Compiling Fondue Analysis: From Fondue to Netsilon (A case study)

Frédéric FONDEMENT

frederic.fondement@epfl.ch

Software Engineering Lab

Swiss Federal Institute of Technology Lausanne

Switzerland

1/20/2004

Contents

- Introduction
- Netsilon overview
- Awaited interface
- Static aspects
- Dynamic aspects
- Conclusion

Contents

- Introduction
  - Prototyping Fondue analysis
  - Translating Fondue to Netsilon
  - Case study: part of IHB logging
- Netsilon overview
- Awaited interface
- Static aspects
- Dynamic aspects
- Conclusion

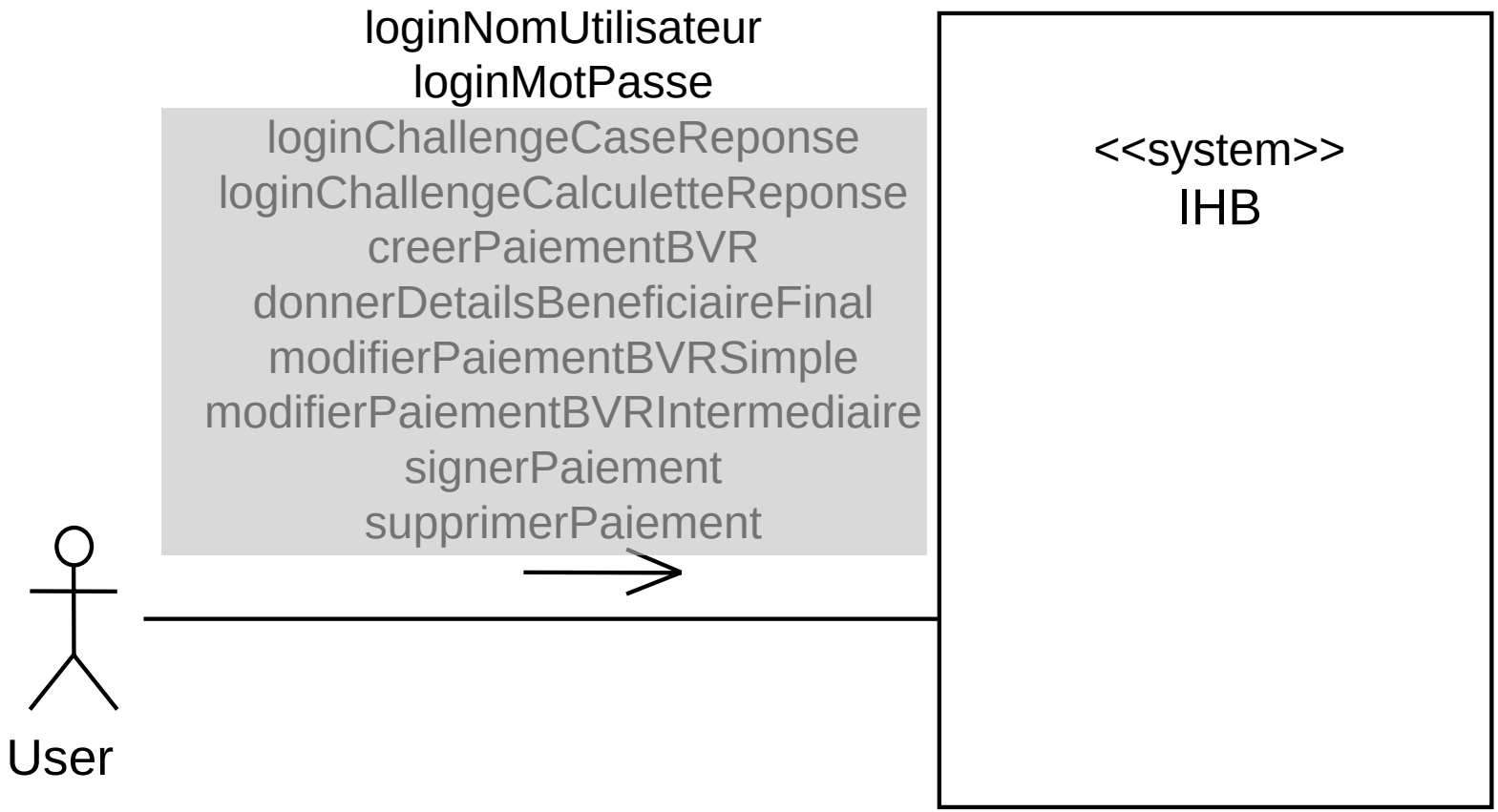
Prototyping Fondue Analysis

- Goal : simulate a Fondue analysis (FA)
- Problem
  - Is that possible
  - What is the exact meaning of specific Fondue elements
- Exploration : try to translate a FA into a well defined executable language (some kind of naïve implementation)
  - To find most important problems to solve
  - To find easiest solutions
  - Not to find an implementation transformation
- Initial goal : mapping messages input / output to real life web application
  - Needs more exploration at this point…

Translating Fondue to Netsilon

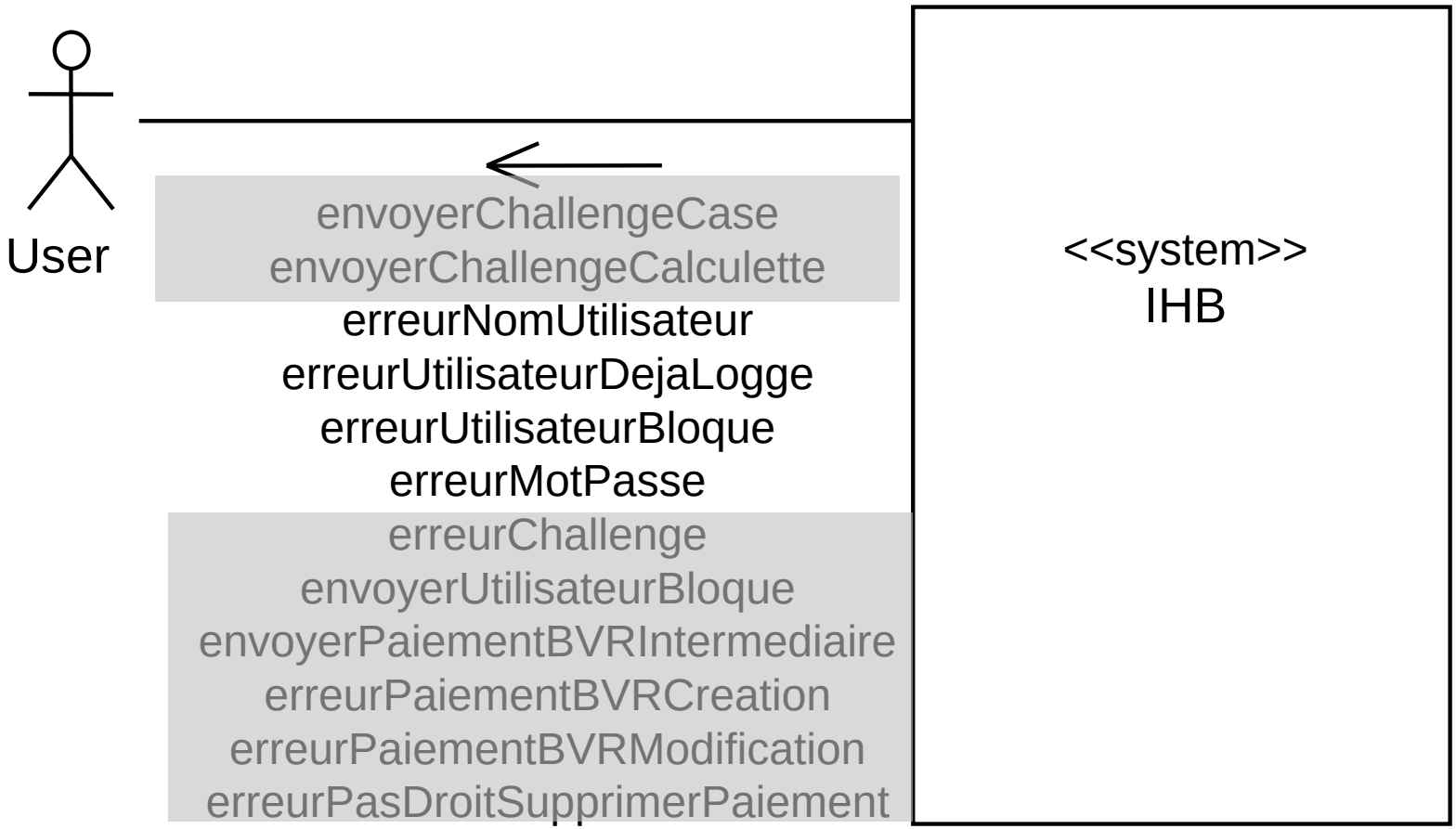
- Netsilon is a web-based application generator
- Why Netsilon :
  - It is model driven (knows what to do with class diagrams)
  - It includes an imperative language but very near from OCL expressions (Xion)
  - In that it is web-based, it generates reactive systems
  - In that it is web-based, it generates server with multi-clients applications
  - Despite it is web-based, it generates persistent data state (including session specific data)
- Problems :
  - Does not know about state-machines
  - Messages sending / receiving management is just HTML presentation / link or form invocation

