Rapport de Modélisation UPPAAL

Système de Livraison par Drones

Réalisé par :

Nabil BOUKERZAZA
Yahya BENBABA

Supervisé par :

Marie DUFLOT-KREMER
Stephan MERZ

Repository GitHub :
`https://github.com/Nabil-Bkz/-Syst-me-de-Livraison-par-Drones`

1er décembre 2025

Table des matières

1 Introduction

1.1 Contexte

Ce projet modélise un système de livraison automatisé par drones dans un environnement portuaire. Les drones transportent des conteneurs depuis une plateforme de chargement vers une plateforme de déchargement.

Problématique : Dans un port moderne, la logistique de transport de conteneurs est un défi majeur. L'utilisation de drones autonomes permet d'optimiser ce processus, mais nécessite une coordination précise pour éviter les collisions, gérer l'autonomie des batteries et réagir aux conditions météorologiques.

Approche : Nous utilisons UPPAAL, un outil de model-checking pour systèmes temps-réel, afin de modéliser et vérifier formellement le comportement du système avant son déploiement.

1.2 Objectifs

- **Modéliser le comportement des drones** : Chaque drone suit un cycle de mission (chargement, transport, déchargement) avec une gestion réaliste de la batterie.
- **Assurer l'exclusion mutuelle** : Une seule plateforme peut accueillir un drone à la fois pour éviter les collisions.
- **Gérer les urgences** : Le système doit réagir correctement aux situations critiques (batterie faible, tempête).
- **Vérifier formellement** : Utiliser le model-checking pour prouver l'absence de deadlock et garantir les propriétés de sûreté.

1.3 Composants du Système

Le système est composé de quatre processus concurrents qui interagissent via des canaux de synchronisation :

Processus	Rôle	Description
`d1`, `d2`	Drones de livraison	Transportent les conteneurs entre les plateformes
`c`	Contrôleur central	Initialise les drones et distribue les missions
`m`	Système météorologique	Simule les conditions météo et déclenche les alertes tempête

Table 1 – Composants du système

2 Déclarations Globales

Les déclarations globales définissent les constantes, variables et canaux partagés par tous les processus du système. Elles constituent le "vocabulaire commun" permettant la coordination entre les différents automates.

2.1 Constantes

Les constantes définissent les paramètres fixes du système. Elles permettent de modifier facilement la configuration sans changer le code des automates.

```
const int ND = 2;                // Nombre de drones
const int N_PLATFORMS = 2;       // Nombre de plateformes

// Constantes batterie
const int BAT_MAX = 100;         // Batterie maximale
const int BAT_COST = 5;          // Consommation par transition
const int BAT_CRITICAL = 10;     // Seuil critique : doit retourner
    charger
const int BAT_MIN_UNLOAD = 20;   // Seuil min pour aller vers
    dechargement
```

Explication :

- `ND = 2` : Le système utilise 2 drones, mais ce paramètre peut être augmenté pour tester la scalabilité.
- `BAT_COST = 5` : Chaque déplacement consomme 5% de batterie, ce qui permet environ 20 transitions avec une batterie pleine.
- `BAT_CRITICAL = 10` : En dessous de ce seuil, le drone doit impérativement se recharger.
- `BAT_MIN_UNLOAD = 20` : Ce seuil empêche un drone de partir vers le déchargement s'il risque de tomber en panne en route.

2.2 Variables

Les variables globales représentent l'état partagé du système, accessible et modifiable par tous les processus.

```
int[0,1] missionType;            // Type de mission (0 ou 1)
int available[ND];               // Disponibilite des drones (1=
    disponible)
int platform_lock[N_PLATFORMS] = {-1, -1};  // Verrous plateformes (-1=
    libre)
bool isStorm = false;            // Etat meteo global
```

Explication :

- `missionType` : Permet de distinguer différents types de missions (extensible pour des scénarios plus complexes).
- `available[ND]` : Tableau indiquant si chaque drone est disponible (1) ou occupé (0). Le contrôleur consulte ce tableau avant d'envoyer une mission.
- `platform_lock[]` : Mécanisme de verrou pour l'exclusion mutuelle. La valeur -1 indique une plateforme libre, sinon c'est l'ID du drone qui l'occupe.
- `isStorm` : Variable booléenne consultée par tous les processus pour savoir si une tempête est en cours.

