Analyzing Multiparty Interaction using Conversation Types

Luís Caires, Hugo Torres Vieira
Nova - New University of Lisbon

Motivation

- Software systems often rely on the collaboration between multiple parties to realize their tasks

  e.g., web-service applications

  How can we ensure protocol safety and progress in such a decentralized and dynamic setting?

- Sessions [Honda93,Honda et al.98] have been widely used to model typeful binary interaction

  How can we extend classical sessions so as to address dynamic multiparty interaction?

Sessions and Conversations

- Session type theory

  Systems modeled in the π-calculus

  Types describe the behavior of a single participant
  (≈ local types [HondaYoshidaCarbone07-08])

  progress analysis based on well-founded ordering of channels
  [Dezani et al.07]

- Conversation type theory (this talk)

  Systems modeled in the π-calculus extended with labels
  (to support distinguished interaction in a single medium)

  Types describe the behavior of a subset of participants
  (mixing global and local type specifications)

  progress analysis based on well-founded ordering of events
  and its propagation in communication

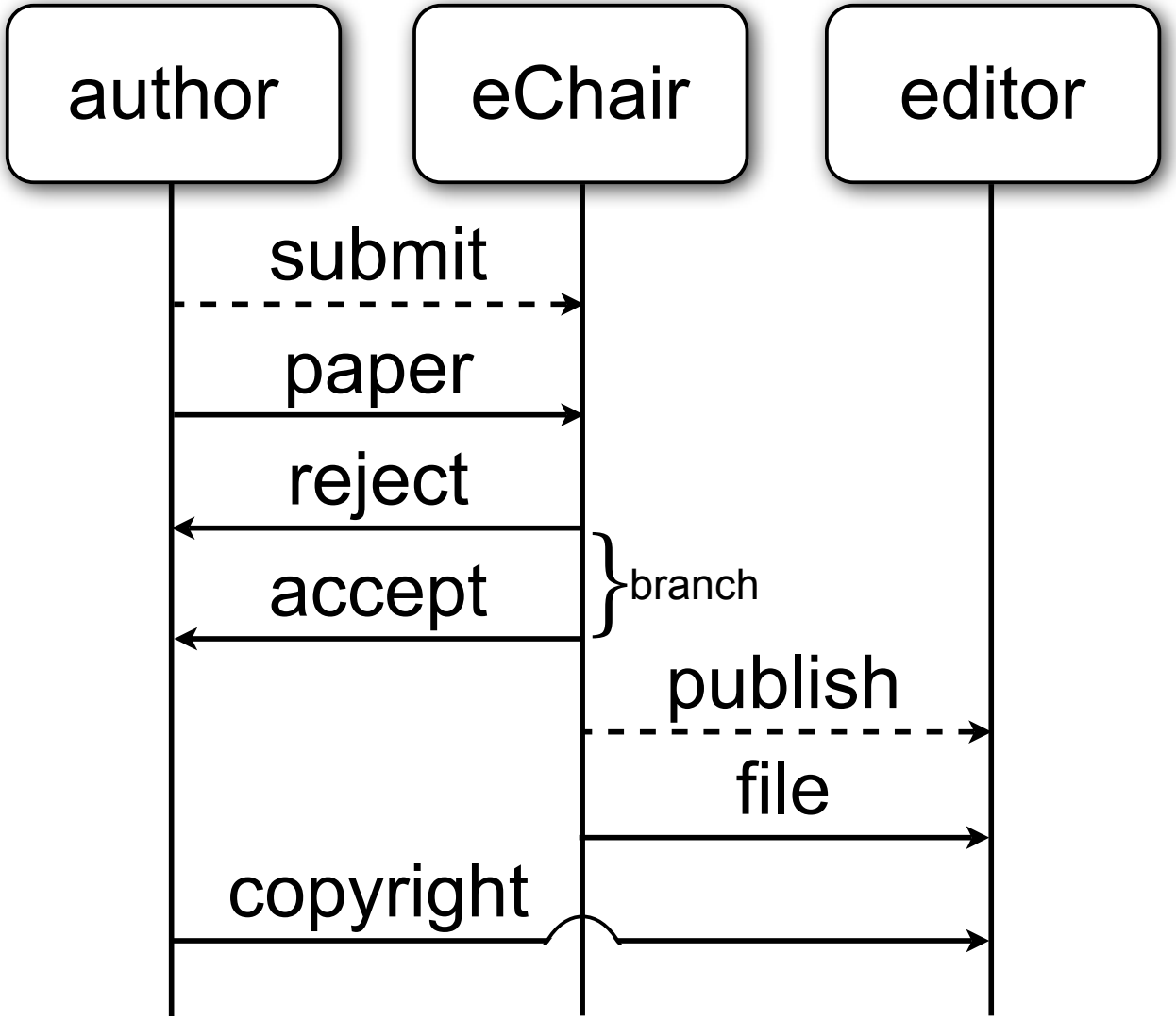
π−calculus + labels

π−calculus + labels

P :: =	$\mathbf{0}$	(Inaction)
	$\mid P \mid Q$	(Parallel Composition)
	$\mid (\boldsymbol{\nu} a)\, P$	(Name Restriction)
	$\mid$ **rec** $X.P$	(Recursion)
	$\mid X$	(Variable)
	$\mid \Sigma_{i \in I}\, \alpha_i.P_i$	(Prefix Guarded Choice)
α :: =	$n \bullet \text{label}?(x_1,\ldots,x_k)$	(Input)
	$\mid n \bullet \text{label}!(n_1,\ldots,n_k)$	(Output)

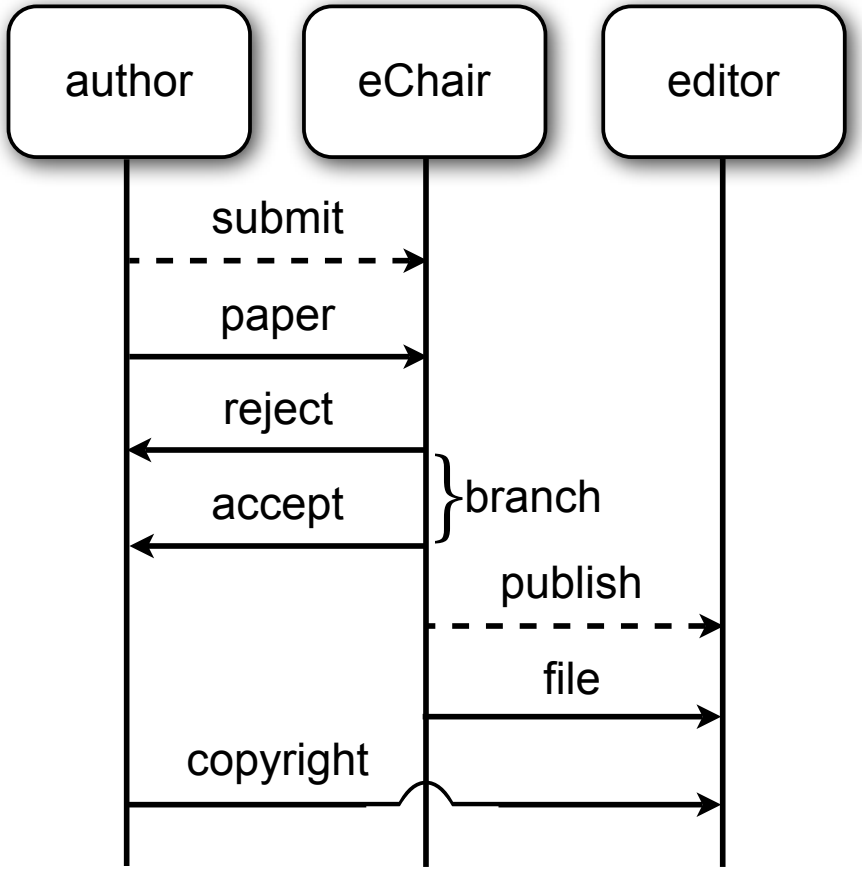
The eChair System

The eChair System



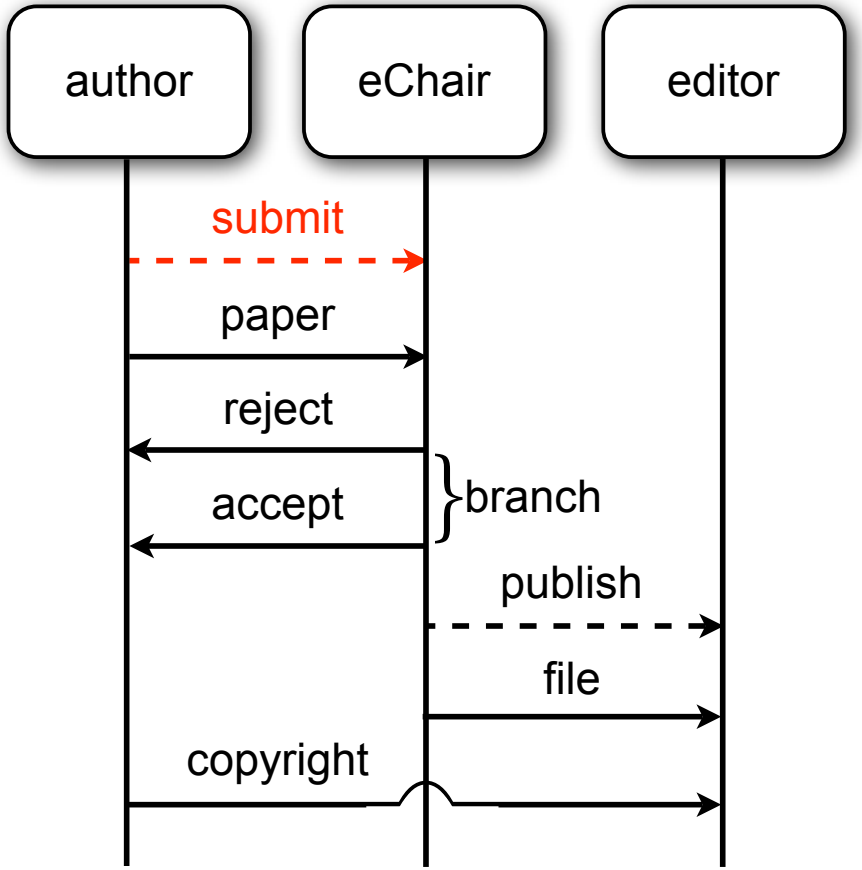
eChair System Code

```
(νchat)
(eChair • submit!(chat).
      chat • paper!(pdf).
      (chat • reject?()
       +
       chat • accept?(). chat • copyright!()))
|
*eChair • submit?(x).
      x • paper?(pdf).
      (x • reject!()
       +
       x • accept!().
            editor • publish!(x). x • file!(pdf))
|
*editor • publish?(y).
      y • file?(pdf). y • copyright?()
```



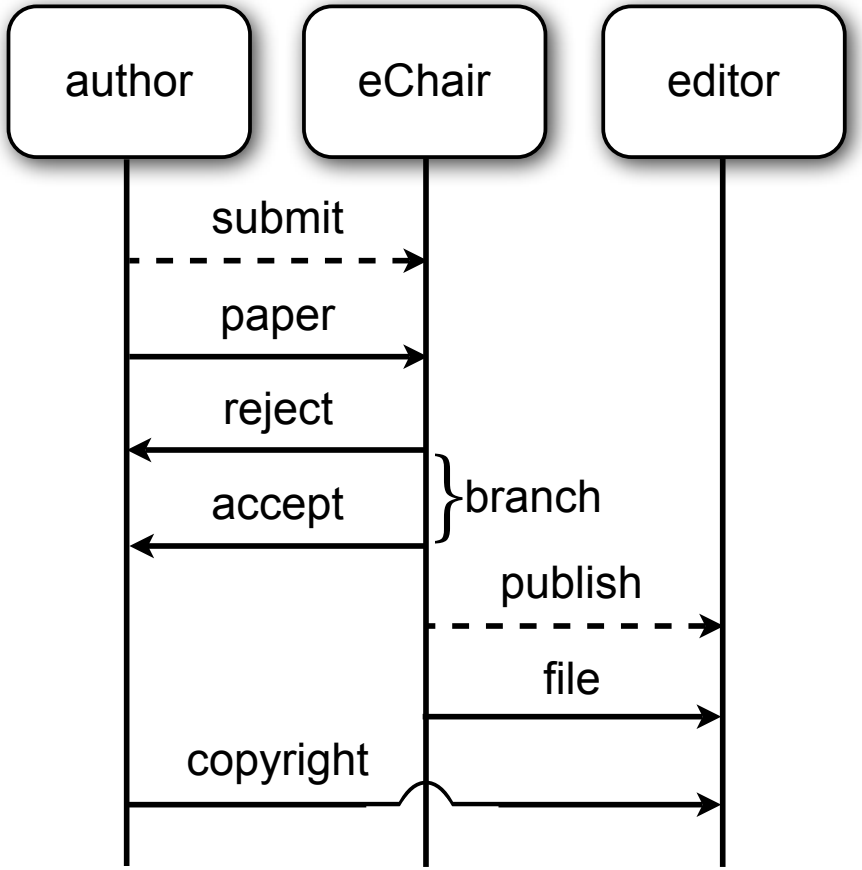
eChair System Run

```
(νchat)
(eChair • submit!(chat).
      chat • paper!(pdf).
      (chat • reject?()
       +
       chat • accept?(). chat • copyright!()))
|
*eChair • submit?(x).
      x • paper?(pdf).
      (x • reject!()
       +
       x • accept!().
            editor • publish!(x). x • file!(pdf))
|
*editor • publish?(y).
      y • file?(pdf). y • copyright?()
```



