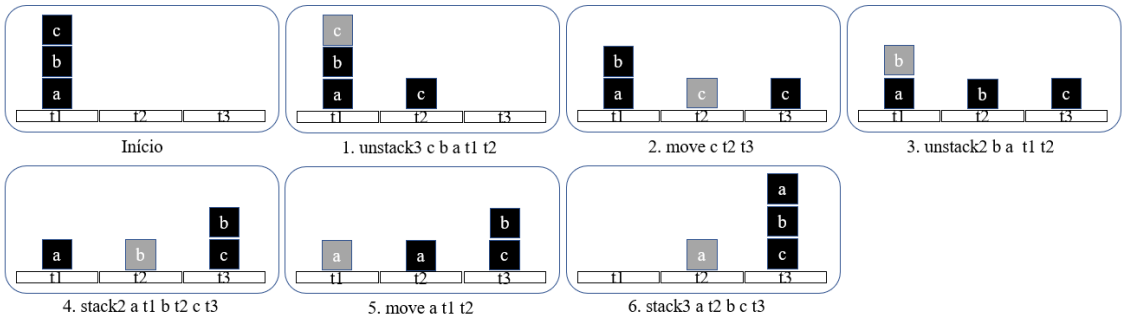


Figura 3 - Passo-a-passo da execução do plano 2

Ainda é possível observar que neste exemplo além de um estado inválido não estar contido no plano foi possível atingir o estado objetivo com um menor número de passos. Porém, acredita-se que este cenário não seja repetido para todos possíveis casos, assim pode haver situações onde um maior número de passos seja necessário para que um estado objetivo seja atingido após a alteração de ações do domínio de planejamento.

A Figura 4 apresenta a comparação entre o estado (B) gerado no primeiro plano e o estado (C) gerado após a alteração da ação *stack2* (Listagem 3). No domínio original a ação *stack2* considerava somente quatro parâmetros, sendo dois blocos e duas mesas, porém, o efeito era empilhar o bloco *a* sobre o bloco *b*, gerando o estado (B). Após a inclusão de novos parâmetros à ação, representando três blocos e três mesas, o efeito gerou o estado (C).

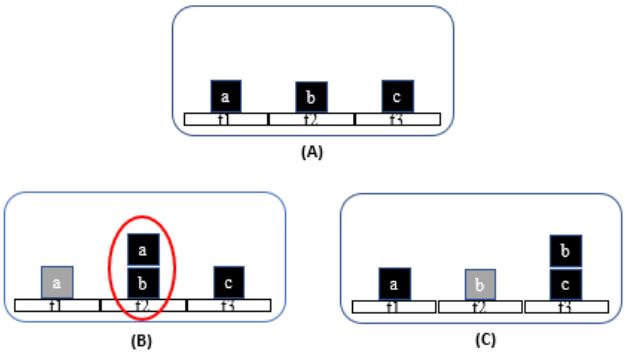


Figura 4 - Comparação dos estados antes e depois

O resultado obtido a partir das alterações feitas no domínio de planejamento, comprovou que modificar ações de um domínio pode impedir que estados inseguros façam parte dos planos gerados automaticamente. Portanto, uma estratégia para construção automática de domínios de planejamento, pode auxiliar na criação de planos com maior validade para o domínio.

A

Figura 5 apresenta a proposta de um sistema baseado em conhecimento para definição de restrições para a construção de domínios de planejamento. O gerador de domínios é o modulo responsável por modificar domínios de planejamento, adicionando parâmetros, precondições e efeitos à definição das ações, a partir do conhecimento de estados inválidos obtido através de uma base de conhecimento.

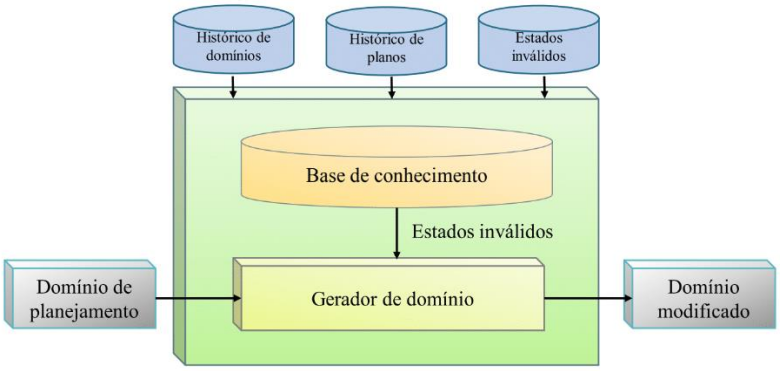


Figura 5 - Sistema baseado em conhecimento para definição de restrição de domínios de planejamento

O sistema baseado em conhecimento define uma estratégia para a geração de domínios de planejamento usando predições feitas sobre estados inseguros às operações de satélites. Construir uma base de conhecimento com predição de estados inseguros e converter em restrições de domínio de planejamento é o objetivo desta estratégia.

A base de conhecimento é o modulo responsável por disponibilizar informações importantes sobre os estados do domínio. Os estados inválidos nas operações com satélites são desconhecidos devido à inúmera quantidade de dados sobre a operação. A estratégia para validação de planos de operação de voo proposta por SOUZA demonstrou que é possível predizer estados dos satélites usando técnicas de Mineração de Dados [SOUZA, 2011]. A classificação de estados inválidos pode identificar quando um estado é inseguro. Portanto, uma arquitetura composta por um sistema capaz de encontrar informações desconhecidas sobre domínio permitirá a criação de planos válidos sem que um plano seja rejeitado posteriormente.

Por que esta estratégia seria importante se um especialista já modelou o domínio com as restrições? A área de operação de satélites possui um histórico em que podem haver uma infinidade de estados inválidos desconhecidos que humanamente seria impossível de ser modelado, porém um sistema que saberá ler todo esse volume e traduzir em domínios válidos.

4. Conclusão

A inclusão de novas restrições ao domínio de planejamento permite alterar ações de um plano gerado automaticamente. As mudanças no domínio de planejamento demostraram que

é possível impedir que um plano contenha ações que gerem um estado inválido, por exemplo, possibilitando que o objetivo do plano seja alcançado sem a passagem por estados inválidos. Os problemas com planejamento automático na área espacial podem ser evitados quando restrições adequadas são incluídas ao domínio de planejamento. Um plano da área espacial é recusado quando ações inseguras à missão tiverem que ser executadas para que o objetivo do plano seja alcançado. Portanto, impedir que ações impróprias sejam executadas depende do conhecimento de restrições sobre o domínio.

A predição de estados dos satélites é importante para validar um plano, assim como, a predição de restrições de domínio pode ser o caminho para a criação de planos mais adequados no contexto da operação de satélites. Conclui-se que para gerar planos sem a passagem por estados inválidos é importante que as ações modeladas sobre o domínio de planejamento contenham precondições e efeitos condizentes apenas com estados válidos.

Referências

[KNOBLOCK, 1998] Knoblock, C., Barrett, A., Christianson, D., Friedman, M., Kwok, C., Golden, K., … Weld, D. (1998). PDDL | The Planning Domain Definition Language 1 Introduction. *Most*.

[SOUZA, 2011] Souza, Primavera Botelho De. (2011) A Classification Model to Generate Prognosis of Satellite.

[TOMINAGA, 2011] Tominaga, J., Ferreira, M., Silva, J. (2011) A rule-based satellite simulator for use in flight operations planning.

[BECKER, 2015] Becker, K. AI Planning with STRIPS. Copyright (c) 2015 Kory Becker disponível em https://github.com/primaryobjects/strips

State Validation in Automated Planning

Caio Gustavo Rodrigues da Cruz[1][a], Mauricio Goncalves Vieira Ferreira[1][b]
and Rodrigo Rocha Silva[2,3][c]

[1]*Space Engineering and Technology, National Institute for Space Researches (INPE), São José dos Campos, Brazil*
[2]*Centre for Informatics and Systems, University of Coimbra (CISUC), Coimbra, Portugal*
[3]*FATEC Mogi das Cruzes, São Paulo Technological College, Mogi das Cruzes, Brazil*

Keywords: Planning Preference, Invalid States, PDDL.

Abstract: The crescent number of automated systems in satellites raises several security and reliability concerns, that are worsened with the time. Plan validation techniques were created to validate flight operation plans generated automatically. The execution of automatically generated plans on satellite flight operations can result in degraded or invalid states. Verifying the possibility of removing these states of a plan through a state validation technique is the objective of this paper. Analyzing the action that generated and, in planning time, remove the invalid states from the plan steps enables the planner to find the final state without any invalid state. Therefore, implementing a state validator in the automated planner prevents the plan from containing any invalid state.

1 INTRODUCTION

The concept of plan quality in automated planning is a very important issue. In several real-world planning domains, we must address problems with a large set of solutions, or with a set of goals that cannot be completely achieved. Besides, in the expected solutions there may be constraints during the trajectory, in intermediary states of final state. In these cases, the mode how the goal is reached can be more important than the goal itself. For this reason, it is important to generate plans of better quality achieving all problem goals, when possible, or some subset of them (Baier et al., 2008).

Automated planning is an AI area focused on solving problems. A planning domain is comprised of a finite set of possible states $S = \{s_1, s_2 ... s_k\}$, and a finite set of actions $A = \{a_1, a_2 ... a_k\}$, applicable to the domain states (Ghallab et al., 2004). A planning problem is originated when the need to transform an initial state s_i in a set of final states S_g. The action sequence $< a_1, a_2, ... a_k >$, when applied in the results of the order in the end state is called plan. A description of the state transition system used in automated planning follows.

$$s_1 = \gamma(s_i, a_1),\ s_2 = \gamma(s_1, a_2), ...,\ s_k = \gamma(s_{k-1}, a_k)$$
$$and\ s_k \in S_g$$

The function $\gamma(s,a)$ applies an action to a state, resulting in a state transition system. For each generated state, an action is applied until the generated state is corresponds to the final state. A problem can be solved in countless ways i.e., infinite step sets can transform the initial state in the final state (Ghallab et al., 2004).

Automated planning is achieved using a planner software that finds the step sequence that transforms the initial state in the goal state (McDermott et al., 1998). The planner uses two files as input in order to solve a problem: The domain file, that defines the applicable actions, and the problem file, that defines the initial and final states.

STRIPS (Stanford Research Institute Problem Solver) is a technique used to find solutions from a domain and a problem (Fikes and Nilsson, 1971). STRIPS goes through all the possible states after applying the domain actions until it finds the final state (Fikes and Nilsson, 1971). The most common language in automated planning is PDDL (Planning Domain Definition Language) introduced in 1998 by Drew McDermott (McDermott et al., 1998).

The theme of planning is applied to many real-world domains and issues. In the space area flight plan generation for the control of artificial satellites is an example of the planning application. In most

[a] https://orcid.org/0000-0001-6143-3908
[b] https://orcid.org/0000-0002-6229-9453
[c] https://orcid.org/0000-0002-5741-6897

Rodrigues da Cruz, C., Ferreira, M. and Silva, R.
State Validation in Automated Planning.
DOI: 10.5220/0009411903960406
In *Proceedings of the 22nd International Conference on Enterprise Information Systems (ICEIS 2020) - Volume 1*, pages 396-406
ISBN: 978-989-758-423-7

cases, INPE's Satellite Tracking and Control Center (STCC) performs its satellite control operations manually. Finding solutions to flight operation automatization is a challenge that can be solved with planning (Tominaga et al., 2011).

Concern over the generation of higher quality satellite control plans in research such as Souza's, suggests the creation of a diagnostic generator to validate whether the automatically generated s match the situation of the satellites in operation. The approach considers a rejected plan if a state classified as unsafe for the mission compose the plan of the state sequence, so a new plan should be generated using different steps to reach the goal state. The rejection of a plan implies the generation of a new plan, with different steps but still reaching the end state (Souza et al., 2012). In this context, preventing some steps of being generated in the plan is a challenge to the satellite flight operation planning domain.

The flight operation planning for satellites is complex to be solved with classical planning techniques alone, because they are not considered to be constraints on the transition states of a plan. This deficiency open ways for many surveys' themes in AI planning, for example, the planning with constraints, with user preferences, of complex problems and about uncertainty. These themes are found in literature in works that create new planning languages, techniques or implement planners to meet specific constraints of the planning problem.

How seen if the use of the classical planning based in STRIPS been used to solve these problems, the generate of plans without consider constraints in satellite domain, can create invalid plans. An approach that eliminate specific states of solution planned of automatic form is the motivation this work for solve the problem in question.

The goal is to propose a solution based on classical planning that incorporates and considers at the time of planning states that are degraded and should not be part of the plan state sequence. In this paper, it is proposed to create a new method in a STRIPS-based scheduler that validates the states at planning time. The proposal creates a filter of states that cannot compose the solution. Thus, contributing to valid plans can be automatically generated using a planner. The strategy in this paper is how to find a valid plan in classic planning.