Case Study : part of logging activity



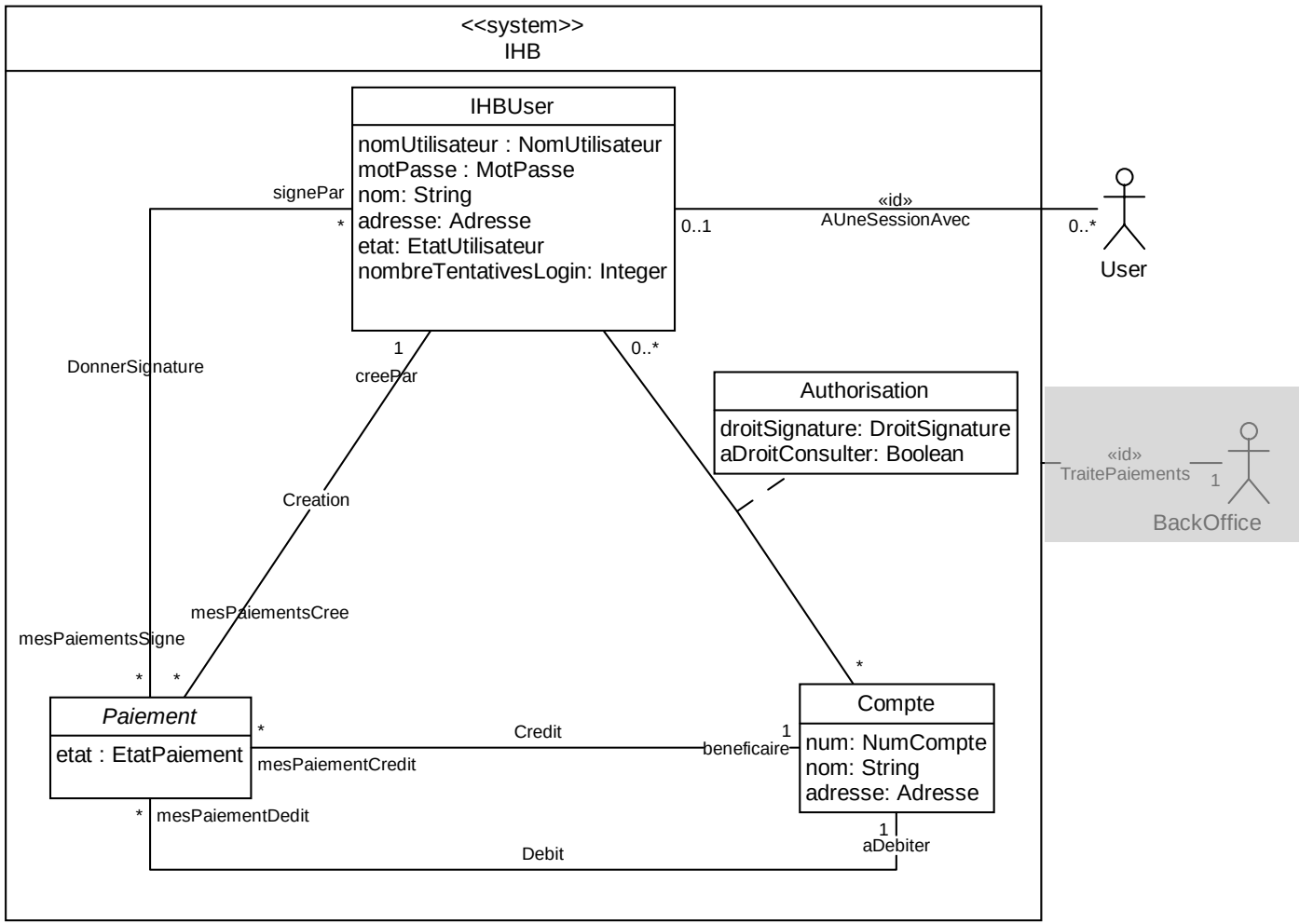
©Shane Sendall for Unicible

Case Study : part of logging activity

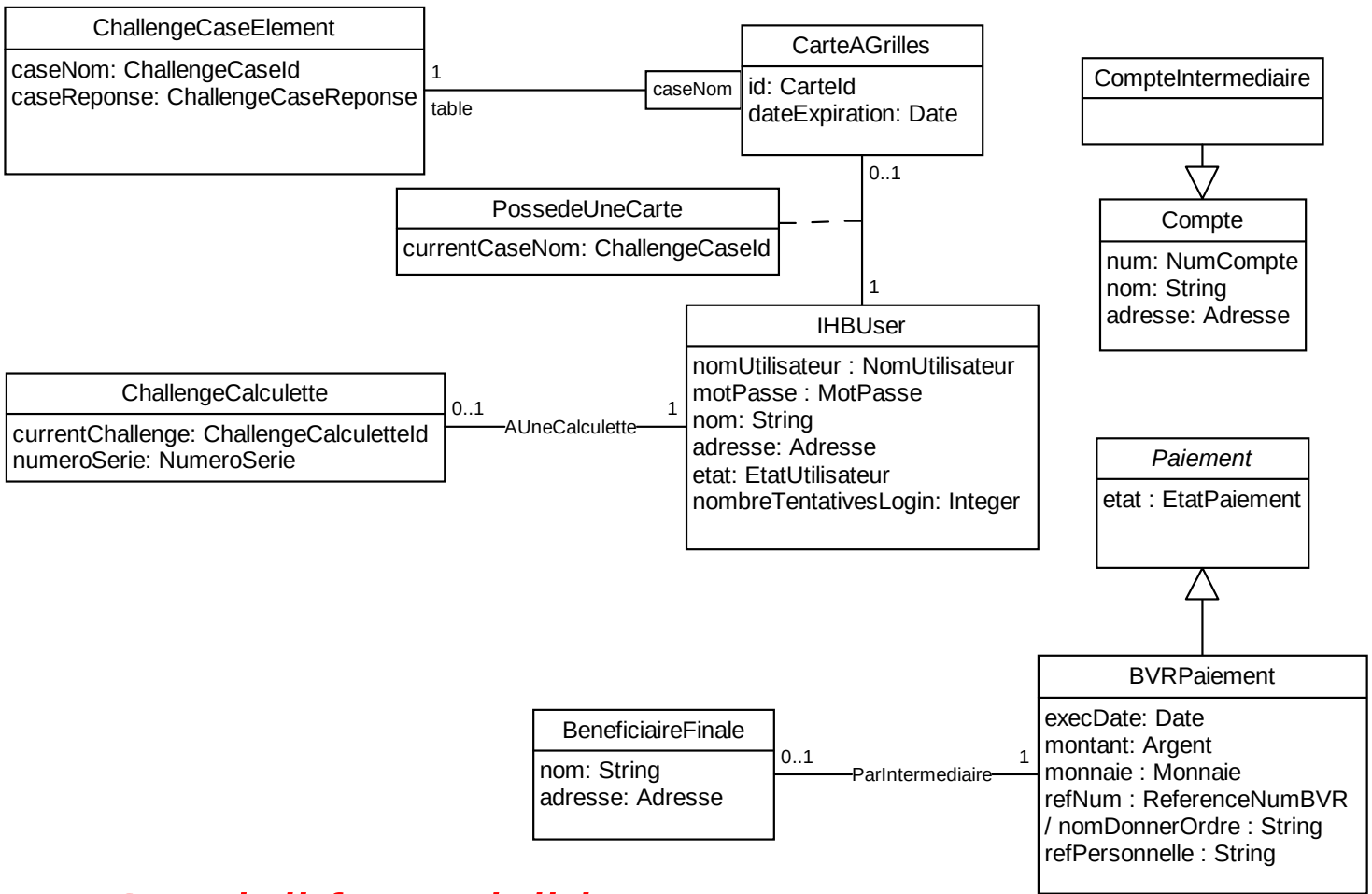


©Shane Sendall for Unicible

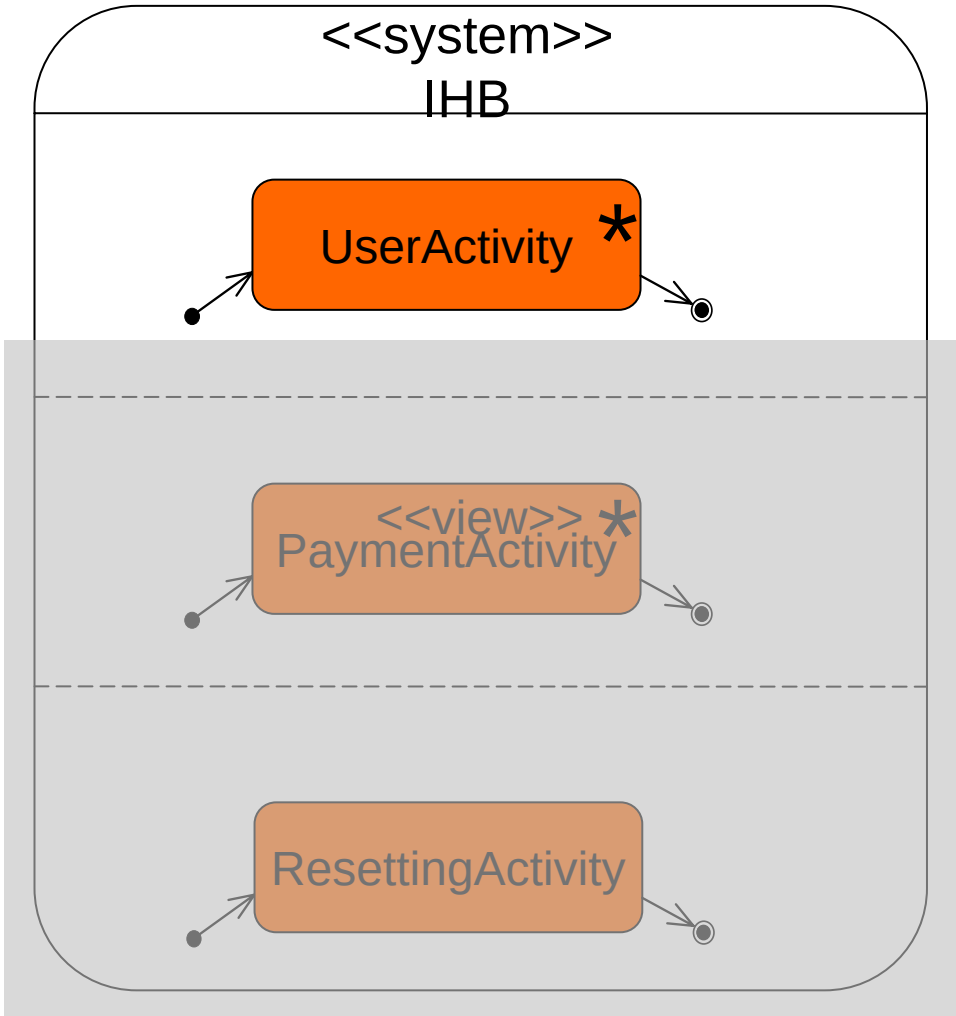
Case Study : part of logging activity



Case Study : part of logging activity

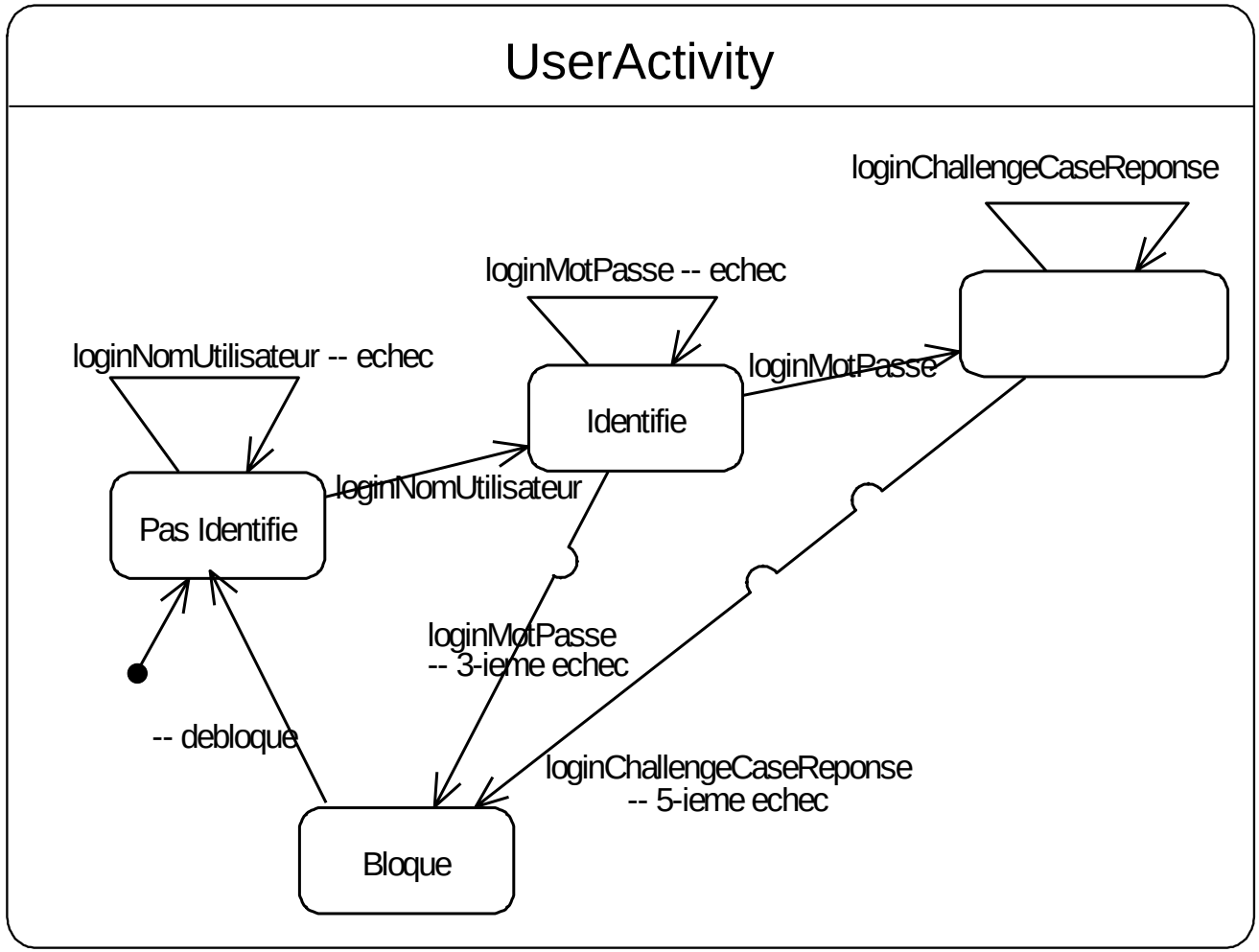


Case Study : part of logging activity



©Shane Sendall for Unicible