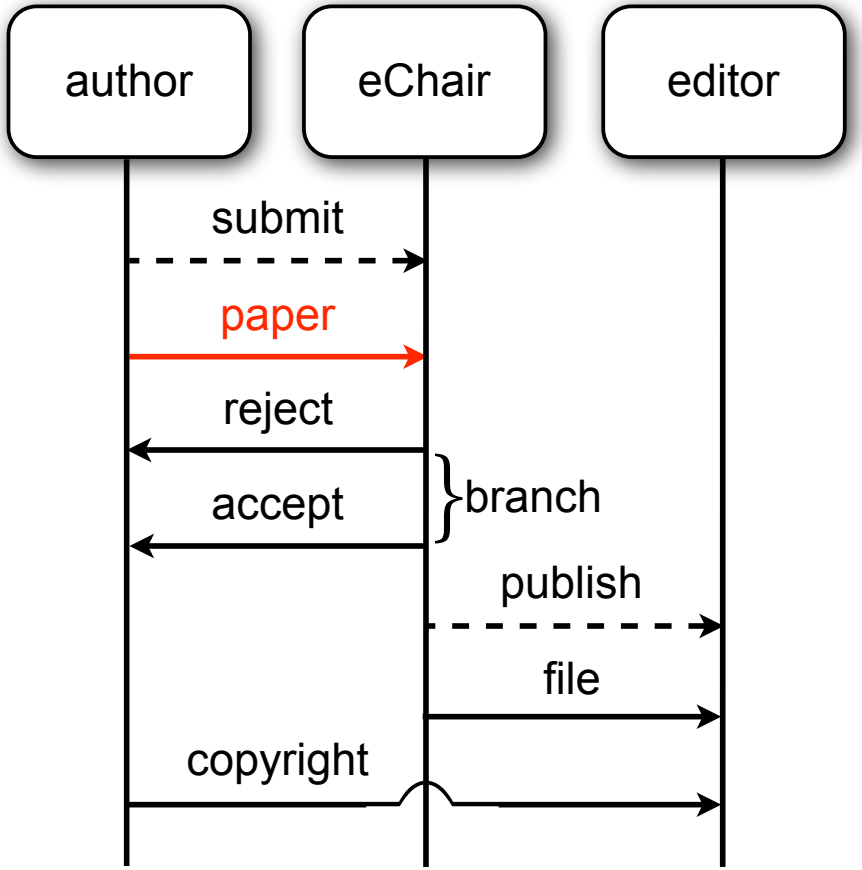
eChair System Run

```
(νchat)
(eChair • submit!(chat).
      chat • paper!(pdf).
      (chat • reject?()
       +
       chat • accept?(). chat • copyright!())
|
*eChair • submit?(x).
      chat • paper?(pdf).
      (chat • reject!()
       +
       chat • accept!().
            editor • publish!(chat). chat • file!(pdf)))
|
*editor • publish?(y).
      y • file?(pdf). y • copyright?()
```



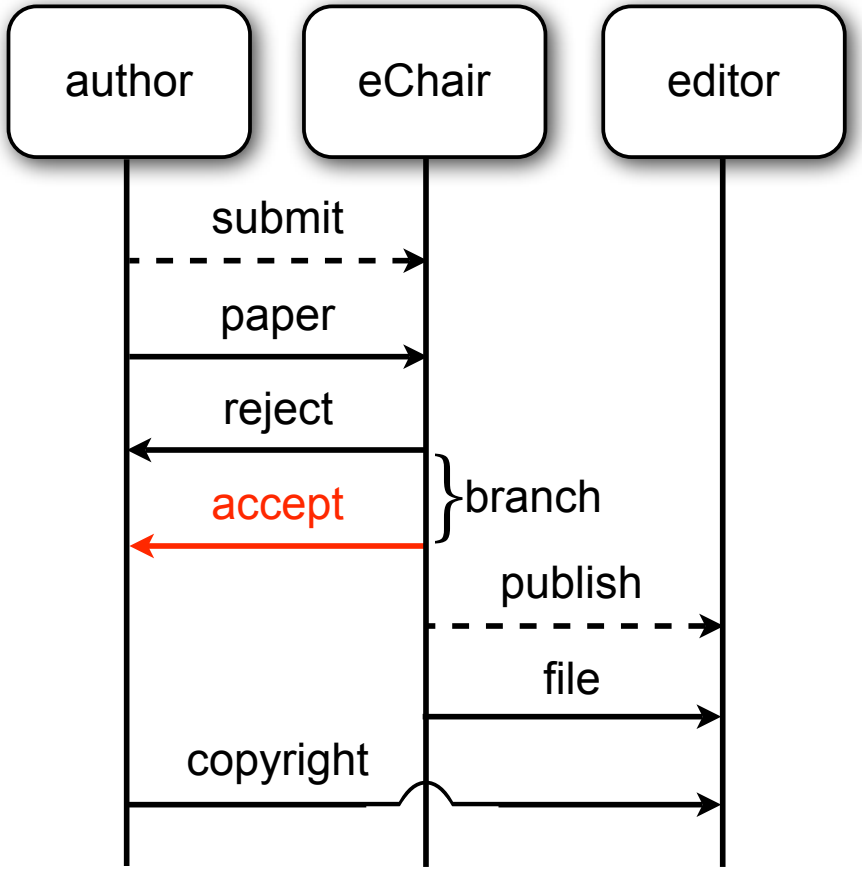
eChair System Run

```
(νchat)
(eChair • submit!(chat).
      chat • paper!(pdf).
      (chat • reject?()
       +
       chat • accept?(). chat • copyright!())
|
*eChair • submit?(x).
      chat • paper?(pdf).
      (chat • reject!()
       +
       chat • accept!().
            editor • publish!(chat). chat • file!(pdf)))
|
*editor • publish?(y).
      y • file?(pdf). y • copyright?()
```



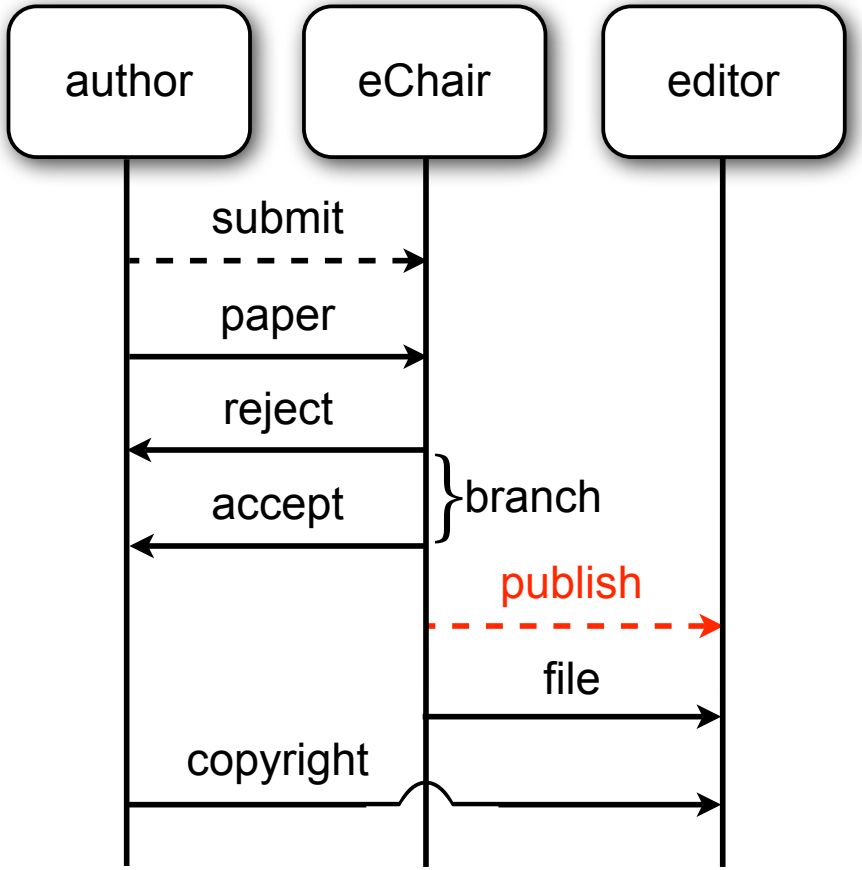
eChair System Run

```
(νchat)
(eChair • submit!(chat).
     chat • paper!(pdf).
     (chat • reject?()
      +
      chat • accept?(). chat • copyright!())
|
*eChair • submit?(x).
     chat • paper?(pdf).
     (chat • reject!()
      +
      chat • accept!().
          editor • publish!(chat). chat • file!(pdf)))
|
*editor • publish?(y).
     y • file?(pdf). y • copyright?()
```



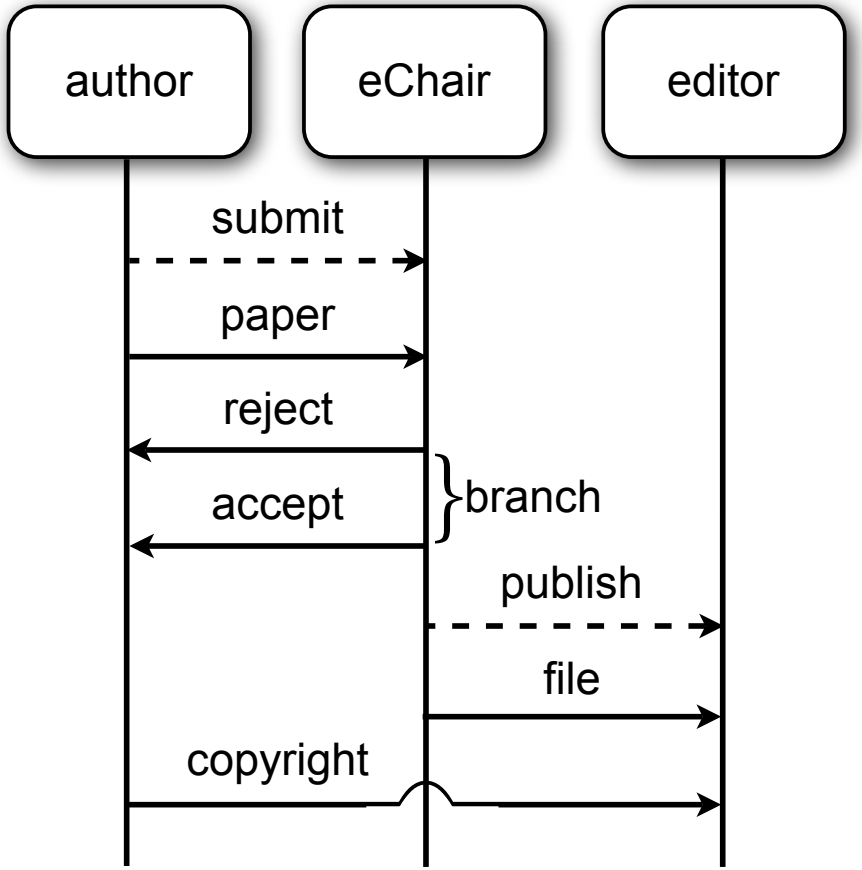
eChair System Run

```
(νchat)
(eChair • submit!(chat).
     chat • paper!(pdf).
     (chat • reject?()
      +
      chat • accept?(). chat • copyright!()
|
*eChair • submit?(x).
     chat • paper?(pdf).
     (chat • reject!()
      +
      chat • accept!().
          editor • publish!(chat). chat • file!(pdf))
|
*editor • publish?(y).
     y • file?(pdf). y • copyright?()
```



