



OUTIL POUR LA CONCEPTION D'IHM APPROCHE « BUS LOGICIEL D'ÉCHANGES DE MESSAGES »



<http://www.irit.fr/~Philippe.Truillet>
v.2.2 – septembre 2018

UNE ARCHITECTURE RÉPARTIE ?

les systèmes informatiques deviennent de plus en plus complexes

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

→ **Nécessité d'une architecture répartie**

le principe : établir une communication interprocessus

- **ET** aller au-delà du niveau d'abstraction de la socket

UNE ARCHITECTURE RÉPARTIE ?

Les inconvénients fréquents des approches réparties ...

- centralisation à un moment donné (où se trouve l'objet/ la méthode distante ?)
- coût d'apprentissage élevé des différentes approches
- 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 modèles d'exécution

APPROCHE RÉPARTIE POUR L'IHM ?

la plupart de middlewares ne sont pas orienté interaction mais échanges/appels d'objets/fonctions ...

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

- séparer le Noyau Fonctionnel de l'interface
- pouvoir émettre et/ou de recevoir des événements et non pas d'appeler des méthodes ou des fonctions !

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

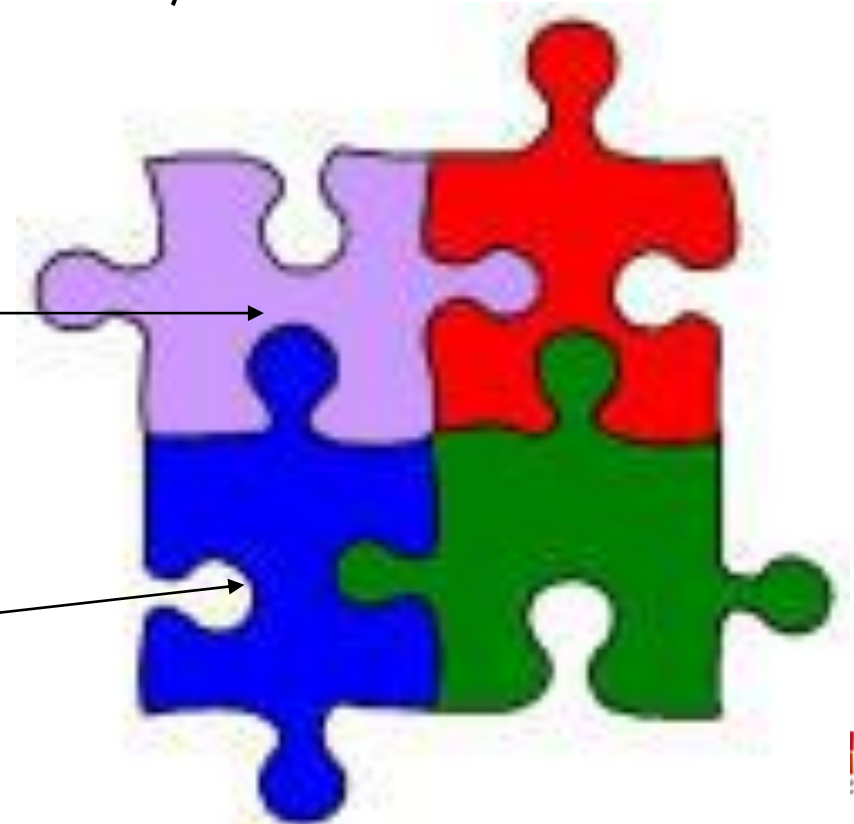
APPROCHE RÉPARTIE POUR L'IHM ?