Our intention to show that the automatic planning of satellite plans should be concerned specifically with the states that make up a plan. And that classical planning can be used to solve this kind of problem if you know the states that the plan should avoid. Our approach envisions mapping these states and incorporating them into the planner.

In works found in the literature, the creation of new languages is common to solve more complex planning problems. Most of these works are related to planning with preferences, which is an area that has been extensively studied in recent years. In the related works session, we present the works with different techniques and planning methods that were proposed in different areas, to create increasingly better solutions in different domains.

About the solution: In this article we will demonstrate a way to generate step constrained plans using as an example a didactic planning problem to validate the implementation of a validator method in a planner.

In the first step prove that at planning time it is possible to disregard the degraded states by creating a new input in the planning domain. The new entry will be read and used in the planner to build a solution that meets the constraints required by the domain. In future research a model will be created to represent the states and convert them to a planner entry.

The rest of this paper is structured as follows: Section 2 describes the methodology used; Section 3 presents the results of the tests solving the automated planning for the blocks world problem; Finally, section 4 presents the conclusions and some future work.

2 RELATED WORK

We found different works on AI planning that address issues such as quality plan generation, complex planning problems, uncertainty planning and user preferences in planning. Among these approaches a commonly used term is preferences, an interdisciplinary topic found not only in AI, but in studies with different perspectives and areas (Domshlak et al., 2011).

The works found on planning that address the theme of preferences are (Boutilier et al., 1999) (Gerevini and Long, 2006) (Tu et al., 2007) (Baier et al., 2008) (Sohrabi et al., 2009). Among these works are several approaches such as planner development, language creation, implementation of techniques in existing planners, extension of planning languages and combination of techniques are used to meet preferences in the planning context.

PBP preference-based planning aims to find more preferred plans in a planning instance. Criteria are provided to determine when one plan is more preferred than another. Preferences are modeled according to language type and can be either quantitative or qualitative. In order to compare when a plan is preferred in the quantitative approach a numerical function is used to an ever-induced overall relationship.

In qualitative language the comparison is in terms of property without number assignment. There are also approaches with the combination of qualitative and quantitative languages (Jorge and Sheila, 2008).

The search for the construction of ideal or near optimal plans is a theme addressed in Boutilier's work. Decision-Theoretic planning uses the Markov Decision Process (MDP) to explore policy making and idea plans. MDP associates a reward function with each state transition, thereby defining user preferences. All possible states are classified quantitatively, and an action is returned depending on the execution history (Boutilier et al., 1999).

The approach used in Partial Satisfaction Planning offers resources to partially solve problems, reaching a subset of objectives. To partially solve the planning problems, techniques based on heuristics were developed. The techniques used are concerned with the quality of the plan, contributing to generate plans with low cost and compatible with the quality of plans from other approaches (Briel et al., 2004).

In the work of Baier et al (2008) a method was created to compile a planning instance and a control procedure into a classic planning instance represented in PDDL. The compilation allows to represent in the planning domain the procedure as a finite state automaton. The representation is made from an additional predicate that modifies the effects and preconditions of the actions, allowing the procedure to be respected (Baier et al., 2008). For planning with preferences Baier et al (2008) proposes to use the relationship between linear temporal logic and automata. The temporal LPP language is used to express preferences through temporal properties of states and actions by qualitatively classifying expressions (Baier et al., 2008).

2.1 The PDDL Language

Based on Lisp syntax, the PDDL LANGUAGE uses a structure based on the widely used variants of strips notations. Establishing a common standard language has had a similar impact on planning research as the introduction of standards in other areas of research: it opens the route to stronger collaboration, exchange of tools, techniques and problems and provides a platform for comparative evaluation of approaches. The language has been, since the beginning, strongly linked to the competition series, with developments in the language being drivers for the direction of the competition challenges.

PDDL has been extended in several stages in order to capture more expressive variants. There have been several explorations of the expressive power of the different variants of PDDL. Recent results include a demonstration that temporal features can be compiled away in polynomial work, subject to certain constraints on the forms of concurrency that can appear in the problem (Rintanen, 2007), while others have examined the compilability of conditional effects, timed initial literals and domain axioms (Nebel, 2000), (Fox et al., 2004), (Thiébaux et al., 2005).

In reference (Gerevini and Long, 2005) and (Gerevini and Long, 2006) extended the PDDL language to a PBP language. PDDL3 uses Hierarchical Task Network (HTN) to include up to three types of preferences, increasing the expressive power over the plan's quality specification. The first is the ability to express goals that apply not only to the final state of the trajectory of states visited by a plan, but also to the intermediate states. These goals take the form of trajectory constraints, familiar from work on temporal logics.

Both extensions to the language are motivated by the desire to see planning bridge the gap between research and application. Many real problems require the specification of goals that are more complex than be easily expressed in earlier versions of PDDL. These include constraints on the states (or invalid state) that a plan visits as well as on the state in which it finishes. It can also be important to specify the relative benefits of different, perhaps conflicting, desirable conditions that a plan should satisfy, so that a plan might be constructed to evaluate these benefits against the costs of achieving them.

Table 1: Comparative table of approaches with preferences.

Approuch	Technique	Preference
PDDL3	Hierarchical Task Network	Violated preferences metric
MDP	Reward function	Classification based on the history of actions performed
PSP	Heuristics	Planning with subset of objectives
Control Procedure Method	Temporal Linear Logic and Automata	Additional predicate on the effects of actions

Table 1 presents a comparison between planning approaches with preferences. It is considered the technique used and how the preferences are models. It is understood that each approach includes preferences in planning differently and to meet specific requirements.

3 STRATEGY TO VALIDATE INVALID STATES IN PLANNING TIME

A hypothesis was created for automatic plan generation that are composed of steps that do not include any invalid states to achieve an objective. The hypothesis is that valid plans can be generated if invalid states are validated in the planner in planning time. Therefore, if the planner finds the end state using a step sequence that transits only through valid states in the domain, the strategy is valid.

The planner applies in the initial state the actions defined in the domain and creates a state tree while the actions are applied. However, in planning time, when an action is applied on the current state, the planner knows what is going to be the state that will be added to the state tree. The moment a domain action is applied to the current state, the new generated state can be validated before composing the state list that the planner uses to find the final state. The implementation of a state validator in the planner will enable the identification of generated invalid states in planning time.

A planning problem can be understood from the following representation: $P = (\Sigma, s_i, S_g)$, where Σ is the state transition system, s_i is the initial state and S_g is a set of the goal states. In the concept of classical planning, a plan is not deterministic and there may be different ways of finding the sequence of actions that transform the initial state S_i into the objective state S_g. Figure 1 represents a non-deterministic state transition system in which the objective state S_g can be achieved using different paths.

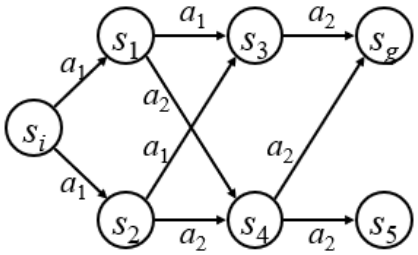


Figure 1: Non-deterministic state transition system.

The characteristic presented in the non-deterministic state transition system allows a planning problem to be solved using different paths. States can be easily ignored and the objective state can still be reached. Allowing thus to ignore states that should not be part of the solution, the planner will find paths without passing through such states.

What can set a state to invalid for a domain? In planning a state is composed of first order atoms that are represented as propositions. Objects that constitute the state can be presented as constant, variable, or function terms. However, in the planning domain it may contain some rule that invalidates a state according to the configuration of atoms, be it the location or actions that relate one or more atoms thus indicating that the state is invalid. In the satellite control domain, for example, the domain is constantly changing due to external environmental conditions that degrade satellite subsystems. When a degradation occurs a new invalid state is added to the domain. The invalid state is a prohibited, degraded or risky scenario for the domain's operation.

The example used in the experiment in this article restricts one of the cranes from unloading a specific container, thus representing an invalid state, when the "unloading" actions are associated with these two atoms.

For the generation of valid plans, a strategy was created so that invalid states about the domain are considered. The planner will have as its input the domain and problem files, as well as another file containing the definitions of invalid states for the domain in question as shown in the Figure 2. The invalid states defined as the new input will be used to validate the generated states in planning time, consequently, when a new state is generated, it will be compared with the invalid state list and the planner will then ignore the invalid state and won't add it to the solution tree.

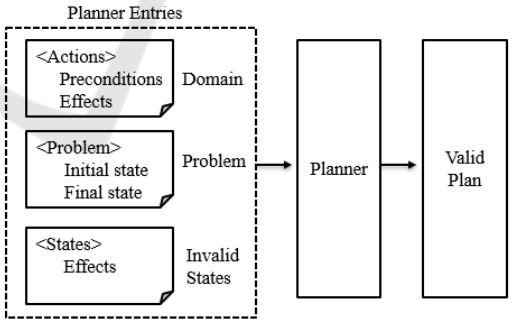


Figure 2: Represent strategy.

To formalize the implementation of the proposed strategy, we modified the concept of problem previously defined as $P = (\Sigma, s_i, S_g)$, to then use the function $V(s_k, I)$ responsible for validating and filtering the states in time, where s_k represents the current state and I represents the instance of invalid states previously configured as input to the planner. The modification transforms the standard function as follows $P = (\Sigma, s_i, S_g, V(s_k, I))$.

In order to validate our plan generation method,

we chose the automated planner AI Planning with STRIPS (Becker, 2015). This planner is a demo project for the STRIPS automated planner library written in NodeJS (Becker, 2015).

Initially we chose a simple planning problem to formulate an example of using the strategy to validate states. The blocks world problem is a classic planning problem, it consists in stacking blocks on top of tables in some arbitrary order (Gupta and Nau, 1991). Some actions of this domain are moving a block from one table to another, stacking two blocks and unstacking two blocks. After implementing the validator method, another planning problem was chosen to validate a scenario with a larger number of states.

The development of this work followed four stages: choosing the planning problem, generating the invalid state file in PDDL, modifying the planner to load the invalid states and implementing a state validation function in the planner.

3.1 Strategy Application in an Invalid State

The problem of the chosen block world consists of six objects, three blocks and three tables. The initial state has blocks *a*, *b* and *c* in this block *c* on block *b*, block *b* on block *a* and block *a* stacked-on top of table *t1* and other two tables, *t2* and *t3*, empty. The goal state is comprised of the blocks in the reverse order block *a* on block *b*, block *b* on block *c* and block *c* over table *t3*, as shown in the Figure 3.

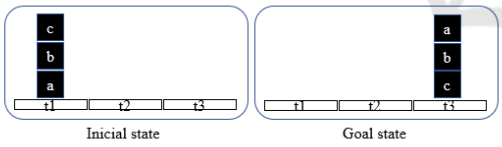


Figure 3: Planning problem.

The planner must find an action sequence that is able to transform the initial state in the goal state using only the actions defined in the problem domain, for instance: move – moves a block from one table to another; *stack2* – stacks a block over another block in another table; *stack3* – stacks a block over another two blocks over another table; *unstack2* – unstacks two blocks, putting the unstacked block over another, empty, table; *unstack3* – unstacks three blocks, putting the unstacked block over another, empty, table.

This problem was executed in the planner and a plan was used. A state transition that the planner encountered is shown in Figure 4. It can be viewed as a sequence of domain actions that were applied to states until the goal was reached. The first action applied was *unstack3*, responsible for unstacking three blocks, resulting in unstacking or block *c* from block *b* to a table *t3*.

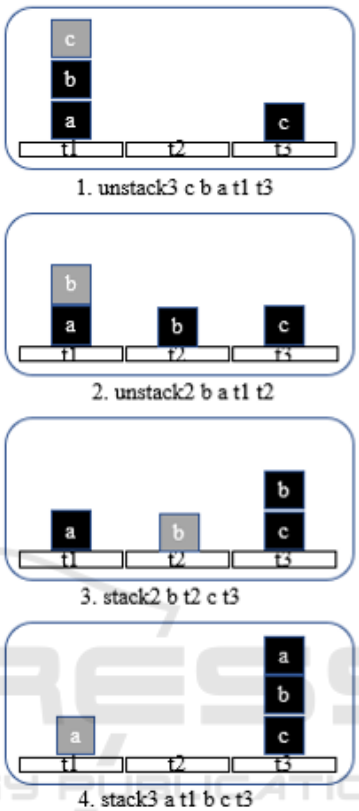


Figure 4: State transaction.

After representing the execution of the plan, we randomly choose one of the traversed states to represent an invalid state. The state chosen was that generated by applying action *2. Unstack2*. The chosen state was set in a new file written in PDDL as shown in Figure 5.

```
1  (define (domain invalidstates)
2    (:requirements :strips)
3    (:action invalid1
4       :parameters (?a ?b ?c ?t1 ?t2 ?t3)
5       :precondition (and (block ?a) (block ?b)
6             (block ?c) (table ?t1) (table ?t2)
7             (table ?t3))
8       :effect (and (on ?a ?t1) (clear ?a)
9             (on ?b ?t2) (clear ?b) (on ?c ?t3)
10            (clear ?c)))
11 )
```

Figure 5: Invalid state file written in PDDL.

