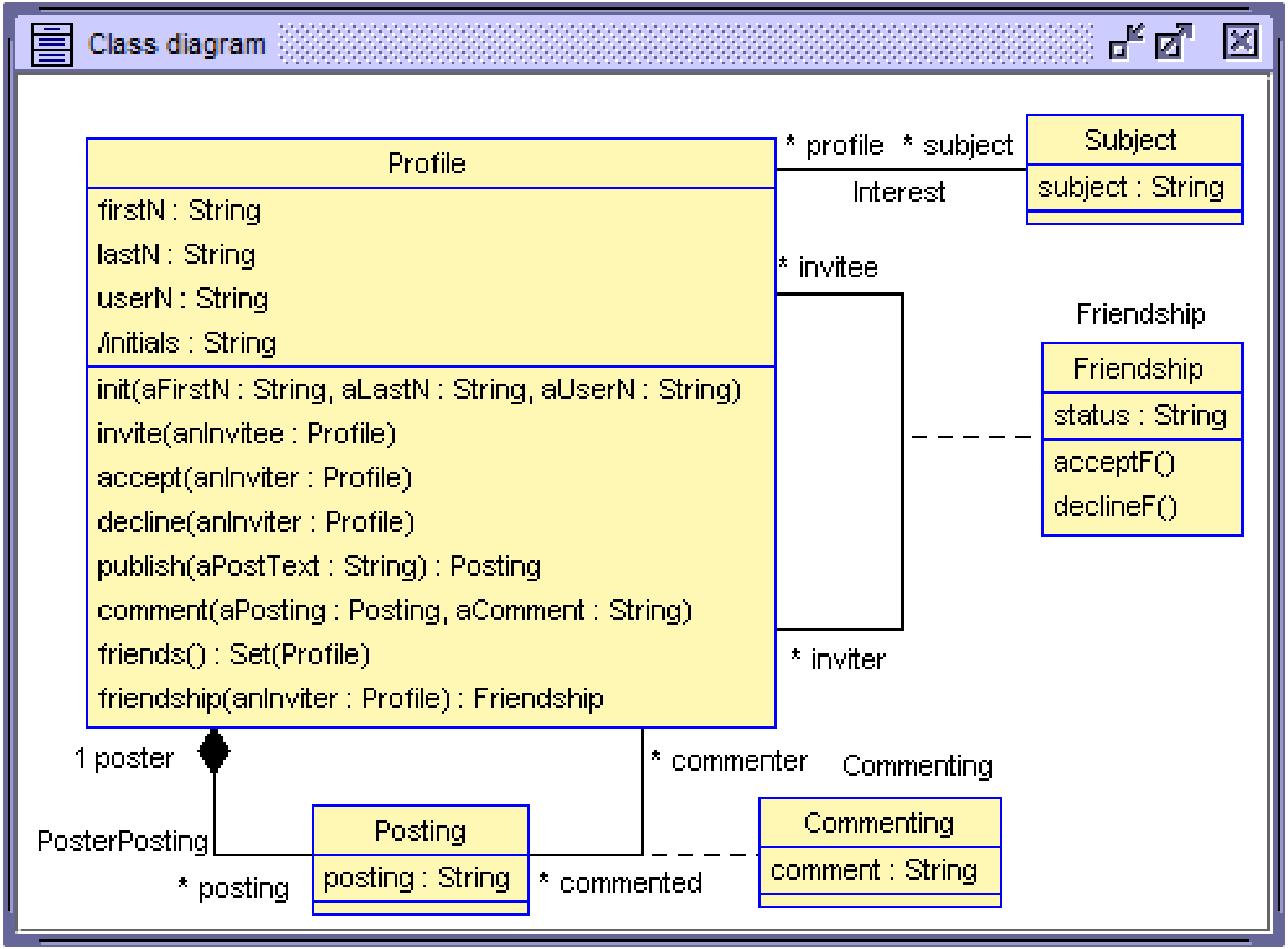
Animation/Documentation and Variations for the Social Network Model

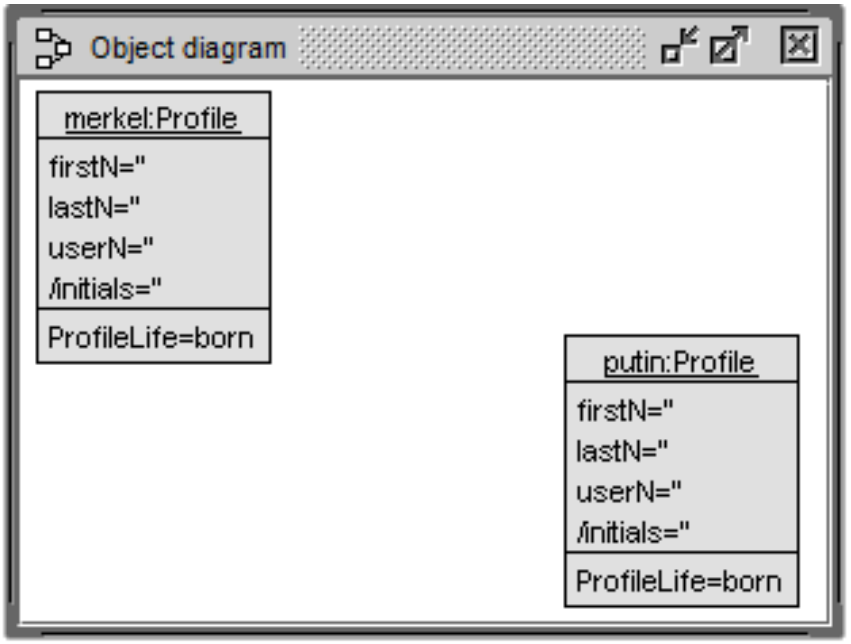

Class diagram
Profile
firstN : String
lastN : String
userN : String
/initials : String
init(aFirstN : String, aLastN : String, aUserN : String)
invite(anInvitee : Profile)
accept(anInviter : Profile)
decline(anInviter : Profile)
publish(aPostText : String) : Posting
comment(aPosting : Posting, aComment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

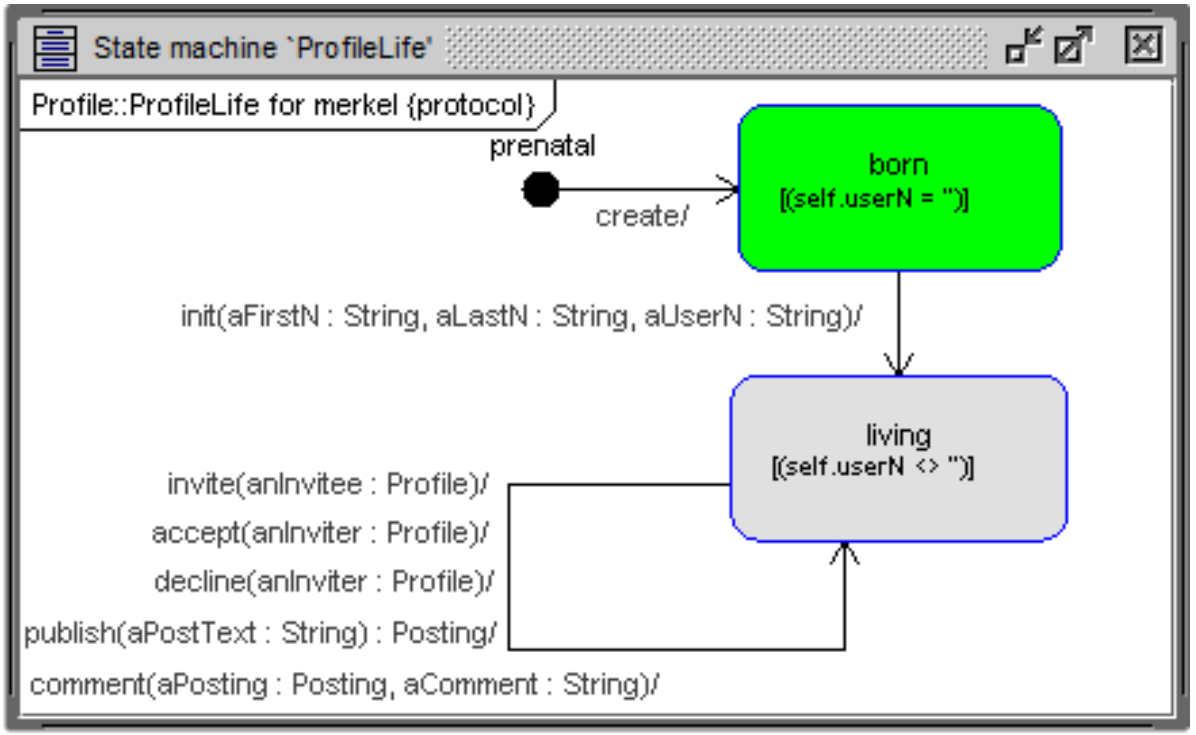
Animation and documentation of a simple scenario

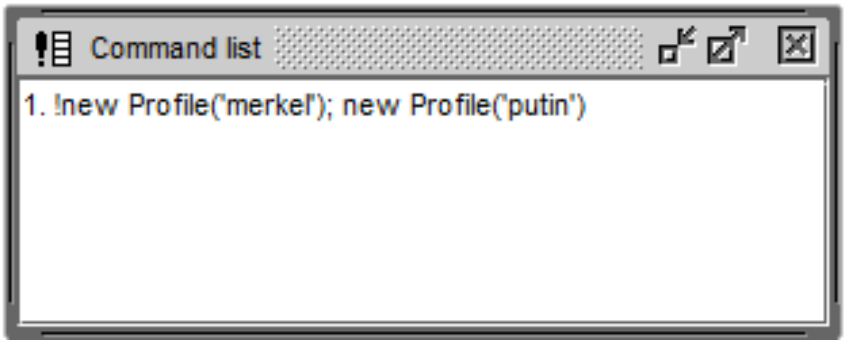
- Scenario: creation of two profiles and establishment of a friendship association link between them
- UML diagrams used: object, statechart, sequence and communication diagram together with USE command list
- Statechart diagrams together with object diagrams and USE command list
- Sequence diagrams together with USE command list
- Communication diagrams together with USE command list
- OCL queries on the USE shell

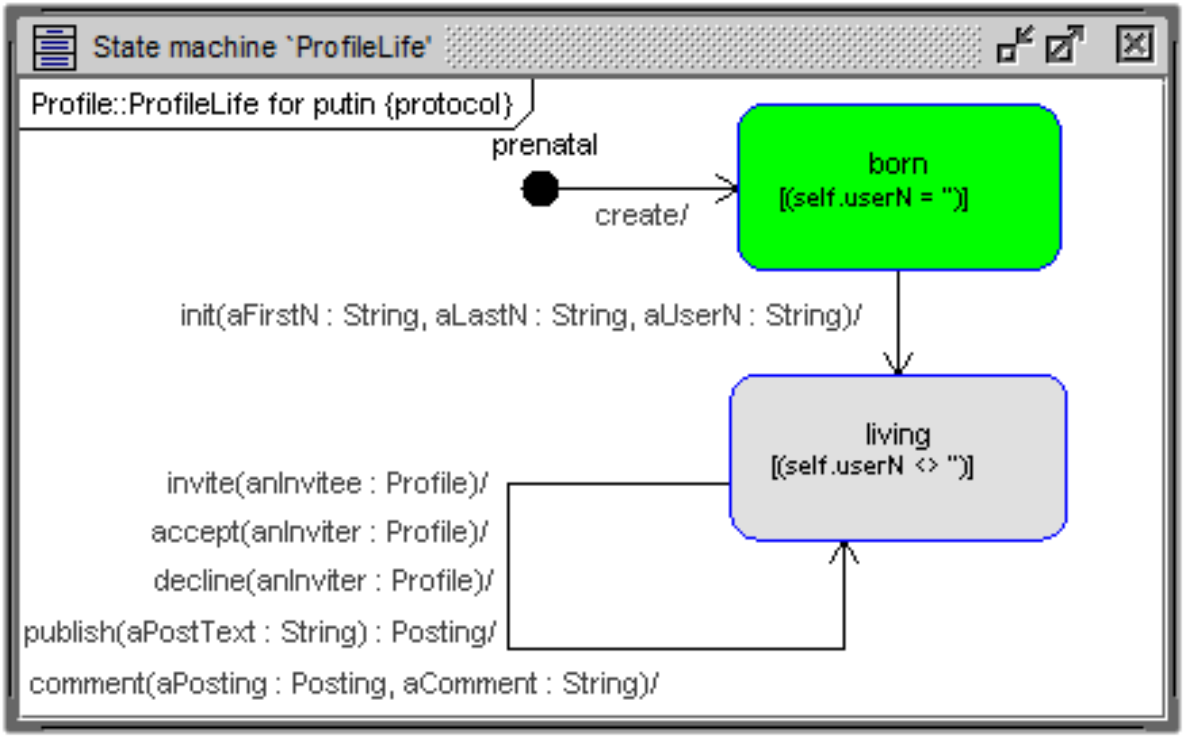
Animation and documentation of a simple scenario

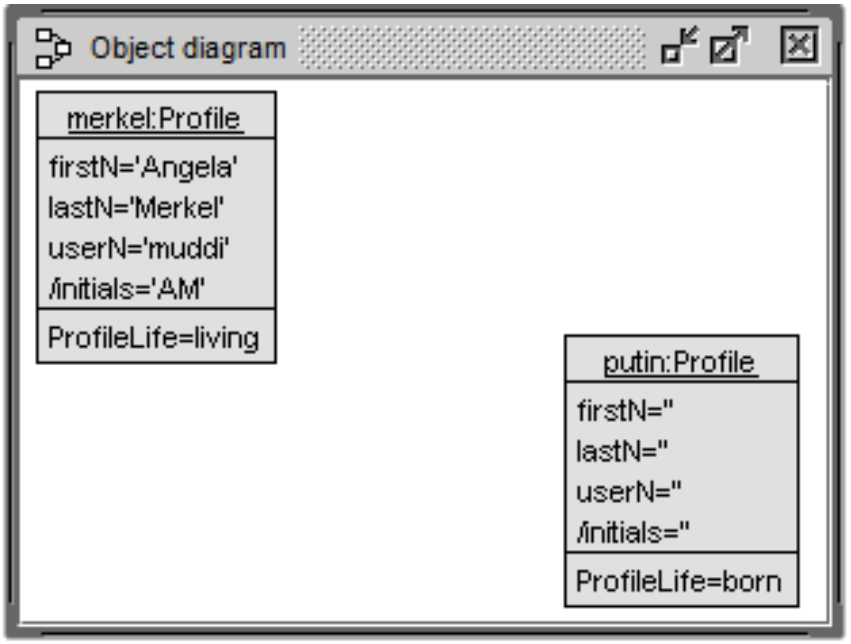
- Scenario: creation of two profiles and establishment of a friendship association link between them
- UML diagrams used: object, statechart, sequence and communication diagram together with USE command list
- Statechart diagrams together with object diagrams and USE command list
- Sequence diagrams together with USE command list
- Communication diagrams together with USE command list
- OCL queries on the USE shell

