IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Interaction et Vérification

A. Beugnard

IV – C3

2017

On a vu :

- des interfaces pour décrire des points d'interaction
- des interactions simples avec 2 acteurs (émetteur, récepteur, tuple space)
- des interactions simples avec rôle unique (diffusion, barrière de synchronisation)

Comment décrire des interactions avec beaucoup de rôles différents, des acteurs qui entrent et qui sortent de l'interaction ?

- Des contrats pour les interfaces
- Des langages pour les interactions
- Des diagrammes pour les interactions

Des interactions plus abstraites :

- Publish/subscribe
- Négociation
- Vote
- Enchères
- …
- Abstractions de communication

- Les acteurs (leur rôle)
- Pour chaque rôle le contrat d'interface
  - (syntaxte) signature des services offerts et requis
  - (sémantique) précondition et postcondition de chaque service
  - (synchronisation) règles d'usage temporel des services
  - (QoS) qualité de service : performance, disponibilité, sécurité, etc
- Les interactions entre rôle

Contrat de synchronisation

Description locale (à un rôle) des règles d'utilisation de ses services.

- automate (hypothèse, chaque service est atomique)
- interface automata [HdA01]
- *Counter variables* [Ger77]
- Une thèse de 1995 [McH95]

Idée : des automates qui distinguent les messages entrants, sortants et internes avec des règles de composition précises.

Idée : des compteurs sont associés aux entrées et sorties de chaque méthode. On sait combien de clients sont en cours de traitement pour chaque méthode.

Les compteurs sont utilisés pour exprimer des conditions d'accès à une méthode

Par exemple, pour des méthodes `read` et `write` :

- readers = #begin(read) - #end(read) ;
- writers = #begin(write) - #end(write) ;

Stack $\{size \geq 0\}$

`write(o: Object)`
$\{pre : readers = 0 \wedge writers = 0\}$
$\{post : (size = size@pre + 1) \wedge (head() = o)\}$
`read() : Object`
$\{pre : size > 0 \wedge (readers = 0 \wedge writers = 0)\}$
$\{post : size = size@pre - 1 \wedge result = head()@pre\}$
`head() : Object`
$\{pre : readers = 0 \wedge writers = 0\}$
$\{post : size = size@pre\}$

Lifecycle = (`read` + `write` + `head`)*

Interaction entre rôles

En UML2.5 (spec `http://www.omg.org/spec/UML/2.5/`) :

- Collaboration (Classifier) – Ch 11.7 pp213-217 (5p)
- Diagrammes d'état (State Machine) – Ch 14 pp303-370 (68p)
- Actions – Ch 16 pp440-562 (123p)
- Diagrammes d'activité (Activity) – Ch 15 pp371-438 (68p)
- Interaction – Ch 17 pp 563-636 (74p)
  - Diagrammes de séquence (Sequence) Ch 17.8 pp593-596 (4p)
  - Diagrammes de communication (Communication) Ch 17.9 pp597-598 (2p)
  - Diagrammes d'interaction (Interaction Overview) – Ch 17.10 pp 599-600 (2p)
- InformationFlows – Ch 20 pp 667-674 (13p)

Exemple :
`http://www.agilemodeling.com/essays/umlDiagrams.htm`

- Interaction entre instances
- Création, destruction d'instance
- Synchrone, asynchrone, temporisé (pas future)
- Fragments (alternative, case, optional, while, *negation*, etc.)
- Ne montre pas les fils d'activités

Exemple : `http://uml.free.fr/cours/i-p19.html` et `http://agilemodeling.com/artifacts/sequenceDiagram.htm`

- Interactions entre instances ou classes/interfaces (rôles) ; invariants
- Vision abstraite (utile pour des *Design Patterns*)
- Vision détaillée (instances) et échanges
  - ordre des messages
  - synchrone, asynchrone, diffusion sequentielle ou parallèle
  - Pas de propriété (ex. uniformité, groupe membership) [voir cours précédent]

