Izarra Green-M^2

Green Middleware & Methodology

Philippe ROOSE

LIUPPA / T2I – Université de Pau et des Pays de l'Adour

GreenDays@Toulouse 2018

Nous ne parlerons pas de …

~~Cloud Computing~~

~~BigData~~

~~Apprentissage~~

~~Réseaux de neurones~~

~~Deep Learning~~

~~NoSQL~~

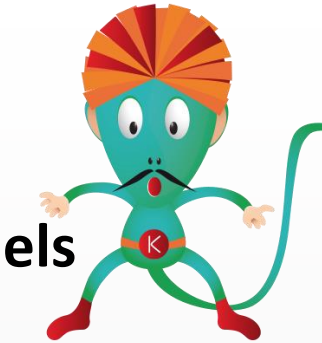
~~Privacy~~

~~Algorithmes génétiques~~

~~EcoSystèmes Numériques~~

~~Masses de données~~

La base de tout...Le Middleware - KALIMUCHO



- **Plateforme (à service) pour applications pervasives à base de composants logiciels**

- **Applications Dynamiques [re-]déploiement sur périphériques mobiles** (smartphones, tablet, PC, etc.).

- **Reconfigurations à chaud** : ie. Reconfiguration sans stopper l'application
  - Quelque soit la raison
  - En fonctions du contexte : fonctionnels, énergétiques, matériel, utilisateur.
  - Points d'adaptations possibles en général : paramètres, fonctions, code/contraintes, objets, composants, assemblages, etc.)

- **Transfert d'informations entre composants logiciels**
  - Avec la gestion automatique de passerelles (Ethernet, Wifi, 3G)

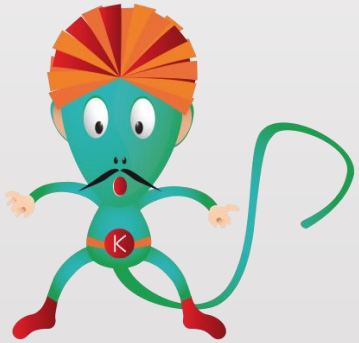
KALIMUCHO

- Permet également de…

- **Réaliser des installations instantanées et temporaires** (short-lived Installation/Deployment) sur des périphériques.
  - Lorsque l'application est fermée, les composants déployés sont détruits (ou gardés en cache).

- **Accéder à des applications non résidentes**

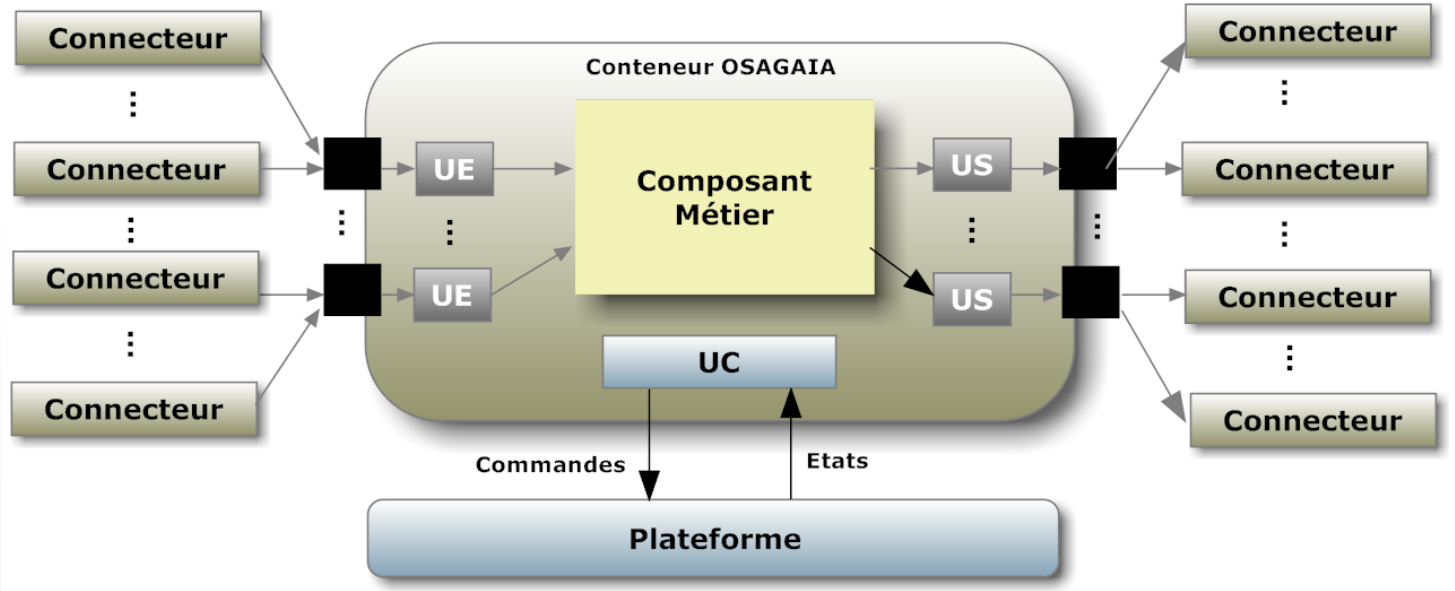
- **Installations et déploiements ad'hoc/contextualisé selon les besoins du moment**
  - Sans passer par une opération guidée par l'utilisateur
  - Sans « [android-]market » ou « any app-store »