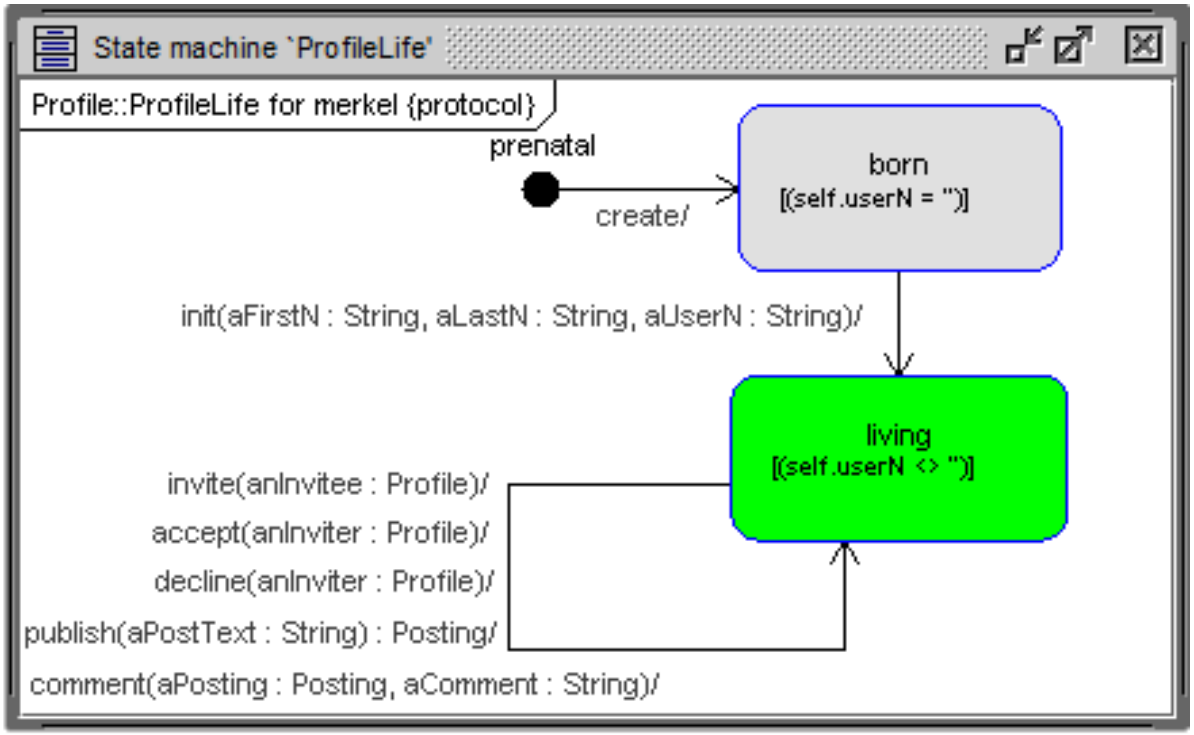

Object diagram
merkel:Profile
firstN="
lastN="
userN="
/initials="
ProfileLife=born
putin:Profile
firstN="
lastN="
userN="
/initials="
ProfileLife=born

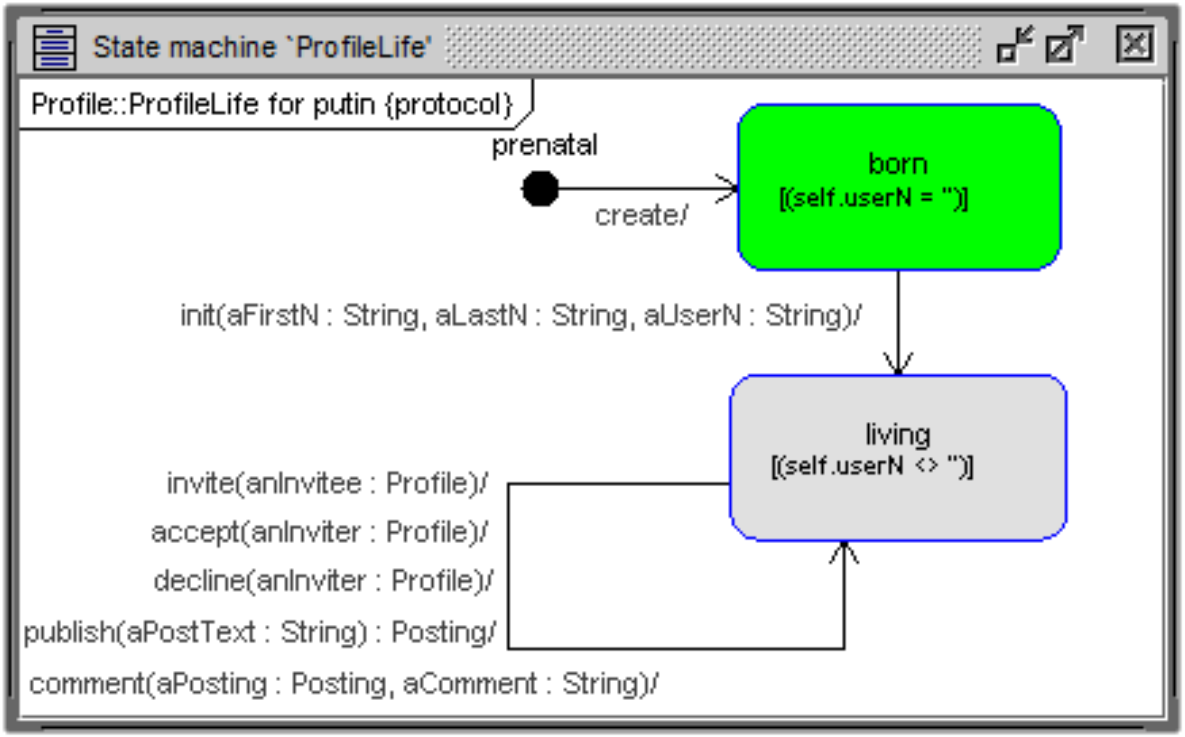

State machine `ProfileLife'
Profile::ProfileLife for merkel {protocol}
prenatal
create/
born
[(self.userN = '')]
init(aFirstN : String, aLastN : String, aUserN : String)/
living
[(self.userN <> '')]
invite(anInvitee : Profile)/
accept(anInviter : Profile)/
decline(anInviter : Profile)/
publish(aPostText : String) : Posting/
comment(aPosting : Posting, aComment : String)/

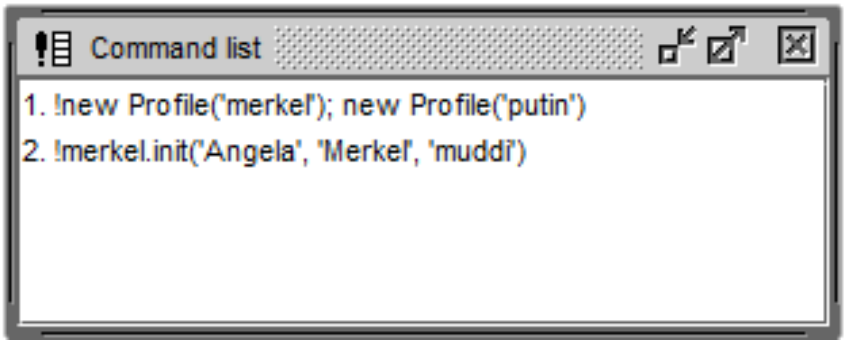

Command list
1. !new Profile('merkel'); new Profile('putin')

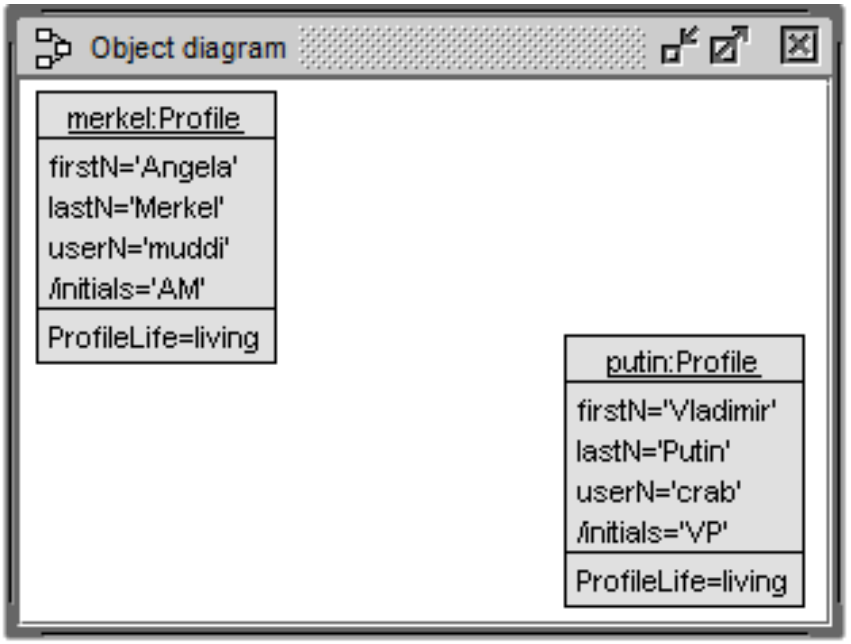

State machine `ProfileLife'
Profile::ProfileLife for putin {protocol}
prenatal
create/
born
[(self.userN = '')]
init(aFirstN : String, aLastN : String, aUserN : String)/
living
[(self.userN <> '')]
invite(anInvitee : Profile)/
accept(anInviter : Profile)/
decline(anInviter : Profile)/
publish(aPostText : String) : Posting/
comment(aPosting : Posting, aComment : String)/

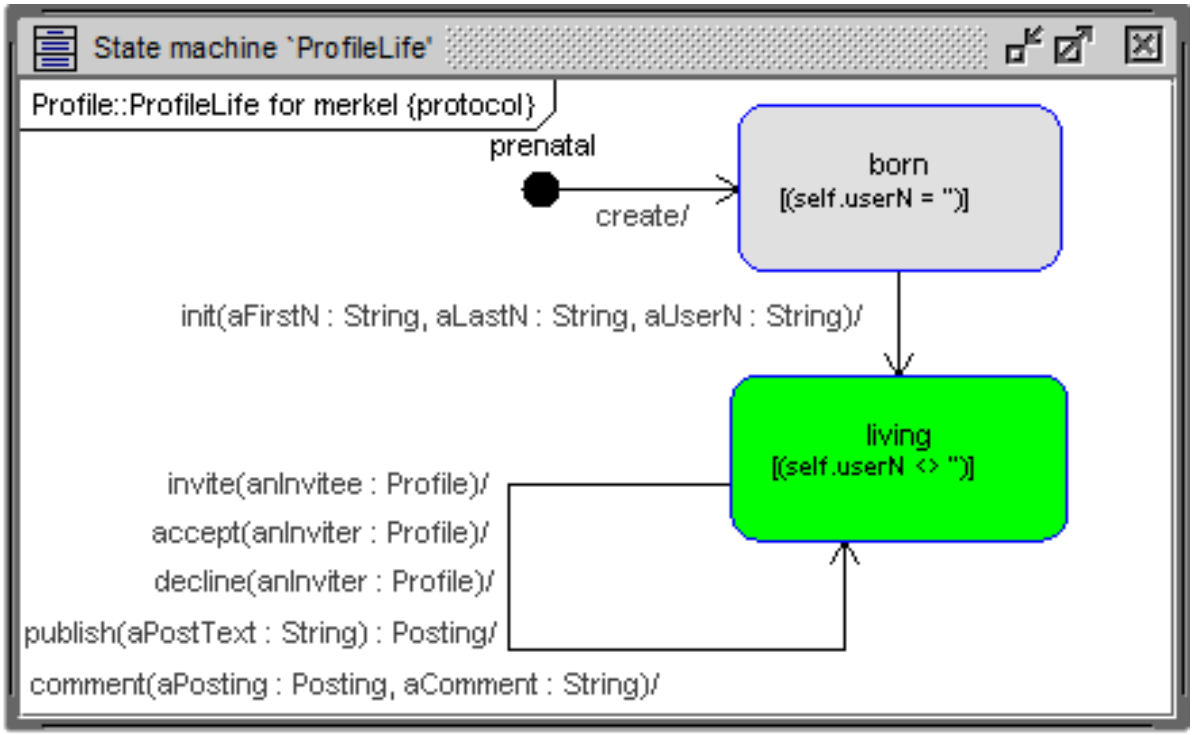

Object diagram
merkel:Profile
firstN='Angela'
lastN='Merkel'
userN='muddi'
/initials='AM'
ProfileLife=living
putin:Profile
firstN=''
lastN=''
userN=''
/initials=''
ProfileLife=born

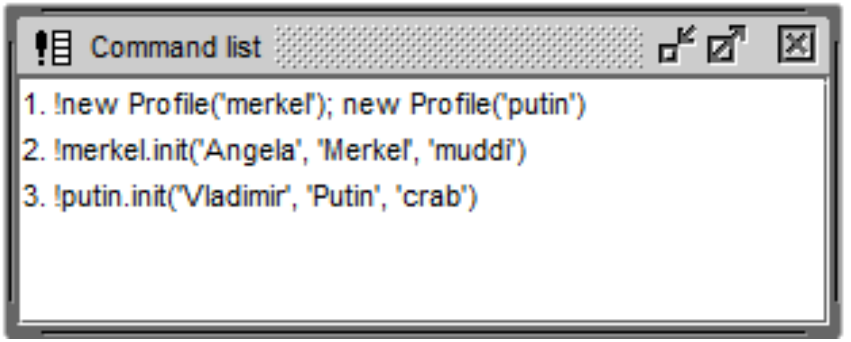

State machine `ProfileLife'
Profile::ProfileLife for merkel {protocol}
prenatal
create/
born
[(self.userN = '')]
init(aFirstN : String, aLastN : String, aUserN : String)/
living
[(self.userN <> '')]
invite(anInvitee : Profile)/
accept(anInviter : Profile)/
decline(anInviter : Profile)/
publish(aPostText : String) : Posting/
comment(aPosting : Posting, aComment : String)/

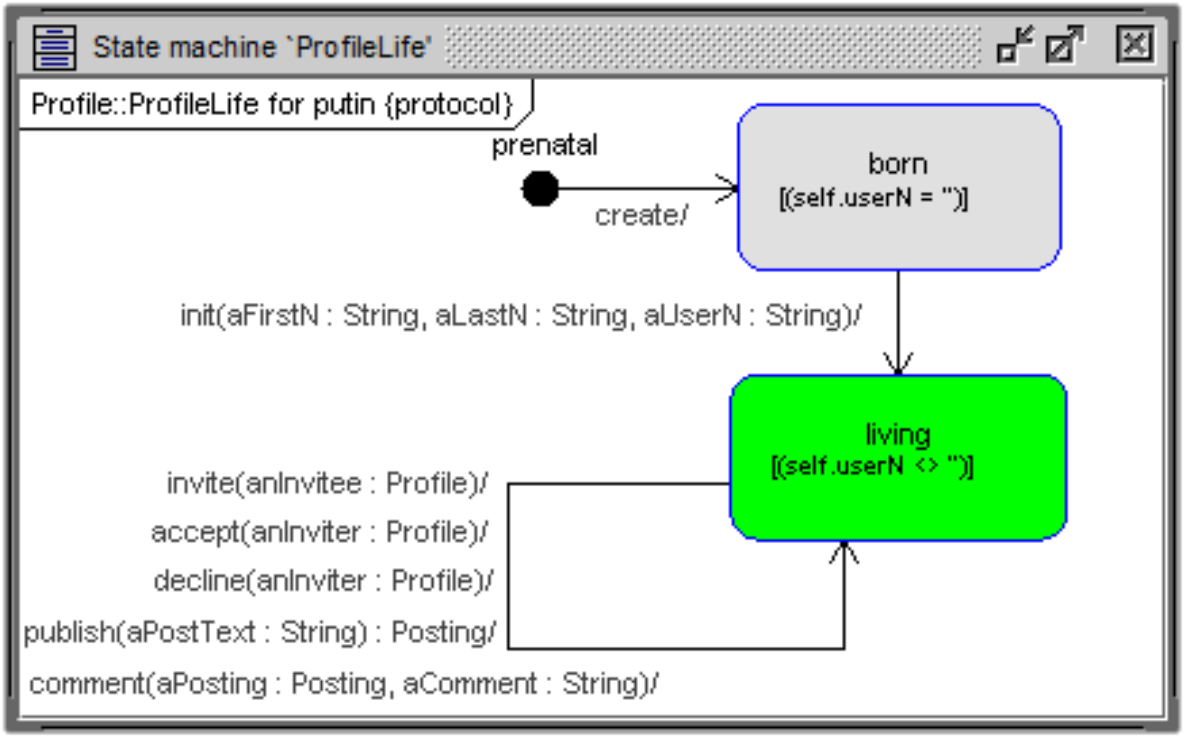

State machine `ProfileLife'
Profile::ProfileLife for putin {protocol}
prenatal
create/
born
[(self.userN = '')]
init(aFirstN : String, aLastN : String, aUserN : String)/
living
[(self.userN <> '')]
invite(anInvitee : Profile)/
accept(anInviter : Profile)/
decline(anInviter : Profile)/
publish(aPostText : String) : Posting/
comment(aPosting : Posting, aComment : String)/

