FELIPE ALMEIDA GONÇALVES
SU YUN CHI

Sistematização do Processo de Verificação de FMS a partir de Modelos Formais em UPPAAL

São Paulo
2009

FELIPE ALMEIDA GONÇALVES
SU YUN CHI

Sistematização do Processo de Verificação de FMS a partir de Modelos Formais em UPPAAL

Monografia apresentada a Escola Politécnica da Universidade de São Paulo referente a disciplina PMR2550 – Projeto de Conclusão do Curso II

Curso de Graduação : Engenharia Mecatrônica

Orientador: Prof. Dr. Diolino José dos Santos Filho

São Paulo
2009

AGRADECIMENTOS

Agradecemos a Deus e aos nossos familiares, por providenciarem nosso sustento, nos apoiarem de maneira incondicional e possibilitarem nossa formação acadêmica e pessoal.

Somos extremamente gratos ao professor Diolino José dos Santos Filho que, junto com sua equipe, nos aconselhou e nos direcionou de modo que seguíssemos o caminho certo e fizéssemos as escolhas mais adequadas no desenvolver do nosso trabalho.

Por fim, agradecemos ao professor Matteo Giovanni Rossi que, apesar da distância, nunca deixou de nos orientar, ao Cleber Alves Sarmento que compartilhou seus conhecimentos conosco e ao Professor Paulo Eigi Miyagi que nos orientou na parte inicial do projeto.

RESUMO

O controle de sistemas flexíveis de manufatura (FMSs) precisa ser validado e verificado antes de ser executado em chão de fábrica.

Considerando que o FMS envolve processos e estrutura que podem ser controlados por uma sistema de controle modular, o presente projeto visa a criação de uma biblioteca de modelos formais de componentes do FMS, além de uma proposta de construção do modelo de controle supervisório, que auxiliem num procedimento sistematizado de modelagem do sistema global do FMS, com o uso da ferramenta UPPAAL que tem como principal recurso o seu eficiente algoritmo de verificação. A linguagem utilizada pela ferramenta baseia-se em autômatos temporizados estendidos e sua lógica de verificação é uma versão simplificada da CTL (*computational tree logic*).

Visto que o universo dos componentes de um FMS é muito amplo, o presente trabalho limitou-se a considerar três classes fundamentais: dispositivos de transformação, manipulação e transporte. Com estas, é possível descrever o funcionamento de uma grande variedade de FMSs.

A modelagem proposta para o sistema de controle é constituída por dois módulos: o módulo de controle dos processos, onde cada autômato descreve a seqüência de atividades para a fabricação de um determinado produto, e o módulo de supervisão que controla o processo global.

O processo de verificação é realizado a partir da análise da interação entre estes módulos de controle e os modelos funcionais dos componentes presentes no FMS.

Palavras-chave: FMS, verificação, UPPAAL, modelos formais, controle supervisório

ABSTRACT

The flexible manufacturing systems (FMSs) control needs to be validated and verified before its execution on shop floor.

Considering the fact that the FMS involves processes and structure that can be controlled by a modular control system, the present project aims at the creation of a library of formal models of components of the FMS, and a supervisory control model construction proposal, that assist a systemized modeling procedure of the global system of the FMS, with the use of UPPAAL that has, as main resource, its efficient algorithm of verification. The language used for the tool is based on extended timed automatons and its verification logic is a simplified version of the CTL (computational tree logic).

Since the universe of the components of a FMS is very large, this project was limited to consider three fundamental classes: devices of transformation, manipulation and transport. With these, it is possible to describe the functioning of a great variety of FMSs.

The control system model proposed is build by two modules: the process control module, where each automaton describes the activity sequence for the manufacture of one specific product, and the supervisor module that controls the global process.

The verification process is accomplished analyzing the interaction between these control modules and the functional models of components present in the FMS.

Keywords: FMS, verification, UPPAAL, formal models, supervisory control

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

AF - autômato finito

AFD - autômato finito determinístico

AFN – autômato finito não determinístico

AGV – veículo automaticamente guiado

AT – autômato temporizado

CNC – controle numérico computadorizado

CTL – *computational tree logic*

FMS – sistema flexível de manufatura

GT – *group technology*

RGV – veículo guiado por trilho

SED – sistema a eventos discretos

UPPAAL – derivado de Universidade de Uppsala e Universidade de Aalborg

SUMÁRIO

1 Introdução

No atual momento de economia globalizada, os novos padrões e atributos de competitividade indicam de forma inequívoca que os sistemas de manufatura precisam ter condições de atender de modo eficaz às variações das exigências do cliente. Dentro desta realidade, sistemas flexíveis de manufatura (FMSs), que são sistemas capazes de processar uma variedade de peças diferentes (Groover, 2008), têm apresentado uma resposta adequada a estas necessidades. Entretanto, para a implementação de um FMS é necessário desenvolver-se um estudo detalhado do comportamento dinâmico que se deseja para o sistema e assim é fundamental utilizar-se técnicas efetivas de modelagem dos processos que se deseja executar no referido sistema.

Considerando que o FMS envolve processos e dispositivos que podem ser estruturados de forma modular, é importante propor-se sistemáticas de construção do modelo do comportamento do FMS, que respeite estas características e que permita que seja feita uma análise antes da real implementação do sistema na fábrica. A possibilidade de analisar um FMS através de simulação permite uma maior eficiência do sistema, diminuindo o tempo de ociosidade das máquinas nas etapas de reconfiguração. É importante que através da simulação seja possível validar o sistema, ou seja, garantir que ele realizará efetivamente a atividade que foi requerida, e, principalmente, verificar se determinados estados podem ser alcançados, evitando possíveis acidentes, desgaste de ferramentas, desperdício de matéria-prima, entre outros problemas. A capacidade de realizar este processo de forma rápida e confiável pode ser considerada um grande diferencial para uma empresa.

Um ponto crucial é que o modelo a ser simulado deve ser formal, isto é, baseado em conceitos matemáticos e, dentro deste contexto, a ferramenta UPPAAL (BENGTSSON; LARSSON, 1996) mostra-se adequada, já que possibilita uma modelagem através de redes de autômatos temporizados que se comunicam entre si, além de possuir o algoritmo de verificação bastante eficaz.

Até então, trabalhos envolvendo modelagem de FMS em eventos discretos utilizaram-se de linguagens como Redes de Petri, pois estas são mais apropriadas para representar o comportamento de sistemas como o referido. No entanto, estes tipos de linguagem não possibilitam a realização de verificação do modelo.

O objetivo do projeto é criar uma biblioteca de componentes para a modelagem de FMS. Estes componentes podem instanciar os modelos dos módulos que compõem o FMS de modo que para cada caso prático específico o projetista possa adicionar ou excluir objetos do modelo, possibilitando modificações de configuração, sem a necessidade de refazer todo o modelo. Em seguida, propõe-se um procedimento de modelagem do controle destes componentes de modo que seja possível executar-se uma série de processos produtivos. Espera-se, através do presente trabalho, obter um método para facilitar, tornar mais confiável e acelerar consideravelmente as atividades de reconfiguração e verificação em sistemas flexíveis de manufatura.

1.1 Organização do texto

No capítulo 1 foi feita uma introdução do trabalho, apresentando as motivações, as justificativas e o objetivo do trabalho. No capítulo 2 é feita a apresentação dos sistemas flexíveis de manufatura, objeto de aplicação do presente projeto. O capítulo 3 descreve os conceitos envolvidos nos autômatos temporizados com os quais a ferramenta UPPAAL trabalha, além de tratar, de maneira introdutória, a teoria de controle supervisório. O capítulo 4 aborda o assunto verificação de modelos, dando enfoque em como este método é aplicado em UPPAAL. No capítulo 5 é explicado detalhadamente o método proposto para realizar verificação de FMS e no capítulo 6 este método é aplicado a um caso real. As conclusões são feitas no capítulo 7, seguidas das referências bibliográficas.

2 Sistema flexível de manufatura

Este capítulo apresentará o sistema flexível de manufatura descrevendo o contexto histórico da sua criação e evolução. Em seguida, faz-se um estudo sobre o assunto de classificação destes sistemas. O capítulo também aborda os diversos componentes de FMSs para modelagem destes que ocorrerá nos capítulos posteriores.

2.1 História

Com a crescente demanda do Mercado, os produtores precisaram de um sistema produtivo capaz de fabricar produtos variados adaptados às preferências do cliente e que pudesse reagir rapidamente as mudanças do mercado por meio de reconfigurações breves do seu sistema (ZHOU; VENKATESH, 1999). O avanço tecnológico, que possibilitou aplicação de máquinas numericamente controladas na indústria, com os investimentos massivos das empresas nas últimas décadas resultou na evolução dos FMSs possibilitando a sua aplicação na indústria. Segundo Luggen (1991), o conceito e o nome de sistema flexível de manufatura foram criados pelo engenheiro de pesquisa e desenvolvimento, David Williams, em Londres nos anos 60. Assim, a combinação do conceito de ferramentas de máquina de controle computacional descentralizado com a idéia de usar as máquinas em turnos completos, foi o início dos FMSs. Desde então, devido a estudos constantes sobre o assunto, os FMSs vêm apresentando um desempenho cada vez melhor e, conseqüentemente, estão mais presentes nas fábricas.

2.2 Definição

As definições de FMS envolvem subjetividades, portanto, estas, dependendo do sistema, do usuário e seu objetivo, podem apresentar inúmeras interpretações.

A) Definição 1 (KUSIAK, 1986)

O sistema flexível de manufatura é um sistema produtivo, com controle computadorizado, capaz de processar uma variedade de tipos de partes.

B) Definição 2 (ZHOU;VENKATESH, 1999 apud RANKY, 1983)

MS é um sistema que lida com alto nível de processamento de dados distribuídos e fluxo de material automatizado usando máquinas com controle computadorizado, conjunto de células, robôs industriais, máquinas de inspeção junto com sistemas de armazenamento e de transporte integrado com computadores.

C) Definição 3 (GREENWOOD, 1988)

O sistema flexível de manufatura, por meio de combinações cuidadosas de controle computacional, comunicações, processo de manufatura e equipamentos relacionados, possibilita a linha de produção de uma organização a responder, de uma maneira rápida, econômica e integrada, às mudanças significativas no seu ambiente operacional. Os constituintes típicos de tal sistema são: equipamentos de processamento, equipamentos de transporte material, sistema de comunicação e sistema de controle computacional sofisticado.

Além destas, existem outras definições mais atuais como, por exemplo:

D) Definição 4 (GROOVER, 2008)

Um FMS é formado por um conjunto de estações de trabalho agrupadas para processarem um grupo de peças similares, denominado família de peças baseado em Tecnologia de Grupo, altamente automatizado. Têm-se assim um conjunto de estações de trabalho interconectadas por um sistema de manipulação e armazenamento automático, e controlado por sistema de computadores distribuídos. Diz-se que tal sistema é flexível por sua capacidade de processar uma variedade de diferentes tipos de peças, simultaneamente, em diferentes estações de trabalho, sendo que o *mix* de produtos e quantidades produzidas podem ser alteradas.

Observando as definições verifica-se que nenhum sistema de manufatura é totalmente flexível, pois um FMS consegue fabricar apenas um número limitado de famílias de peças.

2.3 Classificação de FMS

Segundo o Groover (2008), existem diversas maneiras de classificar os FMSs, pois podem ser dados diferentes enfoques na sua classificação.

2.3.1 Classificação por tipos de operação

Esta consiste em duas categorias, a de operação de processamentos, e a de operação de montagem. Na maioria dos casos um FMS pertence a uma das categorias mas existem casos onde o sistema apresenta ambos os tipos de operação.

2.3.2 Classificação por número de máquinas

O critério de classificação baseia-se no número de máquinas presentes no sistema. A célula de máquina única consiste em um centro de manufatura CNC (controle numérico computadorizado) e seu sistema de armazenamento de onde, periodicamente, as peças feitas são descarregadas e as partes a serem trabalhadas são carregadas.

Segundo o Bonetto (1987), que titula este tipo de sistema como célula flexível simples (figura 2.1), a palavra chave para célula flexível é a autonomia, ou seja, a separação entre operador e máquina por meio de equipamentos numericamente isolados. No entanto, a presença de operadores para tarefas como supervisão, carregamento e descarregamento de peças são essenciais mesmo que estes não representem um trabalho real. Portanto existe um custo direto de mão-de-obra, mas este pode ser diminuído tendo várias máquinas sob a supervisão de um mesmo homem.

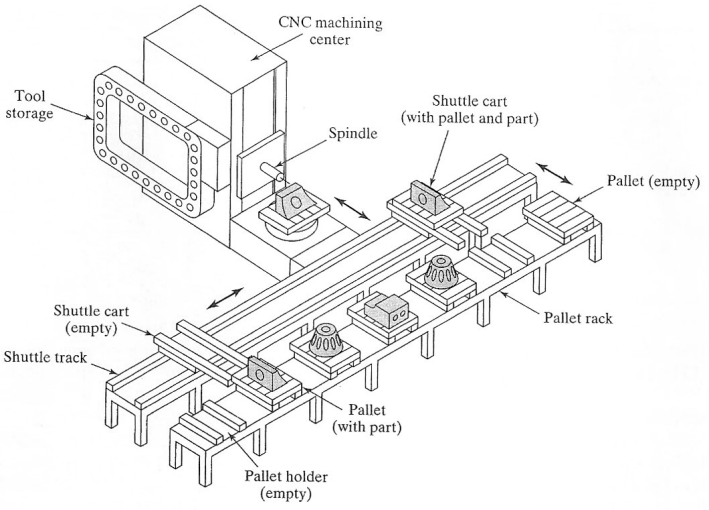


Figura 2. 1 - Célula de máquina única (GROOVER, 2008)

A célula flexível de manufatura consiste em dois a três centros de processamentos e um sistema de manuseio de peças que liga as estações de trabalho com as estações de carga e descarga. Ele pode ser construído de máquinas idênticas (figura 2.2) ou diferentes (figura 2.3).

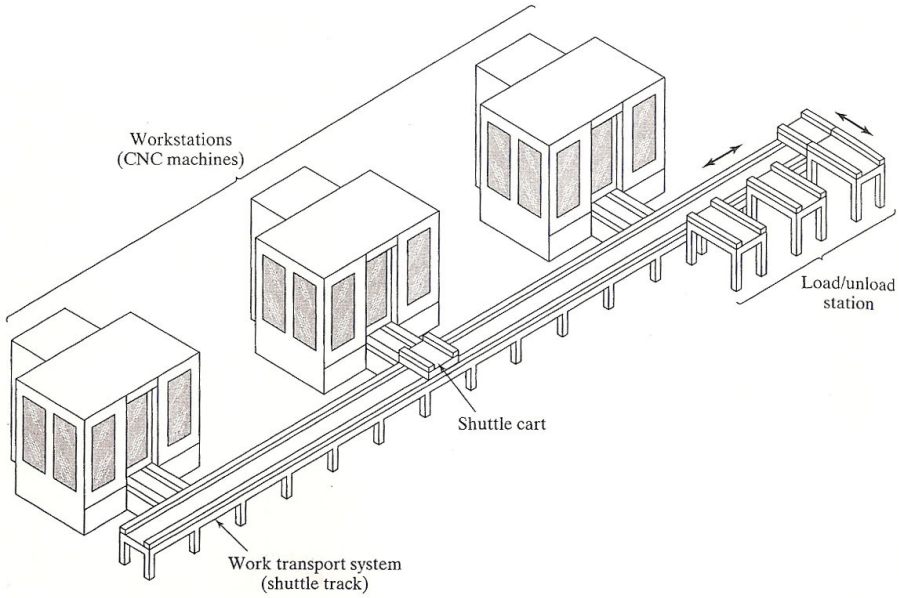


Figura 2. 2 - Célula com máquinas idênticas (GROOVER, 2008)

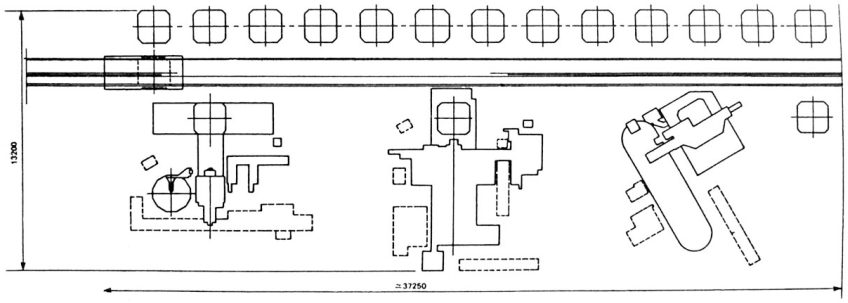


Figura 2. 3 - Célula com máquinas diferentes (BONETTO, 1987)

O sistema flexível de manufatura é constituído por mais de quatro estações de processamento conexas por um sistema de manuseio e controladas por um sistema de computadores distribuídos. O controle dos sistemas desta classe geralmente é mais sofisticada que outras.

As classes deste tipo apresentam diferentes magnitudes em ritmo de produção, volume de produção e investimentos sendo que estes valores tendem a crescer com o aumento de número máquinas, como mostra a figura 2.4.

