

# **IVY : UN OUTIL POUR LA CONCEPTION D'INTERFACES APPROCHE « BUS LOGICIEL D'ÉCHANGES DE MESSAGES »**



<https://www.github.com/truillet>

v.28 – septembre 2025

# UNE ARCHITECTURE RÉPARTIE ?

Les systèmes informatiques deviennent de plus en plus complexes

- en termes de périphériques utilisés divers et variés
- d'informations échangées
- d'interacteurs et de supports utilisés (fixes et/ou mobiles)

→ **il y a nécessité d'une architecture répartie pour prototyper rapidement**

## Principes généraux :

1. Etablir facilement des communications entre processus
2. Aller au delà du niveau d'abstraction (faible) des sockets

# UNE ARCHITECTURE RÉPARTIE ?

Les inconvénients fréquents des approches réparties ...

- un besoin de **centralisation** (où se trouve l'objet/ la méthode distante ?  
→ annuaires, broker, ...)
- un **coût d'apprentissage** élevé des différentes approches
- des architectures fréquemment **spécifiques** (ex : RMI, CORBA, OSGi,...) et peu adaptées à du multi-langage et au développement événementiel

**Ceci amène une incompatibilité des modèles d'architecture et des modèles d'exécution**

# APPROCHE RÉPARTIE POUR L'IHM ?

la plupart de middlewares ne sont pas orienté **interaction** mais plutôt centrés autour d'échanges/appels de fonctions ou d'objets ...

En fait, de quoi a t'on réellement besoin en interaction ?

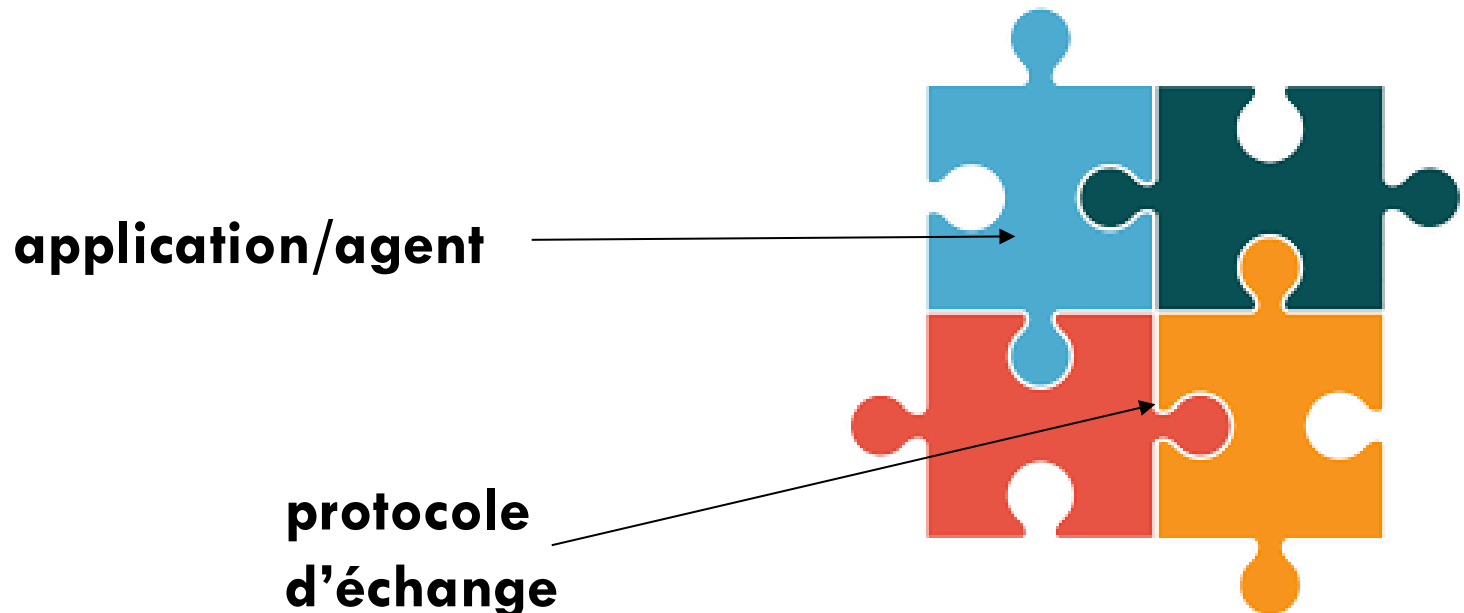
1. de séparer le Noyau Fonctionnel de l'interface
2. de pouvoir recevoir et/ou d'émettre des événements !