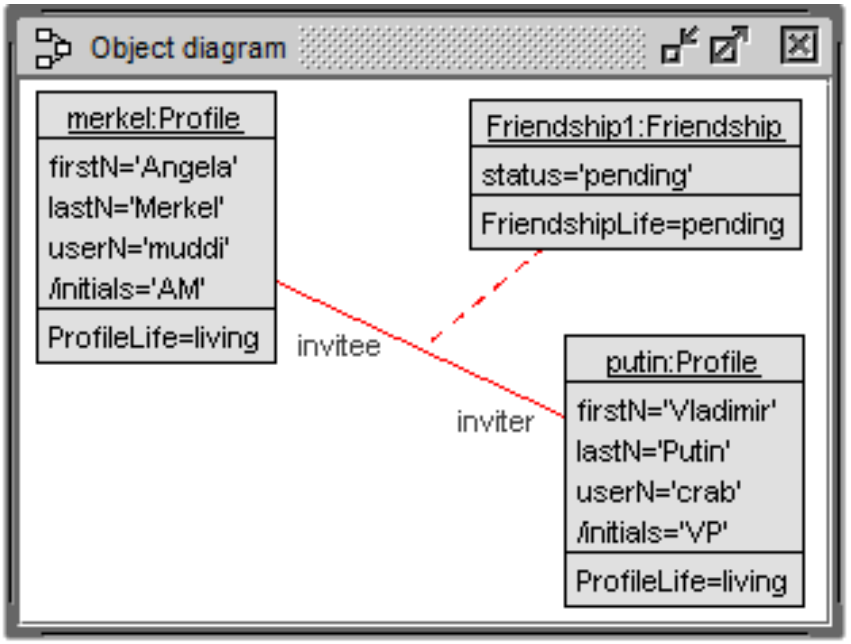

Command list
1. !new Profile('merkel'); new Profile('putin')
2. !merkel.init('Angela', 'Merkel', 'muddi')

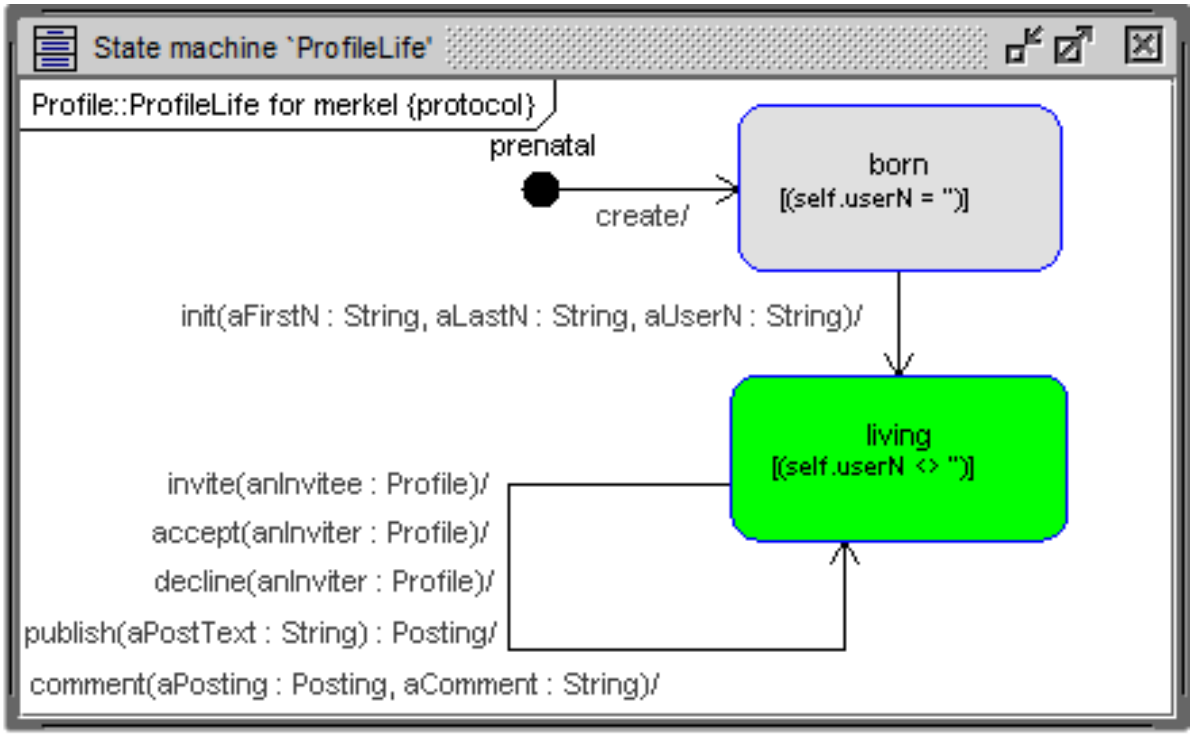

Object diagram
merkel:Profile
firstN='Angela'
lastN='Merkel'
userN='muddi'
/initials='AM'
ProfileLife=living
putin:Profile
firstN='Vladimir'
lastN='Putin'
userN='crab'
/initials='VP'
ProfileLife=living

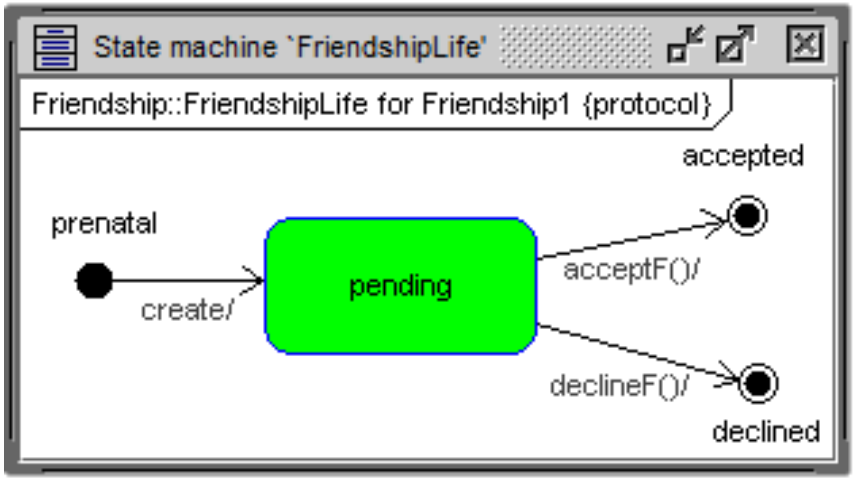

State machine `ProfileLife'
Profile::ProfileLife for merkel {protocol}
prenatal
create/
born
[(self.userN = '')]
init(aFirstN : String, aLastN : String, aUserN : String)/
living
[(self.userN <> '')]
invite(anInvitee : Profile)/
accept(anInviter : Profile)/
decline(anInviter : Profile)/
publish(aPostText : String) : Posting/
comment(aPosting : Posting, aComment : String)/

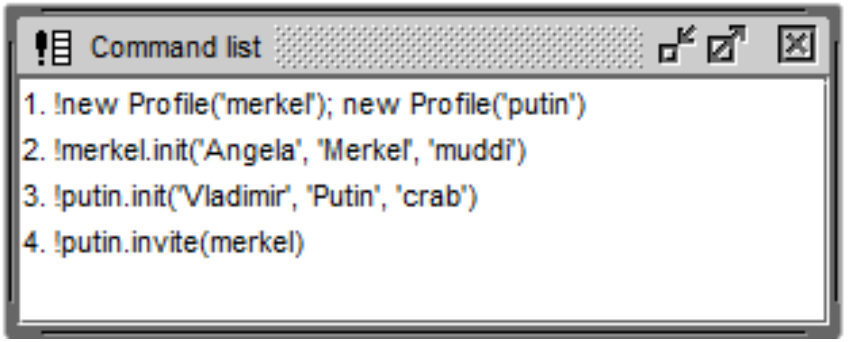

Command list
1. !new Profile('merkel'); new Profile('putin')
2. !merkel.init('Angela', 'Merkel', 'muddi')
3. !putin.init('Vladimir', 'Putin', 'crab')

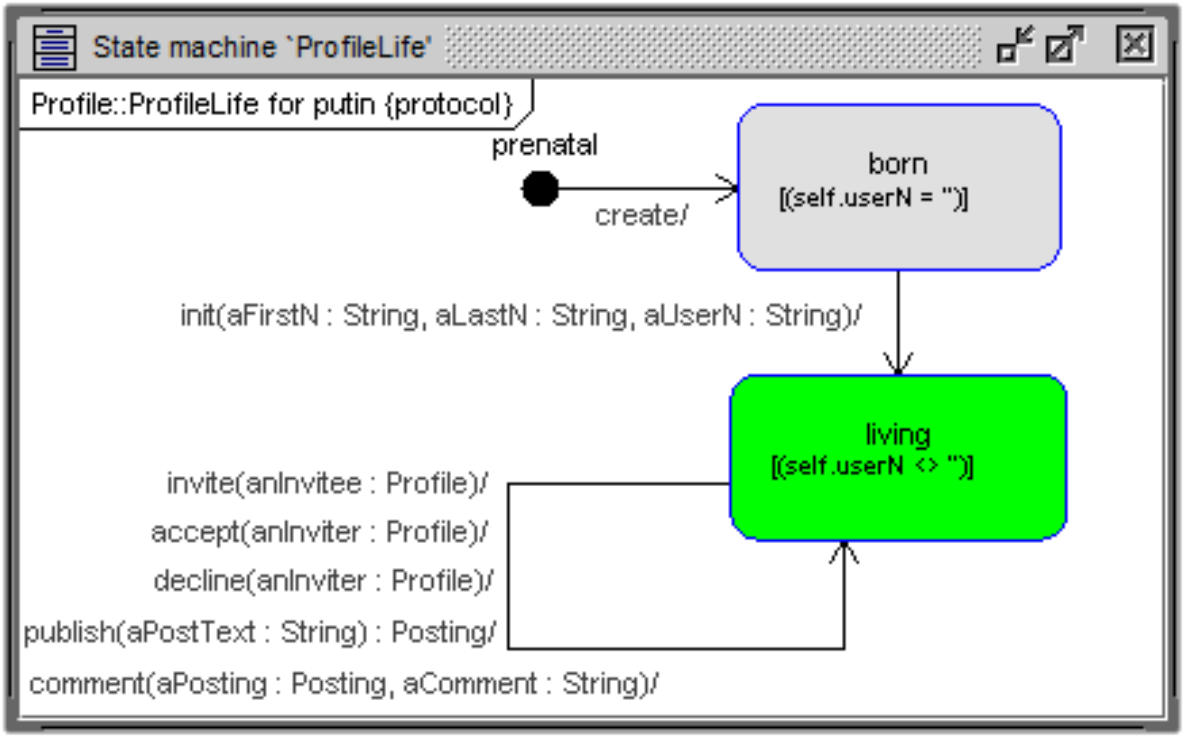

State machine `ProfileLife'
Profile::ProfileLife for putin {protocol}
prenatal
create/
born
[(self.userN = '')]
init(aFirstN : String, aLastN : String, aUserN : String)/
living
[(self.userN <> '')]
invite(anInvitee : Profile)/
accept(anInviter : Profile)/
decline(anInviter : Profile)/
publish(aPostText : String) : Posting/
comment(aPosting : Posting, aComment : String)/

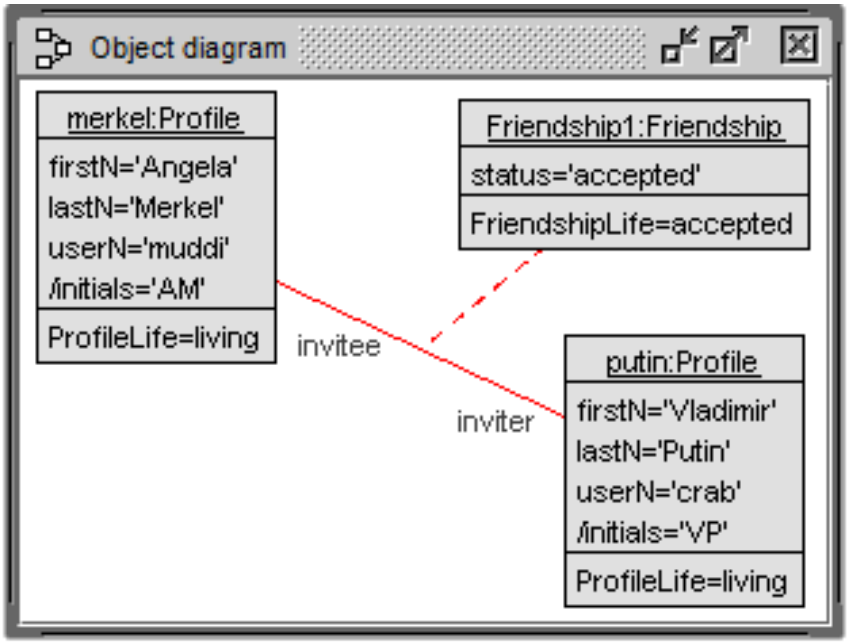

Object diagram
merkel:Profile
firstN='Angela'
lastN='Merkel'
userN='muddi'
/initials='AM'
ProfileLife=living
Friendship1:Friendship
status='pending'
FriendshipLife=pending
invitee
inviter
putin:Profile
firstN='Vladimir'
lastN='Putin'
userN='crab'
/initials='VP'
ProfileLife=living

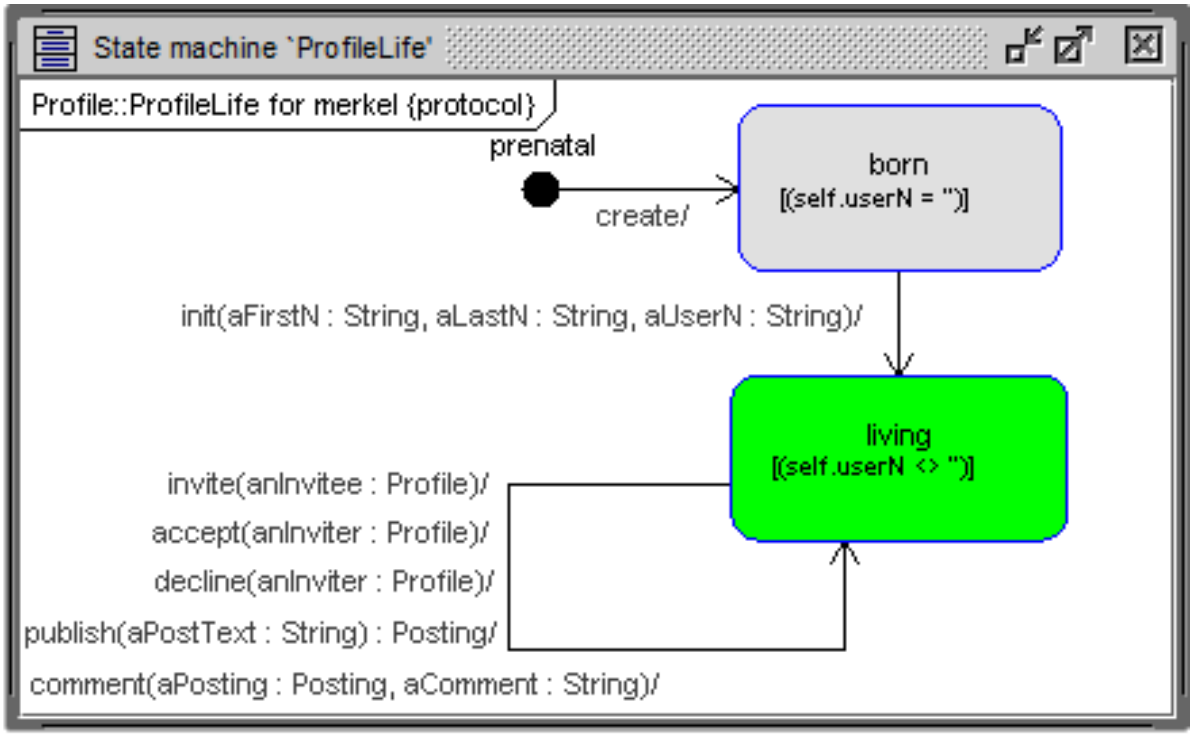

State machine `ProfileLife'
Profile::ProfileLife for merkel {protocol}
prenatal
create/
born
[(self.userN = '')]
init(aFirstN : String, aLastN : String, aUserN : String)/
living
[(self.userN <> '')]
invite(anInvitee : Profile)/
accept(anInviter : Profile)/
decline(anInviter : Profile)/
publish(aPostText : String) : Posting/
comment(aPosting : Posting, aComment : String)/