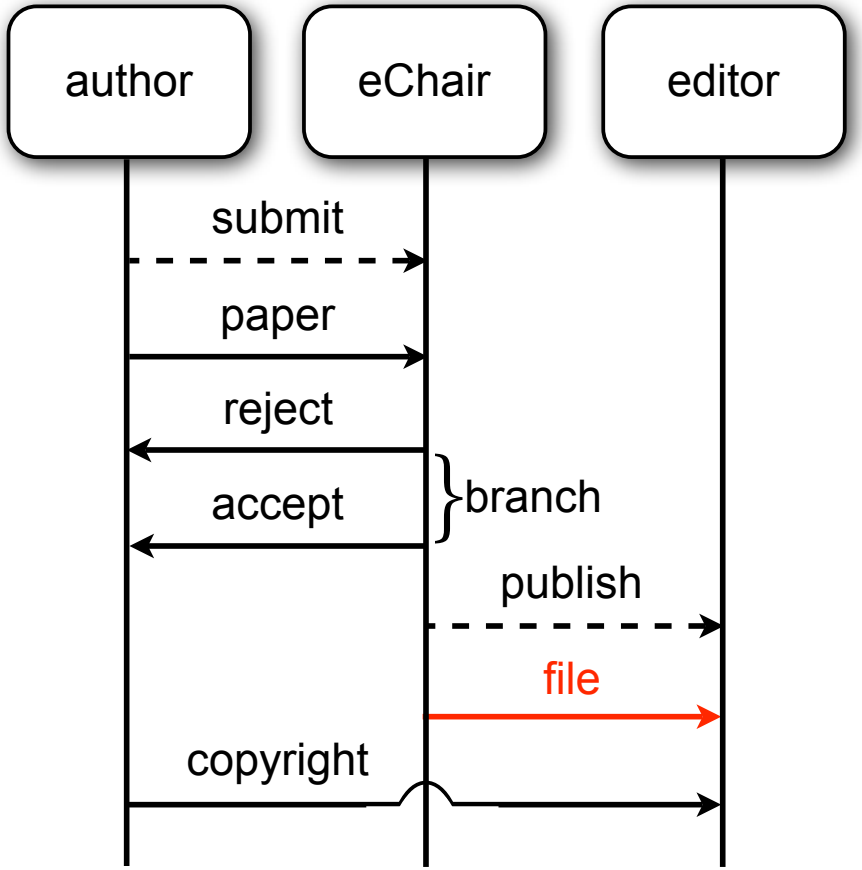
eChair System Run

```
(νchat)
(eChair • submit!(chat).
     chat • paper!(pdf).
     (chat • reject?()
      +
      chat • accept?(). chat • copyright!()
|
*eChair • submit?(x).
     chat • paper?(pdf).
     (chat • reject!()
      +
      chat • accept!().
          editor • publish!(chat). chat • file!(pdf)
|
*editor • publish?(y).
     chat • file?(pdf). chat • copyright?())
```



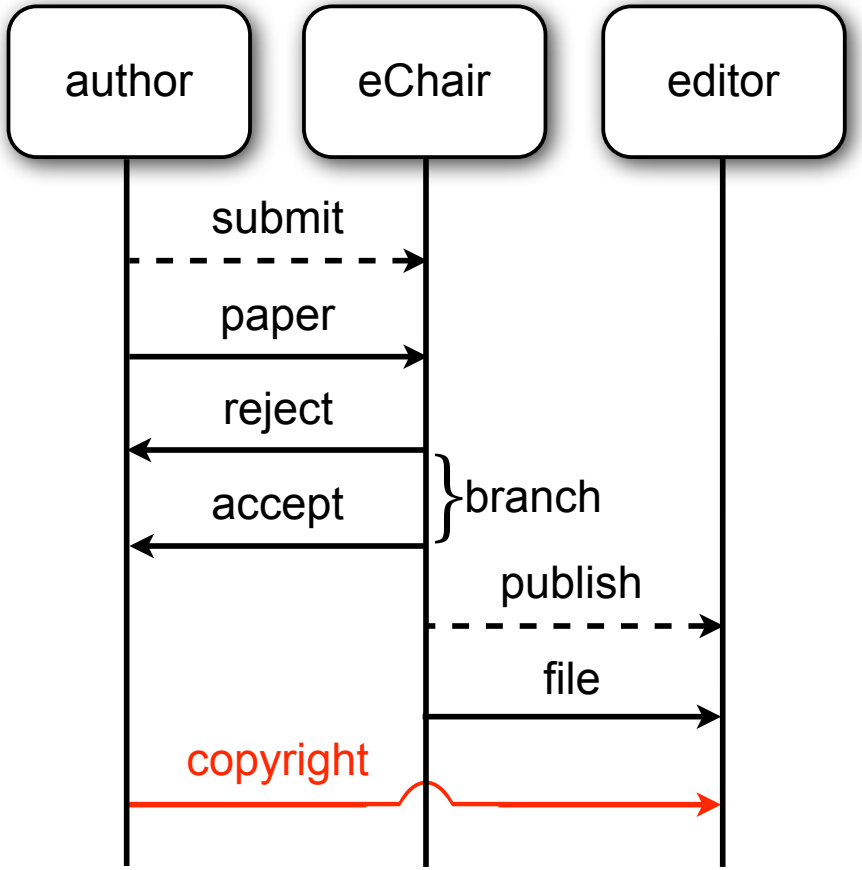
eChair System Run

```
(νchat)
(eChair • submit!(chat).
      chat • paper!(pdf).
      (chat • reject?()
       +
       chat • accept?(). chat • copyright!()
|
*eChair • submit?(x).
      chat • paper?(pdf).
      (chat • reject!()
       +
       chat • accept!().
            editor • publish!(chat). chat • file!(pdf)
|
*editor • publish?(y).
      chat • file?(pdf). chat • copyright?())
```



eChair System Run

```
(νchat)
(eChair • submit!(chat).
      chat • paper!(pdf).
      (chat • reject?()
       +
       chat • accept?(). chat • copyright!()
|
*eChair • submit?(x).
      chat • paper?(pdf).
      (chat • reject!()
       +
       chat • accept!().
            editor • publish!(chat). chat • file!(pdf)
|
*editor • publish?(y).
      chat • file?(pdf). chat • copyright?())
```



Conversation Types

Conversation Types

- Typing judgement

$$P :: n_1{:}B_1 \mid n_2{:}B_2 \mid \ldots \mid n_k{:}B_k$$

says P interacts in n_i accordingly to the B_i spec

- Behavioral types (B) extend session types:

$$B ::= B_1 \mid B_2 \mid \mathbf{0} \mid \text{rec } X.B \mid X \mid \&_{i \in I}\{M_i.B_i\} \mid \oplus_{i \in I}\{M_i.B_i\}$$

Message types (M) are labeled and describe both external and internal message exchanges

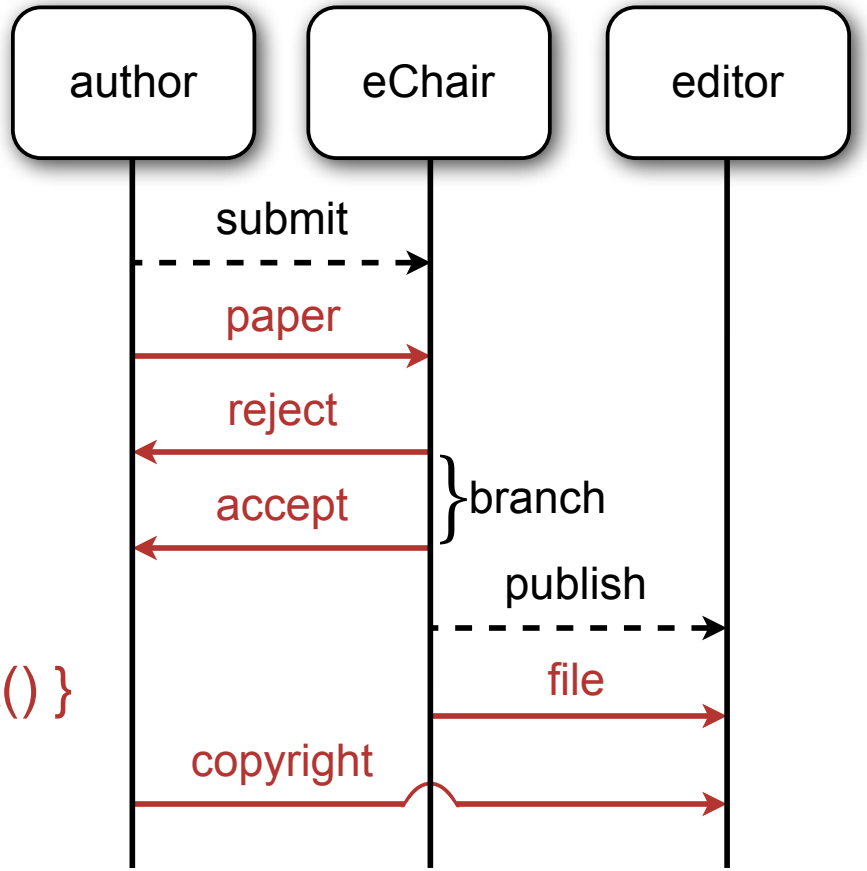
$$M ::= p\ label(B) \qquad p ::= ! \mid ? \mid \tau$$

Typing *chat* Conversation

Typing *chat* Conversation

chat:

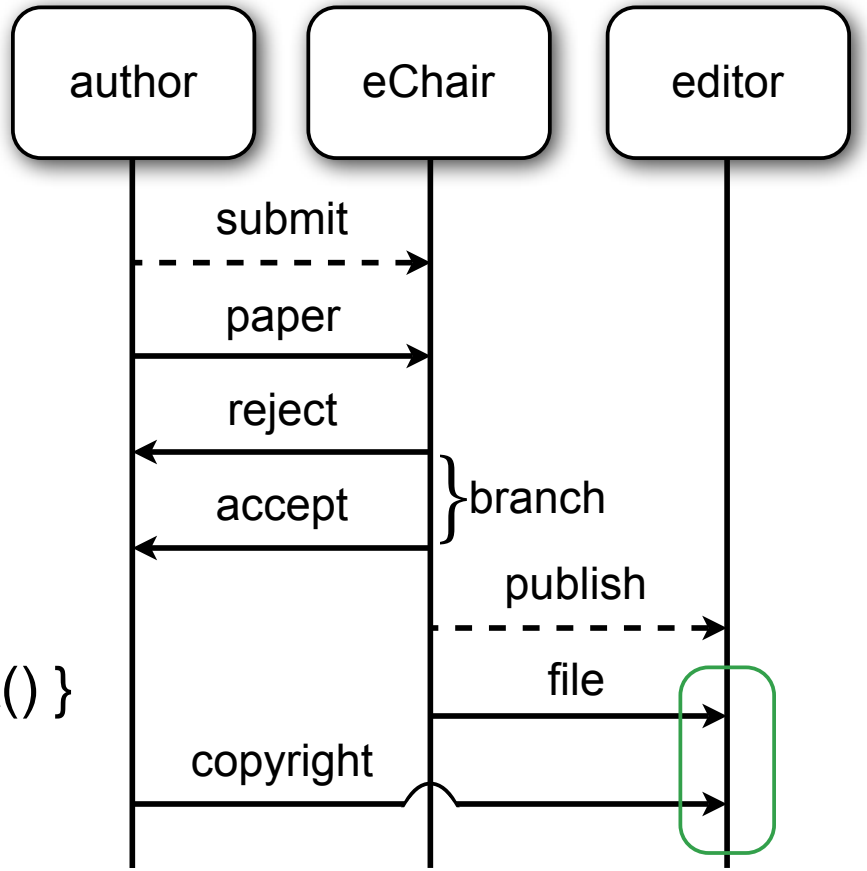
τ paper(T_{pdf}).
  $\oplus$ { τ reject();
    τ accept().
      τ file(T_{pdf}).
      τ copyright() }



Typing *chat* Conversation

chat:

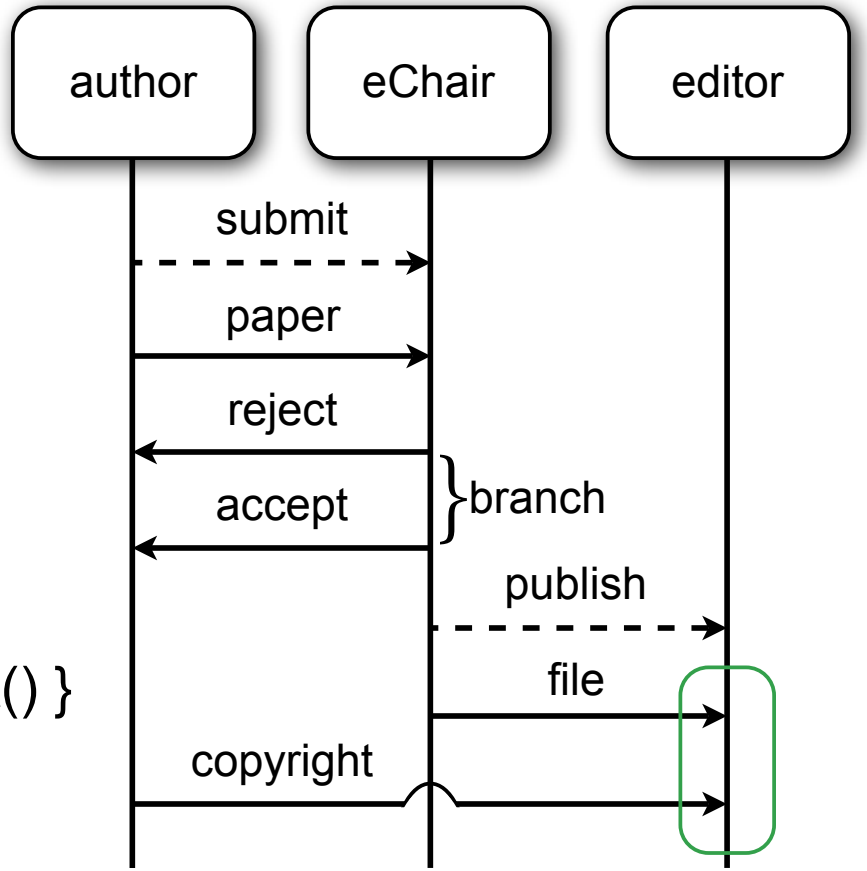
τ paper(T_{pdf}).
  $\oplus$ { τ reject();
      τ accept().
        τ file(T_{pdf}).
        τ copyright() }



Typing *chat* Conversation

chat:

τ paper(T_{pdf}).
  $\oplus$ { τ reject();
    τ accept().
      τ file(T_{pdf}).
      τ copyright() }

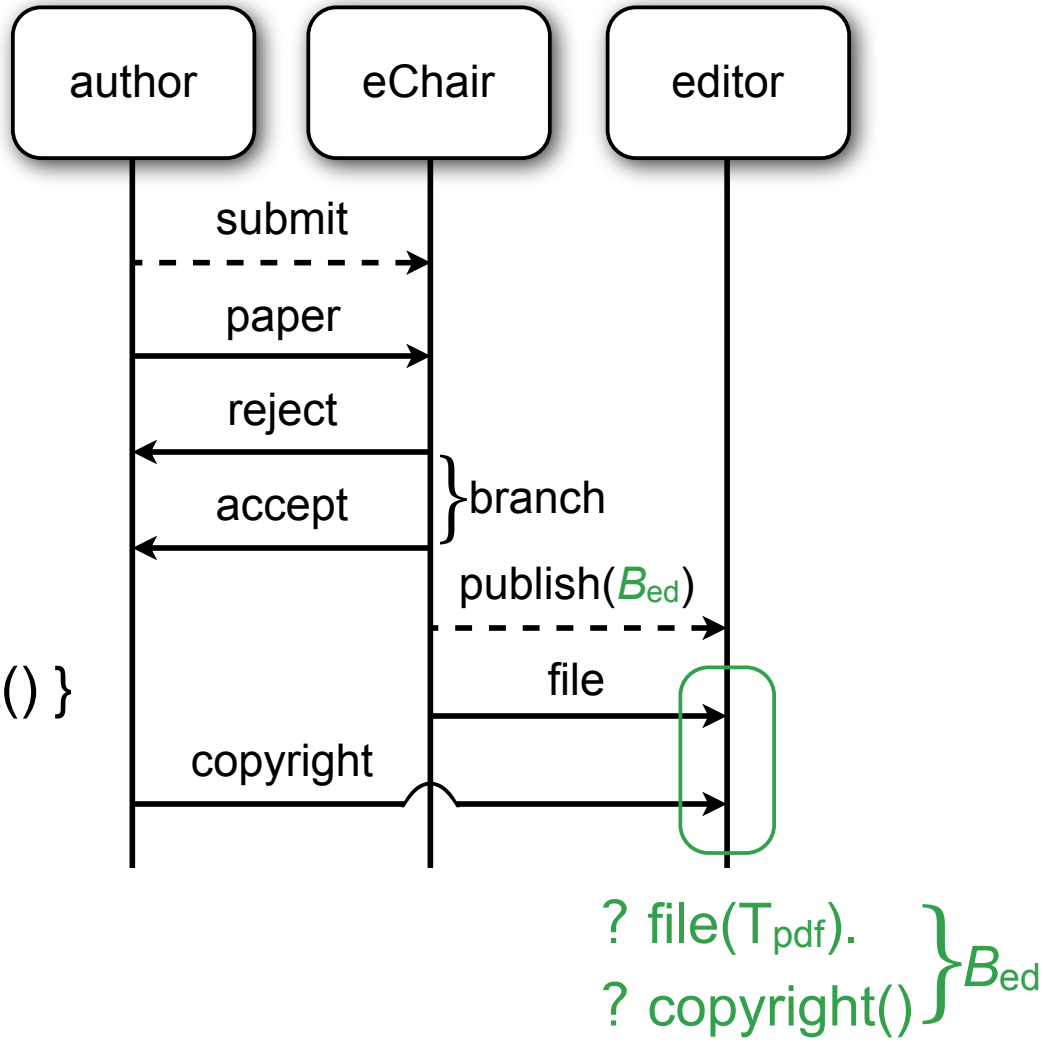


? file(T_{pdf}).
? copyright()

Typing *chat* Conversation

chat:

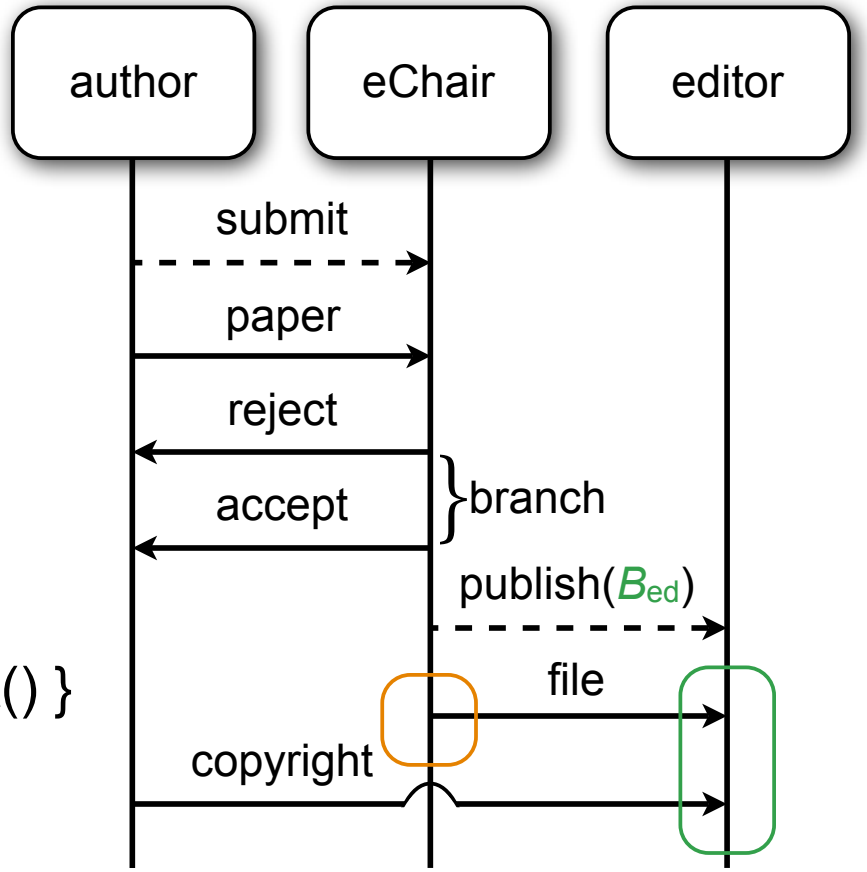
τ paper(T_{pdf}).
  $\oplus$ { τ reject();
      τ accept().
        τ file(T_{pdf}).
        τ copyright() }



Typing *chat* Conversation

chat:

τ paper(T_{pdf}).
  $\oplus$ { τ reject();
    τ accept().
      τ file(T_{pdf}).
      τ copyright() }

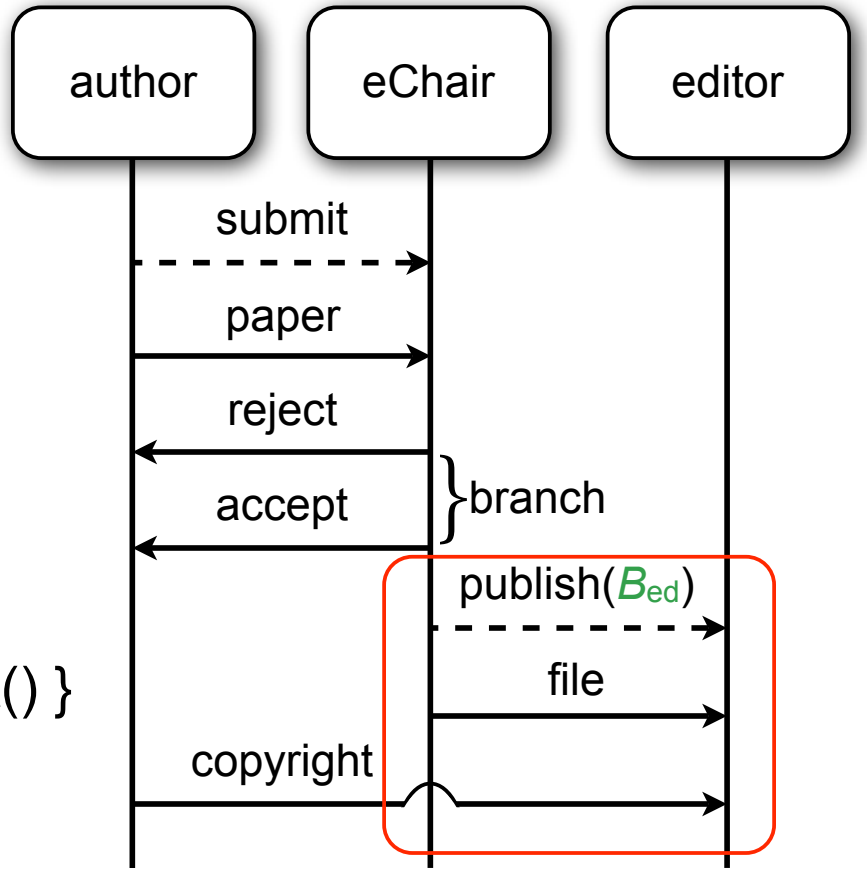


! file(T_{pdf}) ? file(T_{pdf}).
? copyright()

Typing *chat* Conversation

chat:

τ paper(T_{pdf}).
  $\oplus$ { τ reject();
    τ accept().
      τ file(T_{pdf}).
      τ copyright() }

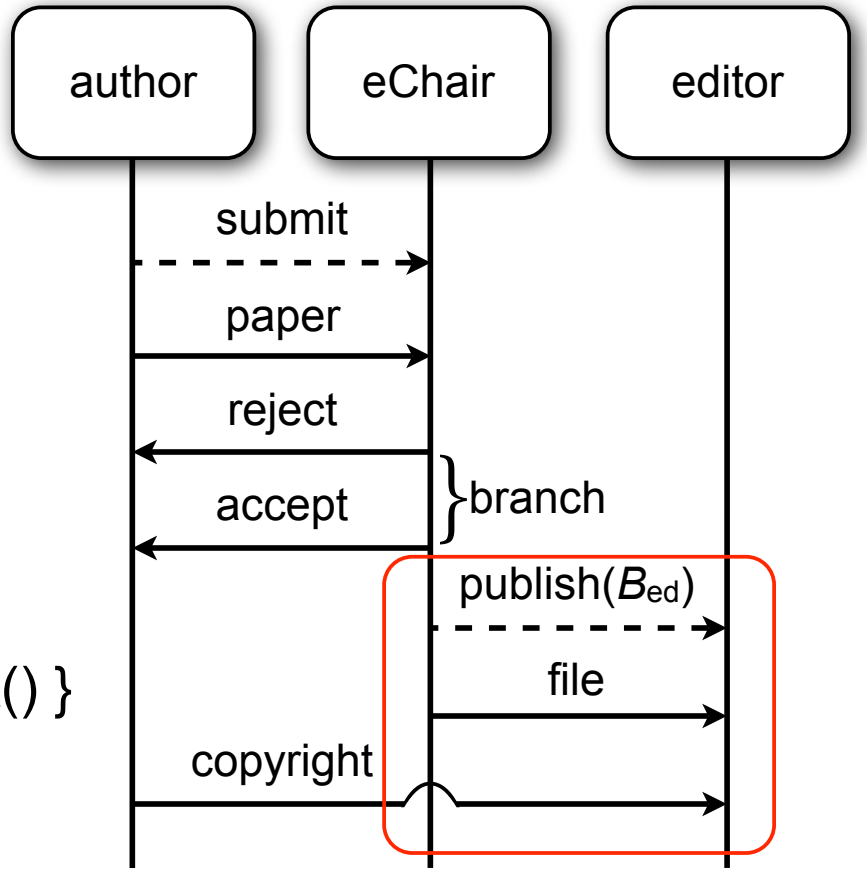


τ file(T_{pdf}). = ! file(T_{pdf}) $\bowtie$? file(T_{pdf}).
? copyright() ? copyright()

Typing *chat* Conversation

chat:

τ paper(T_{pdf}).
  $\oplus$ { τ reject();
    τ accept().
      τ file(T_{pdf}).
      τ copyright() }

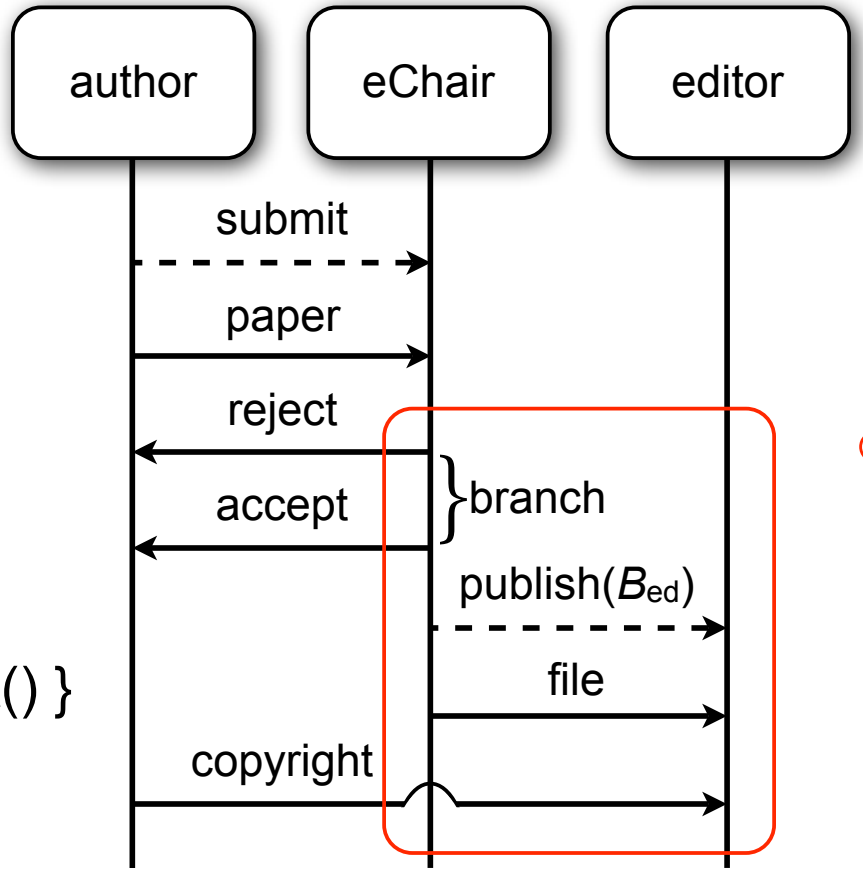


τ file(T_{pdf}).
? copyright()