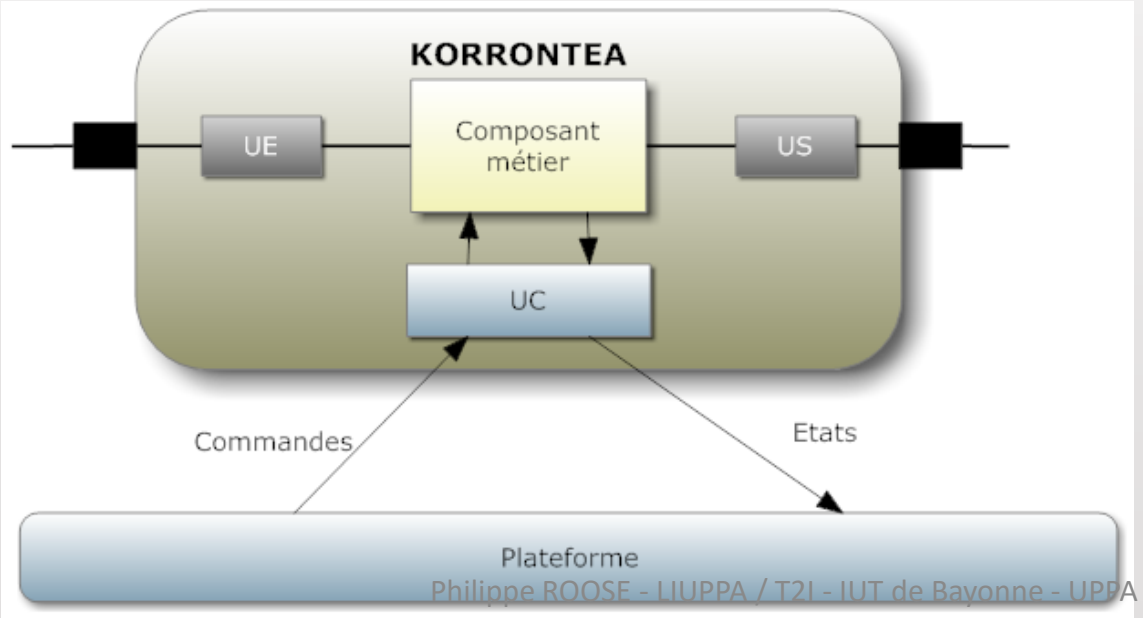
Comparatifs de middleware

	OSGI	CADeComp	FRASCATI	Kalimucho
1 Modèle de composants	**Bundle**	CCM	**Composants fractal**	Osagaia Framework
2 Lien entre les composants	Services Packages	**Ports**	Connexion explicite	Korrontea framework (explicite mais dynamique)
3 Déploiement contextuels	Non	**Oui**	**Oui**	Oui
4 Context-Aware	Non	Non	**Oui**	Oui
5 Reconfiguration	**Ajouter Supprimer**	**Ajouter Supprimer**	**Ajouter Supprimer** Auto Réparation Auto Protection	Ajouter Supprimer Remplacement **Migration**
6 Dynamisme d'exécution	Non	Non	**Oui**	Oui

Kalimucho



- Modèles (de première classe) de composants/connecteurs



KALIMUCHO - DSL

- Commandes de création, suppression, migration, connexion, déconnexion et duplication des flux de sortie des composants/connecteurs.

CreateComponent nomc classe [entrée1 entrée2 …] [sortie1 sortie2 …]

Les listes d'entrée et/ou de sortie peuvent être vides ([null])

Une entrée ou une sortie peut être marquée "not_used" pour être utilisée plus tard

RemoveComponent nomc

SendComponent nomc vers

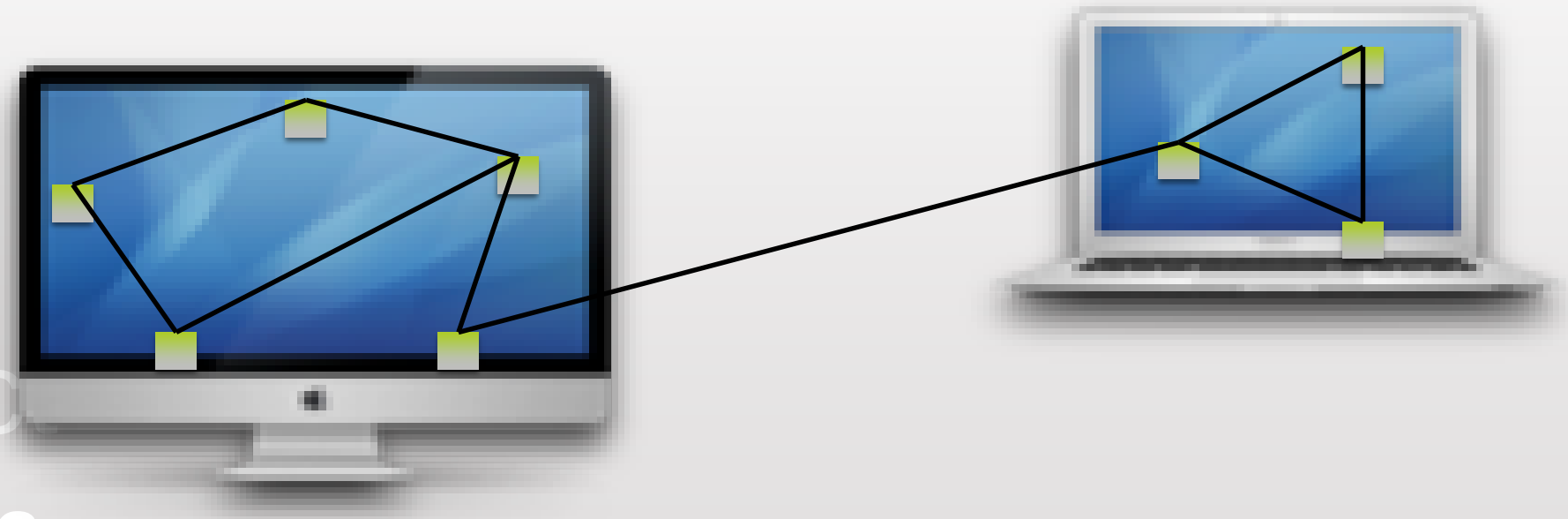
DisconnectInputComponent nomc numéro_d_entrée