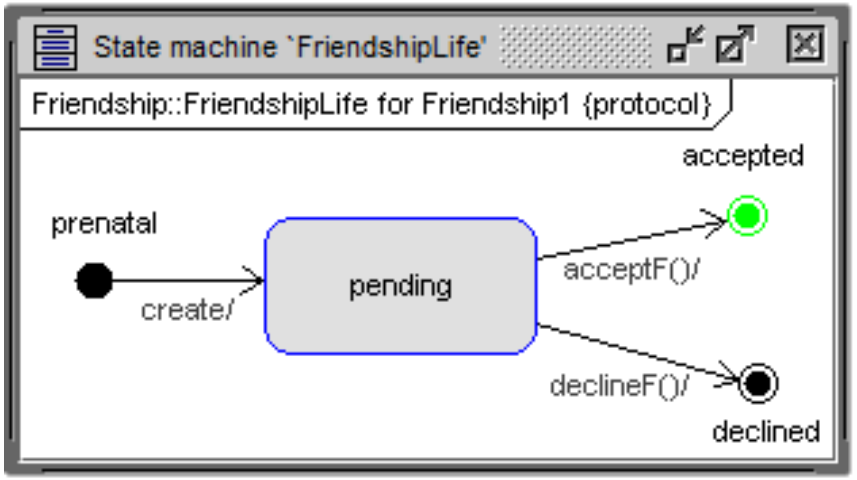

State machine `FriendshipLife'
Friendship::FriendshipLife for Friendship1 {protocol}
prenatal
create/
pending
acceptF()/
declineF()/
accepted
declined

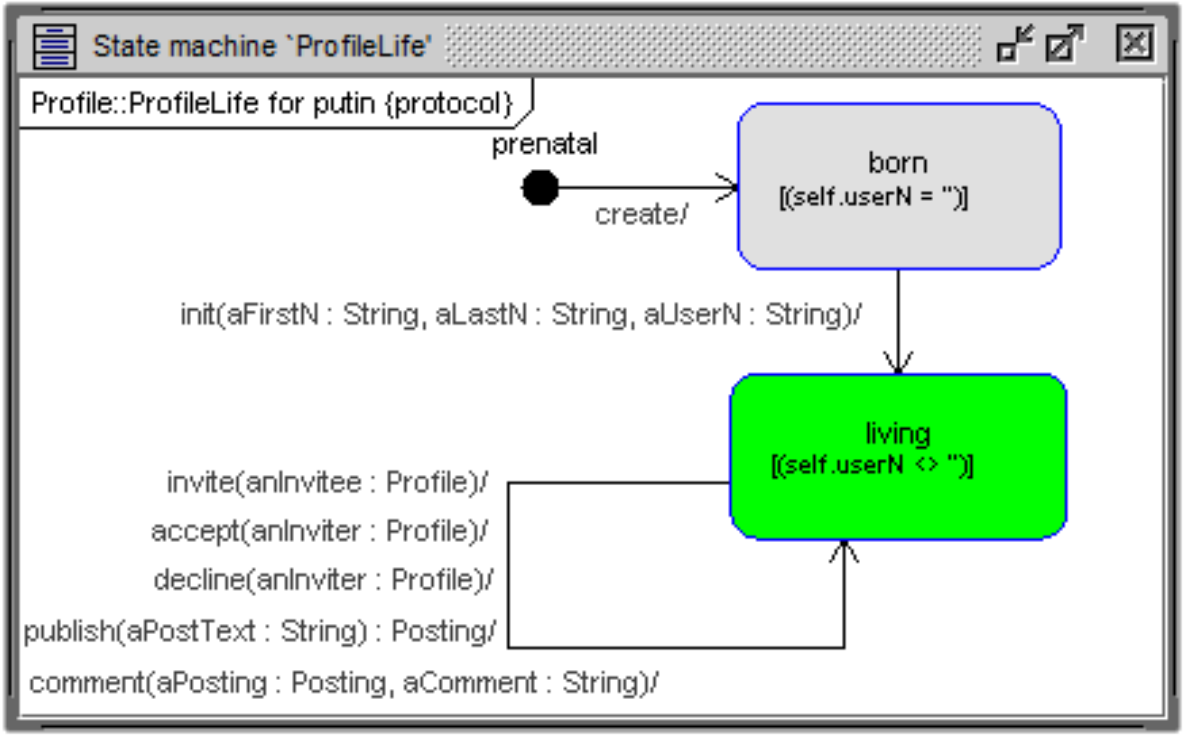

Command list
1. !new Profile('merkel'); new Profile('putin')
2. !merkel.init('Angela', 'Merkel', 'muddi')
3. !putin.init('Vladimir', 'Putin', 'crab')
4. !putin.invite(merkel)

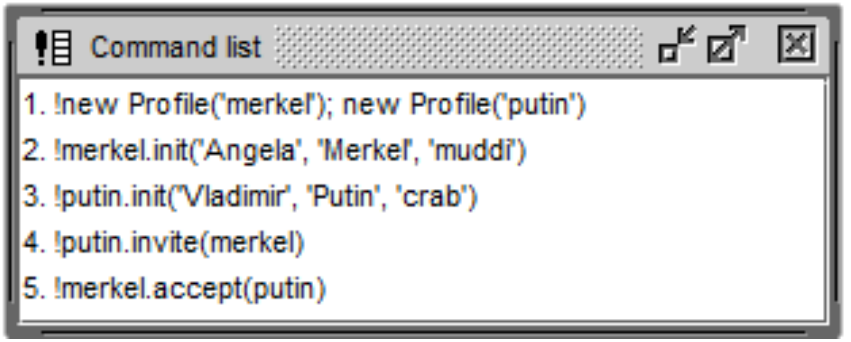

State machine `ProfileLife'
Profile::ProfileLife for putin {protocol}
prenatal
create/
born
[(self.userN = '')]
init(aFirstN : String, aLastN : String, aUserN : String)/
living
[(self.userN <> '')]
invite(anInvitee : Profile)/
accept(anInviter : Profile)/
decline(anInviter : Profile)/
publish(aPostText : String) : Posting/
comment(aPosting : Posting, aComment : String)/


Object diagram
merkel:Profile
firstN='Angela'
lastN='Merkel'
userN='muddi'
/initials='AM'
ProfileLife=living
Friendship1:Friendship
status='accepted'
FriendshipLife=accepted
invitee
inviter
putin:Profile
firstN='Vladimir'
lastN='Putin'
userN='crab'
/initials='VP'
ProfileLife=living


State machine `ProfileLife'
Profile::ProfileLife for merkel {protocol}
prenatal
create/
born
[(self.userN = '')]
init(aFirstN : String, aLastN : String, aUserN : String)/
living
[(self.userN <> '')]
invite(anInvitee : Profile)/
accept(anInviter : Profile)/
decline(anInviter : Profile)/
publish(aPostText : String) : Posting/
comment(aPosting : Posting, aComment : String)/

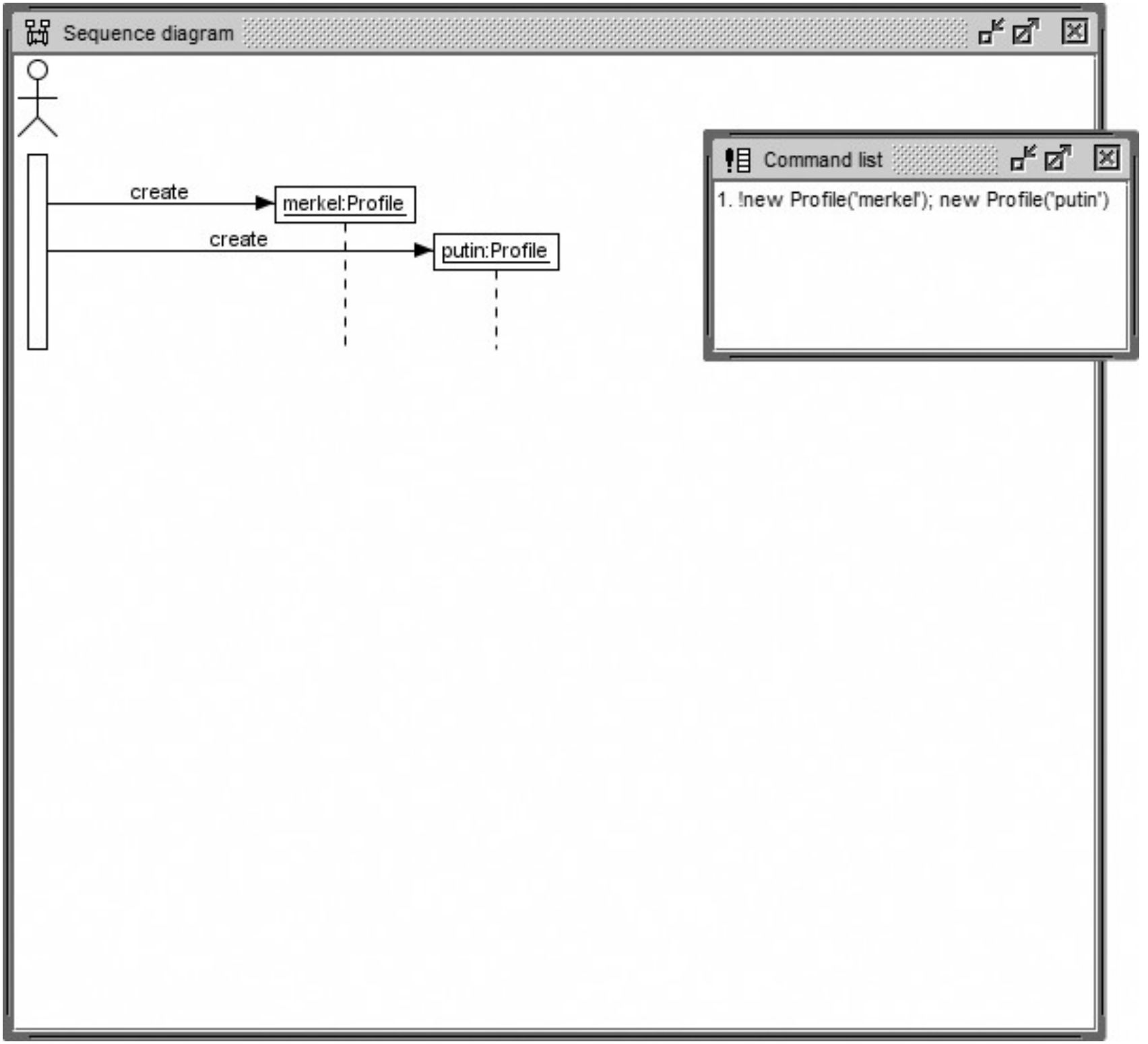

State machine `FriendshipLife'
Friendship::FriendshipLife for Friendship1 {protocol}
accepted
prenatal
pending
acceptF()/
create/
declineF()/
declined

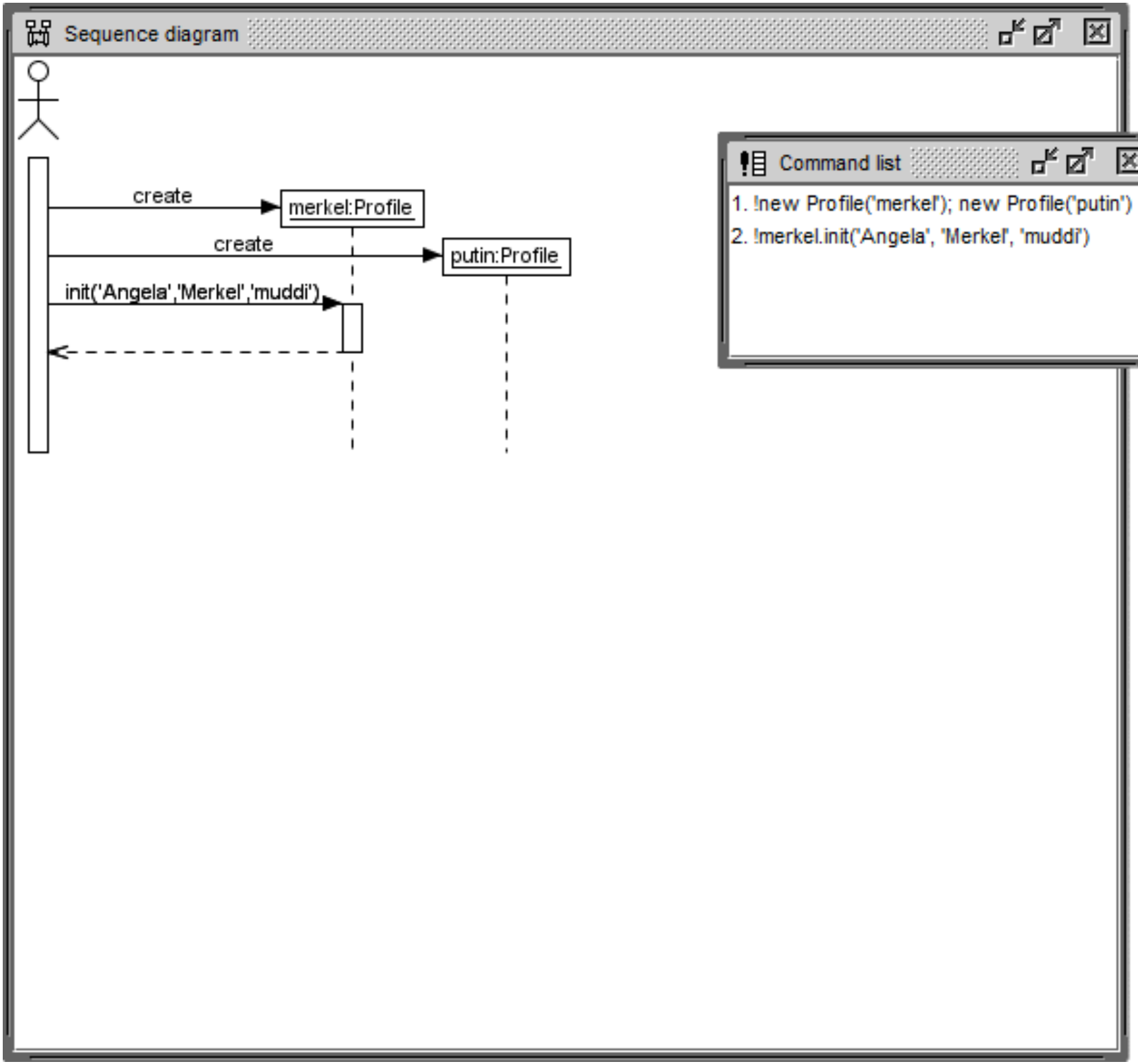

State machine `ProfileLife'
Profile::ProfileLife for putin {protocol}
prenatal
create/
born
[(self.userN = '')]
init(aFirstN : String, aLastN : String, aUserN : String)/
living
[(self.userN <> '')]
invite(anInvitee : Profile)/
accept(anInviter : Profile)/
decline(anInviter : Profile)/
publish(aPostText : String) : Posting/
comment(aPosting : Posting, aComment : String)/

