ContrACT: a software environment for developing control architecture

R. Passama
D. Andreu

ContrACT: a software environment for developing control architecture

- Contrôleur de robot : assemblage d'algorithmes ”robotiques”
- Focus :
  - composition des algorithmes utilisés dans les boucles périodiques temps-réel
    - Asservissements
    - Boucles de perception/observation
  - Supervision
    - Assemblage/configuration dynamique des algorithmes
    - Configuration dynamique de l'ordonnancement
    - Réaction rapide à des événements

ContrACT: a software environment for developing control architecture

- Objectif : une plateforme de programmation
  - Pour faciliter la programmation
  - Pour ”Structurer” le développement (aspects temps-réels)

- Bénéfices attendus :
  - Maximiser la réutilisation des différentes constituantes des boucles
  - Contrôler précisément leur exécution temps-réel

ContrACT: a software environment for developing control architecture

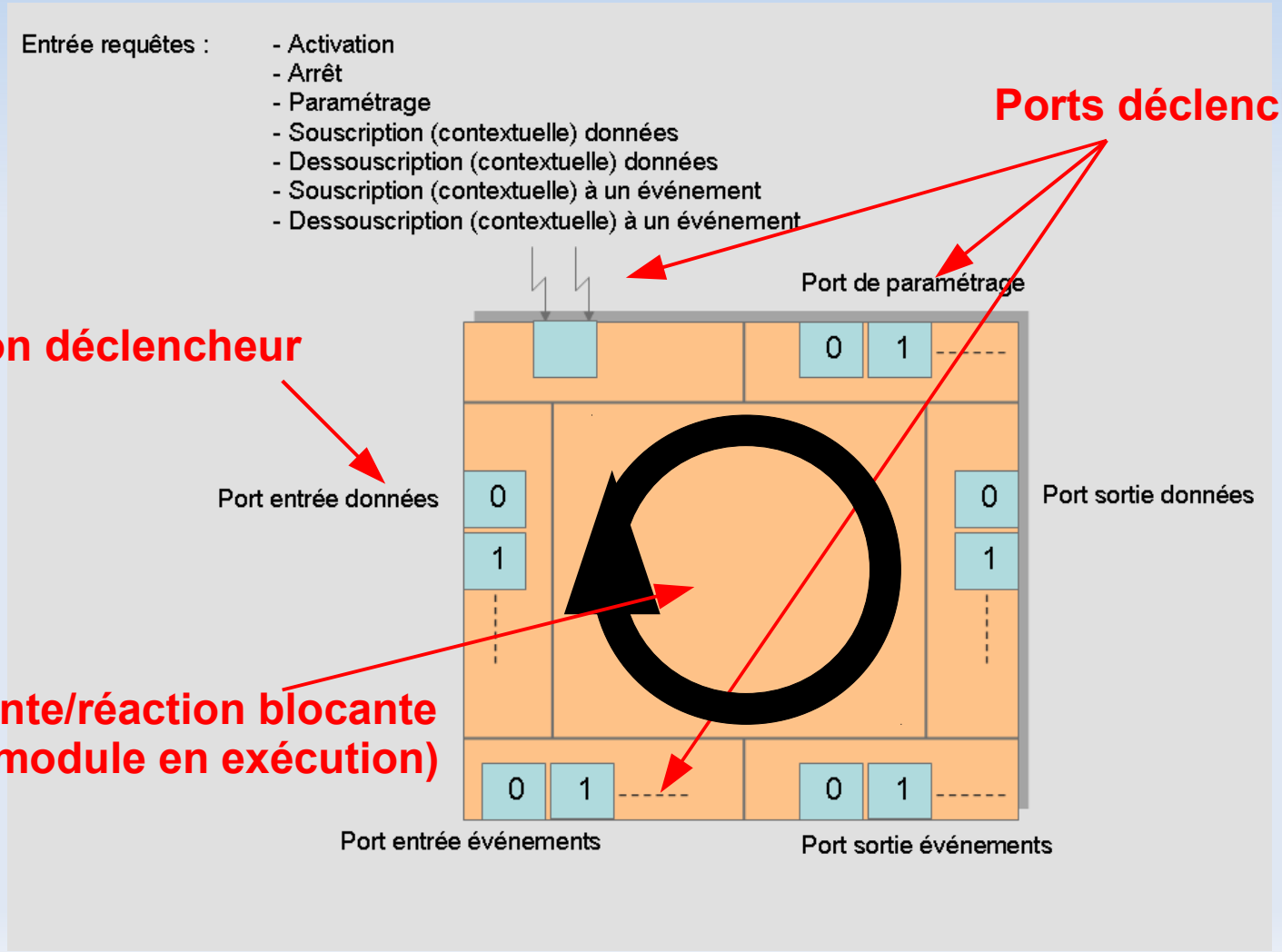
- Plan
  - Introduction
  - Approche générale
  - Middleware temps-réel
  - Editeur d'architectures
  - Conclusion