The state was configured as a common action, using the same definitions of the domain file in PDDL. In the invalid1 (Figure 5, line 3) configuration we defined its parameters as its variables: three blocks and

three tables. Positioning and order are defined in the effects, as in block a over table *t1* - (on *?a ?t1*), block *a* free (clear *a*), block *b* over table *t2*- (on *?b ?t2*), block *b* free (clear *b*), block *c* over table *t3* - (on *?c ?t3*) e block *c* free (clear *c*).

The planner has a PDDL file reading function, receiving as its parameter the file path and loading the file in memory. We used the existing implementation to read the invalid state file. The *StripsManager.loadDomain()* (Figure 6, line 4) function was reused to load the new invalid state file.

Figure 6 shows our modifications to the load function: we added the *invalidStatePath* parameter in function.

```
1 load: function(domainPath, problemPath,
2         invalidStatePath, indice, callback, isCode) {
3
4   StripsManager.loadDomain(invalidStatePath,
5                            function(invalidState) {}
6
7 }
```

Figure 6: Invalid state file loading method.

After loading the invalid states, an object containing all the invalid states definitions was added to a new *invalidState* property in the domain object. The domain object is the main parameter for the planner methods, since it contains the actions used in the plan.

The code snipped where the planner applies the actions, generating new states is shown in Figure 7 The *getChildStates* method (Figure 7, line 1) is responsible for applying the actions to the current plan state. The method has as its parameters the domain definition and the current plan state.

```
1 getChildStates: function(domain, state) {
2   var children = [];
3   var actions =
4       StripsManager.applicableActions(domain, state);
5   for (var i in actions) {
6     var action = actions[i];
7
8     var newState = { state: StripsManager
9                     .applyAction(action, state),
10                    action: action };
11
12    if(!StripsManager.stateValidator(
13      domain.invalidState, newState)){
14      children.push(newState);
15    }
16  }
17  return children;
18 },
```

Figure 7: Generating method of new states.

How is an action applied to the current state The *applicableActions* method (Figure 7, line 4) iterates all the actions in the domain passed as parameter, verifying if their preconditions are met – i.e. the current state corresponds to the precondition. The quantity of applicable actions is the same as the number of generated states. For each action iterated a new state is generated by the *applyAction* (Figure 7, line 9) method. The effects of the execution of an action on a state are defined by the action effects, defined in the domain.

The method iterates the effect list and modifies the current state, adding and removing parts of the state according to the operation, e.g. the action *unstack2* is applied to the state (on *a t1*) (on *b a*) (on *c t3*) (clear *t2*) in the following manner: each part of the current state is compared to the effect parts defined in the *unstack2* action.

```
1 {
2   action: "on",
3   operation: "not",
4   parameters: ["b", "a"]
5 },
6 {
7   action: "on",
8   operation: "and",
9   parameters: ["b", "t2"]
10 },
11 {
12   action: "clear",
13   operation: "not",
14   parameters: ["t2"]
15 },
16 {
17   action: "clear",
18   operation: "and",
19   parameters: ["a"]
20 },
```

Figure 8: Data structure representing an effect.

Figure 8 shows the structure of the *unstack2* effect. Therefore, the method verifies each effect part and if the operation property is “and”, that part is added to the current state. If the operation is “not”, the part is removed from the current state. The action results in the state (on *a t1*) (on *b t2*) (on *c t3*) (clear *a*).

After generating a new state, the *stateValidator* method was included in the implementation to check if the generated state is a valid state. The *stateValidator* method receives as parameter the *invalidState* definition we included in the domain object and the generated *newState*. The validator returns true if the hew state is identical to any configured invalid state. Only valid states can be added to the child state list.

3.2 Implementing the Invalid State Validator

The planner invalid state validator function is shown in Figure 9 compares the states configured in the invalid state file and the states created in planning time to compose the plan.
The validator function is shown iterates over three loops in order to validate the states. The first loop goes through the list of invalid states configured in the

```
1  stateValidator: function(invalidState, newState) {
2    var equalState;
3    for (var h in invalidState.actions){
4      actions = invalidState.actions[h].effect;
5      for(var i in actions){
6        equalState = false;
7        for (var j in newState.state.actions) {
8          if(newState.state.actions[j].operation != "not" &&
9                  actions[i].action ==
10                 newState.state.actions[j].action &&
11                 StripsManager.arraysIdentical(
12                 actions[i].parameters,
13                 newState.state.actions[j].parameters)){
14           equalState = true;
15           break;
16         }
17       }
18       if(!equalState)
19         break;
20     }
21     if(equalState)
22       break;
23   }
24   return equalState;
25 }
```

Figure 9: State validator function.

file, since there can be more than one invalid state. The second loop iterates each of the invalid state's parts, e.g. the state (on *a t1*) (on *b t2*) (on *c t3*) is composed of three parts: in the first iteration, the part (on *a t1*) will be compared. The third loop iterates the new state's parts and compares them.

The data structure that abstracts a single part of a state comprises three properties. The action property (Figure 9, line 9) represents the association between the parameters defined in the parameter's property (Figure 9, line 12) and the operation parameter (Figure 9, line 8) indicates whether the relation exists in that state. When operation is not, the relationship doesn't exist and there is no need to compare the corresponding part with the invalid state part.

The state comparison is comprised of three conditions: if operation is not negative, if the action of both parts is the same and if both parts' parameters are identical. The *arraysIdentical* function is present in the planner implementation, responsible for comparing both states' parameters, verifying if their size and values are equal. Figure 9 shows a full implementation of the *stateValidator* function.

In the second loop (Figure 9, line 5), the *equalState* variable (Figure 9, line 6) is set as false, suggesting that the part wasn't found in the new state yet. In case the third loop doesn't find an equal part and, when it's finished, the *equalState* variable is (Figure 9, line 18) still false, the break command (Figure 9, line 19) is called, since there is no need to continue comparing the parts of that invalid state.

When the condition is met, there is a part of the invalid state in the new state. The *equalState* variable (Figure 9, line 14) is set as true and the break command (Figure 9, line 15) is called, exiting the inner loop and iterating over the next invalid state. In case there are no more parts in the loop and the *equalState* variable (Figure 9, line 21) is true, another break command (Figure 9, line 22) is called since there is no need to continue searching the invalid state list. Finally, the algorithm returns the value of *equalState* (Figure 9, line 24).

4 RESULTS

In this section we present the results obtained by testing the implementation of the state validator method in the planner. Planning was performed for several plans, which included invalid states to be tested in the planner.

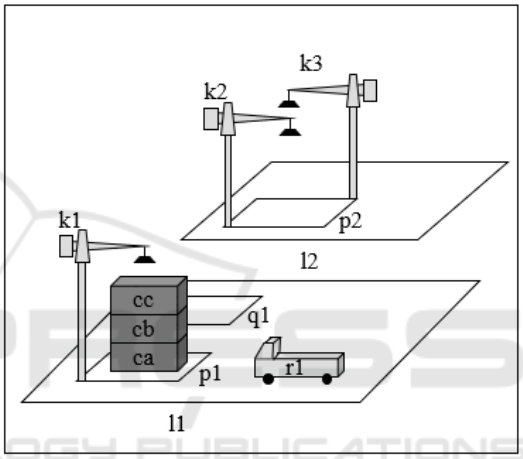


Figure 10: Dock worker robot problem.

We chose the dock worker robot problem for testing because it is a more complex problem than the block world and uses more objects and actions in the planning domain. The problem is to move three containers between two different locations, using cranes and a robot to transport them as shown in Figure 10.

The problem has two locations in the initial state *l1* and *l2*. At location *l1* there are two pile *p1* and *q1*, a crane *k1* and three containers *ca, cb* and *cc* stacked over pile *p1*. At location *l2* has a pile *p2* and two cranes *k2* and *k3*. The initial state configuration and arrangement of problem objects has been arranged to enable the inclusion of invalid states for the test.

The purpose of the problem is to move the containers to location *l2*. The domain file includes the following functions: *move* - moves a robot between two adjacent locations; *load* - loads an empty robot with a container held by a nearby crane; *unload* - unloads a robot holding a container with a nearby crane; *takes* - takes a container from a pile with a crane; *put* - puts a container held by a crane on a nearby pile.

The problem was executed in the planner and a

seventeen-step plan was generated to solve the problem. Table 2 shows the transition from initial state *Si* to objective status *Sg*, including the sequence of actions required to move containers from location *l1* to location *l2* in the plan I.

Table 2: Representation of the plan I states.

Action	State	Representation
Init	Si	Containers stacked on site 1
1. take k1 l1 cc cb p1	S1	Crane k1 grabbed the cc container
2. load k1 l1 cc r1	S2	Robot r1 was loaded with cc container
3. move r1 l1 l2	S3	The robot has been moved to location l2
4. unload k2 l2 cc r1	S4	The crane also unloaded the cc container
5. move r1 l2 l1	S5	Robot r1 has been moved to location l1
6. take k1 l1 cb ca p1	S6	Crane k1 took container cb
7. load k1 l1 cb r1	S7	Robot r1 was loaded with container cb
8. move r1 l1 l2	S8	Robot r1 has been moved to location 2
9. unload k3 l2 cb r1	S9	Crane k3 unloaded container cb
10. move r1 l2 l1	S10	The robot has been moved to location l1
11. take k1 l1 ca pallet p1	S11	Crane k1 took container ca
12. load k1 l1 ca r1	S12	Robot r1 was loaded with container ca
13. move r1 l1 l2	S13	The robot has been moved to location l2
14. put k3 l2 cb pallet p2	S14	Crane k3 placed container cb on pallet
15. unload k3 l2 ca r1	S15	Crane k3 unloaded container ca
16. put k3 l2 ca cb p2	S16	Crane k3 placed container ca on cb
17. put k2 l2 cc ca p2	S17	Crane k3 placed container cc on ca

The test consisted of choosing one of the states generated in plan I to represent an invalid state. We note that during state transition the *ca* container is unloaded by crane *k3* at location *l2*. Suppose there is a rule in the transport domain of these containers that prevents crane *k3* from loading the *ca* container for some reason. So we determined that the first invalid state of this domain is "Crane *k3* unloading the *ca* container".

Looking at Table 2 state S15 is the state chosen as invalid. This means that action *15 - unload k3 l2 ca r1* contains the effect responsible for generating the invalid state. So, to create the invalid state file in PDDL, you must use the unload action effect definition defined in the planning domain file. The effect is represented as follows: *effect (and (unloaded? R) (holding? K? C))*.

The effect definition will be used to map the invalid state. The effect parameters are changed to the names used in the problem. So, define the effect as: *effect (and (unloaded? r1) (holding? k3? ca))*, indicating that *k3* is holding the *ca* container as shown in Figure 11.

```
(:action invalid1
    :parameters (?r1 ?k3 ?ca)
    :precondition (and (robot ?r1) (crane ?k3)
                       (container ?ca))
    :effect (and (unloaded ?r1) (holding ?k3 ?ca))

)
```

Figure 11: Invalid state definition.

The problem was rerun in the scheduler using the invalid state validator method. Plan II was created with different actions to find the objective state, thus fulfilling the restriction added to the states. Table 3 shows the action it was generating, or the invalid state is no longer present in plan II. Action *16. unload k2l2 ca r1* detects that the container is now loaded by crane *k2*.

```
(define (domain invalidstates)
  (:requirements :strips)
  (:action invalid1
     :parameters (?r1 ?k3 ?ca )
     :precondition (and (robot ?r1) (crane ?k3)
        (container ?ca))
     :effect (and (unloaded ?r1) (holding ?k3 ?ca))
  )
  (:action invalid2
     :parameters (?r1 ?k2 ?cc )
     :precondition (and (robot ?r1) (crane ?k2)
        (container ?cc))
     :effect (and (unloaded ?r1) (holding ?k2 ?cc))
  )
  (:action invalid3
     :parameters (?r1 ?k2 ?cb )
     :precondition (and (robot ?r1) (crane ?k2)
        (container ?cb))
     :effect (and (unloaded ?r1) (holding ?k2 ?cb))
  )
  (:action invalid4
     :parameters (?ca ?cc )
     :precondition (and (container ?ca) (container ?cc))
     :effect (and (on ?ca ?cc))
  )
  (:action invalid5
     :parameters (?pallet ?cc ?p2)
     :precondition (and (container ?pallet)
     (container ?cc) (pile ?p2))
     :effect (and  (in ?cc ?p2) (top ?cc ?p2)
     (on ?cc ?pallet))
  )
)
```

Figure 12: Invalid state definition.

After proving that the planner generated another plan

Table 3: Representation of the plan states.

