

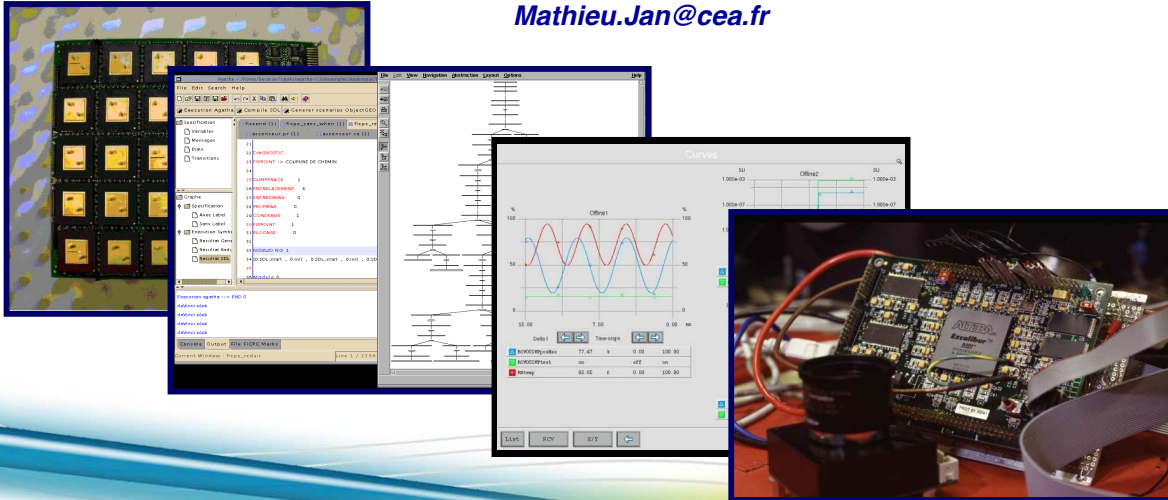
CEA LIST

Introduction aux systèmes cadencés par le temps réel (time-triggered)

Mathieu JAN

Laboratoire des fondements des systèmes temps-réels embarqués (LaSTRE)

Mathieu.Jan@cea.fr



Le Laboratoire Systèmes Temps Réel Embarqué

- Direction : Vincent David (32p + 8PhD)
- Méthodes et outils de conception, logiciel système et architectures multiprocesseurs pour :
 - La gestion du parallélisme
 - De la conception à la mise en œuvre (systèmes monoprocesseurs, systèmes multiprocesseurs répartis, systèmes multi-cœurs/MPSoC)
 - Des environnements temps-réel complexes
 - E.g. domaine nucléaire, automobile, avionique
 - Des systèmes critiques
 - Sécurité de fonctionnement, disponibilité
 - Des systèmes embarqués très contraints
 - Des architectures matérielles conventionnelles ou dédiées
 - Simple cœur (e.g. 68K, IA32, ARM), multicœur hétérogène (e.g. S12XE), multicœur SMP (e.g. IA32), architecture distribuée
- De la recherche fondamentale au transfert industriel
- Savoir faire : support système complet
 - Langages de programmation, chaînes de compilation, génération de code, édition des liens, analyse formelle et dimensionnement automatique
 - Conception et implémentation de noyaux multitâche orientés sûreté

- Concepts de base des systèmes cadencés par le temps réel
 - Comparaison avec une approche par évènement
 - Présentation du fonctionnement
 - Illustration via un aperçu d'OASIS
- Extension aux systèmes distribuées
 - Problématiques
 - Aperçu de TTP & OASIS-D
- Modèle de tâche
 - Application à la description du comportement temporel des tâches
 - Application à la description du comportement réseau

Paradigmes de conception guidé par le temps et les évènements

● Système « Event-triggered »

- Les activités d'un système sont fonction de l'apparition de différents évènements à l'exécution et de leur priorités

● Système « Time-triggered »

- Les activités d'un système sont fonction d'un cadencement temporel statiquement défini à la conception du système
- Ne pas introduire de sources d'asynchronismes dans la conception des applications
 - e.g. pas d'entrées/sorties sous interruptions
- Objectif : dominer les asynchronismes dus aux temps d'exécution variables
 - pour garantir le déterminisme
 - pour maîtriser le dimensionnement (performances)
- Séparation de la notion de cadencement de la notion d'ordonnancement

Concepts de base des systèmes cadencés par le temps réel

● Modèle de temps

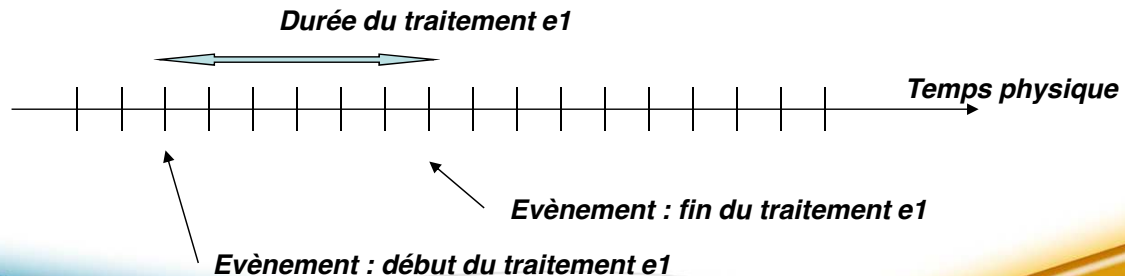
- Temps discret
- Nombre d'instants infinis
- Evènement : activité à un instant donné
 - Exemple : l'observation de l'état d'un système est un évènement
- Relation d'ordre (total) entre les évènements
- La durée d'un traitement doit être supérieure à la précision sur le temps physique

● Modélisation de l'état d'un système

- Ensemble de variables qui évoluent avec le temps
 - Une variable : couple <valeur, temps>

● Utilisation du temps pour la construction des traitements d'un système TT

- Dates d'activation et d'échéance des traitements
- Synchronisation entre tâches



● Echange de données via la mise à disposition de l'état d'une variable (state information)

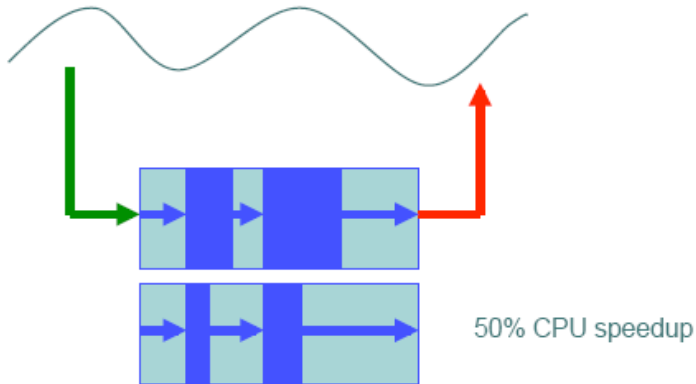
- Producteur : au moins un transfert de données pour chaque nouvelle valeur
- Consommateur : mise à jour de l'état local de la variable distante lors de la réception d'un nouveau transfert
- Possibilité d'y associer une gestion de l'historique des valeurs d'une variable (couple <valeur, temps>)

● Echange de données pour informer d'un changement d'état (event information)