2.3 Fonctions

Les fonctions encapsulent la logique de gestion des plateformes, rendant le code des automates plus lisible.

```
bool isPlatformFree(int p) {
```

```
    return platform_lock[p] == -1;
}

void lockPlatform(int p, int droneId) {
    platform_lock[p] = droneId;
}

void unlockPlatform(int p) {
    platform_lock[p] = -1;
}

int minBat(int a, int b) {
    return (a < b) ? a : b;
}
```

Explication :

— `isPlatformFree(p)` : Vérifie si la plateforme `p` est libre. Utilisée comme garde dans les transitions.

— `lockPlatform(p, droneId)` : Réserve la plateforme pour un drone spécifique. Garantit l'exclusion mutuelle.

— `unlockPlatform(p)` : Libère la plateforme quand le drone a terminé.

— `minBat(a, b)` : Fonction utilitaire pour limiter la batterie à sa valeur maximale lors de la recharge.

2.4 Canaux de Communication

Les canaux permettent la synchronisation entre processus. UPPAAL utilise des canaux `broadcast` où l'émetteur peut envoyer même si aucun récepteur n'écoute.

```
broadcast chan initCh;          // Initialisation des drones
broadcast chan missionCh[ND];   // Envoi de mission au drone i
broadcast chan stormStart;      // Signal debut tempete
broadcast chan stormEnd;        // Signal fin tempete
```

Explication :

— `initCh` : Canal utilisé par le contrôleur pour initialiser tous les drones simultanément.

— `missionCh[ND]` : Tableau de canaux permettant d'envoyer une mission à un drone spécifique (adressage individuel).

— `stormStart/stormEnd` : Canaux broadcast pour les alertes météo. Tous les processus peuvent réagir simultanément.

3 Template Drone

Le template `drone` est le composant principal du système. Il modélise le comportement complet d'un drone autonome, incluant le cycle de mission, la gestion de batterie et les réactions aux urgences.

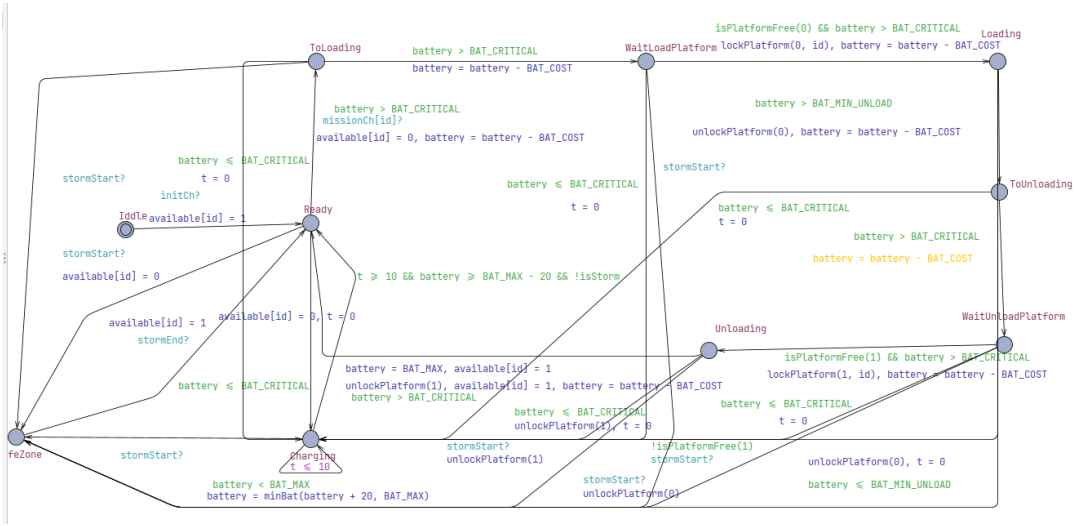

FIGURE 1 – Automate du drone dans UPPAAL

3.1 Capture d'écran UPPAAL

3.2 Déclarations Locales

Chaque instance de drone possède ses propres variables locales, indépendantes des autres drones.

```
int battery = BAT_MAX;   // Batterie initialisee a 100
clock t;                 // Horloge pour le temps de charge
```

Explication :

- `battery` : Niveau de batterie du drone, initialisé à 100%. Décrémenté à chaque transition.
- `t` : Horloge locale utilisée pour mesurer le temps de recharge (minimum 10 unités de temps).

3.3 États

Le drone possède 10 états représentant les différentes phases de son fonctionnement :

Cycle normal : Iddle → Ready → ToLoading → WaitLoadPlatform → Loading → ToUnloading → WaitUnloadPlatform → Unloading → Ready

3.4 Transitions Principales

Les transitions définissent les changements d'état possibles avec leurs conditions (gardes) et actions (assignments).