Case Study : part of logging activity



©Shane Sendall for Unicible (partial and corrected)

Case Study : part of logging activity

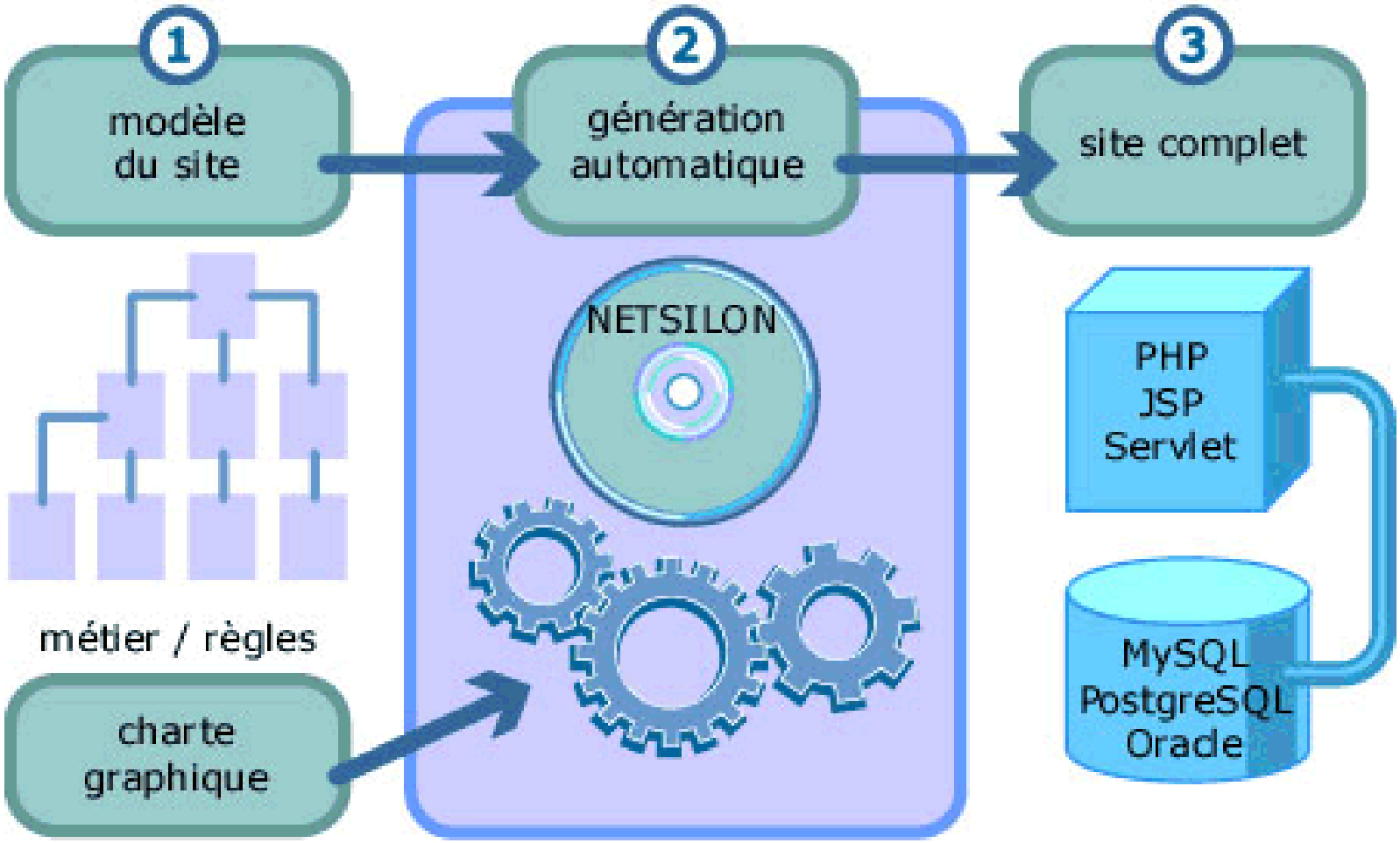
- Invariants
  - nomUtilisateur est unique pour chaque utilisateur
  - l'utilisateur a soit une calculette, soit une carte à grille, mais il ne peut pas avoir les deux ou aucun
- Operation schemas
  - IHB::loginNomUtilisateur(nom : NomUtilisateur)
  - IHB::loginMotPasse(motPasse : MotPasse)