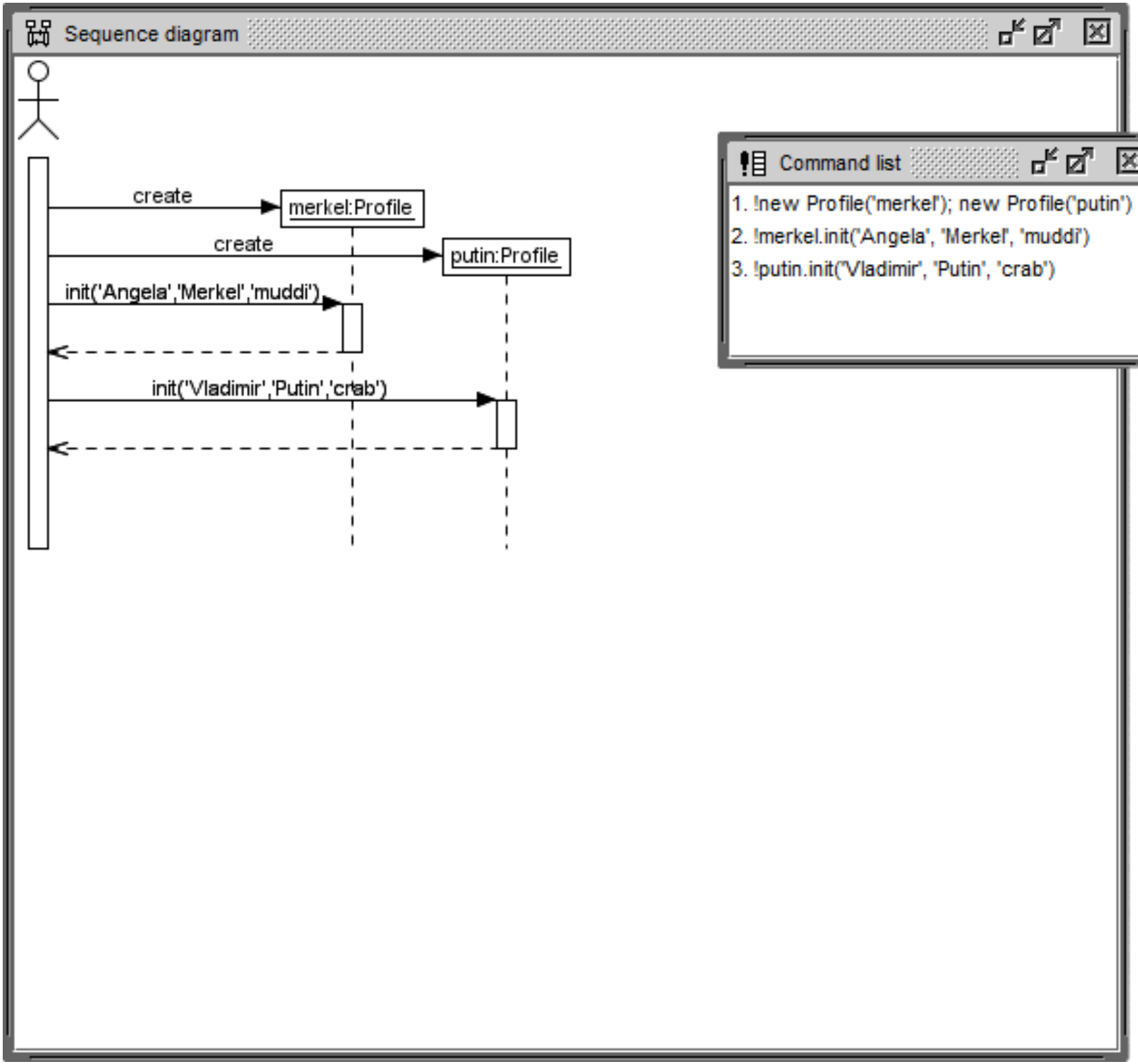

Command list
1. !new Profile('merkel'); new Profile('putin')
2. !merkel.init('Angela', 'Merkel', 'muddi')
3. !putin.init('Vladimir', 'Putin', 'crab')
4. !putin.invite(merkel)
5. !merkel.accept(putin)

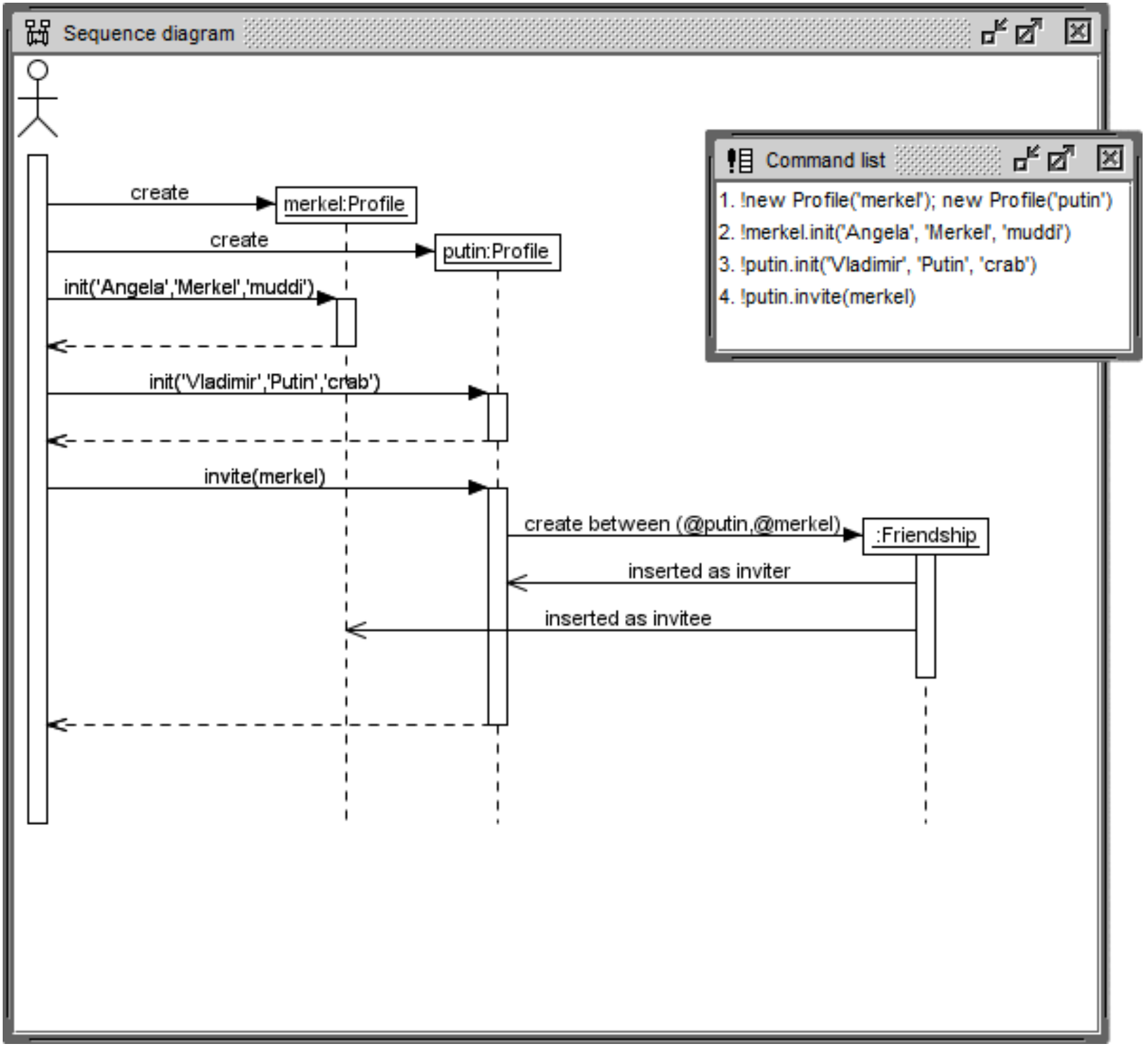
Animation and documentation of a simple scenario

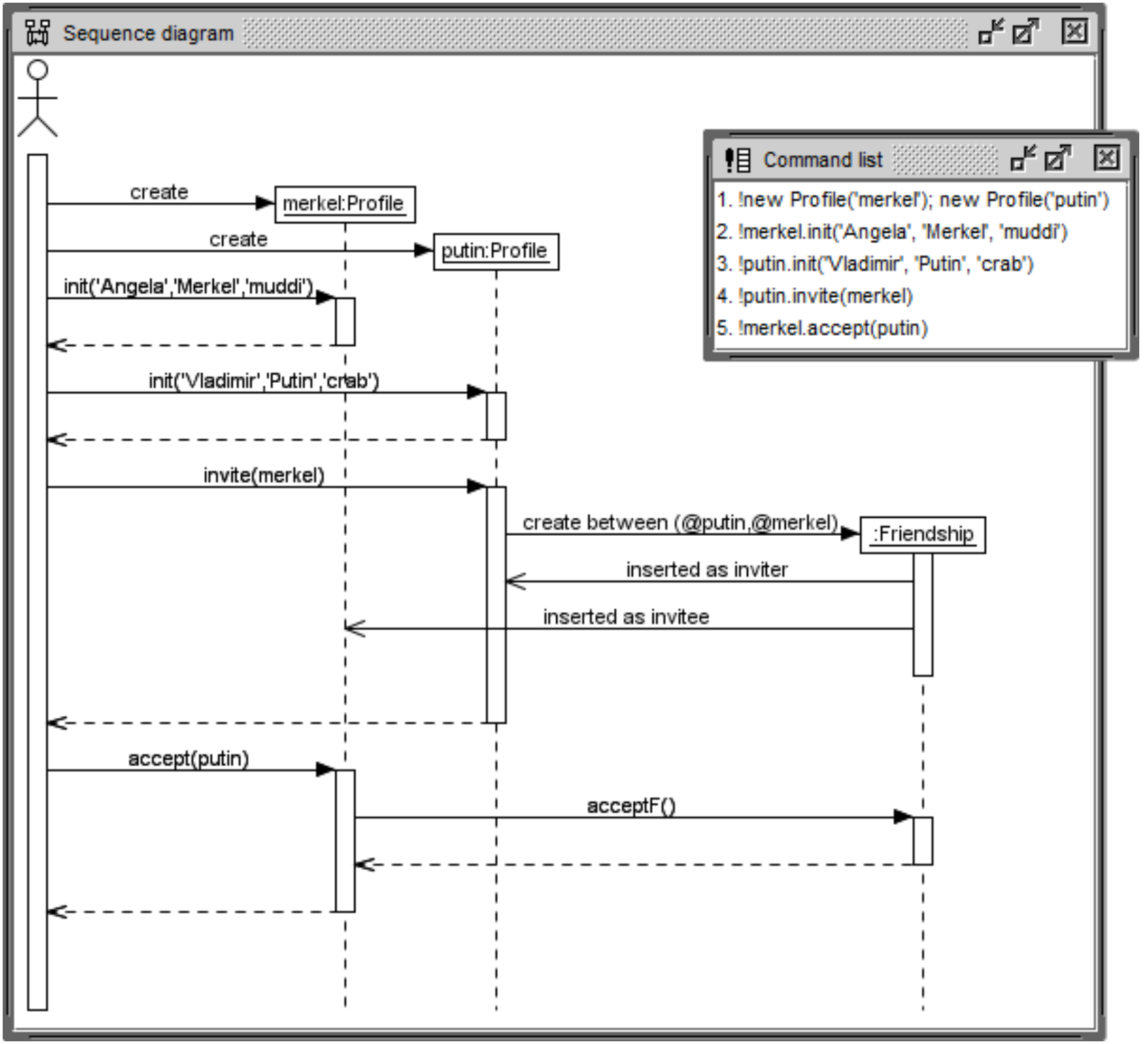
- Scenario: creation of two profiles and establishment of a friendship association link between them
- UML diagrams used: object, statechart, sequence and communication diagram together with USE command list
- Statechart diagrams together with object diagrams and USE command list
- Sequence diagrams together with USE command list
- Communication diagrams together with USE command list
- OCL queries on the USE shell


Sequence diagram
create
merkel:Profile
create
putin:Profile
Command list
1. !new Profile('merkel'); new Profile('putin')


Sequence diagram
create
merkel:Profile
create
putin:Profile
init('Angela','Merkel','muddi')
Command list
1. !new Profile('merkel'); new Profile('putin')
2. !merkel.init('Angela', 'Merkel', 'muddi')

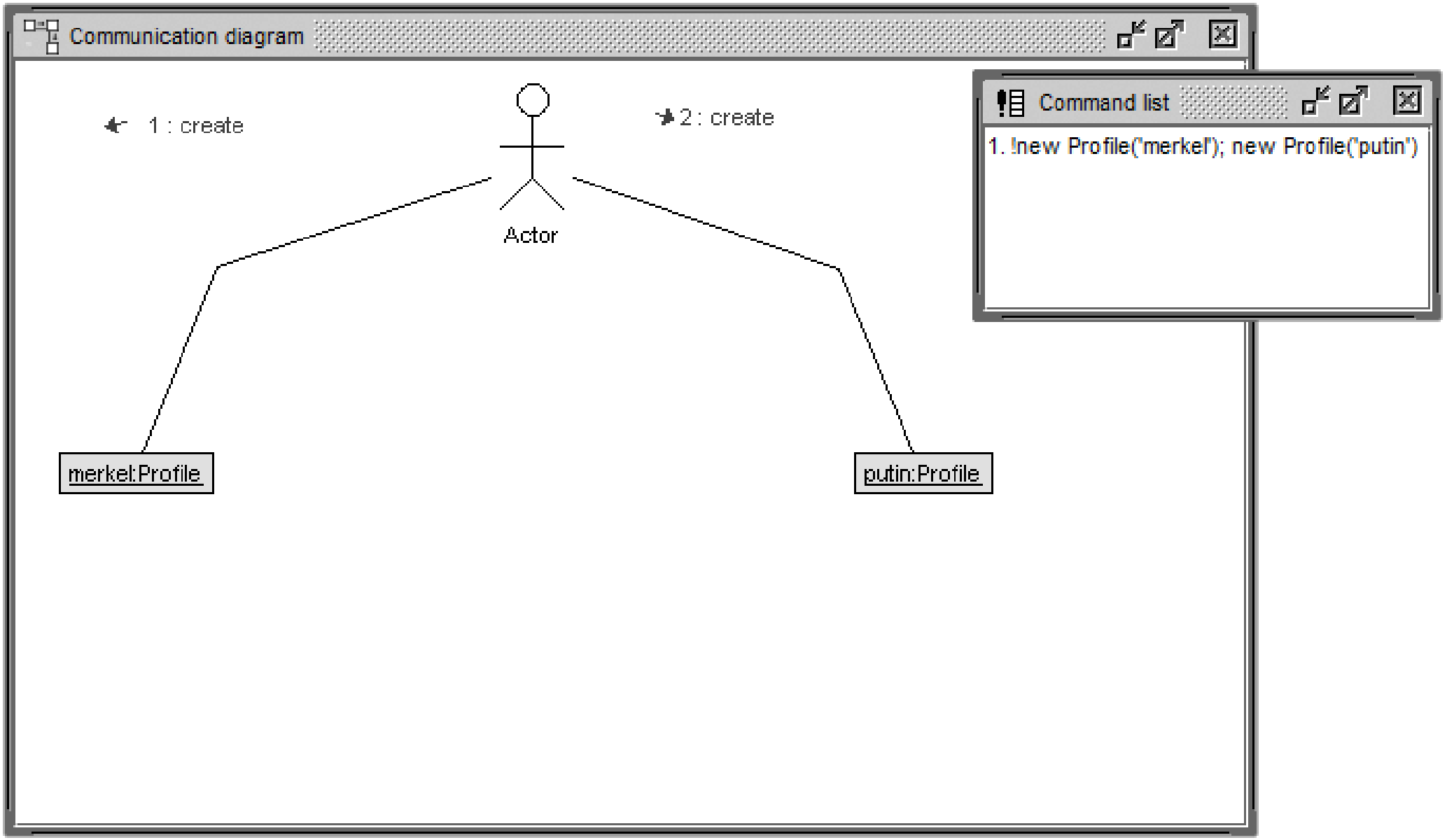

Sequence diagram
create
merkel:Profile
create
putin:Profile
init('Angela','Merkel','muddi')
init('Vladimir','Putin','crab')
Command list
1. !new Profile('merkel'); new Profile('putin')
2. !merkel.init('Angela', 'Merkel', 'muddi')
3. !putin.init('Vladimir', 'Putin', 'crab')

