Hamac

NOTE DE CONCEPTION

Plan de contrôle à tours

Le modèle d'exécution du plan de contrôle de Hamac : une fonction pure, exécutée par tours, dont les propriétés de sécurité sont des invariants vérifiés à chaque tour.

Version 0.1 · 2026-10-19 · PROPOSITION

OCamlPro SAS — forge.ocamlpro.com/OSS/hamac — AGPL-3.0

Sommaire

1 Objectif **2**
2 Le modèle **3**
2.1 Participants 3
2.2 Un tour 3
2.3 Ensemble entendu 3
2.4 Signature 3
2.5 État 3
2.6 Commandes 4
3 Invariants **5**
3.1 Règle de quorum 5
3.2 Ordonnancement sûr par préfixe 5
4 Scénarios **6**
4.1 Déploiement initial 6
4.2 Perte d’un nœud 6
4.3 Partition réseau 6
4.4 Migration — `hamac drain n1` 7
5 Ce que le modèle apporte **8**
6 Contraintes de conception qui en découlent **9**
7 Barrières locales **10**
8 Ce que le modèle ne résout pas **11**
9 Rapport aux autres documents **12**
10 Références **13**

SECTION 1

Objectif

> **NOTE**
> **Statut** : proposition de conception, non implémentée. Ce document fixe le modèle d'exécution du plan de contrôle avant l'écriture de la boucle de réconciliation (Milestone 9, cycle de vie). Il tranche la question « générateur ou orchestrateur » en faveur de l'orchestrateur, et donne le cadre dans lequel les propriétés de sécurité du manifeste deviennent vérifiables.

Un orchestrateur observe un parc, décide, agit, et recommence. Cette boucle est en général écrite comme une réaction à des événements : les décisions dépendent alors de l'ordre d'arrivée des messages, des caches consultés et des délais d'expiration choisis. Un incident n'est pas reproductible, et aucune propriété de sûreté n'est démontrable.

Ce document pose un modèle d'exécution différent : le plan de contrôle est un **programme à tours**. Il n'en découle aucune contrainte de temps réel, aucune hypothèse sur la latence du réseau et aucune horloge synchronisée. Il en découle en revanche le déterminisme, la rejouabilité, et la possibilité d'énoncer les propriétés de sécurité comme des invariants de tour.

SECTION 2

Le modèle

2.1 Participants

- **Le contrôleur** : un processus qui exécute la fonction de transition. Il est unique, son état est persistant, il est redémarrable. Sa réplication est hors périmètre de ce document (§8).
- **Un agent par nœud** : il observe localement (services actifs, santé, ressources, interfaces) et exécute les commandes reçues. Il ne décide de rien, à une exception près (§5.3).

2.2 Un tour

Un tour, ou *superstep*, se décompose en quatre phases :

1. **Collecte** — chaque agent émet son rapport.
2. **Barrière** — le tour se clôt lorsque tous les agents ont rapporté, ou au plus tard après un délai `Δ` (ordre de grandeur : 2 s).
3. **Transition** — le contrôleur calcule.
4. **Diffusion** — les commandes sont envoyées aux agents, qui les exécutent de façon asynchrone.

L'effet d'une commande n'est jamais observé dans le tour qui l'émet : il apparaît dans un rapport ultérieur. **Un tour attend des rapports, jamais des travaux.** Une action longue — démarrer un conteneur, provisionner une machine par PXE — n'allonge pas le tour : elle se traduit par un état qui persiste sur plusieurs tours.

2.3 Ensemble entendu

La barrière se clôt sur délai, donc l'entrée de la transition n'est pas « l'état du parc » — qui n'est jamais connaissable — mais l'ensemble des rapports effectivement reçus à ce tour, noté `heard_of`.

C'est la décision de modélisation centrale : **un nœud mort, un nœud lent, un nœud derrière une partition réseau et un message perdu sont le même phénomène** — une absence de rapport. Le plan de contrôle ne contient donc aucun code de détection de panne, aucune distinction entre « lent » et « mort », aucune branche de bascule parallèle au chemin nominal. Il ne contient qu'un compteur de silence et une condition de quorum.