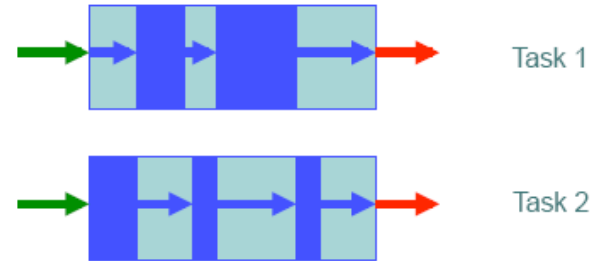
- Producteur : envoi d'un unique message
- Consommateur : stockage dans une file de messages et consommation selon le déroulement (cadencement) des traitements
- Notion de date de visibilité et de durée de péremption des messages

Système cadencé par le temps

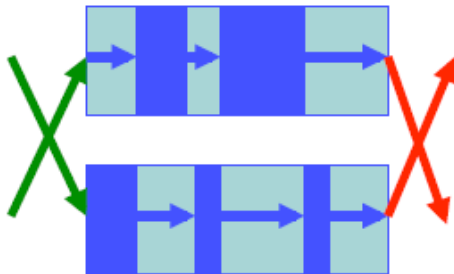
Portabilité



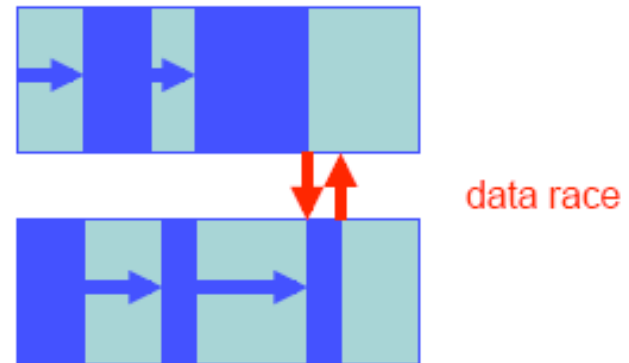
Composabilité



Déterminisme



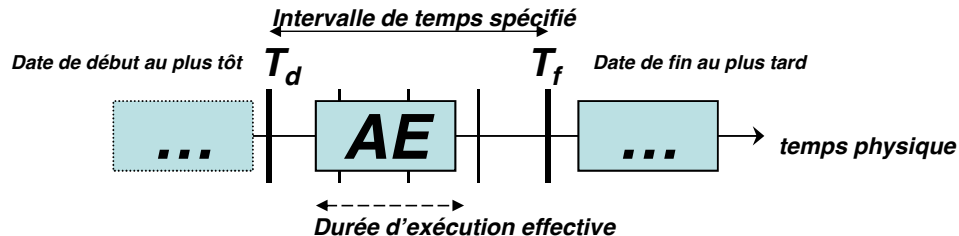
Différence avec les pratiques courantes



- Méthode complète pour la conception et l'exécution d'applications temps-réel sûres de fonctionnement
 - Application OASIS : ensemble d'agents (processus) définis statiquement
- Modèle de programmation : ΨC
 - Expression du parallélisme (définition des agents)
 - Expression de contraintes temporelles
 - Expression des communications
- Modèle d'exécution (multitâche communiquant)
 - Cadencé par le temps réel, multi-échelle
 - Cadencements déduits d'après les contraintes de temps exprimées et les communications
- Outils support permettant ...
 - ... la génération de code et l'implantation sur cible
 - ... l'aide à l'analyse et la validation (simulation POSIX)
- Noyau orienté sûreté de fonctionnement et performance

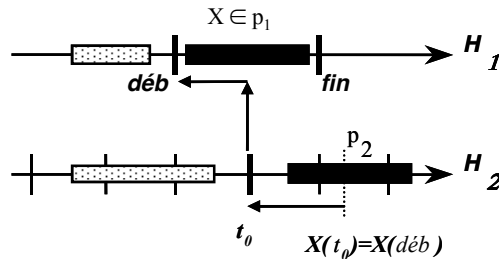
Modèle d'exécution : multi-échelle cadencé par le temps

- Le système est cadencé par le temps réel (Time-Triggered)
 - tous les traitements prennent place entre deux instants du système
 - les exécutions sont asynchrones mais sont synchronisées (synchronisme) au début et à la fin des traitements : isochronisme réel indépendant du matériel
 - Cela définit une action élémentaire (AE)
 - tous les transferts de données entre « tâches » ont lieu à la transition entre deux traitements



Modèle d'exécution

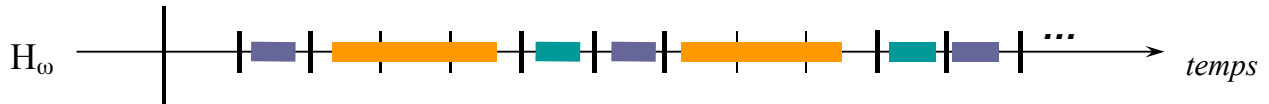
- Cohérence temporelle des échanges (liée aux mécanismes de communication OASIS)
 - encapsulation des données et des traitements (donnée protégée et un seul producteur)
 - les valeurs des données sur lesquelles travaille une AE sont figées à un instant temporel donné (date de début de l'AE)
 - Consommation et production de nouvelles données uniquement aux TSPs



- principe d'observabilité strict
- mise à jour implicite et automatique aux rythmes définis
- dimensionnement automatique et protection des tampons, optimaux et sûrs

Exemple d'écriture du ΨC : construction d'un cadencement simple

- Un agent avance dans le temps par sauts successifs le long d'une échelle de temps (horloge de base de la tâche) définie par le concepteur pour chaque agent
 - Les sauts peuvent être de longueurs différentes (comme ci-dessous 1-3-1-1-3-1-1-...)



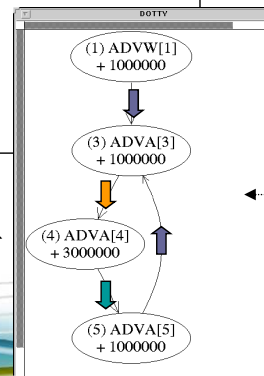
```
clock HBASE= gtc1(0, 1); /* 1 ms */
application demo(inittime= 99) with
HBASE;
agent agltst(starttime= 1) with HBASE
{
  body start
  {
    /* proc 1 */ advance(1);
    /* proc 2 */ advance(3);
    /* proc 3 */ advance(1);
  }
}
```

Initialisation TT :

La valeur initiale du temps TT de l'application est elle aussi prédéfinie par le concepteur

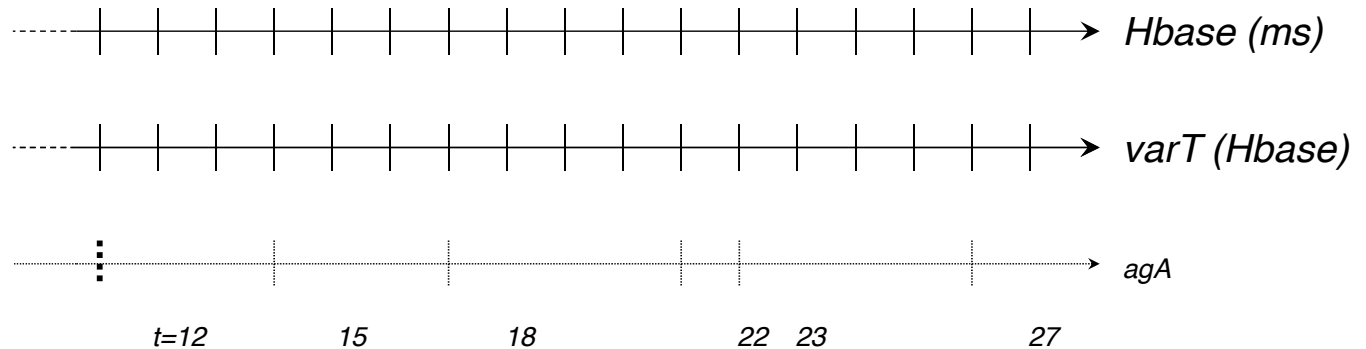
- Chaque tâche démarre à partir d'un instant spécifié de son horloge de base après le démarrage TT de l'application