©Shane Sendall for Unicible

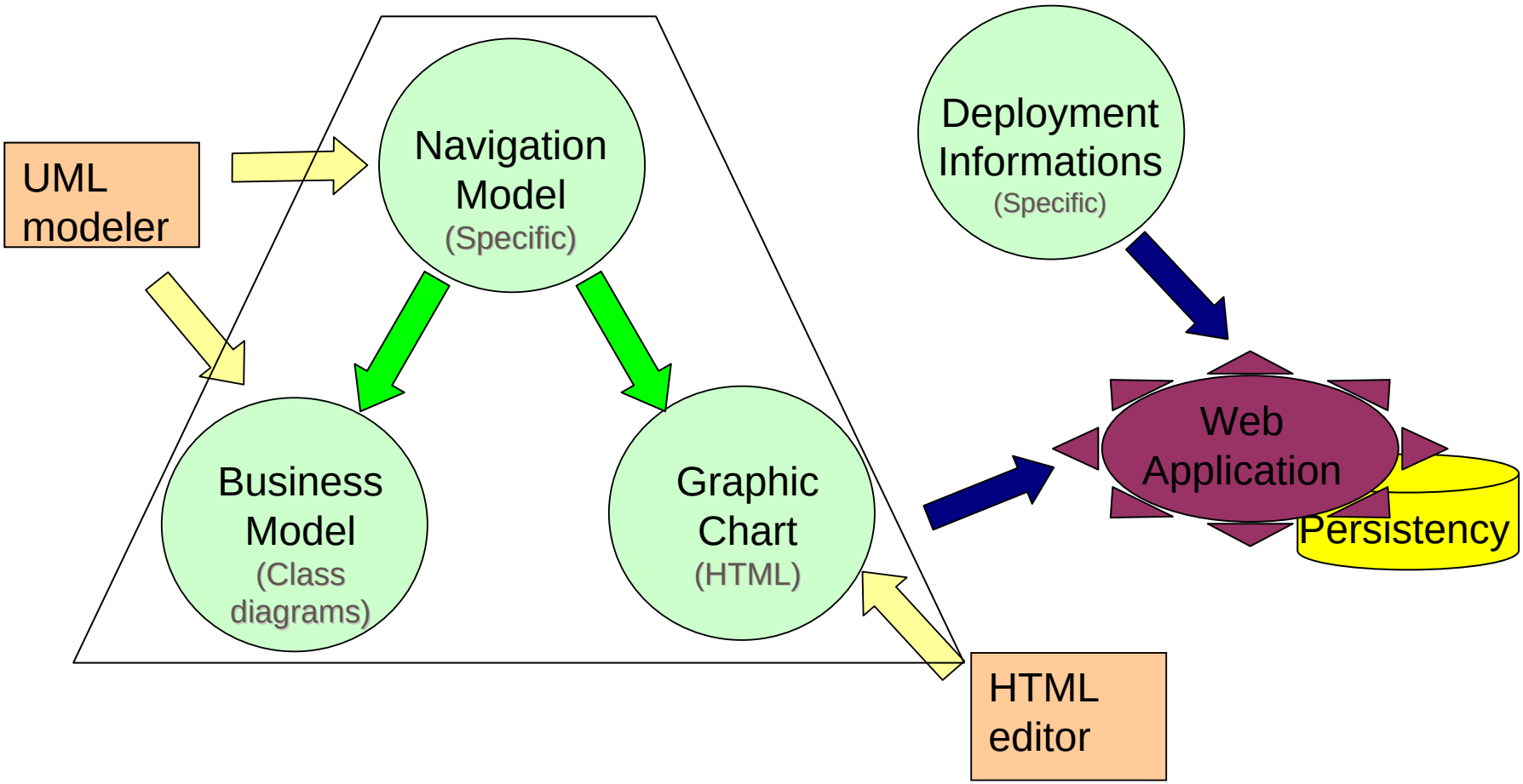
Contents

- Introduction
- Netsilon overview
  - General architecture
  - Business model
  - Presentation and navigation model
  - Xion
- Awaited interface
- Static aspects
- Dynamic aspects
- Interface
- Conclusion

General architecture

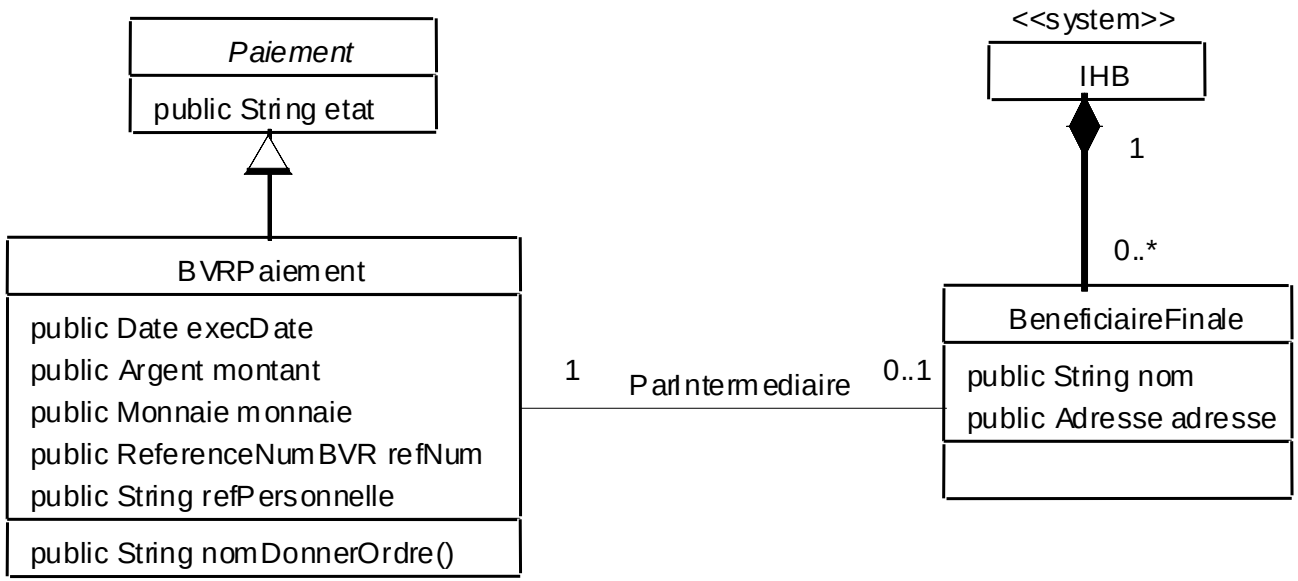


General architecture



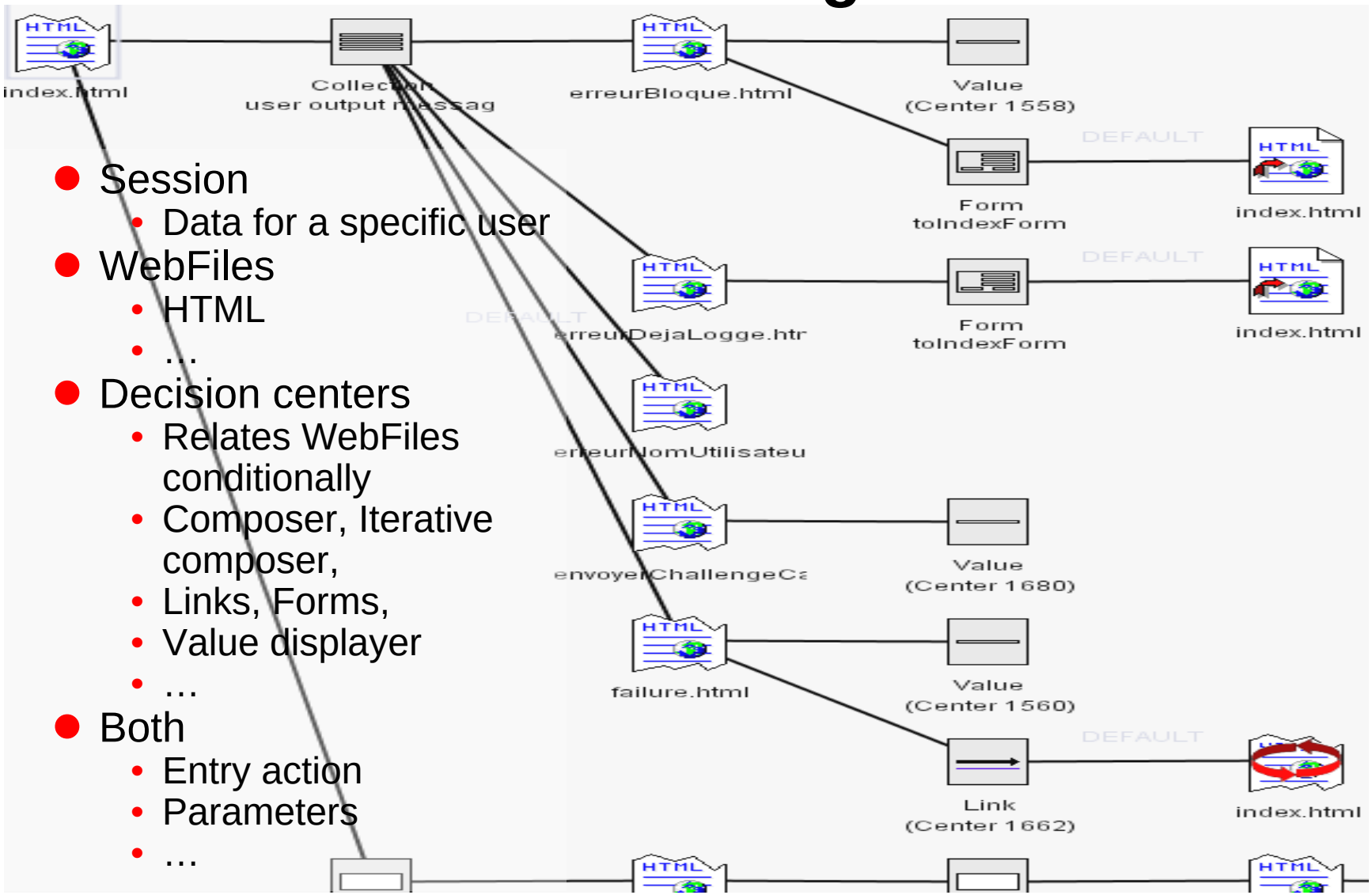
Business model

- Class diagram
  - Class
  - Attributes
  - Operations
  - Methods
  - Constructors
  - Simple inheritance
  - Interfaces (Required / Provided)
  - …