Action	State	Representation
Init	Si	Containers stacked on site 1
1. take k1 l1 cc cb p1	S1	Crane k1 grabbed the cc container
2. load k1 l1 cc r1	S2	Robot r1 was loaded with cc container
3. move r1 l1 l2	S3	The robot has been moved to location l2
4. unload k3 l2 cc r1	S4	The crane also unloaded the cc container
5. move r1 l2 l1	S5	Robot r1 has been moved to location l1
6. take k1 l1 cb ca p1	S6	Crane k1 took container cb
7. put k1 l1 cb pallet q1	S7	Crane k1 placed container cb on pallet
8. take k1 l1 ca pallet p1	S8	Crane k1 grabbed the ca container
9. load k1 l1 ca r1	S9	Robot r1 was loaded with container ca
10. move r1 l1 l2	S10	The robot has been moved to location l2
11. unload k2 l2 ca r1	S11	The crane also unloaded the ca container
12. move r1 l2 l1	S12	The robot has been moved to location l1
13. take k1 l1 cb pallet q1	S13	Crane k1 grabbed the cb container
14. load k1 l1 cb r1	S14	Robot r1 was loaded with cb container
15. move r1 l1 l2	S15	The robot has been moved to location l2
16. put k2 l2 ca pallet p2	S16	Crane k2 placed container ca on pallet
17. put k3 l2 cc ca p2	S17	Crane k3 placed container cc on ca
18. unload k3 l2 cb r1	S18	The crane also unloaded the cb container
19. put k3 l2 cb cc p2	S19	Crane k3 placed container cb on cc

with different actions and found the objective state. We have included other invalid states in the file to test further restrictions. Figure 12 shows the configuration of the five invalid state configurators for this problem.

In the invalid state *invalid1*, *invalid2* and *invalid3* we add restrictions on the cranes in location *l2*. Where *invalid1* restricts crane *k3* from unloading the *ca* container, *invalid2* restricts crane *k2* from unloading the *cc* container and *invalid3* restricts crane *k2* from unloading the container *cb*. The invalid state *invalid4* adds a different condition, which restricts the *ca* container to be over the *cc* container, and *invalid5* restricts the *cc* container to the top of the container stack.

The relation between the plan actions and the generated state transitions is shown on Table 2. The results obtained with the state validator function were successful, since the state mapped is not present in the resulting plan.

The planner considered the invalid state in planning time when generating the new plan. A planning problem can be solved in different manners, using different steps. E.g. the blocks world problem can be solved in an almost infinite stack permutation.

The planning problem was submitted to the planner 5 times. In all the tests, the state validator function worked, always generating plans that do not include any of the invalid states.

4.1 Contributions of this Work

Our state validation method proved it is possible to generate plan solutions even when domain constraints exist. Automated planning for space sciences can benefit from our finding.

Figure 13 shows a possible approach for generating valid plans for satellite operation.

This approach for valid plan generation is comprised of three levels. In the first (layer 1), the degraded states in the context of satellite operation are inserted by specialists or by a data mining process in a degraded state database. The second level (layer 2) will convert the degraded states found in the previous process to invalid states written in PDDL. The invalid states are then used in the third level (layer 3) as input to the planner software.

Thus, as new satellite states are being degraded by time, plans are automatically being generated and validated, containing no degraded states.

5 CONCLUSIONS AND FUTURE WORK

The execution of automatically generated plans on satellite flight operations can result in degraded or invalid states. Avoiding these states is possible through state validation, as described in this work and therefore, problems in generated plans can be avoided completely.

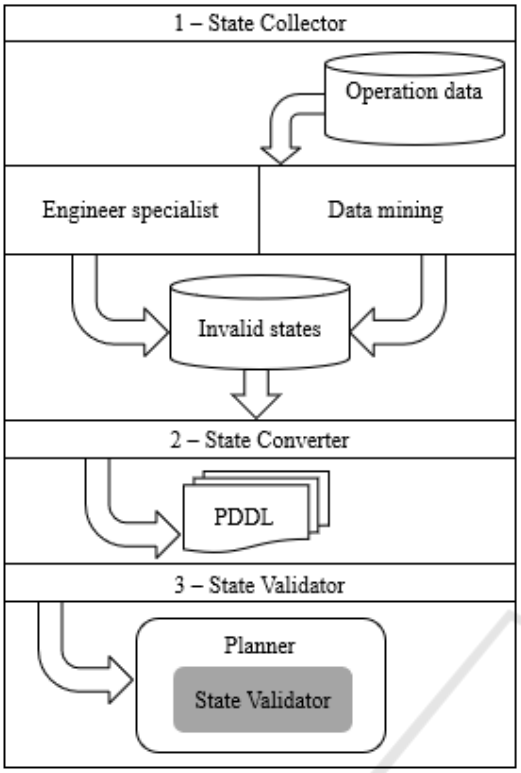


Figure 13: Approach to generating plans.

It is possible to generate plans that do not include certain states identified as invalid, as described in this paper, by implementing a state validator. A list of invalid states can be read by the planner, identifying actions that will be ignored when generating the plans.

We concluded that the implementation of a validation algorithm in the planner is needed at planning time to verify the actions included in the solution stack. While the planner generates the solution to find the end state, the validator verifies and compares the possible resulting states of the actions. If an action can generate an invalid state, it is discarded from the solution stack.

In future work, we intend to develop a technique able to convert invalid states to PDDL. A conversion algorithm will allow the generation of planning domains with preconfigured invalid states. The database and the data structure used to persist the domain invalid states are also challenging.

ACKNOWLEDGEMENTS

This work was partially supported by MURALIS TECNOLOGIA (www.muralis.com.br).

REFERENCES

Baier, J., Fritz, C., Bienvenu, M., and McIlraith, S. (2008). Beyond classical planning: Procedural control knowledge and preferences in state-of-the-art planners. In *AAAI*.

Becker, K. (2015). Ai planning with strips copyright (c) koly becker. In *Available in: https://github.com/primaryobjects/strips*.

Boutilier, C., Dean, T., and Hanks, S. (1999). Decision-theoretic planning: Structural assumptions and computational leverage. In *Journal of Artificial Intelligence Research, v. 11, p. 1-94*.

Briel, M. V. D., S., R., Minh, M. D., and Kambhampati, S. (2004). Effective approaches for partial satisfaction (over-subscription) planning. In *Proceedings of the 19th national conference on Artifical intelligence*.

Domshlak, C., Hullermeier, E., Kaci, S., and Prade, H. (2011). Preferences in ai: An overview.

Fikes, E. and Nilsson, J. (1971). Strips: A new approach to the application of theorem proving to problem solving. In *Stanford Research Institute, Menlo Park, California.*

Fox, M., Long, D., and Halsey, K. (2004). Complexity of concurrent temporal planning. In *Proc. of 17th Int. Conf. on Automated Planning and Scheduling*.

Gerevini, A. and Long, D. (2005). Plan constraints and preferences in pddl3. In *Technical Report RT-2005-08-47*. Dipartimento di Elettronica per l'Automazione, Universitá di Brescia.

Gerevini, A. and Long, D. (2006). Preferences and soft constraints in pddl3. In *ICAPS workshop on planning with preferences and soft constraints, p. 46-53*.

Ghallab, M., Nau, D., and Traverso, P. (2004). Automated planning-theory and practice. In *chapter 1. Elsevier/Mogan Kaufmann*. Elsevier.

Gupta, N. and Nau, D. (1991). Complexity results for blocks-world planning. In *AAAI-91*.

Jorge, A. and Sheila, A. M. (2008). Planning with preferences. In *AI Magazine*.

McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., and Wilkins, D. (1998). Pddl-the planning domain definition language-version 1.2. In *Yale Center for Computational Vision and Control*.

Nebel, B. (2000). On the compilability and the expressive power of propositional planning formalisms. In *Journal of Artificial Intelligence Research 12 (2000) 271–315*.

Rintanen, J. (2007). Complexity of concurrent temporal planning. In *Proc. of 17th Int. Conf. on Automated Planning and Scheduling*.

Sohrabi, S., Baier, J., and Mcilraith, S. (2009). Htn planning with preferences. In *Twenty-First International Joint Conference on Artificial Intelligence*.

Souza, P., Ferreira, M., and Silva, S. (2012). A mathematical model to predict operating states of satellites. In *ESPACEOPS*.

Thiébaux, S., Hoffmann, J., and Nebel, B. (2005). In defense of pddl axioms, artificial intelligence 168 (2005) 38–69.

Tominaga, J., Ferreira, M., and Silva, J. (2011). A rule-based satellite simulator for use in flight operations planning. In *Journal of Computational Interdisciplinary Sciences*.

Tu, P., Son, T., and Pontelli, E. (2007). Cpp: A constraint logic programming-based planner with preferences. In *International Conference on Logic Programming and Nonmonotonic Reasoning, p. 290-296*. Springer, Berlin, Heidelberg.

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Automated Planning with Invalid States Prediction

CRUZ C. G. RODRIGUES[1], SILVA R. ROCHA[23], FERREIRA M. G. VIEIRA[1], AND BERNARDINO JORGE[45], (Member, IEEE)

[1]National Institute for Space Research (INPE), São Jose dos Campos, Brazil (e-mail: caio.cruz, mauricio.ferreira @inpe.br)
[2]Centre for Informatics and Systems, University of Coimbra (CISUC) Coimbra, Portugal (e-mail: rrochas@dei.uc.pt)
[3]FATEC Mogi das Cruzes, São Paulo Technological College, Mogi das Cruzes, Brazil
[4]Polytechnic of Coimbra, ISEC, Rua Pedro Nunes – Quinta da Nora, 3030-199 Coimbra, Portugal (e-mail: jorge@isec.pt)
[5]Centre for Informatics and Systems of the University of Coimbra (CISUC), Polo II - Pinhal de Marrocos, 3030-290 Coimbra, Portugal

Corresponding author: Cruz C. G. Rodrigues (e-mail: caio.cruz2@fatec.sp.gov.br).

ABSTRACT The increase of automated systems in space missions raises concerns about safety and reliability in operations carried out with satellites, due to performance degradation. There are several works in the literature in automatic planning, but many proposed approach plans are generated with invalid states. The invalid state can be understood as a prohibited, degraded or risky scenario for the domain. This work proposes an automated planning with restrictions that enables automatic planners to not generate plans with invalid states. We implement a validator method for the planner software which proves that plan generation matches the restrictions imposed on the domain. In the experiments, we test an automatic planning specific to the aerospace area, where a knowledge base with invalid states is available in the context of the operation of a satellite. Our proposal to carry out the verification of invalid states in automatic planning, can contribute for plans to be generated with higher quality, ensuring that the goal of a plan is achieved through only valid intermediate states. It is also expected that the plans generated will be executed with better performance and will require less computational resources, since the search space is reduced.

INDEX TERMS Automated planning, Domain rule learning, Machine learning, PDDL

I. INTRODUCTION

Plan validation is necessary when automated planning is used to solve complex problems, such as the control of satellites in operations planning involving aerospace area [1]. The precaution in generating plans that are consistent with the current domain situation, has encouraged the development of applications to be responsible for diagnosing and validating automatically generated plans. In addition to safety and reliability, satellite operations requirements have been intensified due to the growing number of automated systems in this domain area [2].

In the planning validation stage, all states through which the domain must go through to reach the objective of the problem are verified in the plan. If there are one or more invalid states that compromise the domain and that cannot be executed, the plan gets considered as rejected [1]. The planning of flight operations for satellites is too complex to be solved solely by classic planning techniques because invalid state restrictions are not considered during the plan's state transition [3].

Several planning domains encounter problems of having a large set of solutions or having a set of goals that cannot be completely achieved. There still exists scenarios where plans of expected solutions can be generated to contain invalid states given the characteristics of the domain. In such cases, the way in which the goal is achieved, may be more important than actually reaching the goal itself. For this reason, it is important to generate valid plans, reaching all the goals of the problem, when possible [4].

The objective of this work is to propose an architecture for the automatic generation of plans with the validation of invalid states through a process of learning and configuring restrictions on the domain. In the approach, a filter is created for the states that are not included in the planned solution. In this case, valid plans are generated respecting the changes that occur in the domain's operating state. In order to validate the application of the architecture, a case study in the aerospace area was used in the experiments. The group of invalid states classified in the learning process, were configured with the respective invalid states and the results

of the planning were compared starting from each configured group.

The rest of this work is structured as follows: Section 2 presents the context and related works. Section 3 describes the proposed invalid state prediction architecture. Section 4 presents the experiment implemented in the aerospace area. Finally, section 5 presents the conclusion and future works.

II. BACKGROUND AND RELATED WORK

This section provides context on planning and related work. There exists several automated planning methods in literature. In [5], [6], [7], [8], [4] the authors include preferences in planning. Other authors like [9], [10], [11], [12] use learning techniques. However, they do not implement an invalid state prediction solution.