→ **une solution (parmi d'autres)** : utiliser un bus « **événementiel** » (et il en existe beaucoup !)

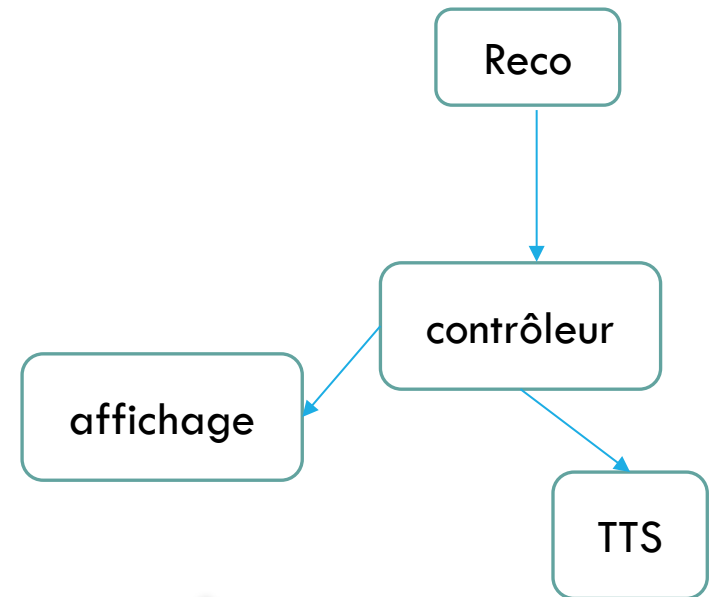
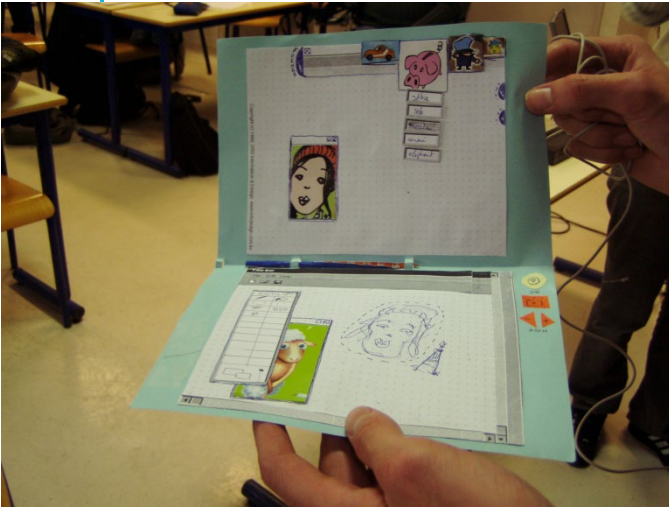
# APPROCHE RÉPARTIE POUR L'IHM ?

Le système interactif peut être vu comme « un *assemblage* » d'agents répartis, chaque agent ayant des capacités de calcul et d'interaction avec ses voisins ...

le travail se situe au niveau du protocole d'échange entre agents  
(la « *sémantique* » de l'événement ...)



# OBJECTIFS DE L'APPROCHE



# AVANTAGES DE L'APPROCHE

pour la **conception**...