ContrACT: a software environment for developing control architecture

- Approche générale:
  - Architecture modulaire : l'entité de base est un module correspondant à une tâche temps-réel
  - Formes de composition
    - Basée flot de données
    - Basée événements
    - Basée requête
  - 2 couches:
    - Décisionelle (supervision ”réactive”)
    - Exécutive (ordonnancement temps-réel)

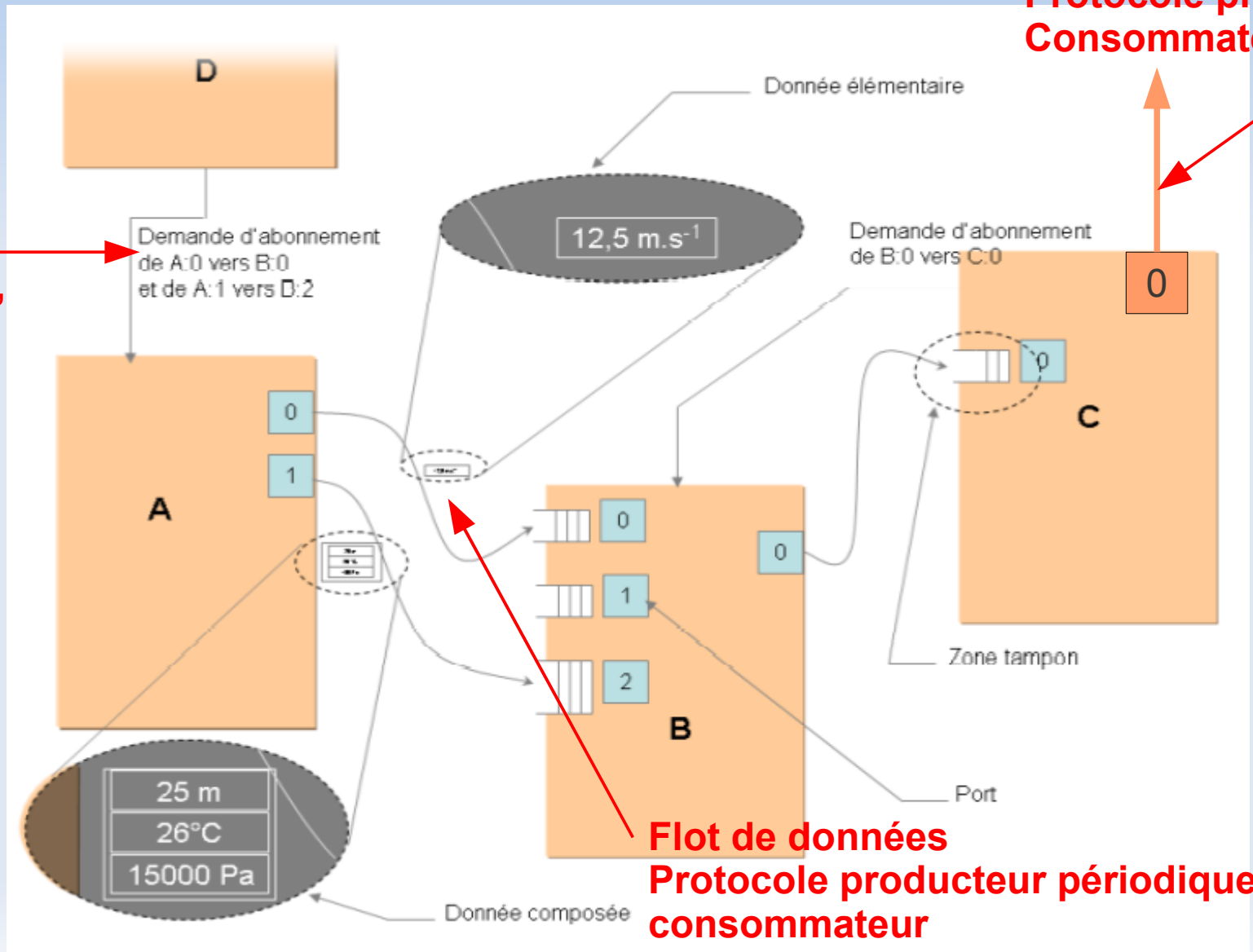
ContrACT: a software environment for developing control architecture