A. CLASSICAL PLANNING

Automated planning is a branch of Artificial Intelligence that focuses on problem solving which organizes actions through a system to achieve the goal of the problem [13]. A planning domain is defined as a finite set of possible states $S = \{s_0, s_1 ... s_k\}$, and a finite set of actions $A = \{a_1, a_2 ... a_k\}$, applicable to the domain states [13]. A planning problem can be represented as follows: $P = (\Sigma, s_i, S_g)$, where Σ is the state transition system, s_i is the initial state and S_g is a set of goal states. In the concept of classical planning, a plan is not deterministic and there may be different ways of finding the sequence of actions that transform the initial state s_i into the objective state S_g.

$$s_1 = \gamma(s_0, a_1),\ s_2 = \gamma(s_1, a_2), ...,\ s_k = (s_{k-1}, a_k) and\ s_k \in S_g$$

The state transition system represented above describes the function $\gamma(s, a)$ that applies an action to a state, resulting in a state transition system. For each generated state, an action is applied until the generated state corresponds to the goal. A problem can be solved in a number of ways. In other words, a set of different stages can transform the initial state into the goal state [13].

In classic planning, STRIPS (Stanford Research Institute Problem Solver) is a technique used to find solutions from a domain and a problem [14]. STRIPS aims to improve efficiency by reducing the size of the search space going through all possible states after applying the domain actions until it finds the goal state [14]. However, the use of classical techniques when used to solve more complex problems can create invalid plans, because in the generation phase it does not consider existing restrictions in the domain [15].

Automated planning is performed using a software called planner. The planner must find a sequence of actions that transform the initial state into the goal state using only the actions contained in the domain [16]. The result is a plan containing the sequence of actions required to find the goal of the problem. The most common language in automated planning is PDDL (Planning Domain Definition Language) introduced in 1998 by [16].

PDDL was established as the standard language commonly used in automated planning. PDDL paves the way for stronger collaboration, providing a platform for benchmarking approaches, exchanging different tools, as well as techniques and problems thanks to a series of international planning competitions that have been associated and extended in several stages to capture more expressive variants [17].

Recent studies demonstrate that temporal characteristics can be compiled in polynomial work and be subject to certain restrictions in the form of simultaneity that can appear in the problem [8]. Although broader research may require formalization by further language restrictions, making it necessary to use even more expression in existing languages or even new language extensions [17].

Over time planning approaches have evolved to solve more complex problems with new techniques. They embrace the inclusion of preferences, pre-processing using macros and application of learning using data mining in different planning contexts.

B. PLANNING WITH PREFERENCES

In literature, there are different works on AI planning that propose to solve more complex planning problems. Some of the approaching issues are the following: generation of quality plans; complex planning problems; planning with uncertainty; user preferences and preferences learning. Most of these works are related to the use of preferences, an interdisciplinary topic that has been extensively studied in recent years and has advanced different techniques and planning forms [18].

In order to improve the quantity of expressed criterion in complex plans, the PDDL language was extended to a PBP (Preference-based Planning). PDDL3 uses HTN (Hierarchical Task Network) to increase the expressive power over the plan's quality specification [19]. Two new features have been added that allows the user to express strong and soft constraints about the structure of the desired plans, as well as strong and soft problem goals. The first feature is the ability to express goals that apply not only to the final state of the trajectory of states visited by a plan, but also to intermediate states. These goals take the form of trajectory restrictions, similarly to work on temporal logics. The second extension is the ability to express soft constraints or mild preferences. Furthermore, the authors make it clear that the extensions are useful for expressing restrictions that manage the plan quality, instead of controlling the knowledge itself [7], [20].

The PBP aims to find the most preferred plans in a planning instance using some criteria to determine when a plan is more suitable than another. The search for the development of ideal plans is a theme addressed in [21], which examines various types of representations to solve planning problems. The theoretical decision planning uses the MDP (Markov Decision Process) to explore the assembly of optimal policies in order to indicate planning problems. The MDP associates a reward function with each state transition, to define

the users' preferences. All the possible states are classified quantitatively where the chosen optimal policy returns an action considering the history of actions performed by the agent [21]. In this approach, obstacles can be found when combining methods that explore different structures at the same time, making it a challenge to integrate or develop additional tools [21].

Another approach based on heuristic techniques was created, named PSP (Partial Satisfaction Planning), which offers resources to partially solve problems, and can meet a collection of objectives [6]. In this approach, the time needed to find the best solutions can take very long, as the search for better plans ends up generating a much larger number of research nodes. Another limiting factor is the use of the evalution function used to trim non-promising states, which can inevitably lose an optimal plan while removing a state from a research space [6].

There are works that compile the planning instance creating a method of controle procedure based on PDDL. [4] allowed the representation of the planning domain as a finite state automaton, adding an predicate for modifying effects and predefined conditions of actions, allowing the procedure to be fulfilled, in a process called PCK (Procedural Control Knowledge) [4]. Although, it is a general approach, the biggest advantage of this technique is the possibility to deal with a variety of domain control specification languages [4].

C. DOMAIN REMODELLING AND LEARNING IN PLANNING

In order to improving the efficiency of planners and making them solve a greater number of problems, modular modules were proposed. The use of reformulation and configuration techniques have contributed to the improvement of domains, such as the domain remodeling to identify irrelevant actions and operators that can be removed or replaced by other actions in the plan. [22] proposed a pre-processing step to remove actions that are generated by a sequence of other actions that do not contain it. This process reduces the number of explored states and improves the search time. Another technique applied to domain remodeling planning is the use of macro operators, which act as shortcuts to deeper states in the branches of the planning search space. For example, [23] adds macro operators in the domain, increasing it to solve as problem instances like a new normal operator.

Machine learning has been used in different contexts in order to improve the planning process. [9] proposed the Marvin planner, which evolved the FF (Fast Forward) planner by applying macro learning. During the plan searching process, [10] creates a predictor to choose the best heuristic. [11] proved that specific remodeling of the planning instance using machine learning may improve the performance of plan generation. There is also the implementation of a division process such as automatic actions, which decompose the most complex operators into simpler ones using the type domain-specific learning proposed in [12].

Although macros act as shortcuts in the search space, they increase the branching factor and potentially make some operators redundant [11]. Removing redundant or irrelevant operators, on the other hand, decreases the branching factor, but it is difficult to find operators that are safe to remove for all instances in a domain. Therefore, according to [11] the combination of macro addition and operator removal in a specific instance context is a valuable direction for investigation.

In approaches using macros, machine learning is applied to learn domain rules. In automated planning, domain models can change over time, with the emergence of new rules or certain situations that may not be available sometimes, resulting in a big problem that generates invalid plans.

An opportunity to also apply data mining is to produce rules or rule mappings for quality measures in the plan. In the work presented by [24], on the use of learning in planning, the planning problems found in space agencies were observed because they require the validation of plans before they are proposed. According to [24] as planning languages are unable to capture the full description of the domain he suggests that automated planning can incorporate a simulation made in the plan validation process to be used during plan generation. He also concludes that data mining can be used to improve the quality of planning in several ways. First, the validator can be seen as a function that can be an approximation in order to speed up the planner. In this case, data mining from the validator can produce an efficient and approximate function that can serve as a quick first cut once the automated planning is completed. An even better approach would be one that can be used much earlier in the planning process in a similar way to useless learning as proposed by [24].

D. ANALYSIS AND COMPARISON

In literature we can find many different planning approaches, which have all been proposed to solve different problems. The difference between one approach and another is the use of different techniques, such as, the inclusion of preferences, remodeling of domains, use of macros, use of machine learning, among other resources that have been incorporated in planning. TABLE 1 shows a comparison of the differential between the planning approaches presented and considers the main resources used by them. It is understood that each approach includes different resources for planning and for solving specific problems.

TABLE 1: Comparative table of planning approaches

Approach	Differential
HTN - Trajectory restrictions	Violated preferences metric
MDP - Reward function	Classification of actions taken
PSP - Heuristics	Planning with objectives subset
PCK - Automata	Additional action predicate
Macro Operators	Shortcuts to deeper states
Marvin Planner	Macro learning application

In this work, similarly to the previously mentioned planning approaches, we propose the inclusion of a new resource for plan validation in automated planning through the verification of invalid states, which is different from the rest of the other works found and which contributes to generate higher quality plans.

III. PROPOSED ARCHITECTURE

In order to create an application to allow a better representation of the domain, whilst considering its changes over time, an architecture was designed using a planner with state validation. In addition to the validation being done during planning time, it significantly reduces the total execution time and the application of a learning process that classifies the domain's current states contributing for plans with higher quality, since invalid states are excluded from the solution.

A. INVALID STATE DEFINITION

An invalid state is formed by current situations that undermine a domain scenario. In order to illustrate the understanding of the invalid states formulation, FIGURE 1 shows an example of the planning problem in the world of blocks. Where three different moments (Instant I, II, III) are presented using the same domain scenario, with three stacked blocks on a table. In this example it is noted that with the variation of time, the domain states planning has changed, as certain circumstances have arisen over time.

The first instant represents the scenario where there is no invalid situation. In the second instant, a given situation becomes invalid, assuming that *block b* is affected by an anomaly that prevents it from sustaining *block c*. In this case, we say that the situation *on (c b)* has become invalid. In the third instant, another invalid state is formed by two invalid situations, assuming that the element *block a* cannot be stacked on the table *table t1*, the situation *on (a t1)* also becomes invalid.

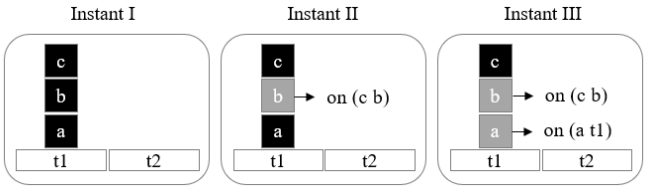


FIGURE 1: Illustration of invalid states

In the scenario presented in Instant III, the two invalid situations – one object stacked on top of another – are similar, but relate elements with different types. In the *on (c b)* situation, there is a relationship between two elements of the block type, yet in the situation *on (a t1)* one element is of the block type and the other is element is of the table type. However, to identify which domain action generates the invalid situation, it is necessary to observe which elements are related to the action's effect. For example, in this domain, the action to stack two blocks, has an effect with the definition *on (block1 block2)* indicating that the effect of applying the action *stack* generates the situation *on (c b)*. The invalid situation *on (a t1)* was generated by the action *mover* that has the definition *on (table block)* in its effect. To summarize, the effect of each action defined in the planning domain is composed by a set of situations. The parameters of each situation assume the values of the objects described in the definition of the planning problem, this way we can formulate the invalid states.

B. ARCHITECTURE OVERVIEW

In this section, we present in FIGURE 2 the architecture that consists of grouping the main components that involve the domain state classification and automated planning processes. The components were grouped into three main layers: the state classification layer, configuration, and planning. The proposed architecture aims to generate plans according to the current changes and situations of the current domain, allowing the prediction of invalid states. Starting with the configuration of preferences included in the configuration layer by a domain expert, it is possible to predict invalid states in the automated planning process. In the next sections, we explain each one of the three layers.

C. CLASSIFICATION LAYER

The classification layer encapsulates the learning process of planning domain. The input data at this layer is obtained through routine operation and the knowledge of experts in the field. The classification criteria is the information that defines all possible classes, in which the domain can be classified. The formulation of these criteria depends exclusively on the domain, as it is the set of characteristics that configure each classification. The state classifier component represents the learning model, which uses the criteria for training the classification algorithms, storing classified data in its history, which provides a prediction for the configuration layer.

D. CONFIGURATION LAYER

The configuration layer is the main component of the architecture and has the objective of integrating the two fundamental processes for generating valid plans: planning and classification. For a plan to be generated while respecting the invalid situations from the domain state at the time of planning, it is necessary that the appropriate changes to the domain are very well represented in the planning process. The prediction of invalid states in the generation of the plan is made from the configuration of a set of invalid situations, corresponding to each of the classes that the domain can assume over time. These rules are stored in the configuration database that has been named as *preferences*.