Chaîne de compilation et de génération de code



Graphe de contrôle et d'exécution de la tâche utilisé par le noyau

Variables temporelles : définition et droits



/ Blocs chez le propriétaire (agA) */*

temporal

```
{
```

```
  unsigned long 1$varT=0 ;
```

```
}
```

display

```
{
```

```
  varT : agB ;
```

```
}
```

/ Bloc chez un consultant (agB) */*

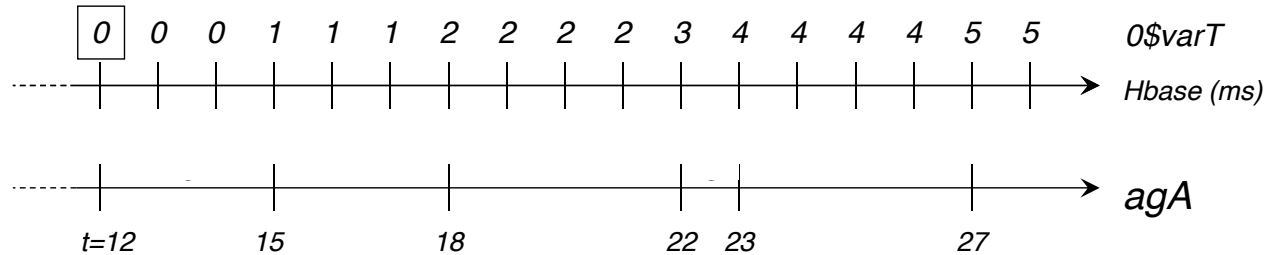
consult

```
{
```

```
  agA : 1$varT ;
```

```
}
```

Variables temporelles : accès aux valeurs passées (1/2)

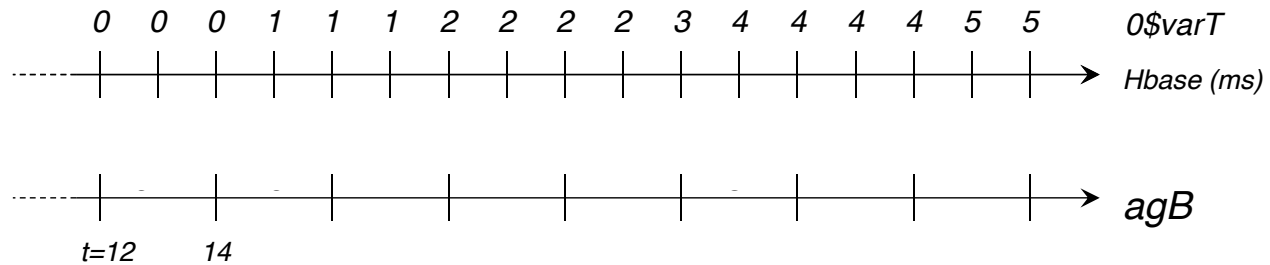


/ Code chez le propriétaire (agA) */*

```
{ varT=1; affiche(`0$varT`);  
  advance(3);  
  ++varT; affiche(`0$varT`);  
  advance(2) with Hpair;  
  ++varT; affiche(`0$varT`);  
  advance(2) with Hpair;  
  ++varT; affiche(`0$varT`);  
  advance(1);  
  ++varT; affiche(`0$varT`);  
  advance(4); /* ... */ }
```

Q : dessinez chronogramme avec 'temporal with Hpair' et le même code ΨC

Variables temporelles : accès aux valeurs passées (2/2)

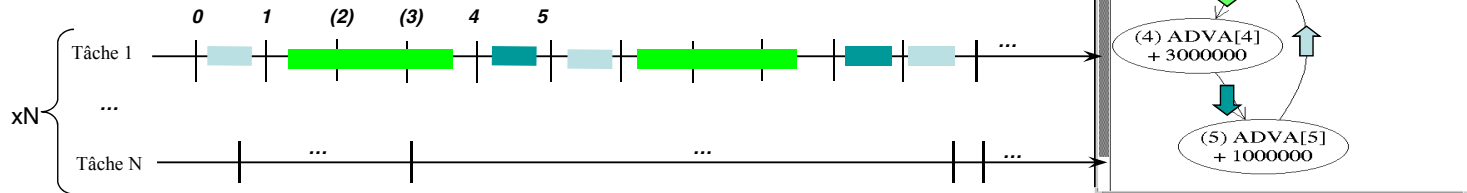


```
/* Code chez le consultant (agB) */  
→ { affiche(agA`0$varT) ;  
  advance(1) with Hpair ;  
  }
```

Q : redessinez ce chronogramme lorsque agB consulte les deux dernières valeurs passées de varT

Ordonnancement et dimensionnement

- Les valeurs des données sont indépendantes :
 - des durées d'exécution réelles, de la politique d'ordonnancement (EDF), de la fragmentation des AE
- Le comportement temps-réel est indépendant de l'ordonnancement
- Propriété de déterminisme des systèmes construits avec OASIS
- Une condition nécessaire et suffisante de dimensionnement
 - un surgraphe représente tous les comportements temporels possibles d'une tâche du point de vue des synchronisations temporelles



- la composition de ces surgraphes permet de connaître tous les intervalles temporels partagés par les traitements de chaque tâche
 - le produit synchronisé est l'opération mathématique de composition des surgraphes
 - le « graphe » obtenu contient TOUS les entrelacements possibles des traitements
- engendre un système d'équations et d'inéquations temporelles

=> Démonstration du dimensionnement

Construction simple d'une « synchronisation multi-échelle »

emacs: exemple_FRA_2.psy

File Edit Apps Options Buffers Tools C Help

```
clock MS = gtc1 ( 0, 1 ); /* horloge de période 1 millisecondes */
clock HA = 1*MS;          /* horloge de période 1 millisecondes */
clock HB = 5*HA;          /* horloge de période 5 millisecondes */

/* déclaration de la date de début de l'application */
application demo (inittime=1000*1000) with MS;

#include "simu_coupleur.h"

/* code source de l'agent AgDemo */
agent AgDemo (starttime=1 with HB) with HA
{
  body start
  {
    long indParc=0;
    long maxParc;

    maxParc=lec_donn();

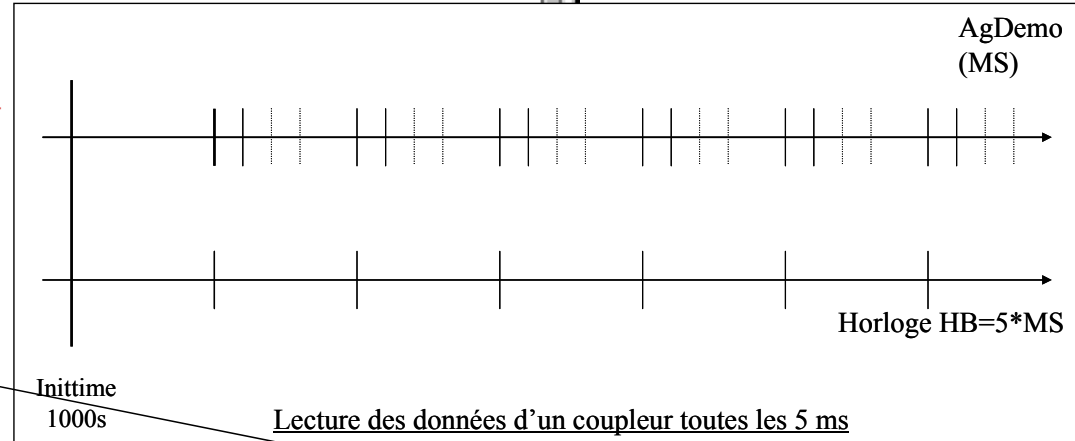
    do [3] {
      advance(1);
      indParc++;
    } while (indParc < maxParc);

    advance(1) with HB;
  }
}
```

Fontifying exemple_FRA_2.psy... done.