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

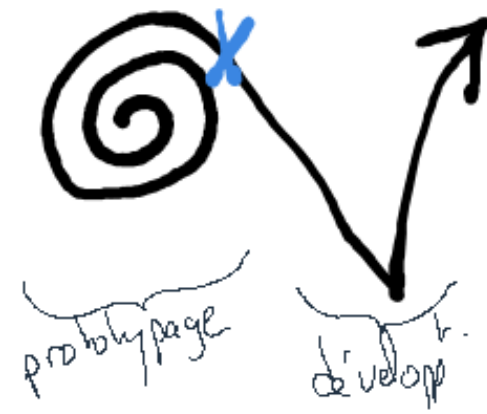
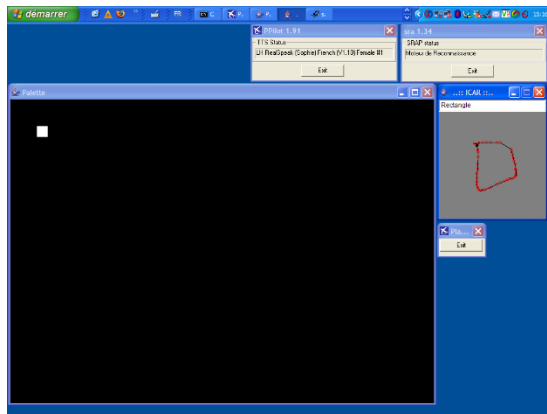
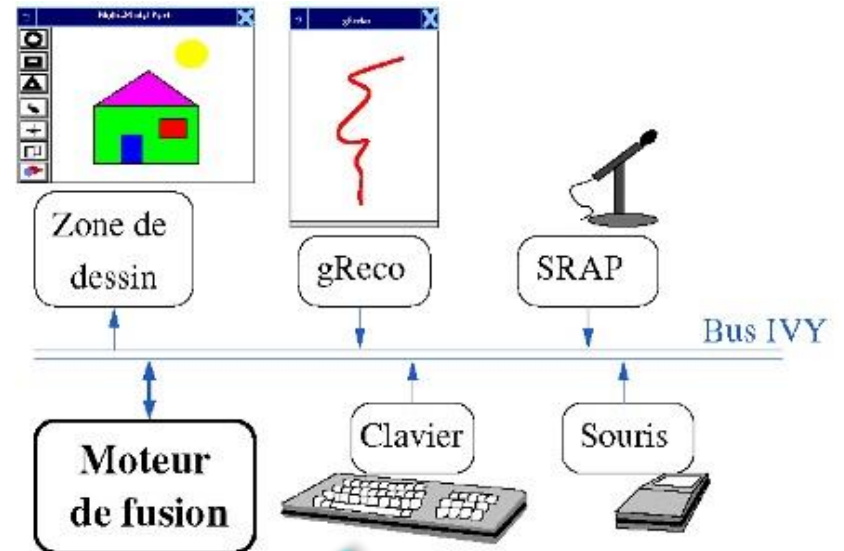
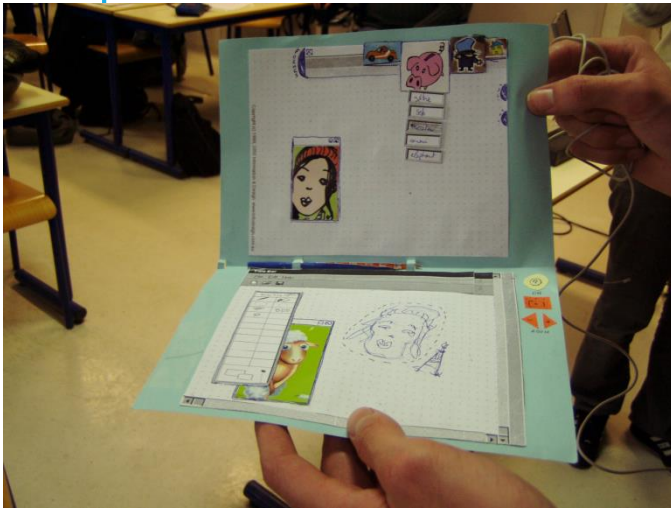
le travail se situe au niveau du protocole d'échange entre agents (la « *sémantique* » de l'événement ...)