Exemple : `http://uml.free.fr/cours/i-p12.html` et `http://uml.free.fr/cours/i-p18.html` ou `http://agilemodeling.com/artifacts/communicationDiagram.htm`

- États et transitions d'un objet, d'un système, d'une machine
- Les états sont des machines (récursivement)
- Les états déclenchent des actions (à l'entrée, la sortie)
- Les transitions peuvent être gardées et déclenchent des actions
- On peut introduire de la concurrence et des barres de synchronisation
- Pas facile de séparer des rôles (représente l'évolution des états d'un objet - ou d'un système - après composition)

Exemple : `http://uml.free.fr/cours/i-p20.html` ou `http://agilemodeling.com/artifacts/stateMachineDiagram.htm`

- Décrit des flots d'activité et de données
- On peut introduire de la concurrence et des barres de synchronisation
- On peut partitioner par rôle
- Dynamique des rôles absente
- Multiplicité et propriétés de communication peu visibles

Exemple : `http://uml.free.fr/cours/i-p21.html` ou `http://agilemodeling.com/artifacts/activityDiagram.htm`

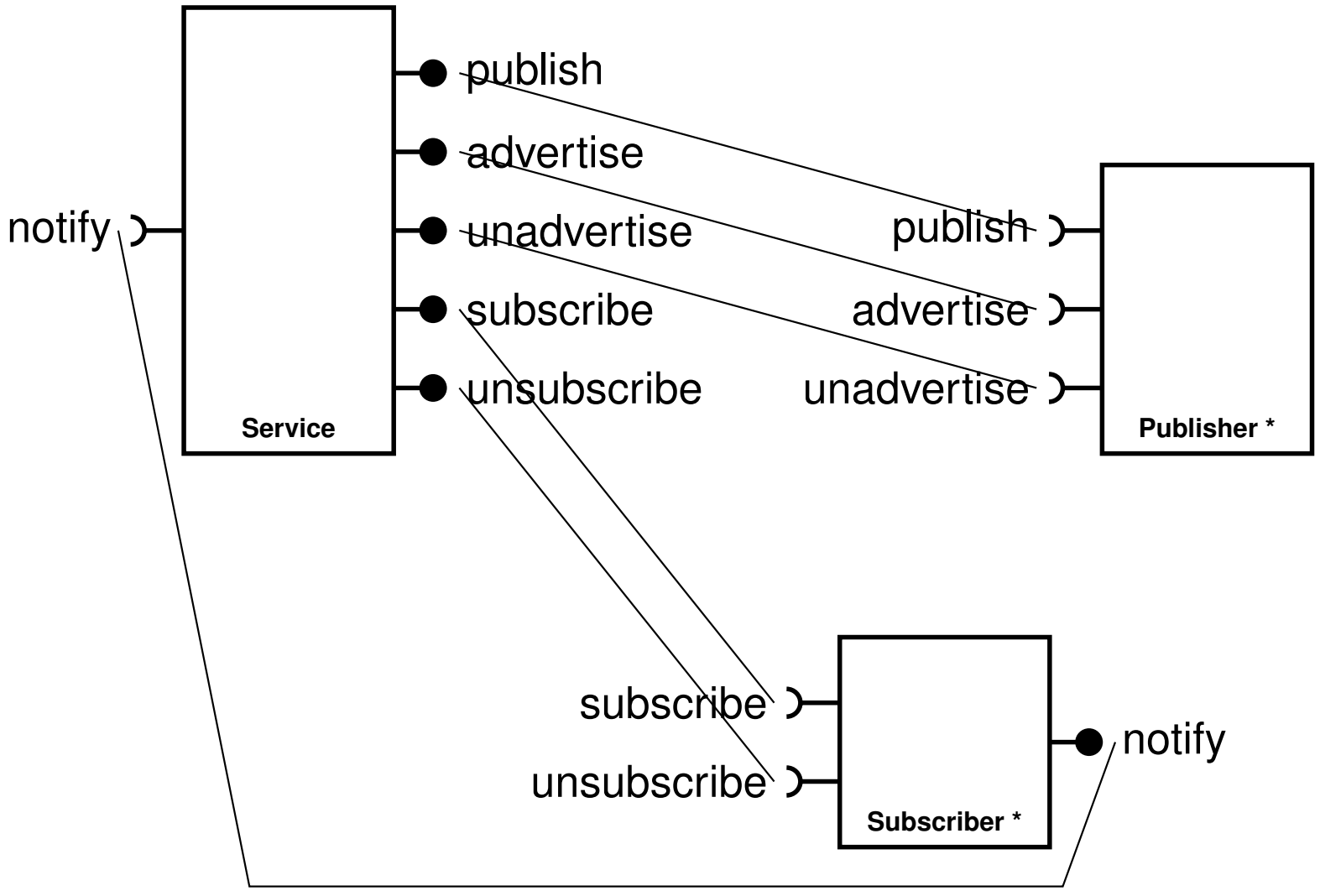
Un mélange des diagrammes de séquence et d'activité en permettant la connexion de fragments.

