Comment choisir son outil d'analyse de systèmes à événements discrets ?

Pascal André[1], Aurélien Pageot[2], Orlann Cailleau[2], and Charbel Bedran[2]

[1] LS2N CNRS UMR 6004 - Nantes Université- pascal.andre@ls2n.fr
[2] Master ALMA - LS2N - Nantes Université- {firstname.lastname}@etu.univ-nantes.fr

Résumé

Lorsqu'on veut étudier la dynamique d'un système de manière méthodique, la première étape est de choisir le formalisme et l'outil qui permettront de modéliser le système, vérifier ses propriétés et/ou le simuler et examinant les traces d'exécution. Ce papier contribue à répondre à cette question en proposant une grille d'évaluation des outils sur quatre thèmes. Le comparatif réalisé sur quelques études de cas, ne se veut pas exhaustif mais représentatif des approches pour illustrer la démarche. Il pourra être enrichi collaborativement à la fois sur les outils et les études de cas.

1 Introduction

En ingénierie des systèmes, on s'appuie sur des modèles pour comprendre et mettre en œuvre des systèmes complexes (concurrents, distribués, temps-réels, embarqués). On s'intéresse ici, en particulier, à des systèmes de contrôle (contrôle/commande comme des feux de circulation, processus de production, domotique...) sous l'angle particulier des systèmes à événements discrets (SED). La question essentielle devient alors quelle approche/outil faut-il utiliser pour étudier (modéliser, analyser, tester) le système cible ? C'est celle que nous nous posons tous face au problème. Pour y répondre, il est nécessaire alors de (i) faire un état de l'art des formalismes et outils permettant de modéliser efficacement des modèles de systèmes dynamiques et pour le valider, (ii) de créer un *benchmark* comparatif des outils et formalismes pour identifier chaque avantages et inconvénients de chacun en fonction des besoins d'ingénierie.

Nous contribuons modestement à ce travail de synthèse en posant une première brique. Ce papier traite d'outils permettant de traiter des systèmes de contrôle à événements discrets tels que des simulateurs, des Réseaux de Pétri (RDP), des automates ou des algèbres de processus, en mettant en lumière leurs caractéristiques. À travers des études de cas concrètes, nous évaluerons la capacité de ces outils à simuler des systèmes de contrôle et à vérifier leurs propriétés. L'objectif est de fournir une comparaison étayée des outils testés sur leur expressivité, la vérification de propriétés et la simulation. Ayant une vision empirique, nous n'avons pas la prétention d'être exhaustifs ni en largeur sur la variété des outils ni en profondeurs sur la finesse des caractéristiques prises en compte. Par exemple, le temps peut s'interpréter différemment d'une approche à l'autre. De même, le *benchmark* s'appuie sur trois cas, il n'est pas suffisamment représentatif. Notre idée est que ce travail soit enrichi par d'autres contributeurs.

L'article est organisé comme suit. Nous commençons par introduire les éléments de la méthodologie dans la section 2. Nous détaillons les études de cas support d'expérimentation (section 2.1), les formalismes et outils (section 2.2), la vérification de propriétés (section 2.3), et les outils sélectionnés (section 2.4) pour les expérimentations. Les expérimentations sont décrites dans la section 3 : l'application des outils au cas (section C) et les résultats du *benchmark*(section 4). Nous discutons de ces résultats en section 5 avant de conclure. Ces travaux étant empiriques, nous proposons une annexe en ligne [1] pour plus de détails.

1. https://uncloud.univ-nantes.fr/index.php/s/KjgZJx2BeXwXNZq

2 Méthodologie

La méthodologie utilisée pour construire un benchmark comparatif est la suivante : (i) définition des études de cas (exemples de systèmes de contrôle) (ii) sélection des approches et outils candidats (critères de sélection), (iii) sélection des caractéristiques retenues pour chaque approche (critères de comparaison), (iv) application aux études de cas. Les études de cas décrivent l'espace du besoin tandis que les approches décrivent l'espace de la solution. Nous aurions pu choisir une approche plus rationnelle visant à définir les caractéristiques du besoin et les caractéristiques des solutions puis définir une relation entre les deux. Cela semble bien complexe et au delà des motivations initiales.

2.1 Etudes de cas

Les outils ont été testés sur trois cas d'études avec des besoins de modélisation, vérification et simulation variables qui illustrent les besoins que l'utilisateur (ingénieur) peut avoir. (1) Le premier cas est le contrôle des feux de circulation à un carrefour avec des propriétés de sûreté de fonctionnement et de vivacité (*safety* & *liveness*). (2) Un cas de processus de production simplifié organisé en Kanban. (3) Le troisième cas d'étude consiste en une boucle de convoyeurs qui bloquent des palettes à chaque poste pour un traitement ou pas des produits dans un système de production fictif (performance). Le premier cas est typique d'une organisation par flot de contrôle tandis que les deux autres sont de type flots de données.

Nous aurions pu prendre des cas classiques de contrôle distribué comme le 'producteur/consommateur', rendez-vous, 'lecteurs/écrivain'. Nous avons souhaité des cas concrets, et les feux tricolores ou le cas Kanban sont simples et compréhensibles par tous. Nous disposons d'une base de modèles (automates synchronisé, RDP, UML...) pour nos enseignements qui ont permis de nous focaliser sur les outils et non les modèles. Ces deux cas sont des cas simplifié d'automate qui possèdes très peu de places/états ainsi que des arcs/transitions. Concernant le cas n°3, c'est un cas pour lequel nous disposons d'un modèle FlexSim réalisé par un collègue spécialiste, qui nous permet de comparer avec les autres approches. Pour des raisons de place, ces cas et leur motivation sont détaillés dans la Section A de l'annexe web[1]. Nous n'avons pas considéré ici la vérification de logiciels ou d'architecture matérielles, domaines d'application d'une partie des outils de vérification formelle mentionnées dans la section suivante.

2.2 Approches SED et outils candidats

Deux approches semblent pertinentes, (1) les outils de vérification formelle et (2) les simulateurs à événements discrets. Des détails sont apportés en Section B de l'annexe [1].