Presentation and navigation model

- Session
  - Data for a specific user
- WebFiles
  - HTML
  - …
- Decision centers
  - Relates WebFiles conditionally
  - Composer, Iterative composer,
  - Links, Forms,
  - Value displayer
  - …
- Both
  - Entry action
  - Parameters
  - …



Xion

Imperative action language

- Implement constructors and methods
- Give conditions and actions on Presentation model

- Query business objects
- Modify business objects

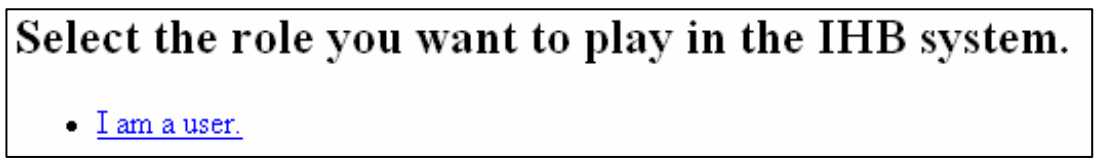
- Based on Java for data flow control (variable declaration, if, for, …)
- Based on OCL for standard library and navigation

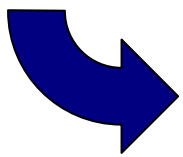
Contents

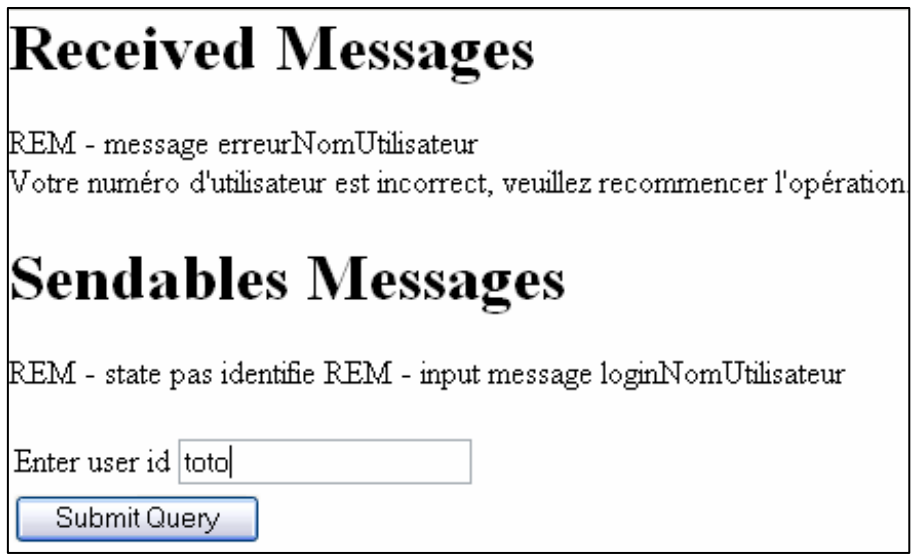
- Introduction
- Netsilon overview
- Awaited interface
  - “Ethics”
  - Let’s see a demo: logging IHB
- Static aspects
- Dynamic aspects
- Conclusion

“Ethics”

- Responsibilities
  - Make possible to play the role of any actor
  - Make possible to send messages according to the protocol
  - Shows messages sent to an actor







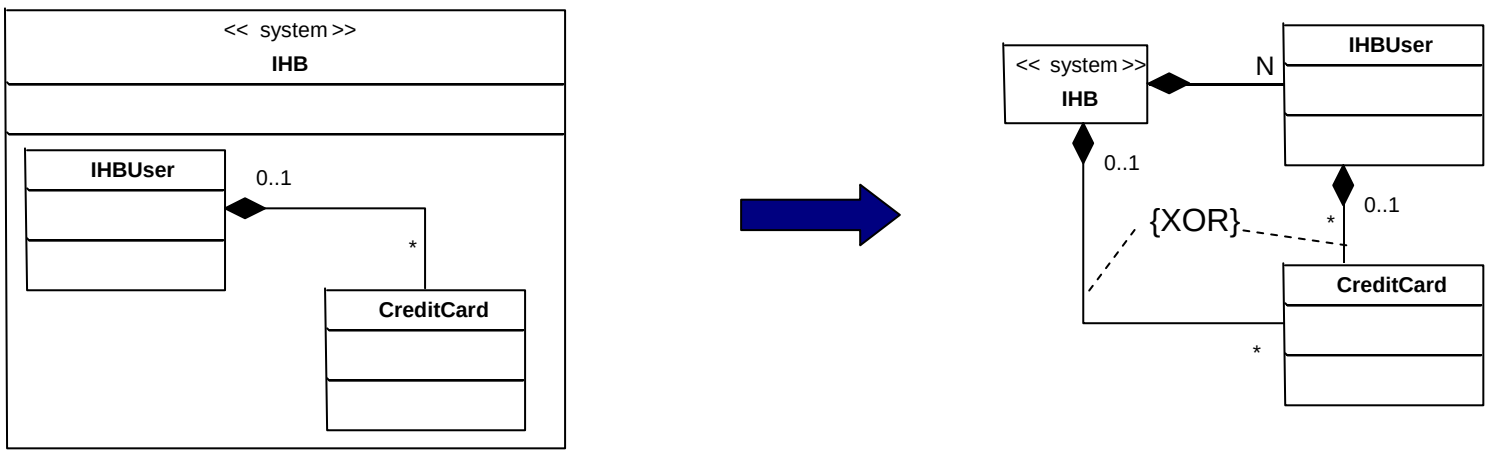
Let’s see a demo : logging IHB

Contents

- Introduction
- Netsilon overview
- Awaited interface
- Static aspects
  - Mapping concepts
  - Mapping actors
  - Mapping messages
- Dynamic aspects
- Conclusion

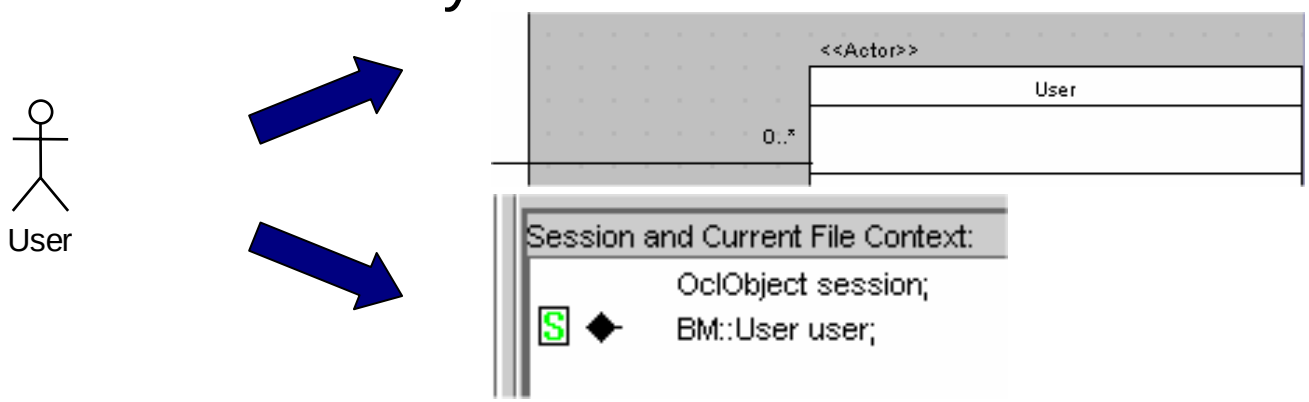
Mapping concepts

- Classes are classes…
  - Attributes are attributes
  - What about data types and redefined types ?
- The system is a singleton class composing (transitively) any class
  - WARNING: take care of composition relationships (if class is not composed then composed by the system, if class may be composed, then may be composed by the system)