Explication des gardes importantes :

- `battery > BAT_CRITICAL` : Empêche le drone de commencer une action s'il n'a pas assez de batterie.
- `battery > BAT_MIN_UNLOAD` : Garantit que le drone a assez de batterie pour terminer la livraison.
- `isPlatformFree(p)` : Assure l'exclusion mutuelle sur les plateformes.
- `t >= 10 && !isStorm` : Le drone ne peut quitter la charge qu'après 10 ut et si pas de tempête.

État	Description
`Iddle`	État initial, le drone attend d'être initialisé par le contrôleur
`Ready`	Le drone est prêt et disponible pour recevoir une mission
`ToLoading`	Le drone se déplace vers la plateforme de chargement
`WaitLoadPlatform`	Le drone attend que la plateforme 0 soit libre
`Loading`	Le drone charge un conteneur (occupe la plateforme 0)
`ToUnloading`	Le drone transporte le conteneur vers la plateforme de déchargement
`WaitUnloadPlatform`	Le drone attend que la plateforme 1 soit libre
`Unloading`	Le drone décharge le conteneur (occupe la plateforme 1)
`Charging`	Le drone se recharge (invariant : `t <= 10`)
`SafeZone`	Zone de sécurité en cas de tempête

TABLE 2 – États du drone

Transition	Guard	Sync	Assignment
Iddle → Ready	-	`initCh?`	`available[id] = 1`
Ready → ToLoading	`battery > BAT_CRITICAL`	`missionCh[id]?`	`available[id] = 0`
WaitLoad → Loading	`isPlatformFree(0)`	-	`lockPlatform(0, id)`
Loading → ToUnloading	`battery > BAT_MIN_UNLOAD`	-	`unlockPlatform(0)`
WaitUnload → Unloading	`isPlatformFree(1)`	-	`lockPlatform(1, id)`
Unloading → Ready	`battery > BAT_CRITICAL`	-	`unlockPlatform(1)`
Charging → Ready	`t >= 10 && !isStorm`	-	`battery = BAT_MAX`
→ SafeZone	-	`stormStart?`	(libère plateforme)
SafeZone → Ready	-	`stormEnd?`	`available[id] = 1`

TABLE 3 – Transitions principales du drone

3.5 Gestion Batterie

La gestion de la batterie est un élément crucial pour la sûreté du système :

— **Consommation** : Chaque transition consomme 5 unités (-5 par déplacement)

— **Seuil critique ($\leq$10)** : Le drone doit immédiatement aller se recharger. Des transitions d'urgence existent depuis chaque état vers `Charging`.

— **Seuil déchargement ($\leq$20)** : Le drone ne peut pas partir vers `ToUnloading` car il risquerait de tomber en panne avec un conteneur.

— **Recharge** : La batterie se recharge par paliers de +20, avec une durée minimum de 10 unités de temps (invariant sur l'état `Charging`).

Pourquoi ces seuils ? Le seuil de 20 pour le déchargement garantit que le drone a assez d'énergie pour : aller à la plateforme de déchargement (5), décharger (5), et retourner à la charge si nécessaire (10 de marge).

4 Template Controller

Le template `controller` représente le centre de contrôle qui coordonne les drones. Il est responsable de l'initialisation du système et de la distribution des missions.

4.1 Capture d'écran UPPAAL

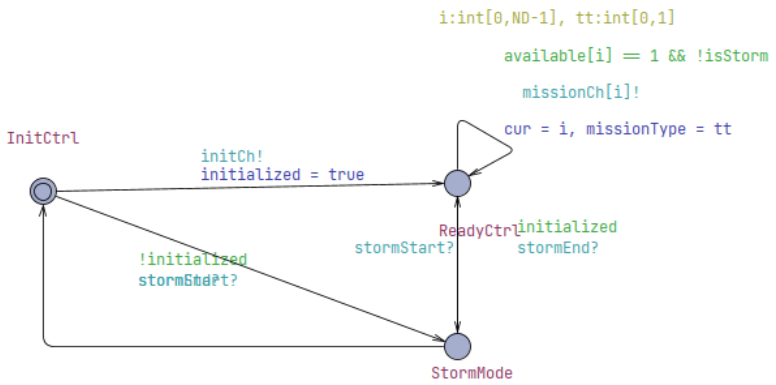

FIGURE 2 – Automate du contrôleur dans UPPAAL

4.2 Déclarations Locales

```
int[0,ND-1] cur;           // Indice du drone actuellement cible
bool initialized = false;  // True apres initialisation des drones
```

Explication :

- `cur` : Mémorise l'identifiant du dernier drone à qui une mission a été envoyée.
- `initialized` : Flag indiquant si les drones ont été initialisés. Permet de gérer correctement le retour après une tempête survenue avant l'initialisation.