2.4 Signature

```
step : state -> heard_of -> state * commands
```

La fonction est **pure et totale** : pas d'entrée/sortie, pas de lecture d'horloge, pas d'accès réseau. Les effets vivent dans une couche d'exécution qui appelle `step` et applique les commandes émises. La partialité de l'observation est portée par le type `heard_of`, ce qui rend structurellement impossible de confondre « je n'ai pas entendu » avec « il est mort ».

Conséquences immédiates : mêmes entrées, mêmes sorties ; le journal des entrées suffit à rejouer un déploiement ou un incident hors production ; les tests ne demandent aucune infrastructure ; un modèle vérifiable peut être extrait plus tard sans réécriture.

2.5 État

Objet	États
Service	`absent`, `placed`, `starting`, `running`, `draining`, `stopped`, `failed`
Nœud	`live`, `silent(k)`, `evicted`
Global	plan courant, numéro de tour

2.6 Commandes

`start`, `stop`, `promote`, `fence`, `fw_add`, `fw_del`, `net_create`, `sozu_set_validator`, `sozu_add_route`, `sozu_add_backend`, `sozu_del_backend`, `provision`. La liste est ouverte ; ce qui compte est qu'une commande soit une **intention adressée à un agent**, jamais une action exécutée par le contrôleur.

SECTION 3

Invariants

Les propriétés de sécurité s'écrivent comme des observateurs : des prédicats sur l'état global, évalués à la fin de **chaque** tour.

RÈGLE

- **I1 — cloisonnement.** Aucun chemin réseau ouvert entre deux zones que le treillis de sécurité n'autorise pas.
- **I2 — unicité.** Au plus un primaire par service à état.
- **I3 — bordure.** Aucun backend joignable sans passer par la validation Biscuit.

La formulation « à chaque tour » n'est pas rhétorique : elle est la propriété différenciante du produit. Un orchestrateur usuel garantit — au mieux — que l'état final d'un déploiement est conforme. Ici, **tout état intermédiaire traversé est conforme**, ce qui est exactement la question posée en audit et à laquelle personne ne sait répondre aujourd'hui.

3.1 Règle de quorum

Toute décision susceptible de violer I2 exige que `heard_of` contienne un quorum du groupe concerné. À défaut, `step` ne décide pas : il signale et attend. La sûreté ne repose donc jamais sur une hypothèse temporelle, seulement sur une condition portant sur un ensemble.

3.2 Ordonnancement sûr par préfixe

L'ordre des commandes n'est pas dicté par les dépendances applicatives mais par les invariants : le solveur produit une séquence dont **chaque préfixe** satisfait I1, I2 et I3. S'il n'existe pas d'ordre satisfaisant, le plan est refusé au lieu d'être tenté.

C'est le même problème de contraintes que le placement — un seul investissement en résolution, deux usages.

SECTION 4

Scénarios

Les tours sont numérotés ; `Δ` vaut 2 s ; le seuil d'éviction vaut 3 tours. Configuration : `api` en zone `dmz` (n1, n2), `db` en zone `core` (n3), Sozu en bordure.

4.1 Déploiement initial

Tour	heard_of	Décision	Commandes
T0	n1, n2, n3	Placement résolu : api→n1, db→n3. Rien ne démarre : I1 exige le cloisonnement d'abord	`net_create(vlan-dmz, vlan-core)`, `fw_add(deny cross-zone)`
T1	n1, n2, n3	Cloisonnement confirmé → ouverture du seul flux autorisé, puis démarrage de la dépendance	`fw_add(api→db:5432)`, `start(db, n3)`
T2	n1, n2, n3	`db` en `starting`, pas encore sain. Aucune commande	—
T3	n1, n2, n3	`db` en `running` → la dépendance de `api` est satisfaite	`start(api, n1)`
T4	n1, n2, n3	`api` en `starting`	—
T5	n1, n2, n3	`api` en `running`. La route n'est pas publiée : I3 l'interdit tant que le validateur est absent	`sozu_set_validator(/api, pubkey, policy)`
T6	n1, n2, n3	Validateur confirmé	`sozu_add_route(/api → n1)`
T7	n1, n2, n3	État observé = état cible. Convergence	—

La durée du déploiement est un nombre de tours : prévisible, comparable d'une exécution à l'autre, opposable.