Mapping actors

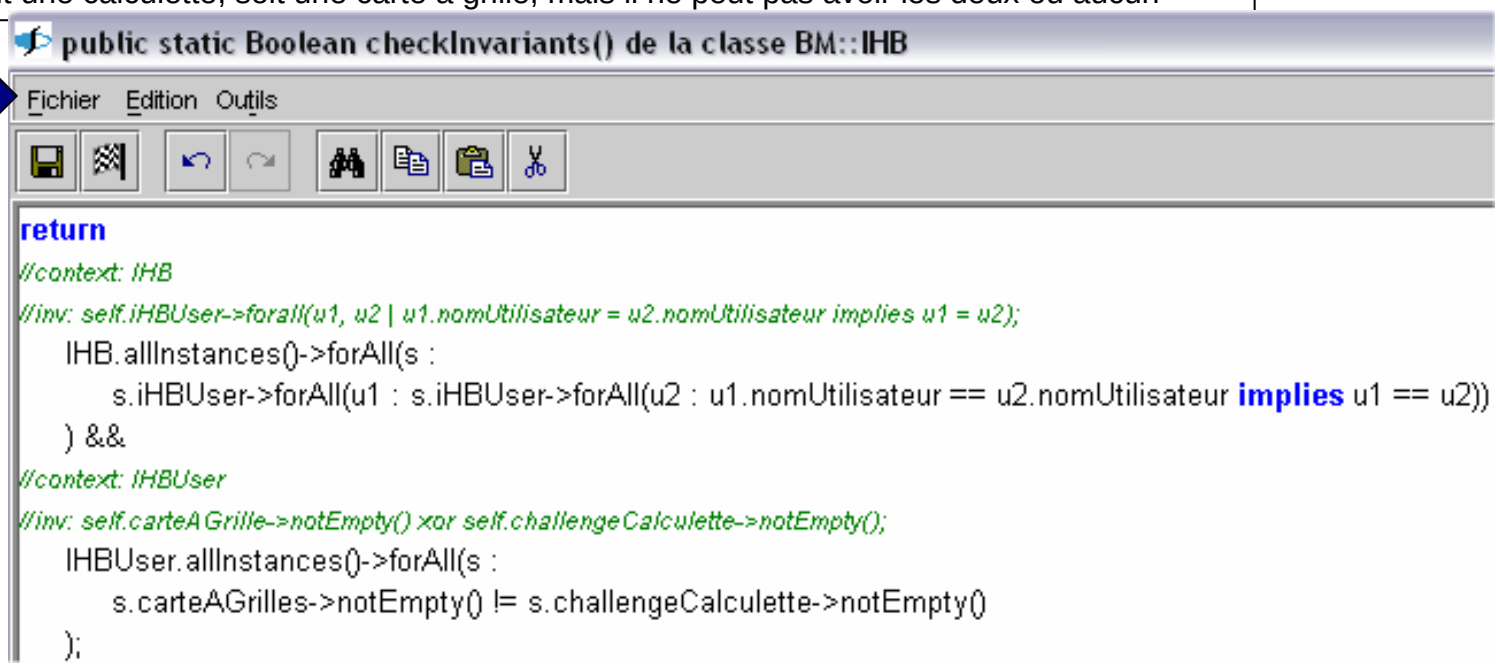
- Actor become Class of the business model
  - No attributes
  - Linked either to the system or to the <<id>> referring class
- Current actor is stored in the session, once the role is defined (must say somewhere who you are)
  - May be many actor simultaneously for one session
  - This could be improved by separating human and machine actors ; e.g. for prototypes composition (client server vs. customer system)
- The actor is destroyed with the session



Mapping concepts

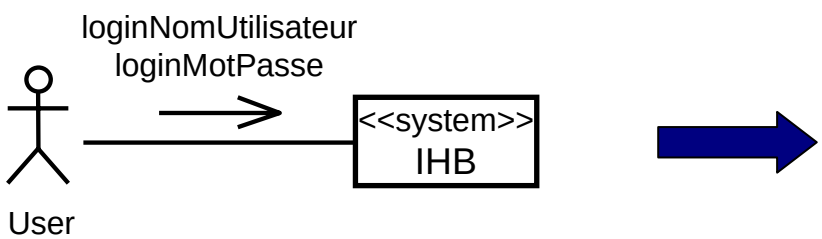
- Invariants are translated in Xion inside a method of the system

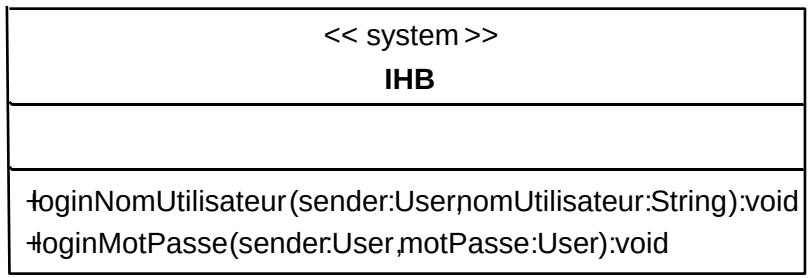
Invariants
context: IHB
inv: self.iHBUser->forall(u1, u2 | u1.nomUtilisateur = u2.nomUtilisateur **implies** u1 = u2);
-- nomUtilisateur est unique pour chaque utilisateur
context: IHBUser
inv: self.carteAGrille->notEmpty() **xor** self.challengeCalculette->notEmpty();
-- l'utilisateur a soit une calculette, soit une carte à grille, mais il ne peut pas avoir les deux ou aucun

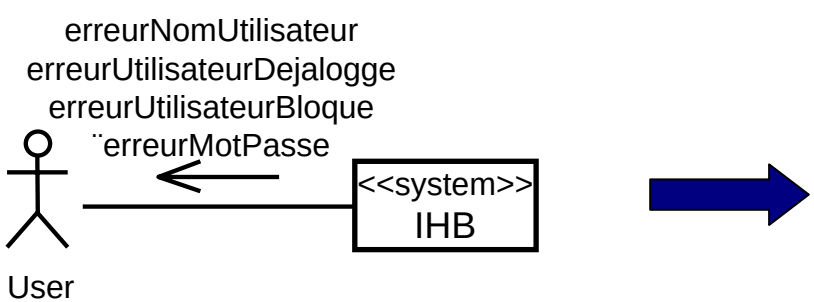


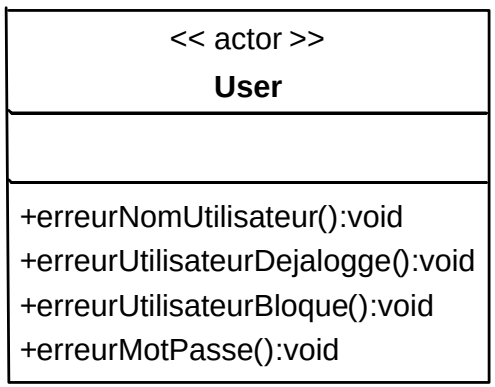
Mapping Messages

- Input messages are operations of the system class
  - Sender is given as parameter
- Output messages are operations of actors who are sent these messages



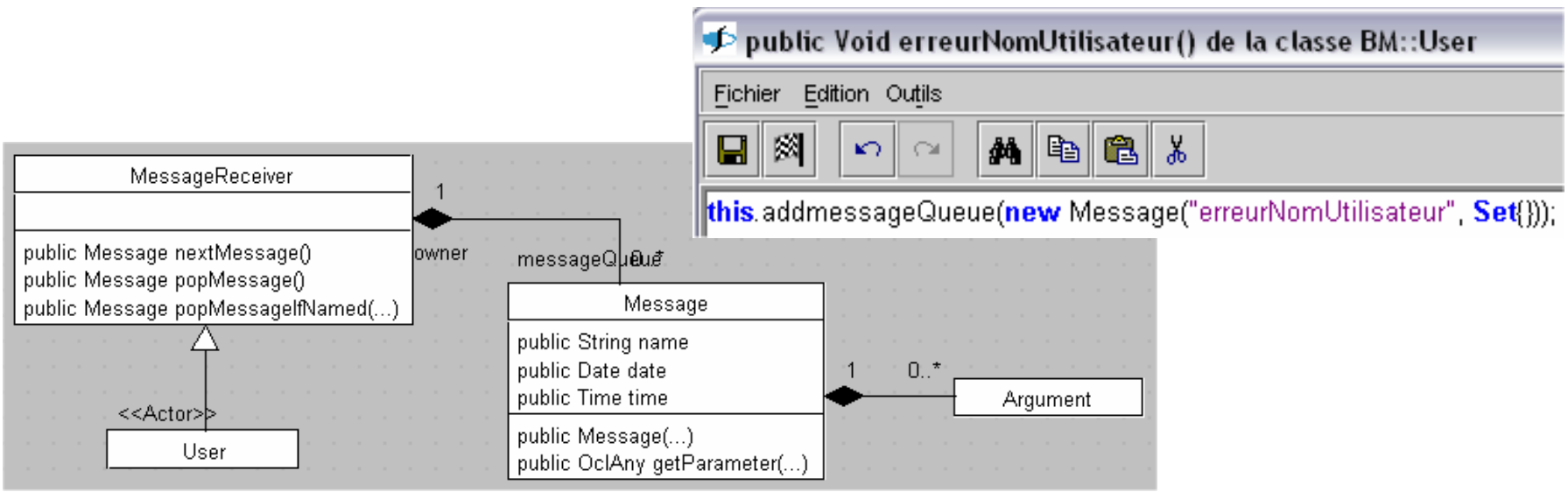






Mapping Messages (Implementation)

