Présenté par

Tahiry Ratsiambahotra

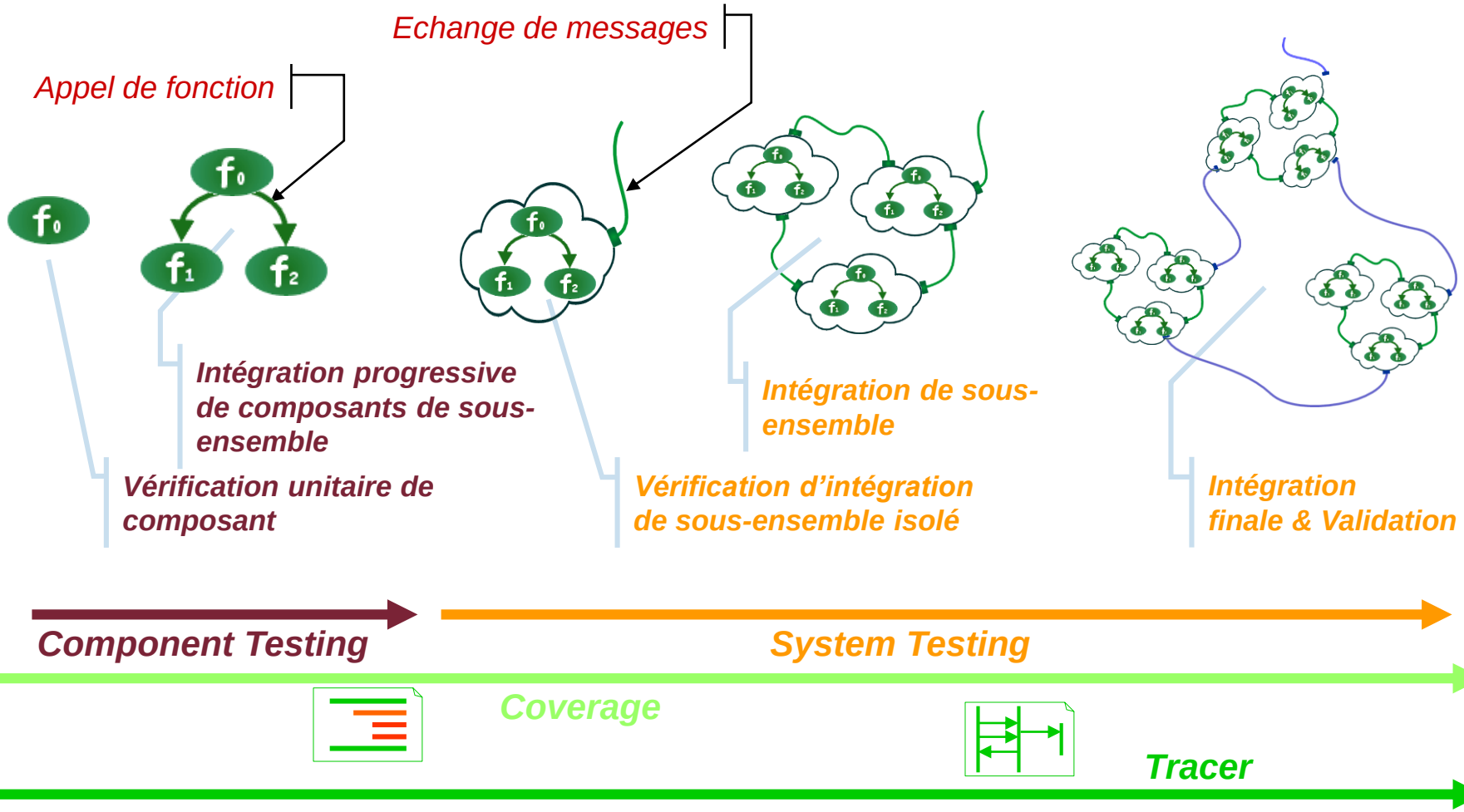
Formation aux Outils de Vérification
TEST DE COMPOSANS ET TEST SYSTEME

Produit RTRT

- **UN PRODUIT, UNE LICENCE, DES OUTILS**
  - **IBM-Rational Test RealTime (30 licences)**
    - **Component Testing => test unitaires de composant**
    - **Coverage => analyse de couverture structurelle**
    - System Testing => tests d'intégration client/serveur
    - Tracer => fourniture de traces de flot de contrôle
    - Purify => détection débordement mémoire
    - Quantify => mesure de temps CPU
    - Rational Test RealTime Virtual Testers xxx
      (2 licences spécifiques) superviseur en interface avec RTRT System Testing
- **3 LANGAGES : C, C++, ADA**
- **3 PLATE-FORMES : SUN SOLARIS, PC NT et PC LINUX**
- **DES RUNTIMES STANDARD (adaptés par EYYWI)**

Situation des outils dans la gamme RTRT

- Place des outils dans le cycle de développement et de vérification



FORCES DU LANGAGE

- Unifiée (sous-traitance à l'échelle internationale)
- Génération de sous cas de test (plan de test optimisé en taille de code)
- Gestion d'obsolescence (AUTAN ➔ ATTOL → RTRT → IBM RTRT)
- Simple facile à apprendre
- Répond aux besoins de vérification vis-à-vis de la DO178B
- Multi plateforme
- Adaptable (via runtime)
- Utilisé par AIRBUS, THALES et tous les sous-traitants

Outil de tests unitaires
RTRT Component Testing for C
PHENIX

Objectifs de la vérification unitaire

En conformité avec l'exigence de certification :

- Il faut prouver que le service réalise la fonction prévue en vérifiant que son comportement est conforme à celui décrit dans sa spécification → tests fonctionnels