DisconnectOutputComponent nomc numéro_de_sortie

ReconnectInputComponent nomc numéro_d_entrée nouvelle_entrée

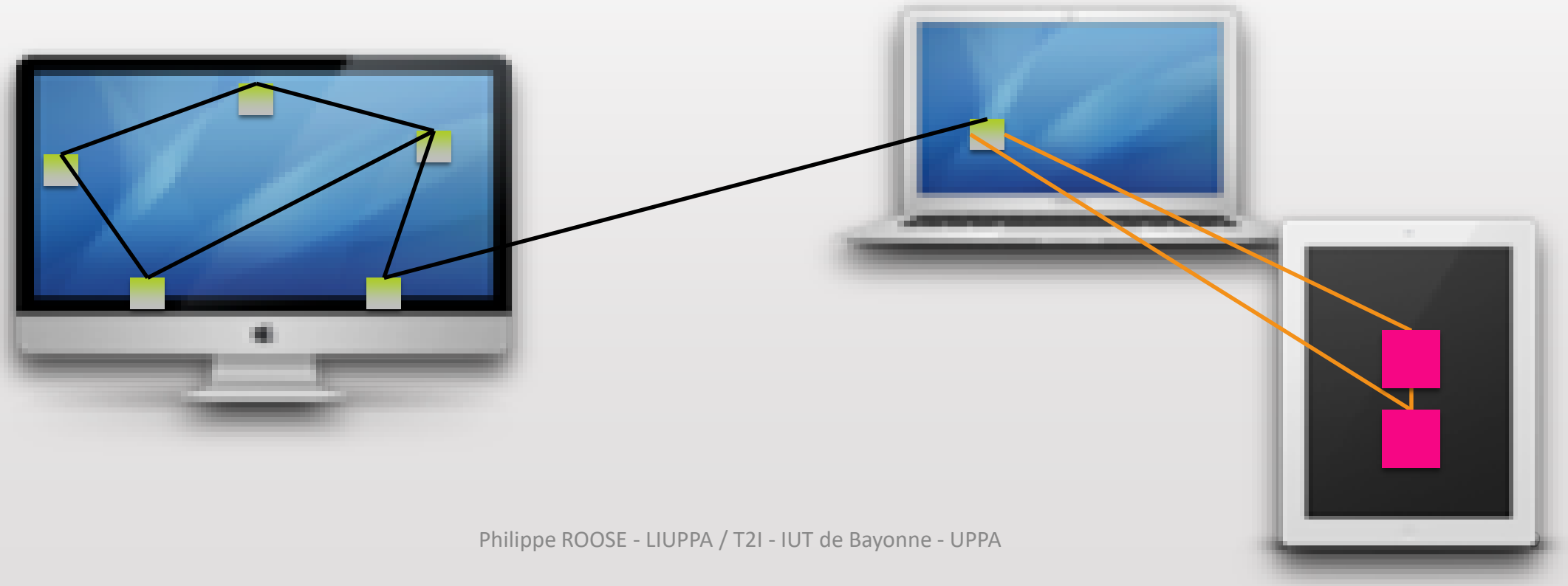
DuplicateOutPutComponent nomc numéro_de_sortie nouvelle_sortie