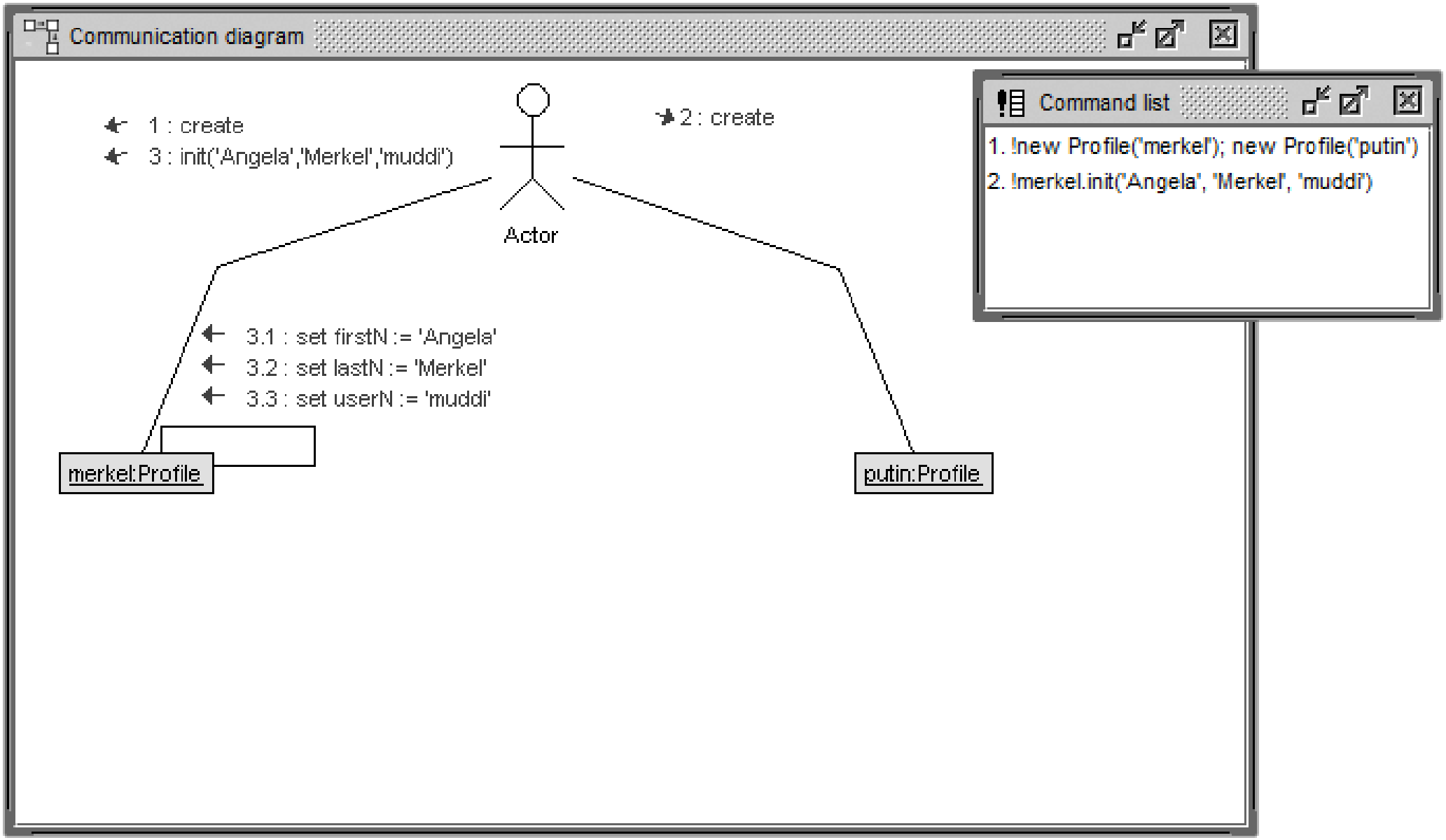

Sequence diagram
create
merkel:Profile
create
putin:Profile
init('Angela','Merkel','muddi')
init('Vladimir','Putin','crab')
invite(merkel)
create between (@putin,@merkel)
:Friendship
inserted as inviter
inserted as invitee
Command list
1. !new Profile('merkel'); new Profile('putin')
2. !merkel.init('Angela', 'Merkel', 'muddi')
3. !putin.init('Vladimir', 'Putin', 'crab')
4. !putin.invite(merkel)

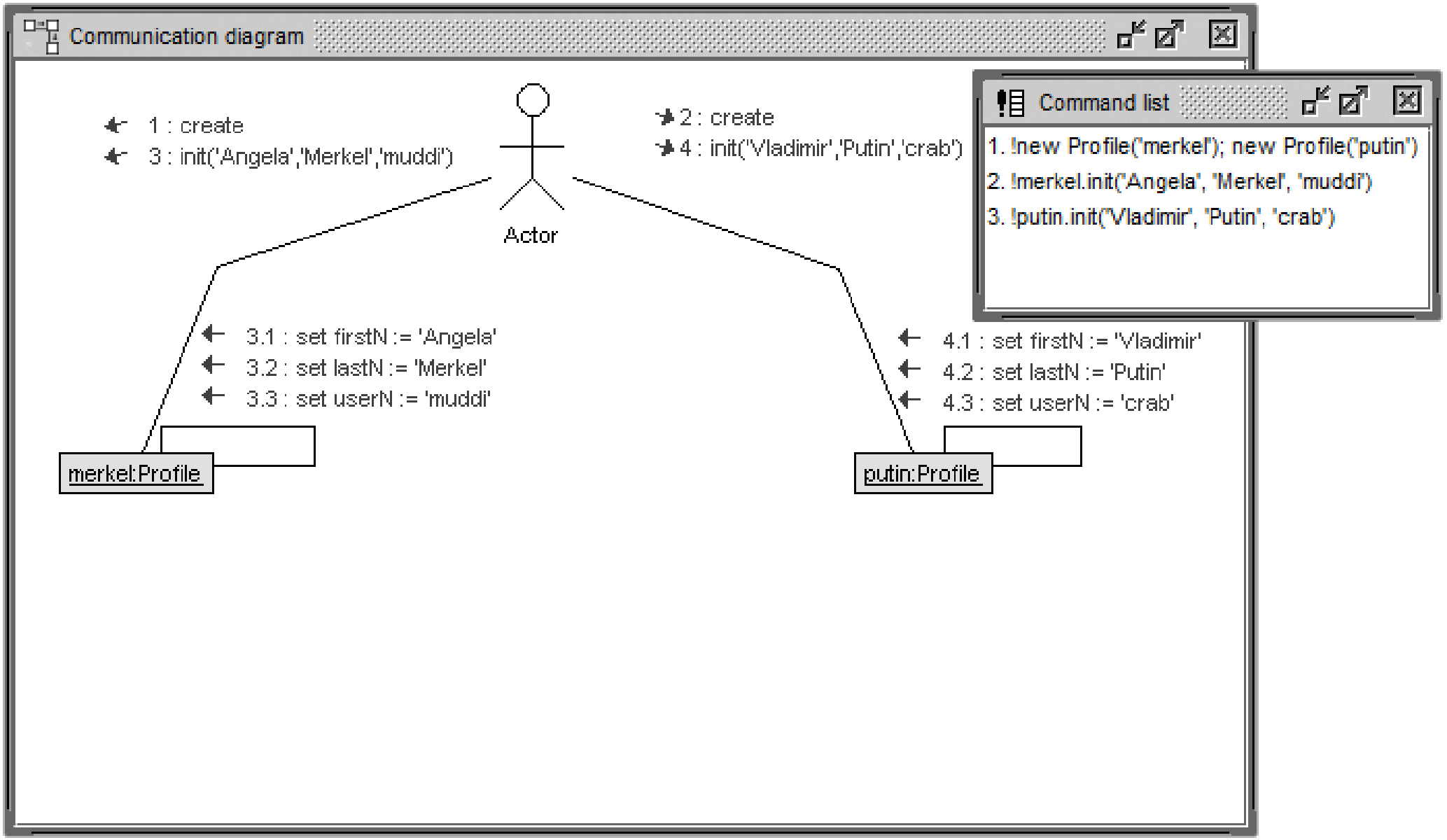

Sequence diagram
create
merkel:Profile
create
putin:Profile
init('Angela','Merkel','muddi')
init('Vladimir','Putin','crab')
invite(merkel)
create between (@putin,@merkel)
:Friendship
inserted as inviter
inserted as invitee
accept(putin)
acceptF()
Command list
1. !new Profile('merkel'); new Profile('putin')
2. !merkel.init('Angela', 'Merkel', 'muddi')
3. !putin.init('Vladimir', 'Putin', 'crab')
4. !putin.invite(merkel)
5. !merkel.accept(putin)

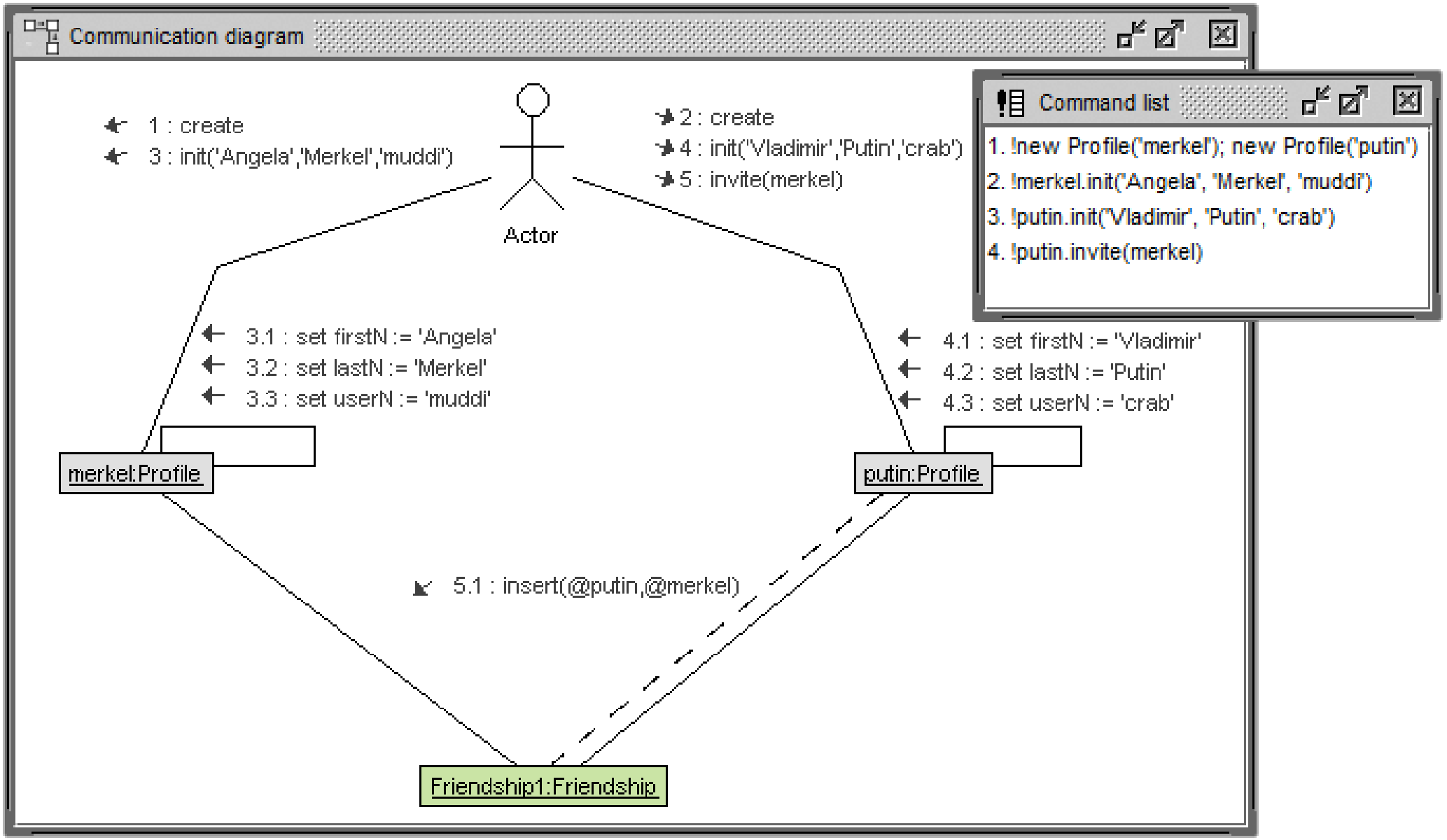
Animation and documentation of a simple scenario

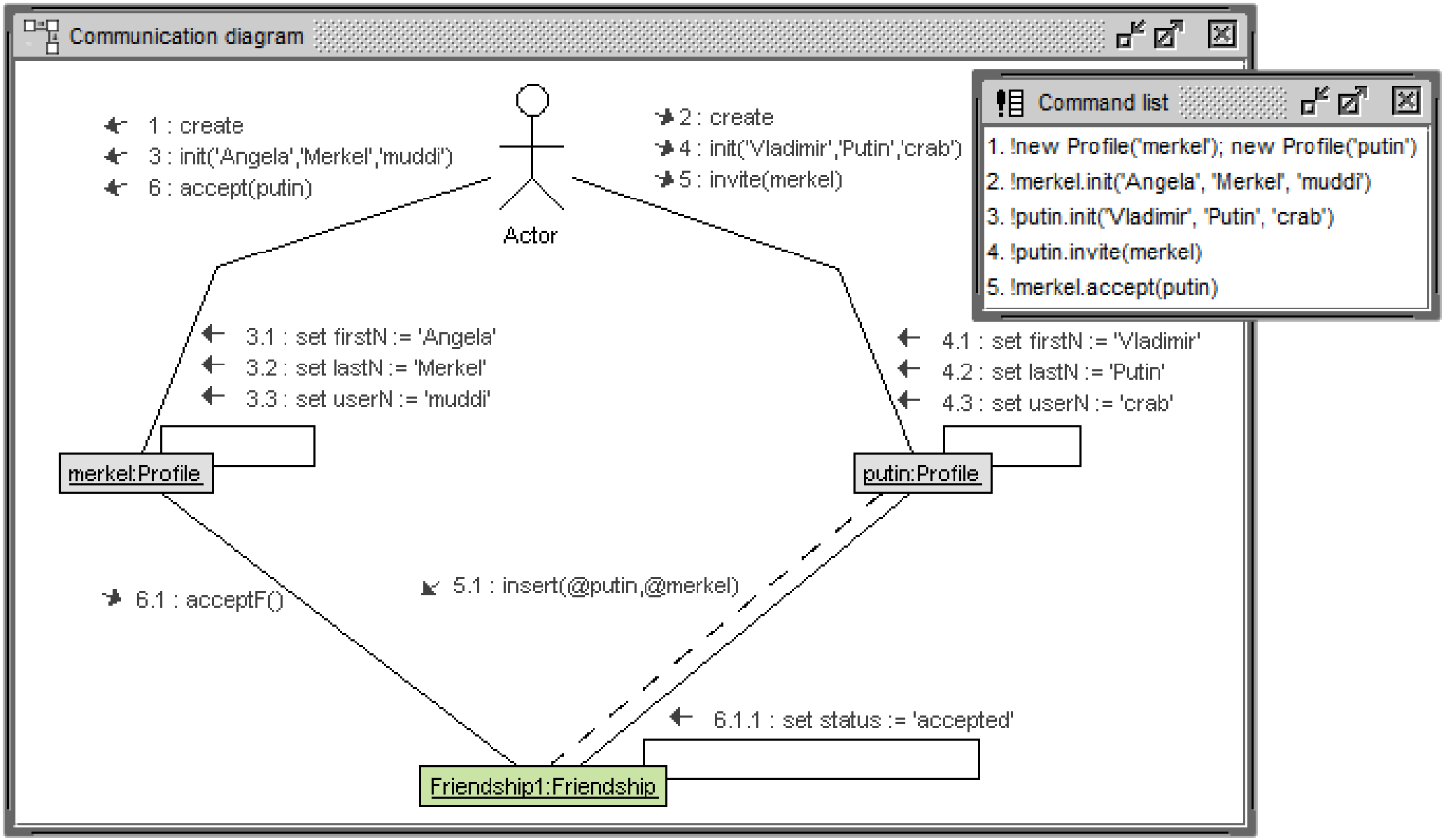
- Scenario: creation of two profiles and establishment of a friendship association link between them
- UML diagrams used: object, statechart, sequence and communication diagram together with USE command list
- Statechart diagrams together with object diagrams and USE command list
- Sequence diagrams together with USE command list
- Communication diagrams together with USE command list
- OCL queries on the USE shell