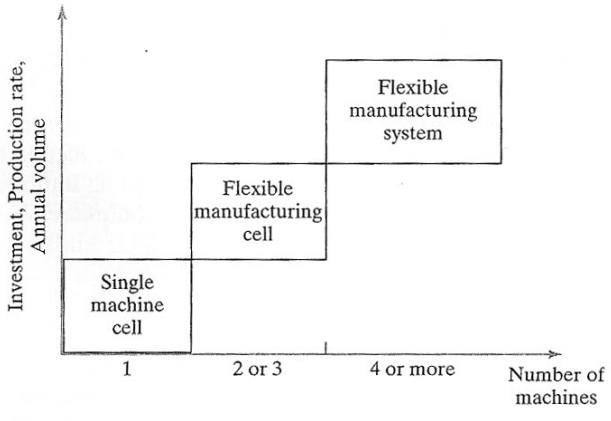


Figura 2. 4 - Características de três categorias (GROOVER, 2008)

2.3.3 Classificação por flexibilidade

Um sistema de manufatura, para adquirir flexibilidade, deve apresentar; habilidade de identificar e distinguir as diferentes peças ou estilo de produtos processados pelo sistema, alterações rápidas de instrução de operação e breve reconfiguração física.

Existem diversos critérios para analisar a flexibilidade de um sistema. Para que um sistema de manufatura seja qualificado como flexível, é essencial que consiga satisfazer seguintes condições: processar diferentes tipos de peças e aceitar mudanças em ordens de produção.

Pode ser categorizado por FMS dedicado aquele que serve para produção de uma variedade limitada de tipos de peças e todo o campo de peças a serem produzidas é pré-determinado. Por FMS de ordem randômico entende-se os sistemas que produzem uma grande família de peças, onde uma nova peça pode ser introduzida no sistema e, além disso, podem ocorrer modificações nas peças existentes e a ordem de solicitação de peças está sujeita a mudanças mais freqüentes. Este último é mais adequado na produção de grandes variedades de peças em menor quantia enquanto o outro é uma melhor solução para fabricar um *mix* de produtos de menor variedade em lotes maiores (GROOVER, 2008). A figura 2.5 mostra um gráfico comparativo que caracteriza esta situação.

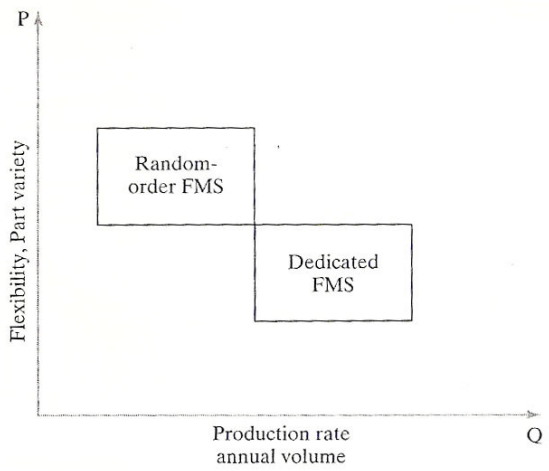


Figura 2. 5 - Características das duas categorias (GROOVER, 2008)

2.4 Componentes de um FMS

Segundo Groover (2008) os FMSs são constituídos por quatro classes de componentes:

a) Estação de trabalho

b) Sistema de manuseio e de armazenamento de materiais

c) Sistema de controle computacional

d) Recursos humanos

A estação de trabalho é a que está presente em maior número no grupo de componentes do sistema e existem diversos tipos de estações dependendo da sua funcionalidade. Em um sistema de usinagem (figura 2.6), as principais estações consistem em máquinas CNC que servem para processos como furação, fresamento, entre outros. Porém, existem outros tipos de estações que são aplicáveis nos FMSs, alguns exemplos destas são estação de carga e descarga, de montagem e de inspeções.

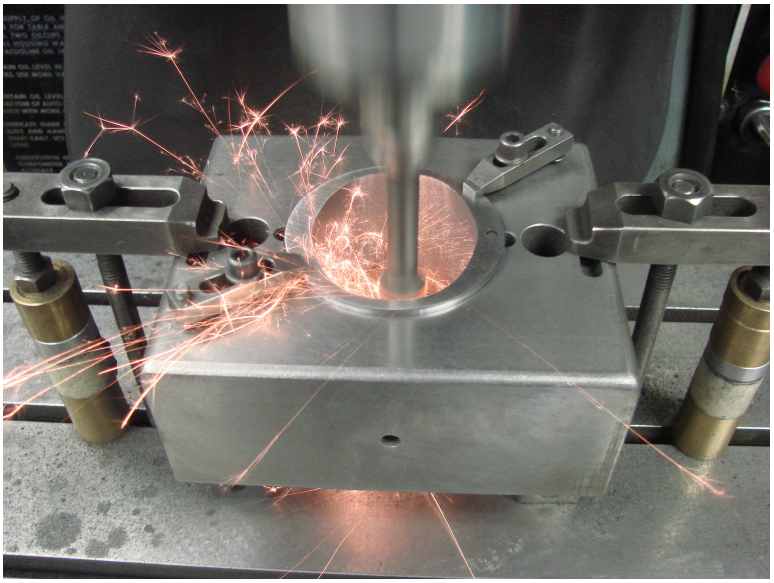


Figura 2. 6 - Máquina CNC de usinagem (S & S PRECISION, 2009)

A segunda maior família de componentes é a do sistema de manuseio e armazenamento de materiais. O sistema de manuseio move as peças pelo FMS e pode ser subdividido em primário, que transporta peças entre as estações e consiste em equipamentos como veículos automaticamente guiados (AGV), mostrado na figura 2.7, veículos guiados por trilho (RGV) e esteiras transportadoras, e em secundários (figura 2.8) que transfere as partes do primário à estação de trabalho posicionando-as adequadamente.

Figura 2. 7 - Exemplo de sistema de manuseio primário (PROMEC, 2009)

Figura 2. 8 - Exemplo de sistema de manuseio secundário (ROBOT MAGAZINE, 2009)

O sistema de armazenamento, além de guardar as matérias-primas, peças semi-acabadas ou prontas, também serve para estocagem de ferramenta das máquinas.

O sistema de controle computacional é usado para controlar as partes automatizadas do sistema por meio de sua interface com os hardwares de FMS como as máquinas CNC e os AGVs. Um sistema de controle típico de FMS é formado por um computador central que controla as atividades realizadas pelo sistema e microcomputadores que controlam os componentes individualmente.

Apesar da grande parte das tarefas de processo de produção em um sistema flexível ocorrer por maquinários automatizados, a intervenção humana é indispensável para o funcionamento do FMS. Portanto, os recursos humanos também podem ser classificados como um dos componentes e são inseridos nas atividades como carga de peças no sistema, descarga de partes prontas, manutenção de equipamentos, entre outras.

3 Autômatos Temporizados

Neste capítulo serão apresentados os fundamentos teóricos sobre autômatos temporizados (ATs) e a técnica de modelagem utilizada no presente trabalho no procedimento que será apresentado no capítulo 5. Os ATs são derivados dos autômatos finitos (AFs) e, portanto, o capítulo iniciará com uma introdução sobre AFs, passando em seguida para uma abordagem sobre ATs. Depois disso será dada uma explicação sobre a forma de comunicação entre os diversos autômatos. Por fim será abordado o assunto síntese de supervisório.

3.1 Autômatos Finitos

Um autômato finito ou máquina de estados, é um formalismo, que permite representar de forma clara, um qualquer processo composto por um conjunto de estados, e transições entre esses estados. A representação dos AFs é feita de maneira gráfica através de círculos, representando os estados, e arcos, representando as transições (ou eventos) entre os estados. Existe também um arco de entrada sem inscrição que indica o estado inicial. Estados representados através de dois círculos são chamados estados finais (BURCH, 2004). A figura 3.1 mostra um exemplo de AF.

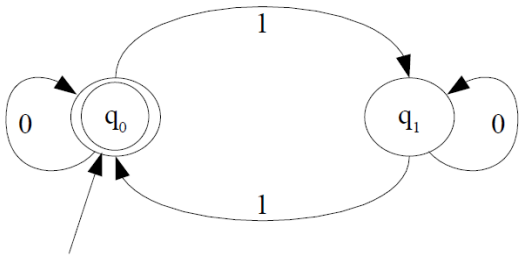


Figura 3. 1 - Exemplo de autômato finito

Mais formalmente um autômato finito é representado por uma tupla $A = (S, \Sigma, s_0, F, \delta)$, na qual:

- S é um conjunto finito de estados não vazio;
- Σ é o alfabeto de entrada, um conjunto finito de símbolos não vazio;
- s_0 é o estado inicial, um elemento de S;
- F é conjunto de estados finais (ou marcados), $F \subset S$;

- δ é a função de transição, recebe como argumentos um estado e um símbolo de entrada e devolve um novo estado (eventualmente o mesmo): $\delta : S \times \Sigma \rightarrow S$. (HOPCROFT; MOTWANI; ULLMAN, 2001)

Ao processar o símbolo associado a uma determinada transição pertencente ao estado atual do autômato, a transição é disparada. Uma *string* sobre um alfabeto Σ é uma seqüência finita de símbolos de Σ, a *string* que não contem nenhum símbolo é representada por ε. O objetivo dos AFs é processar *strings* que podem ser aceitas ou rejeitadas. Se, ao término do processamento da *string*, o autômato encontra-se em um estado final, a *string* é aceita, caso contrário ela é rejeitada. No exemplo da figura 3.1, a *string 101* é um exemplo de *string* que é aceita pelo autômato. Um conjunto de *strings* é chamado de linguagem. Um autômato està associado a duas linguagens: a linguagem gerada (normalmente representada por *L*) que representa todas as cadeias que podem ser seguidas no autômato, partindo do estado inicial; e a linguagem marcada (normalmente representada por L_m) que considera todas as cadeias as quais, partindo do estado inicial, atingem um estado final. (BURCH, 2004).

Um autômato pode ser determinístico ou não determinístico. Autômato finito determinístico (AFD) é aquele em que todos os estados têm uma transição diferente para cada símbolo do alfabeto. Autômato finito não determinístico (AFN) é aquele em que os estados podem ou não ter uma transição diferente para cada símbolo do alfabeto, e ainda podem ter múltiplas transições para o mesmo símbolo partindo de um mesmo estado, portanto os AFDs são um tipo especifico de AFNs. Os AFNs mais poderosos uma vez que possuem a propriedade de representar uma gama maior de classes de processos dinâmicos (HOPCROFT; MOTWANI; ULLMAN, 2001). Um exemplo de AFN é mostrado na figura 3.2.

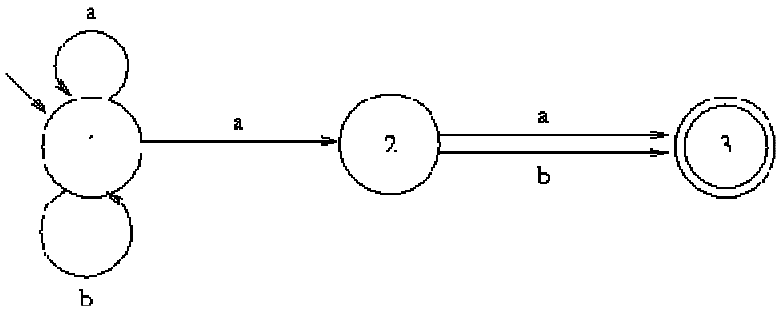


Figura 3. 2 - Exemplo de AFN (HOPCROFT; MOTWANI; ULLMAN, 2001)

A teoria de autômatos é aplicável em vários campos. Por exemplo na informática se pode fazer buscas em textos usando AFs. A maior limitação da

aplicação desta teoria está no fato de que em determinados casos o número de estados necessário para a modelagem do sistema sofre explosão combinatória.

3.2 Autômatos temporizados

Um dos principais modelos de autômato temporizado existentes na literatura é aquele proposto por Alur e Dill (1994), que diz que ATs são AFs comuns estendidos com variáveis relógio. As variáveis relógio são inicializadas com o valor zero quando o sistema começa e são incrementadas de maneira síncrona. Um autômato deste tipo pode ser considerado como um modelo abstrato de um sistema que envolve tempo. Nos ATs, cada transição é associada a condições de restrição (*guards*) que determinam quando a transição está ativada para disparar. Estas condições são da forma $x_i \sim c$ ou $x_i - x_j \sim c$ onde x_i e x_j são variáveis relógio, c é um inteiro constante e $\sim \in \{<, <=, >, >=, ==\}$. As transições podem também ter comandos que zeram as variáveis relógio ou as atualizam para um outro valor inteiro positivo (Fredrik Larsson, 2000). A figura 3.3 mostra um exemplo em que *x* é uma variável relógio.

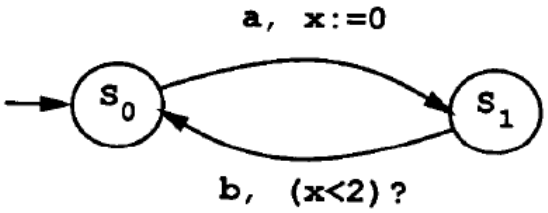


Figura 3. 3 - Exemplo de autômato temporizado (ALUR; DILL, 1994)

3.2.1 Autômato Temporizado de Büchi

A condição de restrição em um arco de um autômato é somente uma condição de ativação da transição representada pelo arco, ou seja, não pode forçar o disparo da transição. Por exemplo, o autômato da figura 3.3 poderia ficar para sempre no estado *S1*, e a partir de *x >= 2*, o sistema estaria em *deadlock*, ou seja, estados impossíveis de serem deixados devido à não presença de transições de saída ou à incapacidade de satisfazer às condições de restrição das transições de saída. No trabalho inicial de Alur e Dill (1990), esse problema é resolvido introduzindo condições

de aceitação para o autômato de Büchi; um subconjunto de estados é marcado como estado final, e somente as execuções que passam por esses estados de maneira freqüente e infinitas vezes são consideradas como comportamento válido para o autômato. No exemplo da figura 3.3, o problema de *deadlock* seria resolvido marcando o estado *S0* como estado final, isso implicaria que o estado *S0* deva ser visitado infinitas vezes, portanto uma vez que o sistema passasse para *S1*, obrigatoriamente voltaria para *S0* antes de $x = 2$.

3.2.2 Timed Safety Automata

Uma versão simplificada de autômato temporizado chamada de *Timed Safety Automata* foi introduzida por Henzinger et al. (1992), para especificar propriedades de progresso usando condições locais invariantes. Devido sua simplicidade, *Timed Safety Automata* tem sido adotado em várias ferramentas de verificação para autômato temporizado, entre elas a ferramenta UPPAAL (apresentada no capítulo 4), utilizada para simulação e verificação no presente trabalho.

Em vez de condições de aceitação, como nos autômatos de Büchi, nos *Timed Safety Automatas*, podem ser colocadas condições de restrição local de tempo, associadas aos estados, chamadas de invariantes de estado. Um autômato deve permanecer em um determinado estado somente enquanto os valores das variáveis relógio satisfizerem a condição invariante daquele estado. Para evitar a possibilidade de que invariantes sejam falsos quando um estado é atingido, e se tornem verdadeiros depois de algum tempo, eles são restritos a fórmulas com os operadores $<$ e $\leq$, isto simplifica as operações computacionais e deixa o *model-checking* mais rápido. No exemplo da figura 3.4 os estados *start* (o círculo duplo representa o estado inicial em UPPAAL), *loop* e *end* possuem condições invariantes relacionadas à variável relógio *y*. Os estados *start* e *end* devem ser deixados quando *y* for no máximo igual a 20 e o estado *loop* deve ser deixado quando *y* for no máximo igual a 50. Dessa forma, garante-se o progresso do sistema e tem-se uma visão local do comportamento de cada estado do autômato em relação ao tempo.

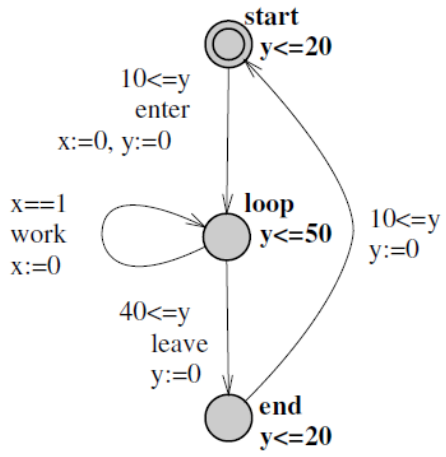


Figura 3. 4 - Exemplo de *timed safety automata* **(BENGTSSON; YI, 2004)**

Seja *C* um conjunto de variáveis relógio e *B(C)* a função que formula condições de restrição, pode-se representar um *timed safety automata*, de maneira formal, através de uma tupla $A=(L, l_0, E, g, r, I)$, onde:

- *L* é um conjunto de estados;
- $l_0 \in L$ é o estado inicial;
- $E \in L \times L$ é o conjunto de transições;
- $g: E \rightarrow B(C)$ são as condições de restrição nas transições;
- $r: E \rightarrow 2^C$ são as variáveis relógio a serem atualizadas nas transições;
- $I: L \rightarrow B(C)$ são os invariantes de estado. (HENZINGER et al., 1992)

3.2.3 Variáveis de Dados

A linguagem de modelagem através de autômatos temporizados utilizada na ferramenta UPPAAL é ainda estendida com variáveis de dados. Estas variáveis são do tipo inteiro ou booleano e podem ser usadas em condições de restrição para ativação de transições, como também podem ser atualizadas a outros valores no disparo da transição. Portanto, para que uma transição seja ativada, ela deve satisfazer não somente às condições impostas pelas variáveis relógio mas também àquelas impostas pelas variáveis de dados. Estas variáveis podem ainda ser usadas na formulação de condições de restrição envolvendo variáveis relógio. Com o uso de variáveis deste tipo é possível especificar melhor a condição em que o sistema se encontra quando está em um determinado estado. A figura 3.5 mostra um exemplo de *timed safety automata* com variáveis de dados em que *d* é uma variável de dado e *t* é uma variável relógio.