Typing *chat* Conversation

chat:

τ paper(T_{pdf}).
  $\oplus$ { τ reject();
    τ accept().
      τ file(T_{pdf}).
      τ copyright() }

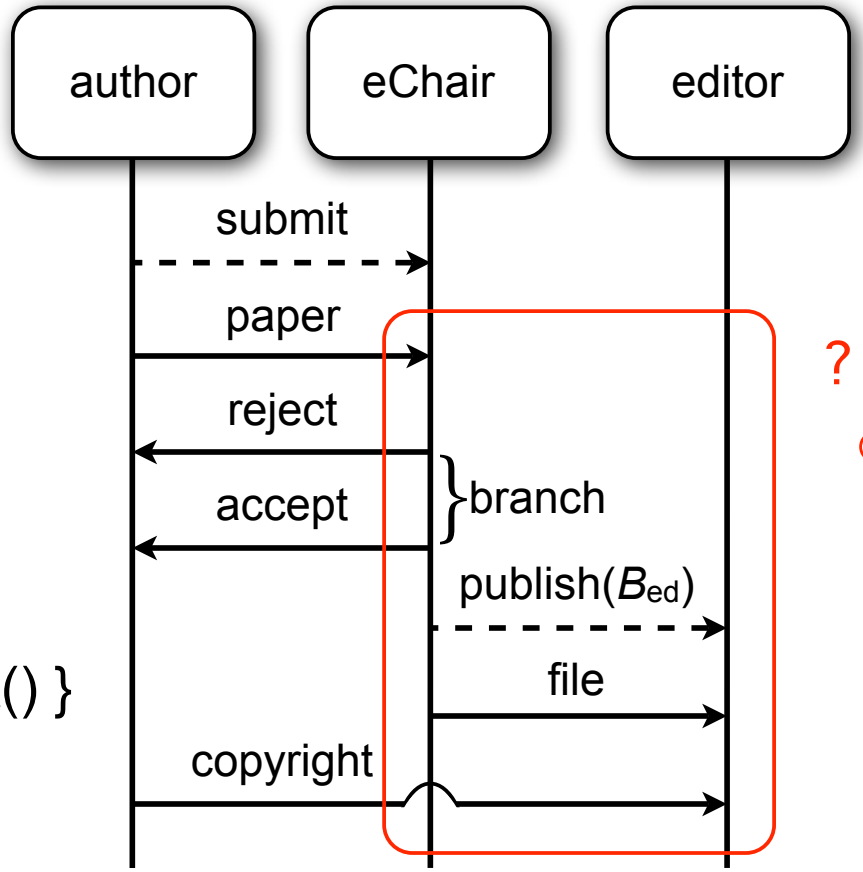


$\oplus$ { ! reject();
  ! accept().
    τ file(T_{pdf}).
    ? copyright() }

Typing *chat* Conversation

chat:

τ paper(T_{pdf}).
  $\oplus$ { τ reject();
    τ accept().
      τ file(T_{pdf}).
      τ copyright() }

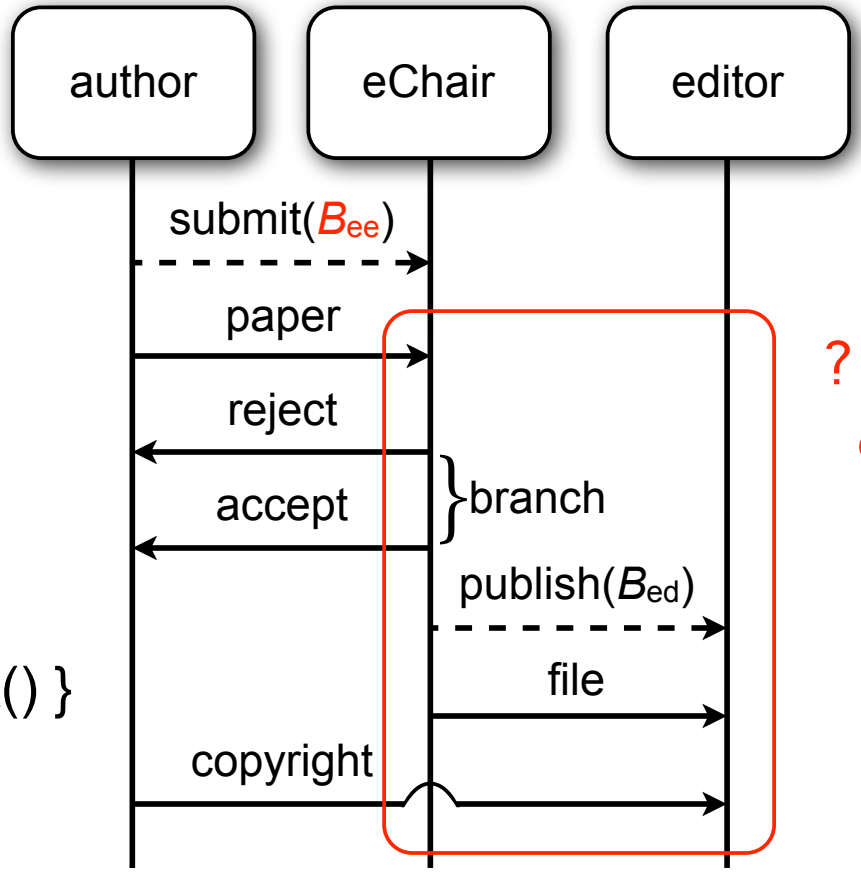


? paper(T_{pdf}).
  $\oplus$ { ! reject();
    ! accept().
      τ file(T_{pdf}).
      ? copyright() }

Typing *chat* Conversation

chat:

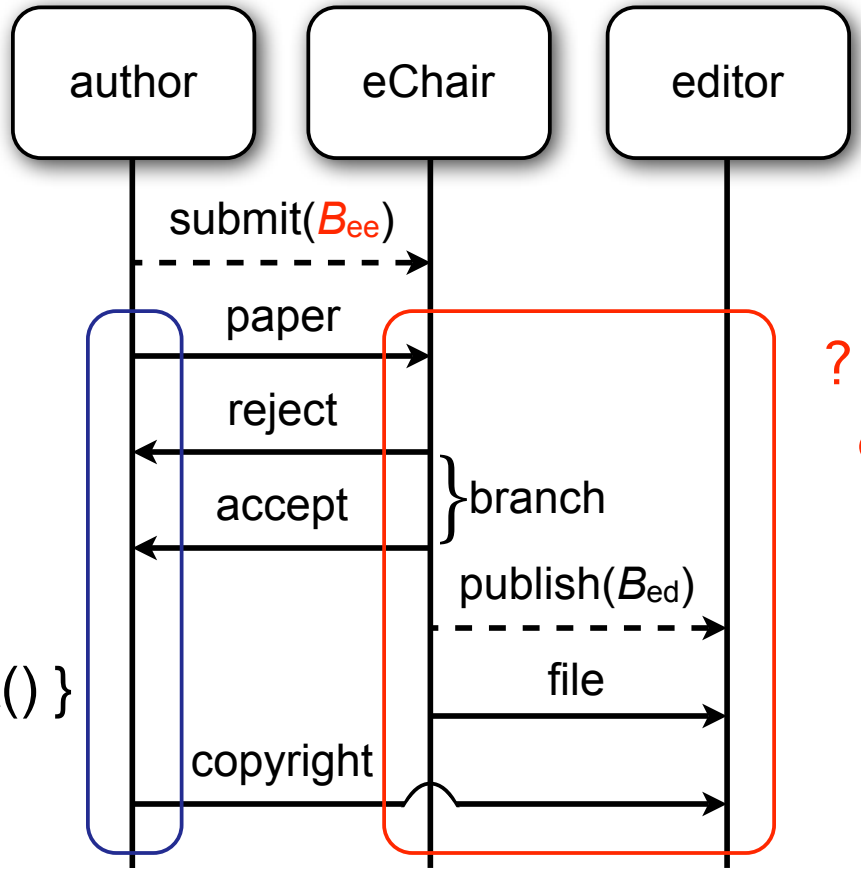
τ paper(T_{pdf}).
  $\oplus$ { τ reject();
    τ accept().
      τ file(T_{pdf}).
      τ copyright() }



B_{ee}

? paper(T_{pdf}).
  $\oplus$ { ! reject();
    ! accept().
      τ file(T_{pdf}).
      ? copyright() }

Typing *chat* Conversation



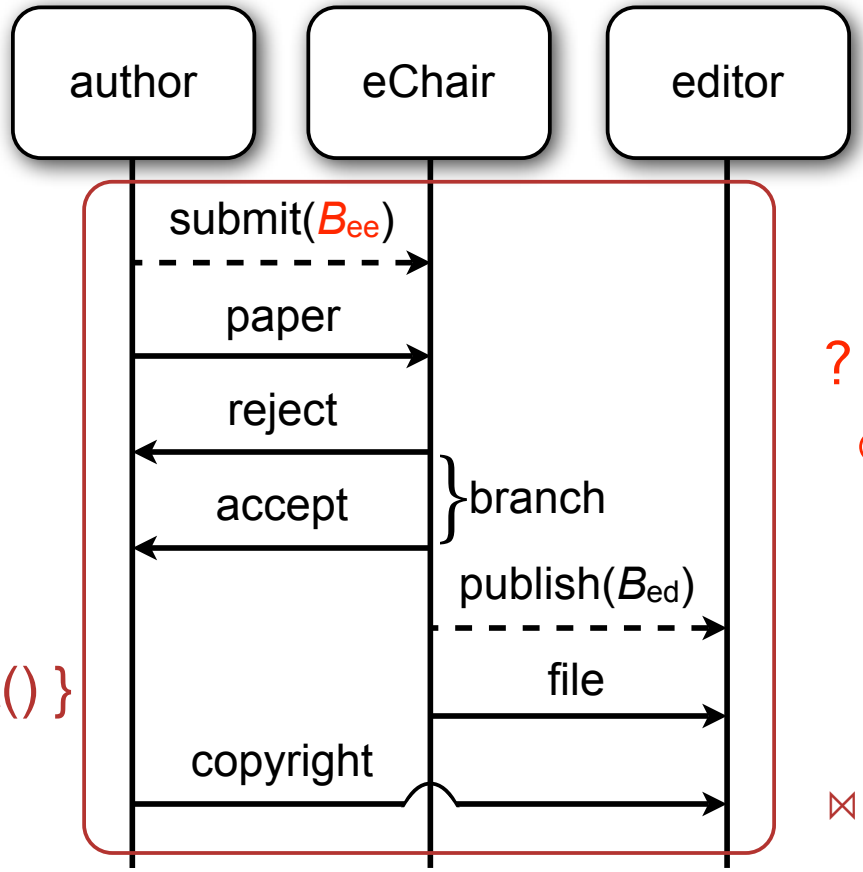
chat:

τ paper(T_{pdf}).
  $\oplus$ { τ reject();
    τ accept().
      τ file(T_{pdf}).
      τ copyright() }

? paper(T_{pdf}).
  $\oplus$ { ! reject();
    ! accept().
      τ file(T_{pdf}).
      ? copyright() }

! paper(T_{pdf}). $\oplus$ { ? reject();
  ? accept(). ! copyright() } B_{au}

Typing *chat* Conversation



chat:

τ paper(T_{pdf}).
  $\oplus$ { τ reject();
    τ accept().
      τ file(T_{pdf}).
      τ copyright() }