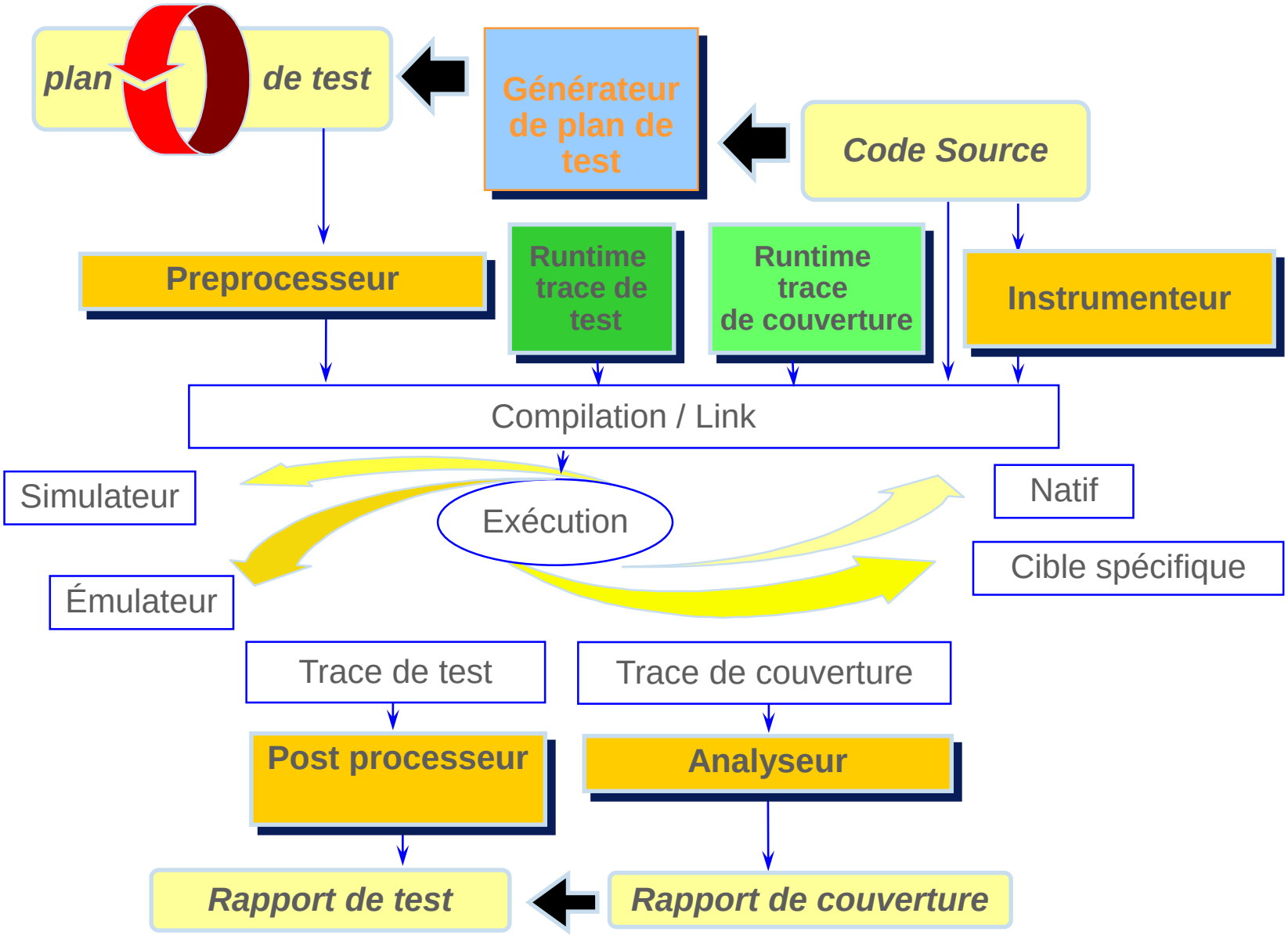
- Il faut également démontrer que le service a un comportement prévisible en cas de données invalides. Si une erreur d'interface apparaît, le service ne doit pas propager celle-ci. Il faut s'assurer que le service réalise uniquement la fonction prévue → tests de robustesse

- Il faut vérifier que le service respecte les contraintes de temps d'exécution (ex: boucle d 'acquisition) → tests de performance