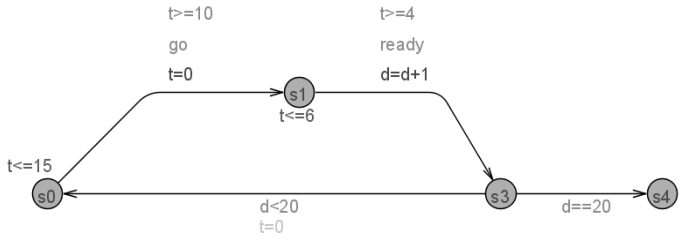


Figura 3. 5 - Exemplo de *timed safety automata* com variável de dado

No restante deste trabalho *timed safety automata* com variáveis de dados será referido apenas como autômato temporizado.

3.3 Comunicação entre Autômatos

Uma das principais características da modelagem de autômatos temporizados em UPPAAL é a possibilidade da criação de uma rede de autômatos temporizados. Esta funciona como uma composição paralela (ou síncrona) $A_1 \mid ... \mid A_n$ de um conjunto de autômatos temporizados A_1, ..., A_n , chamados de processos, combinados em um único sistema (BENGTSSON; YI, 2004). A comunicação entre estes processos pode ser feita, de maneira síncrona, por meio de canais de comunicação de sincronismo ou, de maneira assíncrona, através de variáveis de dados compartilhadas.

Na maneira síncrona, o canais de comunicação são representados por símbolos ou palavras, acompanhados do sufixo *!* e *?* que indicam emissão e recepção, respectivamente. Um exemplo disto é visto na figura 3.6 onde o canal de comunicação representado pela evento *press* é emitido pelo autômato da direita e recebido pelo autômato da esquerda, sincronizando assim o disparo das transições de ambos.

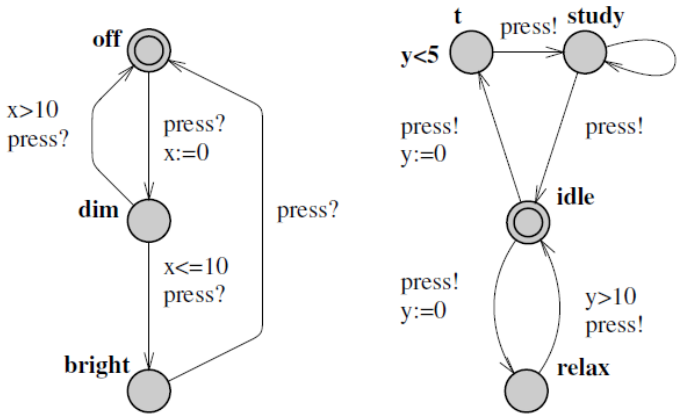


Figura 3. 6 - Exemplo de autômato temporizado com canal de comunicação de sincronismo (BENGTSSON; YI, 2004)

Caso não fosse possível implementar canais de comunicação seria necessário implementar-se autômatos com várias outras combinações de transições de estados provocando uma evolução considerável no número de elementos do autômato. Por exemplo, a figura 3.7 mostra o autômato que substitui o par de autômatos sincronizados da figura 3.6.

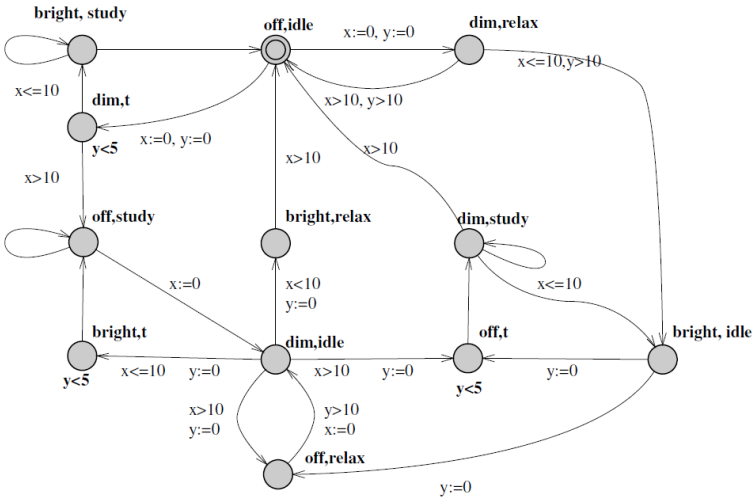


Figura 3. 7 - Composição paralela entre os autômatos da figura 3.6 (BENGTSSON; YI, 2004)

A maneira assíncrona é efetuada quando, dada uma variável de dados compartilhada por dois autômatos, uma atualização desta variável em um dos autômatos possibilita a satisfação de uma condição de restrição no outro autômato. Esta situação é ilustrada nos autômatos da figura 3.8 que compartilham a variável de dado *x* e a variável relógio *t*.

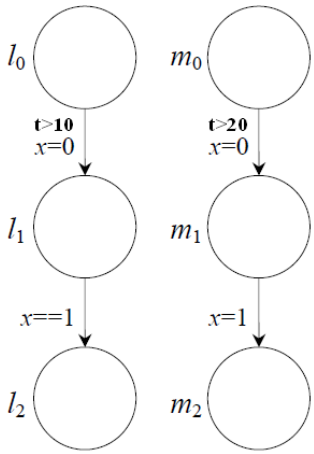


Figura 3. 8 - Exemplo de comunicação entre autômatos através de variáveis compartilhadas

Pode-se observar no exemplo da figura 3.8 que o autômato do lado esquerdo só poderá atingir o estado l_2 quando o autômato do lado direito atingir o estado m_2.

A ferramenta UPPAAL possui ainda dois outros recursos que são importantes nessa questão de rede de autômatos temporizados. Um deles é o chamado canal urgente, que é um canal de comunicação de sincronismo comum, mas com a característica adicional que impossibilita atrasos uma vez que a transição relacionada ao canal urgente é ativada (BENGTSSON; YI, 2004). No exemplo da figura 3.9, esse comportamento é mostrado mais claramente. Nota-se que ambos os processos podem passar de maneira independente do primeiro para o segundo estado. No segundo estado, o primeiro processo deve atrasar por pelo menos 10 unidades de tempo antes que seja permitida a sua sincronização no canal urgente. No segundo estado, o segundo processo deve atrasar por pelo menos 5 unidades de tempo antes que seja permitida a sua sincronização no canal urgente. Assim que os dois processos tiverem passado o mínimo de período de tempo requerido em seus segundos estados, eles devem se sincronizar e passar para os seus terceiros estados.

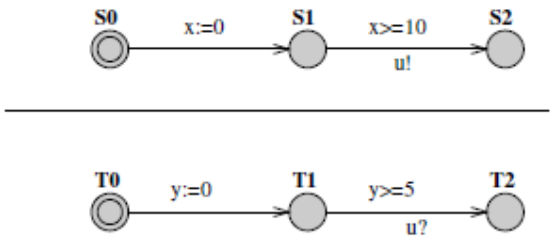


Figura 3. 9 - Exemplo de sincronização com canais urgentes (BENGTSSON; YI, 2004)

O outro recurso é o chamado estado compromissado, que é uma espécie de estado virtual do sistema e que, uma vez atingido, deve ser deixado imediatamente sem atraso, com prioridade sobre qualquer outro estado, ou seja, a próxima transição a disparar deve ser obrigatoriamente aquela do estado compromissado. A utilização de estados compromissados permite um melhor desempenho computacional nas verificações, uma vez que eles não são armazenados na memória. Além disso é possível realizar sincronizações múltiplas através da interação entre estados compromissados e canais de comunicação de sincronismo. A figura 3.10 mostra um exemplo de um autômato com um estado compromissado. Neste exemplo, a variável *x* é atualizada no mesmo instante em que o canal de comunicação *update_variable* é sincronizado. A letra *c* indica o estado compromissado.

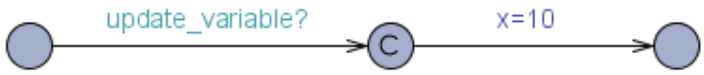


Figura 3. 10 – Exemplo de autômato com estado compromissado

3.4 Síntese do Supervisório

A teoria de controle supervisório elaborada por Wonham e Ramadge (1989) define que a modelagem de sistemas a eventos discretos (SEDs) através de autômatos deve ser feita em duas partes. A primeira consiste na modelagem do sistema a ser controlado, chamado de planta do sistema, que corresponde, em geral, a um conjunto de subsistemas (equipamentos) não coordenados, arranjados segundo uma distribuição espacial dada. O controle da planta é realizado pela segunda parte por meio de um autômato denominado de supervisor, o qual restringe o comportamento do sistema físico (planta), satisfazendo a um conjunto de especificações, de forma que a função coordenada a ser executada pelo sistema global seja cumprida. Portanto, o supervisor interage com a planta, observando os eventos ocorridos e define, de acordo com o estado atual da planta, quais eventos fisicamente possíveis são habilitados. A figura 3.11 ilustra a estrutura de controle em malha fechada de uma planta *G* sob ação do supervisor *S*.

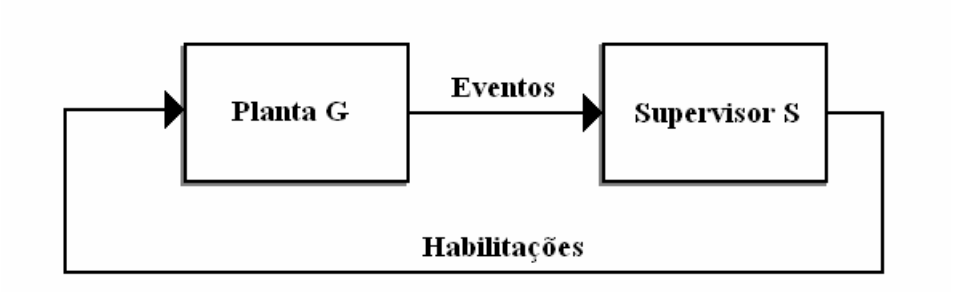


Figura 3. 11 - SED em malha fechada (CURY, 2001)

A planta possui eventos controláveis, suscetíveis a intervenções externas de controle, e eventos não controláveis, cuja ocorrência independe de ações de controle. Eventos controláveis são designados por arcos cortados por uma pequena linha transversal.

Portanto, para resolver-se um problema de controle supervisório, segundo Wonham e Ramadge (1989), o seguinte procedimento deve ser executado:

1. Modelar o comportamento da planta sem coordenação (planta livre);
2. Modelar as especificações de controle;
3. Utilizando os modelos obtidos nos passos anteriores, sintetizar o supervisor.

Na modelagem da planta, cada subsistema, de acordo com sua autonomia, deve ser modelado por um autômato, e o comportamento global (sem coordenação) é obtido pela composição síncrona dos modelos individuais. Da mesma forma, cada especificação de controle é modelada por um autômato de forma isolada, e a composição síncrona de todas as especificações resulta no autômato que modela a especificação global para o sistema. Por fim, faz-se a composição síncrona do autômato da planta com o autômato da especificação global gerando a modelagem que representa o sistema sob supervisão.

Formalmente, pode-se definir um sistema controlado por um supervisor da seguinte maneira:

- $G = (Q, \Sigma, q0, \delta, Q_m)$ é a planta;
- $\Sigma = \Sigma_c \cup \Sigma_u$ são os eventos separados em controláveis e não controláveis;
- $\Gamma = \{\gamma \in 2^{\Sigma} : \gamma \supseteq \sum_u\}$ é a estrutura de controle para G, onde $\gamma \in \Gamma$ é uma opção de controle;
- *S* é o supervisor sobre o mesmo alfabeto de *G* e suas mudanças de estado são ditadas por ocorrências de eventos na planta;
- *S/G=S||G* é o sistema controlado, no qual *S* habilita e desabilita os eventos controláveis de G;

- $L(S/G)=L(S||G)$ e $L_m(S/G)=L_m(S||G)$ são as linguagens gerada e marcada, respectivamente, que definem o comportamento do sistema.

A principal dificuldade na implementação da teoria de controle supervisório está no fato de que a síntese de controladores para SED é um procedimento computacional cuja complexidade cresce com o número de estados dos modelos da planta e das especificações. Com o aumento do número de componentes que integram o sistema, o número de estados do sistema modelado tende a explodir. Essa limitação tem sido intensamente tratada na literatura para viabilizar sua aplicação em problemas de grande porte. Uma outra limitação da teoria está relacionada à inclusão ou alteração de subsistemas no modelo, o que acarreta na reconstrução de todo o modelo.

4 Verificação de Modelos

O capítulo será iniciado com uma breve introdução sobre verificação de modelos, passando em seguida para a apresentação da ferramenta UPPAAL. Depois disso será explicada a importância do uso dos *templates* - recurso da ferramenta utilizada no presente trabalho – no processo de modelagem. Por fim será apresentada a lógica de verificação utilizada pela ferramenta.

4.1 Introdução à verificação de modelos

A verificação de modelos é uma das principais atividades no projeto de um sistema de controle. Com este método pode-se assegurar que o sistema irá se comportar como o esperado sob quaisquer possíveis situações de operação (MIYAGI; VILLANI; VALETTE, 2006). Antigamente, esta tarefa era executada através de testes manuais de validação, no entanto, por maior que fosse o número de testes efetuado, não se podia ter certeza sobre o correto comportamento do sistema. Com o aumento do uso de sistemas em tempo real, tem se tornado cada vez mais importante o desenvolvimento de métodos de verificação, ainda mais que muitos destes sistemas são embarcados e atuam em cenários críticos e perigosos, de modo que uma falha de projeto do sistema poderia causar grandes danos. Um dos principais métodos de verificação é o *model-checking*. Uma ferramenta completamente automática, através de algoritmos, examina se os estados de um modelo (descrição comportamental de um sistema) satisfazem uma determinada propriedade (NIELSEN, 2000]. Essa checagem é feita pelo usuário por meio de proposições lógicas que retornam o valor verdadeiro ou falso.

4.2 A ferramenta UPPAAL

UPPAAL é uma ferramenta computacional, com acesso livre na internet, integrada com ambientes para modelagem, validação e verificação de sistemas em tempo real modelados como rede de autômatos temporizados estendidos com variáveis de dados. Ela foi desenvolvida em conjunto pela Universidade de Uppsala com a Universidade de Aalborg e tem sido aplicada com sucesso em vários casos

reais (DAVID; YI, 2000). A primeira versão de UPPAAL foi lançada em 1995 e, desde então, vem sendo alterada e desenvolvida, melhorando sua performance e proporcionando mais recursos aos usuários. A ferramenta é caracterizada por uma interface que usa Java e seu motor de verificação é escrito em C++. Além disso, ela é baseada em uma arquitetura *client-server* onde a interface gráfica atua como *client*. A interface gráfica proporciona um editor (figura 4.1), de uso fácil para desenhar autômatos temporizados, um simulador (figura 4.2) e uma interface para interagir com o *model-checker* (figura 4.3). Um dos aspectos fundamentais da ferramenta UPPAAL está no *server* que trata de particionar o modelo, interpretando-o, e proporcionando funcionalidades para o *model-checker.* Os algoritmos usados nesta parte são bastante complicados e foram otimizados para reduzir o tempo de processamento e o uso de memória da ferramenta (BEHRMANN, 2003).