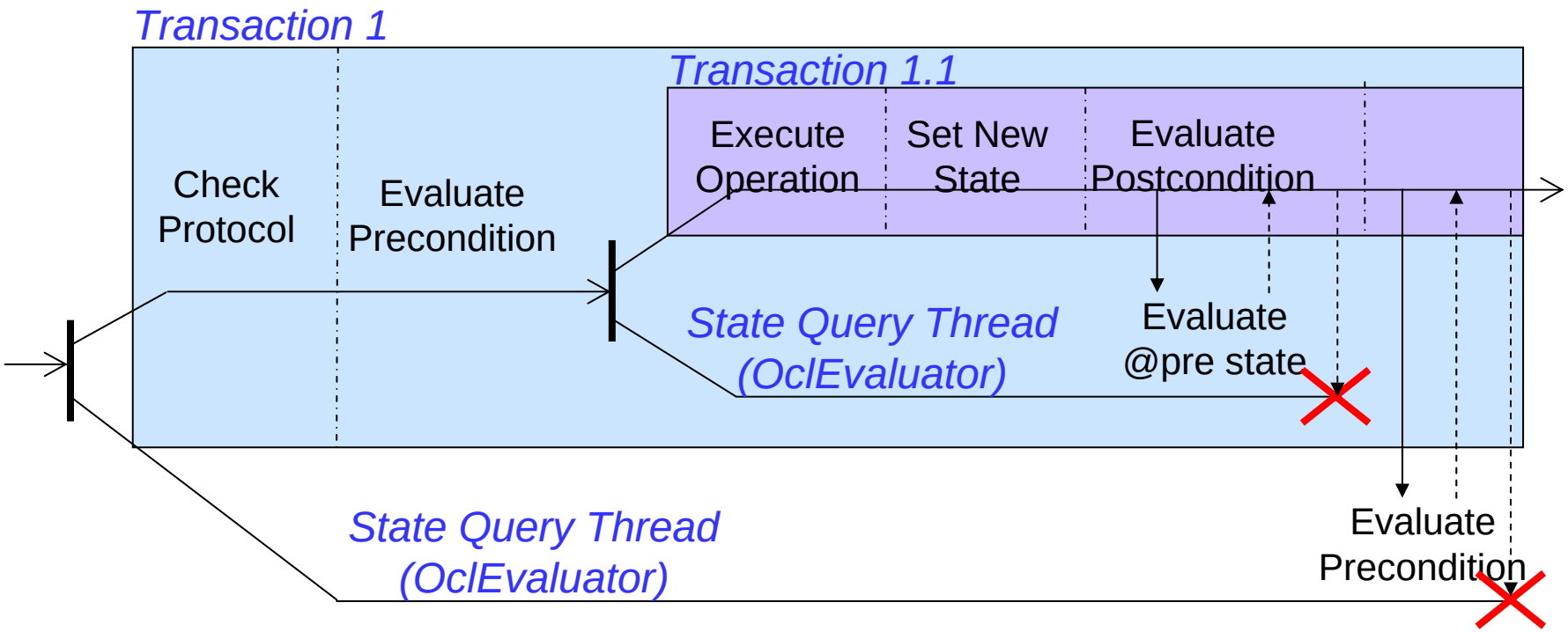
- Implementation of INPUT messages will be shown in dynamic aspects
  - Has something to do with protocol model and operation schemas
- Implementation of OUTPUT message is made by a message queue
  - They are popped by the interface



Contents

- Introduction
- Netsilon overview
- Awaited interface
- Static aspects
- Dynamic aspects
  - Executing operation schemas
  - Storing protocol model
  - Mapping operation schemas
  - Mapping protocol models
- Conclusion

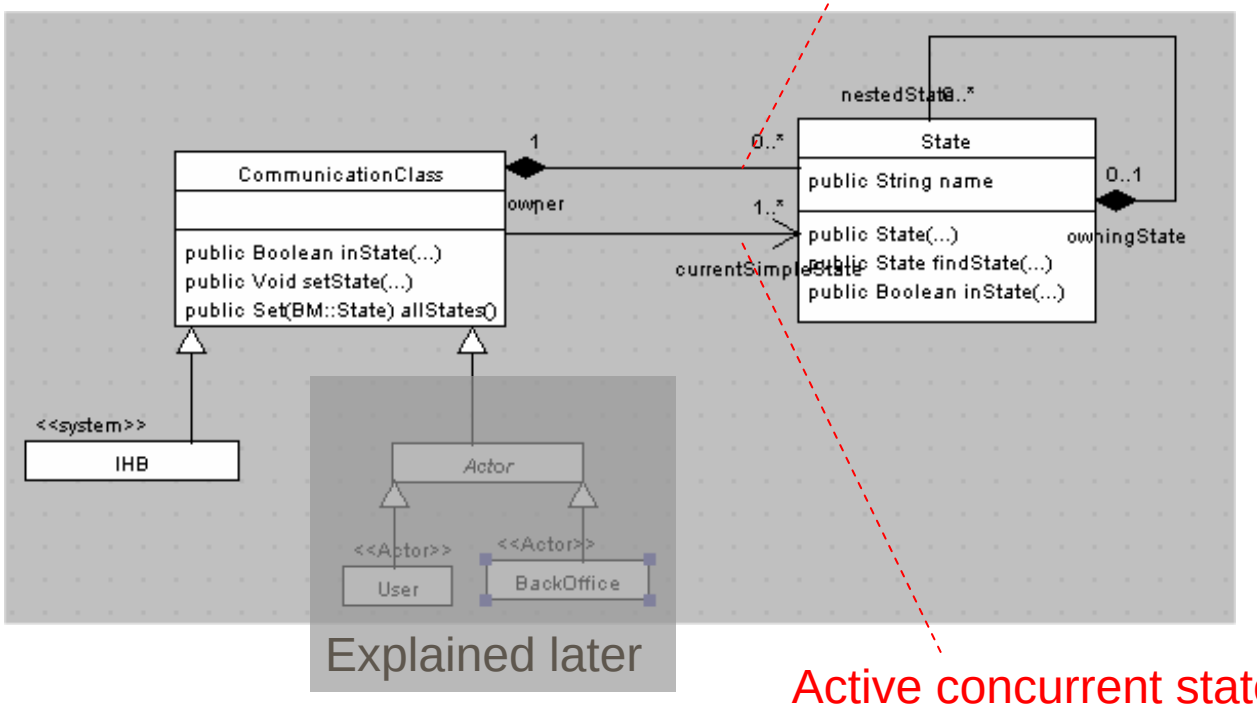
Executing operation schemas



- "Execute Operation": what do that mean ?
- Netsilon does not offer fine grained nested transactions
- No OCL evaluator available ; translate OCL constraints in Xion
- Protocol state should be stored somewhere…

Storing protocol model

- Apply the state pattern
- Or be as lazy as me :



Mapping operation schemas

Belongs to the system

- Check invariants (Optional ?)
- Check protocol
  - Is the message accepted in the current state
- Evaluate pre-accessible alias
- Evaluate pre-condition
- Translate post-condition to action
  - Some kind of magic behind here (ask Slavisa !)
  - What to do with @pre statements
  - What to do with multiple possible system state (despite minimal set, etc.)
  - What to do with post accessible alias
- Change protocol state
- Evaluate post-accessible alias
- Check invariants and post-condition

Mapping operation schemas

- The operation schema…

Operation : IHB::loginMotPasse(motPasse : MotPasse);
Description : Cette opération est responsable d'assurer que le mot de passe (motPasse) associé au nom d'utilisateur donné précédemment est valide, et que si ce n'est pas le cas, alors le nombre de tentatives ne dépasse pas 3. Si ce nombre excède 3, alors l'utilisateur devient bloqué.
Notes: L'utilisateur correspondant est "trouvé" en naviguant sur l'association AuneSessionAvec.
Use Cases : S'authentifier sur le site ;
Scope : -- reste à faire
Aliases :
u : IHBUser **Is** sender.iHBUser; -- l'utilisateur qui correspond au sender
Messages : User::{ErreurMotPasse; EnvoyerUtilisateurBloque; EnvoyerChallengeCase;};
Pre :
u.isDefined() **and** -- u existe (il a une session avec le système)
not u.etat = EtatUtilisateur::logge; -- l'utilisateur n'est pas déjà loggé
Post :
if u.etat = EtatUtilisateur::bloque **then** -- s'il a dépassé le nombre de tentatives permises
sender^envoyerUtilisateurBloque () -- il est informé qu'il est bloqué
elsif u.motPasse <> motPasse **then** -- si le mot de passe n'est pas correct
u.nombreTentativesLogin = u.nombreTentativesLogin@pre + 1 **and** -- le nombre de tentatives est incrémenté d'un
sender^erreurMotPasse() -- il est informé de l'erreur sur le mot de passe
if u.nombreTentativesLogin = 3 **then** -- s'il a dépassé les trois tentatives permises
u.etat = EtatUtilisateur::bloque **and** -- il est bloqué
u.nombreTentativesLogin = 0 -- le compteur est remis à zéro
end
else -- autrement, le mot de passe est correct
u.nombreTentativesLogin = 0 **and** -- le compteur est remis à zéro
u.possedeUneCarte.caseNomCourant = choisirCase_f(u) **and** -- faire avancer la case courante (pour la prochaine fois)
sender^envoyerChallengeCase(u.possedeUneCarte.caseNomCourant) -- le case nom a été envoyé
endif;

Mapping operation schemas

- …and the Xion operation

```
if (! checkInvariants()){
    self.failure('Invariants checking failed before execution.');
}
//Pre possible alias
IHBUser u = sender.iHBUser;
//Pre-condition
if (! (u != null && u.etat != #logge)) {
    self.failure("Precondition of operation loginMotPasse failed");
    return;
}
//Post possible alias
//Post-condition as action
if (u.etat == #bloque) {
    sender.erreurUtilisateurBloque();
} else if (u.motPasse.mot != motPasse) {
    u.nombreTentativesLogin++;
    sender.erreurMotPasse();
    if (u.nombreTentativesLogin == 3) {
        u.etat = #bloque;
        u.nombreTentativesLogin = 0;
    }
} else {
    u.nombreTentativesLogin = 0;
    u.possedeUneCarte.currentCaseNom = self.choisirCase_f(u);
    sender.envoyerChallengeCase(u.possedeUneCarte.currentCaseNom);
}
```