4.2 Perte d'un nœud

n3, qui porte `db`, cesse de rapporter.

Tour	heard_of	Décision	Commandes
T40	n1, n2	n3 → `silent(1)`. Un silence n'est pas une panne	—
T41	n1, n2	`silent(2)`	—
T42	n1, n2	`silent(3)` : seuil atteint. Quorum présent (2/3) → décision permise. Cloisonner **avant** de promouvoir	`fence(n3)` sur n1, n2
T43	n1, n2	Fence confirmé : I2 est garanti même si n3 revient	`promote(db-replica, n2)`
T44	n1, n2	Nouvelle primaire saine	`fw_update(api→n2:5432)`, `sozu_reload`
T45	n1, n2	Convergence. Indisponibilité ≈ 5 tours ≈ 10 s	—

Variante déterminante. Si à T42 `heard_of = {n1}` — pas de quorum — alors `step` ne promeut rien, signale, et attend. Le split-brain est écarté par une condition dans une fonction pure, et non par le réglage d'un délai.

4.3 Partition réseau

Le réseau se coupe entre `{contrôleur, n1, n2}` et `{n3}` ; aucun nœud n'est mort.

Vu du contrôleur, la situation est **identique au scénario précédent** — il n'a pas à distinguer, et c'est précisément l'intérêt du modèle.

Côté n3, une règle symétrique s'applique : un agent qui n'a reçu aucune commande depuis `k` tours passe en **mode conservateur** et arrête de lui-même les services dont il ne peut plus garantir la conformité, à commencer par les primaires à état. Cet auto-cloisonnement rend la promotion sûre même lorsque la commande `fence` n'a pas pu atteindre n3.

La correction de l'ensemble tient alors dans une inégalité :

```
k_agent < seuil_contrôleur
```

L'agent isolé se tait avant que le contrôleur ne promeuve. Toute la sûreté distribuée du système repose sur ces deux constantes, ce qui la rend vérifiable à la configuration et démontrable.

Retour de partition. n3 réapparaît dans `heard_of` avec un rapport indiquant son état et le tour de son auto-arrêt. `step` le réintègre comme ressource disponible, jamais comme primaire, et marque ses données divergentes : clone ou `pg_rewind` avant de redevenir réplique.

4.4 Migration — `hamac drain n1`

Séquence produite par le solveur, chaque préfixe satisfaisant les invariants :

1. `fw_add(n2 → db:5432)` — le flux depuis n2 est ouvert, celui de n1 subsiste
2. `start(api, n2)`, attendre `running` (2 à 3 tours)
3. `sozu_add_backend(/api → n2)` — les deux instances servent
4. `sozu_del_backend(/api → n1)` — plus aucun trafic vers n1
5. `stop(api, n1)`
6. `fw_del(n1 → db:5432)`

À aucun tour il n'existe une route pointant vers un backend non démarré, ni un flux réseau ouvert sans service qui le justifie. L'ordre inverse provoquerait une coupure ; ouvrir le flux après le démarrage laisserait un tour pendant lequel `api` ne peut pas joindre `db`. Ces erreurs sont exclues par construction, non par relecture.

SECTION 5

Ce que le modèle apporte

1. **Un instantané global cohérent, gratuitement.** Sans barrière, obtenir un cliché cohérent demande un protocole dédié. Avec barrière, l'état à la fin du tour est global et bien défini par construction.
2. **Un raisonnement séquentiel.** Plus d'entrelacements ni de courses : les preuves sont des inductions sur le numéro de tour.
3. **Le déterminisme et la rejouabilité**, donc la reproduction d'incident.
4. **La disparition du code de panne.** Un compteur de silence et une condition de quorum remplacent la machine à états de détection de défaillance.
5. **Une métrique unique.** Temps de déploiement, délai de bascule, délai d'éviction : tout se compte en tours.
6. **Le cas normal est le tour sans commande.** Un système convergé passe l'essentiel de son temps à confirmer qu'il n'a rien à faire — propriété que l'on veut précisément pouvoir démontrer.