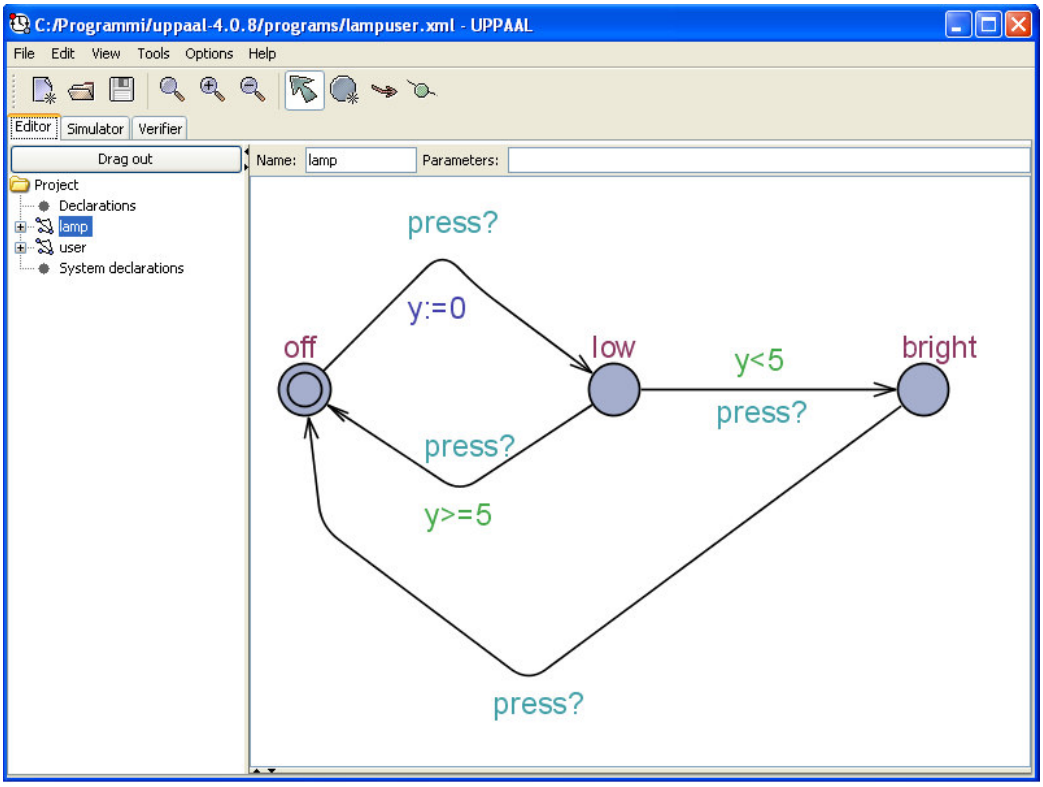


Figura 4. 1 - Ambiente de modelagem em UPPAAL

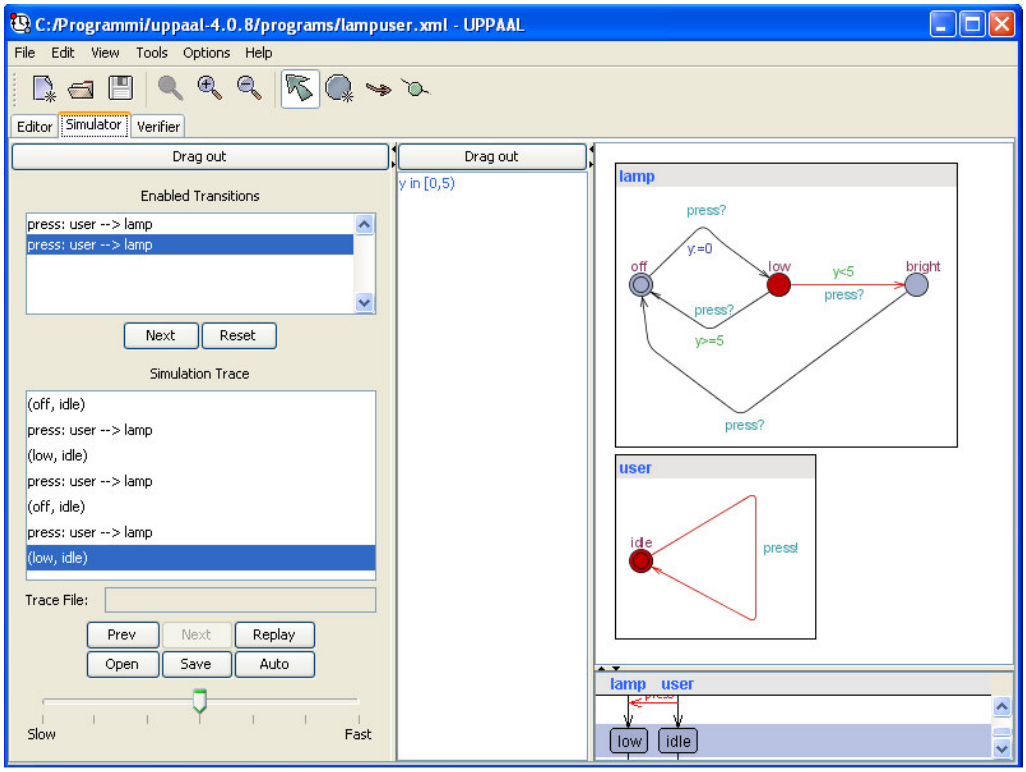


Figura 4. 2 - Ambiente de simulação em UPPAAL

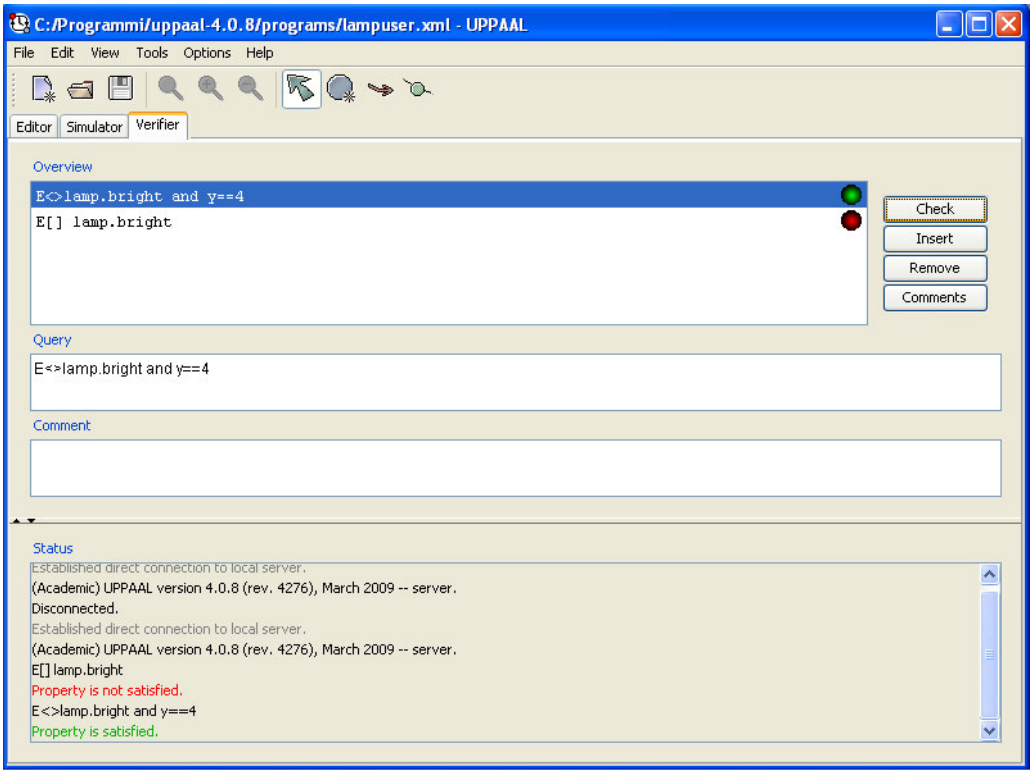


Figura 4. 3 - Ambiente de verificação em UPPAAL

4.3 Proposta de uso de *templates* da ferramenta UPPAAL

Considerando o fato de que FMSs possuem processos e dispositivos que podem ser estruturados de forma modular, um atributo da ferramenta UPPAAL de suma importância na realização do presente projeto é que ela possibilita a representação de modelos dos processos do sistema através do uso de *templates*. *Templates* são autômatos definidos com um conjunto de parâmetros (declarados globalmente) os quais podem ser do tipo *int, bool* ou *chan*. Estes parâmetros são substituídos por argumentos dados na declaração dos processos (BEHRMANN; DAVID; LARSEN , 2004). Os *templates* podem ainda ter declarações locais de variáveis. O uso desse recurso facilita bastante a construção de modelos com vários autômatos similares, que se distinguem apenas por diferentes parâmetros. *Templates* do mesmo tipo, mas parametrizados com diferentes parâmetros, são chamados através de diferentes canais de comunicação de sincronismo. Um exemplo disto pode ser visto na figura 4.4.

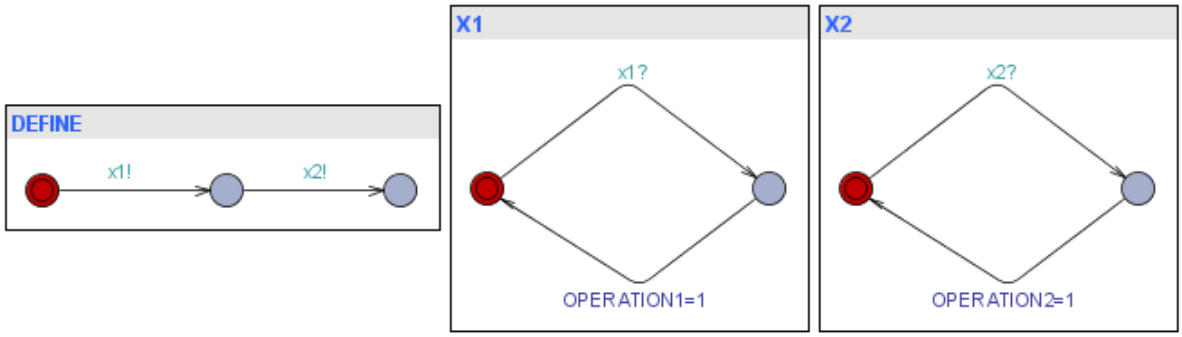


Figura 4. 4 - Exemplo do uso de *templates* em UPPAAL

Neste exemplo, o *template DEFINE,* que não possui parâmetros, chama dois *templates X1* e *X2*. Estes são *templates* do tipo *X*, que foi modelado com dois parâmetros: *chan x* e *bool OPERATION*. Os parâmetros *x1, x2, OPERATION1* e *OPERATION2* são declarados globalmente e alocados aos seus respectivos *templates* na declaração destes.

Outra contribuição importante do recurso do uso de *templates* no presente projeto é dada na parte de diminuição da complexidade do modelo global do sistema através da proposta de controle atômico e divisão hierárquica do modelo explicada anteriormente.

4.4 Lógica de verificação

A ferramenta UPPAAL utiliza uma linguagem declarativa do tipo *query* que é uma versão simplificada da lógica CTL (*computational tree logic*). Essa versão simplificada usa proposições que representam perguntas do tipo “Partindo do estado inicial, pode algum estado ser alcançado?” ou “Partindo do estado inicial, pode-se ter certeza que algum estado não será alcançado?” (BENGTSSON; LARSSON, 2006). Desse modo é possível checar propriedades de alcançabilidade, em particular se certas combinações de estados e restrições nas variáveis relógio e de dados são alcançadas partindo do estado inicial.

Como em CTL, a linguagem *query* usada em UPPAAL consiste de *path formulae* (fórmulas de caminho) e *state formulae* (fórmulas de estado).

Uma fórmula de estado é uma expressão que pode ser avaliada para um estado sem olhar para o comportamento do modelo. A sintaxe desse tipo de fórmula é a mesma dos *guards* (mencionados no item 3.2), porém com a possibilidade do uso de operadores lógicos (*or, and, imply, not*) entre as fórmulas. Como exemplo tem-se a simples fórmula *c == 10* que será verdadeira quando *c* for igual a 10. Utilizando expressões do tipo *P.s*, onde *P* é um processo e *s* é um estado, é possível também checar se um determinado processo está no estado *s*. A ferramenta UPPAAL ainda tem o recurso de expressar *deadlocks* usando um tipo especial de fórmula de estado, esta consiste simplesmente da palavra *deadlock* e é satisfeita para todos os estados em *deadlock*.

Path formulae podem ser classificadas em alcançabilidade, segurança e *liveness*. Fórmulas de alcançabilidade exprimem se existe algum caminho, partindo do estado inicial que leve a um determinado estado, ou seja, checa se eventualmente aquele estado será atingido. Em UPPAAL essa expressão é da forma *E<>s*, onde *s* é uma *state formulae*. Fórmulas de segurança expressam situações como “algo ruim nunca acontecerá” ou “algo ruim possivelmente nunca acontecerá”. Em UPPAAL, dada uma fórmula de estado *s*, esse tipo de propriedade é formulado positivamente com as expressões *A[] s*, que indica que *s* deve ser verdadeira em todos os estados alcançáveis, e *E[] s* indicando que existe um caminho em que s é sempre verdadeira. Fórmulas *liveness* indicam casos do tipo “algo eventualmente ocorrerá”. Em UPPAAL, dadas duas fórmulas de estado *s* e *g*, situações assim podem ser avaliadas com as fórmulas *A<> s*, que significa que *s* será eventualmente satisfeita, ou *s --> g*, que indica que quando *s* for satisfeita, *g* será eventualmente satisfeita (BEHRMANN;

DAVID; LARSEN , 2004). A figura 4.5, onde φ e ψ representam fórmulas de estado, ilustra as fórmulas de caminho aceitas em UPPAAL.

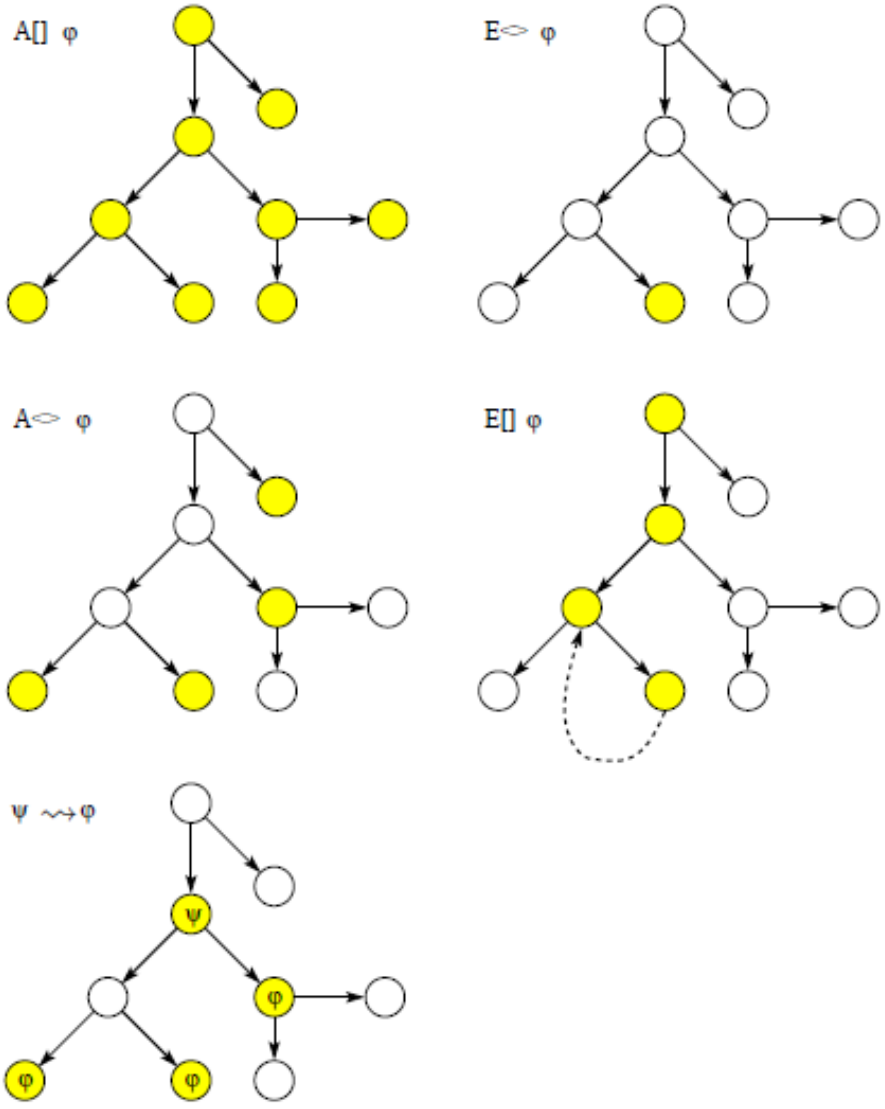


Figura 4. 5 - *Path formulae* em UPPAAL (TUTORIAL UPPAAL, 2004)

5 Sistematização da verificação de FMS

Este capítulo apresentará detalhadamente a proposta de modelagem sugerida pelo presente projeto. Inicialmente serão abordadas as considerações feitas no procedimento adotado na modelagem, seguidas pela apresentação do modelo estrutural proposto. Depois disso, serão apresentados os *templates* relacionados aos componentes de FMS considerados e, por fim, serão explicados os passos para a modelagem dos *templates* da parte de controle sobre estes componentes.

5.1 Considerações iniciais

O procedimento proposto pelo presente projeto tem por objetivo desenvolver uma biblioteca de componentes de FMS parametrizáveis, com o intuito de auxiliar na construção de um modelo global de um FMS para, posteriormente, verificar e simular o mesmo. Para isso, foi proposto também uma metodologia sintética de construção do modelo de controle sobre os componentes. Os modelos propostos foram realizados em autômatos temporizados na ferramenta computacional UPPAAL, capaz de sintetizar redes de autômatos. Inicialmente foram criados os modelos dos componentes, com seus devidos parâmetros, representando a planta física do FMS. Posteriormente foi criado o modelo de controle, formado por duas partes distintas, uma que representa o seqüenciamento de atividades de um processo de fabricação de um determinado produto, e a outra que indica quando novos processos podem iniciar possibilitando a descrição de um sistema onde diversos produtos são trabalhados simultaneamente.

É importante citar que, devido à modularidade dos modelos dos componentes, a proposta de modelagem apresentada permite que o modelo global sofra alterações sem a necessidade remodelá-lo do início.