1. modularité → réutilisabilité
2. usages de plusieurs plateformes et langages afin de passer rapidement de la phase « *papier* » au(x) *prototype(s) moyenne/haute fidélité*

pour **l'évaluation**

1. possibilité de comparer différentes implémentations de technologies
2. possibilité de tester les différents modules séparément

→ **meilleure visibilité du système**

# LE BUS IVY



ivy est un **bus logiciel** qui permet un échange d'informations textuelles entre des applications réparties sur différentes machines tournant sous différents OS et écrites avec des langages différents ...

créé en 1996 au CENA (DGAC) pour des besoins de prototypage rapide

ivy est simple (<https://hal-enac.archives-ouvertes.fr/hal-00940960/document>)

- à comprendre,
- à mettre en œuvre
- et c'est gratuit ;-)



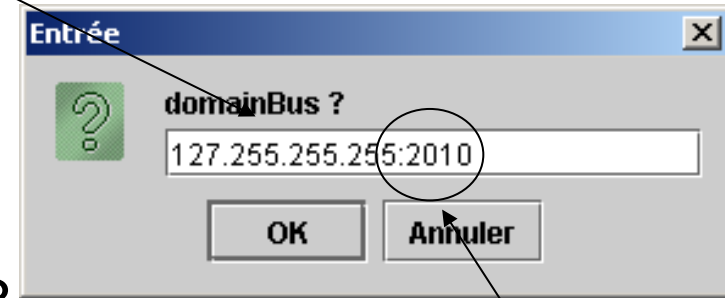
# LE BUS IVY



■ ivy

adresse IP  
adresse de broadcast  
adresse de multicast

- ivy n'utilise pas de serveur central
  - il utilise néanmoins la couche TCP/IP
  - chaque agent propose un ou des services
  - chaque agent réagit à un ou des événements



port de communication

- avec une **sémantique proche** de la programmation événementielle (Java, #, , ...)



# LE BUS IVY

Pour fonctionner, il faut :

- une adresse IP ou (mieux) une adresse de broadcast / multicast
- un point de rendez-vous → un port de communication

1. **Tous les agents connectés sur le même segment de réseau et le même port peuvent se connecter entre eux**
2. Peut importe l'ordre de lancement des agents !
3. Les agents peuvent apparaître/disparaître sans problèmes

# LE BUS IVY

## ivy est disponible

- en ada95, C, C++, C#, Flash , java, nodeJS, ocaml , perl, perl/Tk, Processing.org, Python (2 et 3), ruby, Rust, Tcl, Tcl/Tk, VBA, ...
- sous MacOS, Win32, Win64, Un\*x, linux, Android, ...

**conséquence** : la conception est facilitée en profitant des avantages liés à chaque langage de programmation

# UTILISATEURS (CONNUS)

## Laboratoires



DSNA



CLIPS  
Communication Langagière et  
Interaction Personne-Système  
Fédération IIMAG  
BP 53 - 38041 Grenoble Cedex 9 - France



## Entreprises



**CURTEK SYSTEMS**



**THALES**



# COMMENT PROGRAMMER AVEC IVY ?



: développé au CENA (DTI R&D)

librairie de « mise en réseau » d'agents

Le bus permet d'ajouter la possibilité de recevoir et d'envoyer des messages depuis et vers toutes les APIs nécessaires au développement tout en restant indépendant des OS et des langages !



# LE BUS IVY

Le protocole d'échanges de messages est purement textuel  
(abonnement par *expressions rationnelles* / **regex PCRE**)

**Vous êtes libres du format d'échange ... MAIS il vaut mieux être structuré**  
...

- exemples d'envoi :  
`ICAR command=back`

`IMM media=SRAP action=previous`

Couples de variable/valeur

Essayer les  
regex !

<https://regex101.com/>

Nom de l'application émettrice du message



# LE BUS IVY

Les éléments de **regexp** les plus utilisés :

**^** : « doit débiter par »

**.\*** : n'importe quel caractère répété n fois

**(.\*)** : permet de récupérer un argument (chaîne de caractères)

Ex : **^Appli timestamp=.\* x=(.\*) y=(.\*)**

Le message envoyé « **Appli timestamp=123456 x=12 y=56** »  
permet de récupérer les arguments 12 et 56



# LE BUS IVY

4 étapes essentielles pour créer un agent :