Communication diagram
1 : create
2 : create
Actor
merkel:Profile
putin:Profile
Command list
1. !new Profile('merkel'); new Profile('putin')


Communication diagram
1 : create
3 : init('Angela','Merkel','muddi')
2 : create
Actor
3.1 : set firstN := 'Angela'
3.2 : set lastN := 'Merkel'
3.3 : set userN := 'muddi'
merkel:Profile
putin:Profile
Command list
1. !new Profile('merkel'); new Profile('putin')
2. !merkel.init('Angela', 'Merkel', 'muddi')


Communication diagram
1 : create
3 : init('Angela','Merkel','muddi')
2 : create
4 : init('Vladimir','Putin','crab')
Actor
3.1 : set firstN := 'Angela'
3.2 : set lastN := 'Merkel'
3.3 : set userN := 'muddi'
4.1 : set firstN := 'Vladimir'
4.2 : set lastN := 'Putin'
4.3 : set userN := 'crab'
merkel:Profile
putin:Profile
Command list
1. !new Profile('merkel'); new Profile('putin')
2. !merkel.init('Angela', 'Merkel', 'muddi')
3. !putin.init('Vladimir', 'Putin', 'crab')


Communication diagram
1 : create
3 : init('Angela','Merkel','muddi')
2 : create
4 : init('Vladimir','Putin','crab')
5 : invite(merkel)
Actor
Command list
1. !new Profile('merkel'); new Profile('putin')
2. !merkel.init('Angela', 'Merkel', 'muddi')
3. !putin.init('Vladimir', 'Putin', 'crab')
4. !putin.invite(merkel)
3.1 : set firstN := 'Angela'
3.2 : set lastN := 'Merkel'
3.3 : set userN := 'muddi'
4.1 : set firstN := 'Vladimir'
4.2 : set lastN := 'Putin'
4.3 : set userN := 'crab'
merkel:Profile
putin:Profile
5.1 : insert(@putin,@merkel)
Friendship1:Friendship


Communication diagram
1 : create
3 : init('Angela','Merkel','muddi')
6 : accept(putin)
2 : create
4 : init('Vladimir','Putin','crab')
5 : invite(merkel)
Actor
Command list
1. !new Profile('merkel'); new Profile('putin')
2. !merkel.init('Angela', 'Merkel', 'muddi')
3. !putin.init('Vladimir', 'Putin', 'crab')
4. !putin.invite(merkel)
5. !merkel.accept(putin)
3.1 : set firstN := 'Angela'
3.2 : set lastN := 'Merkel'
3.3 : set userN := 'muddi'
4.1 : set firstN := 'Vladimir'
4.2 : set lastN := 'Putin'
4.3 : set userN := 'crab'
merkel:Profile
putin:Profile
6.1 : acceptF()
5.1 : insert(@putin,@merkel)
6.1.1 : set status := 'accepted'
Friendship1:Friendship

Animation and documentation of a simple scenario

- Scenario: creation of two profiles and establishment of a friendship association link between them
- UML diagrams used: object, statechart, sequence and communication diagram together with USE command list
- Statechart diagrams together with object diagrams and USE command list
- Sequence diagrams together with USE command list
- Communication diagrams together with USE command list
- OCL queries on the USE shell

Scenario documentation with OCL queries

```
! create merkel,putin:Profile
? Profile.allInstances->collect(p|Tuple{PF:p,UN:p.userN})
  Bag{Tuple{PF=merkel,UN=''},Tuple{PF=putin,UN=''}}
? Friendship.allInstances->collect(f|Tuple{FS:f,ST:f.status})
  Bag{}
```

Scenario documentation with OCL queries

```
! create merkel,putin:Profile
? Profile.allInstances->collect(p|Tuple{PF:p,UN:p.userN})
  Bag{Tuple{PF=merkel,UN=''},Tuple{PF=putin,UN=''}}
? Friendship.allInstances->collect(f|Tuple{FS:f,ST:f.status})
  Bag{}

! merkel.init('Angela','Merkel','muddi')
  Bag{Tuple{PF=merkel,UN='muddi'},Tuple{PF=putin,UN=''}}
  Bag{}
```

Scenario documentation with OCL queries

```
! create merkel,putin:Profile
? Profile.allInstances->collect(p|Tuple{PF:p,UN:p.userN})
  Bag{Tuple{PF=merkel,UN=''},Tuple{PF=putin,UN=''}}
? Friendship.allInstances->collect(f|Tuple{FS:f,ST:f.status})
  Bag{}

! merkel.init('Angela','Merkel','muddi')
  Bag{Tuple{PF=merkel,UN='muddi'},Tuple{PF=putin,UN=''}}
  Bag{}

! putin.init('Vladimir','Putin','crab')
  Bag{Tuple{PF=merkel,UN='muddi'},Tuple{PF=putin,UN='crab'}}
  Bag{}
```

Scenario documentation with OCL queries

```
! create merkel,putin:Profile
? Profile.allInstances->collect(p|Tuple{PF:p,UN:p.userN})
  Bag{Tuple{PF=merkel,UN=''},Tuple{PF=putin,UN=''}}
? Friendship.allInstances->collect(f|Tuple{FS:f,ST:f.status})
  Bag{}

! merkel.init('Angela','Merkel','muddi')
  Bag{Tuple{PF=merkel,UN='muddi'},Tuple{PF=putin,UN=''}}
  Bag{}

! putin.init('Vladimir','Putin','crab')
  Bag{Tuple{PF=merkel,UN='muddi'},Tuple{PF=putin,UN='crab'}}
  Bag{}

! putin.invite(merkel)
  Bag{Tuple{PF=merkel,UN='muddi'},Tuple{PF=putin,UN='crab'}}
  Bag{Tuple{FS=Friendship1,ST='pending'}}
```

Scenario documentation with OCL queries

```
! create merkel,putin:Profile
? Profile.allInstances->collect(p|Tuple{PF:p,UN:p.userN})
  Bag{Tuple{PF=merkel,UN=''},Tuple{PF=putin,UN=''}}
? Friendship.allInstances->collect(f|Tuple{FS:f,ST:f.status})
  Bag{}

! merkel.init('Angela','Merkel','muddi')
  Bag{Tuple{PF=merkel,UN='muddi'},Tuple{PF=putin,UN=''}}
  Bag{}

! putin.init('Vladimir','Putin','crab')
  Bag{Tuple{PF=merkel,UN='muddi'},Tuple{PF=putin,UN='crab'}}
  Bag{}

! putin.invite(merkel)
  Bag{Tuple{PF=merkel,UN='muddi'},Tuple{PF=putin,UN='crab'}}
  Bag{Tuple{FS=Friendship1,ST='pending'}}

! merkel.accept(putin)
  Bag{Tuple{PF=merkel,UN='muddi'},Tuple{PF=putin,UN='crab'}}
  Bag{Tuple{FS=Friendship1,ST='accepted'}}
```

Scenario documentation with OCL queries

```
! create merkel,putin:Profile
? Profile.allInstances->collect(p|Tuple{PF:p,UN:p.userN})
  Bag{Tuple{PF=merkel,UN=''},Tuple{PF=putin,UN=''}}
? Friendship.allInstances->collect(f|Tuple{FS:f,ST:f.status})
  Bag{}

! merkel.init('Angela','Merkel','muddi')
  Bag{Tuple{PF=merkel,UN='muddi'},Tuple{PF=putin,UN=''}}
  Bag{}

! putin.init('Vladimir','Putin','crab')
  Bag{Tuple{PF=merkel,UN='muddi'},Tuple{PF=putin,UN='crab'}}
  Bag{}

! putin.invite(merkel)
  Bag{Tuple{PF=merkel,UN='muddi'},Tuple{PF=putin,UN='crab'}}
  Bag{Tuple{FS=Friendship1,ST='pending'}}

! merkel.accept(putin)
  Bag{Tuple{PF=merkel,UN='muddi'},Tuple{PF=putin,UN='crab'}}
  Bag{Tuple{FS=Friendship1,ST='accepted'}}

? Friendship.allInstances->collect(f|
   Tuple{TER:f.inviter.userN,TEE:f.invitee.userN,ST:f.status})
  Bag{Tuple{TER='crab',TEE='muddi',ST='accepted'}}
```

Animation and documentation of a simple scenario

- Scenario: creation of two profiles and establishment of a friendship association link between them
- UML diagrams used: object, statechart, sequence and communication diagram together with USE command list
- Statechart diagrams together with object diagrams and USE command list
- Sequence diagrams together with USE command list
- Communication diagrams together with USE command list
- OCL queries on the USE shell

Variations for USE model definition and USE shell commands

- Textual USE model definition [repeated slides]
- Variation: implementation Profile::publish():Posting (A)
- Variation: postcondition Profile::comment() (B)
- Variation: invariant Profile::uniqueUserName (C)
- Variation: attribute Friendship::status (D)
- Variation: composition PosterPosting – multiplicity (E)
- Variation: association Interest - role names (F)
- Variation: invariant outside class - inside class (G)
- Variation: shell variables - shell queries (H)

Variations for USE model definition and USE shell commands

- `Variation: Identifier 'Subject/subject'` (I)
- `Variation: Replacement associationclass by class` (J)
- `Variation: Subject tree through composition` (K)

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)

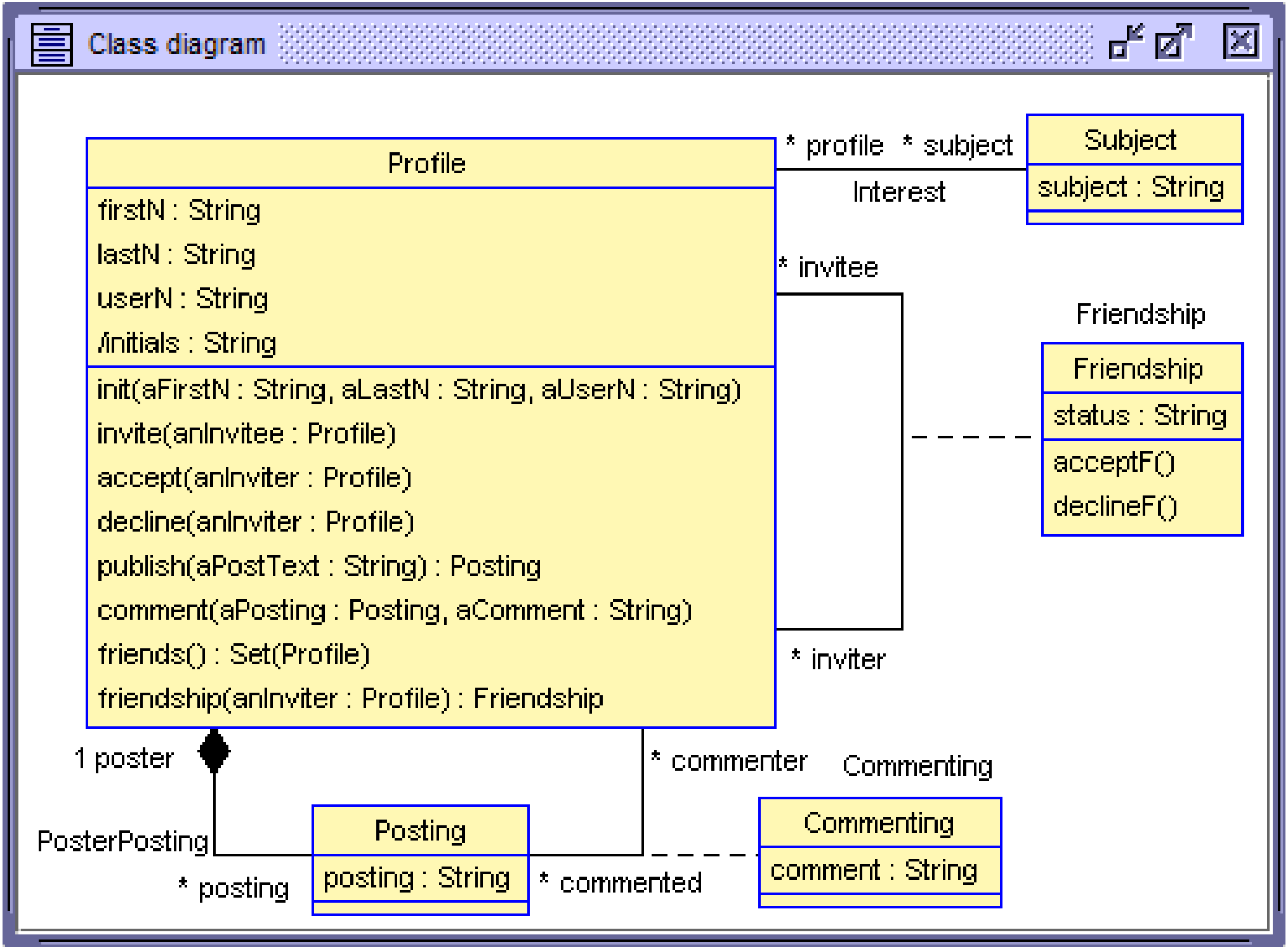
```
associationclass Friendship between
  Profile [*] role inviter
  Profile [*] role invitee
attributes
  status:String init:'pending'
operations
  acceptF()
    begin self.status:='accepted' end
  declineF()
    begin self.status:='declined' end
statemachines
  psm FriendshipLife
  states
    prenatal:initial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

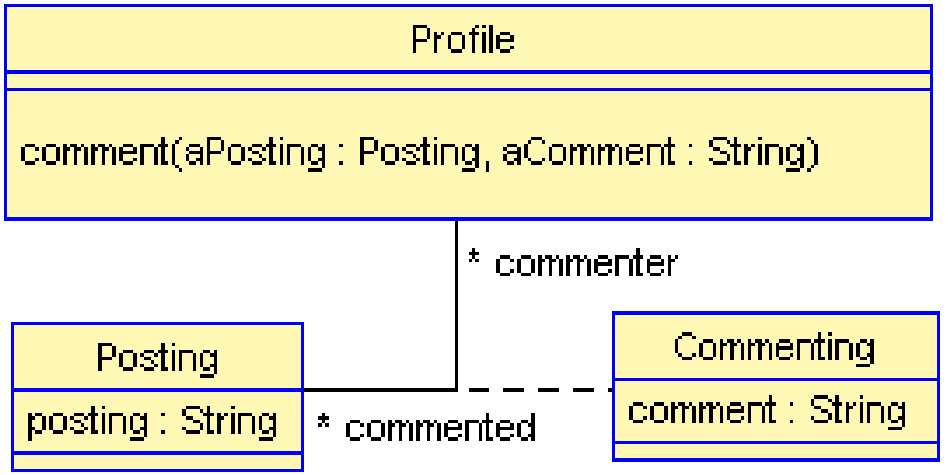

Class diagram
Profile
firstN : String
lastN : String
userN : String
/initials : String
init(aFirstN : String, aLastN : String, aUserN : String)
invite(anInvitee : Profile)
accept(anInviter : Profile)
decline(anInviter : Profile)
publish(aPostText : String) : Posting
comment(aPosting : Posting, aComment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

Textual model definition in USE - Variations A - Implementation