application/agent

**protocole
d'échange**



OBJECTIFS DE L'APPROCHE



OBJECTIFS DE L'APPROCHE

intérêts pour la **conception...**

- modularité = réutilisabilité
- usages de plusieurs plate-formes et langages afin de passer rapidement de la phase « *papier* » au(x) *prototype(s) moyenne/haute fidélité*

et pour **l'évaluation**

- 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 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 (<http://www.eei.cena.fr/products/ivy>)

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

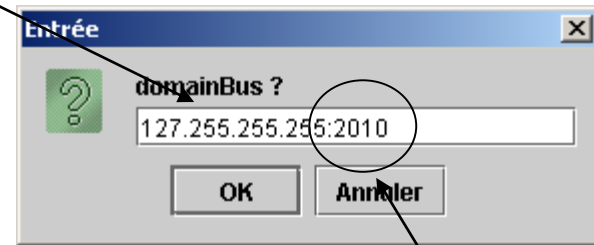
LE BUS IVY



■ ivy

adresse IP
adresse de broadcast
adresse de multicast

- ivy n'est pas basé sur un serveur centralisé
- chaque agent propose un ou des services
- chaque agent réagit à un ou des événements



port de communication

- proche de la programmation événementielle (Java, .NET, X-window, ...)



LE BUS IVY

ivy est disponible

- en ada95, C, C++, C#, Flash , java, ocaml , perl, perl/Tk, Processing , python, ruby, 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



Entreprises



COMMENT PROGRAMMER AVEC IVY ?



: développé au CENA (DTI R&D)

librairie de « mise en réseau » d'agents

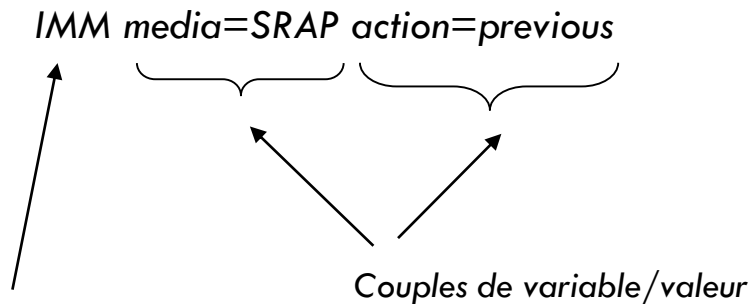
Le bus permet d'ajouter la possibilité de recevoir et d'envoyer des messages avec toutes les APIs nécessaires au développement (ex : SAPI pour la parole [Windows], ARToolkit pour la réalité augmentée, Processing pour de l'animation graphique, ...) tout en restant indépendant des OS et des langages !



LE BUS IVY

le protocole d'échanges de messages est purement textuel
(abonnement par expressions régulières / regexp)

- exemples d'envoi :
ICAR command=back



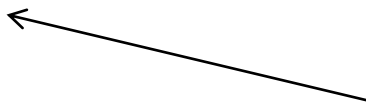
Nom de l'application émettrice du message

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)
            {
                . . . . .

                // 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

exemples d'abonnement :

`^ICAR command=(.*)`

`^IMM (.*) action=(.*)`

4 étapes :

1. Créer le bus
2. « Lancer le bus »
3. Définir les comportements (envoi/réception)
4. Arrêter le bus avant de quitter

CONCLUSIONS

l'approche « bus événementiel » permet :

- 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/>
- <https://svn.tls.cena.fr> (*Subversion*)

Sites spécifiques « librairies »

- **Python** : <https://gitlab.com/ivybus/ivy-python> et <https://pypi.python.org/pypi/ivy-python>
- **C** : <https://github.com/esden/ivy-c/>
- **Java** : <http://lii-enac.fr/~jestin/homepage/software.html>

Site « ivystore » de l'IRIT (*en cours de révision*)

- <http://www.irit.fr/~Philippe.Truillet/ivystore>