Kalimucho – adaptation structurelles

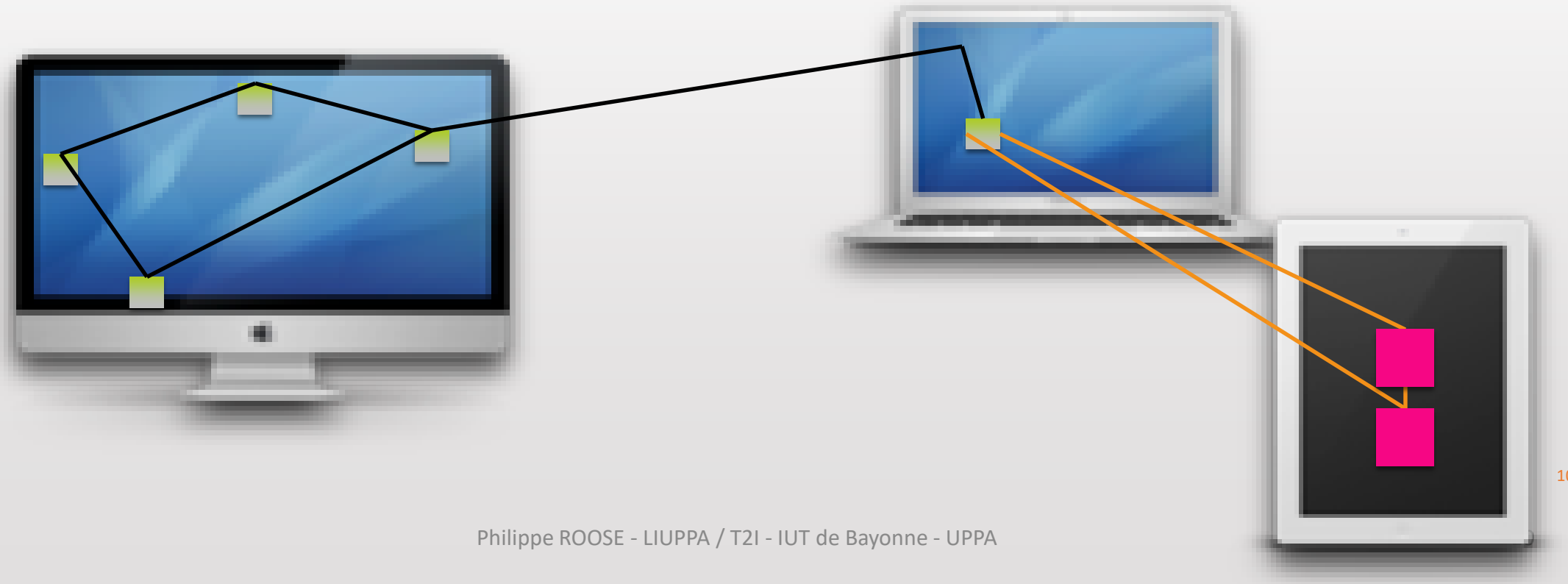


Ac
On-
The-

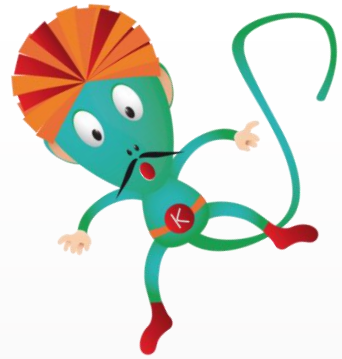
Kalimucho – adaptation structurelles



Kalimucho – adaptation structurelles



Kalimucho – en chiffres



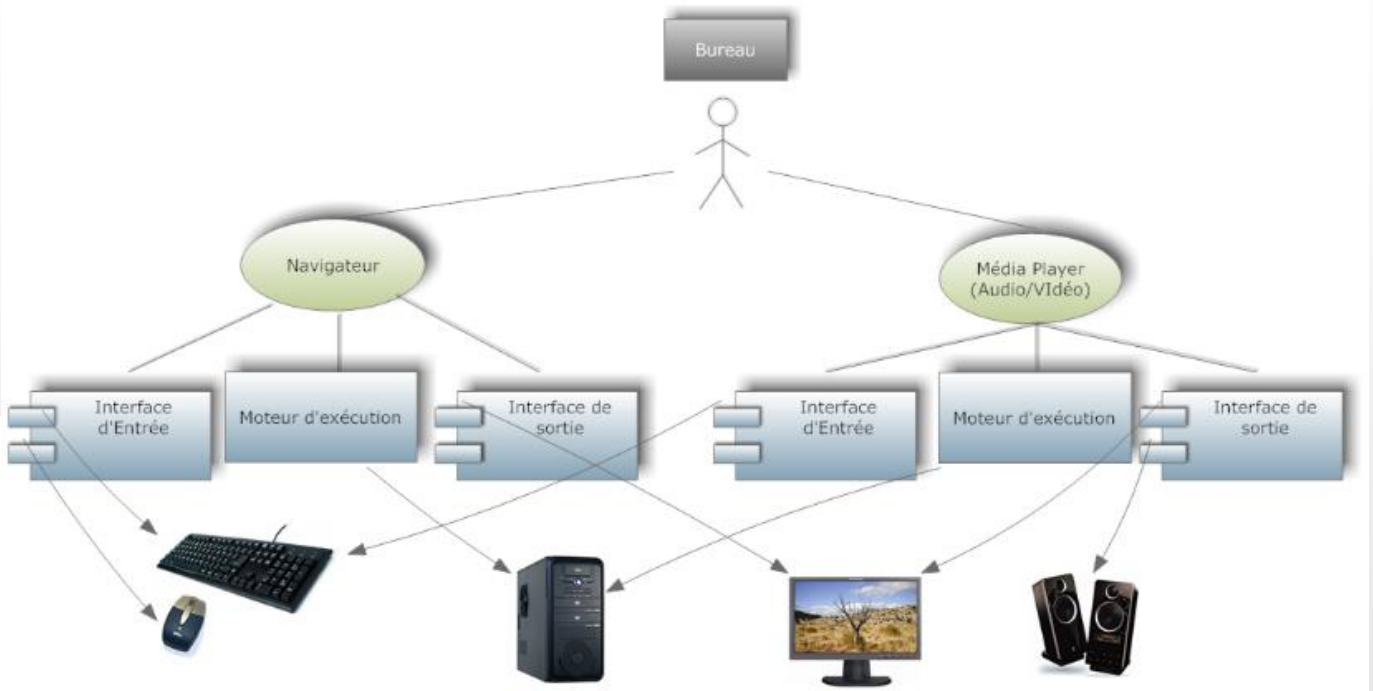
- **Taille de la plateforme**
  - PC (JAR) // Android(APK) < 1Mo