4.3 États

Le contrôleur a seulement 3 états, reflétant sa logique simple mais efficace :

4.4 Transitions

Explication du comportement :

- **Initialisation** : Le contrôleur envoie `initCh!` pour réveiller tous les drones simultanément.
- **Envoi de mission** : La transition self-loop sur `ReadyCtrl` utilise un `select` pour choisir un drone `i` disponible. La garde `!isStorm` empêche l'envoi pendant une tempête.

État	Description
`InitCtrl`	État initial, le contrôleur doit d'abord initialiser les drones
`ReadyCtrl`	État nominal, le contrôleur peut envoyer des missions aux drones disponibles
`StormMode`	Mode dégradé pendant une tempête, aucune mission n'est envoyée

TABLE 4 – États du contrôleur

Transition	Guard	Sync	Assignment
InitCtrl → ReadyCtrl	-	`initCh!`	`initialized = true`
InitCtrl → StormMode	-	`stormStart?`	-
ReadyCtrl → ReadyCtrl	`available[i] && !isStorm`	`missionCh[i]!`	`cur = i`
ReadyCtrl → StormMode	-	`stormStart?`	-
StormMode → ReadyCtrl	`initialized`	`stormEnd?`	-
StormMode → InitCtrl	`!initialized`	`stormEnd?`	-

TABLE 5 – Transitions du contrôleur

— **Gestion tempête** : Le contrôleur peut recevoir `stormStart?` depuis n'importe quel état et passe en `StormMode`.
— **Retour après tempête** : Si les drones étaient initialisés, retour à `ReadyCtrl`. Sinon, retour à `InitCtrl` pour les initialiser.

5 Template Meteo

Le template `meteo` simule les conditions météorologiques. Il peut déclencher des tempêtes de manière non-déterministe, forçant tout le système à se mettre en mode sécurité.

5.1 Capture d'écran UPPAAL

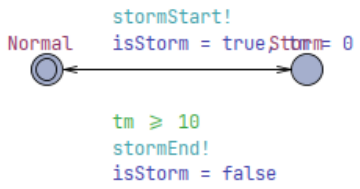

FIGURE 3 – Automate météo dans UPPAAL

5.2 Déclarations Locales

```
clock tm;  // Horloge pour duree tempete
```

Explication : L'horloge `tm` mesure la durée de la tempête. Elle est réinitialisée au début de chaque tempête et la transition vers `Normal` n'est possible que si `tm >= 10`.

5.3 États

État	Description
`Normal`	Conditions météo normales, le système fonctionne normalement
`Storm`	Tempête en cours, tous les drones doivent se mettre en sécurité

TABLE 6 – États du système météo

5.4 Transitions

Transition	Guard	Sync	Assignment
Normal → Storm	-	`stormStart!`	`isStorm = true, tm = 0`
Storm → Normal	`tm >= 10`	`stormEnd!`	`isStorm = false`

TABLE 7 – Transitions du système météo

Explication :
- **Déclenchement non-déterministe** : La transition vers `Storm` n'a pas de garde, donc une tempête peut survenir à tout moment. Cela permet de tester la robustesse du système.
- **Durée minimale** : La garde `tm >= 10` garantit que la tempête dure au moins 10 unités de temps.
- **Synchronisation broadcast** : `stormStart!` et `stormEnd!` sont reçus par tous les processus simultanément.

6 Système Global

Cette section décrit comment les différents templates sont instanciés et composés pour former le système complet.

6.1 Instanciation

```
d1 = drone(0);
d2 = drone(1);
c = controller();
m = meteo();

system d1, d2, c, m;
```

Explication :
— `drone(0)` et `drone(1)` : Deux instances du template drone avec des identifiants différents (0 et 1).
— `controller()` et `meteo()` : Instances uniques du contrôleur et du système météo.
— `system d1, d2, c, m` : Composition parallèle des 4 processus.