In order to ensure that the knowledge the planning domain representation is correctly mapped between the layers of the approach, we use KPlanOO - An object-oriented Meta-model to describe planning domains [25]. The ontology applied in the KPlanOO domain representation, allows the creation of configuration and processes integration in a flexible and

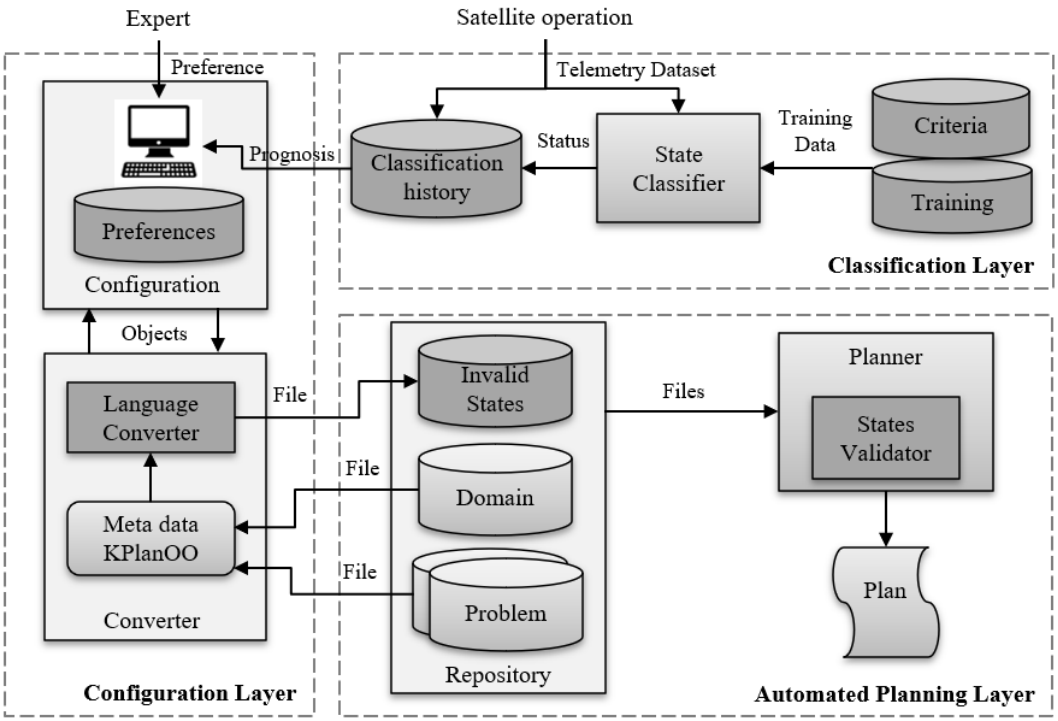


FIGURE 2: Architecture overview

standardized way. In addition to being a generic model, it enables the solution to be reused without depending on the planning language that was used in the process.

The following is a brief description of some object types defined in the KPlanOO representation structure, that are used in the architecture of this work, according to [25]:

- **Domain**: represents the planning domain description, composed of a list of actions, that have a list of probable states for problem objects with defined types.
- **Problem**: its purpose is to represent the problem description for a given domain. It is composed of an instance of the *Init class*, an instance of the *Goal class* and also a list of elements that will represent the objects to be used by the planner.
- **Action**: it is a domain's actions abstraction, which contemplates the specification of "Super Action", through a self-relationship. Each action has an effect and a precondition, which are objects of the scenario type.
- **Scenario**: entity that represents a scenario composed of the object's states defined in the problem in question. These being of types declared in the domain.
- **Situation**: entity that represents the object's situations that will be manipulated in the planning. These situations are the set formed by the objects state and the conditions linked to this state.
- **Element**: abstracts the objects that will be part of the problem. Element has a type and can have values (Value).
- **Type**: is the abstraction of the types that can be created in a domain. For example: satellite, subsystem, orbit, etc

...

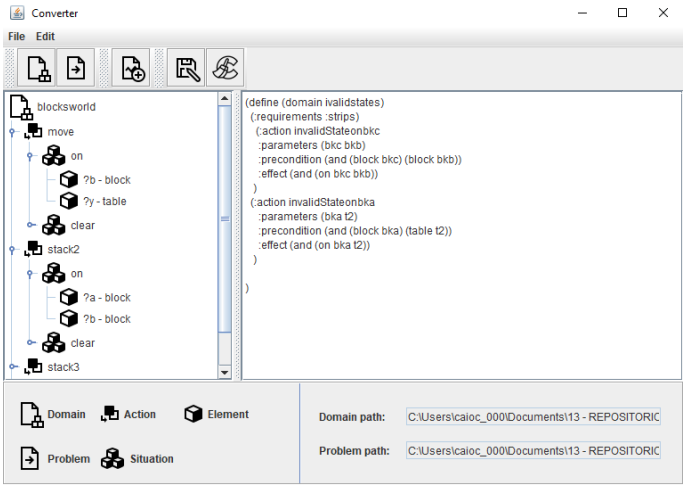


FIGURE 3: Graphical interface of the state converter

The language converter reads the domain and planning problem files, and using the domain representation KPlanOO structure, it loads the domain objects in a graphical interface, as shown in FIGURE 3. In the tree view that presents the planning items, the situations defined in each domain action is displayed and listed according to their type: action; situation or element.

The invalid states file is made from an expert choice of situations associated with the problem elements, and which are generated from the effects defined in the domain's actions. In order to assemble the file definition, an expert visualizes the

loaded objects and chooses within an action which situation will be invalidated. As described in the section III-A, after choosing a situation for each element, an object defined in the problem must be chosen. Finally, the chosen situation's set will be converted into the language used in the planner code.

FIGURE 4 shows, through the class diagram, the structure of the state converter component. The KPlanOO representation classes are used in the interface. The *IConverter* interface was created to define the following conversion methods:

- **ConvertModelToInvalidState:** is responsible for building the file of invalid states in a planning language. The received parameter depends on a list of objects of type *Action*, which are used to construct the invalid states and return to the definition in an object of type *String*.
- **ConvertDomainToModel:** is responsible for loading the planning domain file from an object of type *String*. The file data is read, instantiated, and returned in an object of type *Domain*.
- **ConvertProblemToModel:** is responsible for loading the planning problem file from an object of type *String*. A list of objects of type *Element* is returned.

The classes highlighted in the diagram represent the specific implementations necessary for the conversion of the planning language. The converter can be implemented using other languages, granted that new classes implement these interfaces and convert the domain to the respective language used. The class that implements the *IConverter* interface is related to a domain object constructor class. Based on the inheritance of the *DomainBuilderAbstract* class, which encapsulates the construction of the domain through a methods group responsible for facilitating the conversion of each object into language code, using the methods *addAction* and *addSituation* it is possible to create the files with the planning language.

Finally, the expert will associate an invalid state file to each class defined in the classification layer according to the domain's need. Thus, being able to configure groups of invalid situations, as in the example presented in the section III-A, each instant can be considered a domain class, whereas a different situation's group is invalid in each class.

E. AUTOMATED PLANNING LAYER

The automated planning layer represents the planning process with a predicion of invalid states. The planner component was altered with the inclusion of a state validator, which is responsible for permitting invalid states to be validated during the planning time. In addition to receiving the domain and problem files, the planner also receives the file of invalid states that contains the definition of states which cannot make up the plans [15].

[15] proposes the creation of an additional function within the planner. Which uses the same planning language that was used by the planner to describe invalid states and represent them in memory. The state is configured as a common action using the same notations as a domain file [15]. Listing 1 presents the example of the definition of invalid states illustrated in *Instance III* of FIGURE 1. In this example, in the invalidState's action, two variables are defined as parameters, and in the preconditions, the type is associated to each parameter, being two blocks. The positioning and order of the stacked blocks is defined in the effects, such as: *(and (on ?c ?b)*.

Listing 1: Setting invalid states

```
(define (domain invalid-state-settings)
  (:requirements :strips)
    (:action invalidState
     :parameters (?c ?b)
     :precondition (and (block ?c)
                        (block ?b))
     :effect (and (on ?c ?b))
    )
)
```

The approach uses the existing implementation in its own planner to read the new file and load it into memory. The function that applies the actions generating new child states, has been adapted to validate the states. [15] implement the States Validator function in the planner. Because during the planning, after the generation of each new state, the function is called to validate if the generated state corresponds to an invalid state.

Listing 2: State Validator Function

```
function
 StateValidator(newState, invalidStates)
  var equalState
  for each invalidStates.defs do
    actions <- invalidStates.defs.effect
    for each actions do
      equalState <- false
      for each newState.defs do
       if newState.defs.operation != not
        and newState.defs.action ==
        actions.action and
        Equal(newState.defs.params,
            actions.params) then
         equalState <- true
    return equalState;
end
```

Listing 2 shows the definition of the State Validator algorithm. The function receives the new generated state and the list of invalid states as parameters. The comparison of the states is based on three conditions: if the operation is not negative; if the action of both parties is the same; and if the parameters of the parts are the same. The Equal function

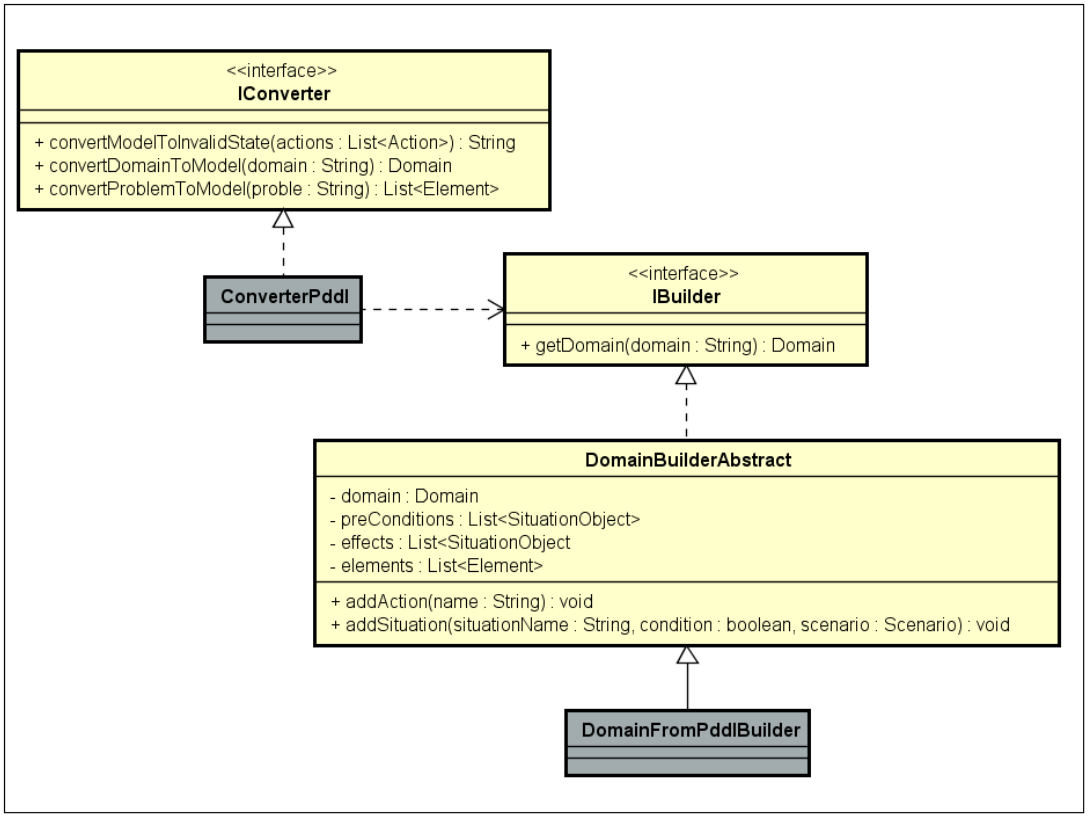


FIGURE 4: Static Structure of classes

compares the state's parameters, verifying if the size and values are identical.

The validation is done by comparing all the definitions in the *invalidStates* list. If any invalid state has all of its parts in the new state, the function returns true and the generated state is discarded from the planning. The planner continues to generate and validate other states until it finds the objective state of the problem [15].

This section described the architecture designed to improve the automated planning process. The application of the data classification process and configuration of invalid states are presented through a case study in the experiment's sections.

IV. EXPERIMENTS

This section describes the application of automated planning architecture with the prediction of invalid states in an aerospace area case study. The telemetry's data from a spacecraft, obtained through simulation was used to develop a classification model, configure invalid states and integrate a planning process. The experiment was carried out to achieve the following objectives:

- Create a case study to apply a learning process for invalid states;
- Present the result of the integration of the learning and planning processes through the automatically generated plans;
- Validate if the configured invalid states were considered when generating the plan, and corresponding to the classification presented by the current state of the domain.

A. MATERIALS AND METHODS

In this experiment we used the Atom SysVAP [26] finite automata validation system. The simulator allows the following: setting the values of environment variables and the machine's state, sending commands to change states, storing data generated by the simulation [26]. We simulated the operation of a satellite to obtain telemetry data and defined a case study to create a learning process for invalid states.

The case study considered the analysis and creation of invalid states related to the power supply subsystem of a satellite. According to the study of [27], 27% of failures in spacecrafts are due to the power supply subsystem and data mining techniques have been applied to detect these anomalies in satellite telemetry [28]–[30]. The simulation consisted of putting the satellite in a situation of extremely low battery, through the operation with several subsystems connected at the same time, causing greater battery consumption.

The chosen domain was the BR2 nano satellite that is present in the example models of the Atom SysVAP simulator [26]. This domain is composed of five subsystems, three of which are payload (SDATF, SLP, SMDH), the OBC - on-board computer and the transmitter. This domain has the following set of remote controls, which can be executed

in the simulator's plan: **get_tm** - generates the telemetry package of the subsystems; **repair** - repairs the alert state of the onboard control subsystem status; **com_t_on** - turns on the transmitter's transmission mode; **com_t_off** - turns off the transmitter's transmission mode; **sub_on** - connects a subsystem and **sub_off** - shuts down a subsystem [26].