Déclaration de plusieurs horloges (échelles de temps) :

1ms et 5 ms



Resynchronisation au prochain tick de l'horloge à 5ms quelque soit le nombre de tours de boucle

Contrainte temporelle d'un tick de l'horloge à 1ms

Exemple simple de gestion de la gigue

- Affichage d'un tiret toutes les secondes avec une maîtrise de la gigue
 - Par construction le cadencement temporel implémenté garantit une gigue entre deux occurrences d'affichage de [750;1250ms] (si système ordonnançable)

```
#define SEC_in_ms 1000
#define JIT_in_ms 250

clock MS= gtc1(0,1); /* 1ms */
clock HSEC= SEC_in_ms*MS;
clock HJIT= SEC_in_ms*MS + JIT_in_ms;

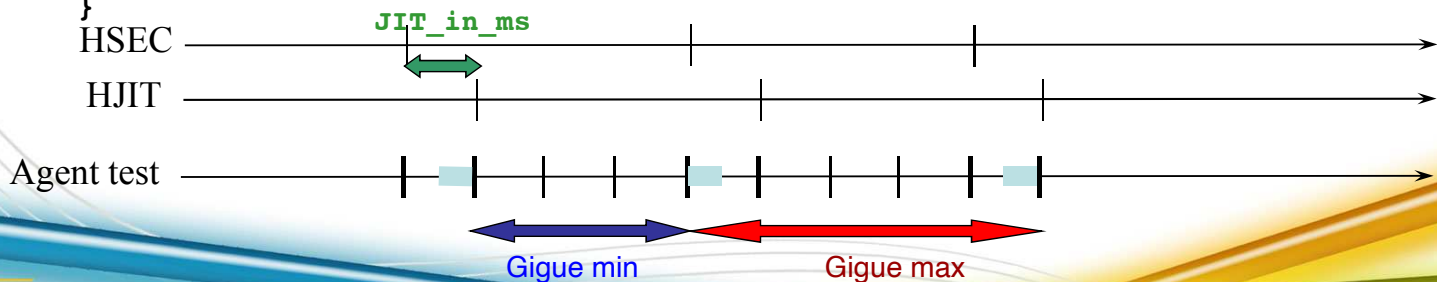
application dash(inittime= 10) with HSEC;

agent test(starttime= 1) with HSEC {
  body start {
    edit("-"); advance(1) with HJIT;
    advance(1);
  }
}
```

Paramètre du
comportement temporel

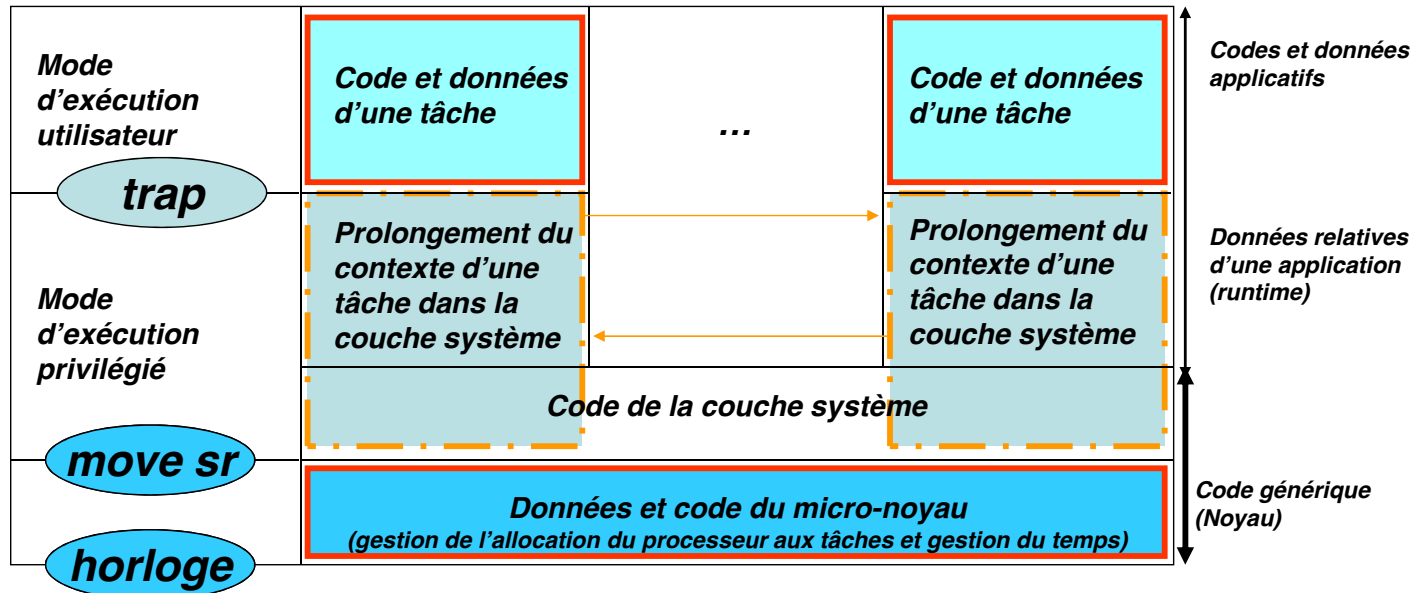
Déclaration des horloges

Maîtrise de la gigue



- Un système contient *toujours* des défauts résiduels
- Principe de déterminisme fort dans OASIS
 - Confiner les anomalies et maîtriser leurs conséquences pour **conserver la propriété de déterminisme même en cas d'anomalies**
 - il faut
 - détecter à l'exécution toutes les tentatives de violation de la séparation des tâches
 - contrôler le dimensionnement des ressources d'exécution (tampons, quotas CPU, etc.)
 - contrôler l'enchaînement séquentiel des actions, etc.
- Partitionnement spatial et temporel au sein d'OASIS
 - Unité d'exécution et de confinement d'anomalie : l'agent
 - Isolation spatiale (illustré sur le transparent suivant)
 - entre les différents agents
 - entre les différentes parties du noyau
 - entre le noyau et les agents
 - Isolation temporelle
 - Surveillance des budgets (WCET) associés aux actions élémentaires (AE) : ségrégation de la défaillance au sein de la tâche et maintien de disponibilité du système
Impact limité à l'agent, possibilité de définir une stratégie d'exécution en mode dégradé
 - Surveillance des échéances des AE : détection d'une erreur de dimensionnement