- Module : tâche temps-réel LXRT



ContrACT: a software environment for developing control architecture

- Formes de composition



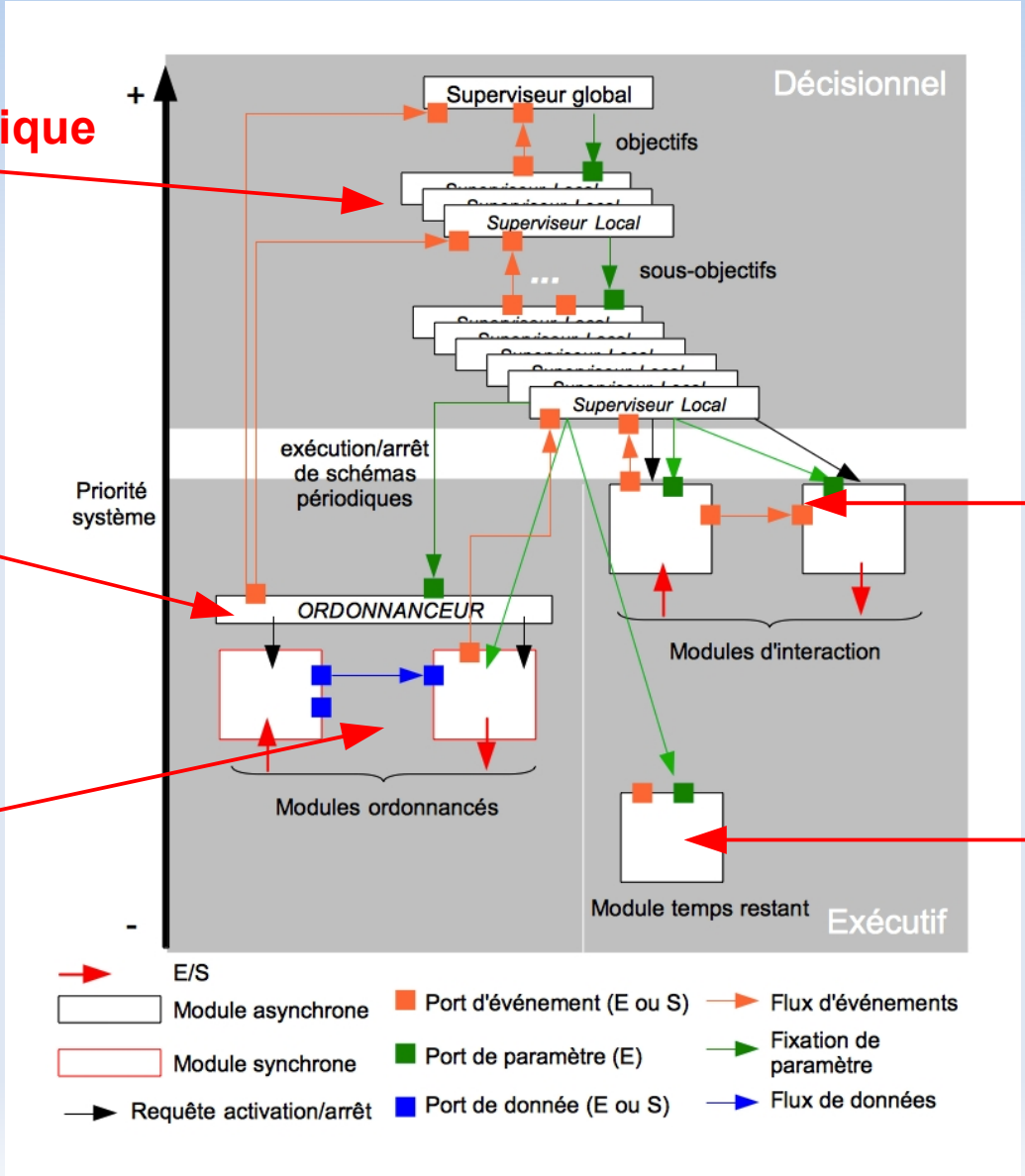
ContrACT: a software environment for developing control architecture