Les outils de **vérification formelle** de formalismes classiques comme les automates (ou systèmes de transitions) [BBL+99], réseaux de Petri [GV03] et algèbres de processus [CRS18]. Ces formalismes sont très abstraits, vis-à-vis des concepts des cas d'étude (contrairement à l'autre catégorie) mais bénéficient de nombreux travaux théoriques et des outils Les réseaux de Petri (RDP) et certains types d'automates sont visuels et donc plus conviviaux. Les algébres de processus sont plus algorithmiques et familières pour les informaticiens. Il existe de nombreuses variantes (et théories sous-jacents) *e.g.* avec ou pas de types de données, de temps, des gardes, d'actions, de non-déterminsimes, de probabilités... L'expressivité du langage de modélisation en dépend. Bérard et al. [BBL+99] proposent un éventail de langages et outils pour la modélisation et la vérification par *model-checking* de la dynamique des systèmes réactifs tels que SMV — Symbolic Model Checking, SPIN — Communicating Automata, DESIGN/CPN — Coloured

Petri Nets, UPPAAL — Timed Systems, KRONOS — Model Checking of Real-time Systems, HYTECH — Linear Hybrid Systems. On trouve aussi nuSMV ou DEVS.

Les outils de **simulation de flux** comme AnyLogic ou FlexSim [Nor03] sont pratiques dans le domaine de la modélisation de processus de production industrielle, mais sont aussi lourds à appréhender. La création d'un nouveau modèle de chaîne de production demande beaucoup de ressources tant pour la structure détaillée que la description du comportement dynamique. Ces outils sont puissants et permettent de définir des scénarios de simulation, les étalonner et comparer les performance est aussi chronophage et nécessite de l'expérience, qu'on ne trouve pas forcément sur les lieux de production notamment les PME-PMI. Un inconvénient crucial est le manque d'outils de vérification formelle. Enfin, et ce n'est pas négligeable, ils ne s'adaptent pas simplement à d'autres contextes comme le contrôle de processus réels-réel ou de systèmes embarqués. *De ce fait nous ne les avons pas inclus à ce stade dans le comparatif.*

2.3 Analyse des Propriétés

Les modèles dont nous traitons ici sont des spécifications formelles [GMBS96] dont l'intérêt est de pouvoir démontrer formellement des propriétés. Trois approches sont possibles : l'analyse statique qui consiste à explorer le modèle (structure et caractéristiques, types et espace d'état...), la preuve par théorèmes (il faut une structure logique sous-jascente au modèle) et l'analyse dynamique qui explore ses comportements dans le temps (évolution dans le temps, analyse de traces par exemple, simulation sous certaines hypothèses). Le *model checking* [BBL+99] est ainsi une technique de preuve formelle automatique [2] qui explore un espace d'état pour démontrer une propriété donnée. Elle est adaptée à l'étude (des propriétés) de la dynamique des système.

Plus le formalisme est expressif, plus il est difficile à vérifier formellement et statiquement des propriétés, on se tourne alors vers le test et la simulation, qui fonctionnent sous certaines conditions. Noter, et c'est fondamental, qu'on doit spécifier formellement on seulement le modèle du système mais aussi les propriétés attendues et il y a des langages pour cela, notamment les logiques temporelles [BBL+99] qu'on utilise pour explorer les traces d'exécution. Nous détaillons notre approche de la vérification dans la Section B.5 de l'annexe web[1].

2.4 Approches et outils sélectionnés

Pour sélectionner les outils du comparatif, les critères suivants sont retenus : (i) outils gratuit disponible, (ii) notoriété, (iii) ergonomie, (iv) représentativité (couverture), (v) modularité des spécifications, (vi) techniques de vérification, (vii) documentation... Nous avons commencé par des outils conviviaux comme Tina ou Romeo avant de nous attaquer aux autres. Nous avons sélection au moins un outil dans chaque sous-catégorie RdP (Romeo [GLMR05], Tina [BRV04], CPN Tools [RWL+03]), Automate (UPPAAL [BLL+96]), Algèbres de processus (SPIN [Hol04]). Nous avons aussi fait des tests avec Renew, Imitator, Itemis et JFLAP. Nous détaillons les outils dans la Section B.6 de l'annexe web[1].

3 Expérimentations

Nous avons expérimenté les études de cas avec différents outils de modélisation et de vérification dédiés à la vérification formelle de la dynamique des systèmes. Entre autres, nous avons

2. La preuve est aussi partiellement automatisée par des *theorem provers*, mais l'humain doit généralement intervenir dans la construction de la preuve.

expérimenté avec des outils de réseaux de Petri (Tina, Romeo, CPN tools), des automates communicants (UPPAAL), ou des processus (SPIN). Pour des raisons de places nous nous illustrons nos expériences sur le cas d'étude n°3 avec Tina pour les RdP et UPPAAL pour les automates. Tina a été choisi car il met en oeuvre simplement la vérification et la simulation avec horloge centralisée. Le reste est disponible dans la Section C de l'annexe web[1]. Dans ce qui suit, nous illustrons nos expériences avec le cas n°3 et un outil de chaque catégorie.

3.1 Cas d'étude A.3 avec un RDP sur Tina 3.4.4

Le cas est décrit plus précisément dans la Section A.3 de l'annexe web[1]. Une boucle de plusieurs convoyeurs alimente chaque poste. Le poste 1 place deux produits sur la palette. Le poste 2 (resp. 3) en prend 1 et le replace ailleurs sur le convoyeur. Une fois les deux produits déposés, la palette avance au poste 4 qui les stocke.

Le RDP temporisé de la Figure 1 modélise chaque élément décrit dans le cas d'étude. Bien que le système soit représenté comme un réseau unique, nous l'avons conçu par parties que nous avons ensuite assemblé par fusion/ajout de places ou de transitions [3]. Sur la Figure 1 nous avons ajouté des commentaires graphiques pour identifier les parties. On y voit plus distinctement les 4 postes ainsi que les 5 convoyeurs qui forment le convoyeur circulaire [4].