1. Créer un agent
2. Définir les comportements (envoi/réception)
3. « Lancer l'agent sur le bus » et le maintenir « en vie » (mainloop)
4. Arrêter l'agent avant de quitter





# LE BUS IVY

2 mécanismes « basiques » : la réception (**bindMsg**) et l'émission (**sendMsg**) de messages

```
/* listener ivy */
bus.bindMsg("ivyEcoBe! to=(.*) event=(.*)", new IvyMessageListener() {
    public void receive(IvyClient client, String[] args)
    {
        try
        {
            /* envoi vers le robot EcoBe!*/
            if (args[1].compareTo("Stop")==0)
            {
                . . . . .
            }
        }
    }
});
```

Fonction callback  
exécutée quand le  
message est levé

```
        // Envoi que sur trames GGA -> les autres ne servent qu'à mettre à jour les champs
        bus.sendMsg(name + " type=" + DT + " temps="+time + " lat="+lat + " long="+lon + " alt="+
altitude + " vitesse="+vitesse + " cap="+cap + " mode="+mode + " HDOP="+ HDOP + " Nb_Satellites=" +
Nb_Satellites + " Force_Signal=" + Force_Signal);
```



# LE BUS IVY

Chaque argument de la **regex** (que l'on retrouve entre « ( ) ») est accessible dans votre programme au travers d'un tableau de valeurs et directement exploitable

**La fonction callback s'exécute quand la regex est vérifiée**

Ex :

Regex  $\rightarrow$  Appli  $X=(.*)$   $Y=(.*)$

*Messages envoyés (depuis un autre agent)*

Appli  $x=12$   $Y=14$  n'aura aucun effet

Appli  $X=12$   $Y=14$  permettra de récupérer  
2 arguments de type « *string* » dans `args[0]` et `args[1]`  
(en java)

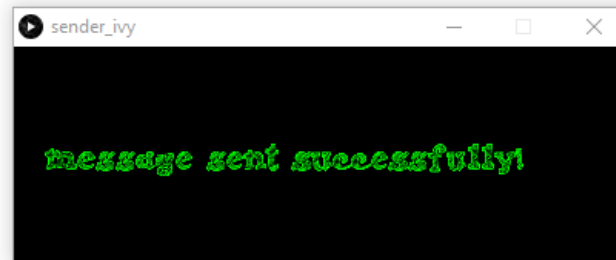


# LE BUS IVY

Recevoir un message consiste donc :

1. À extraire les arguments intéressants
2. A les « caster » dans le type souhaité
3. Les utiliser

Un exemple  
(Processing) →





# LE BUS IVY

```
3 void mousePressed()
4 {
5     try
6     {
7         bus.sendMessage("Demo_Processing Command=Damn ... New message");
8     }
9     catch (IvyException ie)
10    {
11    }
12 }
```

```
try
{
    bus = new Ivy("demo", " demo_processing is ready", null);
    bus.start("127.255.255.255:2010");

    bus.bindMsg("^Demo_Processing Command=(*)", new IvyMessageListener()
    {
        public void receive(IvyClient client, String[] args)
        {
            message = args[0];
            try
            {
                bus.sendMessage("Demo_Processing Feedback=ok");
            }
            catch (IvyException ie) {}
        }
    });
}
```



# LE BUS IVY

## Attention (problème récurrent) !

1. Pour fonctionner, un agent ivy doit rester « **vivant** » (c'est-à-dire s'exécuter dans une boucle infinie – *mainloop* - permettant de capturer/envoyer des événements)

# CONCLUSIONS

L'approche « *bus événementiel* » permet au final :

- de **se focaliser sur les problèmes de conception** et non sur la façon de les implémenter
- et **de prototyper très rapidement** pour « *donner à voir* » et « *donner à tester* »

# LIENS

## **Sites officiels d'ivy**

- <http://www.eei.cena.fr/products/ivy/download>  
(*maintenance très aléatoire*)
- <https://gitpub.recherche.enac.fr/ivy> (*Git*)

## **Site spécifique « librairies »**

- <https://github.com/truillet/ivy>