Design of Information Systems

UML Modeling Concepts and Introduction to USE

Martin Gogolla
University of Bremen, Germany
Database Systems Group

Goals of object-oriented modeling

- Assume (simple) software development process:
  Requirements, Design, Implementation, Testing, Maintenance
- Following: Central steps within Design
- Integrated description of structure and behavior of objects
- Representation for properties of objects and
  relationships between objects
- Development of object descriptions capturing
  state transitions and object lifecycles
- Options to describe type level and instance level aspects
- Modeling language used here for Design:
  Unified Modeling Language (UML)

Good reference book on UML (Unified Modeling Language)

James Rumbaugh, Ivar Jacobson, Grady Booch
Unified Modeling Language Reference Manual, Second Edition
ISBN 0321245628
Pearson Higher Education

Excerpts (tables, quotations, diagrams, ...) in the course slides

Table 3-1: *UML Views and Diagrams*

Major Area	*View*	*Diagram*	*Main Concepts*
structural	static view	class diagram	association, class, dependency, generalization, interface, realization
	design view	internal structure	connector, interface, part, port, provided interface, role, required interface
		collaboration diagram	connector, collaboration, collaboration use, role
		component diagram	component, dependency, port, provided interface, realization, required interface, subsystem
	use case view	use case diagram	actor, association, extend, include, use case, use case generalization

Table 3-1: *UML Views and Diagrams (continued)*

Major Area	*View*	*Diagram*	*Main Concepts*
dynamic	state machine view	state machine diagram	completion transition, do activity, effect, event, region, state, transition, trigger
	activity view	activity diagram	action, activity, control flow, control node, data flow, exception, expansion region, fork, join, object node, pin
	interaction view	sequence diagram	occurrence specification, execution specification, interaction, interaction fragment, interaction operand, lifeline, message, signal
		communication diagram	collaboration, guard condition, message, role, sequence number

Basic UML diagrams (explained by social network example model)

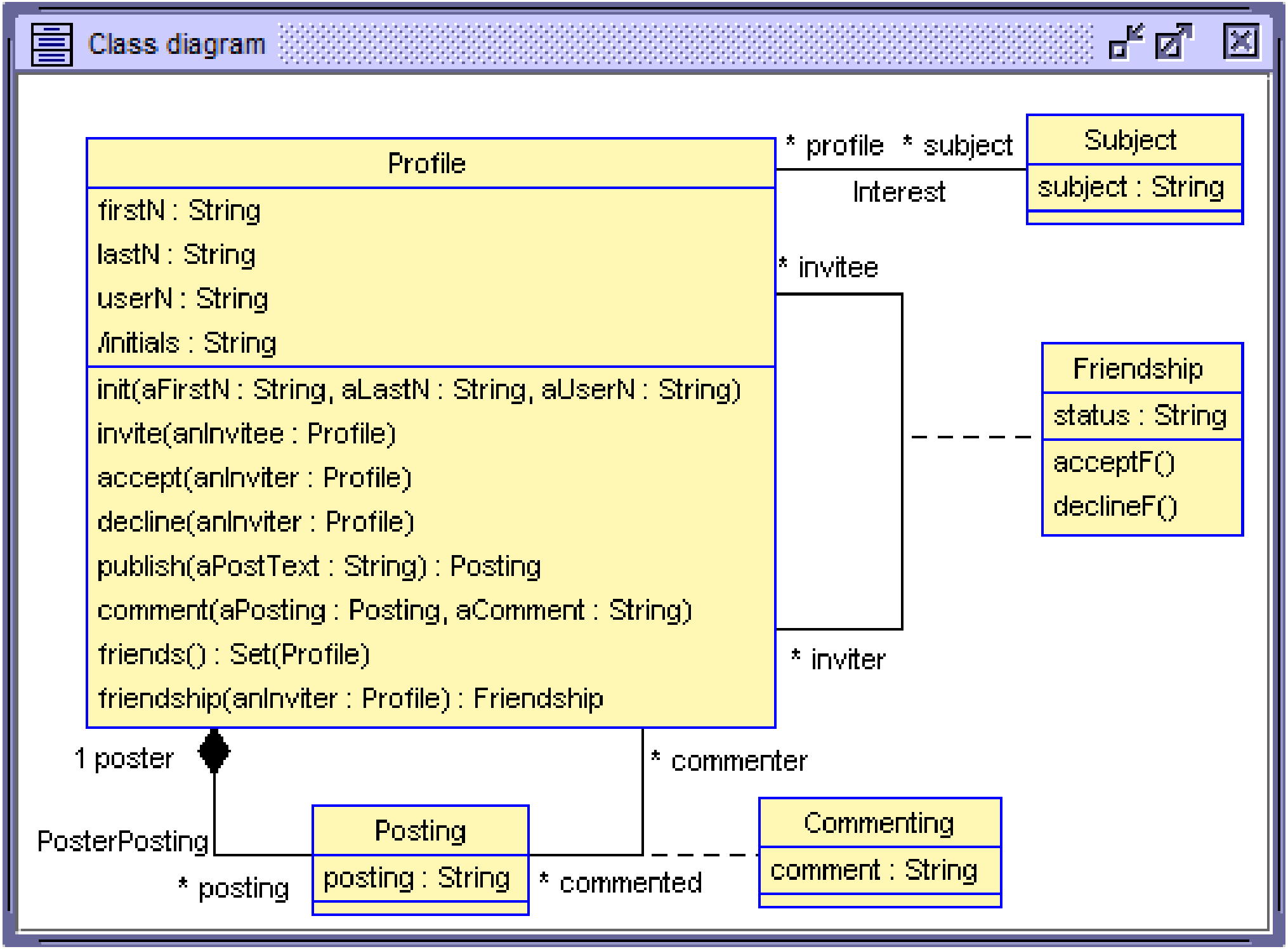
- Class diagram (Structure)
- Use case diagram (Behavior) (*)
- Object diagram (Structure)
- State chart diagrams (Behavior)
- Sequence and communication diagram (Interaction, Behavior)
- Activity diagram (Behavior) (*)
- Diagrams enriched and made precise by expressions written in OCL (Object Constraint Language) being part of UML
- OCL expression itself without side-effect: no system state change
- Class invariants, operation definitions, operation pre- and postconditions (operation contracts), ...
- Explained with UML-based Specification Environment (USE); not supported in USE: (*)

Short How-to for UML-based Specification Environment (USE)

- steps to install USE under Windows XY
- google: use ocl bremen
- -> https://sourceforge.net/projects/useocl/
- download zip file "use-X.Y.Z.zip"; save file on Desktop
- unzip file to Desktop directory use-X.Y.Z
- on Desktop context menue "new link" (e.g. "use-4.2.0");
  let the link point to:
- use-X.Y.Z/bin/start_use.bat
- double click your link to start USE with an empty model
- USE offers CLI (Command Line Interface; shell) and
  GUI (Graphical User Interface)
- optional: adjust CLI via properties (font, colors, size, position, ...)

USE and OCL for the impatient: 9 OCL expressions on the CLI

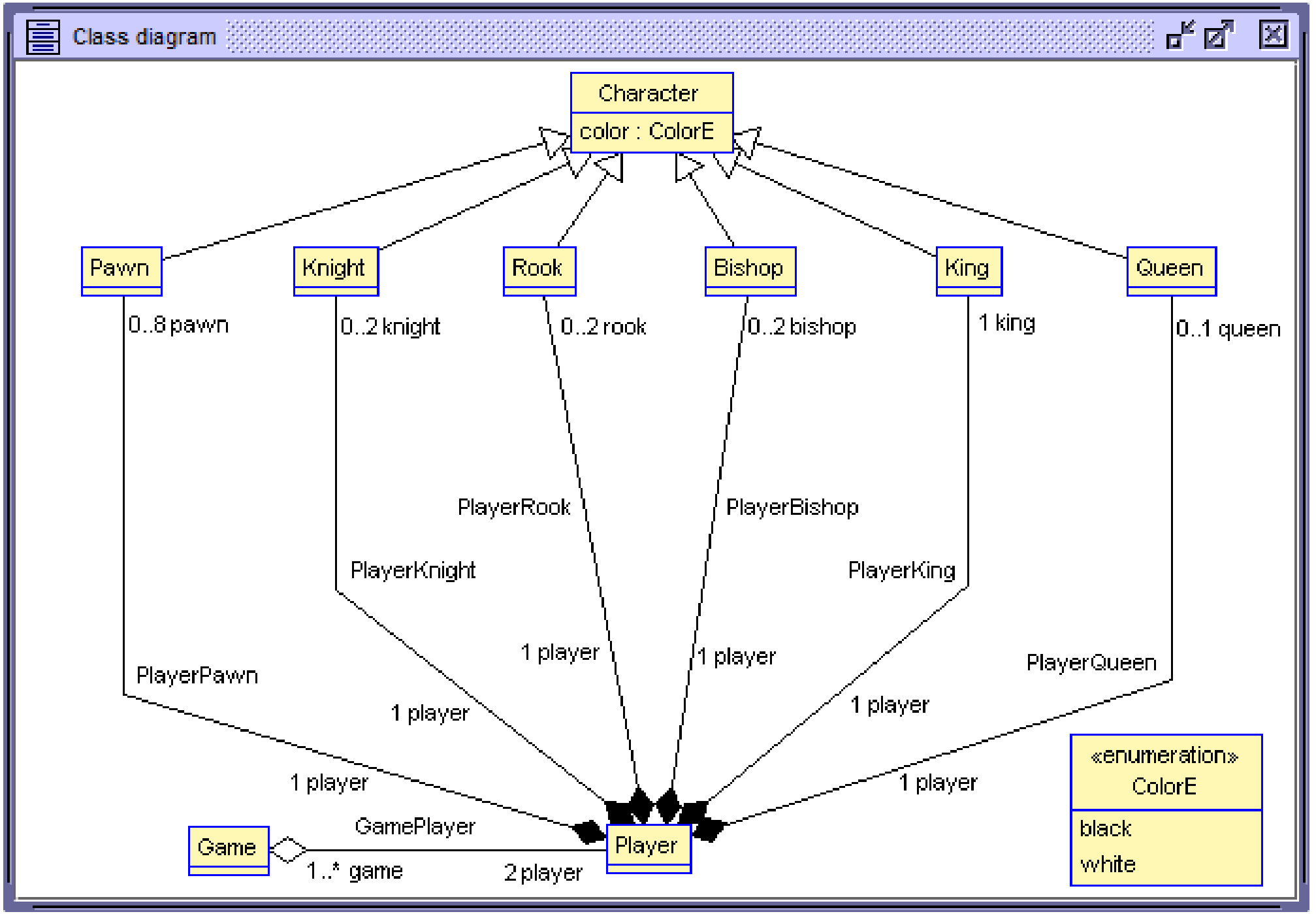
- `?21+21`
- `?20.9+21.1`
- `?20.9+21.1=42`
- `?'for'+'tytwo'`
- `?Set{7,9,5}`
- `?Bag{7,9,5,7}`
- `?Bag{7,9,5,7}=Bag{5,7,9}`
- `?Set{7,9,5,7}=Set{5,7,9}`
- `?Set{7,9}->union(Set{9,5})->select(i|i<9)`
- OCL: datatypes, collections, operations


Class diagram
Profile
firstN : String
lastN : String
userN : String
/initials : String
init(aFirstN : String, aLastN : String, aUserN : String)
invite(anInvitee : Profile)
accept(anInviter : Profile)
decline(anInviter : Profile)
publish(aPostText : String) : Posting
comment(aPosting : Posting, aComment : String)
friends() : Set(Profile)
friendship(anInviter : Profile) : Friendship
* profile
* subject
Interest
Subject
subject : String
* invitee
* inviter
Friendship
status : String
acceptF()
declineF()
1 poster
PosterPosting
* posting
Posting
posting : String
* commenter
* commented
Commenting
comment : String

Class diagram concepts (in example)

- USE diagrams customizable via context menue (right click); diagram parts hideable; diagram layout storable
- Class, attribute, operation, parameter, (return) type
- OCL collection kind Set(T),
  more collection kinds: Bag(T), Sequence(T), ...
  superclass Collection(T) > Set(T), Collection(T) > Bag(T), ...
- (Binary) association, association class (with attributes), reflexive association, composition, generalization, aggregation
- composition, aggregation: part-whole relationships, acyclic on objects, composition (exclusive) strong binding (0..1), aggregation (sharable) weak binding (0..*)
- Role, association name, multiplicities
- Roles used for navigation from one object to other objects

Example for class diagram concepts (generalization, multiplicities)