- Architecture 2 couches

Organisation hiérarchique de superviseurs

Module chargé de l'ordonnancement temps-réel

Modules périodiques (asservissements)



Gestions entrées Sorties asynchrones

Calculs ”longs” non périodiques

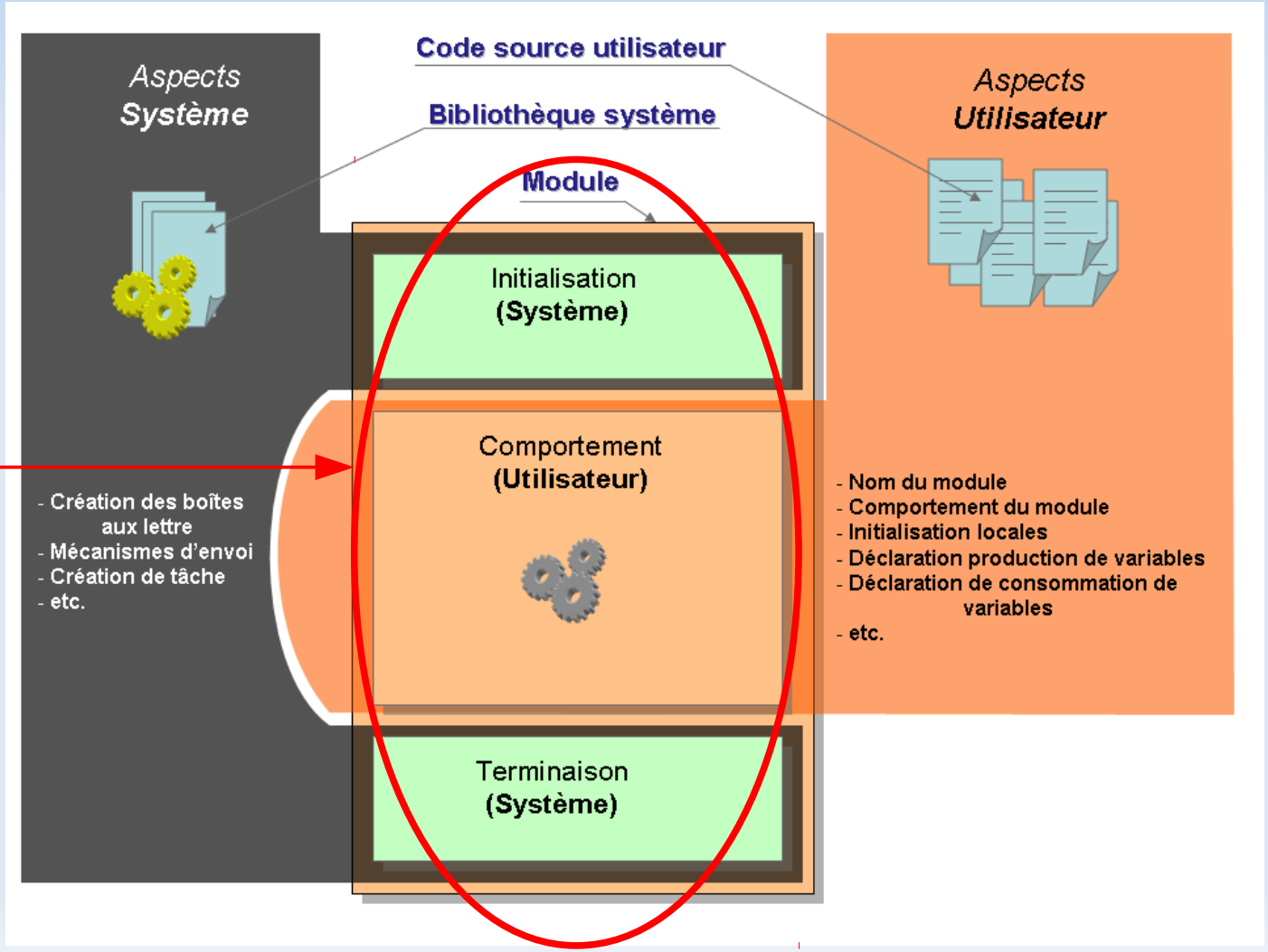
ContrACT: a software environment for developing control architecture