5.2 Modelo estrutural do sistema

Para evidenciar as relações e interações que ocorrem dentro do sistema global, será apresentado o modelo estrutural do mesmo proposto pelo presente trabalho. Para construir o modelo, podemos dividir o sistema em duas grandes partes:

a parte operativa (controlada) e a parte de controle. O presente projeto definiu que a parte controlada fosse limitada a 3 classes de componentes: os componentes de transformação, de transporte e de manipulação. Esta escolha foi feita, primeiramente porque o universo de componentes de um FMS é muito amplo e a consideração de todos estes componentes existentes atualmente poderia inviabilizar projeto. Além disso, as três classes selecionadas são suficientes para abstrair vários tipos de processo de manufatura que envolvem flexibilidade. O componente transporte considerado consiste de um dispositivo que segue um trajeto pré-definido e tem a capacidade de transportar somente uma peça por vez. Como componente de manipulação, foi considerado um manipulador que consiga mover objetos (no caso, podem ser materiais para serem fabricados ou peças semi-acabadas ou peças prontas) e no sistema global tem a função de carregar e descarregar estações de trabalho e esteiras transportadoras posicionando as partes adequadamente. Por fim, como componente transformação, foi escolhida uma estação de trabalho flexível capaz de realizar diversos tipos de operações de manufatura. Cada um destes componentes se comunica com o controle por meio de variáveis compartilhadas e canais de comunicação de sincronismo. É importante enfatizar que as comunicações existentes nos componentes se resumem estritamente àquelas com o controle, ou seja, não ocorrem comunicações entre os componentes do objeto de controle (planta).

A parte de controle é constituída por uma parte de processos, onde cada autômato descreve a seqüência de atividades na fabricação de um determinado produto, e outra parte de supervisão que determina o instante quando o processo de fabricação de um produto pode iniciar. Neste sistema de controle existe uma hierarquia, de forma que o os autômatos dos processos envolvidos são chamados pelo supervisor. Essa modelagem origina uma rede de autômatos integrados por elementos de comunicação que evitam a complexidade do modelo global. Existe ainda uma hierarquia entre a parte de controle e os componentes já que os mesmos são chamados pelos processos. Uma representação gráfica que exprime a relação hierárquica do modelo é mostrada na figura 5.1.

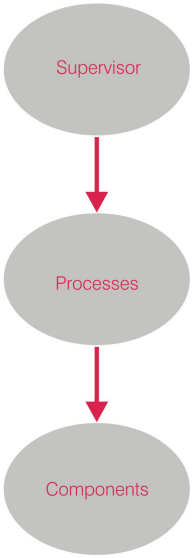


Figura 5. 1 - Relação hierárquica dos autômatos

Outra parte bem definida pelo modelo estrutural proposto diz respeito à comunicação entre os diversos autômatos. A figura 5.2 representa entre quais tipos de autômatos essa comunicação ocorre.

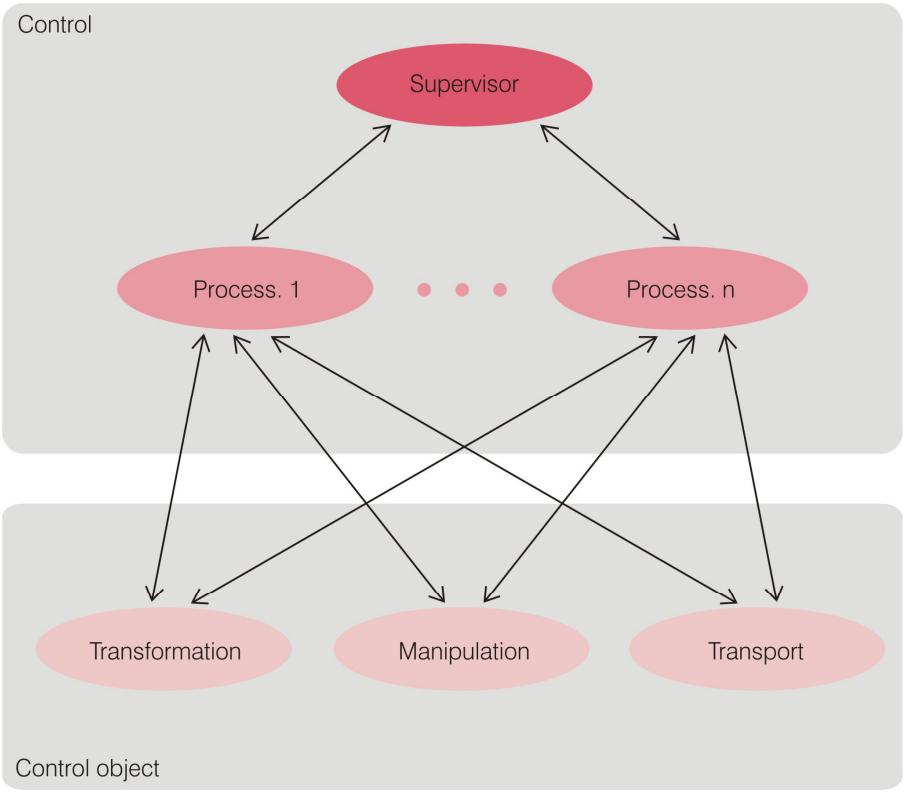


Figura 5. 2 - Comunicação entre os autômatos

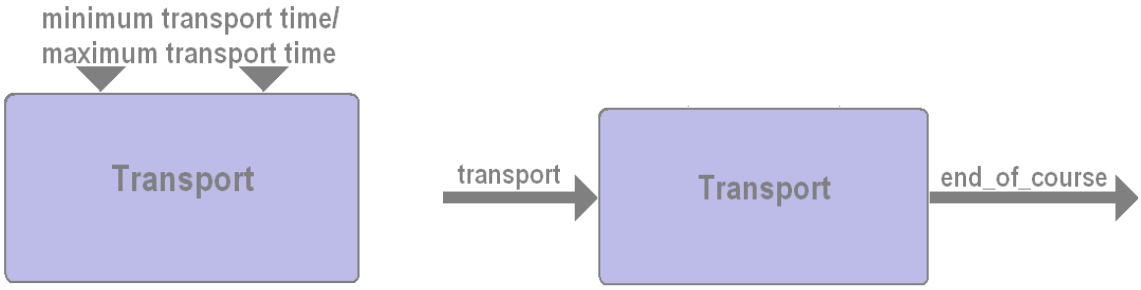
Nota-se que existe uma troca de dados e sincronizações, em ambas as direções, entre o autômato de supervisor com aqueles de processo, isto também ocorre entre os autômatos de processo com os de componente. É importante ressaltar que não há comunicações entre os autômatos de componente, nem entre os autômatos de processo, como também não há entre os autômatos de componente com o supervisor.

5.3 Desenvolvimento da biblioteca de componentes

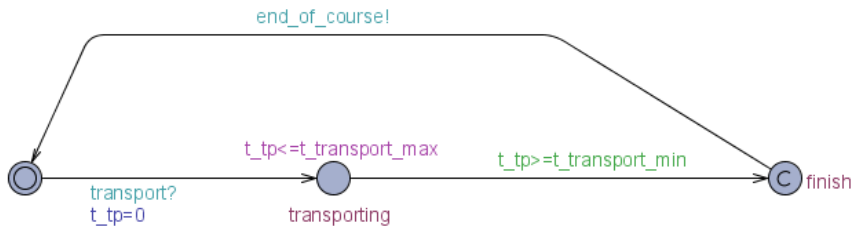
Como citado no item 5.2, o presente projeto considerou FMSs compostos por três tipos de componente: transporte, manipulação e transformação. Estes são de construção fixa, portanto, o usuário deverá definir apenas os parâmetros.

5.3.1 Modelagem do componente transporte

O modelo do componente transporte proposto apresenta um estado inicial indicando que o componente está desativado, seguido por um estado que representa que uma peça está sendo transportada, e depois um outro estado mostrando que a peça chegou ao fim do percurso. A partir deste último, o estado inicial pode voltar a ser ativado. A realização deste ciclo se completa em certo período de tempo pertencente a um intervalo definido por um tempo máximo de transporte e um tempo mínimo de transporte, estes são parâmetros que devem ser definidos pelo usuário (figura, 5.3a). O modelo proposto para o componente transporte é ilustrado na figura 5.3c.



(a) (b)



(c)

Figura 5. 3 - Modelo do componente transporte e seus parâmetros

O *template* em questão tem como parâmetros os canais de comunicação *transport* e *end_of_course* (figura, 5.3b), e as variáveis inteiras *t_transport_max* e *t_transport_min*, além disso, possui uma variável de relógio *t_tp* declarada localmente. A ativação do *template* se dá através do canal de comunicação *transport* que recebe um sinal do *template procedure* (tratado no item 5.4) indicando que o transporte deve começar, este canal é declarado como urgente para que não tenha atraso no início da atividade de transporte. Neste instante, a variável relógio local *t_tp* é zerada e o autômato passa para o estado *transporting*. Este será obrigatoriamente deixado, devido ao invariante de estado e à condição de restrição na transição, quando *t_tp* pertencer ao intervalo fechado definido pelos valores de *t_transport_min* e *t_transport_max*. Uma vez atingido, o estado *finish* deve ser deixado imediatamente enviando uma mensagem ao *template procedure*, através do canal de comunicação *end_of_course*, de modo que não haja atraso entre o fim da operação de transporte e o envio do sinal. Este último estado é compromissado pois não representa um estado real do componente, mas foi inserido no modelo devido à sua possível utilidade na etapa de verificação.

5.3.2 Componente manipulação

O modelo do componente manipulação é constituído por dois ciclos que compartilham o mesmo estado inicial o qual descreve manipulador como desativado. No ciclo que representa operação de carga, o estado inicial é seguido por um estado que indica o manipulador carregando uma peça e, quando a operação se completa, o autômato segue para outro estado, indicando o fim da operação de carga. No outro ciclo, que por sua vez representa a operação de descarga, o estado inicial é seguido

por um estado que indica o manipulador descarregando uma peça e, quando a operação se completa, o autômato segue para um outro estado, indicando o fim da operação de descarga. A partir destes últimos, o estado inicial pode voltar a ser ativado. Os ciclos se completam em um intervalo de tempo determinado por tempos mínimos e máximos de carga e descarga os quais são parâmetros definidos pelo usuário (figura 5.4a). O modelo proposto para o componente manipulação é ilustrado na figura 5. 4c.

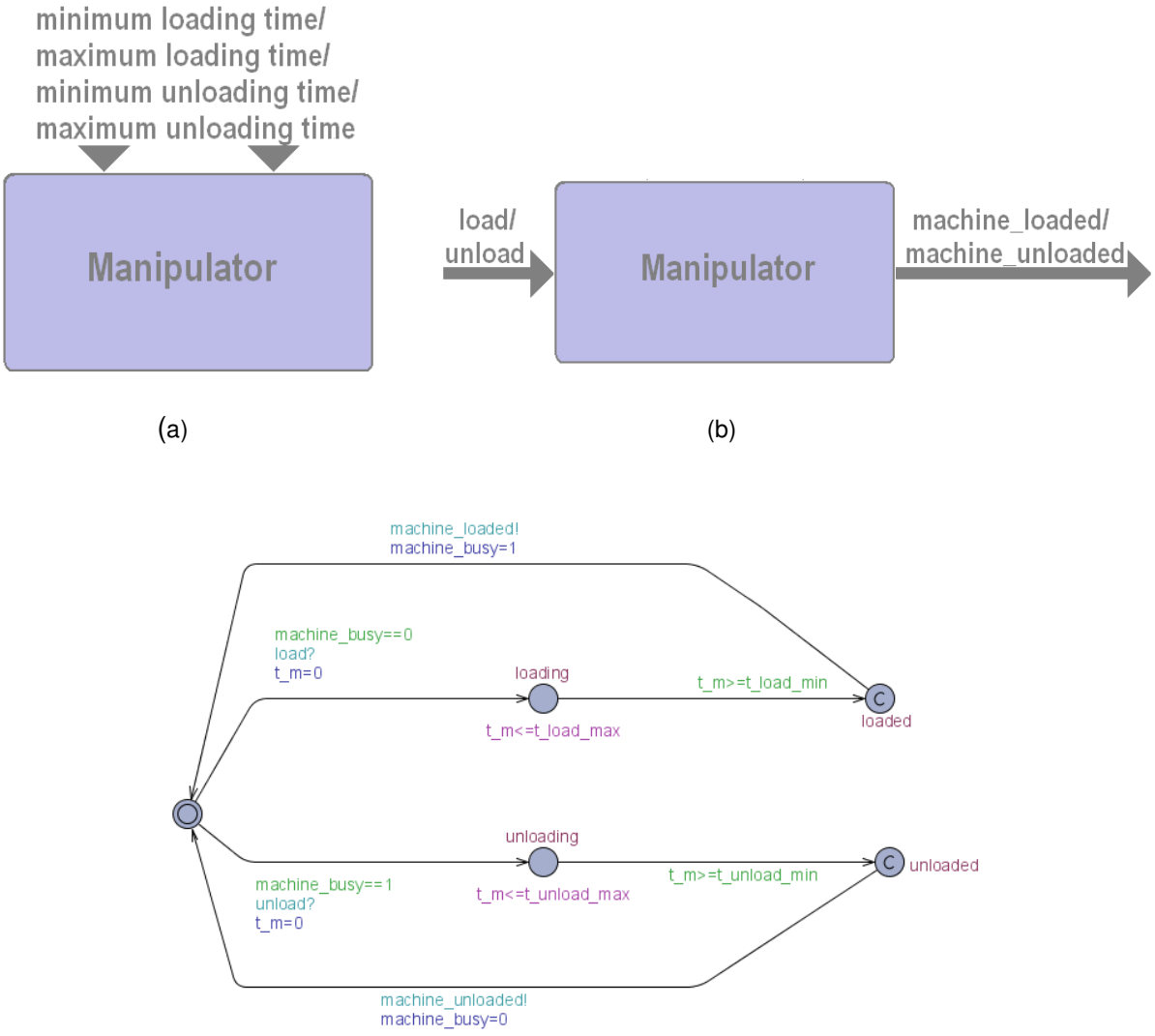


Figura 5. 4 - Modelo do componente manipulação e seus parâmetros

O *template* em questão tem como parâmetros os canais de comunicação *load, unload, machine_loaded* e *machine_unloaded* (figura 5.4b), e as variáveis inteiras

t_load_max, *t_load_min*, *t_unload_max* e *t_unload_min* , além disso possui uma variável relógio *t_m* declarada localmente. Pode-se notar a presença de outro parâmetro, o *machine_busy,* variável do tipo booleana, que indica o estado da estação (ocupada ou desocupada) onde a peça é carregada ou descarregada. A ativação do *template*, no ciclo de carga, se dá através do canal de comunicação *load*, quando um sinal for recebido pelo *template procedure* e o parâmetro *machine_busy* for igual a zero, ou seja, a estação referente estiver livre. O canal de comunicação *load* é declarado como urgente para que não haja atraso no início da atividade de carregamento da peça. Neste instante, o ciclo é iniciado, a variável relógio local *t_m* é zerada e o autômato passa para o estado *loading*. Este será obrigatoriamente deixado, devido ao invariante de estado e à condição de restrição na transição, quando *t_m* pertencer ao intervalo fechado definido pelos valores de *t_load_min* e *t_load_max*. Uma vez atingido, o estado *loaded* deve ser deixado imediatamente enviando uma mensagem ao *template procedure*, através do canal de comunicação *machine_loaded*, de modo que não haja atraso entre o fim da operação do manipulador e o envio do sinal. Ao mesmo tempo, o parâmetro *machine_busy* é atualizado para o valor *1*, indicando que a estação carregada está ocupada. Como no caso do componente transporte, este último estado é compromissado, pois não representa um estado real do componente, mas foi inserido no modelo devido à sua possível utilidade na etapa de verificação.

O *template* também pode ser ativado pelo canal de comunicação *unload*, quando um sinal for recebido pelo *template procedure* e o valor da variável *machine_busy* for igual a *1*, ou seja, se existir uma peça a ser descarregada na estação referente, assim sendo a transição ocorrerá e o ciclo de descarga se iniciará. O canal de comunicação *unload* também é declarado como urgente para que não tenha atraso no início da atividade de descarregamento da peça. Neste instante, a variável relógio local *t_m* é zerada e o autômato passa para o estado *unloading*. Este será obrigatoriamente deixado, pelo fato de existir um invariante no estado e uma condição de restrição na transição, quando *t_m* pertencer ao intervalo fechado definido pelos valores de *t_unload_min* e *t_unload_max*. Uma vez atingido, o estado compromissado *unloaded* deve ser deixado imediatamente enviando uma mensagem ao *template procedure*, através do canal de comunicação *machine_unloaded*, de modo que não haja atraso entre o fim da operação de descarga e o envio do sinal. Também neste caso, o parâmetro *machine_busy* é atualizado, mas agora com o valor zero para

indicar que a estação foi descarregada e está disponível para operações seguintes. Este último estado também é compromissado pela mesma razão do último estado do ciclo de carga.

O parâmetro *machine_busy* foi introduzido neste autômato para evitar uma situação clássica de *deadlock* que ocorre no sistema em que existe um único manipulador para carregar e descarregar uma determinada máquina. O problema ocorre quando o manipulador pega uma peça para carregar a estação e a mesma se encontra ocupada. Esta é uma situação de travamento de sistema, pois a estação deve esperar o manipulador se disponibilizar enquanto o manipulador aguarda a disponibilidade da máquina. A implementação com este parâmetro impede que o manipulador pegue a peça quando a máquina está ocupada evitando o bloqueio do sistema. Além de evitar esse clássico *deadlock*, o parâmetro *machine_busy* evita também uma situação incoerente do sistema que seria o carregamento de duas peças em uma mesma máquina.