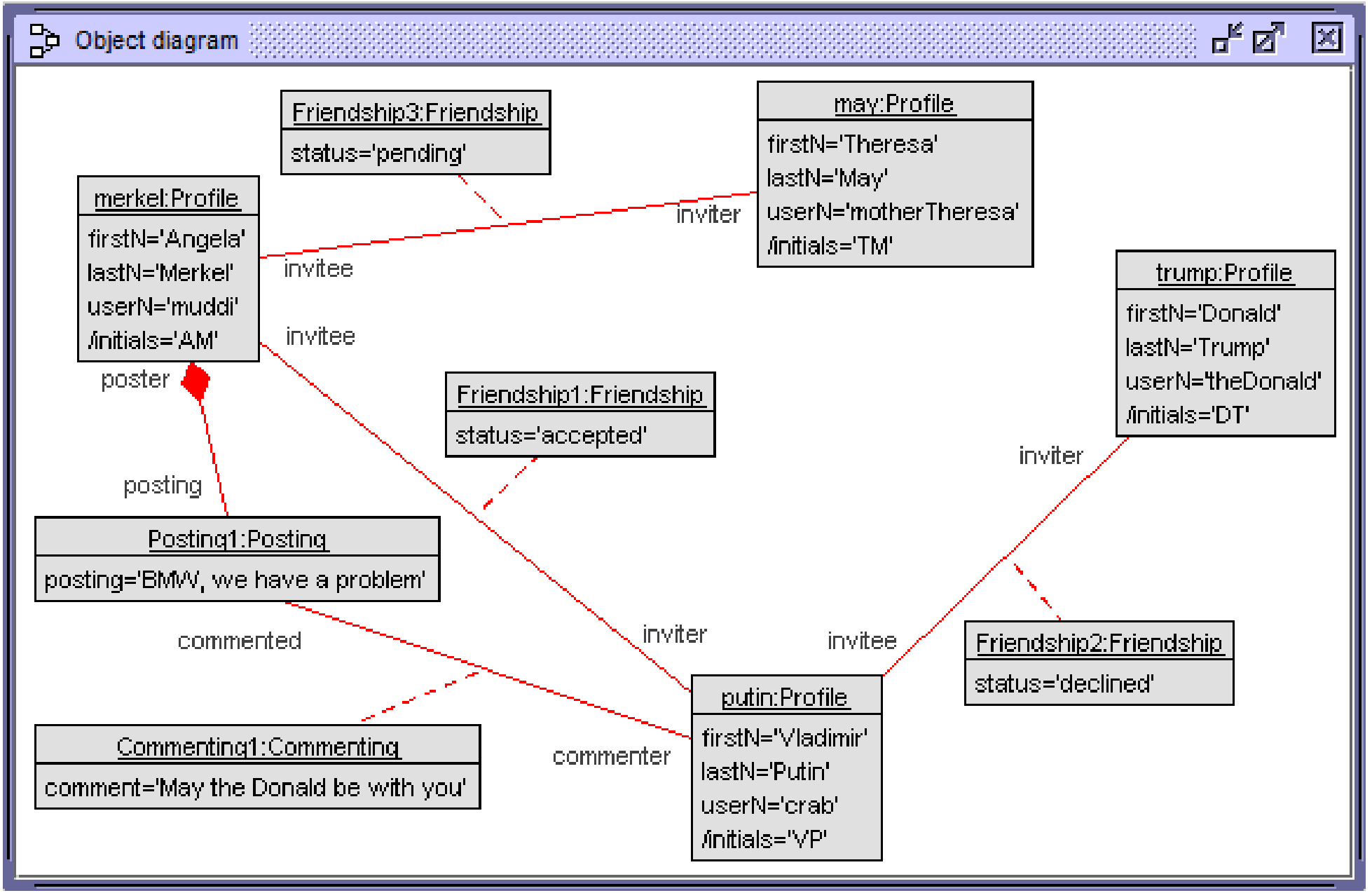
Invariant and operation definition with OCL

```
class Profile
operations
  friends():Set(Profile)= -- 'self.inviter->union(self.invitee)'
    friendship[inviter]->select(oclInState(accepted)).invitee->union(
    friendship[invitee]->select(oclInState(accepted)).inviter)->asSet

context Profile
  inv asymmetricFriendship: invitee->intersection(inviter)->isEmpty()
  inv uniqueUserName: Profile.allInstances->isUnique(userN)

context Commenting inv commentOnlyByFriends:
  commented.poster.friends()->includes(commenter)
```

Concepts

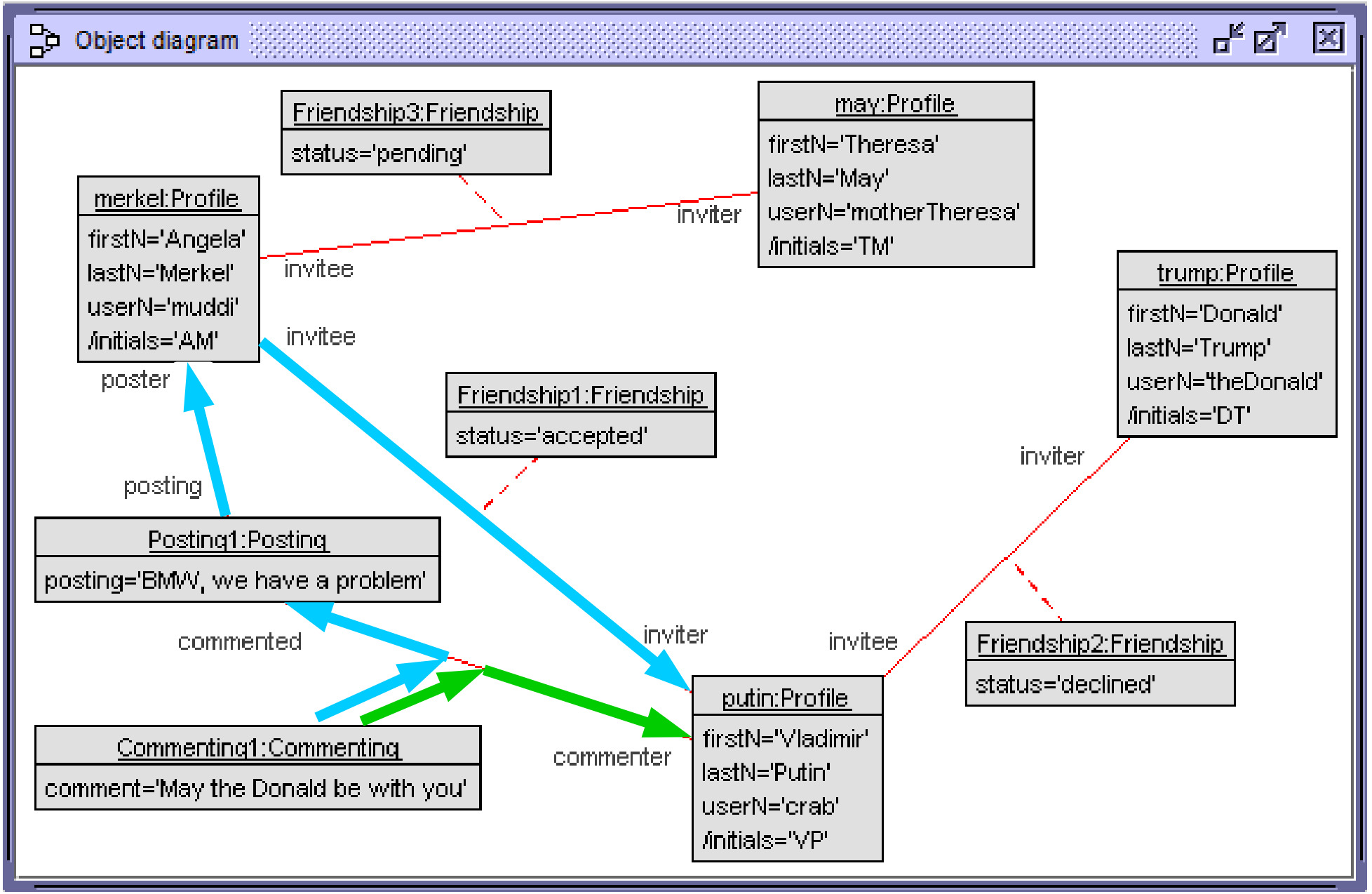
- Class invariant with name and boolean OCL expression as body
- Query operation definiton; class-valued or collection-valued
- Roles for navigation (e.g., invitee or inviter)
- Collection operations: allInstances, intersection, isEmpty, isUnique, ...

Object diagram: example and concepts



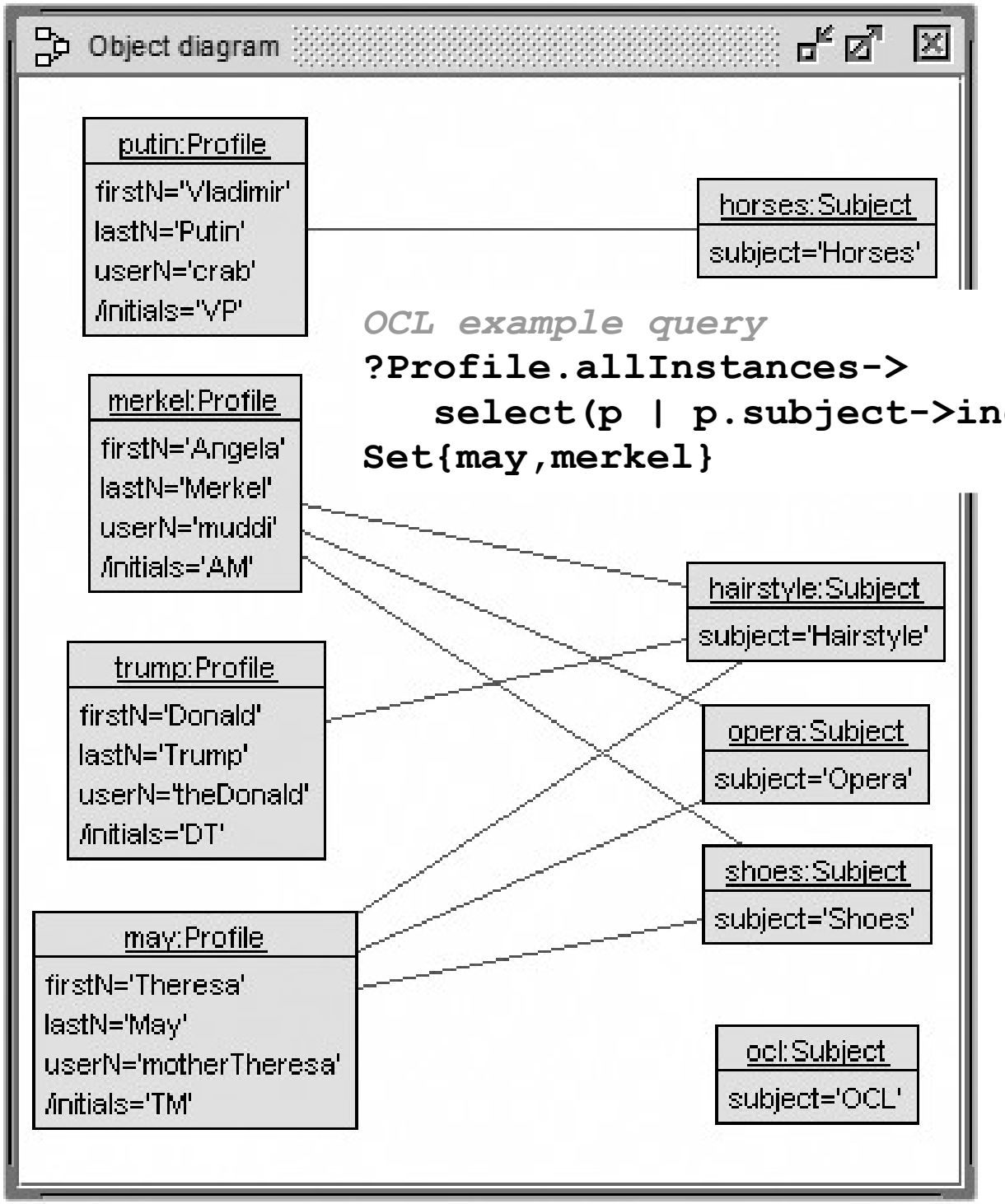
- Object (class), link (association), value (attribute), link object (assoc class)
- Instance level in object diagram, type level in class diagram

OCL class invariant: example evaluation in object diagram

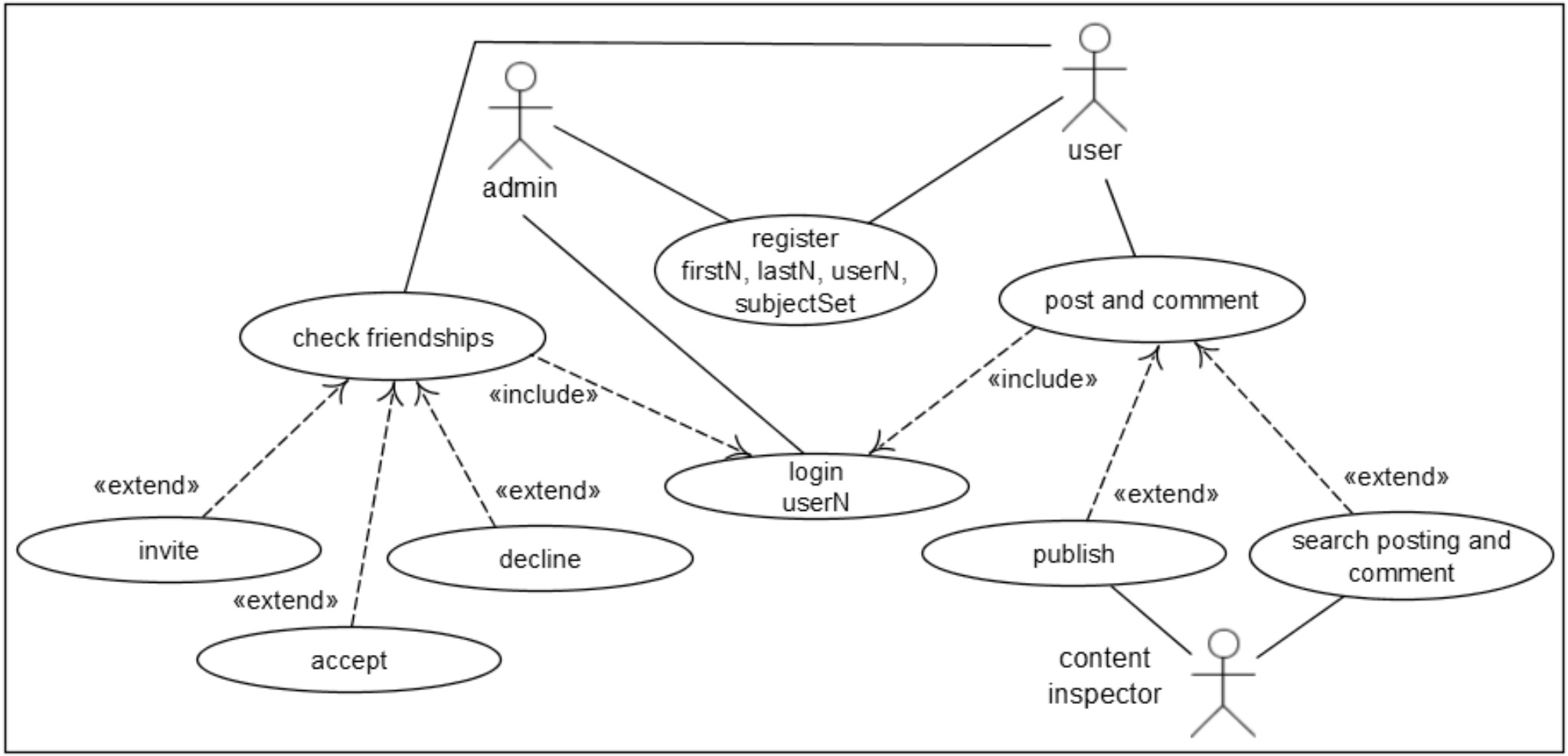



```
context self:Commenting inv commentOnlyByFriends: -- 'self' optional
  self.commented.poster.friends()->includes(self.commenter)
```