6.2 Fonctionnement Global

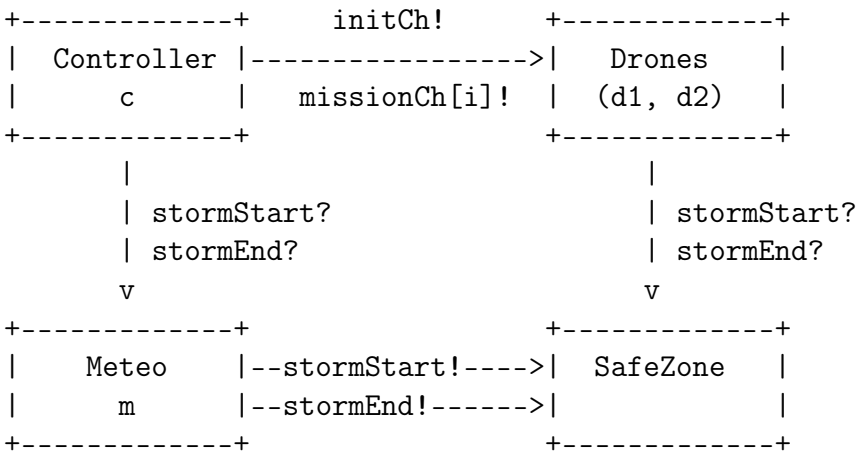

FIGURE 4 – Architecture du système

Flux de communication :
— Le **contrôleur** envoie des commandes aux **drones** (initCh, missionCh)
— La **météo** envoie des alertes à **tous** les processus (stormStart, stormEnd)
— Les **drones** partagent l'accès aux **plateformes** via les verrous globaux

6.3 Cycle de Mission

Une mission typique se déroule en 4 phases :

1. **Initialisation** : Le contrôleur envoie `initCh!`. Tous les drones passent de `Iddle` à `Ready` et deviennent disponibles (`available[id] = 1`).
2. **Envoi de mission** : Le contrôleur sélectionne un drone disponible et envoie `missionCh[i]!`. Le drone passe en `ToLoading` et devient indisponible.
3. **Exécution** : Le drone suit le cycle complet :
   — Se déplace vers la plateforme de chargement
   — Attend si la plateforme est occupée
   — Charge le conteneur (verrouille la plateforme 0)
   — Se déplace vers la plateforme de déchargement
   — Attend si la plateforme est occupée
   — Décharge le conteneur (verrouille la plateforme 1)
4. **Fin** : Le drone revient en `Ready` et redevient disponible pour une nouvelle mission.

6.4 Gestion Tempête

Le protocole de gestion des tempêtes garantit la sécurité de tous les drones :

1. **Détection** : Le système météo envoie `stormStart!`
2. **Réaction contrôleur** : Passe en `StormMode`, arrête d'envoyer des missions
3. **Réaction drones** : Tous les drones vont en `SafeZone`
   — Si sur une plateforme : libère le verrou avant de partir
   — Si en transit : abandonne la mission en cours
4. **Attente** : La tempête dure au minimum 10 unités de temps
5. **Fin** : Le système météo envoie `stormEnd!`
6. **Reprise** : Tous les processus reprennent leur fonctionnement normal

7 Propriétés et Vérification

La vérification formelle est l'étape clé qui garantit que notre modèle respecte les exigences de sûreté et de vivacité. UPPAAL utilise la logique temporelle TCTL pour exprimer ces propriétés.

7.1 Capture d'écran des Résultats

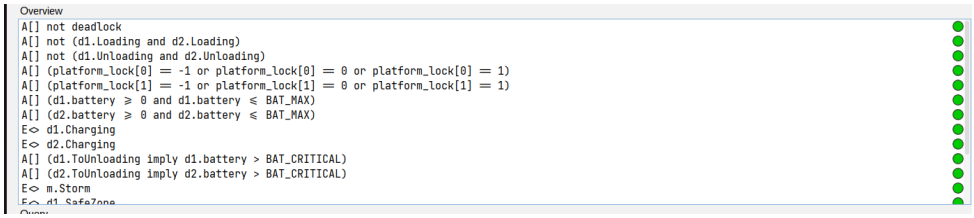

FIGURE 5 – Résultats de la vérification dans UPPAAL

7.2 Propriétés de Sûreté (Safety)

Les propriétés de sûreté garantissent que "quelque chose de mauvais ne se produit jamais". Elles sont exprimées avec le quantificateur `A[]` (pour tous les chemins, toujours).