Organisation défensive des programmes exécutables



- contextes d'exécution distincts hors de CS et différents internes à CS, contexte μ N indépendant
- segmentation particulière des exécutables et des données
 - conception orienté sûreté et exploitation spécifique des mécanismes de protection liés au H/W
- contrôle de conformité de l'exécution de chaque tâche
- contrôle toutes les hypothèses de dimensionnement (quotas, tampons, etc.)
- les préemptions de tâches sont toutes contrôlées par le μ Noyau atomique
- CS 100% préemptible

- Concepts de base des systèmes cadencés par le temps réel
 - Comparaison avec une approche par évènement
 - Présentation du fonctionnement
 - Illustration via un aperçu de Giotto et OASIS
- Extension aux systèmes distribuées
 - Problématiques
 - Aperçu de TTP & OASIS-D
- Modèle de tâche

■ Nécessité d'un temps physique unique

- Nécessité de mettre en place des mécanismes de synchronisation afin de maintenir globalement cohérent l'ensemble des horloges locales
- Sinon, incohérence sur la datation des évènements et perte de la relation d'ordre total entre les évènements

■ L'utilisation du réseau est également cadencé par le temps réel

- Si pas de temps physique global : conflits potentiels sur l'accès au réseau pour la transmission de données
- Nécessité de définir une politique d'accès / partage du réseau basé sur le temps pour garantir des bornes sur les temps de transferts
 - Approches Time-Division Multiple Access (TDMA)
- Composants standards (Components Off-The-Shelf - COTS)
 - diffusion à grande échelle, robustesse de certains composants
 - pas entièrement conforme avec les exigences (effort d'intégration et d'adaptation)

L'approche TTP

• Nœud TTP

- Communications uniquement via une interface mémoire particulière CNI

• Contrôleur TTP

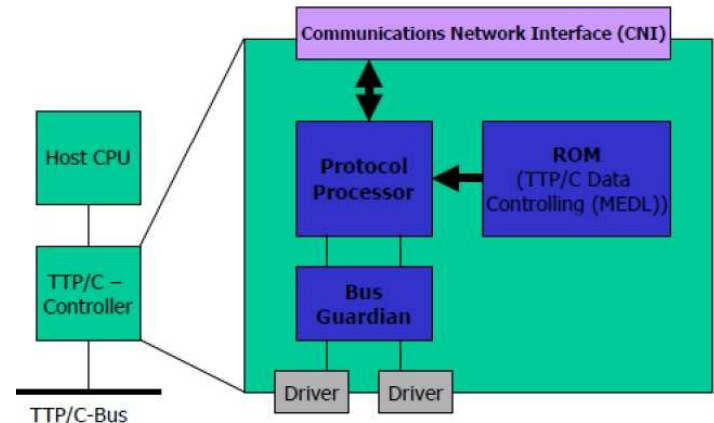
- Synchronisé via une table de description des message (MEDL)
 - Description des instants d'émission/réception, des adresses CNI
- Fonctionnement indépendant du noeud

• TTP Frames

- I-frames (synchronisation)
- C-state (diffusion de l'état courant : temps, position MEDL, mode d'opération)
- N-frame (données applicatives)

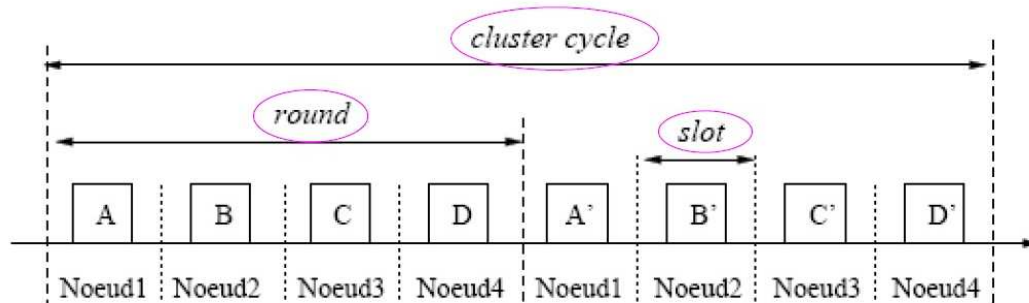
• Gardien de bus

• Redondance des liens de communication (gérée de manière transparente)



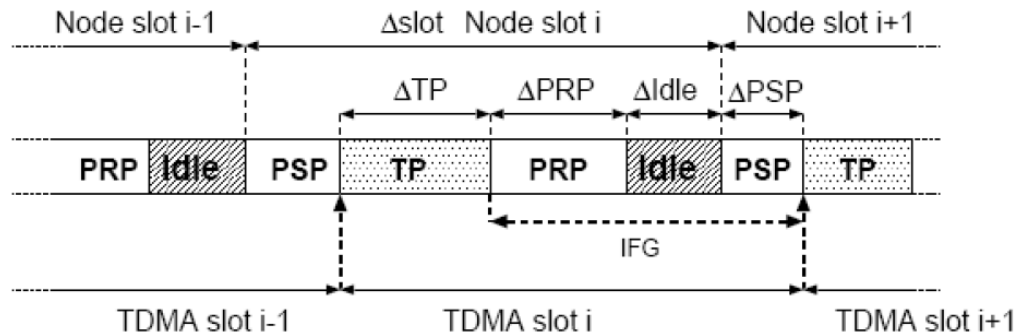
Stratégie d'accès au réseau

- « TTP cluster cycle » : définition de l'ordonnancement des accès réseau
 - Un cycle contient plusieurs « round » TDMA
 - Une « round » TDMA contient plusieurs slots
 - Le message envoyé par un nœud dans son slot TDMA varie dans les différentes « round » TDMA
 - Un slot TDMA peut être multiplexé (e.g., utilisé par différents nœuds mais dans des « round » TDMA distinctes)



- Topologie : bus ou étoile

Stratégie d'accès au réseau



- **PSP – Pre-Send Phase** : exécution des algorithmes liés à l'émission d'une trame (ne concerne que l'émetteur du slot en cours)
- **TP – Transmission Phase** : transmission effective
- **PRP – Post-Receive Phase** : exécution des algorithmes liés à la réception d'une trame (concerne tous les nœuds)
- **Idle Phase** : possibilité de configurer des durées de slot + longue que nécessaire à priori

- Continuité et intégration avec le modèle OASIS
 - intègre déjà de manière réaliste les contraintes temporelles
 - délais de communication jamais estimés comme nuls
 - diffusion avant l'échéance, approvisionnement après l'échéance
 - couches systèmes et matérielles transparentes pour le développeur
 - application conçue indépendamment de l'architecture
- Interface TDMA avec Ethernet standard (physical layer)
 - aucune collision par construction (une collision devient une erreur)
 - garantir par construction la ponctualité des communications
 - dimensionnement et ordonnancement statique du réseau
 - pour vérifier et valider le réseau hors-ligne
 - pour détecter un comportement fautif