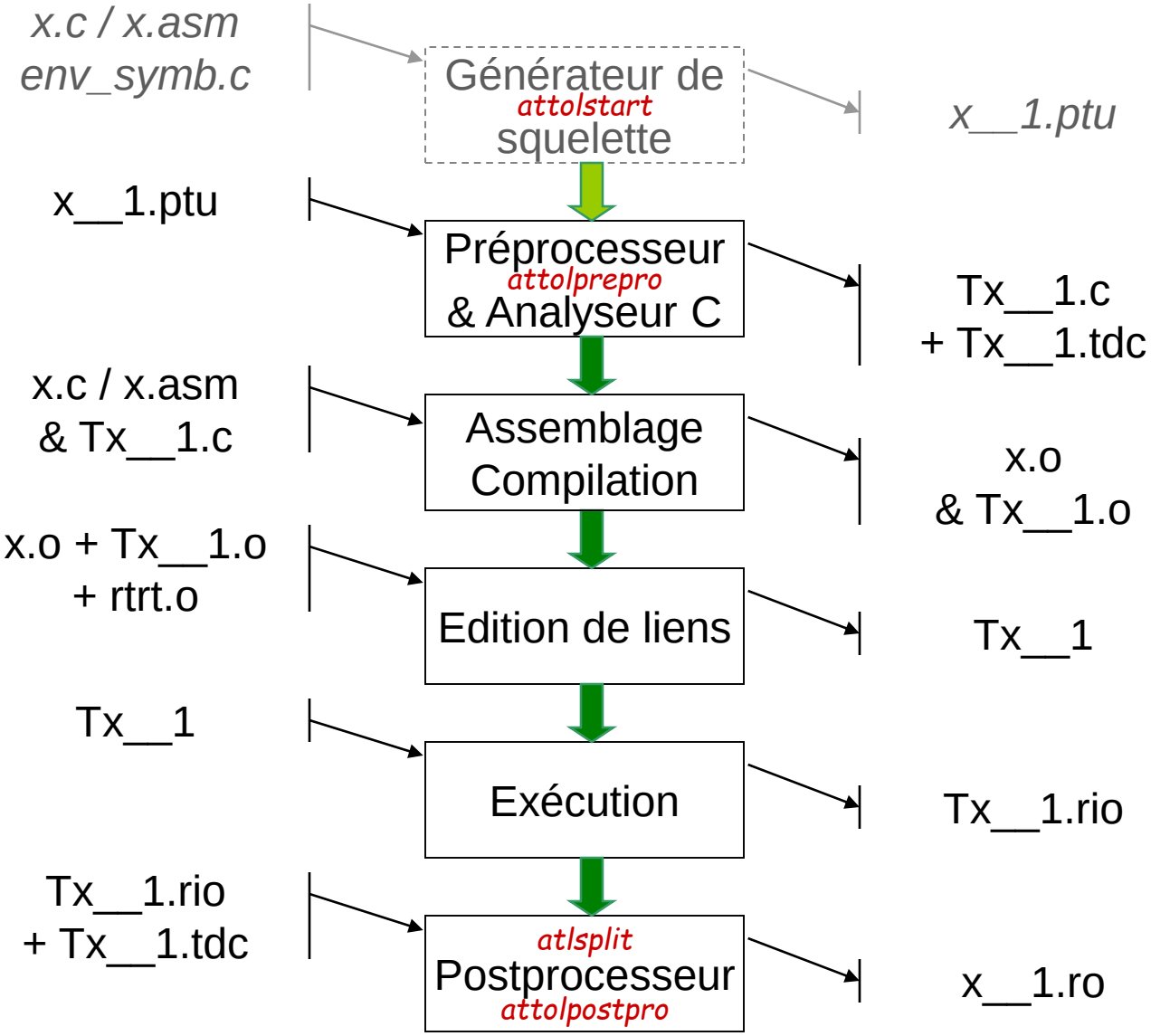
Stratégie de vérification des sources

- Intégration progressive des modules (avec services appelés réels ou simulés) pour valider les interfaces
- Possibilité de simuler les services appelés par des muets ou de réécrire les services dans le plan de test
- Possibilité d'instrumenter le code
  - pour intégrer des contrôles automatiques de données
  - pour vérifier la couverture structurelle du code
  - pour résoudre des problèmes de portabilité
- Le choix des valeurs en entrée du service repose sur la notion de classe d'équivalence : un test utilisant une valeur d'une classe est équivalent à un test traitant tout autre valeur de cette même classe
- Le code doit toujours s'exécuter sur machine cible

Principe d'utilisation



Flot de données pour le test unitaire



La nomenclature des noms des objets est donnée à titre indicatif

Langage de description de plan de test RTRT Component Testing

Types d'instruction

- Les instructions commencent par un mot-clé en majuscule (DEBUT, TEST, VAR, TAB, STRUCT, ENVIRONNEMENT, …)

- Les instructions en langage natif C
  - Déclarations des paramètres des services, des variables retour, des variables locales
    ➔ elles commencent par # et sont analysées
  - Autres instructions C
    ➔ elles commencent par # ou par @ si on ne veut pas les analyser

- Des commentaires
  - -- : aide à comprendre le plan de test
  - COMMENT : aide à comprendre les résultats de test

Structuration d'un plan de test

```
ENTETE x , 1 , 1
#int a;
#int ret_fct1;
#extern int fct1(int);
DEBUT
INITIALIATION
FIN INITIALISATION

ENVIRONNEMENT env_1
    VAR  a , INIT=0 , VA=INIT
FIN ENVIRONNEMENT

    SERVICE fct1
    UTILISE env_1
    TEST 1
        FAMILLE nominal
        ELEMENT
           #int y;
           VAR a, INIT=2, VA=3
           VAR y, INIT=0, VA==
           #asm("CALL fct1") ou #ret_fct1 = fct1(a);
        FIN ELEMENT
    FIN TEST 1

    TEST 2
        FAMILLE robustesse
        ELEMENT
         ...
        FIN ELEMENT
    FIN TEST 2
    FIN SERVICE

    SERVICE fct2
     ...
    FIN SERVICE
```

Déclaration des actions d'INITIALISATION au sein du plan de test (idem TERMINAISON)

Déclaration d'environnement utilisé au fil du plan de test

Décomposition en SERVICE, TEST et ELEMENT

Mise en place d'un environnement d'initialisation de la fonction à tester

Test des valeurs obtenues par rapport aux valeurs attendues

Test Boîte noire

Instructions de choix du moment d'exécution

DEBUT
INITIALISATION
  instructions RTRT
  instructions en langage C
FIN INITIALISATION
…..

Exécutées avant le premier test du premier service

…..
TERMINAISON
  instructions RTRT
  instructions en langage C
FIN TERMINAISON

Exécutées après le dernier test du dernier service

Instructions de mise en place d'environnement et de contrôles des données

VAR tableau_C, INIT=initialisation, VA=valeur attendue
TAB structure_C, INIT=initialisation, VA=valeur attendue
STR variable_C, INIT=initialisation, VA=valeur attendue

S'appliquent à n'importe quel type de variable
Les tableaux et les structures peuvent être traités

- soit globalement
- soit partiellement
  (plage de valeurs ou sous-ensembles de champs)
- soit poste par poste ou champs

Exemples

```
VAR x, ...
VAR y[4], ...
VAR z.champ, ...
VAR p->valeur, ...
TAB y[0..100], ...
TAB y, ...
STR z, ...
STR *p, ...
```

Instructions d'initialisation de variable

INIT = expression
INIT DANS (expr, expr,..)
INIT (variable) DANS/AVEC (expr, expr,..)
INIT DE expr A expr INCREMENT expr
INIT DE expr A expr NB_VALEUR nombre
INIT DE expr A expr NB_HAZARD nombre
INIT ==

Pour générer des sous-cas de tests poste à poste

Pour générer des sous-cas de tests par combinaison avec les valeurs de <variable>