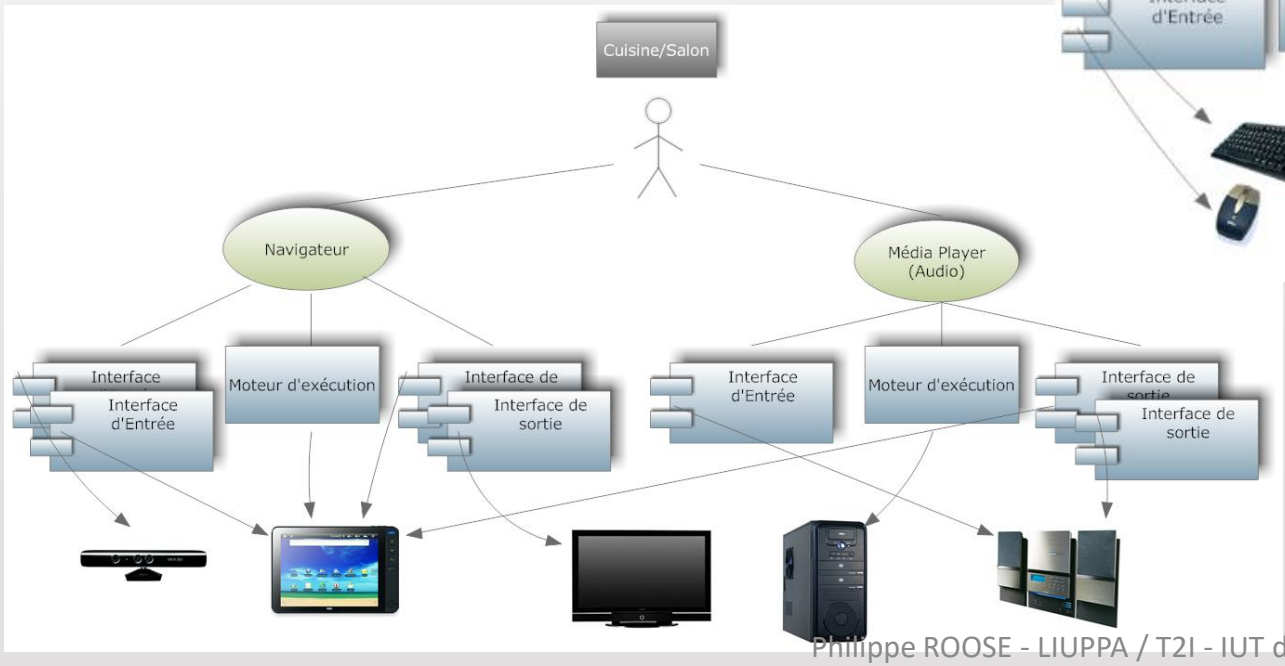
- **Temps d'exécution de commandes (sur Android Nexus One)**
  - Création composant: 2 à 20 ms
  - Suppression composant : 15 ms (à partir de la fin de la méthode stop)
  - Création d'un connecteur interne : 3 à 15 ms
  - Création d'un connecteur distribué: 10 à 100 ms
  - Suppression d'un connecteur interne: 3 à 15 ms
  - Suppression d'un connecteur distribué: 3 à 25 ms
  - Déconnexion/Reconnexion d'une entrée: 2 à 7 ms
  - Duplication d'une sortie: 2 à 7 ms

- **3 brevets, 1 marque déposée, Présentations CES Las Vegas, etc.**
- **www.kalimucho.com**

Mesures de Kalimucho	PC	Android
Taille	827 Ko	1032 Ko
Nombre de lignes de code	17 000	20 000
Nombre de classes	178	262
Nombre de threads lancés au démarrage	20	21
Nombre de méthodes ou de blocs "synchronized"	279	244
Nombre d'opérations "wait" et "notify"	87	72