=

! paper(T_{pdf}). $\oplus$ { ? reject();
  ? accept(). ! copyright() }

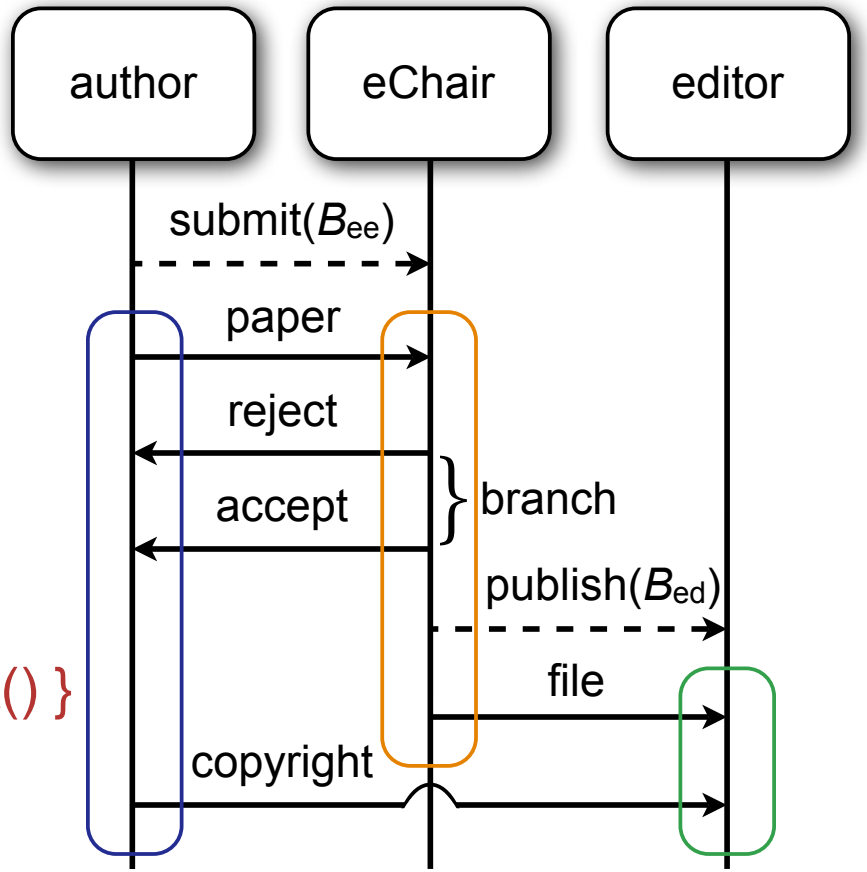
$\bowtie$

? paper(T_{pdf}).
  $\oplus$ { ! reject();
    ! accept().
      τ file(T_{pdf}).
      ? copyright() }

Typing *chat* Conversation

chat:

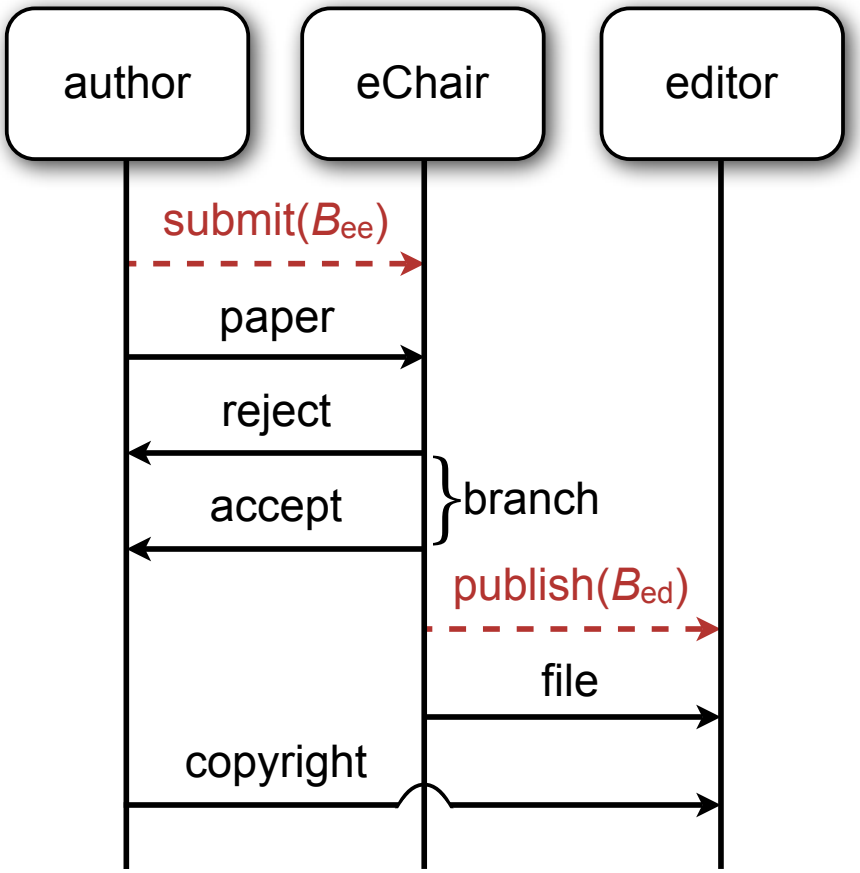
τ paper(T_{pdf}).
  $\oplus$ { τ reject();
    τ accept().
      τ file(T_{pdf}).
      τ copyright() }



= $B_{au} \bowtie B_{ec} \bowtie B_{ed}$

Typing eChair System

$(\nu chat)$
$(eChair \bullet \text{submit!}(chat).$
$\quad chat \bullet \text{paper!}(pdf).$
$\quad (chat \bullet \text{reject?}()$
$\quad +$
$\quad chat \bullet \text{accept?}().\ chat \bullet \text{copyright!}()))$
$|$
$*eChair \bullet \text{submit?}(x).$
$\quad x \bullet \text{paper?}(pdf).$
$\quad (x \bullet \text{reject!}()$
$\quad +$
$\quad x \bullet \text{accept!}().$
$\quad\quad editor \bullet \text{publish!}(x).\ x \bullet \text{file!}(pdf))$
$|$
$*editor \bullet \text{publish?}(y).$
$\quad y \bullet \text{file?}(pdf).\ y \bullet \text{copyright?}()$
$::$
$eChair : \tau\ \text{submit}(B_{ee}) \mid *?\ \text{submit}(B_{ee}) \mid$
$editor : \tau\ \text{publish}(B_{ed}) \mid *?\ \text{publish}(B_{ed})$



Results

Results

Theorem (Subject Reduction)

Let P be a well-typed process, $P :: T$.

If $P \rightarrow Q$ then there is T' such that $T \rightarrow T'$ and $Q :: T'$.

Proposition (Error Freeness)

Let P be a well-typed process. Then P is not an *error:*

P has no communication errors; P has no illegal message races

Corollary (Type Safety)

Let P be a well-typed process. If $P \rightarrow^* Q$ then Q is not an *error*.

Corollary (Conversation Fidelity)

Let P be a well-typed process, $P :: T$.

Then all conversations in P follow the protocols prescribed by T.

Proving Progress of Conversations

Proving Progress of Conversations

- We complement conversation typing with a proof system to ensure deadlock absence.

  As traditional methods (Lynch80, Kobayashi06, Dezani et al.07) we rely on imposing an ordering on **events.**

- Judgement $\Gamma;\Delta \vdash P$

  Events in *P* follow a well-founded order determined by $\Gamma;\Delta$.

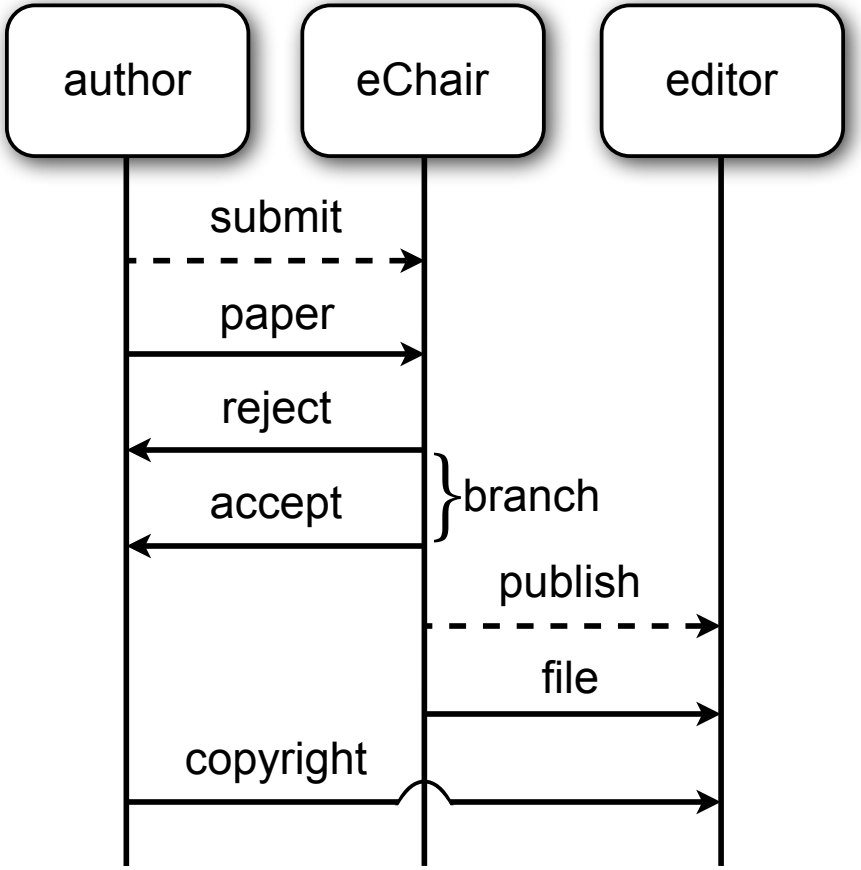
- Events (*channel.*label.$(x)\Gamma$) are synchronizations in labeled channels passing channel references

  Each event has associated the ordering admissible $(x)\Gamma$ for the channel which is to be passed in the message

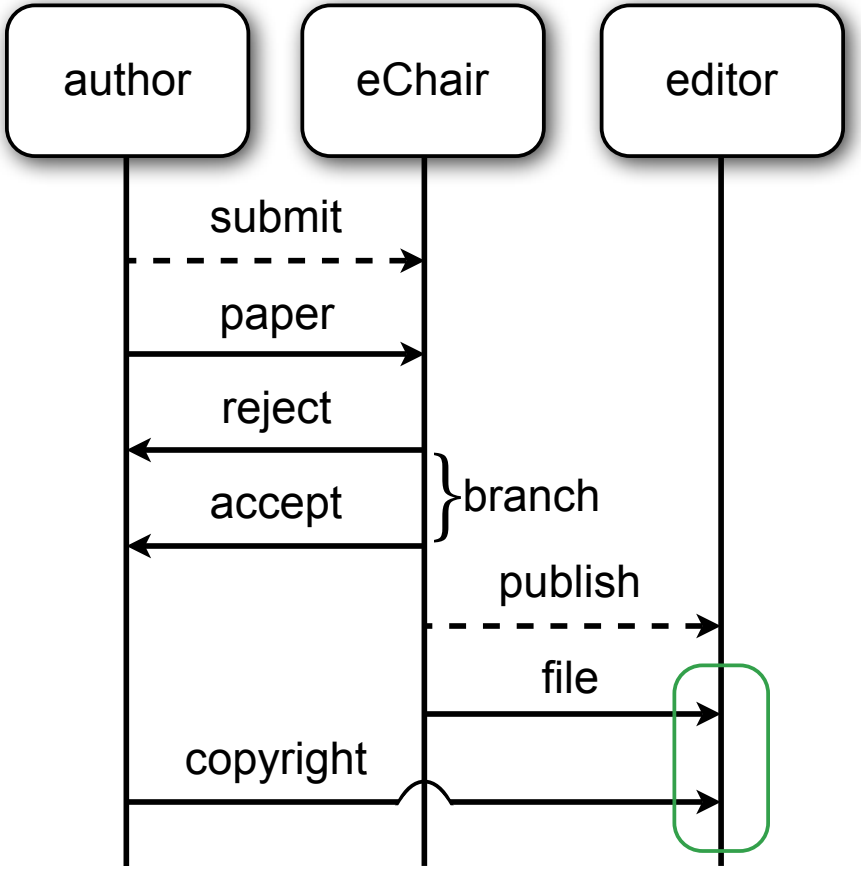
  received/sent channels must comply with the prescribed order