Exemples

```
VAR x,          INIT = pi/4-1, ...
VAR y[4],       INIT DANS { 0, 1, 2, 3 }, ...
VAR y[5],       INIT(y[4]) AVEC { 10, 11, 12, 13
VAR z.champ,    INIT DE 0 A 100 NB_HASARD 3, ...
VAR p->valeur,  INIT ==, ...
TAB y[0..100],  INIT = sin(I1), ...
TAB y,          INIT = {50=>10,autres=>0}, ...
STR z,          INIT = {0, "", NIL}, ...
STR *p,         INIT = {valeur=>4.9, valide=>1}, ...
```

Instructions de contrôle de variable

VA = expression ou variable
VA = expr, DELTA tolérance
MIN = expr, MAX = expr
VA DANS (expr, expr, ...)
VA (variable) DANS/AVEC (expr, expr, ...)
VA ==

pas de génération automatique de résultats attendus

attention au cas de non test !

Prise en compte d'une tolérance pour les flottants
attendue * (1-10^{-6}) ≤ obtenue ≤ attendue * (1+10^{-6})

Exemples

```
VAR x, ...,          VA = pi/4-1
VAR y[4], ...,       VA DANS {0, 1, 2, 3}
VAR y[5], ...,       VA(y[4]) DANS {10, 11, 12, 13}
VAR z.champ, ...,    MIN = 0, MAX = 100
VAR p->valeur, ..., VA ==
TAB y[0..100], ..., VA = cos(I1)
TAB y, ...,          VA = {50=>10, autres=>0}
STR z, ...,          VA = {0, "", NIL}
```

Langue des mots-clés

- Les mots clés sont exprimés en français ou en anglais

FRANÇAIS	ANGLAIS
A	TO
AVEC	WITH
BOUCLE	LOOP
BORNES	BOUNDS
CODE	CODE
COMMENT	COMMENT
CONST	CONST
DANS	IN (*)
DE	FROM
DEBUT	BEGIN
DEFINIR	DEFINE
ELEMENT	ELEMENT
ENTETE	HEADER
ENVIRONNEMENT	ENVIRONMENT
FAMILLE	FAMILY
FIN	END
FORMAT	FORMAT
INCLURE	INCLUDE
INCREMENT	STEP
INIT	INIT
INITIALISATION	INITIALISATION

FRANÇAIS	ANGLAIS
MAX	MAX
MIN	MIN
MUET	STUB
NB HASARD	NB RANDOM
NB VALEUR	NB TIMES
PTU	PTU
SERVICE	SERVICE
SI	IF
SIMUL	SIMUL
SINON	ELSE
SINON SIMUL	ELSE SIMUL (*)
STR	STR
TAB	ARRAY
TERMINAISON	TERMINATION
TEST	TEST
TEST SUITE	NEXT TEST
TYPE SERVICE	SERVICE TYPE
UTILISE	USE
VA	EV (*)
VAR	VAR

Exemple de plan de test / Fonction add (1)

```
ENTETE add, ,
##include <stdlib.h>
##include <string.h>

DEBUT

#int add(int a , int b)
#{
#return (a+b);
#}

DEFINIR MUET STUB1
#int foo1(int _in param1)
#{
# return(param1);
#}
FIN DEFINIR

DEFINIR MUET STUB2
#int foo2(char _inout param1[100])
#{
# return(1);
#}
FIN DEFINIR

SERVICE add
TYPE_SERVICE externe

    #int a;
    #int b;
    #float fl;
    #int ret_add;
    #int pfoo2[10];
```

Exemple de plan de test / Fonction add (2)

```
TEST 1 -- TEST MULTIPLE AVEC MUET
FAMILLE nominal
   ELEMENT
   VAR a,           init dans {1,2,3},  va = init
   VAR b,           init = 3,           va = init
   VAR ret_add,     init = 0,           va ==
   MUET STUB1.foo1 (1)1, (3)3, (1)1, (1)1
   MUET STUB2.foo2 ((pfoo2,pfoo2))1
   #foo1(1);
   #foo1(3);
   #foo2(pfoo2);
   #foo1(1);
   #foo1(1);
   #ret_add = add(a, b);
   FIN ELEMENT
FIN TEST -- TEST 1

TEST 2 -- MULTIPLE TEST KO
FAMILLE nominal
   ELEMENT
   COMMENT  Ce test doit être KO
   VAR a,           init dans {1,2,3},  va = 0
   VAR b,           init = 3,           va = 0
   VAR ret_add,     init = 0,           va = 0
   #ret_add = add(a, b);
   FIN ELEMENT
FIN TEST
```

Exemple de plan de test / Fonction add (3)

```
TEST 3 -- INITIALISATION COMBINEE
FAMILLE nominal
   COMMENT  Ce test doit être OK avec 3 sous-cas de tests

   ELEMENT
   VAR a,           init dans {1,2,3},          va = init
   VAR b,           init(a) avec {3,2,1},       va = init
   VAR ret_add,     init = 0,                   va = 4
   #ret_add = add(a, b);
   FIN ELEMENT

FIN TEST -- TEST3

TEST 4 -- VALEUR ATTENDUE EN HEXA
FAMILLE nominal
   COMMENT  Ce test doit être OK

   ELEMENT
   VAR a, int#h,              init = 0AH ,      va = init
   VAR b, int#b,              init = 1B,        va = init
   VAR ret_add, int#h,        init = 0,         va = 0BH
   #ret_add = add(a, b);
   FIN ELEMENT

FIN TEST -- TEST 4
FIN SERVICE --add
```

Exemple de plan de test / Notations (1)

```
SERVICE no_service
TYPE_SERVICE nominal

#char   TAB1[10];
#char   TAB2[10];
#char   *p;
#double dd1,dd2;
#float  fl;
#float  fl2=1.0e-4;
#char   ch='a';
#unsigned int nonsig=100;
#unsigned int nonsig2=100;
#unsigned short sh=1;
--
#char bool;
#typedef struct {
# int a;
# char ch[10];
#} monstruct;
--
#monstruct tab[2];
--
#int *ptr;
#int matrice[20][20];
#int a1, a2, a3, a4;
--
#char ch1[10];
#char ch2[10];
--
```