Un peu d'interactivité…

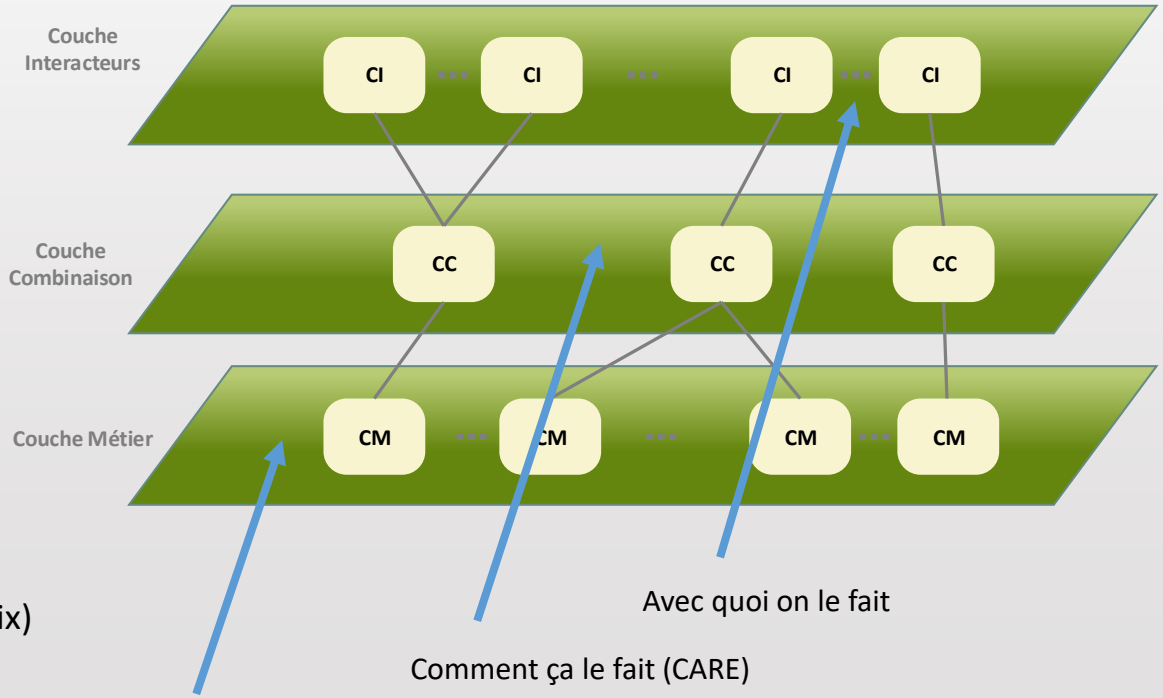




Définitions

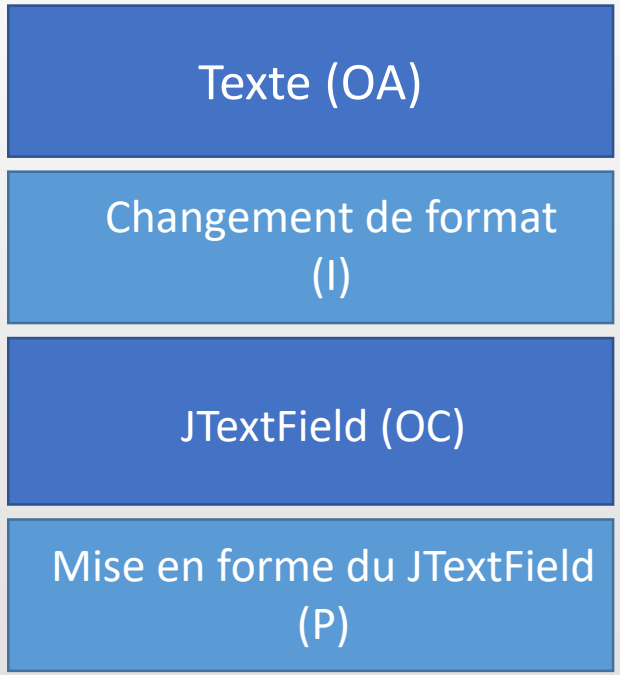
- **Modalité d'interaction** = Un couple (périphérique ; langage d'interaction)

- **Multimodalité** = Possibilité de combiner plusieurs modalités différentes
- **4 propriétés (CARE)**
  - **Complémentarité**
    - Plusieurs modalités se complètent obligatoirement
  - **Assignation**
    - Pas de choix (on assigne une modalité à l'interaction)
  - **Equivalence**
    - Plusieurs modalités équivalentes : choix
  - **Redondance**
    - Plusieurs modalités équivalentes : addition obligatoire (pas de choix)



Modèle d'interacteur : **txupito**

- Txupito est divisé en quatre parties
- Deux composants
  - Physique (P) // Interpréteur (I)
- Deux connecteurs
  - Objet Abstrait (OA) // Objet Concret (OC)
- Txupito offre plusieurs comportements
  - Changement de sémantique
  - Changement de modalité
  - Assemblage d'interacteur



Chaque « boite » =
1 composant Kalimucho

Green IT ~~écosystème~~

- **IT**
  - Fabrication / Recyclage
  - Electricité (consommation, €, etc.)
  - Datacenters (consommation, €, etc.)
  - …
- **Computing**
  - Datacenters (agrégation, sélection des plus proches, des plus ‘verts’, etc.)
  - Minimisation passerelles réseau
  - …
- **Applicatifs**
  - Middleware, Fog/Edge
  - Applications, Contexte
  - Interactions
  - SI
  - …

Très peu de choses…très très peu !
Uniquement du « à posteriori » ou du ad’hoc (exmple Société Greenspector)

Pas d’approche « globale »

Quels (autres) leviers pour économiser ?

- Mobilité
  - En **déplaçant/remplaçant** des traitements, on peut optimiser la consommation

    - **CPU** : Regrouper des traitement sur des périphériques et alléger d'autres
    - **Réseau** : Regrouper 'localement' des composants/traitements échangeant beaucoup d'informations
    - **Batterie** : Remplacer certains composants/traitement par d'autres moins précis mais moins gourmands; Déplacer des composants/traitements pour alléger la charge

Quels (autres) leviers pour économiser ?