OCL query in object diagram [underlying (*,*) multiplicity]

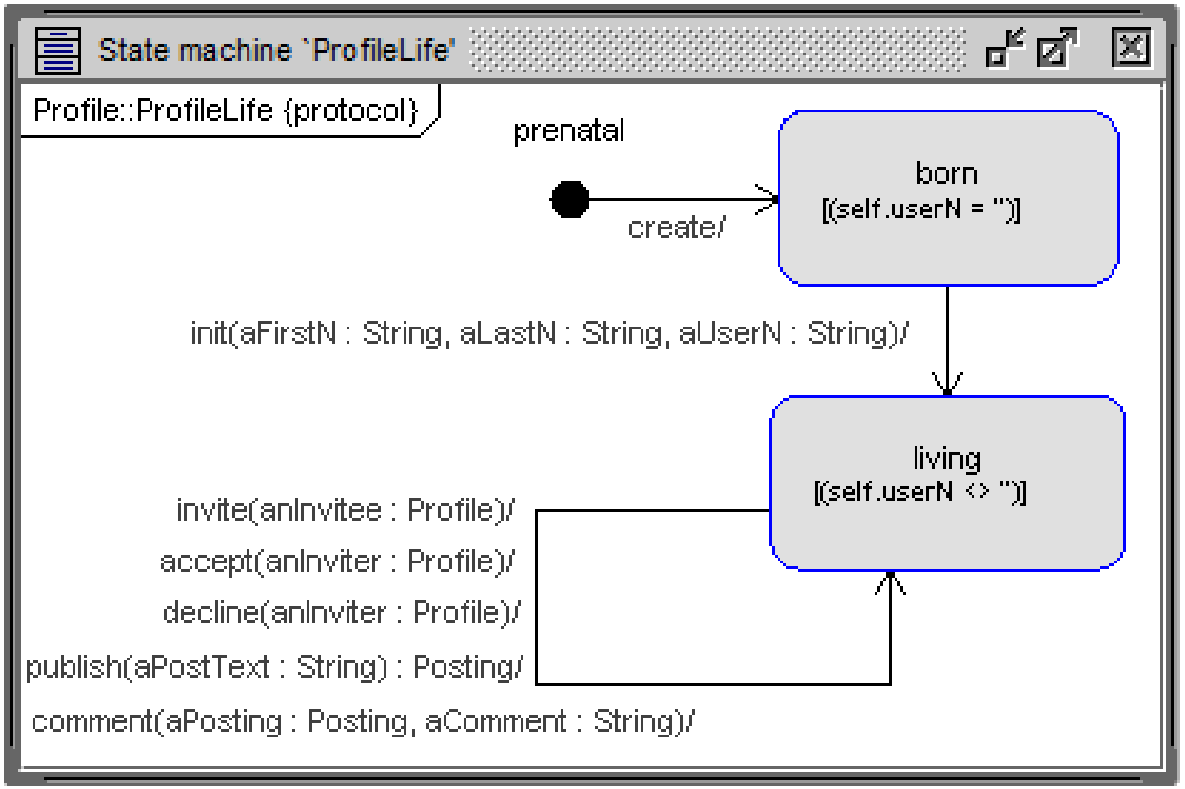


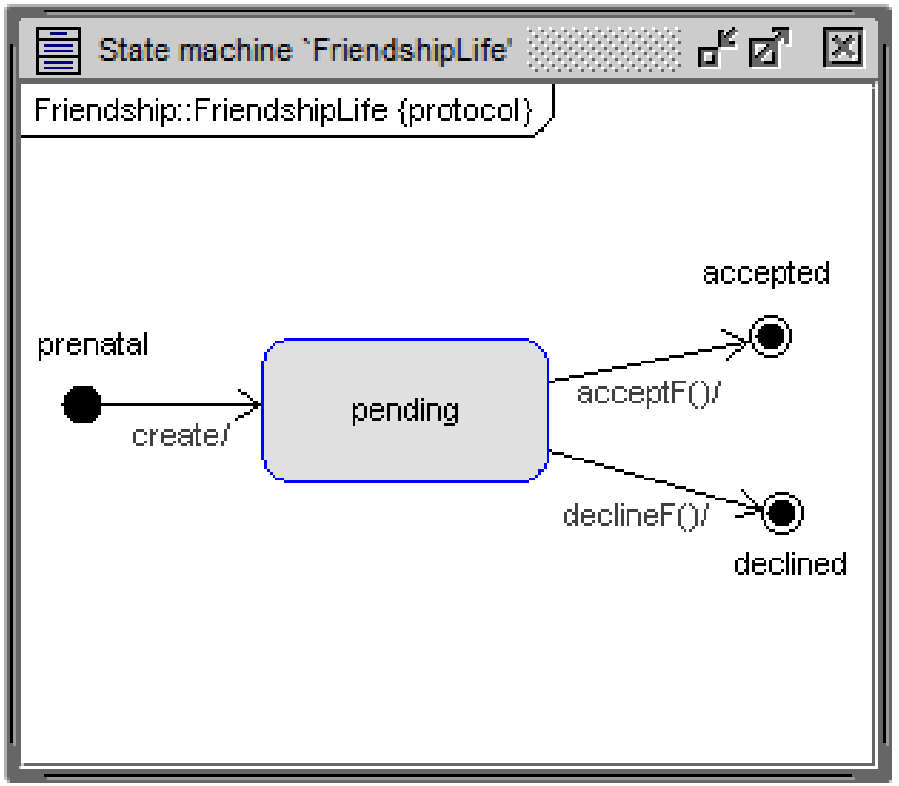
Use case diagram: example and concepts



- Actor, use case ("specification of an action sequence"), attributes
- Use case relationships: <<include>> (mandatory, once), <<extend>> (optional, repeatable), generalization
- Use case diagram not supported in USE

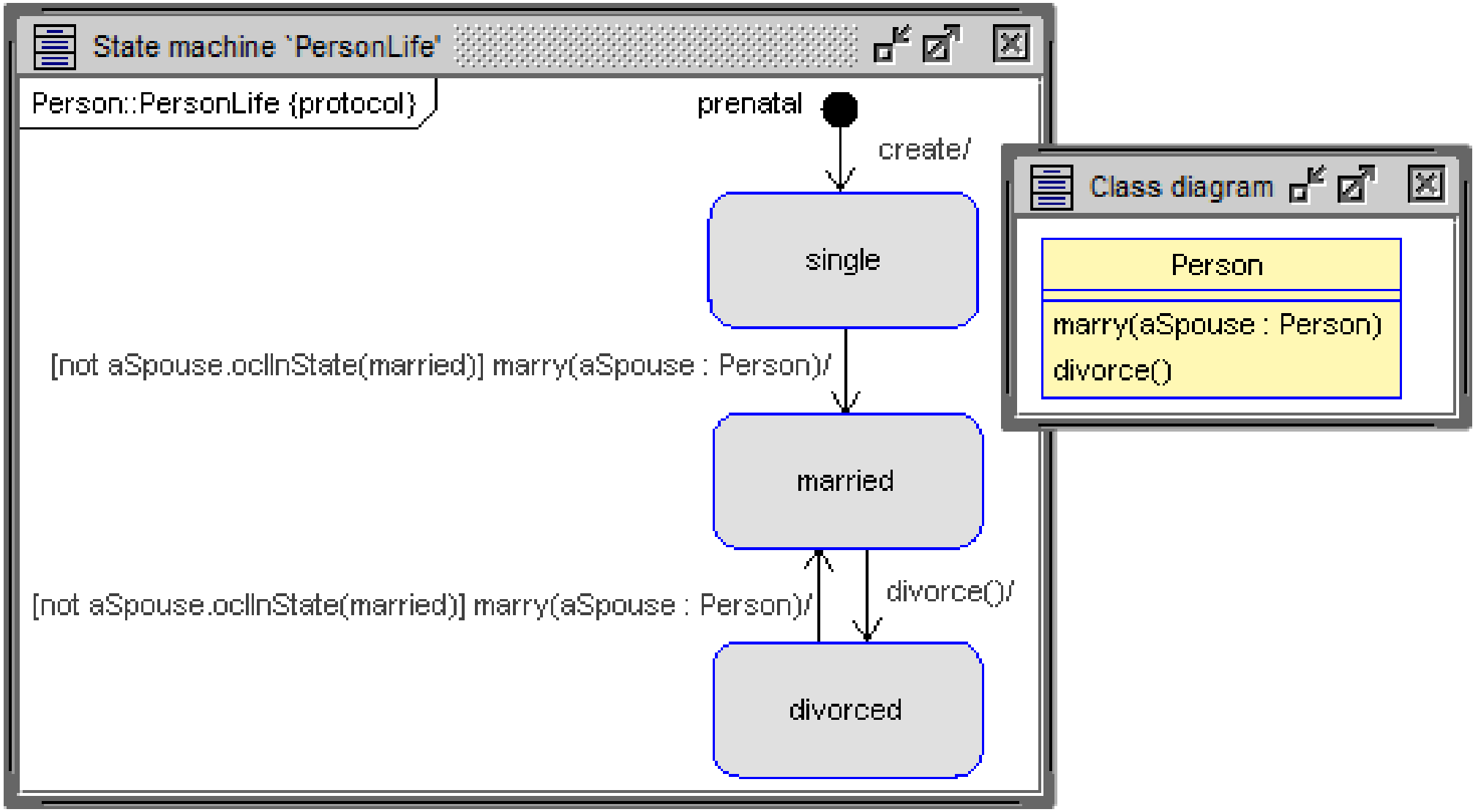
State chart diagram: example and concepts





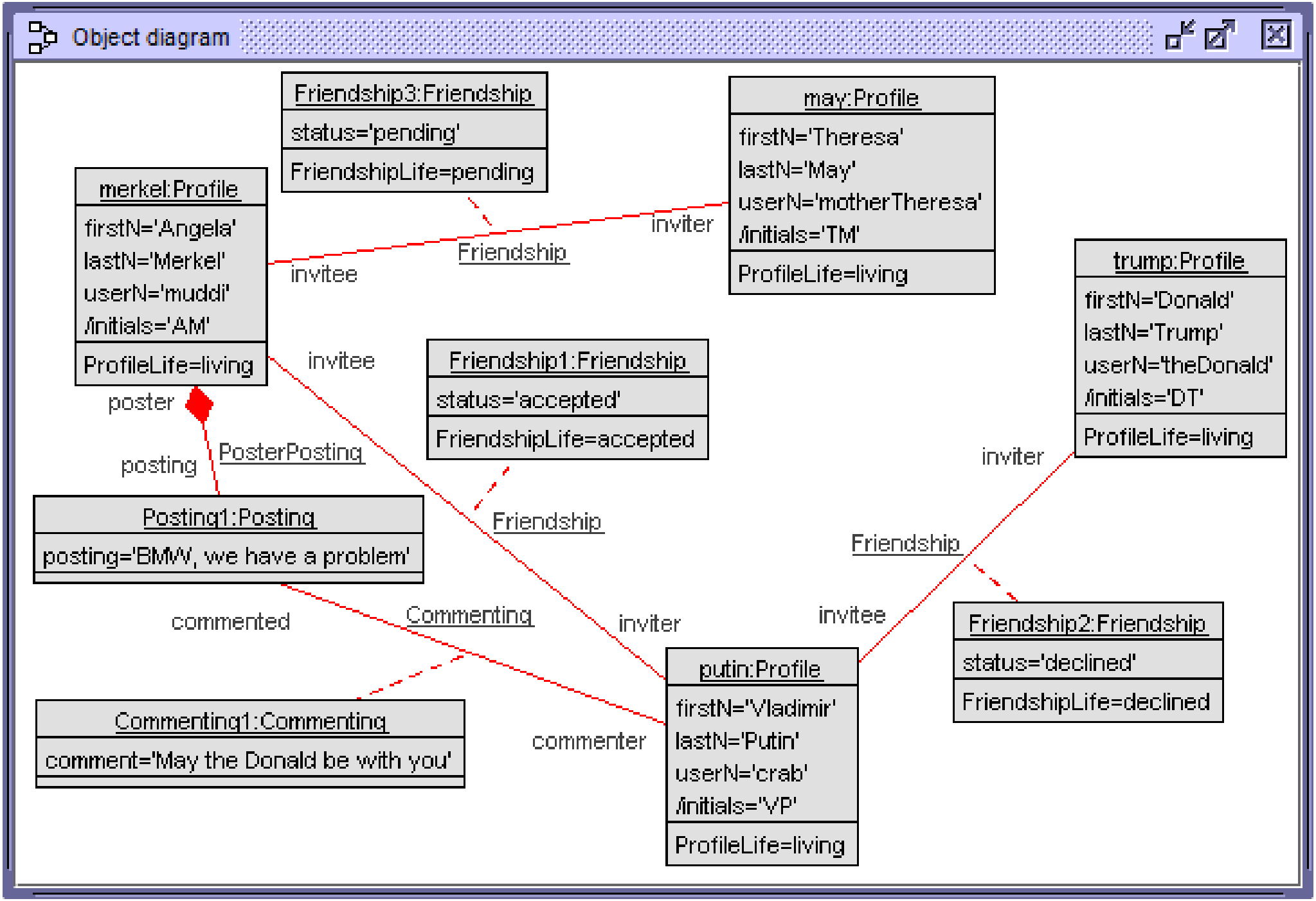
- State, state invariant
- Initial, final, normal state
- State transition: [guard] event / [postcondition]
- determine object life cycles
- determined here: protocol state machines; operation call sequences