- Middleware temps-réel
  - Librairie RTAI pour programmer les modules
  - Squelettes de code des différents types de modules
    - Synchrones (périodiques)
    - Asynchrones
  - Implémentations spécifiques
    - Superviseurs
    - Ordonnanceur

ContrACT: a software environment for developing control architecture

- Librairie RTAI pour programmer les modules

Squelette de code d'un module :
- interface entre Système et utilisateur
- restrictions imposées en fonction du type



ContrACT: a software environment for developing control architecture

- Squelette de code

Identification ContrACT + priorité temps-réel du module

Déclaration des ports

- Données

- Evénements

Point d'entrée principal du programme (contient la boucle de réaction)

```
...

// information on module

MODULE_DESCRIPTION("say what the module does");
MODULE_AUTHOR("Your name");

//Exchanges Management Variables
int input, output;
float notify, react;

// module name
char MODULE_NAME[] = "MOD";
//module priority
int MODULE_PRIORITY = 37;

// input data flows : example = { &Variable, PortReception, Size,
Periodicity (important if module name is set), Name of emitter module(can
be unused by setting value to "---"), PortModuleEmetteur} ou { &Variable,
PortReception, Size, 0, "---", 0}
ModuleUse IUSE[] = { {&input, 0, sizeof(int), 0, "---", 0}, IUSE_TERM };

// output data flows : example = { &Variable, PortEmision, Size}
ModuleProduce IPRODUCE[] = { {&output, 0, sizeof(int)}, IPRODUCE_TERM };

//input event flows : example = { &Variable, PortReception, size, module
name (can be unused by setting value to "---"), PortEmision, oneshot}
ModuleReact IREACT [] = { {&react, 0, sizeof(float), "---", 0, 0},
IREACT_TERM };

//output event flows : example = { &variable, PortEmision, size }
ModuleDetect IDETECT [] = { {&notify, 0, sizeof(float)}, IDETECT_TERM };

// size of parameters that can be received in the request mailbox:
size_t PARAM_SIZE_IN[] = { sizeof(double), PARAM_SIZE_TERM };

// size of parameters or events that can be sent in request mailboxes of
other modules :

size_t PARAM_SIZE_OUT[] = {sizeof(char)*256, PARAM_SIZE_TERM };

// description of the module behavior

...
int ModuleMain(int argc, const char * argv[]) {
...
}
```

ContrACT: a software environment for developing control architecture

- Implémentations Spécifiques : Ordonnanceur applicatif
  - **Unique** dans l'architecture
  - Algorithme EDF (**Earlier Deadline First**)
  - Ordonnancement **configurable dynamiquement** : ajout/retrait ”à la volée” de modules.
  - Contraintes gérées : **temporelles** (fréquence et délai critique d'une succession de modules, durée nominale d'un module), **précédence** et **exclusion mutuelle** entre modules
  - **Détection** des erreurs temporelles et **notification** aux superviseurs