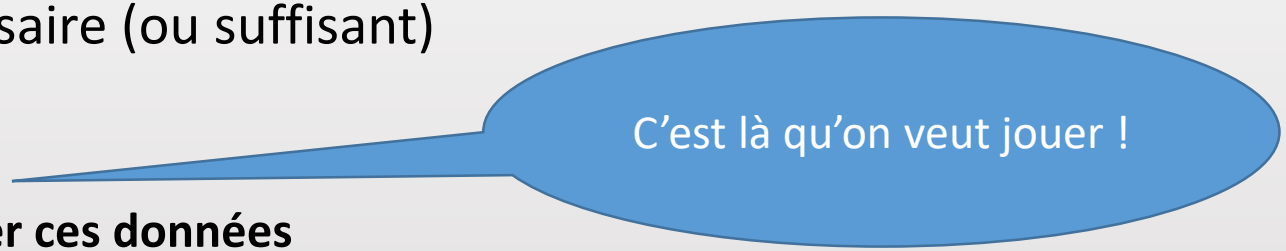
- Interactions
    - En remplaçant certains modalités d'interactions par d'autres
        - L'écran peut représenter jusqu'à 40% de la consommation d'une application
    - En déplaçant certaines modalités d'interactions

Exemple d'actions **sur plusieurs axes**

- Application s'exécutant sur un terminal mobile (e.g. un smartphone).
  - **Le réseau fait intervenir à la fois les appareils électroniques mais également des données**
  - Intuitivement: Plus on transmet de données, plus on consomme de l'énergie.

Exemple d'actions sur plusieurs axes

- Quelques solutions possibles:
  - a) des spécialistes travaillent sur les aspects électroniques afin de réduire la consommation de tels appareils.
  - b) d'autres travaillent sur l'agrégation de données afin de mieux transmettre les données et ainsi optimiser les accès et donc l'énergie utilisée pour leur transmission.
  - c) choisir l'interface (wifi, bluetooth par exemple) la moins consommatrice tout en respectant le débit nécessaire (ou suffisant)
  - **d) au niveau applicatif :**
    - **limiter, retarder ou filtrer ces données**
    - **déplacer l'exécution de tout ou partie de l'application sur un autre hôte afin de minimiser les transmissions**
    - **remplacer un traitement par un autre mieux adapté au contexte matériel**
    - **Remplacer/déplacer des interactions**



Modes d'action

- Comment : Kalimucho can do it !

Certes… Kalimucho « can do it »…mais Kalimucho « can't think it ». Il n'a aucune intelligence (et ce n'est pas le seul).

- Pb Décisionnel :
  - Quels sont les composants que je peux déplacer (et comment je le sais)
    - Composants de calcul, accès capteur, interface/interaction (I/O)
    - => Nécessite une connaissance des composants (type, conso moyenne, qtité I/O, etc.)
    - => une sorte de **GreenId**
  - Politique de redéploiement
    - Réseau, énergie… mais aussi : **Utilisabilité !**
  - **Pb NP-Complet => Heurisques**

Utiliser des microservices

- Paolo Di Francesco “microservices are small services that make up an application, each running in its own process and communicating with lightweight mechanisms “ (Paolo Di Francesco)
  - Mise à jour facilité
    - 1 fonctionnalité à mettre à jour = 1 module
    - 1 nouvelle fonctionnalité = 1 nouveau module
  - Découplage
  - Hétérogénéité (facilité de déploiement sur des périphériques de différents types/platformes)

  - => Netflix utilise les microservices

Kaligreen

- KaliGreen = Algo décentralisé permettant l'échange de services piloté par l'énergie

- Version light de l'algo
  - Identifier les microservices qui consomment le plus d'énergie.
  - Extraire les meta-données (CPU, taille, bande passante utilisée, etc.)
  - Ajouter dans un verteur de données
  - Envoyer le vecteur aux périphériques connectés qui évalueront la possibilité d'héberger le service
    - VectorID
    - ID du périph qui l'a produit
    - Moyenne puissance CPU nécessaire
    - Moyenne RAM nécessaire
    - Moyenne stockage requis
    - Résolution de l'écran (si necessaire) + temps moyen usage

  - Déplacer le microservice sur le meilleur périphérique hôte candidat

Algorithm 1 Algorithm running in an emergency state device D_i

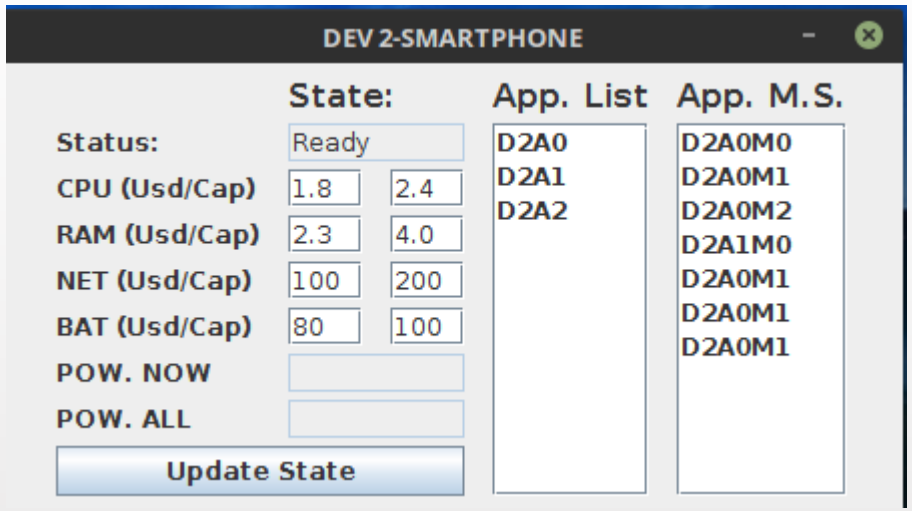
```
1:  while isEmergencySituation() == true do
2:    for all Microservices msApp_i do
3:      EvaluateEnergyImpact()
4:    end for
5:    msApp_h = selectHeaviestMicroservice()
6:    build Vector V_h with metadata of msApp_h
7:    for all Devices connected D_j do
8:      sendVector(V_h)
9:    end for
10:   WaitForCandidatsReponses()
11:   for all candidates Devices canD_j do
12:     D_b = selectBestCandidate()
13:   end for
14:   sendConfirmation(D_h)
15:   RecieveConfirmation(D_h)
16:   for all Non selected candidates Devices canD_u do
17:     sendNegativeConfirmation(canD_u)
18:   end for
19:   moveMicroservice(msApp_h, D_b)
20: end while
```