The TABLE 2 lists the characteristics of the data set sent by the telemetry package. *Battery* telemetry is a numerical value that indicates the battery level of the satellite (this level varies between 13% and 19%). The *total drop* is an index of the total energy discharge from the battery, calculated using the sum of the energy consumption of each subsystem by the time of operation (the value varies between 0 and 1). *Orbiting* telemetry indicates the reading of the environment value: *Sun* - when the satellite battery is being charged by the sun, *Ecl* - when the satellite is in eclipse (without sunlight) and the battery is not being charged. *OBC* telemetry indicates whether the system is in a normal (On) or an alert state (Alert). Telemetry *communication* indicates the operating mode of the transmitter, *Re / TR* - indicating that the transmit and receive mode is active or *Receiver* - only the active receiver mode. The remaining telemetry data *(SDATF, SLP, SMDH)* corresponds to the payloads that indicate when they are **on** or **off**.

TABLE 2: Telemetry data

Name	Type	Values
Battery	numeric	13% - 19%
Total drop	numeric	0 - 1
Orbit	categorical	Sun, Ecl
OBC	categorical	Alert, On
Communication	categorical	Re/Tr, Receiver
SDATF	categorical	On, Off
SLP	categorical	On, Off
SMDH	categorical	On, Off
Time(s)	numeric	-

The data process analysis and the creation of the data classification model was carried out using the Orange Data Mining tool [31]. When performing the satellite operation simulation, the generated telemetry data was loaded and analyzed to form the classification criteria and to train the classification algorithm.

B. CASE STUDY

The telemetry data generated in the simulation with the battery experiment was exported to *.csv* format and loaded into the Orange tool. FIGURE 5 shows the scatter plot of the battery level variation during the simulation. It is observed that at the instant 25,000s the battery level begins to gradually decrease until it reaches 40,000s where even with the presence of the sun (represented by the color red) the battery level continues to discharge. The battery discharge scenario occurs until 50,000s, when the OBC subsystem goes on alert, shutting down the other satellite subsystems, to ensure the battery is not fully discharged. After this moment, the battery level starts to charge again.

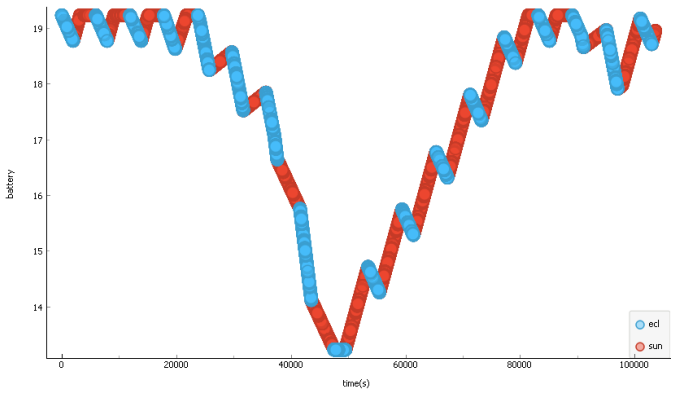


FIGURE 5: Scatter plot of battery level time series

In the next subsections, we describe the execution of the experiment in three stages. In the classification stage, we analyzed the data, developed the classification criteria and the choice of an algorithm to classify the data. During the configuration we chose the groups of invalid states that were configured. During the planning we showed the result of the plans generated using the experiment setup.

1) Classification

One of the ways to automatically identify abnormalities in a data set is through the use of outliers detection algorithms [32]. The Orange Outlier Detection widget [31] was used with the covariance estimator method to visualize the abnormalities present in the simulation data. FIGURE 6 shows the result of the outliers identified in the time series.

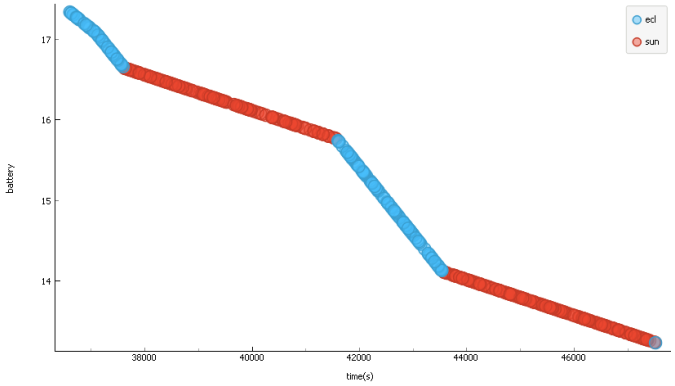


FIGURE 6: Outlier analysis scatter plot

Based on the analysis of the abnormal behavior identified in the outliers and the study of classification of battery states presented on [33], the criteria for classification of states presented in TABLE 3 was developed. Considering the battery level, the orbit and the total discharge, three classes were created: Forbidden, Alert, Safe. The *Forbidden* state represents the situation where the battery is no longer charged when illuminated by the sun. The *Alert* state represents the situation where the battery level is below 18% in the sun and 17.5% in eclipse. The *Safe* state represents the common

situation when the battery level is higher than the values of 18% in sun and 17.5% in eclipse.

TABLE 3: Classification criteria definition table

Battery	Total Drop	Orbit	Operating State
$> 17.5\%$	< 0.000533	ECL	SAFE
$> 18\%$	< 0.000533	SUN	SAFE
$< 17.5\%$	< 0.000533	ECL	ALERT
$< 18\%$	< 0.000533	SUN	ALERT
$< 16.65\%$	> 0.000533	ECL/SUN	FORBIDDEN

The Orange tool provides the Feature Constructor widget used for creating new features in a data set. The classification parameters defined in TABLE 3 were inserted in the widget to create a new feature with the value of the class established from the data characteristic. FIGURE 7 shows the result of the classification through a scatter chart where the classes are identified by colors.

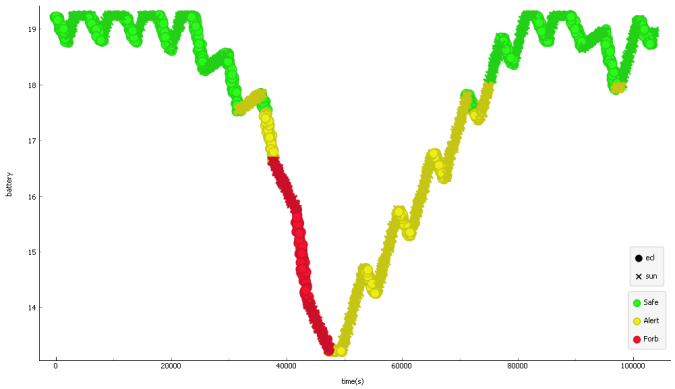


FIGURE 7: Scatter plot data classified

With the classified data, a data prediction model was created using the Predictions widget connected to the Orange Naive Bayes classifier model [31]. We use Naive Bayesian algorithm for fault detection in satellite batteries because it presents good results as demonstrated in [29], the classifier Naive Bayesian presents a good result as a classifier for fault detection in satellite batteries. The model uses the classified data to make the predictions in the newly received telemetry observations.

2) Configuration

The files with the domain definitions and planning problem to be used in the validation of the automated planning layer, were created based on the commands present in the simulation, as described in the section IV-A. The objective of the planning problem is to send the telemetry data obtained in the subsystems. However, the *OBC* subsystem must obtain the telemetry data from the payload subsystems and generate a packet to be sent by the transmitter.

In the domain actions, the following preconditions were defined to achieve the objective of the problem: i) the transmitter must be configured in Re/Tr mode to send the telemetry data; ii) the subsystem must be turned on for the *OBC* to obtain the telemetry data; iii) and the package must be generated with a pair of telemetry data corresponding to a maximum of two payload subsystems. The initial state of the planning problem was defined with all payload subsystems turned **off** and the transmitter configured in Receiver mode. The main objective of the problem is to send telemetry data from the satellite.

In order to configure the invalid states in the automated planning process, for each operating state defined in TABLE 3, a group of invalid states has been assigned as shown in TABLE 4. In the first group it was defined that in the SAFE operating state, the scenario where the SLP subsystem is on represents an invalid situation. In the second group in the ALERT operating state, the scenario where the SLP and SMDH subsystems are on simultaneously is invalid. Lastly, in the third group if both SLP, SMDH or SDATF subsystems are on individually or simultaneously it is invalid.

TABLE 4: Invalid state configuration table

Group	Operating State	Invalid Situations
G1	SAFE	ON - SLP
G2	ALERT	ON - SLP and SDATF
G3	FORBIDDEN	ON - (SLP and SDATF and SMDH) and (SLP or SDATF or SMDH)

Listing 3: Setting group G1 invalid states

```
(define (domain invalidstatesG1)
  (:requirements :strips)
    (:action invalidState
     :parameters (?slp)
     :precondition (and (subsystem ?slp))
     :effect (and (poweron ?slp))
    )
)
```

The invalid state file was generated using the corresponding configuration layer interface, where the files generated in each of the groups configured in this experiment are listed below. Listing 3 shows the invalid state's file configured for the G1 group. The invalid situation is configured from the definition of the action effect *subsystemon*, where the expression *(poweron ?slp)* represents the situation where the SLP subsystem is on.

Listing 4: Setting group G2 invalid states

```
(define (domain invalidstatesG2)
  (:requirements :strips)
    (:action invalidState
     :parameters (?slp ?sdatf)
     :precondition (and (subsystem ?slp)
                        (subsystem ?sdatf))
     :effect (and (poweron ?slp)
                  (poweron ?sdatf))
    )
)
```

Listing 4 shows the definition of the G2 group invalid state's file. In order to represent the simultaneous turn on of two subsystems, the effect was configured as shown in the expression *(poweron ?slp) (poweron ?sdatf)* combining the SLP and SMDH subsystems turned on simultaneously.

The configuration of the third invalid states group is shown in Listing 5. Each subsystem is defined in a different action, representing that the subsystems cannot be turned on individually or simultaneously.

Listing 5: Setting group G3 invalid states

```
(define (domain invalidstatesG3)
  (:requirements :strips)
    (:action invalidSlp
     :parameters (?slp)
     :precondition (and (subsystem ?slp))
     :effect (and (poweron ?slp))
    )
    (:action invalidSdatf
     :parameters (?sdatf)
     :precondition (and
            (subsystem ?sdatf))
     :effect (and (poweron ?sdatf))
    )
    (:action invalidSmdh
     :parameters (?smdh)
     :precondition (and (subsystem ?smdh))
     :effect (and (poweron ?smdh))
    )
)
```

3) Planning

The classification model defined and the planning domain's invalid states configured according to the architecture definition in section III-B. The invalid state planner proposed by [15] was used to execute the planning problem, operating the same domain and problem with the different groups of invalid states. Listing 6 shows the plan generated using the G1 group invalid state's file.

Listing 6: Plan executed with G1 group

```
1. changetransmitter transmissor rectr
2. subsystemon sdatf
3. gettm sat obc sdatf
4. subsystemon smdh
5. gettm sat obc smdh
6. subsystemoff sdatf
7. subsystemoff smdh
8. generatepkg obc sdatf smdh
9. sendtelemetry transmissor rectr obc

- Solution found in 9 steps!
Depth: 9, 134883 child states.
Runtime: 476579.710ms
```

As shown in Listing 6 in step 1 - the transmitter was changed to Re/Tr mode; in step 2 - the SDATF subsystem was turned on; In step 3 - the OBC obtained the telemetry from the SDATF; In step 4 - the SMDH subsystem was turned on; In step 5 - the OBC obtained SMDH telemetry; In steps 6 and 7 - the subsystems were turned off; In step 8 - the telemetry package was generated; and in step 9 - the transmitter sends the telemetry package generated by the OBC. Note that the SLP subsystem defined in group G1, was not linked by the action *subsystemon* during the steps of the generated plan.

Listing 7 shows the plan generated using the G2 group invalid state's file. The restriction imposed on the G2 group defined that the SLP and SDATF subsystems were not turned on simultaneously, so in the plan, after obtaining the telemetry of the SDATF subsystem in step 3, the subsystem was turned off in step 4 before the SLP subsystem was turned on, thus respecting the restriction imposed on the invalid states. The restrictions were followed by the planner in both groups, where we can see that the subsystems were linked respecting the configured rules.

Listing 7: Plan executed with G2 group

```
1. changetransmitter transmissor rectr
2. subsystemon sdatf
3. gettm sat obc sdatf
4. subsystemoff sdatf
5. subsystemon slp
6. gettm sat obc slp
7. subsystemoff slp
8. generatepkg obc sdatf slp
9. sendtelemetry transmissor rectr obc

- Solution found in 9 steps!
Depth: 9, 2388 child states.
Runtime: 18887.102ms
```

In terms of the performance of plan generation, even though the depth of the search tree is the same, it is observed that with the increase of invalid situations in the G2 group, the number of child states generated in the search space is 50 times less, as well as the execution time which is 25 times shorter when compared to the G1 group plan. Due to the fact that the planner ignores invalid states and generated fewer branches.