5.3.3 Componente transformação

A modelagem escolhida para representar o componente transformação é composta por um estado inicial, representando que o componente está desativado, seguido por um estado que representa que uma peça está sendo trabalhada, e depois um outro estado mostrando que a máquina terminou o seu serviço. A partir deste último, o estado inicial pode voltar a ser ativado. A realização deste ciclo se completa em um certo período de tempo pertencente a um intervalo definido por um tempo máximo de transformação e um tempo mínimo de transformação, parâmetros dados pelo usuário (figura 5.5a). O modelo proposto para o componente transformação é ilustrado na figura 5.5c.

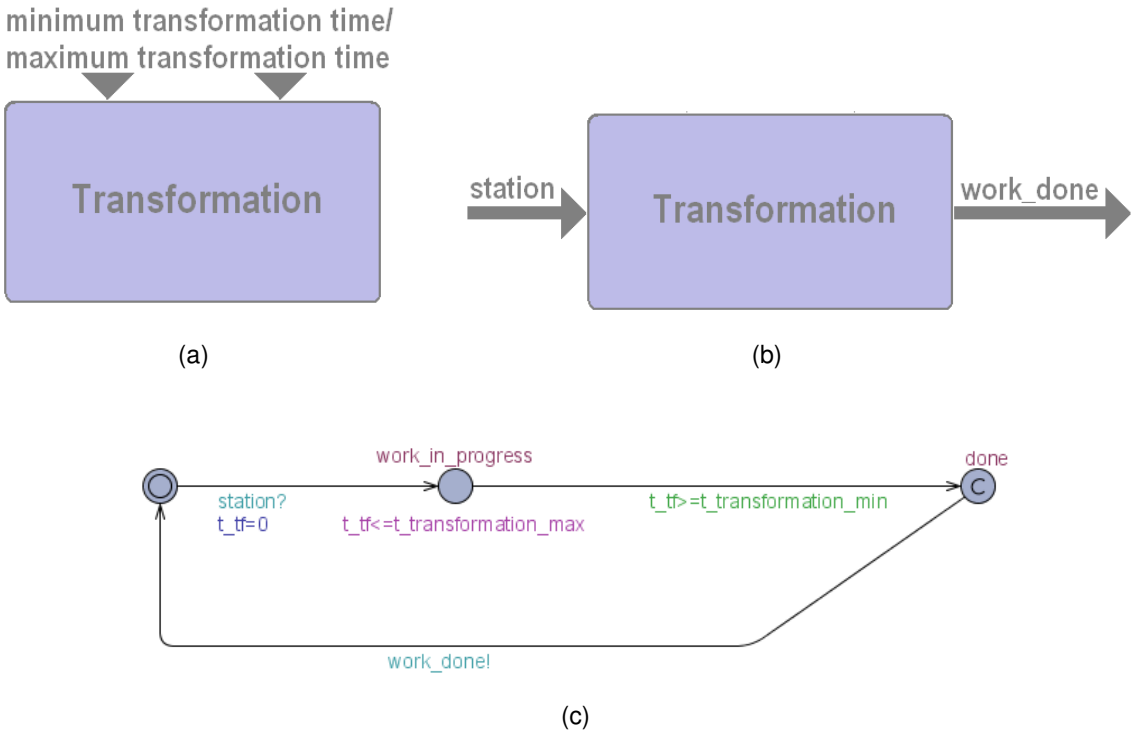


Figura 5. 5 - Modelo do componente transformação e seus parâmetros

O *template* em questão tem como parâmetros os canais de comunicação *station* e *work_done* (figura 5.5b), e as variáveis inteiras *t_transformation_max, t_transformation_min* e *operation*, além disso possui uma variável relógio *t_tf* declarada localmente. A variável *operation*, de acordo com o seu valor, define para qual operação a máquina está designada a trabalhar, cabe ao usuário associar o valor da variável com um tipo específico de operação. Por exemplo, se *operation = 3*, o usuário pode decidir que o valor *3* equivale à operação de torneamento. A ativação do *template* se dá através do canal de comunicação *station* que recebe um sinal do *template procedure* indicando que o processo de transformação deve começar. Este canal é declarado como urgente para que não haja atraso no início da atividade de transformação. Neste instante, a variável relógio local *t_tf* é zerada e o autômato passa para o estado *work_in_progress*. Este será obrigatoriamente deixado, devido à presença do invariante de estado, quando *t_tf* pertencer ao intervalo fechado definido pelos valores de *t_transformation_min* e *t_transformation_max*. O estado *done*, ao ser alcançado, deve ser deixado imediatamente enviando uma mensagem ao *template procedure*, através do canal de comunicação *work_done*, de modo que não haja atraso entre o fim da operação de transformação e o envio do sinal. Com nos casos

interiores, este último estado é compromissado pois não representa um estado real do componente, mas foi inserido no modelo devido à sua possível utilidade na etapa de verificação.

5.4 Procedimento para a modelagem do sistema de controle

Como mostrado no item 5.2, o sistema de controle proposto é composto por duas partes, uma referente ao processo, representada pelo *template procedure*, e a outra parte constituída pelo supervisor, representada pelo *template* de mesmo nome. Diferente dos componentes, a parte de controle deverá ser inteiramente construída pelo usuário.

5.4.1 O *template procedure*

O *template procedure* refere-se à parte de controle que dá o seqüenciamento de atividades de um processo de fabricação de um determinado produto, ou seja, representa o caminho percorrido pela peça no processo. Foi admitida a presença de *buffers* infinitos na entrada e saída de máquinas, o que permitiu oculta-los na modelagem. Este *template* se comunica tanto com o *template* supervisor quanto com os autômatos que representam os diversos componentes. A figura 5.6 ilustra a modelagem do *template procedure* proposta.

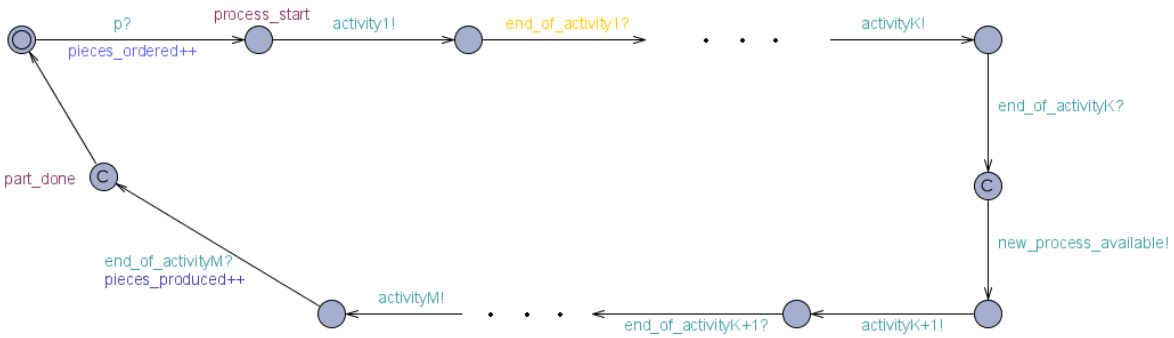


Figura 5. 6 - Ilustração do *template procedure*

O *template procedure* possui, como parâmetros, os canais de comunicação *p* e *new_process_available* e representa um processo que envolve *M* atividades executadas pelos diversos componentes. Além disso, o *template* possui uma variável

relógio *t_process* que pode ser usada na verificação. O estado inicial representa que o processo ainda não começou. O processo começa quando, através do canal de comunicação *p*, o *template* recebe um sinal do supervisor indicando que o processo de uma nova peça pode ser iniciado. Deste modo o canal é declarado como urgente e assim que a condição de início de processo for satisfeita, o processo começará sem atraso. Nessa mesma transição, a variável global *pieces_ordered* é incrementada, indicando o número de peças solicitadas. O estado seguinte é o *process_start* que representa o começo do processo. A partir dele, o *procedure* envia um sinal, por meio do canal urgente de comunicação *activity*, mudando de estado e informando um componente (este deve estar parametrizado com o mesmo canal) que ele deve começar sua atividade. O canal de comunicação *activity* equivale a um dos canais de comunicação *station, load, unload* ou *transport,* que são os canais pelos quais os componentes recebem sinais do *procedure.* Existem casos em que os parâmetros dos componentes devem ser mudados no meio do processo (por exemplo, quando uma mesma estação realiza duas operações diferentes no mesmo ciclo). Tal mudança é efetuada pelo *template procedure* na chamada do componente. O próximo estado representa que o componente ordenado a trabalhar pelo *procedure* ainda está em atividade, este será deixado quando o componente enviar um sinal pelo canal *end_of_activity,* o qual equivale a um dos canais de comunicação *work_done, machine_loaded, machine_unloaded* ou *end_of_course,* que são os canais pelos quais os componentes emitem sinais ao *procedure*. Em seguida, o *procedure* continua a ordenar novas atividades em seqüência (uma nova atividade só começa quando a anterior termina) até chegar na atividade *K*.

Ao término desta atividade, em sincronia com o recebimento do sinal enviado através do canal *end_of_activityK*, o *procedure* envia um sinal para o supervisor, por meio do canal de comunicação *new_process_available*, informando que um novo processo pode iniciar. Geralmente a disponibilidade do primeiro recurso (componente do FMS) é suficiente para entrada da peça seguinte na linha de produção, mas existem casos onde isto não é verdade, por isso é necessário fazer uma análise do sistema para a determinação de tal condição.

Depois disso, o processo continua executando novas atividades até atingir a atividade *M*, última do processo. Ao fim desta, junto com o recebimento do sinal enviado por *end_of_activityM*, a variável *pieces_produced* é incrementada, indicando que uma nova peça foi produzida. O estado a seguir é chamado de *part_done* que é

um estado compromissado, pois foi inserido no modelo apenas a fim de ser utilizado na verificação. Por fim, o *template* volta ao estado inicial, onde fica aguardando um comando do supervisor para dar início a um novo processo.

5.4.2 O *template* supervisor

O autômato de supervisor tem função de ordenar o inicio de um novo processo no momento em que a execução deste é possível. O supervisor envia um sinal para o *template procedure* começar e recebe um sinal deste último, quando a condição para o início de um novo processo for satisfeita.

A figura 5.7 ilustra a modelagem de um supervisor, proposta pelo presente projeto, para a fabricação de peças do mesmo tipo. Portanto, os processos envolvem mesmos recursos e estes são utilizados na mesma ordem.

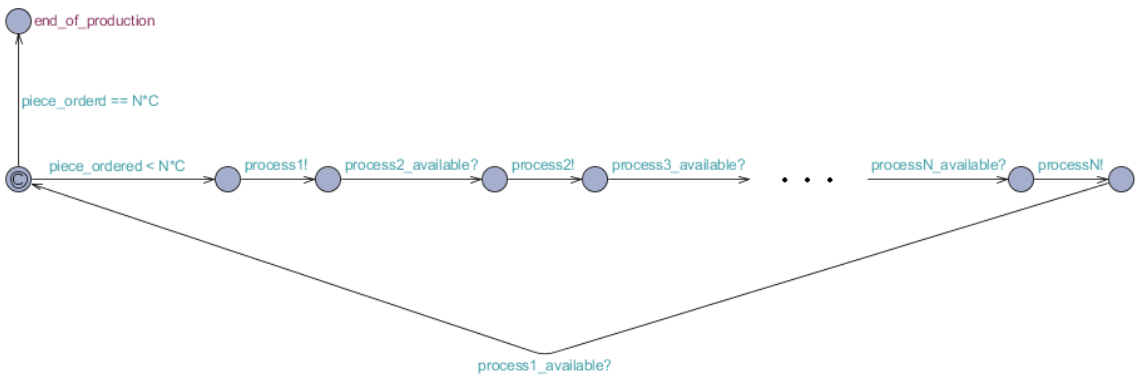


Figura 5. 7 - Ilustração do *template* supervisor

O supervisor ilustrado funciona em ciclos onde, em cada um deles, é ordenada a fabricação de um determinado número de peças, indicado por *N*. Este número representa a quantidade máxima de peças que podem estar sendo processadas simultaneamente. Para determinar um valor ótimo de *N* devem ser feitos estudos mais aprofundados sobre a dinâmica e a eficiência do sistema de produção o que envolve parâmetros, não considerado no presente trabalho, como capacidade de buffers dentre outros. O parâmetro *C*, estipulado pelo usuário, é o numero de ciclos que deverão ser executados e multiplicando este com *N* se dá o número total de peças que serão produzidas. Este último será usado para determinar quando os ciclos devem ser concluídos. A condição de fim de produção foi estabelecida no início do

ciclo apenas como exemplificação, para um sistema que produza um número de peças diferente de *N*C*, esta condição pode ser testada antes da chamada de um processo intermediário.

O ciclo pode se iniciar enquanto a quantidade de peças solicitadas é menor que o número de partes que devem ser produzidas (*pieces_ordered < N*C*). Uma vez iniciado, o autômato emite um sinal para que o *template procedure* que representa o primeiro processo do ciclo seja ativado. Este sinal é enviado pelo canal de comunicação *process1*. Em seguida, o supervisor aguarda a recepção de um sinal, enviada pelo mesmo *template procedure*, no canal *process2_available* que confirma a disponibilidade do sistema para a entrada da peça seguinte. Desta mesma forma, o autômato segue ativando novos processos até que, depois da solicitação do *processN*, e quando o *process1* estiver disponível, o supervisor retorna ao seu estado inicial, a partir do qual, o estado *end_of_production* poderá ser atingido, se o número de peças ordenados for igual ao número de peças que devem ser fabricadas (*piece_ordered = = N*C*), encerrando o pedido de fabricação. Caso a meta não tenha sido alcançada, deve ser continuada a execução de ciclos. O estado inicial é compromissado pois o teste referente ao número de peças ordenadas deve ser feito sem atraso, com prioridade sobre as outras transições do modelo. O autômato supervisor não possui parâmetros, no entanto o número total de peças fabricadas (*N*C*) deve ser colocado diretamente na modelagem. Ao término de todo este procedimento apresentado, é possível fazer a validação do modelo, para então realizar verificação do mesmo.

6 Estudo de caso

Este capítulo mostra a aplicação da proposta do presente projeto em um caso real. O caso a ser estudado é simples mas serve como ilustração. Uma vez que o presente trabalho propõe um procedimento de modelagem de FMSs, é possível realizar tal procedimento envolvendo diferentes composições de FMS.

6.1 Características do sistema

O sistema em questão produzirá *20* peças no total, sendo que o número máximo de peças simultaneamente presentes no sistema será igual a *2* (*2 templates procedure* no modelo). A planta do sistema é composta por duas esteiras transportadoras, uma de saída e outra de entrada, um robô manipulador e uma máquina flexível de manufatura. A ilustração deste sistema real é mostrada na figura 6.1. As características de funcionamento dos componentes são mostradas na definição de parâmetros na figura 6.2.

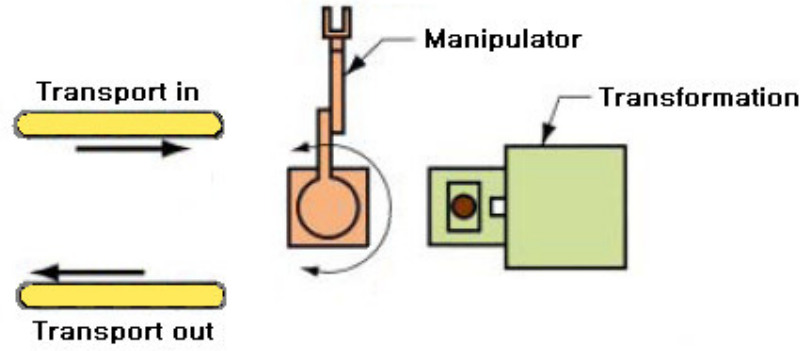


Figura 6. 1 - Ilustração do sistema

6.2 Modelo do sistema em UPPAAL

O modelo do sistema é constituído por um *template* supervisor (*supervisor*), dois *templates procedure* (*PA* e *PB*), dois *templates* componente transporte (*T1* e *T2*),

um *template* componente manipulação (*M1*) e um *template* componente transformação (*TF1*).

Os parâmetros dos *templates* são declarados na tela de *global declarations* como expõe a figura 6.2, sendo que a parte delimitada pelo quadrado vermelho indica os valores que o usuário deve definir referentes às especificações dos componentes.