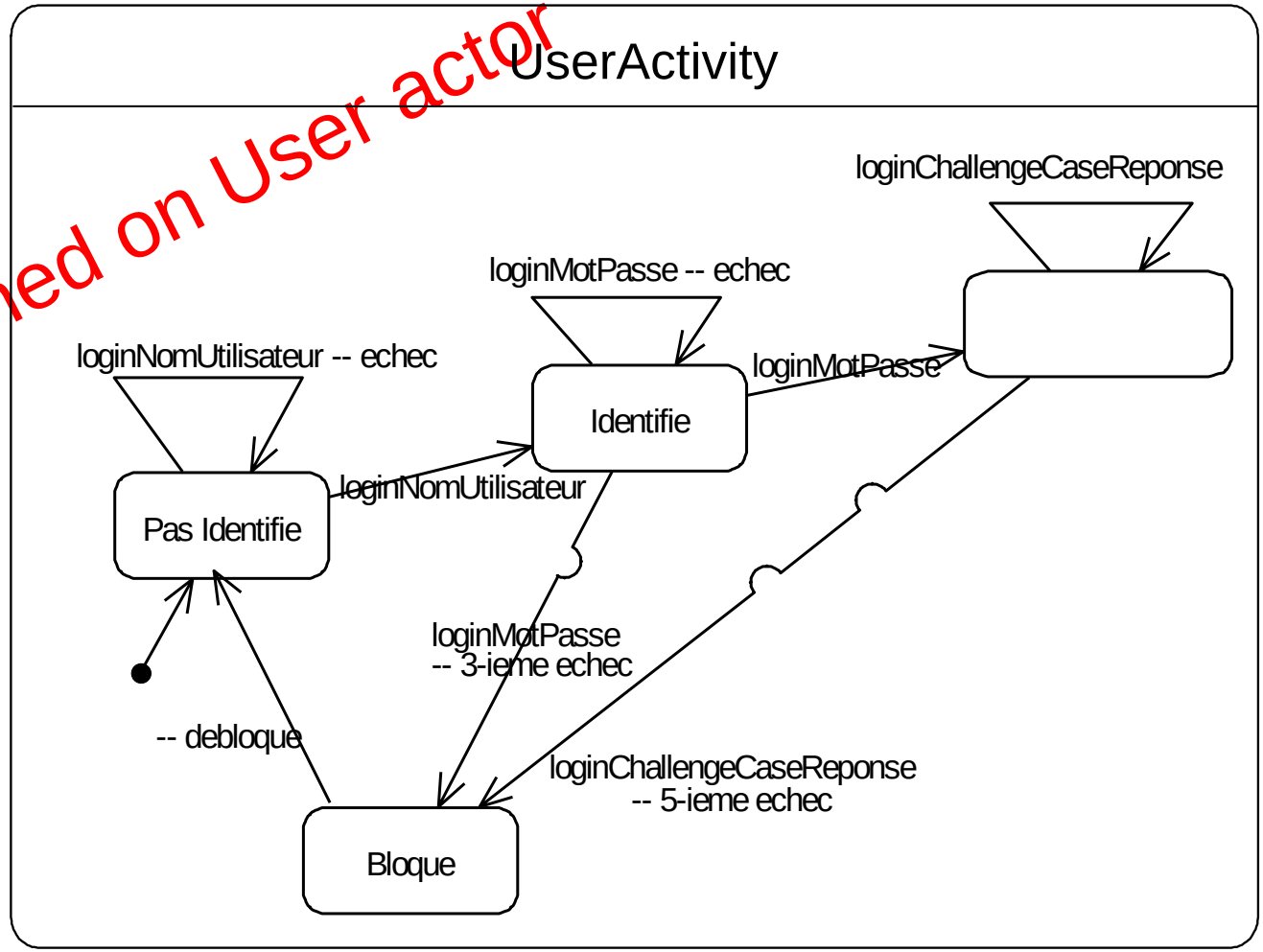
```
//Protocol output state
//It is stated each user activity is related to the number of users (invented - not defined in Fondue)
//Guards are missing (invented - not defined in Fondue) ;
//    invented version (missing in the analysis) ; guards are ordered unlike defined by state machines
//Missing transition from Identifie to Bloque in the Analysis
if (sender.inState('Identifie')) { //source state
    if (sender.iHBUser.etat == #bloque)
        sender.setState('Bloque', Set{'Identifie'});
    else if (sender.iHBUser->isEmpty())
        ; //Nothing to do : source and target are the same
    else if (sender.messageQueue->exists(message : message.name == 'envoyerChallengeCase'))
        sender.setState('EnChallenge', Set{'Identifie'});
    else
        self.failure('Cannot find output state !');
} //in this special case, there is just one possible source state - no else statement
```

- Protocol must be deterministic
  - Guards evaluated after the operation executed
  - Kind of post-condition with access to parameters and alias
  - Need a guard evaluation order

Mapping protocol models

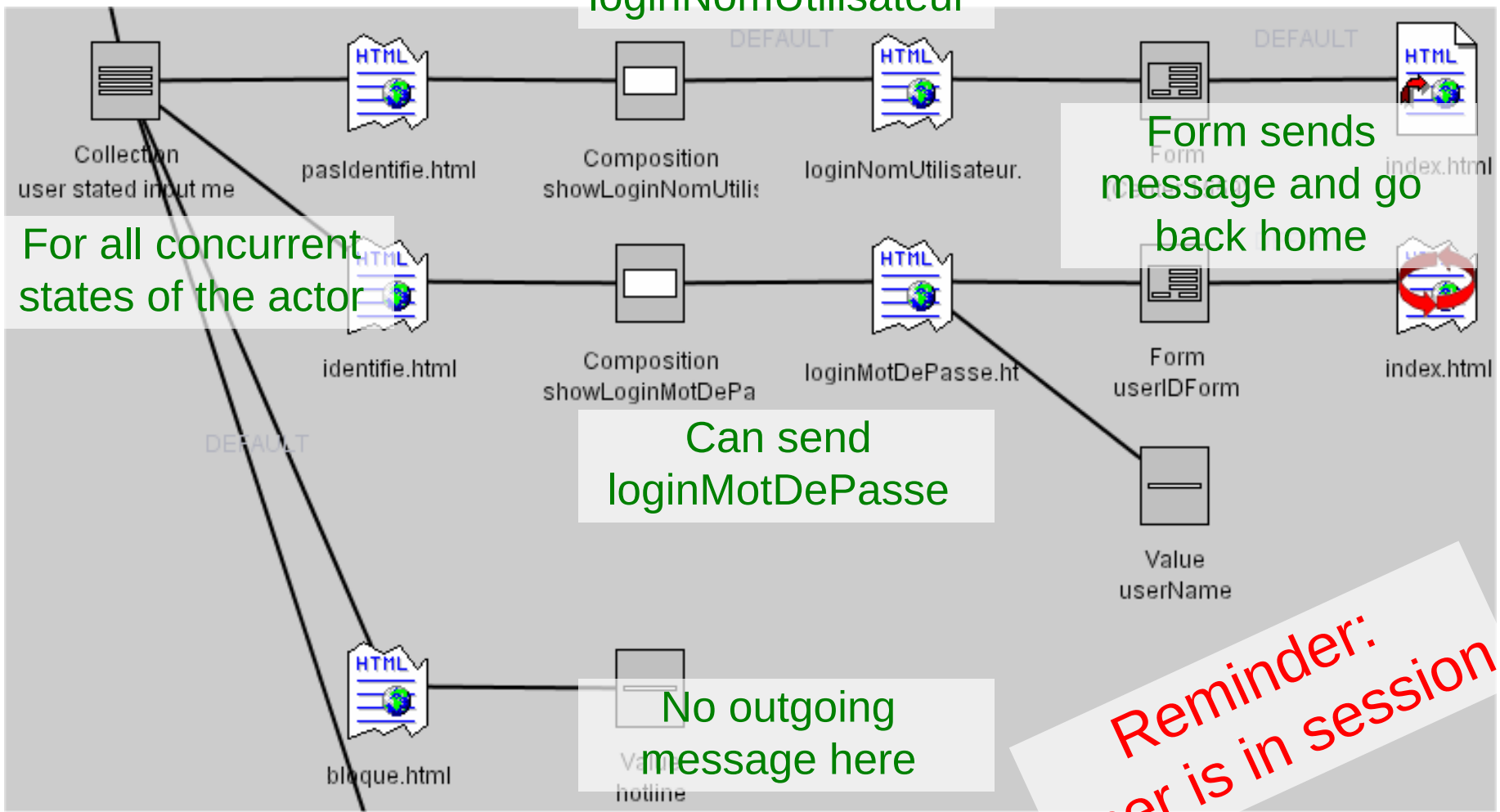
- Important problem here : what to do with multiple states
  - Creation condition
  - Destruction condition
- Empiric observation : multiple states are always related to actor or class instances
  - One concurrent state per actor / class instance
- Solution adopted here
  - Multiple states removed
  - Content of multiple state located into actor / class concerned (as done in UML)
  - We are still dealing with input messages here (everything is seen from the system point of view)
- Other possible solution
  - Make a template state (not UML)

Mapping protocol models



©Shane Sendall for Unicible (partial and corrected)

Mapping protocol models



Contents

- Introduction
- Netsilon overview
- Awaited interface
- Static aspects
- Dynamic aspects
- Conclusion

Conclusion

- Mapping post conditions to implementation
- Multiple state instanciation / end
  - Put states on classifiers / actors
- Protocol indeterminism
  - Messages guards in protocol model
    - Defined like post conditions
    - May be ordered
  - No need redundant information between protocol and class attributes
- Difference between human and system actors
  - Multiple systems prototyping
- OCL constraints detect bugs at runtime