Exemple : `http://agilemodeling.com/artifacts/interactionOverviewDiagram.htm`

Publish/Subscribe

Démarche

- identifier les rôles
- pour chaque rôle décrire les interfaces
- assembler l'ensemble
  - décrire les interactions
  - décrire les propriétés voulues


publish
advertise
unadvertise
subscribe
unsubscribe
notify
Service
publish
advertise
unadvertise
Publisher *
subscribe
unsubscribe
Subscriber *
notify

View Given an event e_i, its view, namely $V(e_i)$, is defined as the set of all interested subscribers, i.e., those subscriber processes that are active (i.e., in a running state) when event e_i has been published and have at least one active subscription satisfied by e_i.

Σj the set of storing subscriptions, also known as the subscription table, related to the j-th node in the publish/subscribe service.

A process p_j performs a series of four operations by interacting with *Service*. Such operations can be formally expressed by means of the low-level communication primitives of sending a notification, namely send, and receiving a notification, namely rcv, as follows :

1. Publish : an event e_i is published if p_j sends its notification to *Service*, even if its view (the subscribers) may be empty ;
2. Notify : p_j is notified by *Service* of a published event e_i when e_i is received by *Service*, and one of the subscriptions of p_j contained in Σj is satisfied by e_i ;
3. Subscribe : A new subscription $C_{j,k}$ is created by p_j and stored in *Service* ;
4. Unsubscribe : An existent subscription $C_{j,k}$ is erased from the subscription table of *Service*.

For completeness we have also drawn two additional operations at the publisher side. (. . .) it is not expressed in this specification.

L'article cite 2 propriétés, mais ne donne aucun détail :

- (ii) receiving and buffering events ;
- and (iii) dispatching the received notifications to the interested subscribers.

Qu'en pensez-vous ?

- Contrat des interfaces ?
- Groupes de subscribers ?
- Garanties de diffusion des notifications ?
- Histoire et/ou perte d'événements ?
- . . .

Un autre article qui décrit formellement un Publish/Subscribe.
`http://www.sciencedirect.com/science/article/pii/S0164121209002350`

Utilise la méthode Z.

S'intéresse à l'initiative des processus (mode push ou mode pull).

Leur conclusion (partielle) :

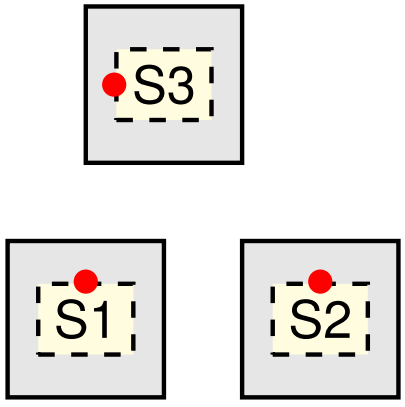
> *Our formalization allows us to characterize key structural properties of the Publish/Subscribe architectural style formally and at a high level of abstraction.*

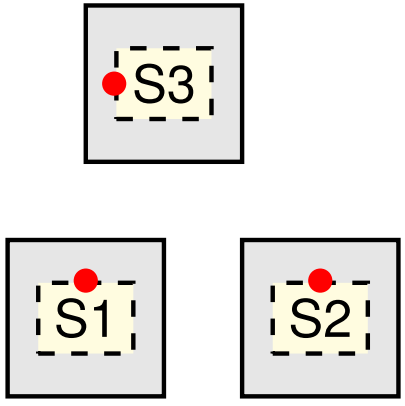
Une thèse (Master) qui prend en compte les aspects interaction.