=> Cadencement global des communications connu de tous

Concept de communications intégrées

- **Préserver la couche système et le μ Noyau**

- maintenir le niveau de sûreté atteint
- disposer d'une solution modulaire

- **Concept d'ombre**

- clone mémoire du producteur
- localisée sur chaque CPU consommateur

- **Concept de pilote**

- tâche TT de la couche système
- contrôle les préparations et réceptions

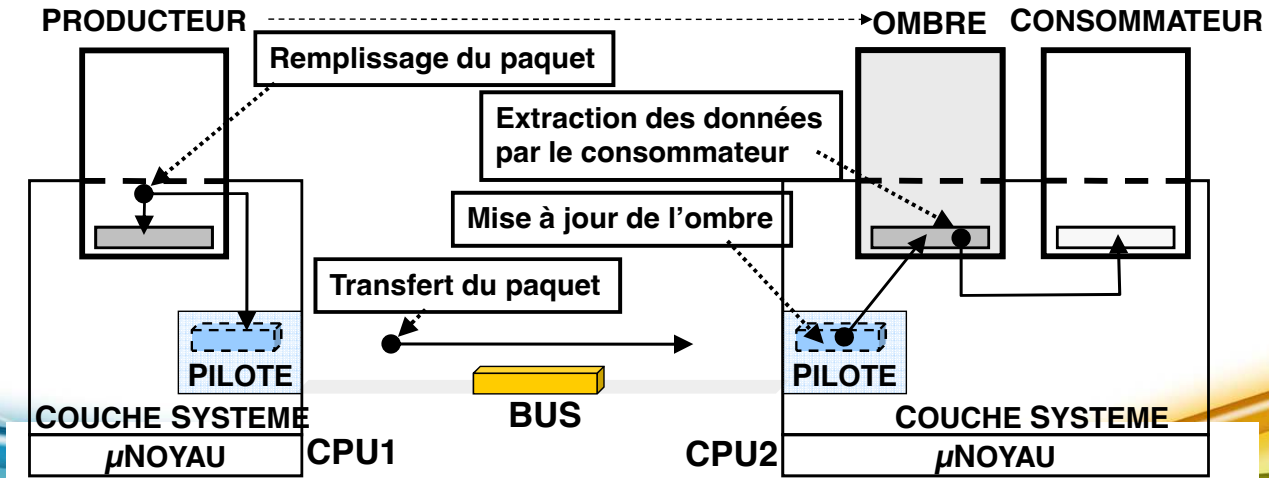
- **Cadencement du réseau**

- par construction, un seul CPU occupe le média de communication à un instant donné

- **Optimisation possible (A-TDMA)**

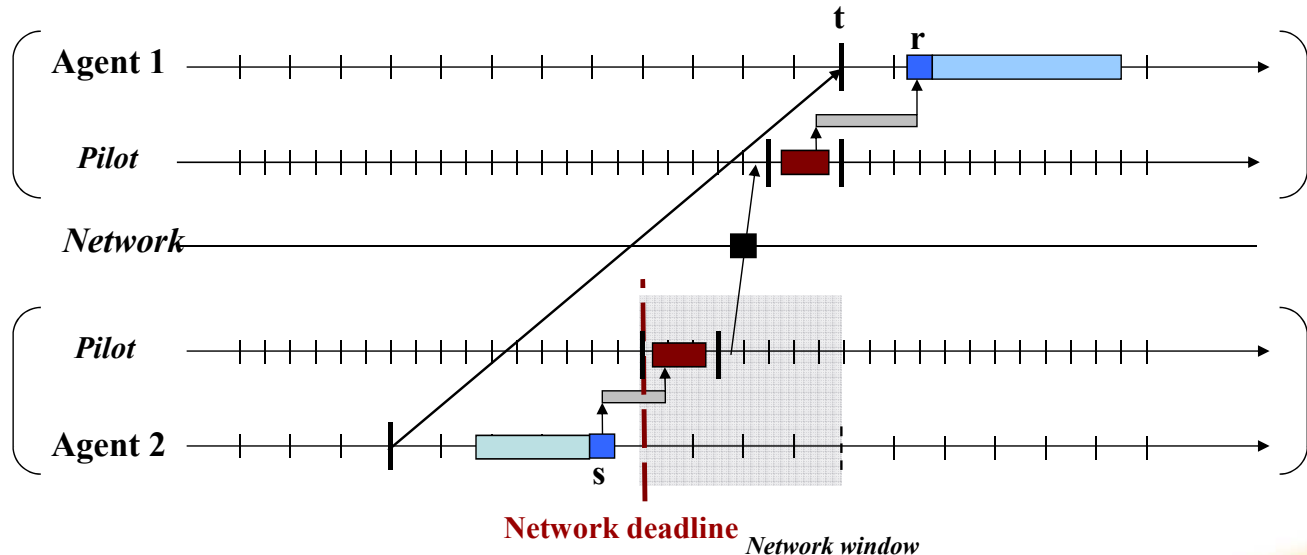
- utilisation du mécanisme de détection de porteuse
- « chevauchement » de plages par construction
 - seulement entre deux émetteurs consécutifs
 - inversion de paquets impossible

↪ espace entre deux paquets consécutifs optimal



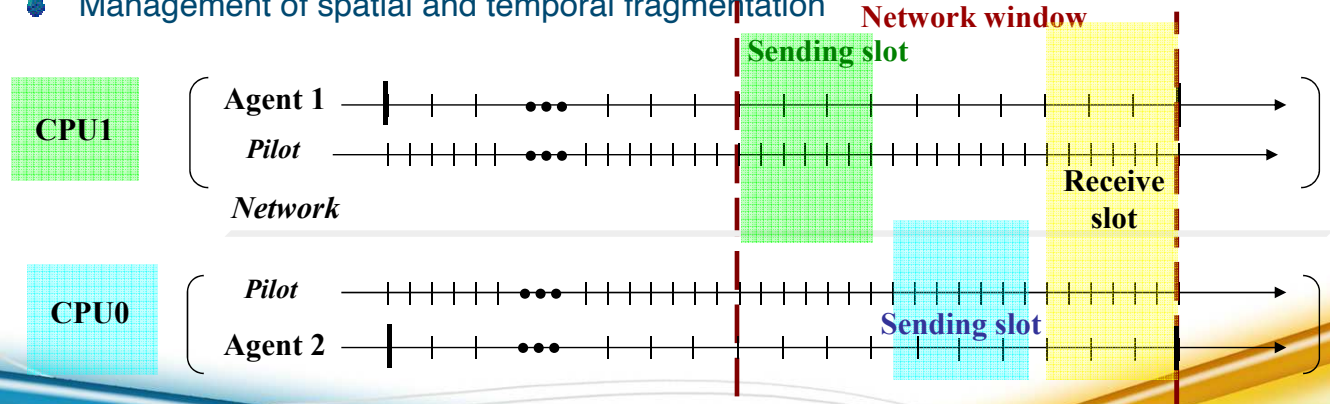
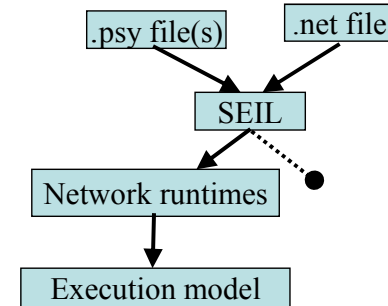
● L'échéance réseau

- Prise en compte des délais de transmission
- Surcontraindre l'échéance d'un traitement
- Libération d'une fenêtre de transmission des données



Off-line generation – network cycle

- The network cycle is computed off-line for each application
- Sizing & scheduling of TDMA slots
 - Depends on agent network needs : possible emission time & letters size
 - Network capacities & MTU (spatial fragmentation)
- Approach: build a linear system of equations & inequations to cope with the constraints of the system
 - Extract constraints from ψC and the network capacities
 - Construct network requirements for each agent
 - Synchronous product in order to obtain the network cycle
 - Generate network runtimes if the system has a solution
 - Goal: minimize the overhead on the CPU
- Management of spatial and temporal fragmentation