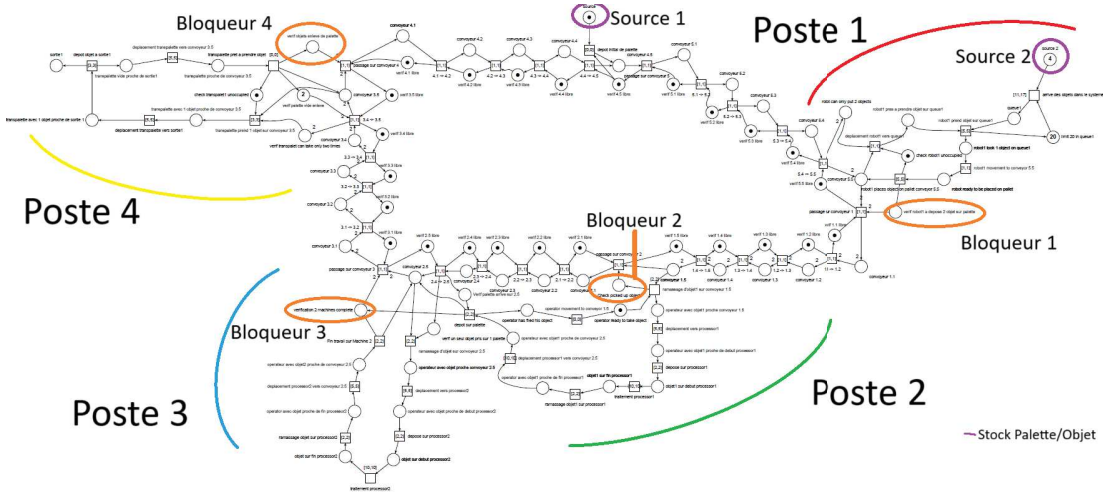

FIGURE 1 – Troisième cas d'étude modélisé sur Tina

Chaque convoyeur est composé est composé de 5 places pour 5 emplacements de palettes, chaque place d'un convoyeur dispose d'une place de garde supplémentaire permettant de limiter le passage des palettes. Cette garde empêche les palettes de se superposer dans le système en créant une file de palettes. Le temps est ajouté à chaque transition indiquant la durée de chaque

3. C'est une technique de base commune aux RdP. D'autres outils permettent de représenter différemment cet assemblage : par exemple on utilise des variables partagées de marquage dans Romeo (*cf.* Section C.9 de l'annexe web[1]) alors que dans CPN Tools (*cf.* Section C.6 de l'annexe web[1]) le réseau est hiérarchiquecomme un arbre et on peut en développer les branches. Ainsi, une "substitution transition" est une transition définie par un sous-réseau (subpage) on a aussi des "fusion sets" pour éviter la multitude d'arcs dans un graphe planaire.

4. Dans un système industriel, une boucle de convoyage automatisée n'est pas faite d'un seul tenant, elle est modulaire et donc extensible ; des connecteurs permettent de brancher les sections pour faire circuler les fluides (électricité, huile, gaz...).

étape. Tina n'utilisant que le formalisme des jetons simples le flux des palettes sans objets se déroule avec un seul jeton et une fois que deux objets sont sur les palettes ce sont deux jetons qui se déplacent simultanément de place en place. Chaque poste nécessite une garde pour simuler un bloqueur et permettre le blocage des palettes. (i) Au premier bloqueur, la palette attend que deux objets soient déposés dessus avant de pouvoir passer au convoyeur suivant. (ii) Au deuxième bloqueur, l'opérateur 1 prend un objet, puis la palette est libérée (iii) Au troisième bloqueur, l'opérateur 2 prend un objet, puis la palette attend que les objets traités par les opérateurs 1 et 2 soient replacés sur la palette. (iv) Au quatrième bloqueur, la palette est vidée grâce au transporteur, puis elle retourne dans le cycle. Le RDP dispose également de deux emplacements supplémentaires : l'un pour choisir le nombre de palettes dans le système (source1), et l'autre pour choisir le nombre d'objets à traiter (source2).

Une fois le système modélisé avec Tina, il est possible d'en vérifier des propriétés grâce au outils (structural analysis, reacheability analysis, checker). Par exemple, la Figure 2 montre l'analyse de la structure d'un RDP par Tina avec le nombre de places, transitions, arcs ainsi pour chaque transitions du modèle est indiqué ces places d'entrées et de sorties associés.

```
Struct version 3.8.5 -- 12/01/24 -- LAAS/CNRS

parsed net rodic5place_V3

87 places, 51 transitions, 182 arcs

net rodic5place_V3
tr {1.1 -> 1.2} {convoyeur 1.1}*2 {verif 1.2 libre}  -> {convoyeur 1.2}*2 {verif 1.1 libre}
tr {1.2 -> 1.3} {convoyeur 1.2}*2 {verif 1.3 libre}  -> {convoyeur 1.3}*2 {verif 1.2 libre}
tr {1.3 -> 1.4} {convoyeur 1.3}*2 {verif 1.4 libre}  -> {convoyeur 1.4}*2 {verif 1.3 libre}
tr {1.4 -> 1.5} {convoyeur 1.4}*2 {verif 1.5 libre}  -> {convoyeur 1.5}*2 {verif 1.4 libre}
```

FIGURE 2 – Analyse de la structure du RDP du troisième cas d'étude modélisé sur Tina

La figure 3 indique quant à elle un des résultat d'une vérification des état du RDP. Cette vérification diffère en fonction de beaucoup de paramètres. Ici, il est possible d'identifier plusieurs propriétés comme la vivacité. Lorsqu'on calcule le graphe de marquage, on dispose explicitement de l'espace d'état sous forme d'un automate, on obtient ici 299 états sur cette versions (et 466 sur la version dont le graphe est représenté en Figure 14 de l'annexe web[1]).

places	87	transitions	51	net	bounded	Y	live	?	reversible	?

abstraction	count	props	psets	dead	live
states	299	87	154	?	?
transitions	379	51	51	?	?

FIGURE 3 – Analyse de l'espace d'état (graphe de marquage) du cas d'étude avec Tina

Concernant la simulation (*cf.* Figure 4), l'intérêt de Tina par rapport aux autres outils testés est qu'il conserve une horloge globale de simulation pour le temps. Il permet de plus de simuler pas à pas ou globalement. En tenant compte d'une règle aléatoire pour les intervalles de temps, on obtient un temps de simulation, ici on obtient 342 avec 3 palettes et 10 produits.