State chart diagram example and concept 'guard'

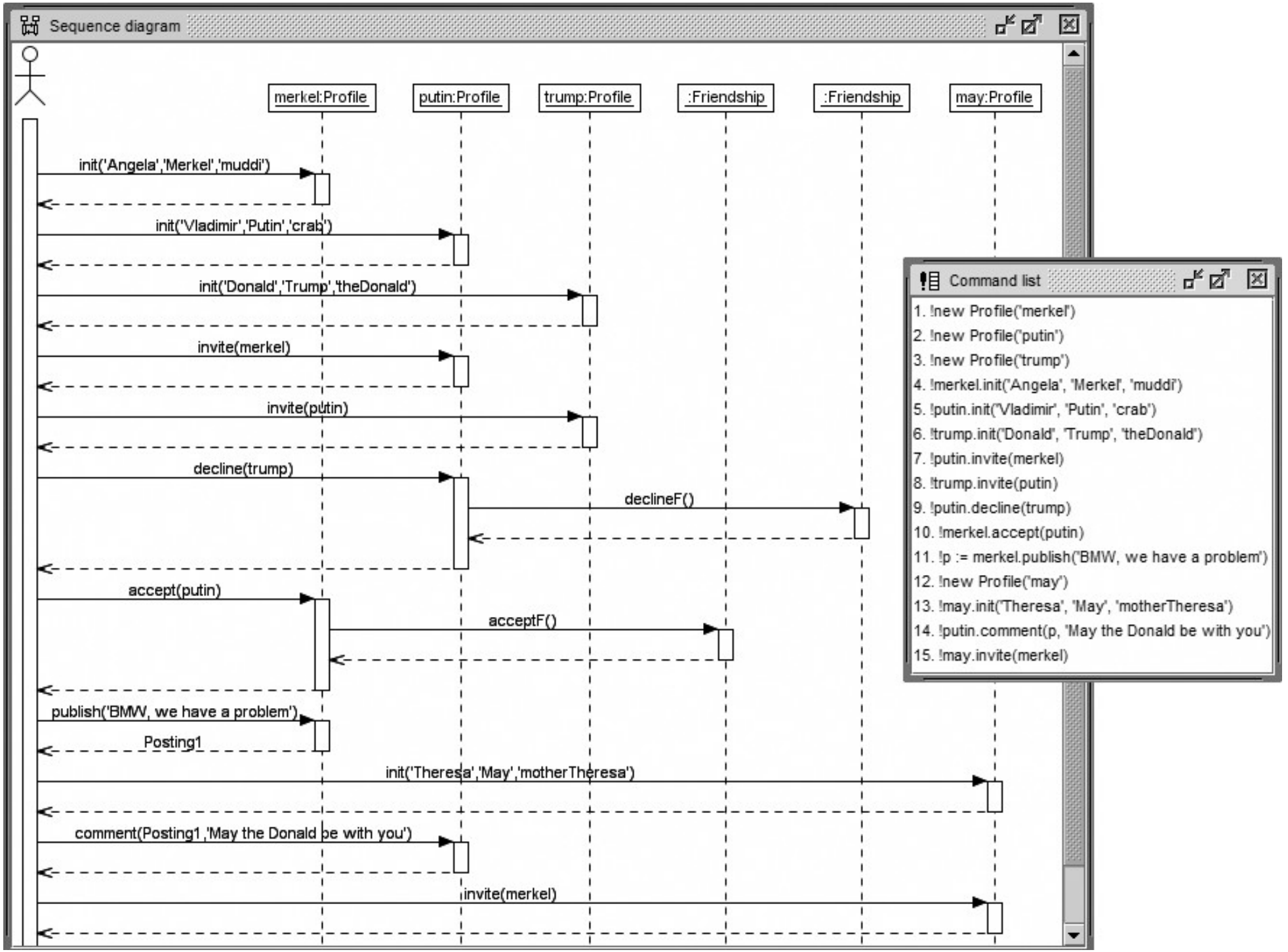


- State transition: [guard] event / [postcondition]
- guard, also called precondition:
  [not aSpouse.oclInState(married)] marry(aSpouse)

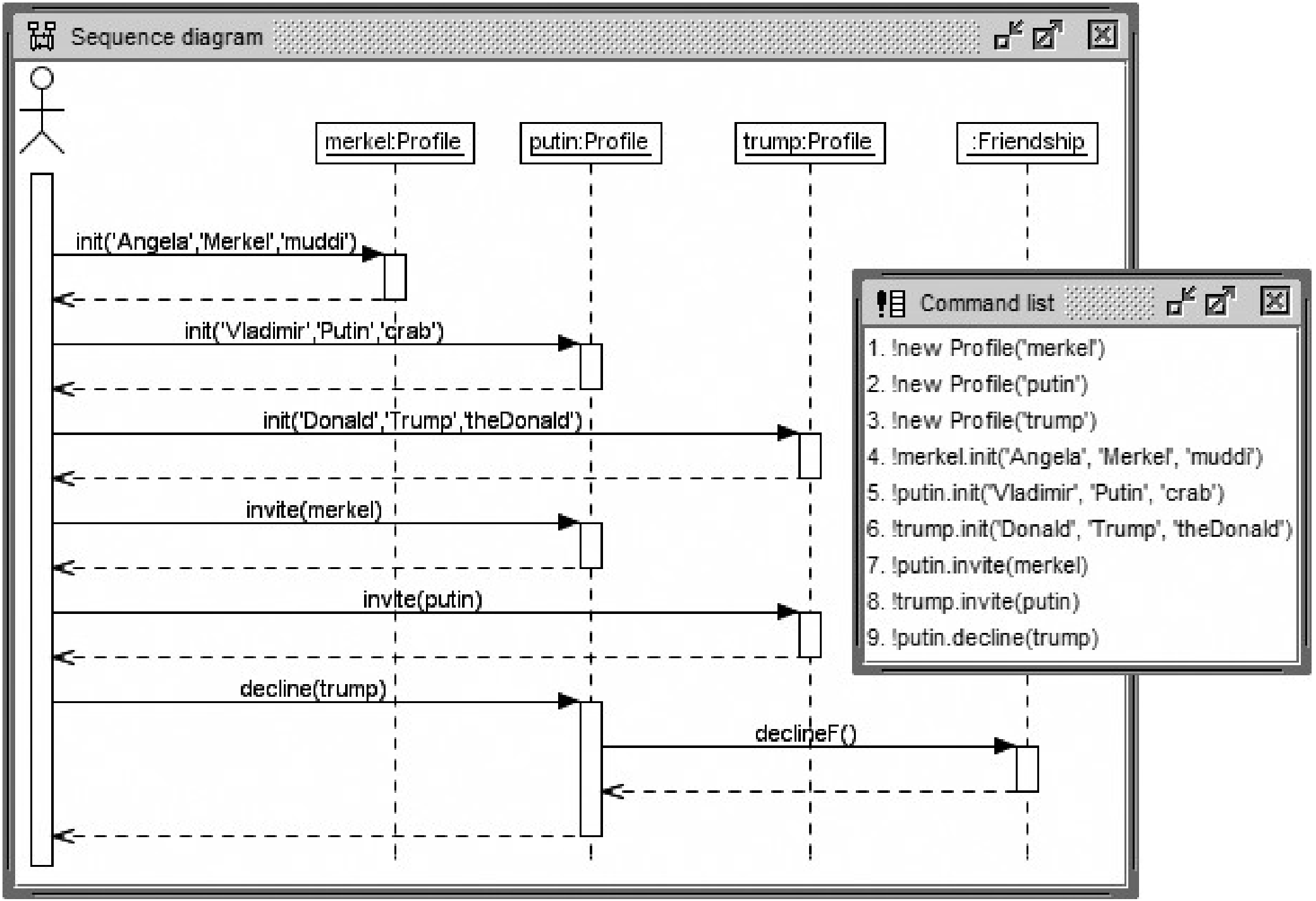
Object diagram with roles, association names, state chart status



Sequence diagram (large)



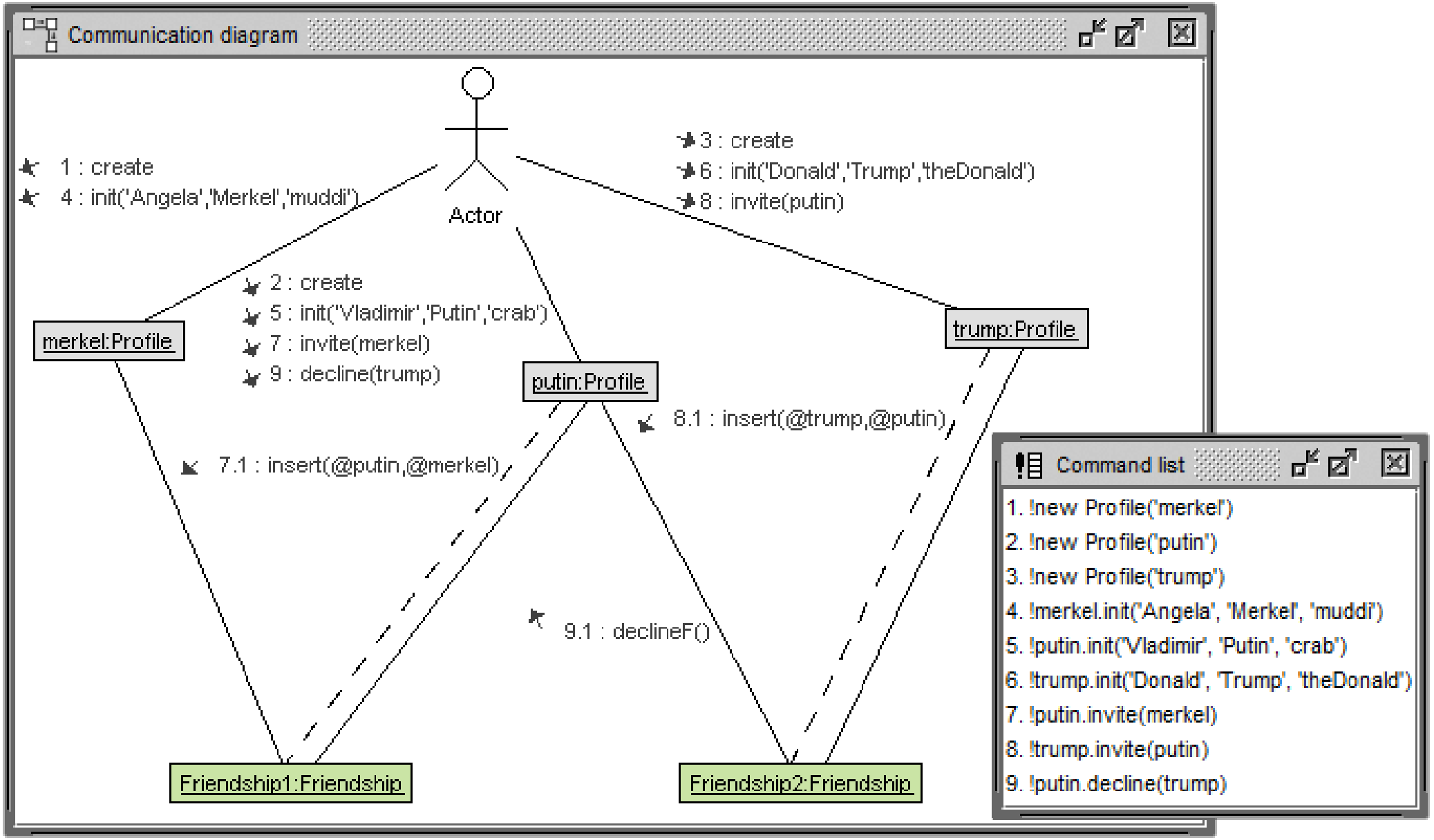
Sequence diagram (starting part of large example)



Sequence diagram concepts

- Object (or object role)
- Lifeline
- Activation
- Message representation
  - solid arrow from caller to callee indicating message call
  - dashed arrow from callee to caller indicating message completion, optional with return value
- Link representation: link shown as link object with lifeline