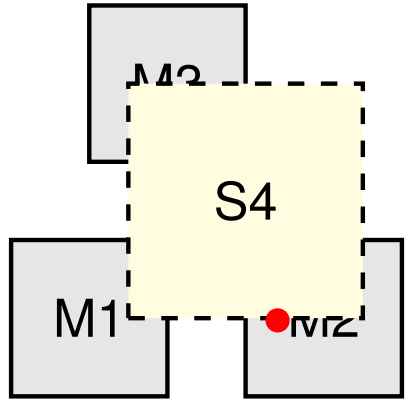
Mais partiellement. . .

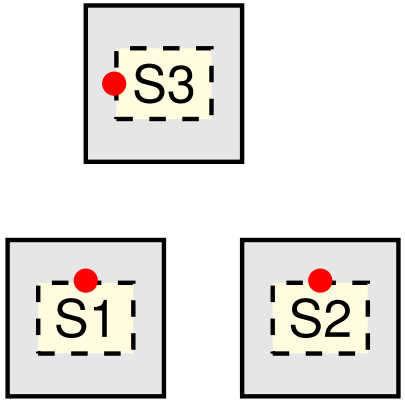
Rien sur la diffusion et ses propriétés. . .

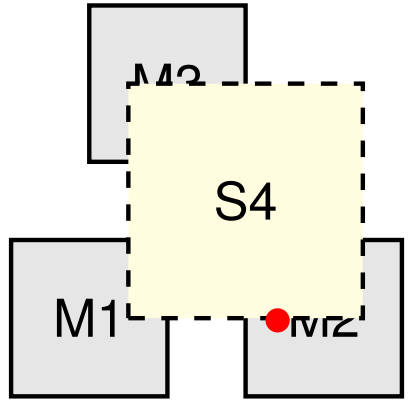
Les propriétés sont-elles préservées quelque soit la taille du système ?