In the planning of the G3 group a plan was not generated, since the planner did not find a solution to send the telemetry from the satellite because at least two payload subsystems must be connected, as described in the section IV-B2. It is concluded that the three groups of invalid states were met in the executed plans and therefore the objectives of the experiment that were defined at the beginning of this section were achieved.

Finally, the case study allows the learning process to be applied based on the classification of operating states. The processes integration result was proven through the plans generated in an automated way and through the different

steps verification generated in each of the plans, respecting invalid states configured in each group. In the example, the satellite situation battery was used to define when the subsystems can be turned on, thus preventing the satellite from going on alert and its battery being completely discharged.

V. CONCLUSION

Plan validation is not a trivial task in the automated planning of complex domains, since the generated plans can pose risks to the domain security without a validation step. Based on the creation of a new definition that models the invalid states during planning and allows the plans to be generated following restrictions according to the current state of the domain, the approach presented in this work guarantees the planning of valid plans. With the state's validation being done during planning time, the need to execute a plan validation stage after the planning process was eliminated, contributing to the reduction of the total planning time and avoiding replanning.

To the best of our knowledge, there is no other approach that uses a planner modified to validate invalid states, including planning restrictions obtained from domain learning, it was possible through the use of an object-oriented planning meta-model, to join the planning and learning processes in a standard way. Our proposal enables the solution to be reused and allows the configuration to be easily implemented with other planning languages.

In summary, the architecture implementation allows to generate the plans in an independent mode. Because the changes in the domain's behavior could be considered in the automated planning, without the domain definitions modification and planning problem.

A. CONTRIBUTIONS

The main contribution of this work is the proposal to validate plans during the planning time, presented as a new way to validate invalid states within the planner itself. As with other planning approaches, data mining is used to improve the process as well as learning new rules. In this work, we also contribute with the use of learning to configure invalid states groups and integrate a viable solution to the automated planning process.

The architecture presented also helps to update the planning of changes that can occur in the domain, because identifying the changes that occur in some domains can take a long time. However, problems such as invalid plans execution can be eliminated with a process that identifies such changes immediately.

B. FUTURE WORK

As the restrictions increase, the number of plans not found may also increase. As shown in the last experiment result of this work, when the invalid state prevents the planner from finding the solution to the problem. When this type of situation occurs, palliative solutions or contingency plans must be created.

As future work, we intend to use techniques for meeting partial plan objectives combined with the validation of invalid states. Using the invalid states feature to improve performance and planning time, as well as learning, can contribute to improve performance because the planner eliminates search paths as it finds invalid states.

REFERENCES

[1] P. Souza, "A mathematical model to predict operating states of satellites," in *SpaceOps 2012*, 2012, p. 1272995.
[2] J. Tominaga, M. Ferreira, and J. Silva, "A rule-based satellite simulator for use in flight operations planning," *Journal of Computational Interdisciplinary Sciences*, vol. 2, no. 2, pp. 111–121, 2011.
[3] L. Cardoso, M. Ferreira, and V. Orlando, "An intelligent system for generation of automatic flight operation plans for the satellite control activities at inpe," in *SpaceOps 2006 Conference*, 2006, p. 5575.
[4] J. A. Baier, C. Fritz, M. Bienvenu, and S. A. McIlraith, "Beyond classical planning: Procedural control knowledge and preferences in state-of-the-art planners." in *AAAI*, 2008, pp. 1509–1512.
[5] B. Nebel, "On the compilability and expressive power of propositional planning formalisms," *Journal of Artificial Intelligence Research*, vol. 12, pp. 271–315, 2000.
[6] M. V. D. Briel, R. Sanchez, M. B. Do, and S. Kambhampati, "Effective approaches for partial satisfaction (over-subscription) planning," in *Proceedings of the 19th national conference on Artifical intelligence*, 2004, pp. 562–569.
[7] A. Gerevini and D. Long, "Preferences and soft constraints in pddl3," in *ICAPS workshop on planning with preferences and soft constraints*, 2006, pp. 46–53.
[8] J. Rintanen *et al.*, "Complexity of concurrent temporal planning." in *ICAPS*, vol. 7, 2007, pp. 280–287.
[9] A. Coles and A. Smith, "Marvin: Macro-actions from reduced versions of the instance," 2004.
[10] C. Domshlak, E. Karpas, and S. Markovitch, "To max or not to max: Online learning for speeding up optimal planning," in *Twenty-Fourth AAAI Conference on Artificial Intelligence*. Citeseer, 2010.
[11] M. Alhossaini and J. C. Beck, "Instance-specific remodelling of planning domains by adding macros and removing operators," in *Tenth Symposium of Abstraction, Reformulation, and Approximation*, 2013.
[12] C. Areces, F. Bustos, M. A. Domínguez, and J. Hoffmann, "Optimizing planning domains by automatic action schema splitting." in *ICAPS*. Citeseer, 2014.
[13] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: theory and practice*. Elsevier, 2004.
[14] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artificial intelligence*, vol. 2, no. 3-4, pp. 189–208, 1971.
[15] C. G. R. Cruz, M. G. V. Ferreira, and R. R. Silva, "State validation in automated planning." in *ICEIS (1)*, 2020, pp. 396–406.
[16] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "Pddl–the planning domain definition language–version 1.2," *Yale Center for Computational Vision and Control, Tech. Rep. CVC TR-98-003/DCS TR-1165*, 1998.
[17] M. Vallati, L. Chrpa, M. Grześ, T. L. McCluskey, M. Roberts, S. Sanner *et al.*, "The 2014 international planning competition: Progress and trends," *Ai Magazine*, vol. 36, no. 3, pp. 90–98, 2015.
[18] C. Domshlak, E. Hüllermeier, S. Kaci, and H. Prade, "Preferences in ai: An overview," 2011.
[19] A. Jorge, S. A. McIlraith *et al.*, "Planning with preferences," *AI Magazine*, vol. 29, no. 4, pp. 25–25, 2008.
[20] A. Gerevini and D. Long, "Plan constraints and preferences in pddl3," Technical Report 2005-08-07, Department of Electronics for Automation, University of Brescia, Brescia, Italy, Tech. Rep., 2005.
[21] C. Boutilier, T. Dean, and S. Hanks, "Decision-theoretic planning: Structural assumptions and computational leverage," *Journal of Artificial Intelligence Research*, vol. 11, pp. 1–94, 1999.
[22] P. Haslum and P. Jonsson, "Planning with reduced operator sets." in *AIPS*, 2000, pp. 150–158.
[23] A. Botea, M. Enzenberger, M. Müller, and J. Schaeffer, "Macro-ff: Improving ai planning with automatically learned macro-operators," *Journal of Artificial Intelligence Research*, vol. 24, pp. 581–621, 2005.

[24] J. Frank, "Using data mining to enhance automated planning and scheduling," in *2007 IEEE Symposium on Computational Intelligence and Data Mining*. IEEE, 2007, pp. 251–260.
[25] R. Silva, M. G. Ferreira, and N. Vijaykumar, "An object-oriented metamodel as ontology for describing domains and problems for planning space applications planning," in *SpaceOps 2010 Conference Delivering on the Dream Hosted by NASA Marshall Space Flight Center and Organized by AIAA*, 2010, p. 2172.
[26] A. A. S. Ivo, "A tool for evaluating satellite flight plans using state models," Master Thesis (MSc in Engineering and Management of Space Systems), National Institute of Space Research (INPE), 2013. [Online]. Available: http://urlib.net/rep/8JMKD3MGP7W/3FB2RJE?ibiurl.language=pt-BR
[27] M. Tafazoli, "A study of on-orbit spacecraft failures," *Acta Astronautica*, vol. 64, no. 2-3, pp. 195–205, 2009.
[28] D. R. Azevedo, A. M. Ambrósio, and M. Vieira, "Applying data mining for detecting anomalies in satellites," in *2012 Ninth European Dependable Computing Conference*. IEEE, 2012, pp. 212–217.
[29] M. A. Galal, W. M. Hussein, E. El-din abdel Kawy, and M. M. Sayed, "Satellite battery fault detection using naïve bayesian classifier," in *2019 IEEE Aerospace Conference*. IEEE, 2019, pp. 1–11.
[30] S. Abdelghafar, A. Darwish, A. E. Hassanien, M. Yahia, and A. Zaghrout, "Anomaly detection of satellite telemetry based on optimized extreme learning machine," *Journal of Space Safety Engineering*, vol. 6, no. 4, pp. 291–298, 2019.
[31] J. Demšar, T. Curk, A. Erjavec, Č. Gorup, T. Hočevar, M. Milutinovič, M. Možina, M. Polajnar, M. Toplak, A. Starič *et al.*, "Orange: data mining toolbox in python," *the Journal of machine Learning research*, vol. 14, no. 1, pp. 2349–2353, 2013.
[32] O. Mining, "Orange visual programming documentation," *Oct-2020*, 2020.
[33] P. B. de Souza, M. G. V. Ferreira, and J. D. S. da Silva, "A tool for prediction of satellite future states," *Journal of Aerospace Computing, Information, and Communication*, vol. 7, no. 12, pp. 406–414, 2010.

CRUZ C. G. RODRIGUES Master in Space Systems Engineering and Management from the National Institute for Space Research - INPE, São José dos Campos, São Paulo, Brazil, 2021. Graduated in System Analysis and Development from the FATEC Mogi das Cruzes São Paulo Technological College, Mogi das Cruzes, São Paulo, Brazil, 2016.

He works as a software developer at Muralis Technology. Has an article published in Proceedings of the 22nd International Conference on Enterprise Information Systems (ICEIS 2020).

SILVA R. ROCHA PhD in Computing from Technological Institute of Aeronautics - ITA, Master in Applied Computing from the National Institute for Space Research - INPE and graduated in Computer Science from the University of Mogi das Cruzes in Brazil.

He is a full professor at the Paula Souza Center in São Paulo and Member of the Center for Informatics and Systems of the Department of Informatics Engineering at the University of Coimbra. He is the author of more than 30 papers in international journals and conferences, he develops, and guides works in Big Data and data mining in academic and business contexts.

FERREIRA M. G. VIEIRA PhD in Applied Computing from the National Institute for Space Research (2001). Master in Applied Computing from INPE (1996). Full Professor in the Postgraduate Course in Space Engineering and Technology (ETE): concentration area Engineering and Management of Space Systems.

He is a researcher and coordinator of the Satellite Control Center at INPE. Member of the International Committee for Standardization of Software in the Space Area (CCSDS). Member of the organizing committee of the international space congress - SPACEOPS. He produced more than 170 articles, supervised 16 doctors, 24 masters and 2 pos doctorates. Currently he supervises 4 doctoral students and 7 master's students. He works in the area of research and development of software for satellite control. Scientific advisor to FAPESP in the area of Software Engineering. He was already a Productivity Scholarship Developer. Tec. and Innovative Extension level 2.

BERNARDINO J. (Member, IEEE) received the Ph.D. degree from the University of Coimbra, in 2002. From 2005 to 2010, he was the President of ISEC. From 2017 to 2019, he was also the President of ISEC Scientific Council. In 2014, he was a Visiting Professor at CMU. He is currently a Coordinator Professor with the Polytechnic of Coimbra—ISEC, Portugal.

He has authored more than 200 publications in refereed conferences and journals. His main research interests include big data, NoSQL, data warehousing, dependability, and software engineering. He participated in several national and international projects. He is a member of ACM. He is currently the Director of the Applied Research Institute (i2A), Polytechnic of Coimbra.

PUBLICAÇÕES TÉCNICO-CIENTÍFICAS EDITADAS PELO INPE

Teses e Dissertações (TDI)

Teses e Dissertações apresentadas nos Cursos de Pós-Graduação do INPE.

Manuais Técnicos (MAN)

São publicações de caráter técnico que incluem normas, procedimentos, instruções e orientações.

Notas Técnico-Científicas (NTC)

Incluem resultados preliminares de pesquisa, descrição de equipamentos, descrição e ou documentação de programas de computador, descrição de sistemas e experimentos, apresentação de testes, dados, atlas, e documentação de projetos de engenharia.

Relatórios de Pesquisa (RPQ)

Reportam resultados ou progressos de pesquisas tanto de natureza técnica quanto científica, cujo nível seja compatível com o de uma publicação em periódico nacional ou internacional.

Propostas e Relatórios de Projetos (PRP)

São propostas de projetos técnico-científicos e relatórios de acompanhamento de projetos, atividades e convênios.

Publicações Didáticas (PUD)

Incluem apostilas, notas de aula e manuais didáticos.

Publicações Seriadas

São os seriados técnico-científicos: boletins, periódicos, anuários e anais de eventos (simpósios e congressos). Constam destas publicações o Internacional Standard Serial Number (ISSN), que é um código único e definitivo para identificação de títulos de seriados.

Programas de Computador (PDC)

São a seqüência de instruções ou códigos, expressos em uma linguagem de programação compilada ou interpretada, a ser executada por um computador para alcançar um determinado objetivo. Aceitam-se tanto programas fonte quanto os executáveis.

Pré-publicações (PRE)

Todos os artigos publicados em periódicos, anais e como capítulos de livros.