Communication diagram: example and concepts



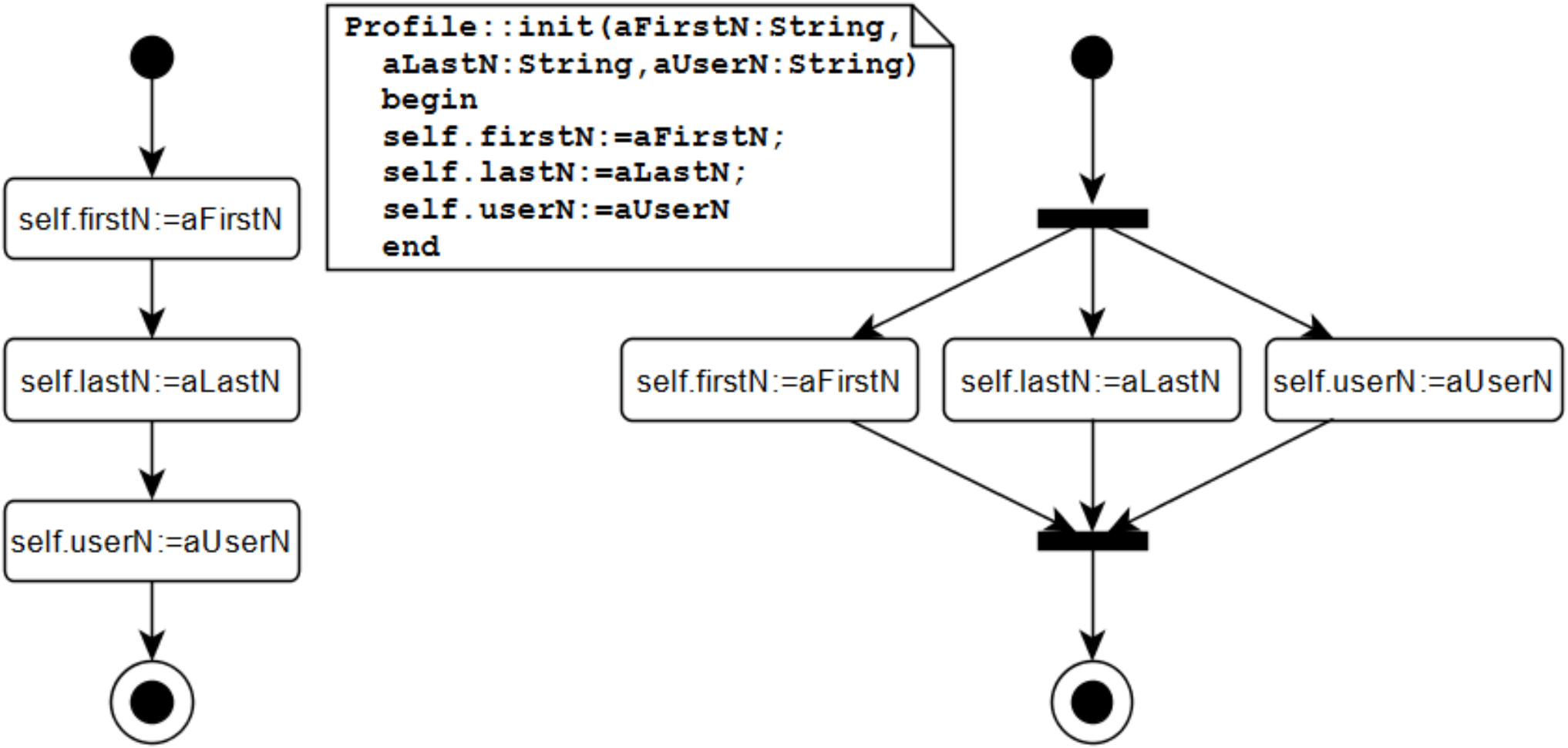
- Object, message, message number, link representation

Sequence vs Communication diagram

Both sequence diagrams and communication diagrams show interactions, but they emphasize different aspects. A sequence diagram shows time sequence as a geometric dimension, but the relationships among roles are implicit. A communication diagram shows the relationships among roles geometrically and relates messages to the connectors, but time sequences are less clear because they are implied by the sequence numbers. Each diagram should be used when its main aspect is the focus of attention.

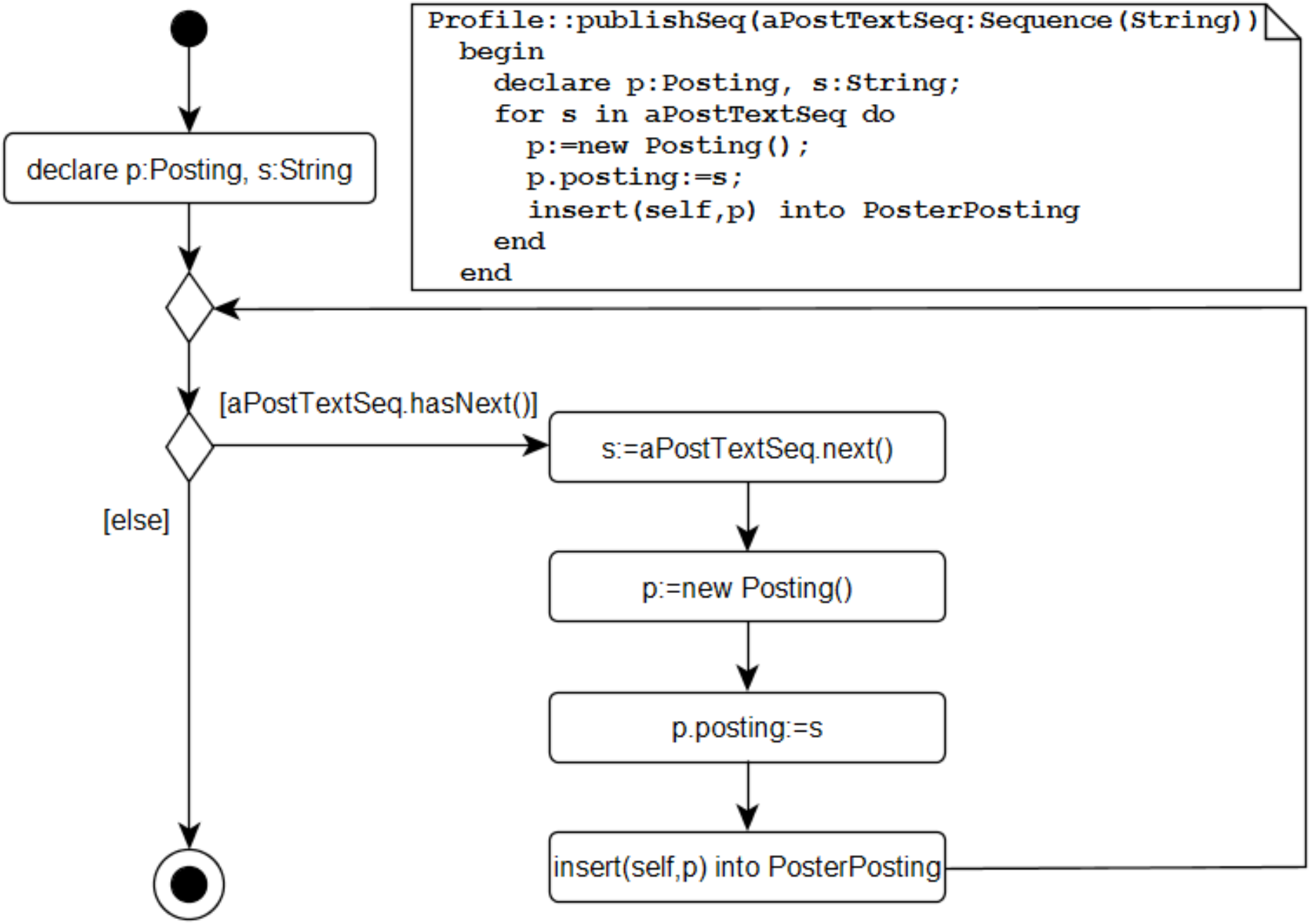
UML Reference Manual, p. 40

Activity diagram: example and concepts (A)



- Initial, final, action, fork, join node
- Activity diagram not supported in USE

Activity diagram: example and concepts (B)



- Merge, decision node

Template for textual model definition in USE (part A)

```
class Profile
attributes
  firstN:String init: ''
  ...
  initials:String derived:
    firstN.substring(1,1).concat(lastN.substring(1,1))
operations
  init(aFirstN:String, aLastN:String, aUserN:String)
    begin
    self.firstN:=aFirstN; self.lastN:=aLastN; self.userN:=aUserN
    end
    pre  aUserNNonEmpty: aUserN<>''
    post userNAssigned:  aUserN=userN
  ...
constraints
  inv uniqueUserName: Profile.allInstances->isUnique(userN)
  ...
statemachines
  psm ProfileLife
  states
    born      [userN='']
    ...
  transitions
    born      -> living { init() }
    ...
  end
end
```

Template for textual model definition in USE (part B)

```
associationclass Friendship between
  Profile [*] role inviter
  Profile [*] role invitee
attributes
  status:String init:'pending'
...
end

composition PosterPosting between
  Profile [1] role poster
  Posting [*] role posting
end

associationclass Commenting between
  Profile [*] role commenter
  Posting [*] role commented
attributes
  comment:String
...
end

association Interest between
  Profile [*]
  Subject [*]
end
```

Textual model definition in USE (complete model part A)

```
model SocialNetwork

class Profile
attributes
  firstN:String init: ''
  lastN:String init:  ''
  userN:String init:  ''
  initials:String derived:
    firstN.substring(1,1).concat(lastN.substring(1,1))
operations
  init(aFirstN:String, aLastN:String, aUserN:String)
    begin
    self.firstN:=aFirstN; self.lastN:=aLastN; self.userN:=aUserN end
    pre  aUserNNonEmpty: aUserN<>''
    post userNAssigned: aUserN=userN
  invite(anInvitee:Profile)
    begin new Friendship between (self,anInvitee) end
    pre  notAlreadyTried: invitee->union(inviter)->excludes(anInvitee)
    post madeFS: friendship[inviter]->
           select(oclInState(pending)).invitee->includes(anInvitee)
  accept(anInviter:Profile)
    begin self.friendship(anInviter).acceptF() end
    pre  pendingFS: friendship[invitee]->
           select(oclInState(pending)).inviter->includes(anInviter)
    post acceptedFS: friendship[invitee]->
           select(oclInState(accepted)).inviter->includes(anInviter)
```

Textual model definition in USE (complete model part B)

```
decline(anInviter:Profile)
  begin self.friendship(anInviter).declineF() end
  pre  pendingFS: friendship[invitee]->
         select(oclInState(pending)).inviter->includes(anInviter)
  post declinedFS: friendship[invitee]->
         select(oclInState(declined)).inviter->includes(anInviter)
publish(aPostText:String):Posting
  begin declare p:Posting;
  p:=new Posting(); p.posting:=aPostText;
  insert(self,p) into PosterPosting; result:=p
  end
  pre  nonEmpty: aPostText<>''
  post newPosting: Posting.allInstances->exists(p |
         p.posting=aPostText and result=p)
comment(aPosting:Posting,aComment:String)
  begin declare c:Commenting;
  c:=new Commenting between (self,aPosting); c.comment:=aComment
  end
  pre  aPostingNonNullACommentNonEmpty:
         aPosting<>null and aComment<>''
  post commentingExists: Commenting.allInstances->exists(c |
         c.comment=aComment and aPosting.commenting->includes(c) and
         self.commenting->includes(c))
```

Textual model definition in USE (complete model part C)

```
  friends():Set(Profile)=
    friendship[inviter]->select(oclInState(accepted)).invitee->union(
    friendship[invitee]->select(oclInState(accepted)).inviter)->asSet()
   friendship(anInviter:Profile):Friendship=
    self.friendship[invitee]->any(fs|fs.inviter=anInviter)
constraints
  inv asymmetricFriendship: invitee->intersection(inviter)->isEmpty()
  inv uniqueUserName: Profile.allInstances->isUnique(userN)
statemachines
  psm ProfileLife
  states
    prenatal:initial
    born     [userN='']
    living   [userN<>'']
  transitions
    prenatal -> born   { create }
    born     -> living { init() }
    living   -> living { invite() }
    living   -> living { accept() }
    living   -> living { decline() }
    living   -> living { publish() }
    living   -> living { comment() }
  end
end
```

Textual model definition in USE (complete model part D)

```
associationclass Friendship between
  Profile [*] role inviter
  Profile [*] role invitee
attributes
  status:String init:'pending'
operations
  acceptF()
    begin self.status:='accepted' end
  declineF()
    begin self.status:='declined' end
statemachines
  psm FriendshipLife
  states
    prenatal:initial
    pending
    accepted:final
    declined:final
  transitions
    prenatal -> pending  { create }
    pending  -> accepted { acceptF() }
    pending  -> declined { declineF() }
  end
end
```

Textual model definition in USE (complete model part E)

```
composition PosterPosting between
  Profile [1] role poster
  Posting [*] role posting
end

class Posting
attributes
  posting:String
end

associationclass Commenting between
  Profile [*] role commenter
  Posting [*] role commented
attributes
  comment:String
end

constraints

context Commenting inv commentOnlyByFriends:
  commented.poster.friends()->includes(commenter)
```

Textual model definition in USE (complete model part F)

```
class Subject
attributes
  subject:String
constraints
  inv noDuplicates:
    Subject.allInstances->size
    =
    Subject.allInstances.subject->asSet->size
end

association Interest between
  Profile [*]
  Subject [*]
end
```