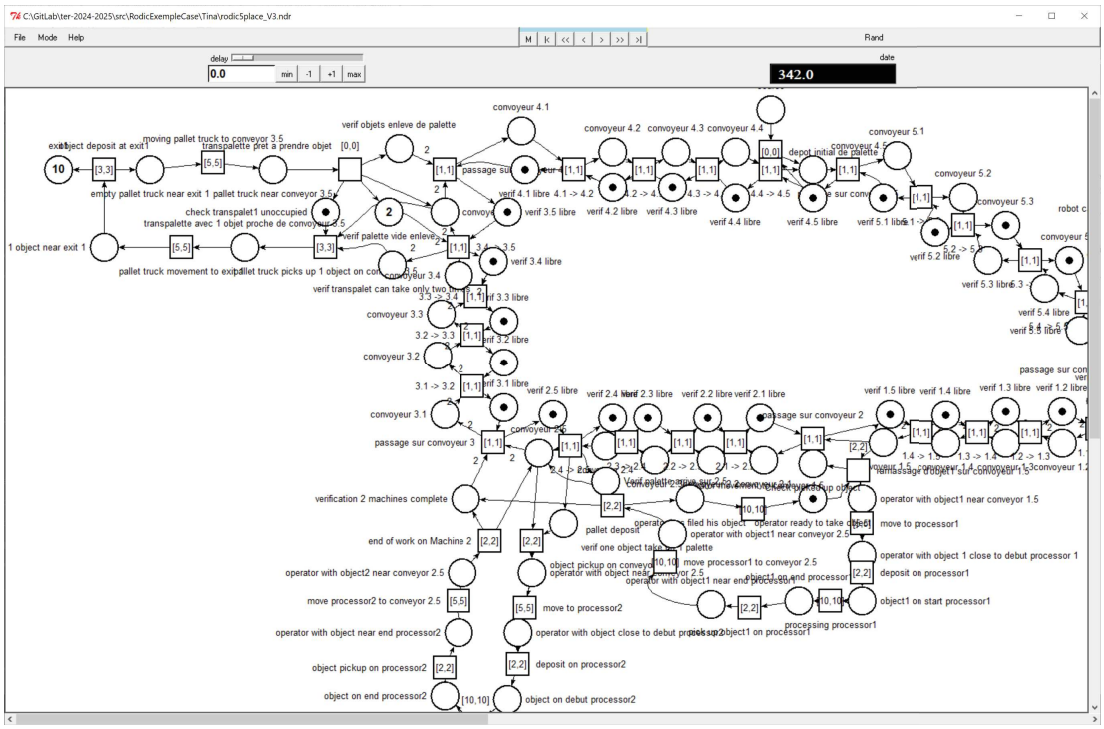

FIGURE 4 – Cas d'étude simplifié traité avec Tina

3.2 Cas d'étude A.3 avec un automate sur UPPAAL

Par nature, les automates (ou *finite state machines*) suivent une approche modulaire et distribuée. Deux modes d'interactions sont courants [5] : (i) *implicite* par variables partagées. Les automates n'échangent pas directement, c'est leur contexte qui varie. On retrouve les problèmes classiques de condition de verrouillage d'accès en lecture/écriture pour conserver la cohérence des valeurs. L'horloge globale dans une simulation ou une exécution peut-être considérée ainsi. (ii) *explicite*, les automates interagissent par synchronisation seule ou par communication (asynchrone par envoi de message dans une boite aux lettres, synchrone par des canaux de communication). Dans les deux cas, l'interaction peut être pair-à-pair ou généralisée (*broadcasting* pour les messages). Par exemple, MEC [Arn90] est un *model-checker* pour automate synchronisés sur les étiquettes de transitions par des vecteurs de synchronisation et le produit synchronisé des automates. Dans les Statecharts, les messages son envoyés en synchrone et les événements en asynchrone. En UPPAAL, deux automates communiquent (et se synchronisent) par des messages sur un canal. Les variables globales sont partagées.

Comme illustré par la Figure 5, le système est structuré en 15 automates : 4 forment le convoyeur et 5 bloquent les palettes entre le convoyeur et le poste de travail permettant d'attendre que le poste est fini sont action avant de laisser la palette avancé. Les 4 derniers sont dédiés aux postes de travail : (i) Poste 1 : Un robot charge deux objets un par un sur la palette

5. Noter que les approches par processus communicants *e.g.* avec SPIN ou CADP fonctionnent sur les mêmes principes.

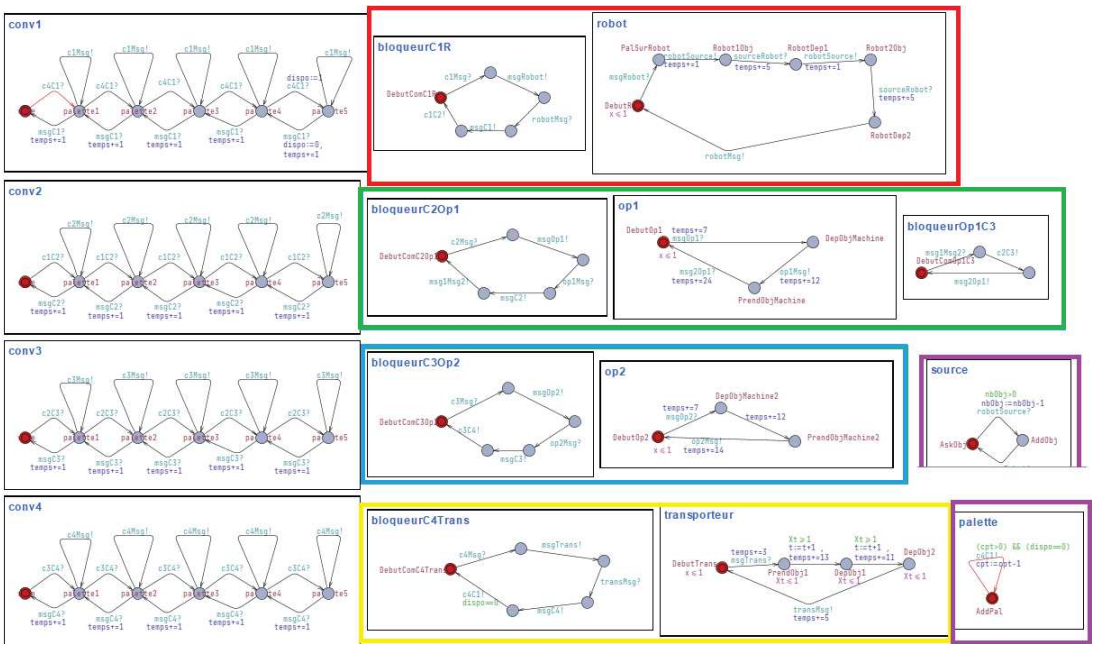

FIGURE 5 – Troisième cas d'étude modélisé sur UPPAAL

bloquer en attendent les objets (encadré en rouge). (ii) Poste 2 : Un opérateur récupère un objet d'un premier bloqueur, le dépose dans une machine pour traitement, puis le replace sur la palette bloquer au seconde bloqueur (encadré vert). (iii) Poste 3 : Un opérateur récupère un objet du bloqueur, le dépose dans une machine pour traitement, puis le replace sur la palette (encadré vert). (iv) Poste 4 : Un transporteur récupère les objets de la palette bloquer pour les déposer dans un dépôt (encadré en jaune). Enfin, les deux derniers automates permettent de sélectionner le nombre de palettes et d'objets à faire circuler dans la chaîne de production, assurant ainsi l'arrêt de l'automate lorsqu'il n'y a plus d'objets à traiter (encadré en violet).

Une fois modélisé, on vérifier des propriétés du système comme la sûreté et la vivacité dans l'onglet vérification d'UPPAAL. On peut observer sur la Figure 28 de l'annexe web[1]) que les propriétés vérifié sont signalées en vert et en rouge celles dont la vérification a échoué. Pour la simulation, nous avons obtenu la trace d'exécution de la Section C.12 de l'annexe web[1]).