Exemple de plan de test / Macro ET à 8 entrées (1)

```
SERVICE ET8
TYPE_SERVICE externe

    TEST TFONC1
    FAMILLE nominal
        ELEMENT

        VAR BE1, INIT = FALSE,             VA=init
        VAR BE2, INIT dans {FALSE, TRUE}, VA=init
        VAR BE3, INIT dans {FALSE, TRUE}, VA=init
        VAR BE4, INIT dans {FALSE, TRUE}, VA=init
        VAR BE5, INIT dans {FALSE, TRUE}, VA=init
        VAR BE6, INIT dans {FALSE, TRUE}, VA=init
        VAR BE7, INIT dans {FALSE, TRUE}, VA=init
        VAR BE8, INIT dans {FALSE, TRUE}, VA=init

        VAR BS1, INIT = TRUE,              VA=FALSE

        #Env_ET8(BE1,BE2,BE3,BE4,BE5,BE6,BE7,BE8,&BS1);

        FIN ELEMENT
    FIN TEST TFONC1
```

Exemple de plan de test / Macro ET à 8 entrées (2)

```
TEST TFONC2
    FAMILLE nominal
        ELEMENT

        VAR BE1, INIT = TRUE,              VA=init
        VAR BE2, INIT = FALSE,             VA=init
        VAR BE3, INIT dans {FALSE, TRUE}, VA=init
        VAR BE4, INIT dans {FALSE, TRUE}, VA=init
        VAR BE5, INIT dans {FALSE, TRUE}, VA=init
        VAR BE6, INIT dans {FALSE, TRUE}, VA=init
        VAR BE7, INIT dans {FALSE, TRUE}, VA=init
        VAR BE8, INIT dans {FALSE, TRUE}, VA=init

        VAR BS1, INIT = TRUE,              VA=FALSE

        #Env_ET8(BE1,BE2,BE3,BE4,BE5,BE6,BE7,BE8,&BS1);

        FIN ELEMENT
    FIN TEST TFONC2

[...]
```

Avec 8 tests TFONCi

⇒ 256 ou 64 sous-cas de tests générés automatiquement

⇒ la table de vérité du ET8 est couverte

```
ENTETE "I_St2_Date_Et_Heure__1","","
DEBUT
#/*****************************************************************/
#/*                    SPECIFIQUE AU TEST                         */
#/*****************************************************************/
#LCD_T_DateAndTime            DATE_AND_TIME;
INITIALISATION
FIN INITIALISATION
SERVICE I_St2_Date_Et_Heure__1
        TEST 1
        FAMILLE nominal
        # i = 0;
        ELEMENT 1
           -- Initialisation de la date and time
        VAR DATE_AND_TIME.year,                 INIT DANS {0, 12, 99},          VA = INIT
        VAR DATE_AND_TIME.month,                INIT DANS {1,  9, 12},          VA = INIT
        VAR DATE_AND_TIME.day,                  INIT DANS {1, 29, 31},          VA = INIT
        VAR DATE_AND_TIME.hour,                 INIT DANS {0, 15, 23},          VA = INIT
        VAR DATE_AND_TIME.minute,               INIT DANS {0,  3, 59},          VA = INIT
        VAR DATE_AND_TIME.second,               INIT DANS {0, 17, 59},          VA = INIT
        VAR DATE_AND_TIME.simulated,            INIT DANS {FALSE, TRUE},        VA = INIT
        FIN ELEMENT -- 1
        FIN TEST -- 1
    FIN SERVICE I_St2_Date_Et_Heure__1
```