Scenario / Test case definition on USE shell with SOIL statements

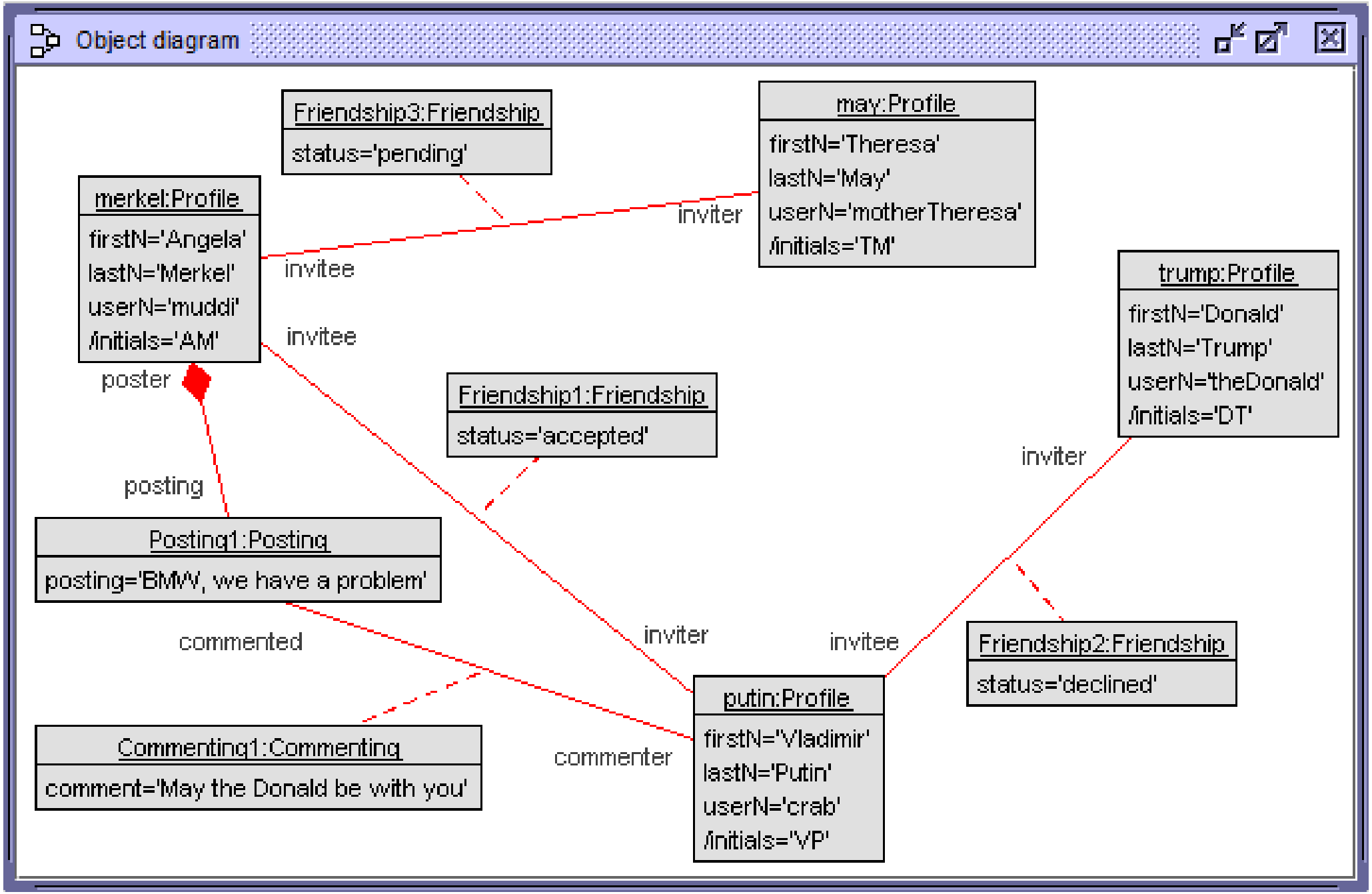
```
!create merkel,putin,trump:Profile
!merkel.init('Angela','Merkel','muddi')
!putin.init('Vladimir','Putin','crab')
!trump.init('Donald','Trump','theDonald')
!putin.invite(merkel)
!trump.invite(putin)
!putin.decline(trump)
!merkel.accept(putin)
!p:=merkel.publish('BMW, we have a problem')
!create may:Profile
!may.init('Theresa','May','motherTheresa')
!putin.comment(p,'May the Donald be with you')
!may.invite(merkel)
```

- Object creation
- Operation call on object
- Variable assignment
- Variable access

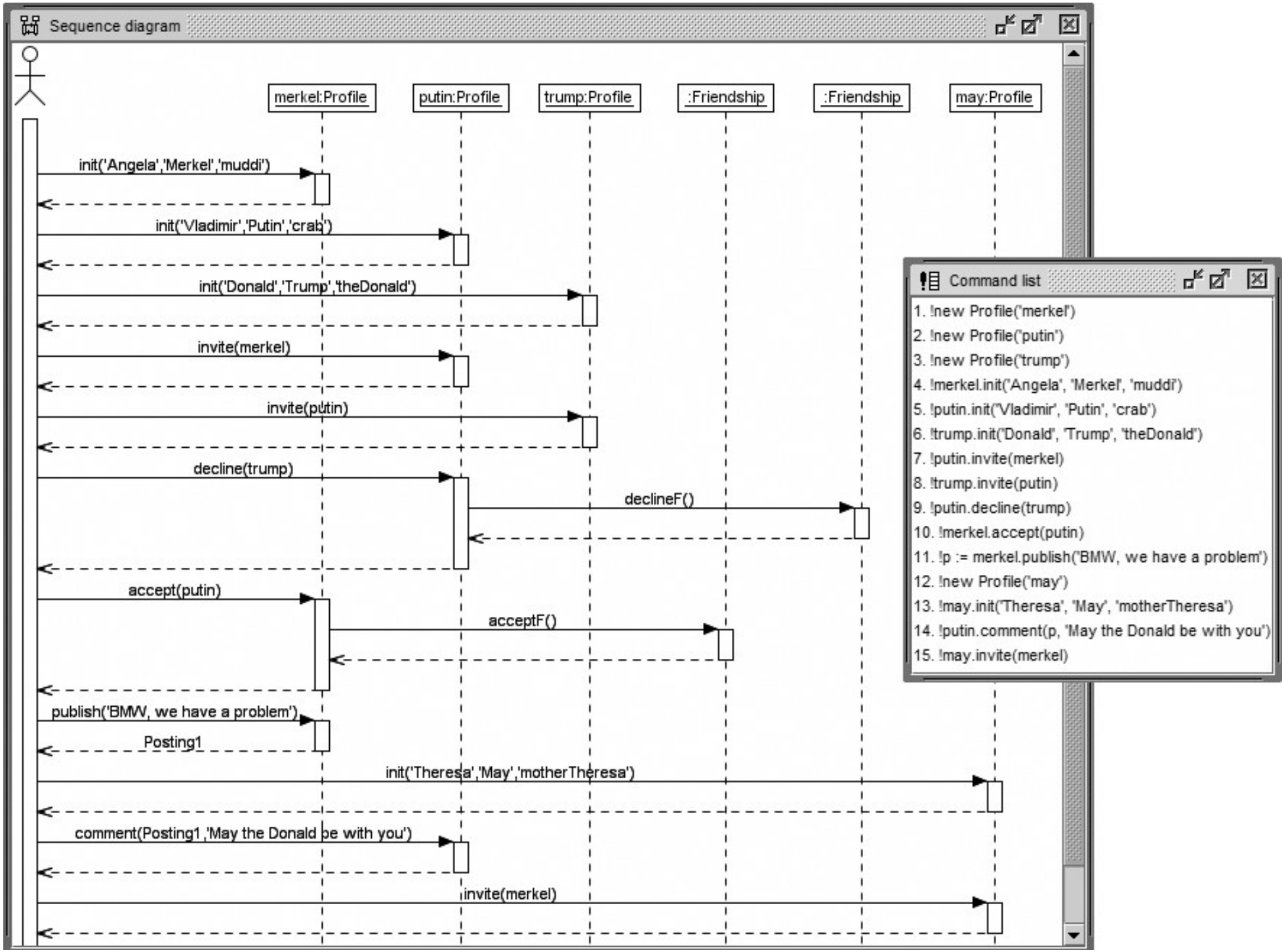
Overview SOIL statements

- SOIL: Simple Ocl-like Imperative Language
- Object creation and destruction; link creation and destruction:
  'create' / 'new', 'destroy', 'insert', 'delete'
- Variable declaration 'declare' v1:OclType, v2:OclType, … ;
  assignment ':=' with OCL expression
- Loops/conditionals:
  'for' var 'in' collection 'do' ... 'end';
  'while' cond 'do' … 'end';
  'if' cond 'then' ... ['else' ...] 'end';
- (Recursive) operation calls: object.operation(parameters)
- SOIL statements
  - in USE file for operation definition
    with 'declare'
  - on USE shell for adhoc actions starting with '!'
    without variable declaration