4 Synthèse des résultats préliminaires

Pour évaluer les différents outils de modélisation et de vérification formelle, un ensemble de critères de comparaison a été défini et regroupé en quatre grandes parties : 1. *généralités*, 2. *expressivité du langage*, 3. *potentiel de vérification*, 4. *potentiel de simulation*. Notre proposition de *benchmark* est détaillée dans les sections suivantes. Nous utilisons des codes pour visualiser le résultat : — **vert** (caractéristique présente, facilité d'accès), — **orange** (fonctionnalité réduite ou accès délicat), — **rouge** (caractéristique absente, difficulté d'accès).

4.1 Généralités

Cette partie s'intéresse au caractéristiques générales des outils et leur *accessibilité*. Nous vérifions les accès, droits et licences. L'*interaction homme-machine* est ensuite évaluée en terme d'ergonomie et de convivialité. L'aide en ligne et la documentation sont indispensables pour mener une analyse SED. La capacité à ajouter des *plugins ou extensions* est également prise en compte pour étendre les fonctionnalités de base. L'analyse inclut la possibilité d'*importer* ou *d'exporter* des modèles pour les réutiliser ou les exploiter dans d'autres outils. Enfin, la *génération automatique de code* est examinée pour voir si l'outil peut produire des programmes exécutables directement à partir des modèles. La Table 1 compare l'échantillon d'outils de modélisation et vérification formelles selon plusieurs critères. Les outils sont classés par formalisme. Nous avons retenus les outils gratuits téléchargeables. Il se distinguent par leur interface homme-

	Composants/Environnements				
	Téléchargement	**Ergonomie**	**Extensions**	**Imports/Exports**	**Génération de code**
RDP					
Tina	Facile, Gratuit, sur leur site web	Facile, Éditeur graphique et éditeur textuel	Moyen, Vous pouvez ajouter votre code de vérification de modèle	Facile, Les modèles peuvent être importés et exportés	Non, Vous ne pouvez pas générer de code à partir de cet outil
Romeo	Facile, Gratuit, sur leur site web	Facile, Éditeur graphique	Non, Vous ne pouvez pas ajouter d'extensions à l'outil	Facile, Les modèles peuvent être importés et exportés	Non, Vous ne pouvez pas générer de code à partir de cet outil
CPN Tools	Facile, Gratuit, sur leur site web	Facile, Éditeur graphique	Non, Vous ne pouvez pas ajouter d'extensions à l'outil	Facile, Les modèles peuvent être importés et exportés	Non, Vous ne pouvez pas générer de code à partir de cet outil
Renew (Reference Net Workshop)	Facile, Gratuit, sur leur site web	Moyen, Éditeur graphique et textuel	Oui, Supporte les extensions	Facile, Les modèles peuvent être importés et exportés	Non, Pas de génération de code
Processus					
SPIN promela	Facile, Gratuit, sur leur site web	Non, il ne fournit pas de support spécifique pour les interfaces utilisateur interactives ou l'interaction directe homme-machine	Moyen, SPIN supporte les extensions via des scripts personnalisés ou des outils supplémentaires, mais ce n'est pas aussi flexible	Moyen, en utilisant des formats spécifiques, mais ce n'est pas aussi flexible que certains autres outils en termes de capacités générales d'import/export	Non, il génère du code C pour la vérification, mais pas pour la génération de code général ou au niveau de l'application
Automate					
UPPAAL	Facile, Gratuit, sur leur site web	Moyen, L'éditeur graphique est facile mais l'éditeur textuel est complexe à comprendre	Non, Vous ne pouvez pas ajouter d'extensions à l'outil	Facile, Les modèles peuvent être importés et exportés	Non, Vous ne pouvez pas générer de code à partir de cet outil
Imitator	Moyen, Nécessite Linux pour l'installation	Facile, Éditeur graphique	Non, Vous ne pouvez pas ajouter d'extensions à l'outil	Facile, Les modèles peuvent être importés et exportés	Non, Vous ne pouvez pas générer de code à partir de cet outil
Itemis Create/Yakindu Statechart Tools	Facile, licence gratuite sur le site web et licence professionnelle avec plus de fonctionnalités	Facile, Éditeur graphique et textuel	Oui, Supporte les extensions	Facile, Les modèles peuvent être importés et exportés	Oui, Génération de code possible
JFLAP	Facile, Gratuit, sur leur site web avec formulaire	Facile, Interface utilisateur intuitive	Non, Pas de support pour les extensions	Facile, Supporte les formats d'import/export	Non, Pas de génération de code

Table 1 – Liste des outils envisagés

machine et sa qualité ergonomique. Certains outils proposent des interfaces graphiques facilitant la modélisation et les interactions intuitives, tandis que d'autres privilégient une approche textuelle via l'ajout de scripts ou des prompts. Le Table 1 permet de voir rapidement quels outils étaient faciles à utiliser et assez complets en fonctionnalités. Cette comparaison générale inclut les outils que nous avons testé [6] mais seule une partie est retenue pour les expérimentations plus détaillée qui suivent. Certains outils ont été écartés en raison de leur complexité d'installation, de leur difficulté d'utilisation, ou de leurs limitations fonctionnelles.

4.2 Expressivité du langage

L'objectif est d'évaluer jusqu'où les langages utilisés par les outils peuvent aller dans la modélisation. La *gestion du temps* est essentielle dans les systèmes temps réel : certains outils utilisent des horloges explicites (comme UPPAAL), d'autres intègrent le temps de manière plus limitée. La *communication entre processus* est également importante, certains outils supportant des canaux de communication ou des échanges de messages. La prise en compte de la *probabilité* dans les modèles est plus rare, mais importante dans des contextes d'incertitude. Les outils permettent d'imposer des *conditions sur les transitions*, mais avec des niveaux de complexité différents. La *modularité* permet de construire des modèles plus grands en assemblant plusieurs petits morceaux. Cela aide à mieux organiser le travail et à réutiliser des parties du modèle. Enfin, la possibilité de définir des *couleurs* (types complexes pour les jetons) ou d'utiliser des *objets* enrichit fortement l'expressivité, mais n'est pas systématiquement présente.