Ordering eChair System Events

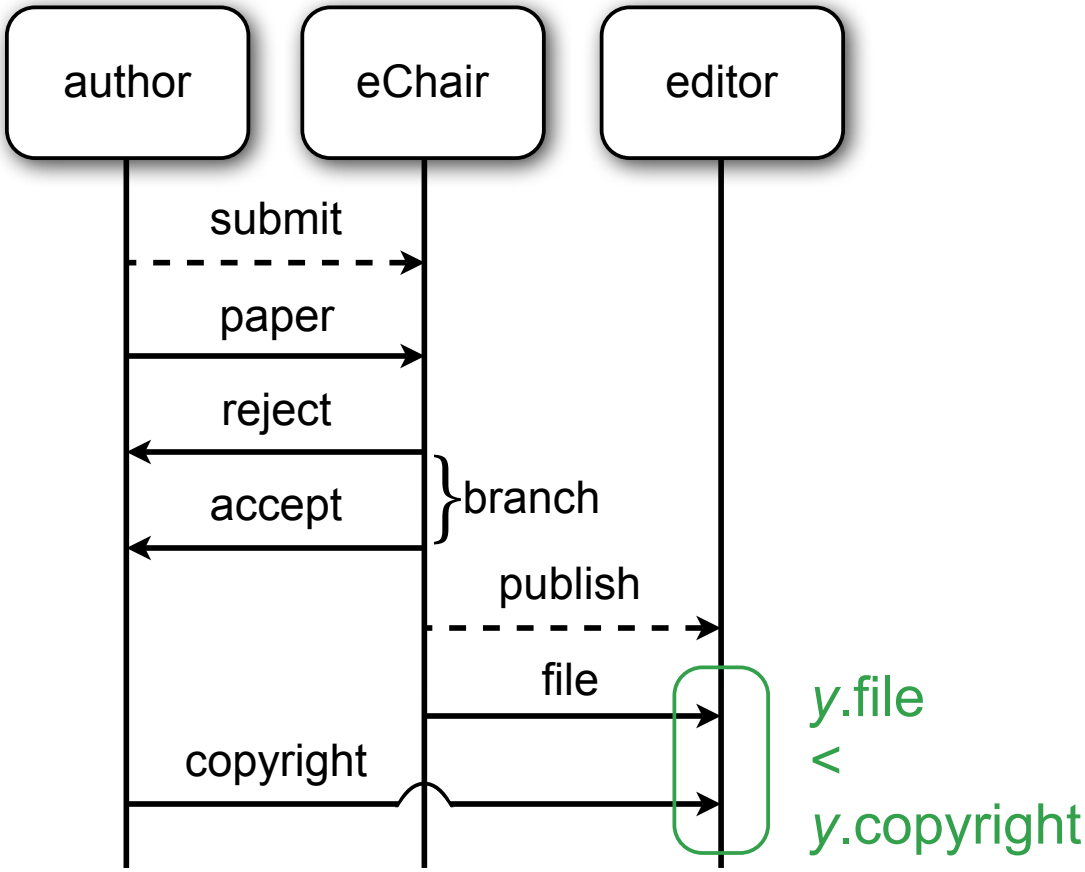
Ordering eChair System Events



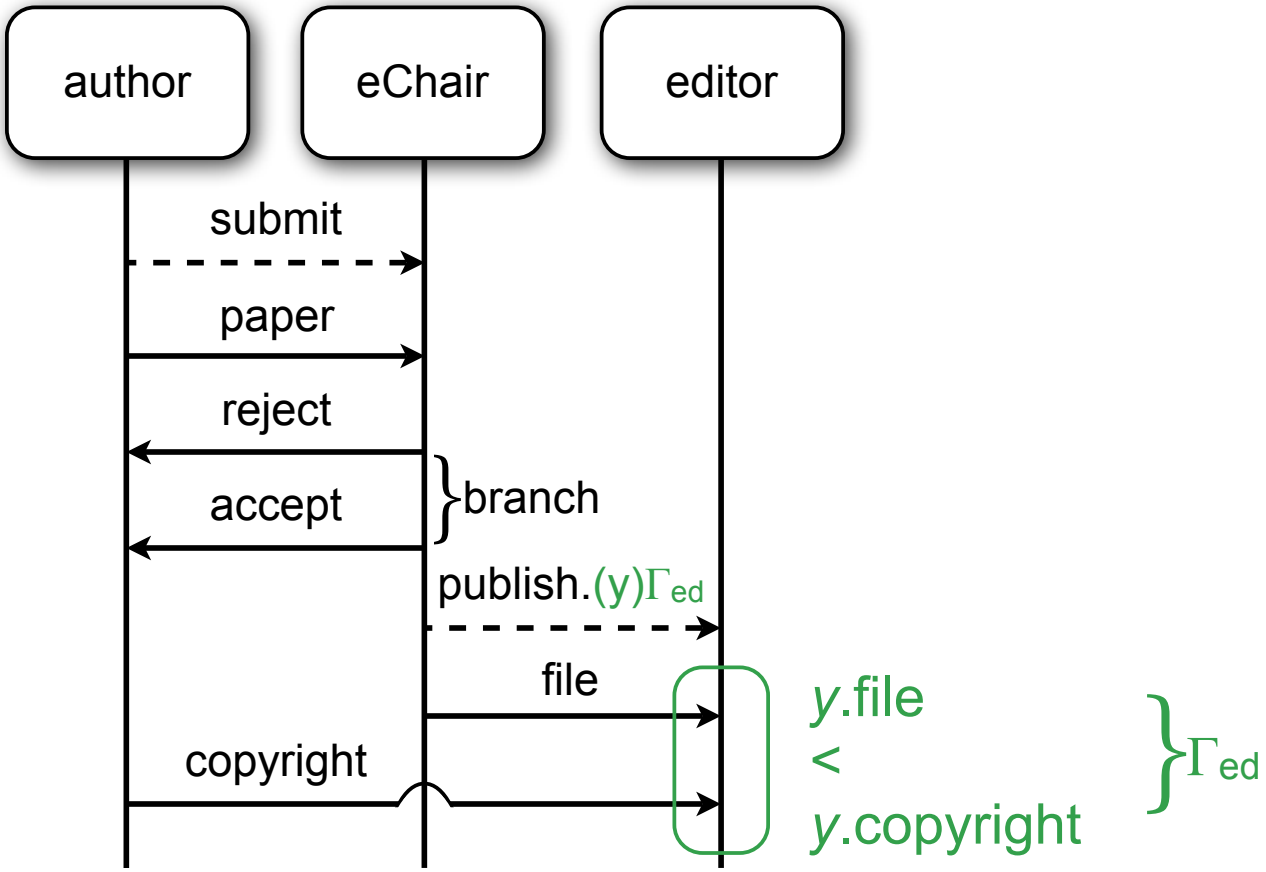
Ordering eChair System Events



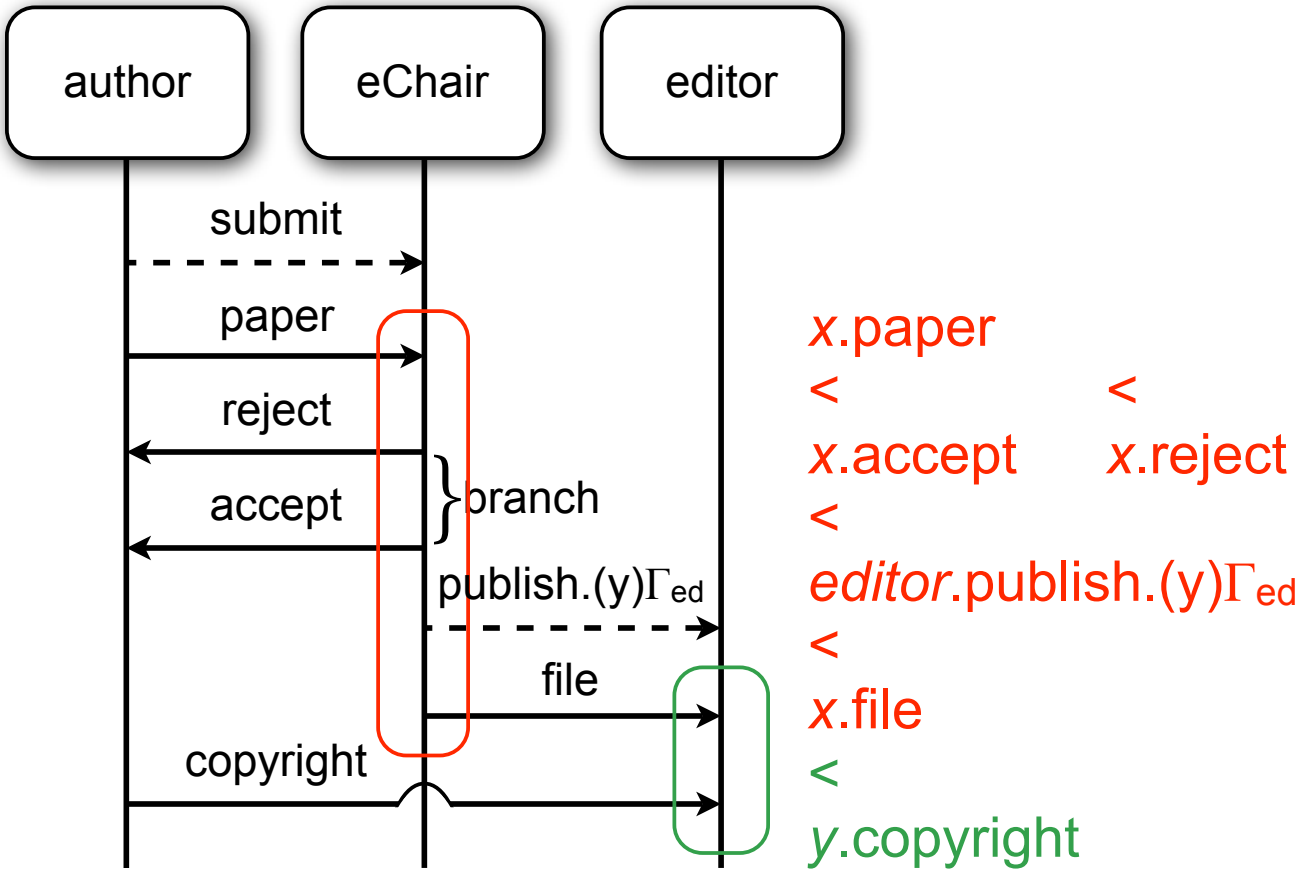
Ordering eChair System Events



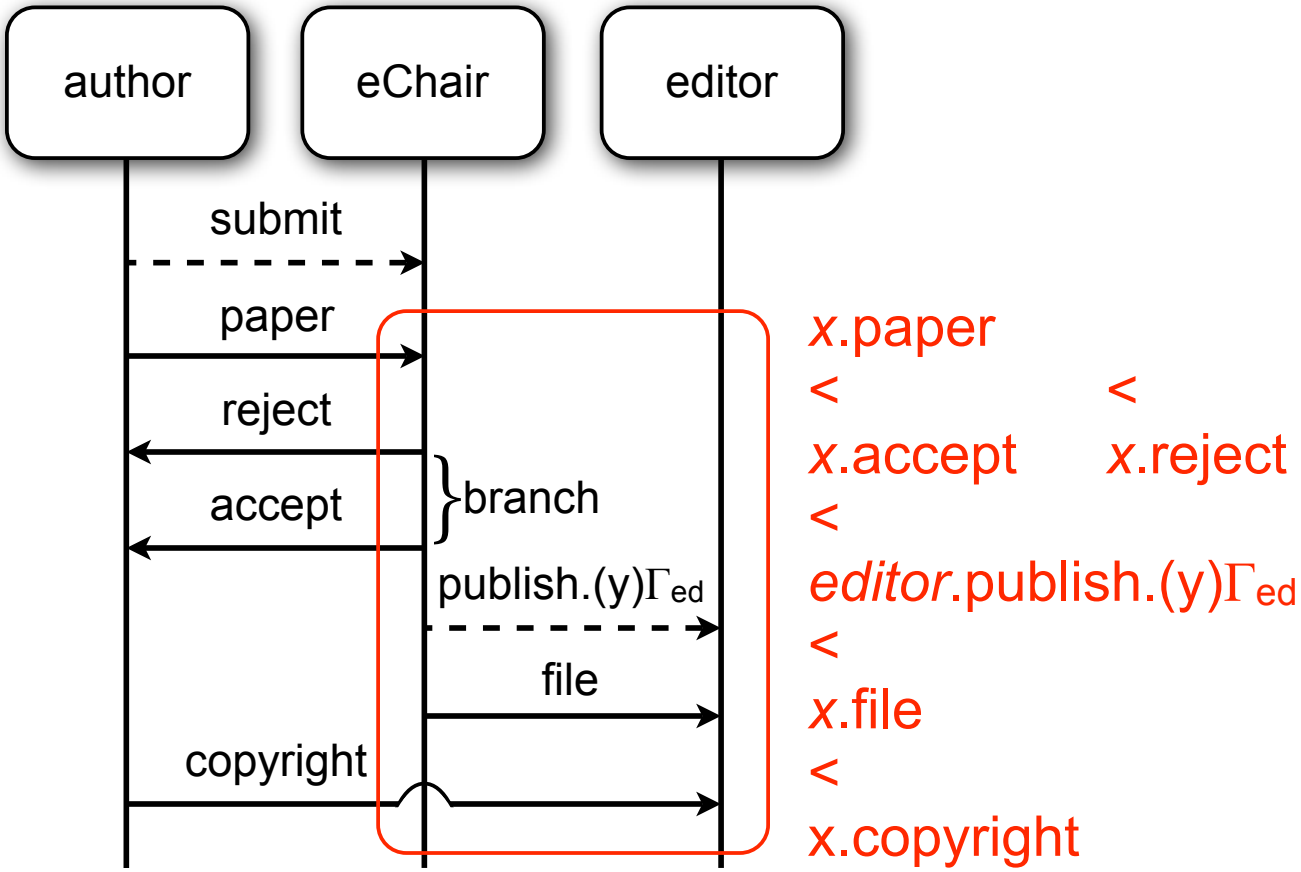
Ordering eChair System Events



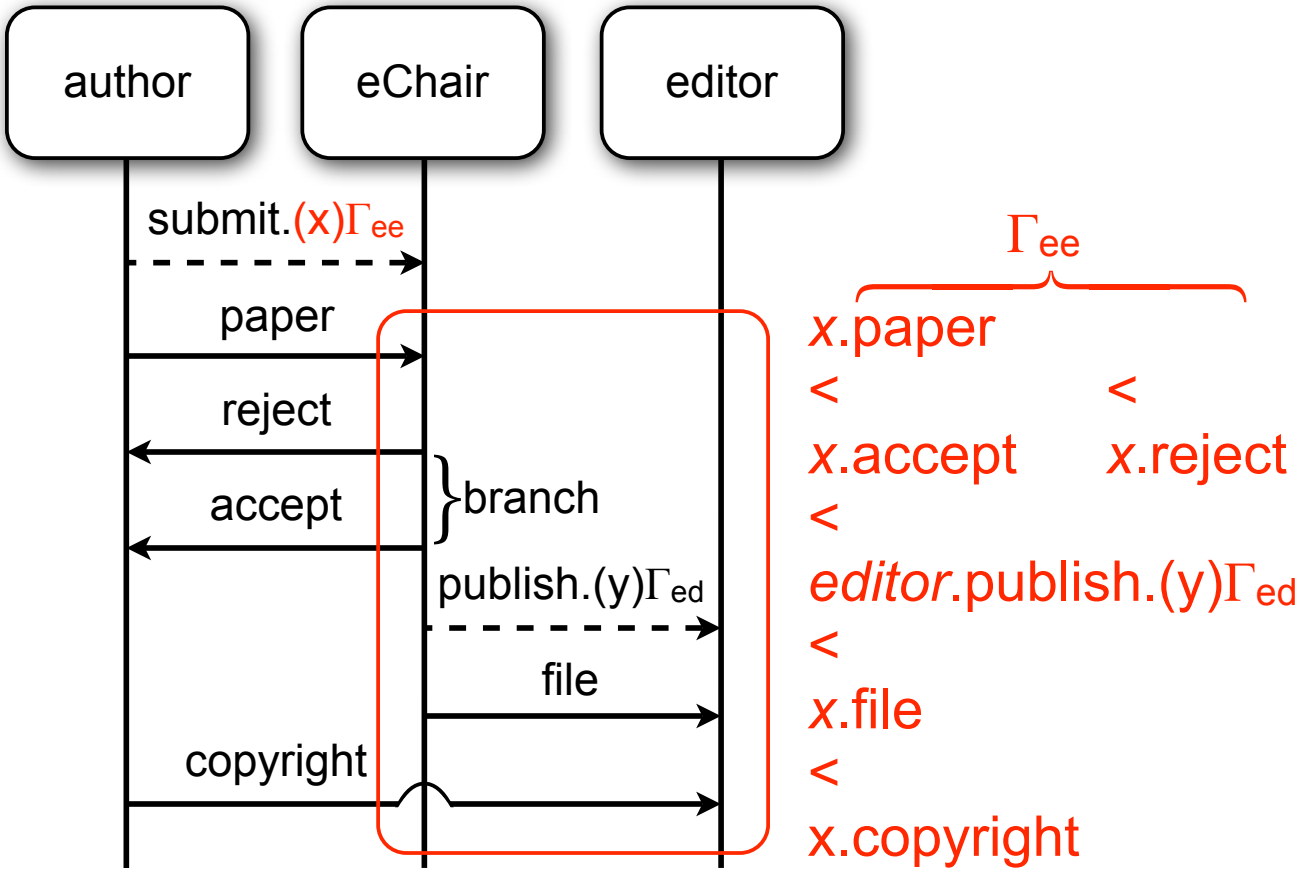
Ordering eChair System Events



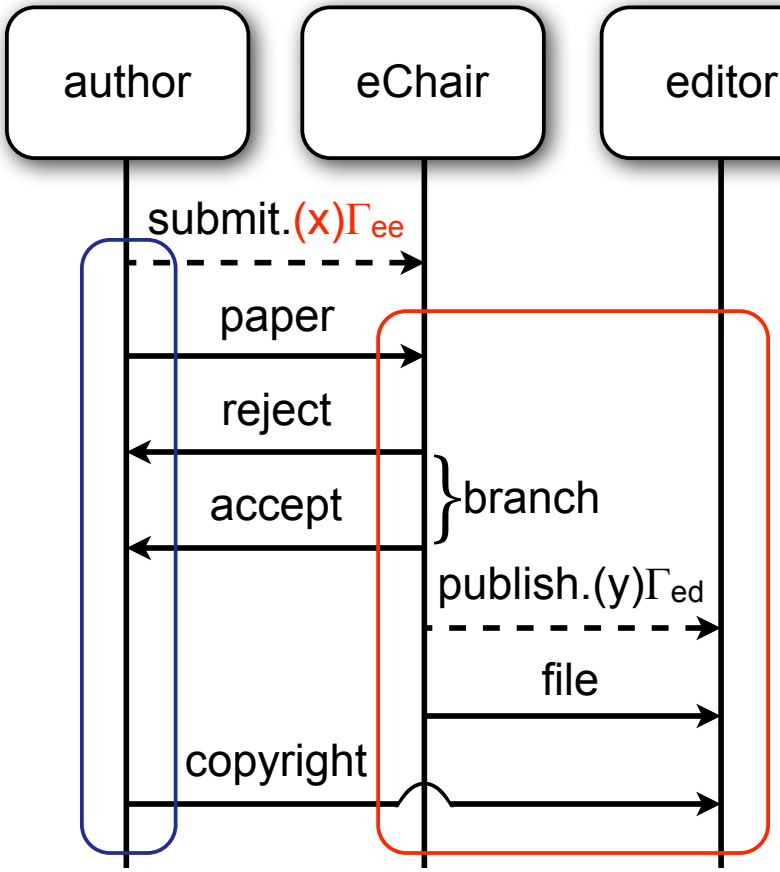
Ordering eChair System Events



Ordering eChair System Events

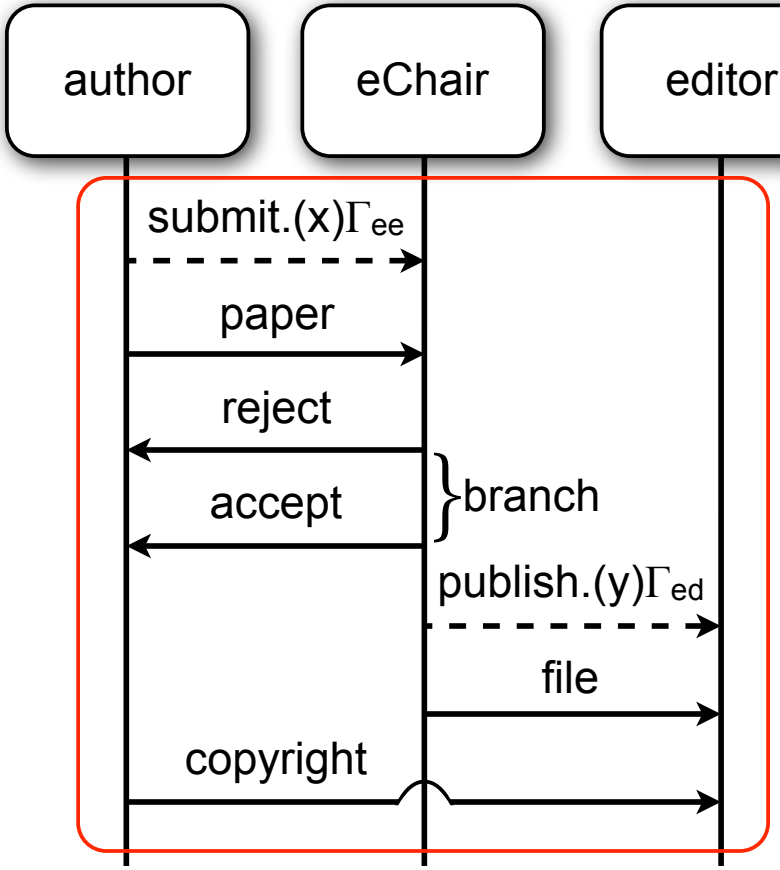


Ordering eChair System Events



eChair.submit.(*x*)Γee
<
chat.paper
<
chat.accept *chat*.reject
<
editor.publish.(y)Γed
<
chat.file
<
chat.copyright

Ordering eChair System Events

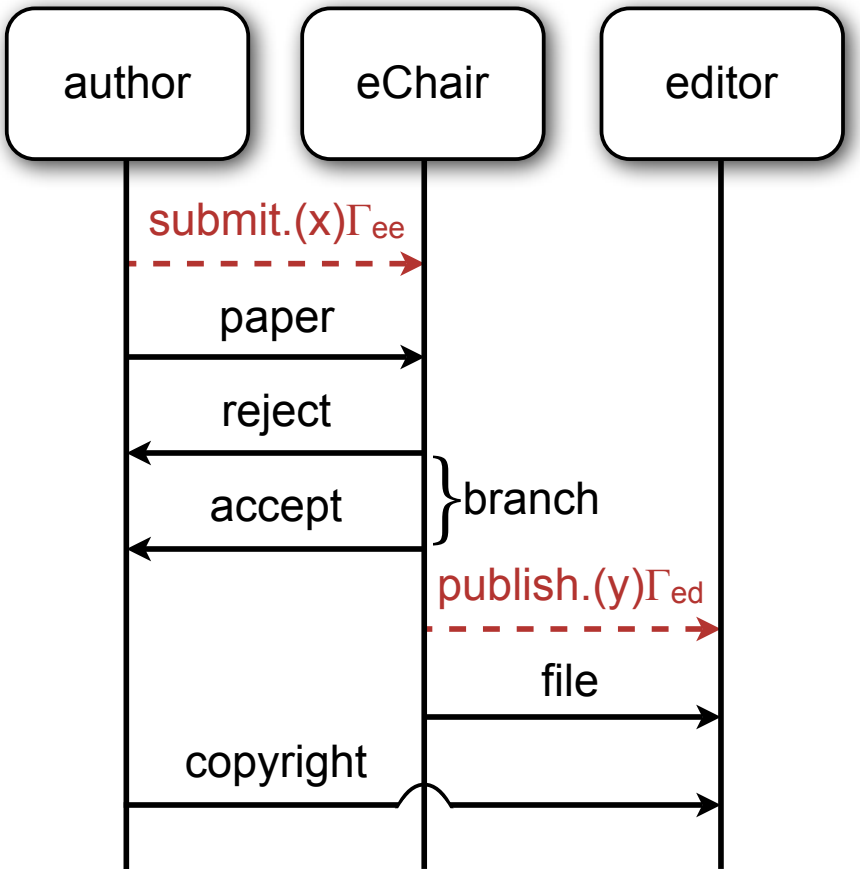


eChair.submit.(*x*)Γee
<
chat.paper
<
chat.accept *chat*.reject
<
editor.publish.(y)Γed
<
chat.file
<
chat.copyright

Typing eChair System

$eChair.\mathrm{submit}.(x)\Gamma_{ee}$
$< editor.\mathrm{publish}.(y)\Gamma_{ed}$
$\vdash$

(**ν***chat*)
(*eChair* • submit!(*chat*).
  chat • paper!(*pdf*).
  (*chat* • reject?()
   +
   chat • accept?(). *chat* • copyright!()))
|
eChair • submit?(*x*).
  x • paper?(*pdf*).
  (*x* • reject!()
   +
   x • accept!().
    editor • publish!(*x*). *x* • file!(*pdf*))
|
editor • publish?(*y*).
  y • file?(*pdf*). *y* • copyright?()



Results

Results

Theorem (Preservation of Event Ordering)

Let P be well-formed and $\Gamma;\Delta \vdash P$. If $P \rightarrow Q$ then $\Gamma';\Delta,\Delta' \vdash Q$.

Theorem (Lock Freeness)

Let P be a process s.t. $P :: T$ and $\Gamma;\Delta \vdash P$. If *closed*(T) and P is not a *finished* process then there is Q such that $P \rightarrow Q$.