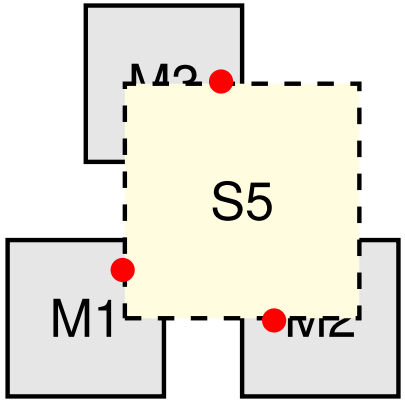

S3
S1
S2

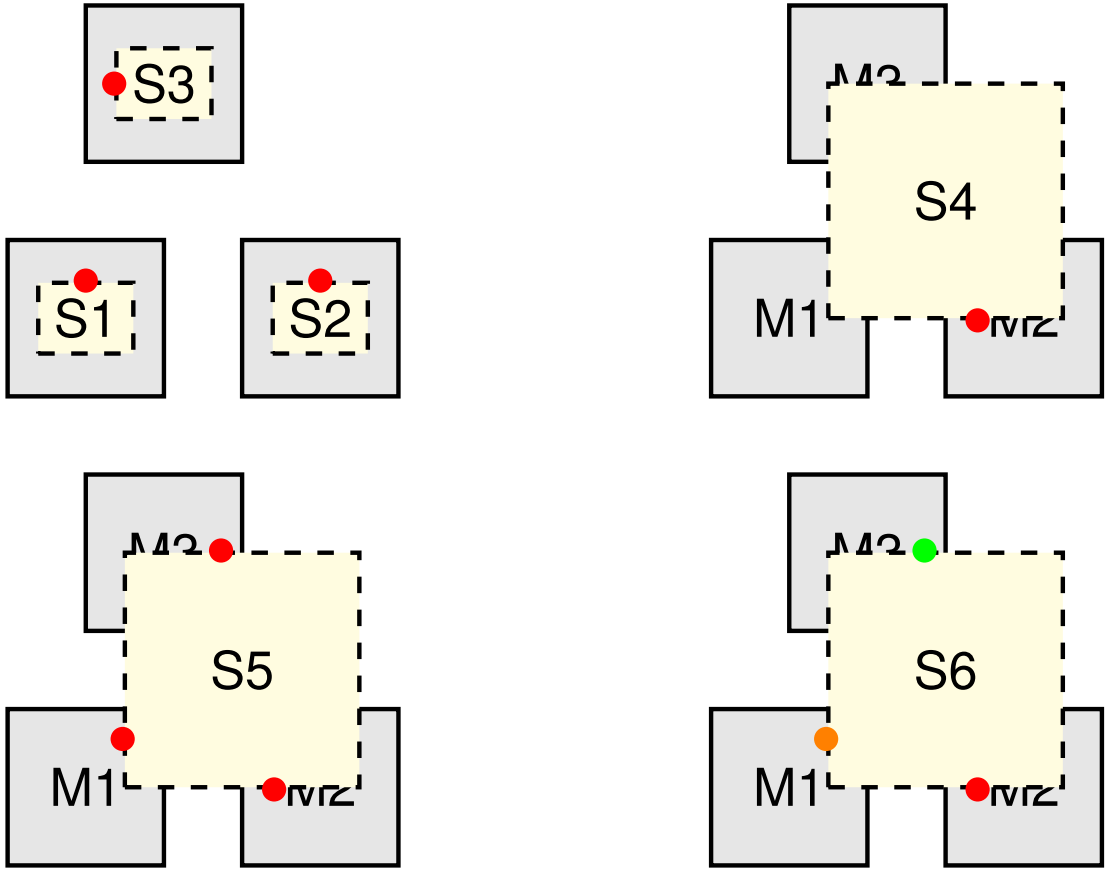

S3
S1
S2


S4
M1


S3
S1
S2


S4
M1


S5
M1


S3
S1
S2
S4
M1
S5
M1
S6
M1

Conclusion

Pour décrire une interaction :

- Identifier les rôles
- Pour chaque rôle, décrire les contrats d'interfaces (4 niveaux)
- Décrire l'interaction - les échanges, la collaboration -
- Décrire les propriétés (hypothèses et exigences)

Pour valider une interaction :

- Exploiter la description (On utilise souvent des automates)
- Exprimer les propriétés attendues (On utilise souvent de la logique temporelle)
- Faites des preuves, du *model checking*

Les prochains cours. . .

Vote

Exercice. . .
À rendre dans 8 jours (1 à 2 pages commentées)

References I

Christian Esposito, Domenico Cotroneo, and Stefano Russo.

On reliability in publish/subscribe services.

Computer Networks, 57(5) :1318–1343, April 2013.

A J Gerber.

Process Synchronization by Counter Variables.

Operating Systems Review, 1977.

T Henzinger and Luca de Alfaro.

Interface automata.

ACM SIGSOFT Software Engineering . . ., 2001.

Akshat Kumar.

Design and validation of a context-aware publish-subscribe model.

Master's thesis, University of Waterloo, 2015.

Imen Loulou, Mohamed Jmaiel, Khalil Drira, and Ahmed Hadj Kacem.

P/s-com : Building correct by design publish/subscribe architectural styles with safe reconfiguration.

Journal of Systems and Software, 83(3) :412–428, March 2010.

C McHale.

Synchronisation in concurrent, object-oriented languages : Expressive power, genericity and inheritance.

PhD thesis, 1995.