	Modélisation					
	Temps	**Communications**	**Probabilité**	**Conditions**	**Modularité**	**Types**
Tina	Oui, le temps peut être ajouté sur les transitions et est supporté dans l'outil	Non, mais cela peut être simulé par des transitions	Non, il n'y a pas de probabilité	Non, il n'y a pas de conditions	Non, il n'y a pas de modularité	Non, seulement des jetons basiques
Romeo	Oui, le temps peut être ajouté sur les transitions mais ne sert qu'à prendre toutes les transitions avant d'avancer	Moyen, les communications peuvent être créées avec l'aide de variables et de bloqueurs	Non, il n'y a pas de probabilité	Oui, des conditions peuvent être ajoutées aux transitions	Oui, il y a de la modularité	Non, seulement des jetons basiques
CPN Tools	Moyen, le temps peut être ajouté aux jetons, les transitions, les arcs. Mais seulement en N	Non, mais cela peut être simulé par des transitions	Oui, utilisable avec certaines fonctions de distribution aléatoire	Oui, des conditions peuvent être ajoutées aux transitions	Non, il n'y a pas de modularité	Oui, des objets peuvent être définis et utilisés comme jetons
SPIN promela	Moyen, le temps peut être simulé, mais il n'est pas intégré nativement	Oui, communications via des canaux	Non, il n'y a pas de support natif pour la probabilité, seulement pour le non-déterminisme	Oui, supporté via des structures de contrôle comme if, for, while, et des gardes	Oui, il y a de la modularité	X
UPPAAL	Moyen, le temps peut être ajouté par des variables ou par un type d'horloge	Oui, communications via des canaux peuvent être faites	Non, il n'y a pas de probabilité	Oui, des conditions peuvent être ajoutées aux arcs	Oui, il y a de la modularité	X

TABLE 2 – Comparaison des outils (Modélisation / Expressivité)

6. La sélection n'est ni exhaustive ni représentative des meilleurs outils.

La Table 2 permet de voir la couverture par les outils des besoins de modélisation comme la gestion du temps, la possibilité de faire des communications entre les parties concurrentes, l'ajout des conditions sur les transitions, l'utilisation de types plus riches (couleurs ou objets dans les RDP), paramètres. La richesse du langage de calcul fonctionnel (familier aux informaticiens) est un atout de même que l'accessibilité du formalisme à des non spécialistes. Tous les outils n'offrent pas la même expressivité ou la même richesse en termes de concepts. Certains comme UPPAAL, utilisent des horloges pour représenter le temps, tandis que d'autres le gère de manière plus limitée. La communication entre processus est également mieux supportée dans des outils comme Romeo, UPPAAL et SPIN. En revanche, la prise en compte de la probabilité est très rare et elle est intégrée que sur CPN Tools. Enfin, les gardes sur les transitions sont possibles dans de nombreux outils mais leur manière d'être exprimées varie selon environnement.

4.3 Vérification

Cette rubrique étudie les capacités (la puissance) des outils à garantir que les modèles satisfont certaines propriétés. Un *aperçu général* permet de voir si une vérification est possible et sous quelle forme. L'*analyse structurelle* examine la structure du modèle sans exécuter toutes les transitions (par exemple, matrices d'incidences, invariants). L'*analyse d'état* explore l'espace des états atteignables pour vérifier des propriétés dynamiques (absence de blocage, atteignabilité). Enfin, certains outils offrent la possibilité de vérifier directement des *prédicats*, en exprimant précisément des propriétés attendues sur le système.

	Vérification formelle		
	Analyse structurelle	**Analyse d'état**	**Prédicats**
Tina	Oui, mais avec les contraintes temporelles oubliées	Oui, les RDP peuvent être convertis en automates directement dans Tina.	Oui, des prédicats peuvent être ajoutés dans l'outil de vérification "vérification de l'accessibilité".
Romeo	Oui, l'outil vérifie la structure de chaque RDP	Oui, l'outil vérifie les places, les transitions, les temps et les variables	Oui, des prédicats peuvent être ajoutés dans la page de vérification
CPN Tools	Oui, mais cela peut être long avec de petits RDP à cause des objets	Oui, l'outil réalise l'analyse d'état	Oui, des prédicats peuvent être ajoutés avant de faire le calcul d'état.
SPIN promela	Non, l'outil ne vérifie pas la structure de l'automate	Oui, il analyse tous les états possibles du système pour vérifier les erreurs	Oui, des prédicats peuvent être définis dans promela pour vérifier des conditions sur l'état ou les actions
UPPAAL	Oui, l'outil vérifie la structure de l'automate	Oui, l'outil vérifie tous les états, les transitions, les variables et signaux	Oui, des prédicats peuvent être définis dans la page de vérification

TABLE 3 – Comparaison des outils (vérification de modèles)

La Table 3 compare les capacités de vérification des différents outils de modélisation. Trois aspects sont étudiés : (1) L'analyse structurelle : Tous les outils excepté SPIN/Promela offrent cette fonctionnalité, avec certaines limitations temporelles pour Tina mais une performance moins bonne pour CPN Tools sur les grands réseaux (*cf.* Figure 2). (2) L'analyse d'état : Tous les outils supportent cette fonctionnalité essentielle, avec SPIN se distinguant par sa capacité à analyser tous les états possibles du système (*cf.* Figure 3). (3) Les prédicats : La plupart des outils permettent de définir des prédicats dans leur interface de vérification, facilitant ainsi la spécification de propriétés à vérifier 20, 24, 28.

4.4 Simulation

La dernière catégorie se concentre sur les outils permettant de *simuler* le comportement dynamique du système modélisé, ce qui est utile pour détecter des erreurs avant toute phase de vérification exhaustive. La simulation est une exigence sur certains domaines où on veut tester ou comparer des versions de systèmes pour choisir le meilleur. L'*analyse de performance* (temps d'exécution, utilisation de ressources) est également envisagée pour évaluer l'efficacité du modèle. Enfin, la possibilité d'obtenir des *traces* d'exécution est un point clé pour comprendre et analyser les séquences d'événements.

	Vérification formelle		
	Simulation	**Analyse de performance**	**Traces**
Tina	Oui, pas à pas ou automatique aléatoire. Le temps est bien géré	Non, il n'y a pas d'outils pour calculer la performance. Mais vous pouvez les développer	Oui, vous pouvez extraire l'historique de la simulation
Romeo	Oui, la simulation peut être effectuée mais seulement pas à pas	Non, il n'y a pas d'outils pour calculer la performance	Oui, vous pouvez extraire la simulation après qu'elle ait été effectuée
CPN Tools	Oui, pas à pas ou automatique aléatoire mais seulement par tranches de 50 itérations	Oui, il est possible d'obtenir les performances souhaitées	Oui, vous pouvez extraire l'historique de la simulation
SPIN promela	Oui, simulations interactives et aléatoires pour explorer les exécutions du modèle	Non, il est axé sur la correction, pas sur l'efficacité	Oui, il génère des traces d'exécution pour montrer la séquence des actions
UPPAAL	Oui, la simulation peut être effectuée pas à pas ou de manière aléatoire	Non, il n'y a pas d'outils pour calculer la performance	Oui, la trace peut être récupérée depuis la page de simulation concrète