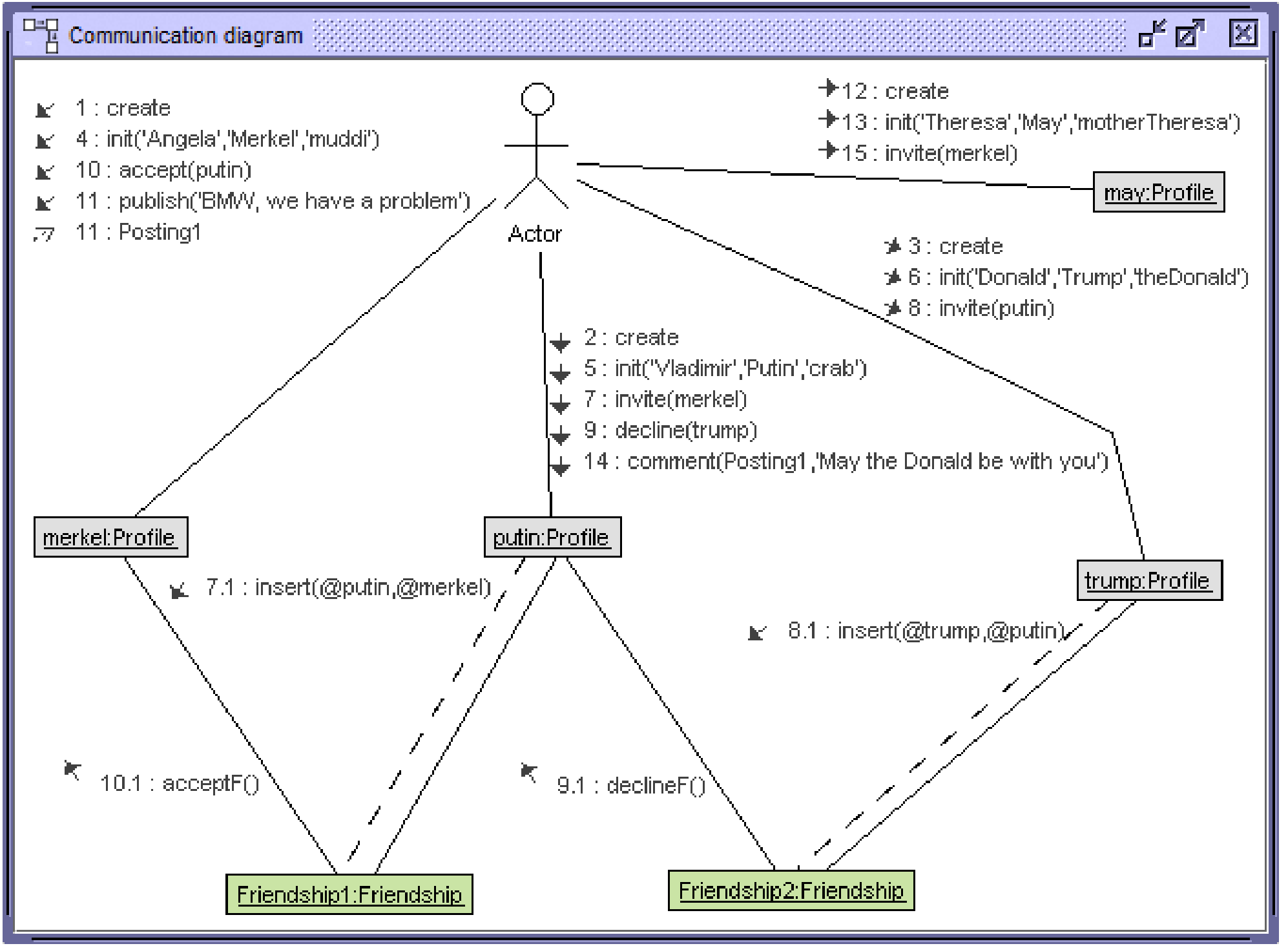
Object diagram

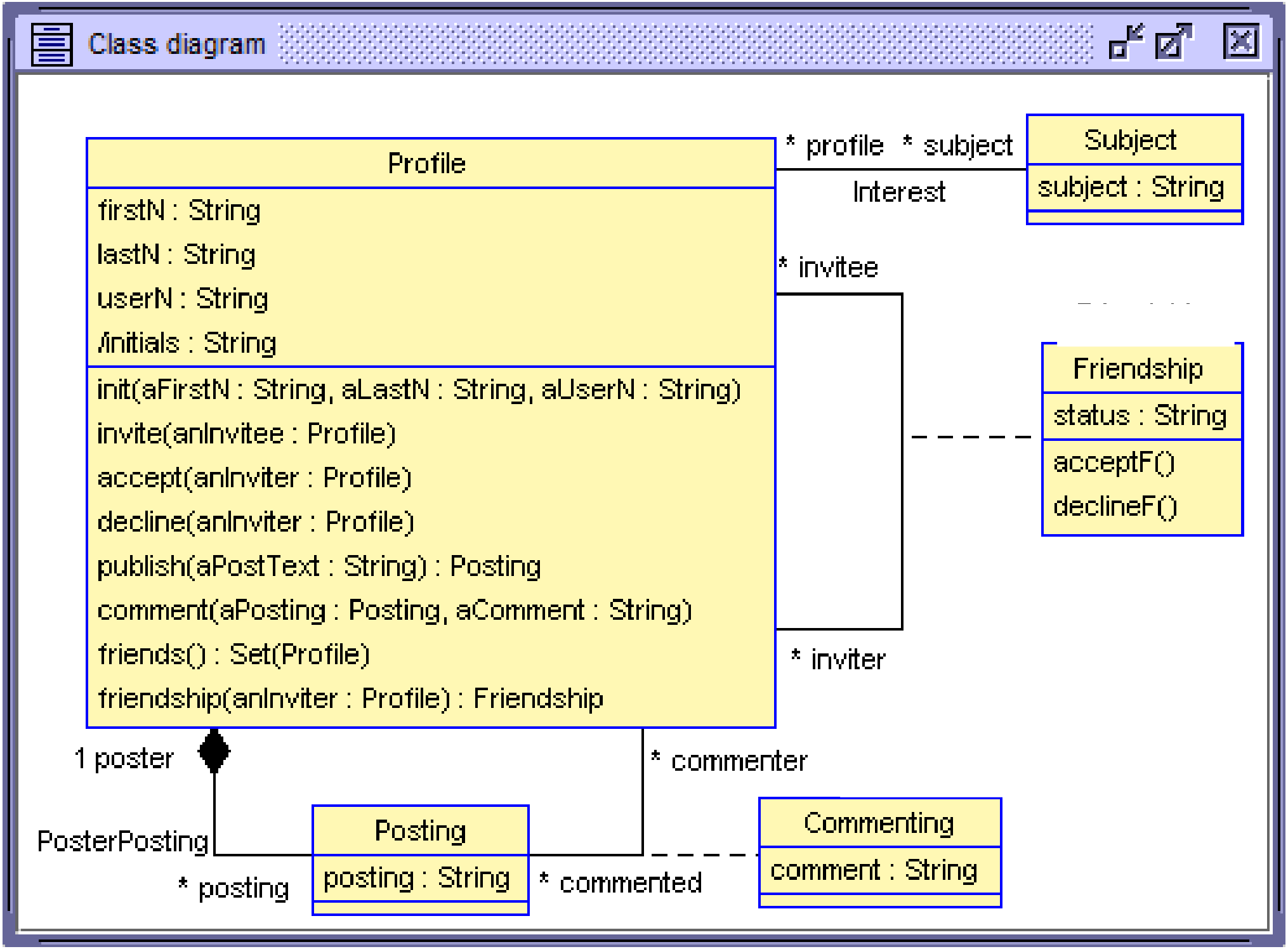


Sequence diagram

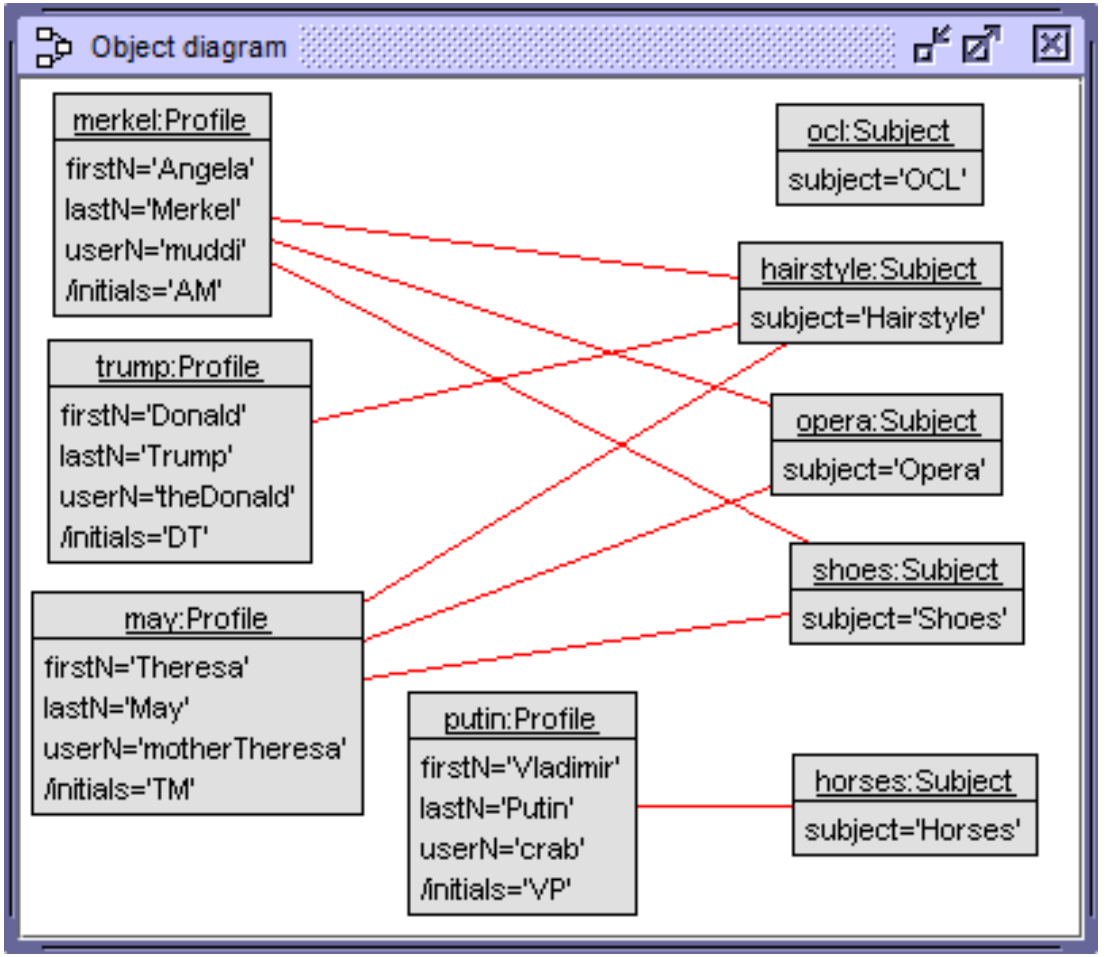


Communication diagram




Class diagram
Profile
firstN : String
lastN : String
userN : String
/initials : String
init(aFirstN : String, aLastN : String, aUserN : String)
invite(anInvitee : Profile)
accept(anInviter : Profile)
decline(anInviter : Profile)
publish(aPostText : String) : Posting
comment(aPosting : Posting, aComment : String)
friends() : Set(Profile)
friendship(anInviter : Profile) : Friendship
* profile
* subject
Interest
Subject
subject : String
* invitee
* inviter
Friendship
status : String
acceptF()
declineF()
1 poster
PosterPosting
* posting
Posting
posting : String
* commenter
* commented
Commenting
comment : String

Evaluation of OCL expressions (Part A)




```
-- allInstances ---------------

?Profile.allInstances
Set{may,merkel,putin,trump}:Set(Profile)

-- includes, excludes ----------------------------------------
?Set{may,merkel,trump}->includes(putin)
false

?Set{putin}->excludes(trump)
true
```

Evaluation of OCL expressions (Part B)

```
-- select, reject ----------------------------------------------

?Profile.allInstances->select(p | p.subject->includes(opera))
Set{may,merkel}:Set(Profile)

?Profile.allInstances->reject(p | p.subject->includes(hairstyle))
Set{putin}:Set(Profile)

-- size, isEmpty, notEmpty -------------------------------------

?Profile.allInstances->select(p | p.subject->size=3)
Set{may,merkel}:Set(Profile)

?Subject.allInstances->select(s | s.profile->size=0)
Set{ocl}:Set(Subject)

?Subject.allInstances->select(s | s.profile->isEmpty)
Set{ocl}:Set(Subject)

?Subject.allInstances->select(s | s.profile->notEmpty)
Set{hairstyle,horses,opera,shoes}:Set(Subject)
```

Evaluation of OCL expressions (Part C)

```
-- forAll, exists ----------------------------------------

?Subject.allInstances->forAll(s | s.profile->notEmpty)
false:Boolean

?Profile.allInstances->select(p | Subject.allInstances->exists(s1,s2 |
   s1<>s2 and s1.profile->includes(p) and s2.profile->includes(p)))
Set{may,merkel}:Set(Profile)

?Profile.allInstances->select(p | Subject.allInstances->exists(s1,s2 |
   s1<>s2 and p.subject->includes(s1) and p.subject->includes(s2)))
Set{may,merkel}:Set(Profile)

-- collectNested, collect, asSet -------------------------------

?Profile.allInstances->collectNested(p | p.subject)
Bag{Set{hairstyle},
    Set{horses},                          ?Set{-2,0,2}->collect(i | i*i)
    Set{hairstyle,opera,shoes},                                 Bag{0,4,4}
    Set{hairstyle,opera,shoes}}:Bag(Set(Subject))

?Profile.allInstances->collect(p | p.subject)
Bag{hairstyle,hairstyle,hairstyle,horses,opera,opera,shoes,shoes}:
  Bag(Subject)

?Profile.allInstances->collect(p | p.subject)->asSet     -- SQL distinct
    Set{hairstyle,horses,opera,shoes} : Set(Subject)
```

Evaluation of OCL expressions (Part D)

```
-- including, excluding ----------------------------------------

?Set{putin,merkel}->including(may)
Set{may,merkel,putin}: Set(Profile)

?Set{putin,merkel}->excluding(putin)
Set{merkel}: Set(Profile)

?Set{putin,merkel}->excluding(may)
Set{merkel,putin}: Set(Profile)

?Bag{opera,shoes,opera}->including(shoes)
Bag{opera,opera,shoes,shoes}: Bag(Subject)

?Bag{opera,shoes,opera}->excluding(opera) -- excluding radical on Bag(T)
Bag{shoes}: Bag(Subject)

-- ( includesAll, excludesAll ) = ( 'supersetOf', 'disjointFrom' ) ---

?Set{opera,shoes,hairstyle}->includesAll(Set{opera,hairstyle}) = true
?Set{opera,shoes,hairstyle}->includesAll(Set{opera,horses}) = false

?Set{opera,shoes}->excludesAll(Set{horses,hairstyle}) = true
?Set{opera,shoes}->excludesAll(Set{horses,opera}) = false
```

Evaluation of OCL expressions (Part E)

```
-- let, Tuple, product ------------------------------------------

-- Profile objects having interest in a given set of subjects
?let INTEREST=Set{hairstyle,opera} in Profile.allInstances->select(p |
   INTEREST->forAll(s | p.subject->includes(s)))
Set{may,merkel}:Set(Profile)

-- Profile objects with a maximum number of interests
?let MAX=Profile.allInstances->collect(p | p.subject->size)->max() in
   Profile.allInstances->select(p | p.subject->size=MAX)
Set{may,merkel}:Set(Profile)

-- Profile object pairs with the same set of interests
?Profile.allInstances->product(Profile.allInstances)
Set{Tuple{first=may,second=may},
    Tuple{first=may,second=merkel},
    ...
    Tuple{first=trump,second=trump}} :
      Set(Tuple(first:Profile,second:Profile)) -- 16 tuples

?Profile.allInstances->product(Profile.allInstances)->
  select(t | t.first.subject=t.second.subject and t.first<>t.second)
Set{Tuple{first=may,second=merkel},Tuple{first=merkel,second=may}}:
  Set(Tuple(first:Profile,second:Profile))
```

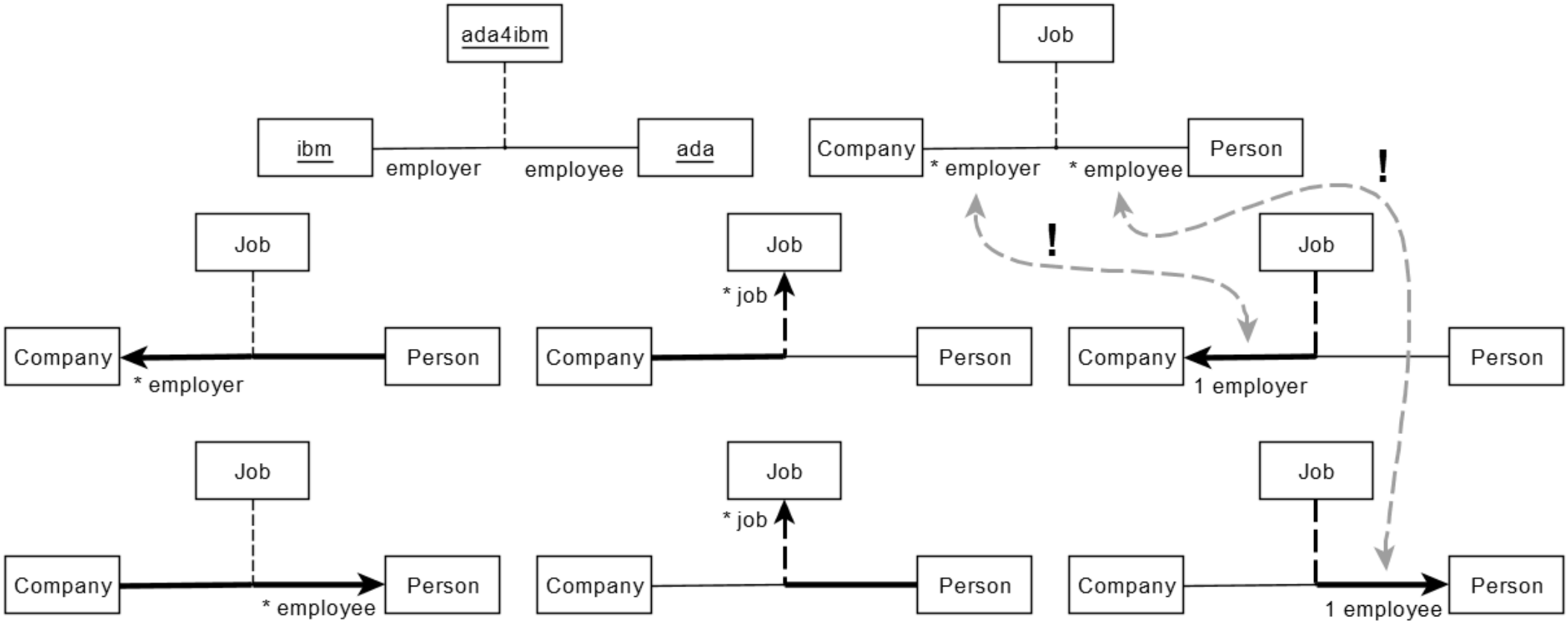
Evaluation of OCL expressions (Part F)

```
-- navigation in plain association class ----------------------------
```

```
ada.employer->
includes(ibm)
```

```
ibm.job->
includes(ada4ibm)
```

```
ada4ibm.employer=
ibm
```