ContrACT: a software environment for developing control architecture

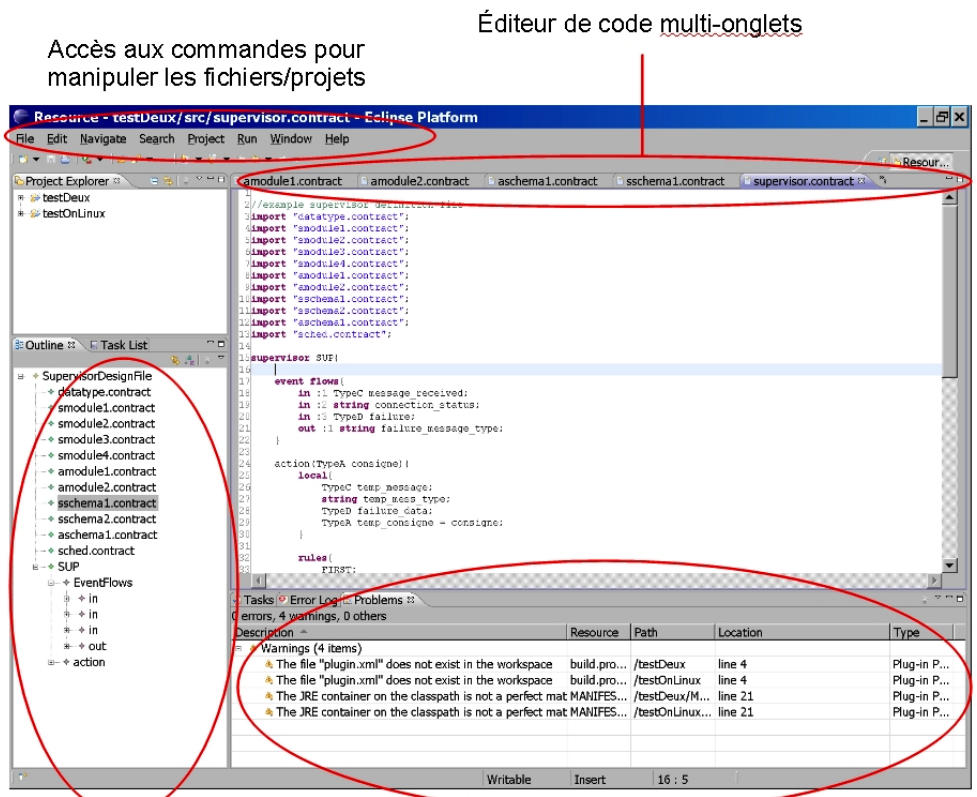
- Implémentations Spécifiques : Superviseurs
  - Potentiellement plusieurs dans l'architecture
  - Comportement
    - **asynchrone** (l'évolution de l'état n'est réalisée que quand l'état à pu changer),
    - **réactif** (réaction très courtes à des changements d'états)
    - **temps-réel** (priorité de réaction supérieure aux modules exécutifs)
  - Basé sur l'évaluation de **règles de supervision** (précondition sur l'occurrence d'événements, actions, postcondition sur l'occurrence d'événements)
  - Actions possibles : ordonnancer un assemblage de modules, assembler/configurer des modules, exécuter des modules asynchrones, envoyer un sous-objectif à un superviseur, notifier des événements, etc.