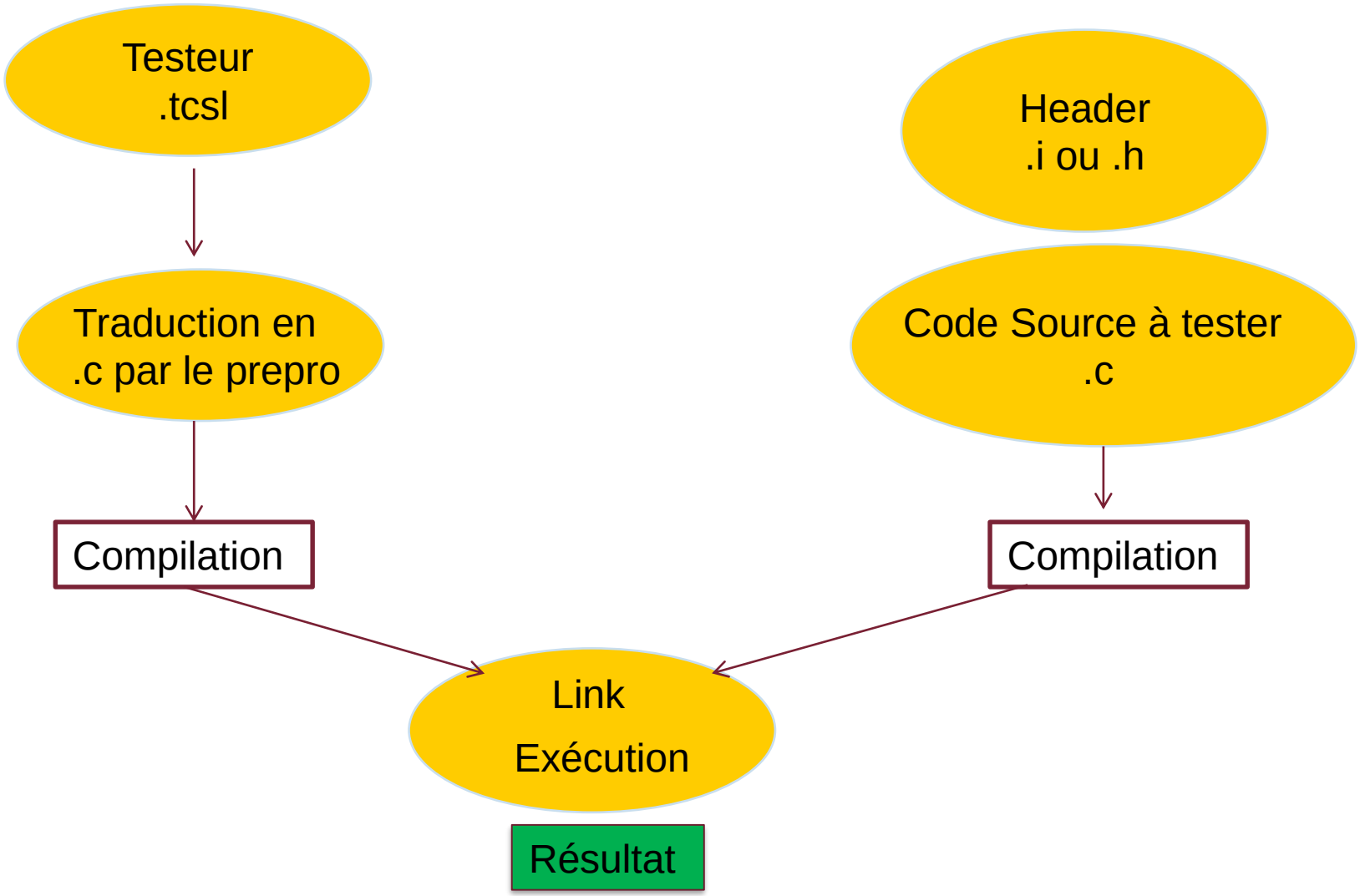
PHENIX

- Preprocessor Highly EfficieNt In linuX

- Suite TCSL => Test Contract Specification Language
- Obligations des deux côtés

- Testeur : INIT
- Application sous test: VA

ARCHITECTURE D'UN TEST



STRUCTURE D'UN PLAN DE TEST

```
PROGENV {%
%}
DEFSIMUL {%
Déclaration et définition en C pur.
Simulation de service
%}
DEFTEST ADD_Test (OPERATION: ADD, TESTNUM: 1){%
TestCase_1_2:
WHEN  {%
TESTVEC
%}
BEHAVIOR {%
ENSURES
%}
CALCTEST {%
%}
TestCase_1_3:
WHEN  {%
TESTVEC
%}
BEHAVIOR ENSURES {%
%}
%}
CALCTEST {%
%}
%}
RUNTEST {%    RUN; %}
```

HEADER

PROGENV {%

%}

DEFSIMUL {%

Déclaration et définition en C pur.

Simulation de service

%}

```
DEFTEST ADD_Test (OPERATION: ADD, TESTNUM: 1){%
WHEN  {%
TESTVEC
Type <var> = val init;
%}
BEHAVIOR {%
ENSURES
Type <var> == val_attendue;
%}
CALCTEST {%
/* appel du service sous test avec les bons paramètres*/
%}
……..
%}
RUNTEST {%    RUN; %}
```

Variables locales à un test:

LOCAL {% int $i_0, $i_1; ... %}

Tableau:

FORALL ($i_0: 7 <= $i_0 <= 8)

RM_E2PROM_PN[$i_0].INFO_PN_SN == \old
  RM_E2PROM_PN[$i_0].

```
DEFTEST add (OPERATION: add, TESTNUM: 1.1)
{%
TestCase_1_1:
WHEN{%
TESTVEC
short int r = 0;
short int a = 2;
short int b = 3;

%}/*WHEN*/
BEHAVIOR PrePostContract_1_1 {%
ENSURES
r == 5;
a == 2;
b == 3;
%}/*BEHAVIOR*/
CALCTEST{%
r=add(a,b);
%}/*CALC*/
RUNTEST {%    RUN; %}
```

Présenté par

Tahiry Ratsiambahotra

FORMATION OUTILS DE VERIFICATION

RATIONAL TEST REAL TIME

SYSTEM TESTING

Attention : cette formation est une présentation générale du produit. Se reporter à la documentation fournisseur pour obtenir des informations plus détaillées

Structuration d'un script de test

```
## include "api.h"
INITIALIZATION init_proc()
TERMINATION end_proc()
EXCEPTION recover_proc()
DECLARE_INSTANCE simulator1
SCENARIO main [ LOOP n]
FAMILY <family name>, ...

    SCENARIO test_case1

    END SCENARIO -- test_case1

    SCENARIO test_case2

        INSTANCE simulator1

        END INSTANCE -- simulator1

        INSTANCE simulator2

        END INSTANCE -- simulator2

    END SCENARIO -- test_case2

END SCENARIO -- main
```

Déclarations des données et des fonctions utiliées au sein du script de test

Initialisation,terminaison et récupération sur erreur par niveau de scénario

Décomposition en SCENARIO
sous SCENARIOS
…

Un scenario peut être divisé en plusieurs INSTANCE pour définir des comportements asynchrones (comme un filtre)

Envoi d'un message/événement par SEND

SEND (<message>, <canal>)

- Envoi d'un message complet sur un canal de communication
- Affichage des caractéristiques du message envoyé dans le rapport de test
- Le message doit préalablement être déclaré par **MESSAGE** et initialisé par **DEF_MESSAGE**
- Permet d'envoyer un message sur plusieurs canaux de communication au travers des couches d'adaptation (**PROCSEND**)

Récupération d'un événement par WAITTIL

WAITTIL (<OK_Expression>, <KO_Expression>)

- Attend deux expressions booléennes composées d'événements
  si <KO_Expression> est VRAI => le scénario est KO (exception)
  si <OK_Expression> est VRAI => le scénario est OK

- Permet d'attendre une combinaison de plusieurs évènements (chacun déclaré par **MESSAGE** et initialisé dans **DEF_MESSAGE**)