Propriété	**Formule**	**Description**
Absence deadlock	`A[] not deadlock`	Le système ne se mais
Exclusion plateforme 0	`A[] not (d1.Loading and d2.Loading)`	Jamais 2 drones ment
Exclusion plateforme 1	`A[] not (d1.Unloading and d2.Unloading)`	Jamais 2 drones en ment
Batterie bornée	`A[] (d1.battery >= 0 and d1.battery <= BAT_MAX)`	Batterie dans [0,100
Sécurité déchargement	`A[] (d1.ToUnloading imply d1.battery > BAT_CRITICAL)`	Assez de batterie p
Cohérence météo	`A[] (m.Storm imply (c.StormMode or c.InitCtrl))`	Contrôleur réagit à

TABLE 8 – Propriétés de sûreté

Explication des propriétés critiques :

— **Absence de deadlock** : Garantit que le système peut toujours progresser. Un deadlock signifierait que tous les processus sont bloqués.
— **Exclusion mutuelle** : Prouve qu'il n'y a jamais de collision sur les plateformes.
— **Sécurité déchargement** : Garantit qu'un drone ne part jamais vers le déchargement avec une batterie insuffisante.

7.3 Propriétés de Vivacité (Liveness)

Les propriétés de vivacité garantissent que "quelque chose de bon finit par se produire". Elles sont exprimées avec `E<>` (il existe un chemin où éventuellement).

Propriété	Formule	Description
Charge possible	`E<> d1.Charging`	Un drone peut atteindre l'état de charge
Tempête possible	`E<> m.Storm`	Une tempête peut survenir
SafeZone accessible	`E<> d1.SafeZone`	Un drone peut aller en zone sécurisée
Livraison possible	`E<> d1.Unloading`	Un drone peut effectuer une livraison

TABLE 9 – Propriétés de vivacité

Importance des propriétés de vivacité :
— **Livraison possible** : Prouve que le système peut effectivement accomplir sa mission principale.
— **Charge possible** : Vérifie que le mécanisme de recharge est accessible.
— **SafeZone accessible** : Confirme que les drones peuvent se mettre en sécurité.

7.4 Tableau Récapitulatif

#	Propriété	Résultat
1	`A[] not deadlock`	✓
2	`A[] not (d1.Loading and d2.Loading)`	✓
3	`A[] not (d1.Unloading and d2.Unloading)`	✓
4	`A[] (d1.battery >= 0 and d1.battery <= BAT_MAX)`	✓
5	`A[] (d2.battery >= 0 and d2.battery <= BAT_MAX)`	✓
6	`A[] (d1.ToUnloading imply d1.battery > BAT_CRITICAL)`	✓
7	`A[] (d2.ToUnloading imply d2.battery > BAT_CRITICAL)`	✓
8	`A[] (m.Storm imply (c.StormMode or c.InitCtrl))`	✓
9	`E<> d1.Charging`	✓
10	`E<> d2.Charging`	✓
11	`E<> m.Storm`	✓
12	`E<> d1.SafeZone`	✓
13	`E<> d2.SafeZone`	✓
14	`E<> d1.Unloading`	✓
15	`E<> d2.Unloading`	✓

TABLE 10 – Résultats de vérification - Toutes les propriétés sont satisfaites

8 Conclusion

8.1 Résumé

Nous avons modélisé et vérifié formellement un système de livraison par drones comprenant :

— **2 drones** avec gestion réaliste de la batterie et comportements d'urgence
— **1 contrôleur** pour la coordination et la distribution des missions
— **1 système météo** pour la simulation des conditions défavorables
— **2 plateformes** avec mécanisme d'exclusion mutuelle

8.2 Résultats

Toutes les **15 propriétés** ont été vérifiées avec succès par UPPAAL :

— **8 propriétés de sûreté** (A[]) : Garantissent l'absence de comportements dangereux
— **7 propriétés de vivacité** (E<>) : Garantissent que le système peut accomplir ses objectifs

8.3 Points Clés

1. **Absence de deadlock** : Le système progresse toujours grâce à l'utilisation de canaux broadcast et de transitions d'urgence.
2. **Exclusion mutuelle** : Le mécanisme de verrous garantit qu'un seul drone occupe chaque plateforme à tout moment.
3. **Gestion de batterie** : Les seuils critiques (10) et de déchargement (20) empêchent tout drone de rester bloqué avec une batterie insuffisante.
4. **Réactivité aux tempêtes** : Tous les composants réagissent correctement aux alertes météo, assurant la sécurité des drones.

8.4 Perspectives

Ce modèle pourrait être étendu pour :

— Augmenter le nombre de drones et de plateformes
— Ajouter des priorités entre missions
— Modéliser des pannes de drones
— Intégrer des contraintes de temps plus réalistes