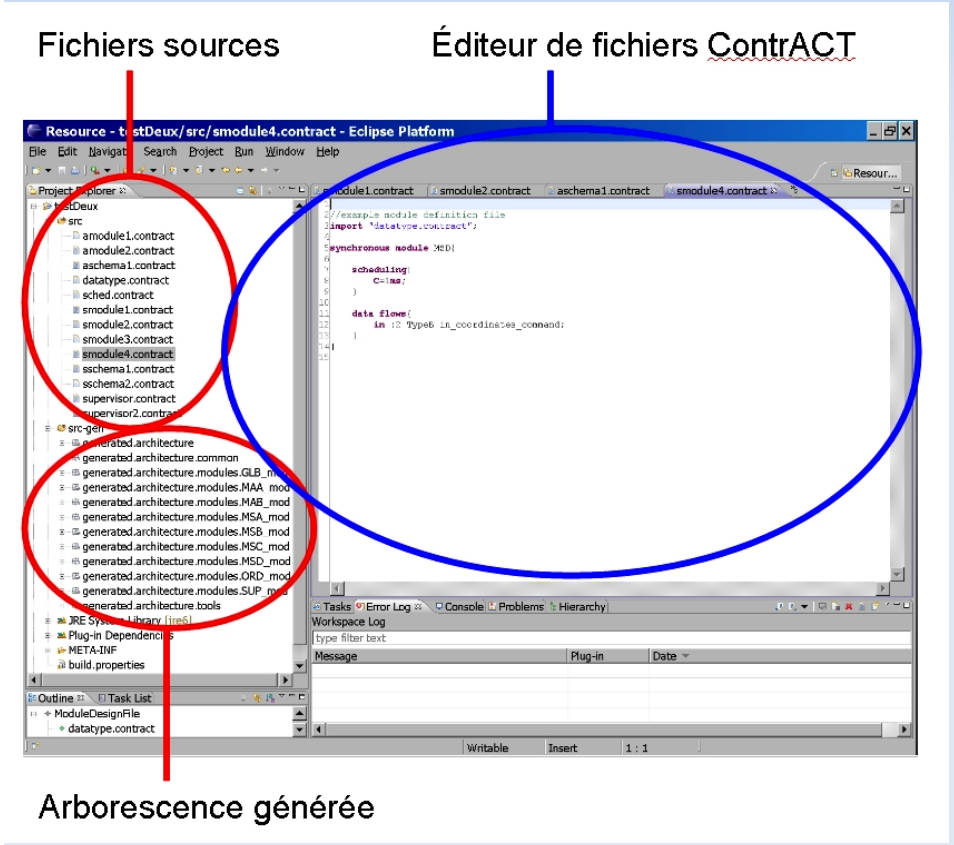
ContrACT: a software environment for developing control architecture

- Editeur
  - Partie édition : utilisation d'un DSL
    - Description des interface des modules
    - Description des schéma (assemblages) périodiques (ordonnancés) ou apériodiques
    - Description complète des superviseurs (règles)
  - Partie génération
    - Génération du code des module/données
    - Points de personnalisations du code ”protégés”
    - Génération des commandes système pour piloter les modules

ContrACT: a software environment for developing control architecture

- Editeur : IHM





ContrACT: a software environment for developing control architecture

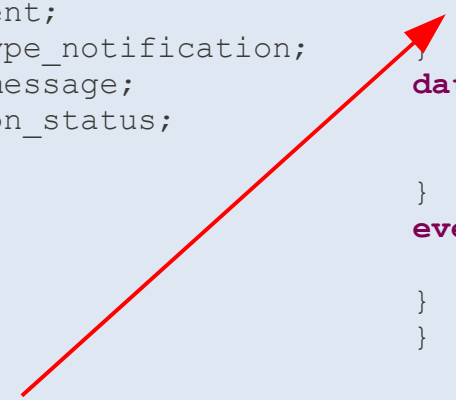
- Editeur : interfaces des modules

Module asynchrone

```
//example module definition file
import "datatype.contract";
asynchronous module MAA{
parameters{
      TypeA param = {22.12,6.98,897.34} :0;
}
event flows{
      in :0 string controlevent;
      out :0 TypeC message_type_notification;
      out :1 TypeE received_message;
      out :2 string connection_status;
}
}
```

Module synchrone

```
//example module definition file
import "datatype.contract";
synchronous module MSC{
parameters{
      TypeA param = {2.1,6.98,5.34} :0;
}
scheduling{
      C=1ms;
}
data flows{
      in :0 TypeA in_coordinates;
      out :1 TypeB out_coordinates_commmand;
}
event flows{
      out :1 TypeD failing_data;
}
}
```