- Concepts de base des systèmes cadencés par le temps réel
 - Comparaison avec une approche par évènement
 - Présentation du fonctionnement
 - Illustration via un aperçu de Giotto et OASIS
- Extension aux systèmes distribuées
 - Problématiques
 - Aperçu de TTP & OASIS-D
- Modèle de tâche

- **Modèles de tâches classiques : périodique, sporadique, etc.**

- Faible expressivité du comportement d'une tâche :
 - Une tâche n'est pas forcément cyclique
 - Modes dégradés, plusieurs phases, etc.
- Forte contrainte sur la conception de système temps-réel
 - Difficultés de conception, d'implémentation et d'analyse (notamment de jitter par exemple)
 - Variation entre les exécutions successives d'une tâche
 - Analyse à posteriori du jitter une fois les temps d'exécution connu et la conception du système réalisé

- **Proposition de modèle de tâche basé sur des automates temporels (timed-constrained automata)**

- Forte expressivité
- Expressing du jitter maximal possible directement dans le modèle

Time-Constrained Automata (TCA) applied to tasks: task-TCA

● Notions

- Block: sequence of instructions represented by an arc
- Nodes: separate two blocks

● Specify temporal constraints on a block

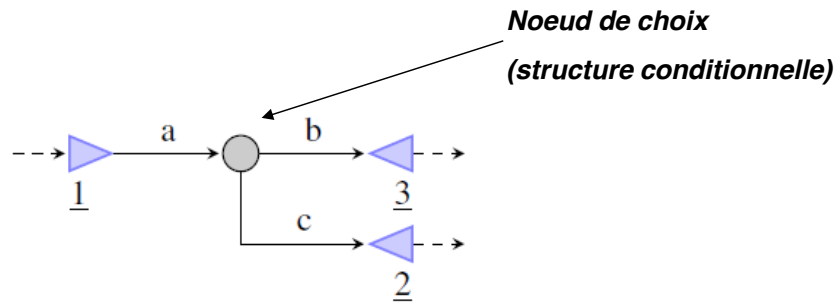
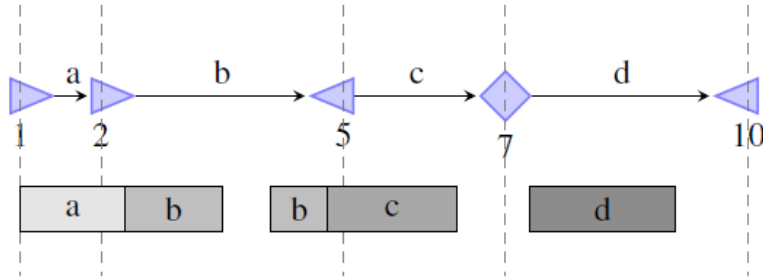
- Make it start after a certain date: after node
 - AFTER(D) primitive
- Make it end before another date: before node
 - BEFORE(D) primitive
- After node + before node: synchronization node
 - ADVANCE(D) primitive




● Execution is correct


- Successive jobs of a task execute in order
- The execution intervals for each job fulfill the timing constraints of all nodes


Exemples (chaînes et arbres)



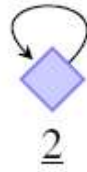
– Légende

 La/les tâche indiquées sur la transition suivante peuvent commencer à s'exécuter

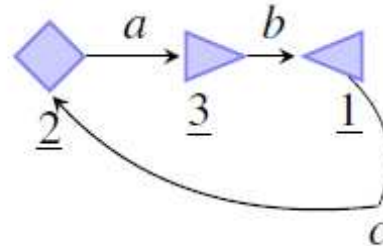
 La/les tâche indiquées sur la transition précédente doivent avoir fini de s'exécuter

 Combinaison des deux cas précédents

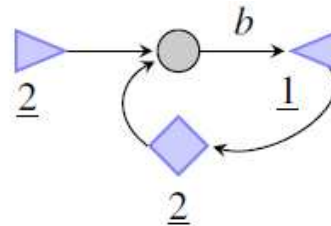
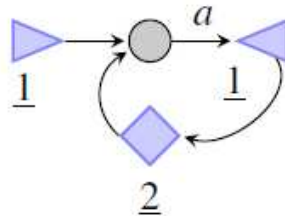
- Exemples (automates)



(a) Periodic task of period 2 with relative deadline equal to the period.



(b) A periodic task (period 5). b is constrained with fine-grained jitter specification (maximum jitter 1).



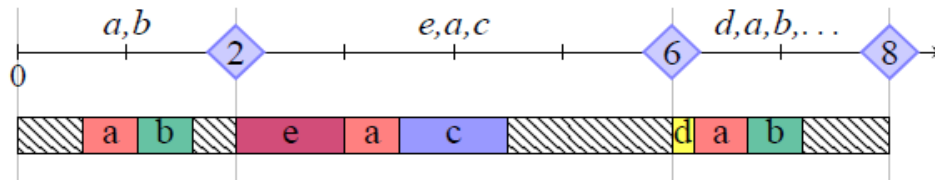
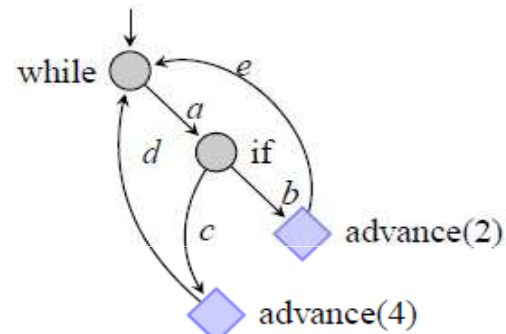
(c) Two periodic tasks with period 2 and relative deadline 1, with respective phase 1 and 2. Time constraints put a and b in mutual exclusion: a can execute every $[2 * k + 1, 2 * k + 2[$ and b every $[2 * k, 2 * k + 1[$.

Time-Constrained Automata (TCA) applied to tasks: task-TCA

- Simple example

- Used to illustrate the computation network requirements

```
while(1) {a:  
  if(...) {b:  
    advance(2);  
  e:  
  else {c:  
    advance(4);  
  d:  
}
```



Overview of communication mechanisms

• Visibility date V_d

- Date after which the data can potentially be consumed by other tasks
 - The last synchronization point of a consumer must be equal or beyond
- Already visible data can not be modified
- Communication latency between task is never null
 - Can only be in the future compared to the current date of the producing task (last \diamond_d node of the task)

• Two communication mechanisms




- Temporal variables: 1-to-n regular data flow of values
 - Have its own periodicity
 - No specific primitive
 - New values are made visible at every \diamond_d node ($V_d = d$)
- Messages: a n-to-1 irregular exchanges of data
 - Explicitly define V_d
 - SEND primitive

Applying TCA to model network behavior of tasks

• TCA used to model data exchanges between remote tasks

- Network-TCA automata
- Express timing constraints on the visibility dates
 - Of the data items owned by a task, i.e. the producer
- Model the sending, the transfer and the reception cost

• Semantic of nodes of the TCA model

- Constrains the exchange of data to start after a given date:  *d*
- Constrains a data item to be available before a given date:  *d*
- Combination of both:  *d*

• Execution is correct

- Successive data exchanges of a task execute in order
- The execution intervals of each data exchange fulfill the timing constraints of all nodes

```
while(1) {a:
  if(...) {b:
    send(Ta, v, 4);
    advance(2);
  e:
  else {c:
    send(Tb, y, 2);
    advance(4);
  d:
}
```

Computation of network-TCA

Temporal variables

- At each \diamond_d node of task-TCA is associated a \diamond_d node in the network-TCA