- Une expression booléénne (ou un événement) peut être encapsulée par **MATCHING**, **NOTMATCHING**, **MATCHED**, **NOTMATCHED**

- La variable **WTIME** est un timeout (incrémentée à chaque déblocage d'attente événementielle) pour éviter les bloquages

Encapsulation d'événement par MATCHING

MATCHING(<*message*> {[, <*canal_de_com*>)]})

• Retourne VRAI si le dernier message reçu durant le **WAITTIL** correspond au message attendu. Si on spécifie un canal, retourne vrai seulement si le message a été reçu sur ce canal de communication

Exemple :

```
CHANNEL ux_socket: ch
SCENARIO Main
DEF_MESSAGE msg_1, EV={100,10}
DEF_MESSAGE msg_2, EV={200,20}
...
WAITTIL(MATCHING(msg_1) || MATCHING(msg_2,ch),WTIME==10)
```

Encapsulation d'événement par MATCHED

MATCHED (<*message*> {[, <*canal_de_com*>)]})

• Retourne VRAI si un des messages reçus durant le **WAITTIL** correspond au message attendu. Si on spécifie un canal, retourne vrai seulement si le message a été reçu sur ce canal de communication

Exemple :

```
CHANNEL ux_socket: ch
SCENARIO Main
DEF_MESSAGE msg_1, EV={100,10}
DEF_MESSAGE msg_2, EV={200,20}
...
WAITTIL(MATCHED(msg_1) || MATCHED(msg_2,ch),WTIME==10)
```

Encapsulation d'événement par NOTMATCHING

NOTMATCHING (<*message*> {[, <*canal_de_com*>)]})

• Retourne VRAI si le dernier message reçu durant le **WAITTIL** **ne correspond pas** au message attendu. Si on spécifie un canal, retourne vrai seulement si le dernier message n'a pas été reçu sur ce canal de communication

Exemple :

```
CHANNEL ux_socket: ch
SCENARIO Main
DEF_MESSAGE msg_1, EV={100,10}
DEF_MESSAGE msg_2, EV={200,20}
...
WAITTIL(WTIME==10, NOTMATCHING(msg_2,ch))
...
IF (NOTMATCHING(msg_2,ch)) THEN
…
```

Encapsulation d'événement par NOTMATCHED

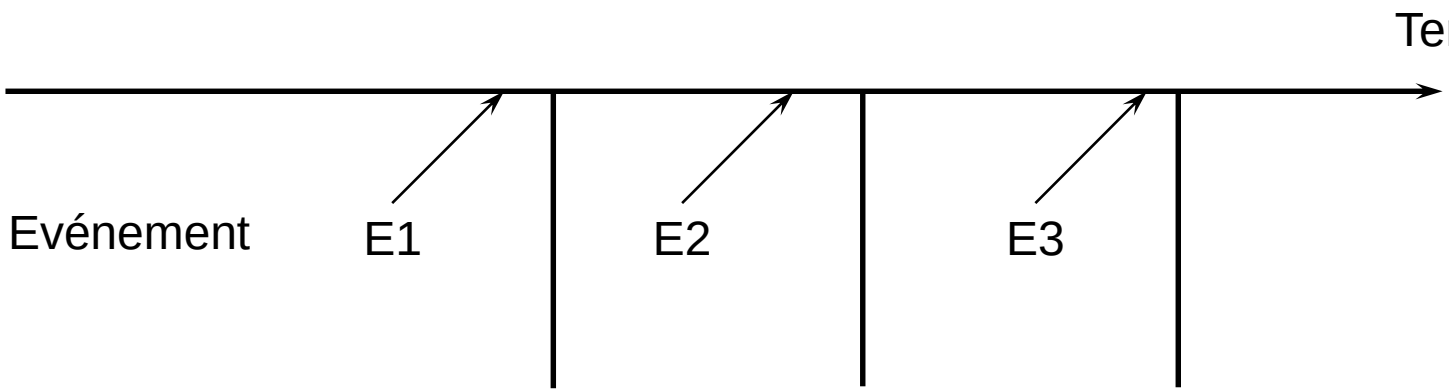
NOTMATCHED (<*message*> {[, <*canal_de_com*>)]})

• Retourne VRAI si un des messages reçus durant le **WAITTIL** **ne correspond pas** au message attendu. Si on spécifie un canal, retourne vrai seulement si le message n'a pas été reçu sur ce canal de communication

Exemple :

```
CHANNEL ux_socket: ch

SCENARIO Main

DEF_MESSAGE msg_1, EV={100,10}

DEF_MESSAGE msg_2, EV={200,20}

...

WAITTIL(WTIME==10, NOTMATCHED(msg_1))

...

IF (NOTMATCHED(msg_1,ch)) THEN

...
```

Gestion des (NOT)MATCH(ING/ED)



MATCHING(E2)	FAUX	VRAI	FAUX
NOTMATCHING(E2)	VRAI	FAUX	VRAI
MATCHED(E2)	FAUX	VRAI	VRAI
NOTMATCHED(E2)	VRAI	VRAI	VRAI
NOTMATCHED(E1)	FAUX	VRAI	VRAI

Initialisation des messages par DEF_MESSAGE

DEF_MESSAGE <message>**, EV =** <expression>

- Initialise les champs du message
- Le message doit être déclaré par **MESSAGE**
- Utilisation de “**VAR** <variable>, **EV** =expression” est possible

Exemple :

```
...
DEF_MESSAGE Request, EV={code=>'R'}
DEF_MESSAGE Ack, EV={code=>['A‘..’Y’]}
DEF_MESSAGE Request, EV={code=>'R', nom=>"ALL"}
```

Déclaration des messages par MESSAGE

MESSAGE <C_type> : <msg_1> [, <msg_2, ..., msg_n]

- Déclare le message/événement au travers d'une variable événementielle à laquelle on associe une structure de données type C

Exemple dans le fichier d'interface :

```
typedef struct {
  struct {
          int data_size;}
header;
    struct {
    data[8192];
      } data;
} Message_t;
```

Exemple dans le .pts

MESSAGE Message_t: msg_1

SCENARIO first

DEF_MESSAGE msg_1, **EV**= {data {data => « FlightReport »} }

WAITTIL(**MATCHING**(msg_1), WTIME == 100)

...