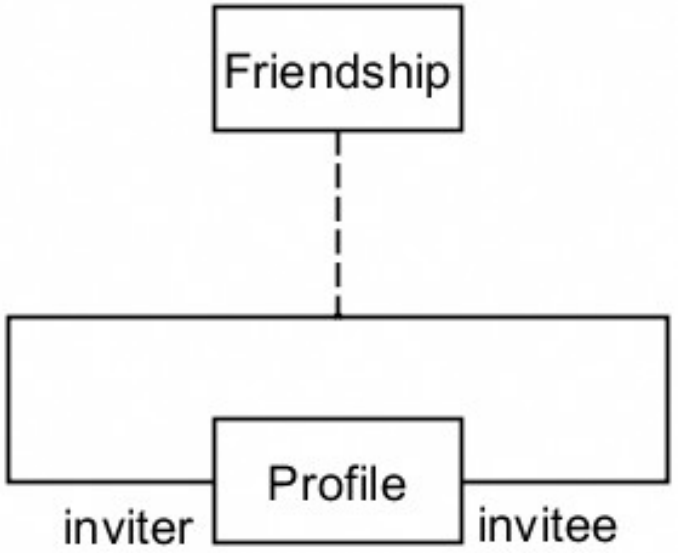

```
ibm.employee->
includes(ada)
```

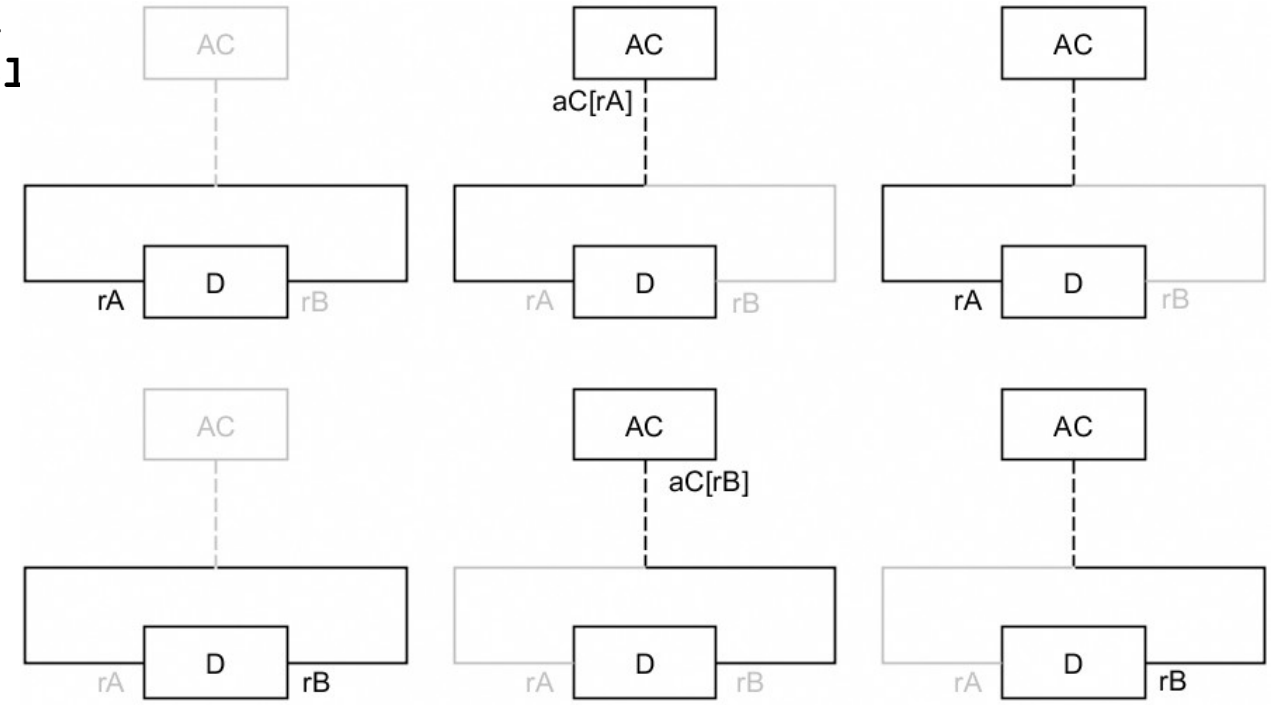
```
ada.job->
includes(ada4ibm)
```

```
ada4ibm.employee=
ada
```

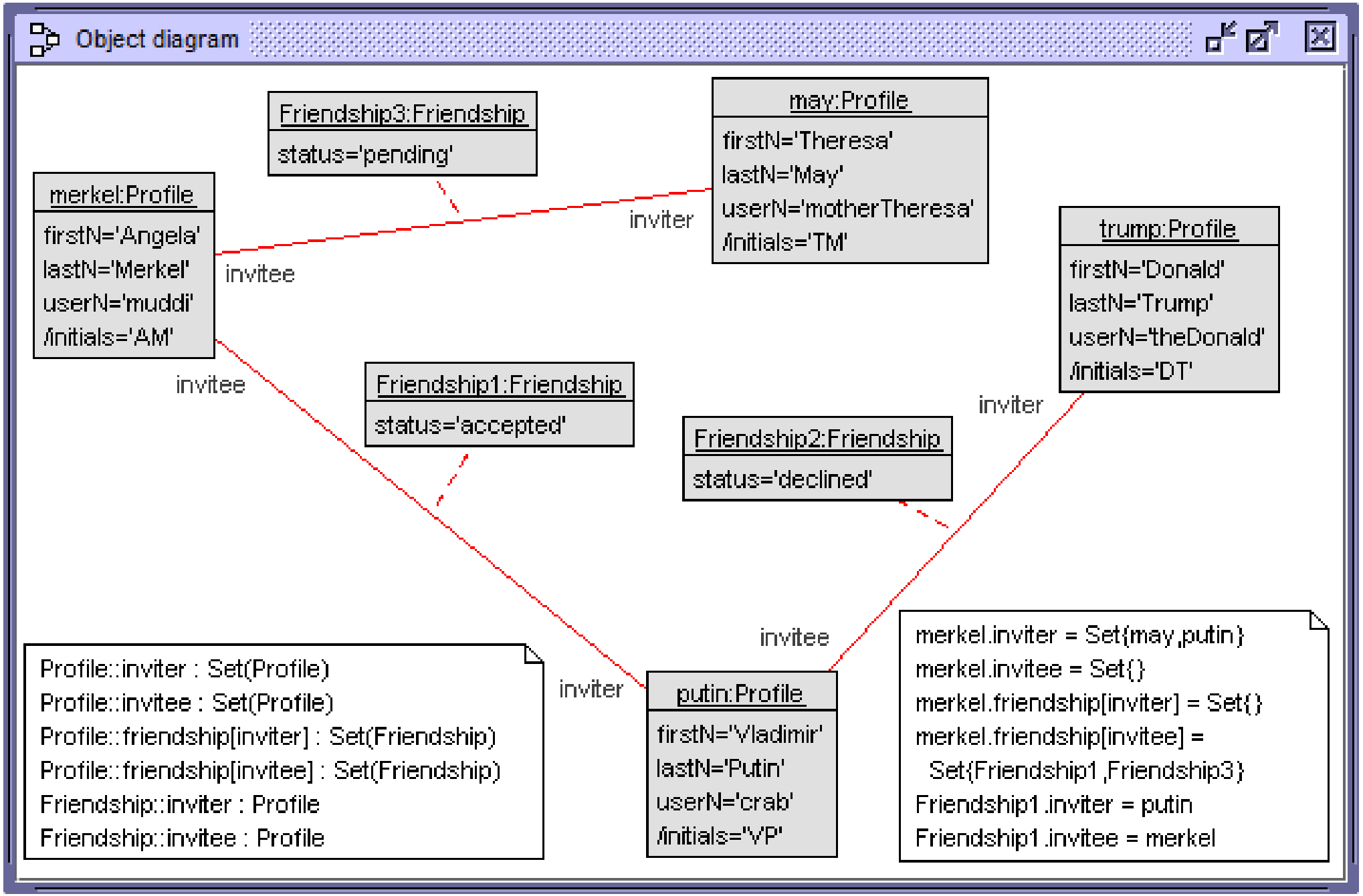
Evaluation of OCL expressions (Part G)

```
-- navigation in reflexive association class ------------------------

Profile::inviter : Set(Profile)
Profile::invitee : Set(Profile)
Profile::friendship[inviter] : Set(Friendship)
Profile::friendship[invitee] : Set(Friendship)
Friendship::inviter : Profile
Friendship::invitee : Profile

merkel.inviter = Set{may,putin}
merkel.invitee = Set{} : Set(Profile)
merkel.friendship[inviter] = Set{} : Set(Friendship)
merkel.friendship[invitee] = Set{Friendship1,Friendship3}
Friendship1.inviter = putin
Friendship1.invitee = merkel
```





Evaluation of OCL expressions (Part H)



Evaluation of OCL expressions (Part I)

```
-- dot shortcut ---------------------------------------------

?merkel.inviter                    ?merkel.inviter->collect(p | p.userN)
Set{may,putin}: Set(Profile)       Bag{'crab','motherTheresa'}: Bag(String)

?merkel.inviter.userN -- dot shortcut on single object
Bag{'crab','motherTheresa'}: Bag(String)

?Set{merkel}.inviter.userN -- dot shortcut on object collection
Bag{'crab','motherTheresa'}: Bag(String)

?Profile.allInstances.inviter.userN -- dot shortcut on object collection
Bag{'crab','motherTheresa','theDonald'}: Bag(String) -- excludes 'muddi'

?trump.invitee.invitee.inviter -- long path; object-valued result
Bag{may,putin}: Bag(Profile)

?trump.invitee.invitee.posting.posting -- long path; data-valued result
Bag{'BMW, we have a problem'}: Bag(String)

?Posting1.poster.initials -- respect multiplicities; gives single value
'AM' : String

?Posting1.poster.inviter -- respect multiplicities; gives Set(T)
Set{may,putin}: Set(Profile)
```

Derived attributes, query operations, invariants: Applying OCL

Derived attributes

```
Profile::initials:String derived:
  firstN.substring(1,1).concat(lastN.substring(1,1))
```

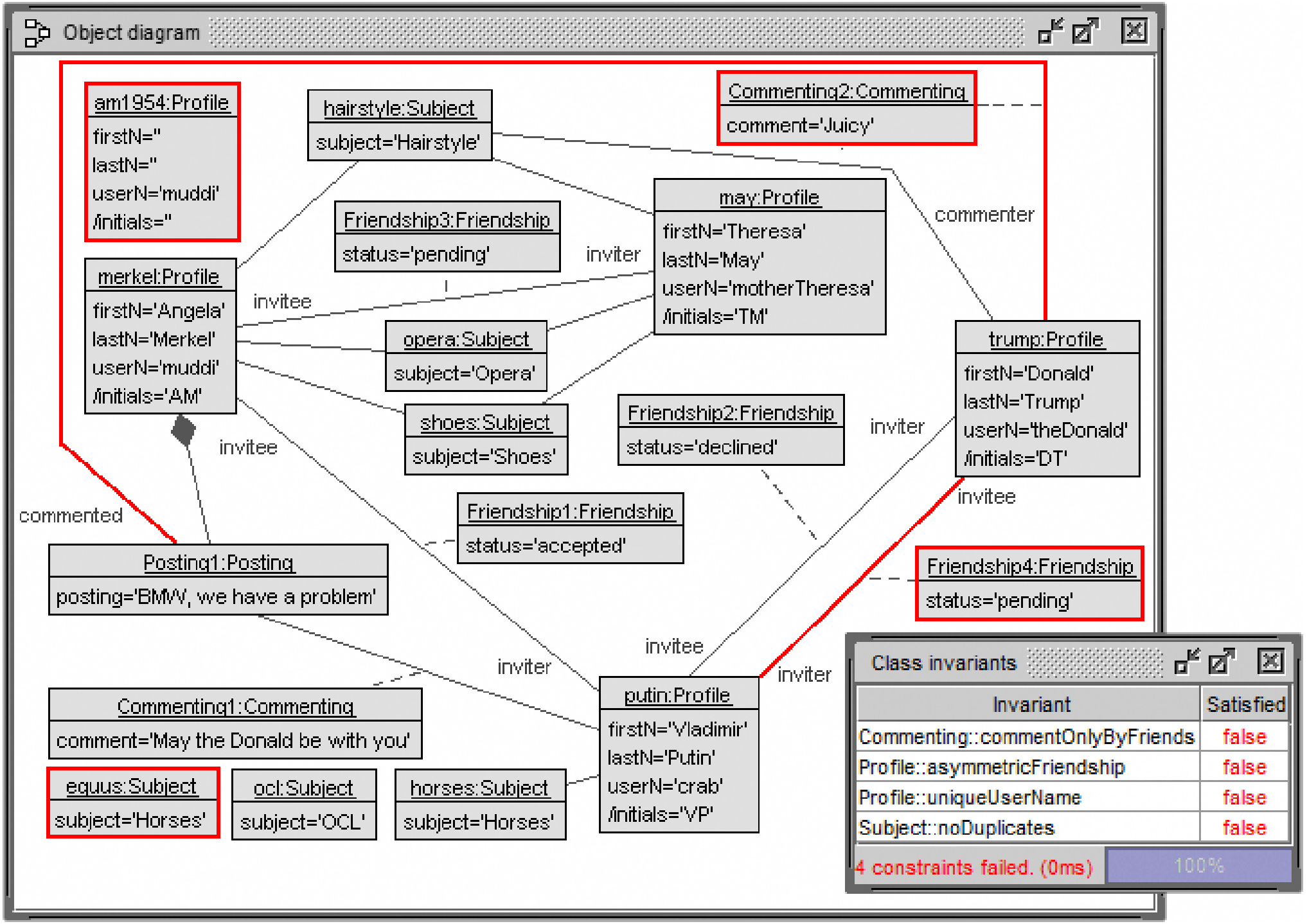
Query operations

```
Profile::friends():Set(Profile)=
  friendship[inviter]->select(oclInState(accepted)).invitee->union(
    friendship[invitee]->select(oclInState(accepted)).inviter)->asSet
Profile::friendship(anInviter:Profile):Friendship=
  friendship[invitee]->any(fs|fs.inviter=anInviter)
```

Invariants

```
context Profile
  inv asymmetricFriendship: invitee->intersection(inviter)->isEmpty()
  inv uniqueUserName: Profile.allInstances->isUnique(userN)

context Commenting inv commentOnlyByFriends:
  commented.poster.friends()->includes(commenter)

context Subject inv noDuplicates:
  Subject.allInstances->size=Subject.allInstances.subject->asSet->size
```

Object diagram with violated invariants



Thanks for your attention!