A type T is *closed* if (roughly) $T = \tau(T)$.

Finished processes only exhibit shared inputs (e.g., persistent services).

Corollary (Progress)

Let P be a process s.t. $P :: T$ and $\Gamma;\emptyset \vdash P$ and *closed*(T).

If $P \rightarrow^* Q$ and Q is not a finished process then $Q \rightarrow R$.

Related Work

- Multiparty Sessions (HondaYoshidaCarbone08)

  Created via multicast atomic service requests, each having multiple channels; support a constant number of participants; global types specify who does what.

  In our approach, a single medium supports a dynamic number of multiple participants, interacting via labeled messages.

- Progress in Multiparty Sessions (Bettini et al.08)

  Builds on (HondaYoshidaCarbone08) improving on the progress result;

  Does not address the interleaving of delegated sessions.

- Dynamic Multirole Sessions (DeniélouYoshida11)

  Multicast atomic service request, multiple session channels; Dynamic number of participants, constant number of roles.

  Our approach addresses systems with dynamic “roles”

Concluding Remarks

- Conversation types

  Simple extension of the session type notion

  Multiparty conversations in a single communication medium

  Unify local and global behavioral specs at the same level

  Unantecipated participants may join / leave a conversation, as long as the “projection invariant” is preserved

  $$P_1 \bowtie P_2 \bowtie \ldots \bowtie P_k = G_\tau$$

  Progress result on systems where processes interact in multiple conversations, supporting repeated accesses to several interleaved, possibly delegated, conversations

- Ongoing work

  Analysis of role based conversation using conversation types