Durée d'exécution d'un cycle

ContrACT: a software environment for developing control architecture

- Editeur : schémas

Schéma périodique

```
import "datatype.contract";
import "smodule1.contract";
import "smodule3.contract";
import "smodule4.contract";
periodic schema schemaCommande1(
     TypeA consigne = {0.0, 0.0, 0.0}){
realtime{
     PERIOD=1s;
     CRITICAL_DELAY=1s;
}
modules{
     MSA ();
     MSC (consigne:0);
     MSD ();
}
scheduling graph{
     MSA -> MSC;
     MSC -> MSD;
}
communications{
     MSA:0 -> MSC:0 * 1;//data flow communication
     MSC:1 -> MSD :2 * 2; //data flow communication
}
}
```

Schéma événementiel

```
import "datatype.contract";
import "amodule1.contract";
import "amodule2.contract";

event schema schemaReceptionMessage(
     TypeA param = {27.12,0.0,0.04}){
modules{
          MAA (param :0);
          MAB ();
}
communications{
          MAA:0 =* MAB:0;//continuous event flow
     }
}
```

Paramètres temps-réel

Graphe de précédence

Liens de communication

fréquences de production

ContrACT: a software environment for developing control architecture

- Editeur : superviseurs

```
supervisor SUP{

event flows{
in :1 TypeC message_received;
in :2 string connection_status;
out :1 string failure_message;
}

fonction1(TypeA consigne){
...
}

fonction2(TypeB param){
...
}
```

Priorité d'évaluation

\+

\-

```
action(TypeA consigne){
local{
TypeC temp_message;
string temp_mess_type;
TypeD failure_data;
TypeA temp_consigne = consigne;
}

rules{
FIRST:
      [elapsed=1ms]
            subscribe MAA:2 =* :2;subscribe MAA:0 =* :1;
            activate schema schemaReceptionMessage()
      [MAA:2<connection_status == "disconnected">]

SECOND:
      [(MAA:2<(connection_status=="disconnected" & test("dedede",temp_consigne))>
      and started(FIRST){infinite})]
            subscribe MSC:1=>:3;
            activate schema schemaCommande1(consigne = temp_consigne)
      [MSC:1(failure_data = failure, temp_mess_type = "commutation")]

THIRD:
      [endof(SECOND)]
            notify :1(failure_message_type=temp_mess_type)
      [started(THIRD)]

FOURTH:
      [endof(SECOND)]
            compute calcul("1.24",temp_consigne.k);
            activate schema schemaCommande2(consigne = temp_consigne)
      [endof(FIRST)]

RECEPTION:
      [MAA:0(temp_message=message_received)]
            parameterize MSC(temp_message.value :1);
            activate resttime module MAD
      [started(RECEPTION)]

}
}
```

Variables locales

Identifiant de la règle

précondition

actions

postcondition

ContrACT: a software environment for developing control architecture

- Editeur
  - Génération de code
    - Génère le **code squelettes** des modules /données (et les fichiers de compilation), **personnalisables** par l'utilisateur
    - Génération complète du code des superviseurs et de l'ordonnanceur (non personnalisables) ainsi que leurs fichiers de compilation
  - Compilation C & Lancement de l'application
    - Se fait ”à la main” via un terminal en utilisant les fichiers de compilation générés (Makefile)

ContrACT: a software environment for developing control architecture

- Conclusion
  - Gestion ”à grain fin” de la modularité et des aspects temps-réel, aux niveaux exécutif et décisionnel
  - Outil logiciel développé et opérationnel
  - **Axes d'évolutions possibles**
    - Gestion spécifique des commutations de schémas
    - Evaluation temps-réel & gestion de l' adaptation (fonction des durées d'exécution réelles)
    - Planification de mission ”native” aux superviseurs & gestion base de connaissance
    - Validation des superviseurs par Réseaux de Petri
    - Gestion spécifique du hardware
    - Evolution vers le paradigme ”composants”