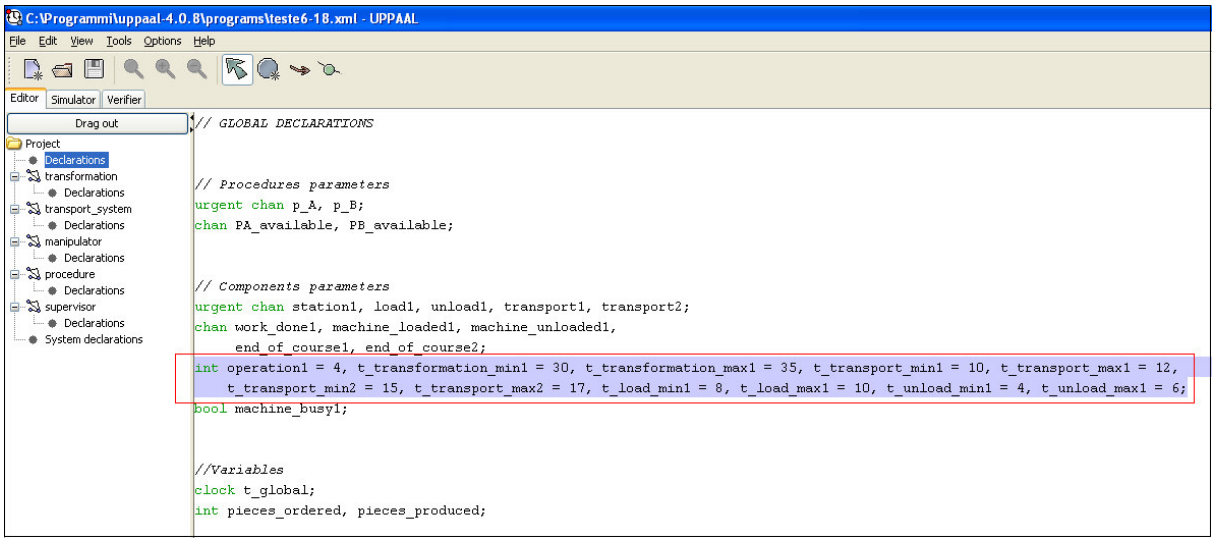


Figura 6. 2 - Declaração dos parâmetros

Nota-se nesta tela a presença de duas variáveis não mencionadas anteriormente. São elas a variável relógio *t_global*, que representa o tempo global do sistema, e a variável inteira *pieces_produced*, um contador para o número de peças produzidas.

Em seguida, os parâmetros são alocados aos seus respectivos *templates* na tela de *system declarations* como mostra a figura 6.3.

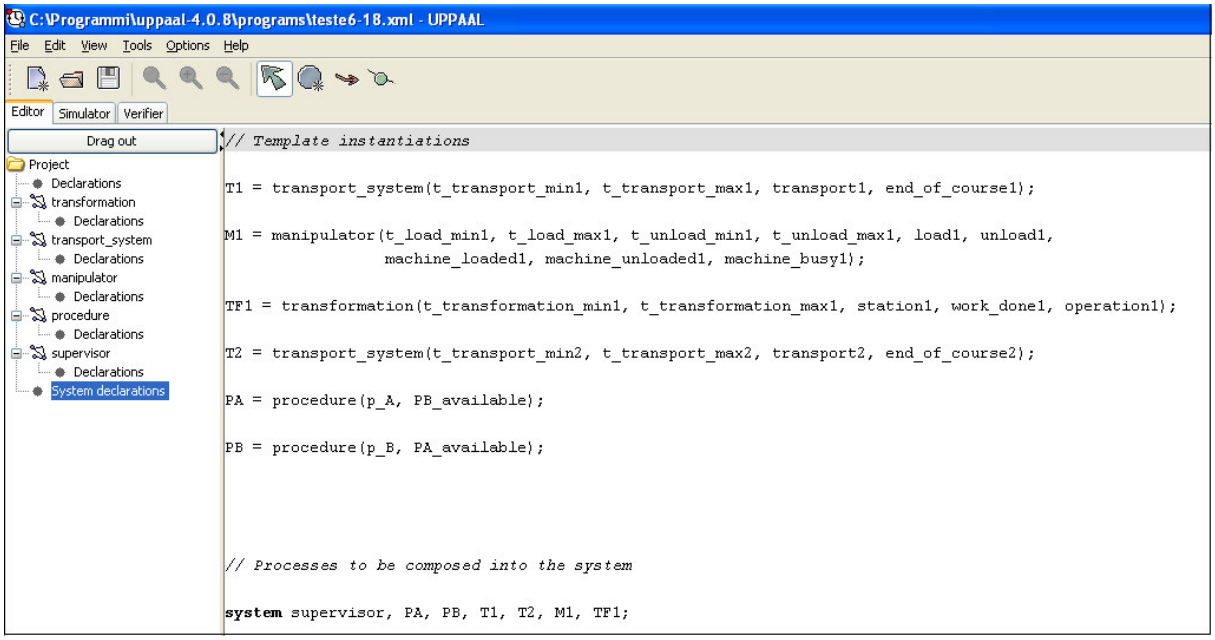


Figura 6. 3 - Alocação dos parâmetros nos *templates*

O *template* de componente transporte *T1* representa a esteira transportadora na qual as peças entram no sistema. A figura 6.4 ilustra o seu modelo, já com os seus respectivos parâmetros.

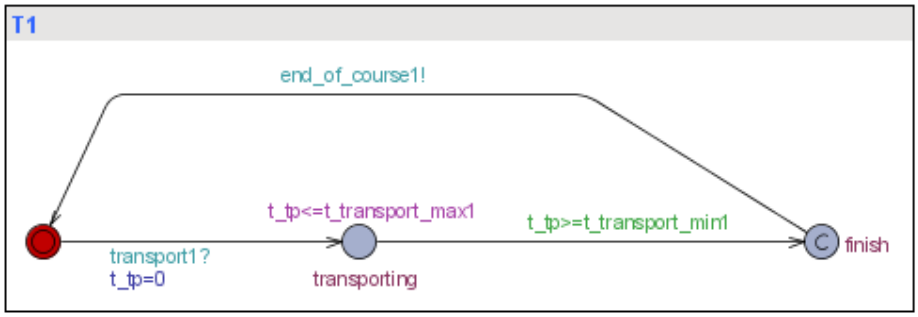


Figura 6. 4 - *Template T1*

O *template* de componente manipulação *M1* representa o braço de robô que carrega na máquina a peça que chega pela esteira transportadora de entrada e, depois que a máquina conclui a operação, descarrega-a, deixando a peça à disposição da esteira transportadora de saída, portanto, *M1* será chamado duas vezes em cada ciclo dos *templates procedure*. A figura 6.5 ilustra o seu modelo, já com os seus respectivos parâmetros. É importante citar que o parâmetro *machine_busy1* refere-se à disponibilidade da máquina representada pelo *template TF1*.

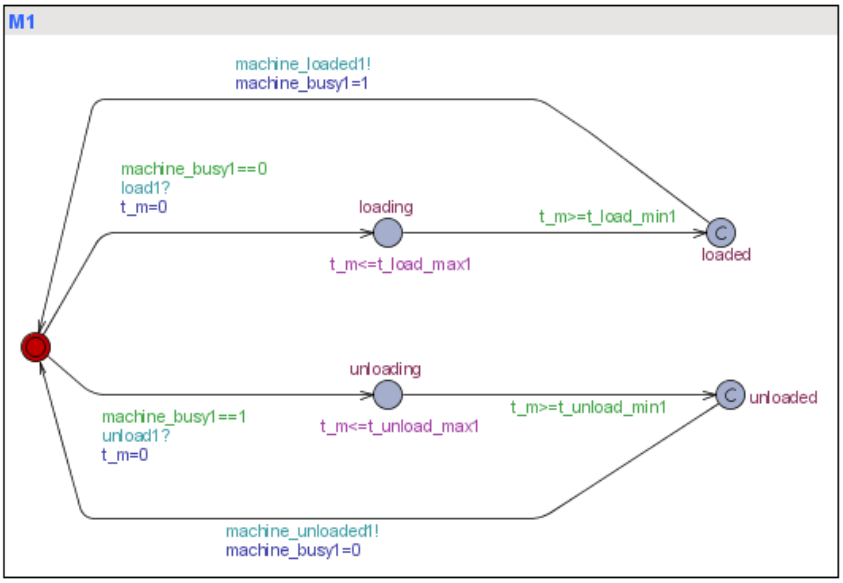


Figura 6. 5 - *Template M1*

O *template* de componente transformação *TF1* representa o módulo flexível de manufatura configurado para realizar a operação correspondente ao valor *4* da variável *operation1* (vide declaração da variável *operation1* na figura 6.2). A figura 6.6 ilustra o seu modelo, já com os seus respectivos parâmetros.

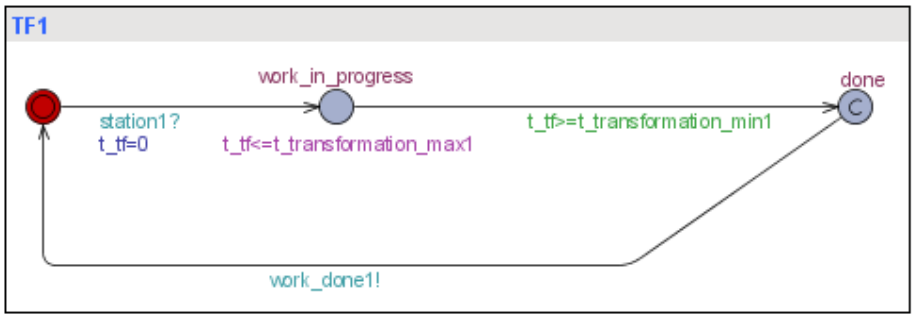


Figura 6. 6 - *Template TF1*

O *template* de componente transporte *T2* representa a esteira transportadora na qual as peças saem do sistema. Este é análogo ao *template T1*, porém com seus próprios parâmetros. A figura 6.7 ilustra o seu modelo, já com os seus respectivos parâmetros.

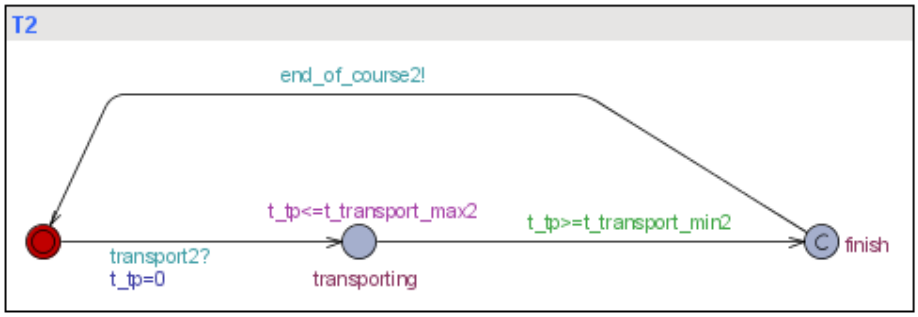


Figura 6. 7 - *Template T2*

Os *templates PA* e *PB* representam os processos executados pelo sistema. Nota-se que eles podem estar ativos simultaneamente. A figura 6.8 ilustra o momento em que existem duas peças em fase de processo, sendo que uma delas mostrada em *PA*, está na esteira transportadora de saída (estado *s11* de *PA*), e a outra, exibida em *PB*, está sendo trabalhada pela máquina flexível de manufatura (estado s7 de *PB*).

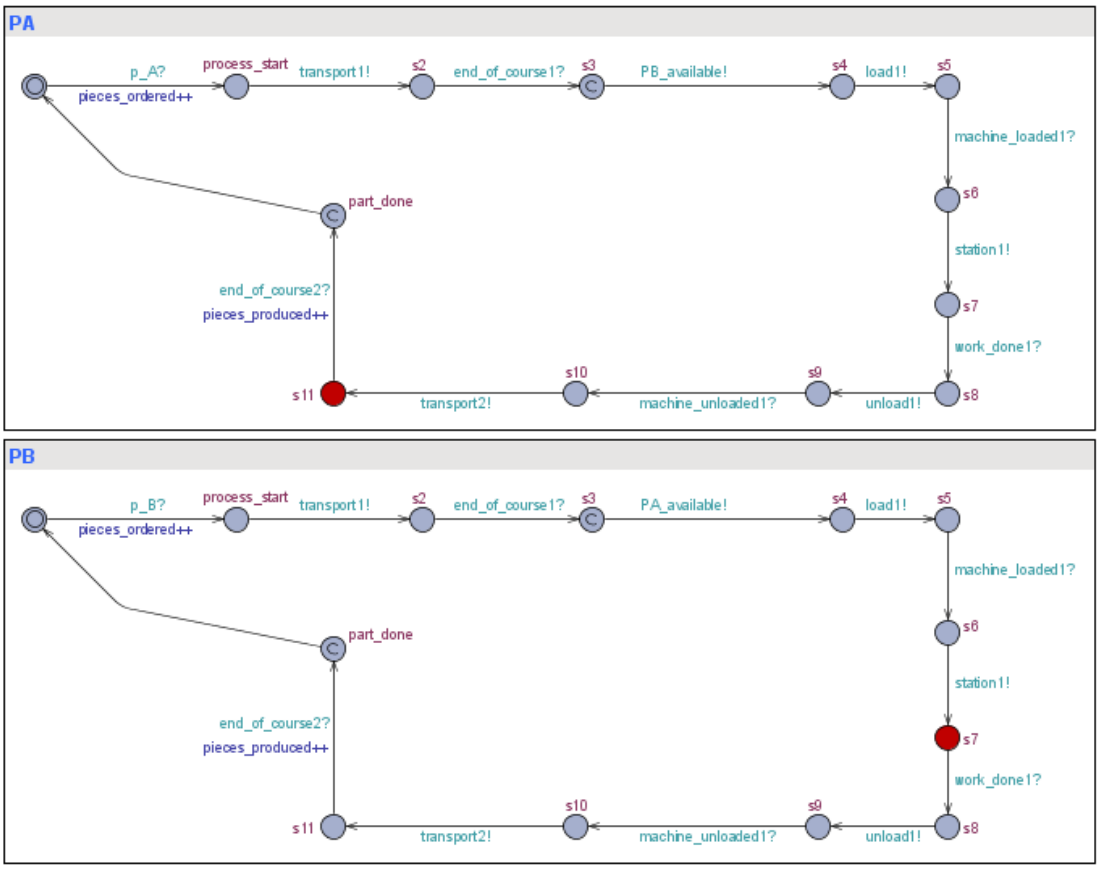


Figura 6. 8 - *Templates PA* e *PB*

O *template* supervisor ativa os processos representados por *PA* e *PB*, quando possível, além de encerrar a produção quando o número total de peças ordenadas for

igual a *20*. A figura 6.9 mostra o *template* esperando a disponibilidade de *PB* para poder ativá-lo quando este se encontrar em seu estado inicial.

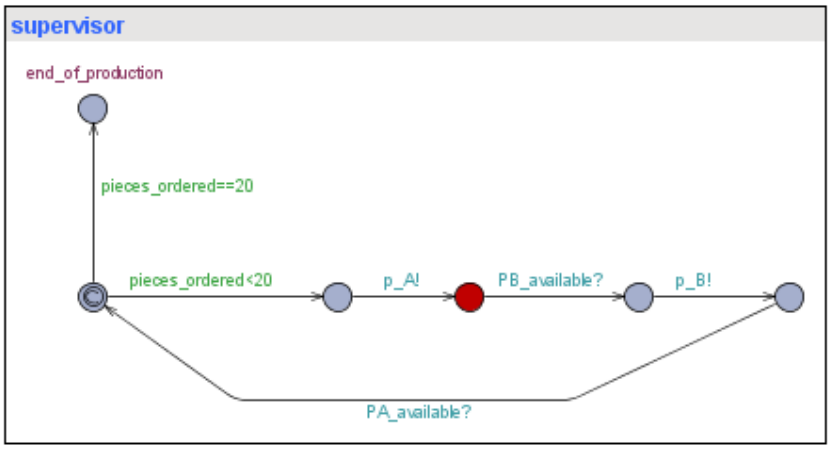


Figura 6. 9 - *Template* supervisor

A troca de informação e a comunicação entre os diversos *templates* é esquematicamente mostrada na figura 6.10.

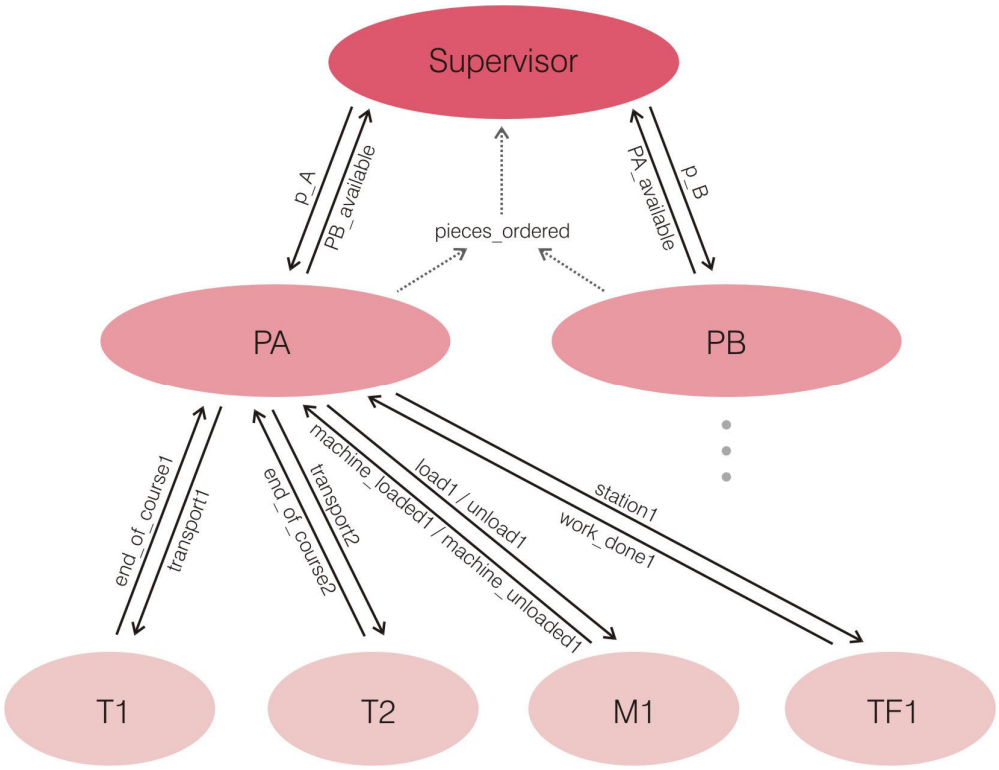


Figura 6. 10 - Troca de informação entre autômatos

As setas pontilhadas representam o compartilhamento de variáveis globais enquanto as setas cheias representam o fluxo de informação pelos canais de comunicação. As comunicações entre *PB* e os componentes foram ocultadas no esquema por questão de melhor representação gráfica, porém são análogas àquelas entre *PA* e o objeto de controle.