```
publish(aPostText:String):Posting
  begin
  declare p:Posting;
  p:=new Posting();
  p.posting:=aPostText;
  insert(self,p) into PosterPosting;
  result:=p
  end

publish(aPostText:String):Posting
  begin
  declare p:Posting;
  p:=Posting.allInstances->any(p|p.posting=aPostText);
  if p=null then
    p:=new Posting();
    p.posting:=aPostText;
    insert(self,p) into PosterPosting;
  end;
  result:=p;
  end

pre  nonEmpty: aPostText<>''
post newPosting: Posting.allInstances->exists(p |
       p.posting=aPostText and result=p)
```

Textual model definition in USE - Variations B - Postcondition

```
class Profile
...
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))

post commentingExists: Commenting.allInstances->exists(c |
      c.comment=aComment and
      c.commented=aPosting and
      c.commenter=self)
```



Textual model definition in USE - Variations C - Invariant

```
class Profile
...
inv uniqueUserName:
      Profile.allInstances->isUnique(userN)

inv uniqueUserName:
      Profile.allInstances->select(p | p.userN=self.userN)->size=1

inv uniqueUserName:
      Profile.allInstances->one(p | p.userN=self.userN)

inv uniqueUserName:
      Profile.allInstances->reject(p | p.userN<>self.userN)->size=1
      -- COL->select(e | p[e]) = COL->reject(e | not p[e])

inv uniqueUserName:
      Profile.allInstances->forAll(p1,p2 |
        p1<>p2 implies p1.userN<>p2.userN)

inv uniqueUserName:
      Profile.allInstances->forAll(p1,p2 |
        p1.userN=p2.userN implies p1=p2)
        -- (A implies B) <=> (not B implies not A)

...
```

Textual model definition in USE - Variations C - Invariant

```
class Profile
...
inv uniqueUserName:
      Profile.allInstances->isUnique(userN)

inv uniqueUserName:
      Profile.allInstances->size =
      Profile.allInstances.userN->asSet()->size
      -- this form of constraint used for class Subject

inv uniqueUserName:
      Profile.allInstances->size =
      Profile.allInstances->collect(p|p.userN)->asSet()->size

inv uniqueUserName:
      Profile.allInstances->one(p | p.userN=self.userN)

inv uniqueUserName:
      Profile.allInstances->exists(p1 | p1.userN=self.userN and
        not Profile.allInstances->exists(p2 |
          p1.userN=p2.userN and p1<>p2))

-- COL->one(e | p[e]) <=>
-- COL->exists(e1 | p[e1] and not COL->exists(e2 | p[e2] and e1<>e2))
...
```

Textual model definition in USE - Variations D - Attribute

```
class Friendship
attributes
  status:String init:'pending'
operations
  acceptF()  begin self.status:='accepted' end
  declineF() begin self.status:='declined' end

class Friendship
attributes
  status:String derived:
    if oclInState(pending) then 'pending' else
    if oclInState(accepted) then 'accepted' else
    if oclInState(declined) then 'declined' else null endif endif
endif
operations
  acceptF()  begin end
  declineF() begin end
  -- operation call changes statechart status
```

Textual model definition in USE - Variations E and F
Multiplicity and Role name

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

composition PosterPosting between
  Profile [0..1] role poster          -- Multiplicity
  Posting [*]    role posting
end

-------------------------------------------------------------

association Interest between          -- Implicit role names
  Profile [*]                         -- Class name with
  Subject [*]                         -- starting lower case letter
end

association Interest between          -- Explicit role names
  Profile [*] role profile
  Subject [*] role subject
end
```

Textual model definition in USE - Variations G – Inside/Outside

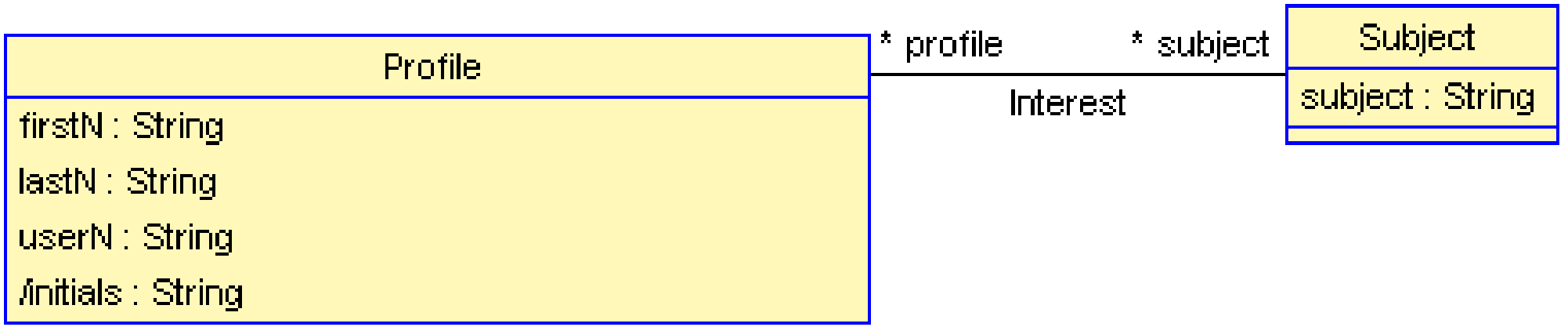
```
associationclass Commenting between
  Profile [*] role commenter
  Posting [*] role commented
attributes
  comment:String
end

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

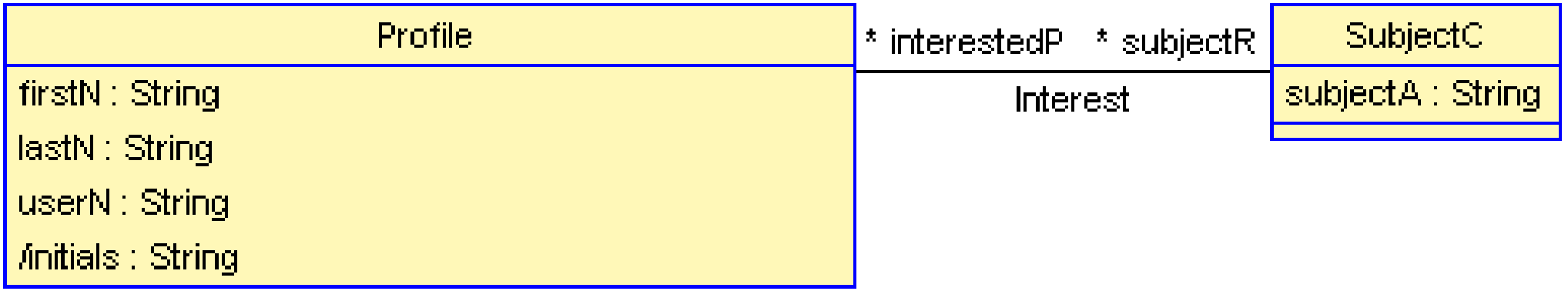
```
associationclass Commenting between
  Profile [*] role commenter
  Posting [*] role commented
attributes
  comment:String
constraints
  inv commentOnlyByFriends:
        commented.poster.friends()->includes(commenter)
end
```

USE shell with SOIL statements - Variations H – USE Shell

```
!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)

!merkel.publish('BMW, we have a problem')
!create may:Profile
!may.init('Theresa','May','motherTheresa')
!putin.comment(Posting.allInstances->any(p|p.poster=merkel),
                'May the Donald be with you')
!may.invite(merkel)

Shell command on several lines
use> \
> !putin.comment(Posting.allInstances->any(p|p.poster=merkel),
>                 'May the Donald be with you')
> .
```

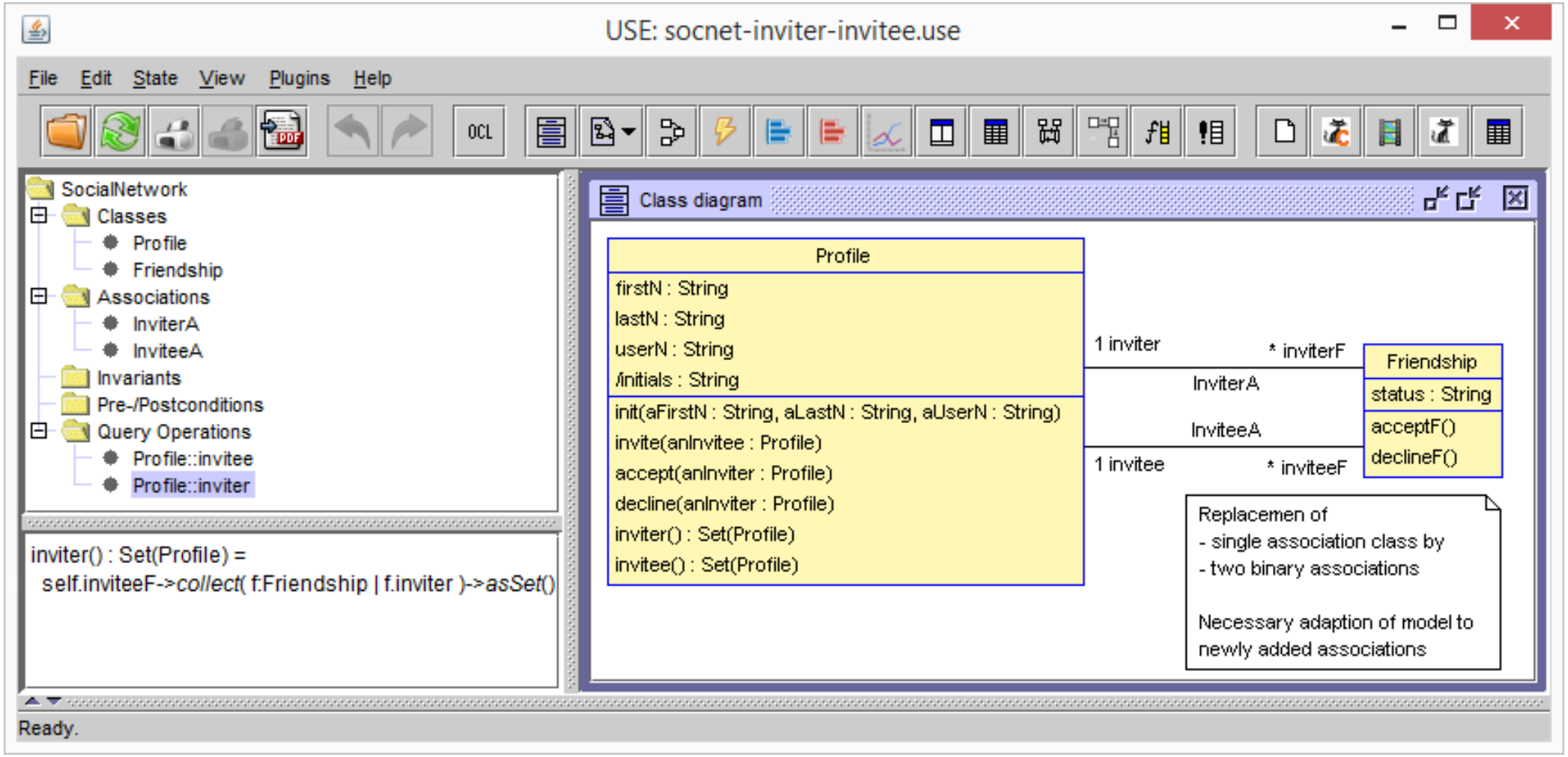
Textual model definition in USE - Variations I – 'Subject'






```
Originally:
Class Subject, attribute subject:String, role subject:Set(Subject)

Possible distinguishing identifier:
Class SubjectC, attribute subjectA:String, role subjectR:Set(SubjectC)
```

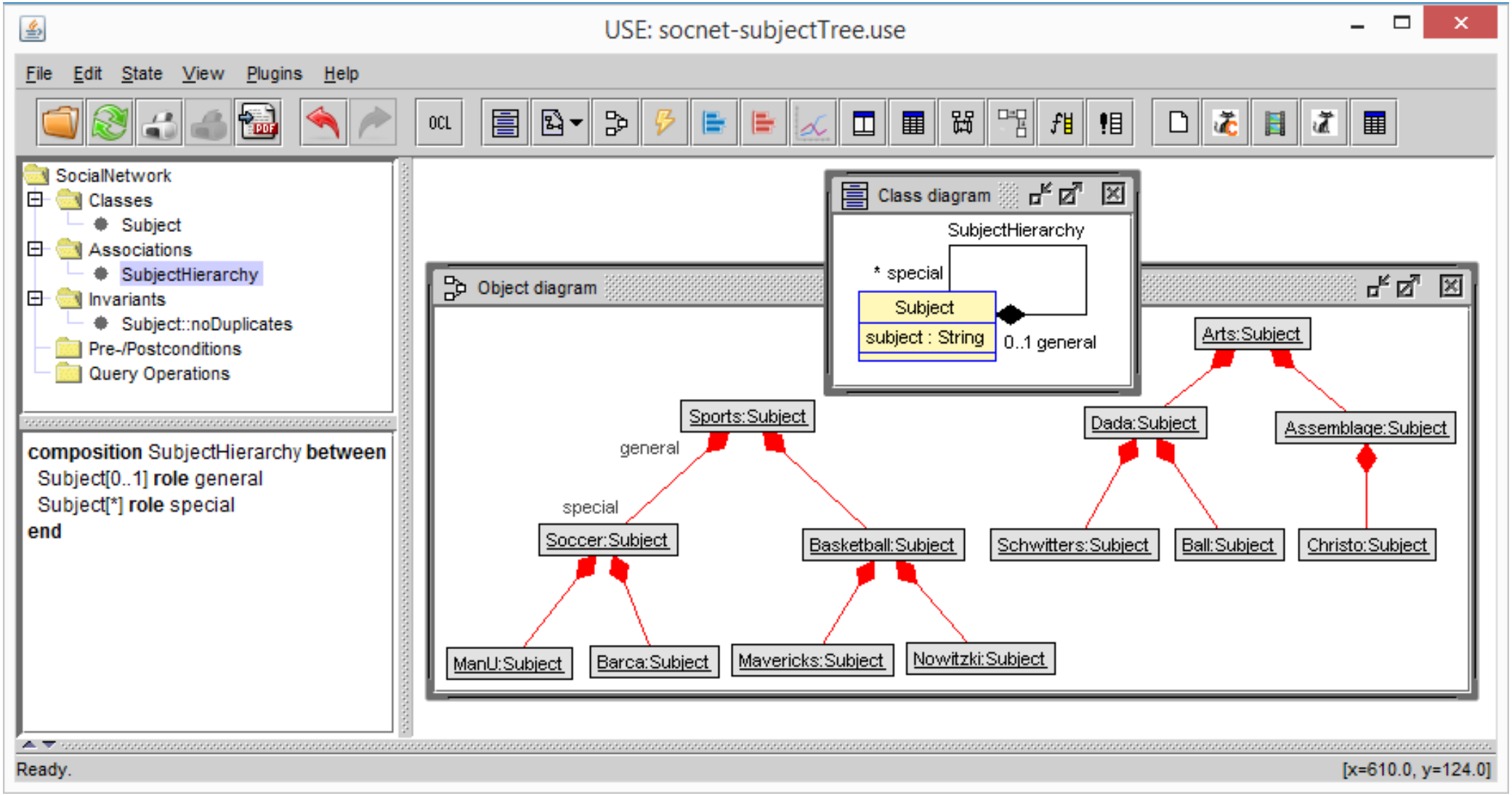
Textual model definition in USE - Variations J
Associationclass replaced by Class



Originally:
Friendship reflexive association class

Possible replacement by class with two associations:
assoc InviterA, assoc InviteeA

Textual model definition in USE - Variations K
Subject tree through reflexive composition



Structuring 'Subject' objects through a tree / hierarchy / ontology
- Subject object can have many specializations
- Subject object can have one or no generalization
- cycle-freeness given through use of 'composition', not 'association'

Thanks for your attention!