SECTION 6

Contraintes de conception qui en découlent

- `step` est pure, totale, sans horloge ni entrée/sortie. **Cette contrainte doit être posée avant l'écriture de la boucle de réconciliation** : une boucle qui lit l'heure et parle au réseau au milieu de son calcul ne redevient jamais pure.
- L'entrée de `step` est `heard_of`, jamais un état du parc supposé complet.
- Les agents n'ont qu'une seule initiative : le repli conservateur.
- Les seuils `k_agent` et `seuil_contrôleur` sont des paramètres de sûreté, pas des réglages de performance ; leur relation est une obligation.
- Le journal des entrées est un artefact de premier ordre : il porte la rejouabilité et prépare la réplication du contrôleur.

SECTION 7

Barrières locales

Une barrière globale sur tout le parc est inutilisable en exploitation : le tour durerait ce que dure le participant le plus lent. Le découpage retenu est donc **une barrière par zone ou par sous-groupe**, les groupes échangeant par messages — un modèle globalement asynchrone, localement synchrone.

La question technique à instruire est celle des conditions sous lesquelles ce découpage préserve la sémantique du tour global (§10).

SECTION 8

Ce que le modèle ne résout pas

- **Le contrôleur est un point unique de défaillance.** Sa panne fige le système dans son dernier état : c'est *sûr* — rien de dangereux ne se produit — mais pas *vivant*. Le répliquer ramène un consensus sur son état, donc un dépôt à quorum. Le choix est repoussable : `step` étant pure et ses entrées journalisées, la réplication est un problème de journal, non de réécriture.
- **La migration d'état** reste le mur, indépendamment du modèle : elle relève du modèle de stockage du manifeste.
- **Le dynamisme du parc** — qui participe à la barrière — est un problème d'appartenance de groupe, équivalent au consensus. L'approche retenue est la recompilation à froid du plan lors d'un changement de topologie, cohérente avec le fait que Hamac est un compilateur.
- **L'explosion d'états** limite la vérification exhaustive : les invariants se vérifient sur une abstraction par zone, complétée par une vérification a posteriori du plan concret.

SECTION 9

Rapport aux autres documents

- **Générateur ou orchestrateur** — tranché : la boucle de réconciliation vit dans Hamac, les backends restent de simples exécutants. Le modèle à tours en est la forme.
- **Niveau de preuve** — il cesse d'être une coquetterie : les invariants I1–I3 et la relation `k_agent < seuil_contrôleur` sont les énoncés à établir.
- **Milestone 9** (cycle de vie, convergence) est la mise en œuvre directe de ce document.
- **Milestone 11** (haute disponibilité PostgreSQL) : le scénario 4.2 est le comportement attendu du provider PostgreSQL HA, et le scénario 4.3 le remède au split-brain constaté.
- La spécification du format de manifeste (`doc/typst/manifest.typ`, en brouillon) doit exposer les seuils et les classes de volume sur lesquels ces scénarios s'appuient.

SECTION 10

Références

- L. G. Valiant, *A Bridging Model for Parallel Computation*, 1990 — le modèle BSP : calcul local, échanges, barrière.
- B. Charron-Bost, A. Schiper, *The Heard-Of Model: computing in distributed systems with benign faults* — la panne comme absence d'information.
- C. Dwork, N. Lynch, L. Stockmeyer, *Consensus in the Presence of Partial Synchrony*, 1988 — sûreté toujours, vivacité sous conditions.
- J. Waldo, G. Wyant, A. Wollrath, S. Kendall, *A Note on Distributed Computing*, 1994 — pourquoi la transparence du distant est un leurre.
- A. Benveniste, P. Le Guernic, J.-P. Talpin et al., travaux sur les systèmes polychrones — endochronie et isochronie, conditions de préservation de la sémantique lors d'une distribution asynchrone (§7).
- A. Girault, *A survey of automatic distribution of synchronous programs* — distribution d'une spécification globale sur plusieurs nœuds.
- L. Lamport, K. M. Chandy, *Distributed Snapshots*, 1985 — le problème que la barrière rend sans objet.

Financement

« Ce projet a été financé par le Gouvernement dans le cadre du plan France 2030 opéré par Cap Digital et financé par l'Union européenne – NextGeneration EU »
« Ce projet a été financé par le gouvernement dans le cadre de France 2030 »
« Financé par l'Union européenne - Next Generation EU dans le cadre du plan France Relance »