6.3 Verificação do modelo

Depois de simular e certificar-se que o modelo está consistente, a atividade de verificação pode ser executada. A figura 6.11 mostra a tela de verificação da ferramenta UPPAAL em que a parte delimitada pelo quadrado vermelho mostra proposições lógicas que respondem qual será o tempo mínimo de produção das *20* peças. Ao fazer a simulação do sistema, chegou-se a um valor de *t_global* igual a 867. Através da verificação, pode-se observar que existem valores de *t_global,* ao fim da produção, menores do que o encontrado na simulação. Por meio de iterações chega-se ao valor mínimo.

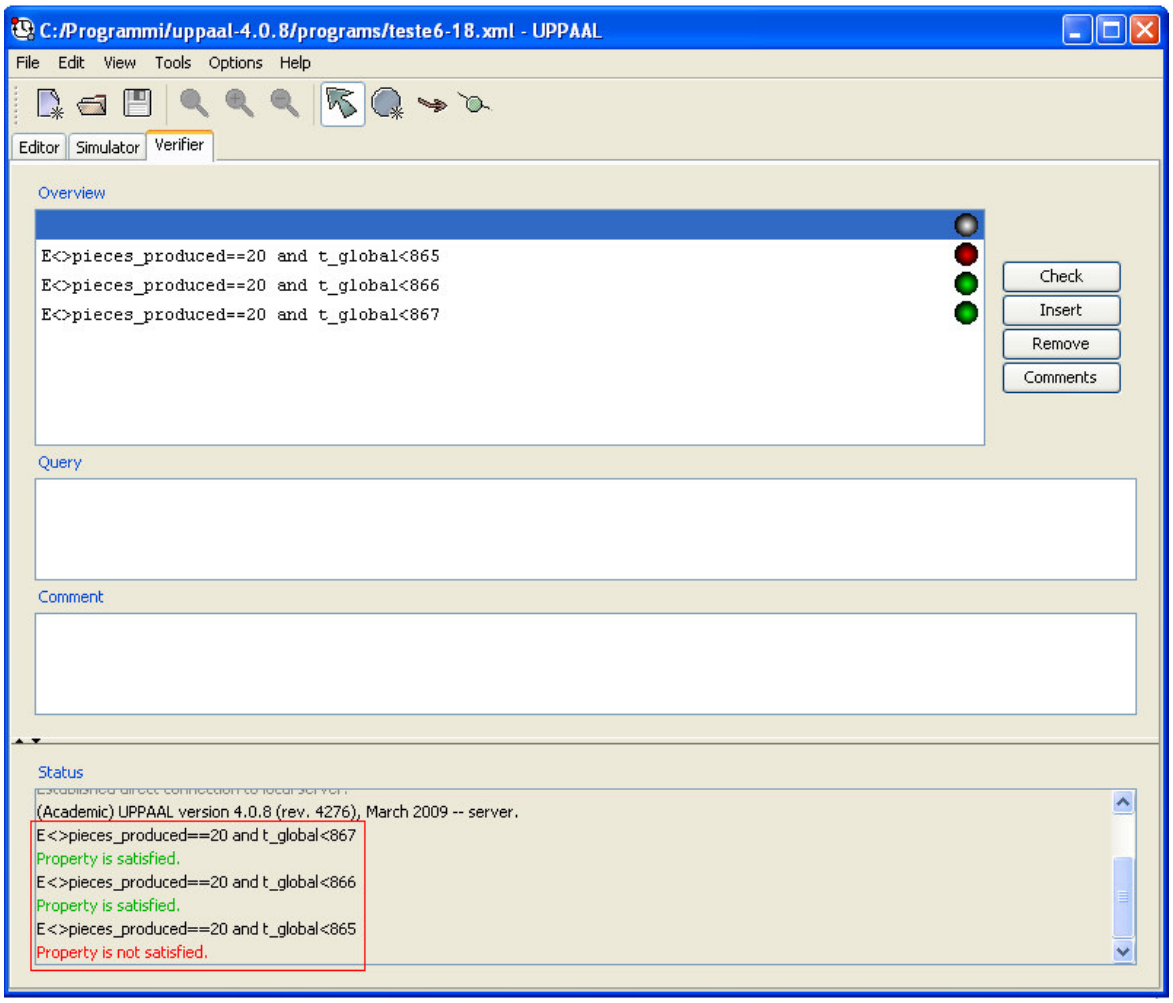


Figura 6. 11 - Exemplo de verificação

Analogamente, pode-se descobrir o *makespan* do processo, ou seja, o tempo mínimo que uma peça permanece no sistema. A figura 6.12 mostra como encontrar este valor.

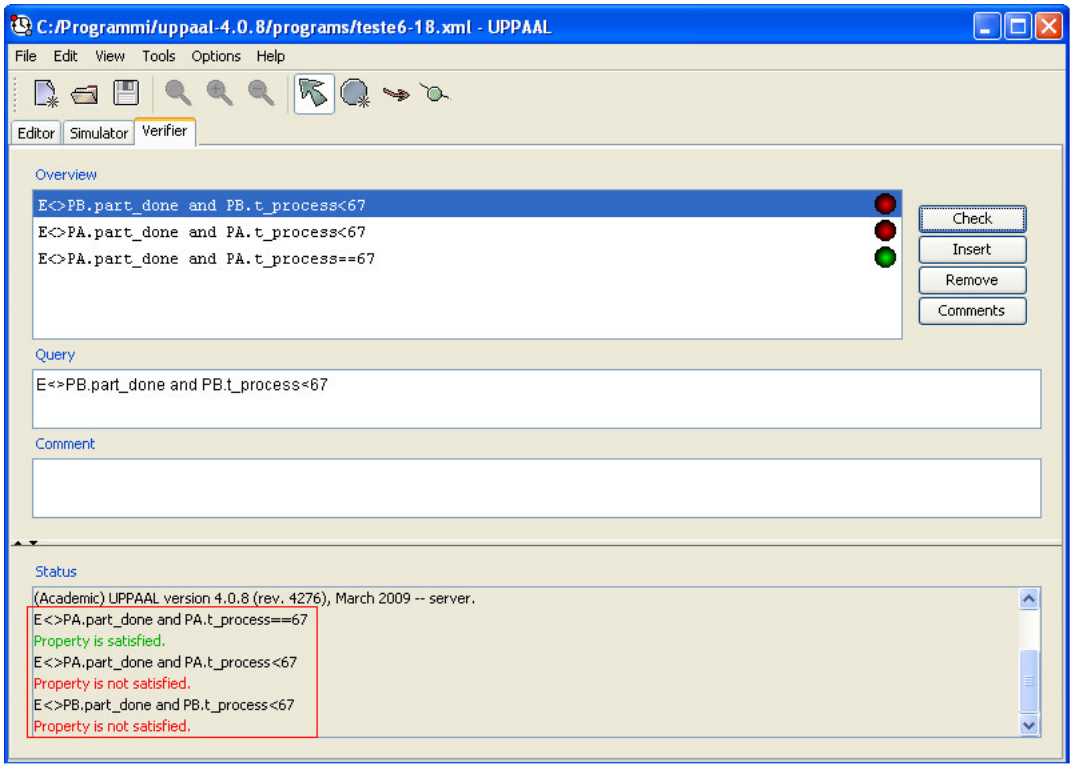


Figura 6. 12 - *Makespan* do processo

Além disso, propriedades de segurança do sistema podem ser checadas. Um exemplo disso é a verificação da situação de *deadlock* em que o manipulador segura uma peça para carregá-la em uma estação já ocupada, esta verificação é mostrada na figura 6.12 e certifica que o sistema está seguro quanto a este caso de travamento. Esta situação já era esperada devido à restrição imposta pela variável *machine_busy.*

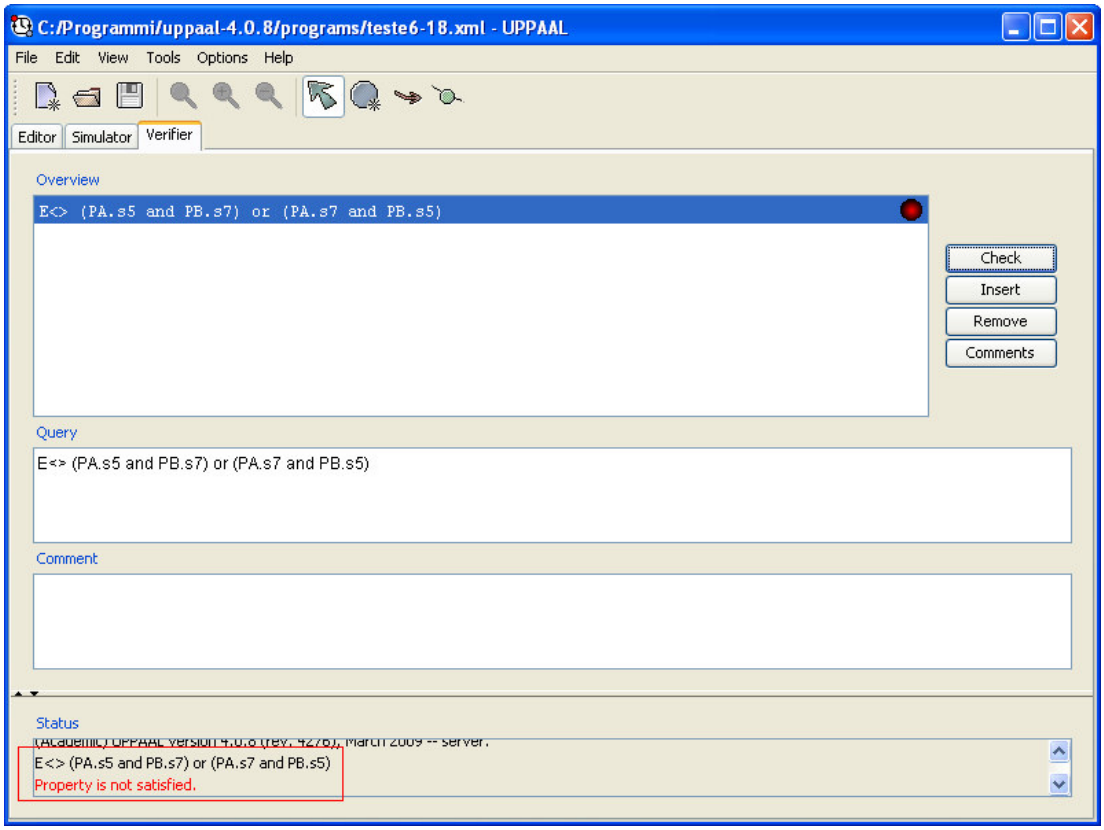


Figura 6. 13 - Verificação de *deadlock*

Vários outros testes podem ser feitos no modelo, dependendo da complexidade estrutural e funcional dos sistemas que estejam sendo modelados.

7 Conclusões

O presente trabalho teve como objetivo propor um método para realizar verificações de modelos formais de sistemas flexíveis de manufatura, por meio de modelagem de componentes do sistema e seu sistema de controle com a ferramenta UPPAAL.

Ao longo do desenvolvimento do projeto foram encontradas algumas dificuldades. Para a modelagem de componentes, a decisão das características relevantes ao projeto gerou debates. Na implementação dos autômatos de controle, a propriedade de autômatos temporizados que restringe em um, o número de estados que podem ser ocupados em cada autômato, dificultou a descrição de processos industriais onde mais de um produto estivesse sendo processado simultaneamente.

O projeto, apesar de ter alcançado os objetivos previstos inteiramente, consegue modelar somente sistemas de menor complexidade funcional naquilo que se refere à questão de processos globais que não possuem seqüenciamento de atividades pré-definido e sistemas de transporte de elevada flexibilidade implicando na definição de rotas de transporte e designação de transportadores em tempo real.

Outro aspecto importante trabalhado foi a questão de utilizar-se os recursos da ferramenta UPPAAL para simplificar a síntese de autômatos temporizados graças à possibilidade de se modelar funções de guarda e lógicas de comunicação entre diferentes autômatos.

Referências bibliográficas

ALUR, R.; DILL, D. L. Automata for Modeling Real-Time Systems. Proceedings of the 17th International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science 443, Springer-Verlag 1990.

ALUR, R.; DILL,D. L., A theory of timed automata. Theoretical Computer Science.vol. 126, no. 2, pp.183-235, 1994.

BEHRMANN, G.; BENGTSSON, J.; DAVID, A.; LARSEN, K. G. ; PETTERSSON, P. ; YI, W. UPPAAL Implementation Secrets. In FTRTFT '02: Proceedings of the 7th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems. 2002, pp. 3-22.

BEHRMANN, G. Data Structures and Algorithms for the Analysis of Real Time Systems. PhD Dissertation, Department of Computer Science, Aalborg University. November 28th, 2003.

BEHRMANN, G.; DAVID, A.; LARSEN, K. G. A Tutorial on Uppaal. Department of Computer Science, Aalborg University, Denmark, 2004.

BENGTSSON, J.; LARSSON, F. UPPAAL A tool for automatic verification of real-time systems. Master of Science thesis. 1996.

BENGTSSON, J.; YI, W. Timed Automata: Semantics, Algorithms and Tools, In Notas de aula sobre Concurrency and Petri Nets. W. Reisig and G. Rozenberg (eds.), LNCS 3098, Springer-Verlag, 2004.

BONETTO, R. Flexible manufacturing systems in practice. London : North Oxford Academic, 1987.

BURCH, C. The science of computing: first edition. Science of computing suite. 2004.

CASSANDRAS, C. G.; LAFORTUNE, S. Introduction to Discrete Event Systems, 2nd Ed., Massachusetts: Kluwer Academic Publishers, 1999.

CLARKE, E.M.; EMERSON, E.A.; SISLA, A.P. Automatic verification of finite state concurrent systems using temporal logic. Programming Languages and Systems 8(2), pp. 244--263, 1986.

CURY, J. E. R. Teoria de Controle Supervisório de Sistemas a Eventos Discretos. Notas de Aula, 2001.

DAVID, A; YI, W. Modelling and Analysis of a Commercial Field Bus Protocol. IEEE Computer Society Press, 2000.

GREENWOOD, N. R. Implementing flexible manufacturing systems. New York : Wiley, 1988.

GROOVER, M. P. Automation, production systems, and computer-integrated manufacturing. 3rd Ed. Upper Saddle River, N.J.: Prentice Hall, 2008.

HEINZINGER.; NICOLLIN, X.; SIFAKIS, J.; YOVINE. S. Symbolic Model Checking for Real-Time Systems. A preliminary version appeared in the Proceedings of the Seventh Annual Symposium on Logic in Computer Science (LICS), IEEE Computer Society Press, 1992.

HOPCROFT, J. E.; MOTWANI, R.; ULLMAN J. D. Introduction to Automata Theory, Languages, and Computation. Addison Wesley; Segunda edição, 2000.

LARSSON, F. Efficient Implementation of Model-Checkers for Networks of Timed Automata. Licentiate Thesis 2000-003, Department of Information Technology, Uppsala University, 2000

LUGGEN, W. W. Flexible manufacturing cells and systems. Englewood Cliffs, N.J.: Prentice Hall, 1991.

MIYAGI, P. E.; VILLANI, E.; VALETTE, R. Landing System Verification Based on Petri Nets and a Hybrid Approach. 2006. 17pag. IEEE University of Sao Paulo/ Instituto Tecnológico de Aeronáutica/CNRS, 2006,

NIELSEN, B. Specification and Test of Real-Time Systems. 212 p. PhD Thesis , Aalborg University. April, 2000.

PROMEC. Disponível em < http://www.promec.pt/images/rocla/Rocla_AGV_1.jpg>. Acesso em junho de 2009.

RAMADGE, P. J.; WONHAM, W. M. The Control of Discrete Event Systems, Proceedings. IEEE, Vol. 77, n. 1, pp. 81-98, 1989.

RANKY, P. G. The design and operation of FMS : flexible manufacturing systems. Bedford : IFS (Publications) Ltd, and U.K. and North Holland Publishing Company, 1983.

ROBOT MAGAZINE. Issue 2. Disponível em <http://www.botmag.com/issue2/images/bottom2.jpg>. Acesso em junho de 2009.

S & S PRECISION. Disponível em <http://www.sandsprecisionnj.com/image/32216881.JPG>. Acesso em junho de 2009.

SARMENTO, C. A.Modelagem de Programas e sua Verificação para Controladores Programáveis. Tese de mestrado, Escola Politécnica da Universidade de São Paulo, 2008.

ZHOU, M.; VENKATESH, K. Modelling, simulation and control of flexible manufacturing systems: A petri net approach. Singapore: World Scientific, 1999.