TABLE 4 – Comparaison des outils (statique / dynamique)

La Table 4 compare les capacités de simulation et d'analyse dynamique des différents outils de modélisation. Cette comparaison montre que les outils varient dans leurs approches avec certains privilégiant l'exploration interactive comme SPIN et d'autres l'analyse quantitative comme CPN Tools, permettant ainsi de choisir l'outil le plus adapté selon le type d'analyse dynamique recherché. Trois aspects sont étudiés : (1) Simulation : Tous les outils offrent des capacités de simulation, mais avec des variations. Les outils RDP permettent généralement une exécution pas à pas ou automatique, tandis que SPIN/Promela se distingue par son approche interactive et aléatoire pour explorer les exécutions du modèle. (2) Analyse de performance : Seul CPN Tools parmi les outils étudiés propose des fonctionnalités natives d'analyse de performance, tandis que UPPAAL offre également une observation des performances dans son interface de simulation. (3) Traces : Tous les outils permettent d'extraire ou de visualiser l'historique des simulations, avec SPIN offrant particulièrement un bon support pour retracer la séquence des actions. Une trace produit une exécution, dont on va pouvoir déterminer les propriétés par exploration, notamment via des logiques temporelles [BBL+99].

5 Discussion

Rappelons que l'objectif initial est de proposer un cadre pour aider un analyste à trouver un outil pour modéliser, vérifier et simuler des SED.

La première interrogation est de savoir s'il n'existe pas déjà une méthodologie pour cela et/ou des benchmarks comparatifs d'outils. (i) Nous avons interrogé une IA générative avec le simple prompt "*je cherche une classification des systèmes à événements discrets prenant en*

compte modélisation, vérification et simulation". Le résultat est disponible dans la Section D.1 de l'annexe web[1]. ChatGPT propose une classification des formalismes sur 4 types (RdP, FSM, Timed Automata, langages DEVS/UML), une sur les 3 activités, sur le niveau de formalisme, par domaine d'application et un tableau récapitulatif. C'est finalement un premier niveau qui correspond à la vision générale d'un étudiant de Master. (ii) Nous avons cherché des classification d'approches SED et de *model checkers* dans la littérature. Il existe de nombreux ouvrages sur DES (en anglais) et nombre s'intéressent à la simulation ou en opposition avec les systèmes continus. [CL08] couvre largement les formalismes mais pas les outils, D'autres couvrent une branche particulière, par exemple les SED stochastiques pour [Zim07] ou le contrôle via RdP et automates pour [SSVS13], là aussi sans outillage. L'outil de recherche du site de référence des RDP [7] ne fonctionne pas. Pour la vérification, [BBL+99] propose une vision pédagogique, mais l'outillage reste illustratif. (iii) Nous avons cherché des benchmarks. Nous n'en avons pas trouvé pour les outils SED. Pour les *model checkers*, il s'agit souvent de démontrer la performance sur une catégorie particulière de problèmes, par exemple [And18] en propose un pour le temps paramétré, ou des challenges comme le Waters INRIA [8] pour le temps-réel ou le Model Checking Contest [9].

Dans cette étude menée en initiation à la recherche, le travail de comparaison et d'expérimentation autour des outils de modélisations et de vérifications formelles nous a permis de mieux comprendre leurs usages, leurs limites, et les avantages qu'ils peuvent offrir selon les besoins du système à modéliser. En explorant différentes approches, nous avons pu mieux comprendre dans quels mesure ces outils peuvent être appliqués à des systèmes réels comme ceux décrits dans nos cas d'étude. Nous discutons ici diverses observations de nos expériences.

Une première observation est que le choix de l'outil dépend fortement du contexte et des objectifs du projet. Par exemple, si l'on cherche à simuler le comportement général d'un système de production, certains outils offrent des interfaces graphiques simples et une simulation rapide. D'autres part, certains outils sont basées essentiellement dans le but de prouver rigoureusement les propriétés vérifiables des systèmes modélisés permettant aux outils plus techniques une vérification formelle complète. Dans notre cas, le domaine d'application cible est celui de la gestion de production. En résumé, il est important de *déterminer son besoin* pour choisir ses outils solutions. Les caractéristiques de nos tableaux peuvent contribuer à le déterminer.

Le cadre est-il une référence dans cet objectif. C'est un *benchmark* qui permet d'initier la démarche et de voir des exemples mais pas plus à ce stade. Le résultat présenté n'est pas exempt de critiques. Si on représente la démarche par un arbre de décision, on peut affirmer qu'il n'est complet ni en largeur (éventail des outils) ni en profondeur (détail des caractéristiques attendues, profondeur de nos analyses) mais nous avons les premiers nœuds. Si en plus on intègre la dimension du domaine d'application, trois cas d'étude n'est pas significatif. Une façon d'approfondir les travaux est chercher et tester de nouveaux outils en incluant les tests des outils non testé dans la Table 1. Il est aussi possible de tester sur de nouveaux cas d'études pouvant être plus complexe et volumineux afin de vérifié les limites de calcul des outils. Dans certains cas par exemple nous nous sommes heurté à la génération d'automate via les RDP du cas d'exemple A.3 car en fonctions du nombres de palettes et d'objets dans le système le nombre d'état devient rapidement trop élevé à calculer. En résumé, l'analyse doit être approfondie, élargie et mise en perspective avec plus de besoins.

La méthodologie de choix de solution consiste ici assez naïvement déterminer si les caractéristiques attendues pour un besoin sont présentes. La réponse est sans doute plus progressive

7. https://www2.informatik.uni-hamburg.de/TGI/PetriNets/tools/search.html
8. https://waters2017.inria.fr/challenge/
9. https://mcc.lip6.fr/2025/

(on part de grandes catégories pour converger) et plus nuancée (n ne peut tout avoir, il faut établir des compromis). Par exemple, plus le langage est expressif, plus on augmente le domaine d'application mais plus il sera difficile de tout vérifier. Il faut faire des choix discriminants et fixer des priorités sur les caractéristiques suivantes : (i) formalisme visuel, (ii) types de données, (iii) algorithmique (calcul fonctionnel), (iv) modularité (réutilisation), (v) aide à la preuve, (vi) gestion du temps et simulation, (vii) analyse des traces, (viii) etc. La prise en compte du temps et des horloges est très variable d'une approche à l'autre. De même la conduite des preuve est plus ou moins technique. Un arbre de décision met en évidence les critères de choix. le *benchmark* devra couvrir un exemple de chaque type de besoin (les feuilles de l'arbre).