IF (msg_1.header.data_size == 4) **THEN**

...

- Comment vérifier si on veut recevoir un msg_1?

- DEF_MESSAGE msg_1, EV= {header=>{data_size =>4},data=>{data=>« aircraft »} }
-
- WAITTIL(MATCHED(msg_1),WTIME==10)

- Comment vérifier si on veut recevoir tout événement sauf un msg_1?

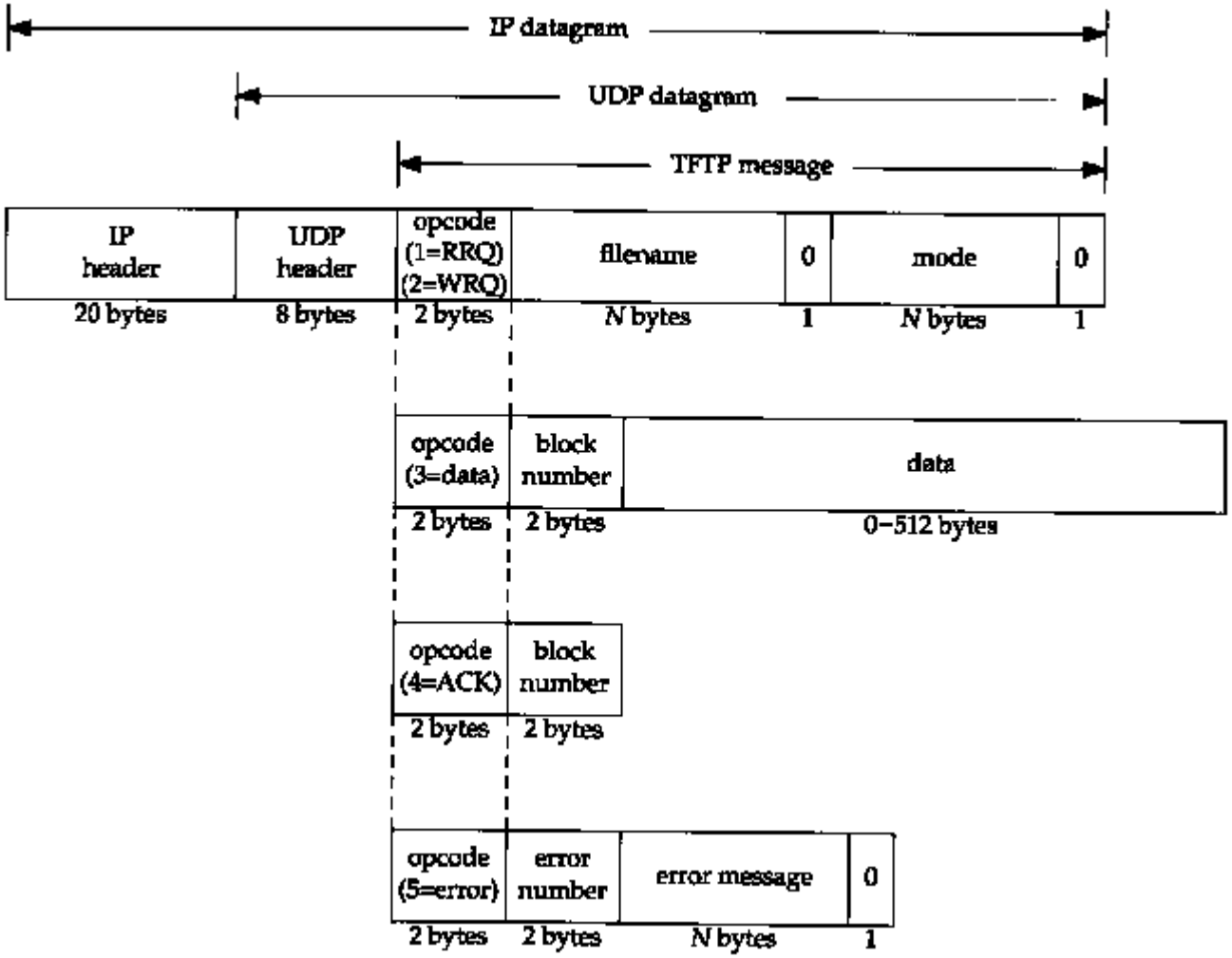
- WAITTIL(NOTMATCHED(msg_1),WTIME==10)

NOTION DE PROTOCOLE TFTP

- Protocole ex TFTP
- Le Client sans la réponse du serveur ne peut pas attendre indéfiniment.

- Paquets
- WRQ (demande d’écriture de fichier)
- ACK acquittement
- DATA bloc n° 1, 2, 3 de 512 octets
- ACK bloc n° 1, 2, 3 de 512 octets
- A la fin quand le block est <512 on sait que c’est la fin.
- On attend 3 sec avant de terminer.
- Timeout 2s.

TFTP encodage



Exos:

- Utiliser RTRT system test pour envoyer un fichier de 2000 octets en mode client.

Messages types

- VAR Wrq.header.data_size,INIT=17
- . VAR Wrq.data.data,INIT ={
- & /* Trame Wrq */
- & 0x00,0x02,
- & /* Nom fichier: "Instance" */
- & 0x49,0x6e,0x73,0x74,0x61,0x6e,0x63,0x65,0x00,
- & /* Mode : "Octet" */
- & 0x6f,0x63,0x74,0x65,0x74,0x00}

-
- -- Receptiond e l'Acquittement du message
- VAR Ack.data, EV={header=>{data_size=>4},data=>{data=>{& /* Trame Oack */
- & 0x00,0x04,
- & /* Option : "Num block" */
- & 0x00,0x00}
-

AIRBUS

AN EADS JOINT COMPANY
WITH BAE SYSTEMS