Messages: more complex

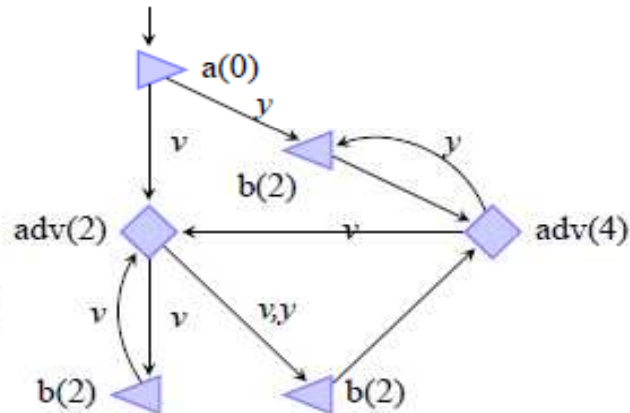
- Transfer of a message can start as soon as the SEND primitive
 - Unnecessary to express such type of constraints in TCA: implied by synchronization point
- Data item must be made available no later than the specified
 - A \triangleleft node is added to translate this constraint
 - Different cases depending on V_d
- If V_d is beyond the next \diamond_d of the task-TCA
 - Availability of the data can be postponed later than the deadline of the job
 - Sum of the next \diamond_d nodes are subtracted from V_d
 - Maintain timing constraints consistent between them V_d
 - \triangleleft nodes must be added in all execution paths that can be reached in the time interval between d and V_d

Example of network-TCA

Extension

- Arcs are labeled with the list of messages that must be made available to consumer tasks

```
while(1) {a:
  if(...) {b:
    send(Ta, v, 4);
    advance(2);
  e:
  else {c:
    send(Tb, y, 2);
    advance(4);
  d:
}
```



Application dependent optimizations

● Producer side: apply on temporal variables only

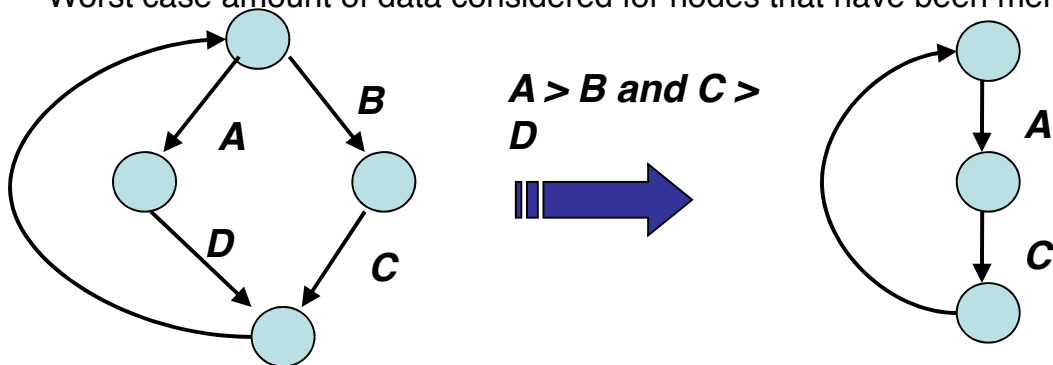
- Period of a temporal variable $n_i D_{tmp}$ necessarily coupled with the temporal constraints of the producer task
- Case : $D_{tmp} \geq \delta t$ (time interval between two synchronizations)
 - Unnecessary to perform a data exchange before the next synchronization point
 - Can be postponed up to the next period of the temporal variable
 - \diamond_d node modified into a \triangleleft_d node

● Consumer side: apply for both communication mechanisms

- Visibility date: does not specify when the consumer tasks actually use the data
 - Depends on the temporal behavior of the consumer tasks
 - Must have reached a synchronization point
- Data exchange can be postponed for the producer to the closest \diamond_{dm} node in all the consumer tasks
 - \triangleleft_d or \diamond_d modified into \triangleleft_{dm}

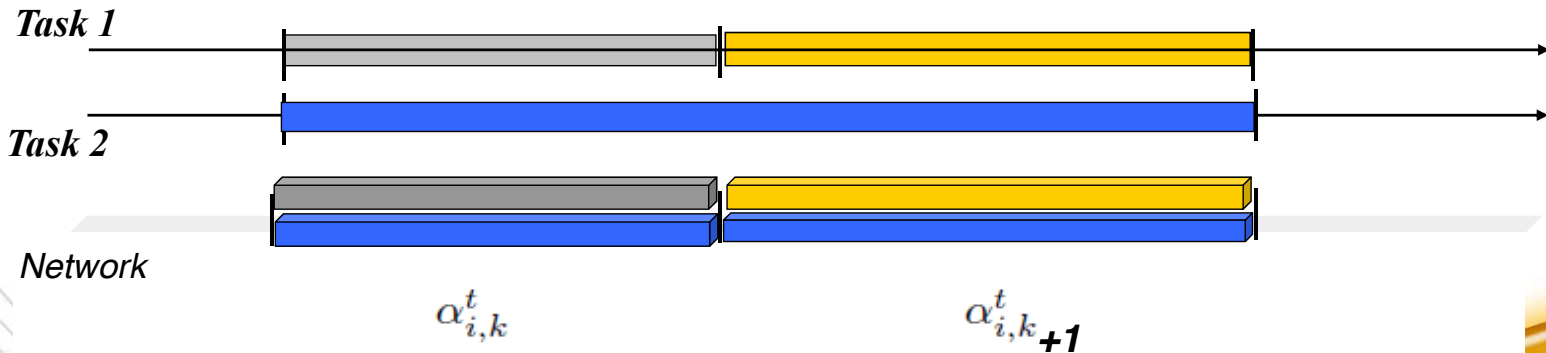
A single execution path: network-TCA*

- Execution of an application: choosing one (possibly infinite) walk in the network-TCA of each task
 - In each execution path, the scheduling can be different
 - Sizing issues for the both the network and the memory
 - The chosen execution path must be made globally known
 - Encode all possible execution paths
- Network-TCA*
 - By construction a network-TCA* satisfies the network-TCA of a task, whatever will be its dynamic behavior
 - The length of the container is independent of the length of the content
 - One-line verification of the sizing is made easier
 - Construction of a network-TCA*
 - Before nodes can be merged with synchronization points
 - Worst case amount of data considered for nodes that have been merged



Network cycle graph

- Product of the network-TCA*
 - Translation into a linear programming problem
- Temporal fragmentation
 - Division in several subintervals an initial timing-constraint
 - Due to the different temporal behavior within a set of network-TCA*



Constraints and objective function

- Integrity of data flows

$$\sum_{j=1}^{p_i} \alpha_{i,j}^t = 1 \quad 0 \leq \alpha_{i,j}^t \leq 1$$

- Temporal inequations

- Network deadline: fraction of time that is reserved for performing data exchange

$$h(Df_w) < ndl_w(d_w - rcv)$$

- Function that computes the required time to transmit a set of packets

- User configurable limit on the network deadline

- Maximum pourcentage of time interval that can be used for the network scheduling

$$W, 0 \leq ndl_w \leq \gamma$$

- Objectif function: minimize the impact on the CPU

- Minimize the network deadlines
- Minimization of effective temporal fragmentation

$$\min \left(\sum_{w \in W} ndl_w - \sum_{t \in Nt} \sum_{i \in V_t} \sum_{j=1}^{p_i} (j * \alpha_{i,j}^t) \right)$$