6 Conclusion

Le travail présenté ici part d'une question simple, quel outil devrais-je choisir pour analyser la dynamique d'un système à événements discrets. La réponse n'est pas simple et dans cet article nous avons proposé se suivre une approche méthodologique pour aider le non-spécialiste à s'y retrouver. Nous proposons un premier niveau de grille d'évaluation pour comparer les outils sur différentes caractéristiques selon le formalisme, la démarche et les techniques associées. Nous avons testé différents outils sur différentes études de cas dans l'idée de produire un *benchmark* de comparaison. On peut représenter l'espace d'exploration par une matrice en trois dimensions : les caractéristiques à prendre en compte, les outils et les domaines d'utilisation. L'algorithme de décision consiste alors à naviguer dans cet espace pour déterminer une ou plusieurs solutions. La réponse à la question doit aussi être modulée selon l'expertise de celui qui se la pose. Actuellement la grille s'adresse au débutant et les avis sont à confirmer par d'autres utilisateurs.

Les perspectives sont nombreuses : (i) Enrichir le référentiel sur les trois axes. La grille devra être enrichie en détaillant chaque caractéristique actuelle par des caractéristiques plus fines, car les outils sont souvent spécialisés et efficients sur de types spécifiques de problèmes formels. L'ensemble des outils n'est ni représentatif (on n'a pas les meilleurs représentants de chaque catégorie) ni exhaustifs (la liste des outils n'est pas explorée). La grille doit être élargie en considérant de nouveaux outils. Enfin, les études de cas ne sont pas représentatives. Le référentiel doit définir des domaines d'application et des cas d'étude par domaine. (ii) L'algorithme de décision est informel. Le structurer serait un plus pour aider l'ingénieur à mener sa réflexion, partant d'une spécification de son besoin, en lien avec l'analogie des cas du *benchmark* à leur satisfaction en trouvant un ou plusieurs outils candidats. (iii) La performance d'un outil est souvent liée à sa spécialisation. Trouver un outil permettra de répondre partiellement à son besoin, moyennant quelques compromis. L'étape supérieure est de trouver la combinaison d'outil permettant de mieux couvrir son besoin.
Un travail collectif serait pertinent pour mener à terme ces perspectives.

Les auteurs remercient l'Agence Nationale de la Recherche (ANR) pour son soutien financier dans le cadre du projet RODIC, numéro de subvention ANR-21-CE10-0017.

Références

[AD94] Rajeev Alur and David L Dill. A theory of timed automata. Theoretical computer science, 126(2) :183–235, 1994.

[And18] Étienne André. A benchmark library for parametric timed model checking. In Cyrille Artho and Peter Csaba Ölveczky, editors, Formal Techniques for Safety-Critical Systems

- 6th International Workshop, FTSCS 2018, Gold Coast, Australia, November 16, 2018, Revised Selected Papers, volume 1008 of Communications in Computer and Information Science, pages 75–83. Springer, 2018.

[Arn90] André Arnold. MEC : a system for constructing and analysing transition systems, page 117–132. Springer Berlin Heidelberg, 1990.

[BBL+99] Béatrice Bérard, Michel Bidoit, François Laroussinie, Antoine Petit, and Philippe Schnoebelen. Vérification de logiciels : techniques et outils du model-checking. Vuibert Informatique. Vuibert, 1999. ISBN 2-7117-8646-3.

[BDL04] Gerd Behrmann, Alexandre David, and Kim G Larsen. A tutorial on uppaal. Formal methods for the design of real-time systems, pages 200–236, 2004.

[BLL+96] Johan Bengtsson, Kim Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL—a tool suite for automatic verification of real-time systems. Springer, 1996.

[BRV04] Bernard Berthomieu*, P-O Ribet, and François Vernadat. The tool tina–construction of abstract state spaces for petri nets and time petri nets. International journal of production research, 42(14) :2741–2756, 2004.

[BV06] Bernard Berthomieu and François Vernadat. Time petri nets analysis with tina. In QEST, volume 6, pages 123–124, 2006.

[CCF+99] R Scott Cost, Ye Chen, Tim Finin, Yannis K Labrou, Yun Peng, et al. Modeling agent conversations with colored petri nets. In Working notes of the Autonomous Agents' 99 Workshop on Specifying and Implementing Conversation Policies, 1999.

[CL08] Christos G Cassandras and Stéphane Lafortune. Introduction to discrete event systems. Springer, 2008.

[CRS18] Rance Cleaveland, A. W. Roscoe, and Scott A. Smolka. Process Algebra and Model Checking, page 1149–1195. Springer International Publishing, 2018.

[Geh19] Vijay Gehlot. From petri nets to colored petri nets : A tutorial introduction to nets based formalism for modeling and simulation. In 2019 Winter Simulation Conference (WSC), pages 1519–1533, 2019.

[GLMR05] Guillaume Gardey, Didier Lime, Morgan Magnin, and Olivier (H) Roux. Romeo : A tool for analyzing time petri nets. In Computer Aided Verification : 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005. Proceedings 17, pages 418–423. Springer, 2005.

[GLMS11] Hubert Garavel, Frédéric Lang, Radu Mateescu, and Wendelin Serwe. Cadp 2010 : A toolbox for the construction and analysis of distributed processes. In Int. conf. on tools and algorithms for the construction and analysis of systems, pages 372–387. Springer, 2011.

[GMBS96] Marie-Claude Gaudel, Bruno Marre, Gilles Bernot, and Françoise Schlienger. Précis de génie logiciel. Elsevier Masson, Barcelona, Spain, 1996.

[GV03] C. Girault and R. Valk. Petri Nets for Systems Engineering : A Guide to Modeling, Verification, and Applications. Springer, 2003.

[Hol04] Gerard J Holzmann. The SPIN model checker : Primer and reference manual, volume 1003. Addison-Wesley Reading, 2004.

[Nor03] Nordgren. Flexsim simulation environment. In Proceedings of the 2003 Winter Simulation Conference, 2003., volume 1, pages 197–200. IEEE, 2003.

[RWL+03] Anne Vinter Ratzer, Lisa Wells, Henry Michael Lassen, Mads Laursen, Jacob Frank Qvortrup, Martin Stig Stissing, Michael Westergaard, Søren Christensen, and Kurt Jensen. Cpn tools for editing, simulating, and analysing coloured petri nets. In International conference on application and theory of petri nets, pages 450–462. Springer, 2003.

[SSVS13] Carla Seatzu, Manuel Silva, and Jan H Van Schuppen. Control of discrete-event systems, volume 433. Springer, 2013.

[Zim07] Armin Zimmermann. Stochastic discrete event systems. Springer, 2007.