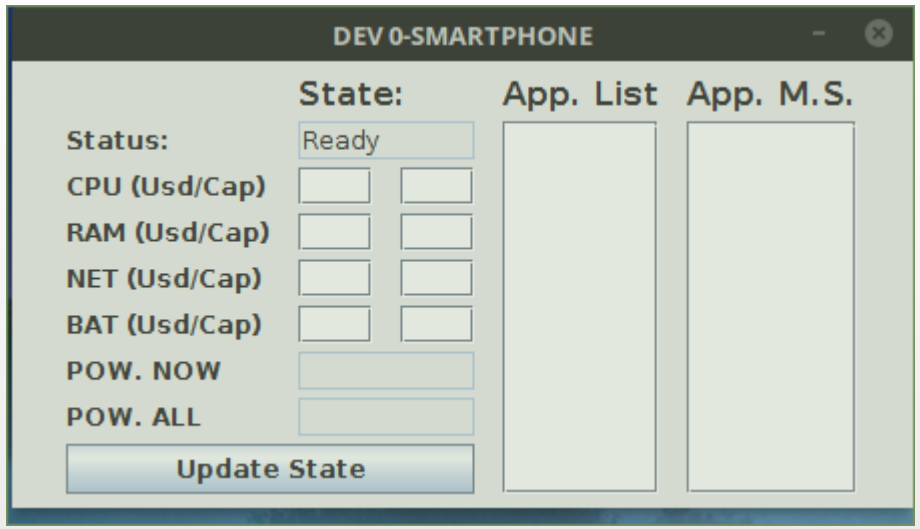
Kaligreen -> Kalimucho

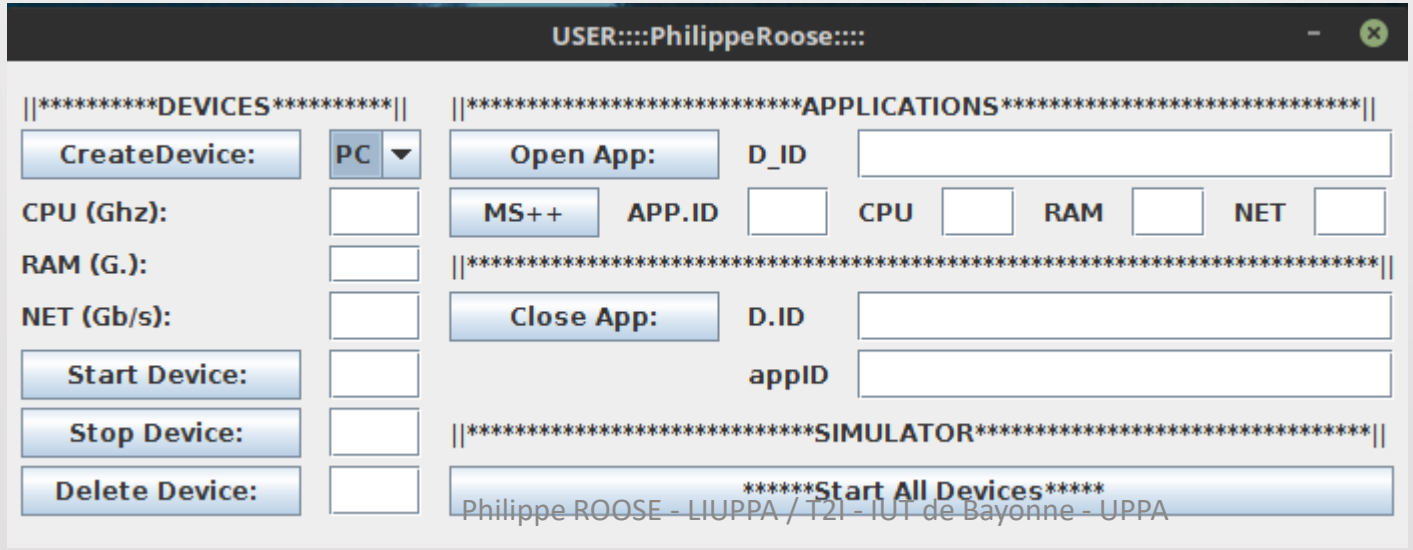
- 1) Kalimucho
- 2) Un ensemble de périphériques /utilisateur
  - D = [D 1 ...D n].
- 3) Un ensemble d'applications
  - APPS =[App 1 ...App n].

  Chacune composée d'un ensemble de microservices:
  - msAPPi = [msApp_i ...msApp_n]
- 4) Un vecteur de communication note V.
  - V_i pour chaque périph D_i .
- 5) Un service de monitoring / périph
  - monDS = [monD i ...monD n].

Prototype - état en cours







Conclusions

- Travaux préliminaires
  - Stagiaire M2 en cours
- Pour l'instant, projet plus « Sustainable » que « Green » !
- Approche plus globale avec Genie Logiciel inside
  - Equipes de recherches 'très green' + autres équipes plus middleware, context management, autonomic computing, génie logiciel