Async Processing Language

비동기 메세지 처리를 서술하고 검증하기 위한 텍스트 기반 언어를 정리한다.

아래 모색은 괜찮다. 하지만, 파이 계산 등에서 이미 해결한 문제로 보인다. 살피는 게 필요하다. 시간이 들어가야 한다. Linear Temporal Logic 같은 시간 처리 개념을 포함하도록 한다.

BNF

```
channel ::= identifier | identifers

identifiers ::= ( identifier {, identifier}+ )

chain ::= "chain" identifier comm-links

comm-links ::= comm | comm >> comm-links | comm && comm-links

comm ::= [alias "=" ] comm-single | [alias "=" ] "{" comm-single+ "}"

comm-single ::=  [process] send | recv [process]

alias ::= identifier

send ::= channel "!" message

recv ::= channel "?" message

process ::= command state

state ::= { assertion* }

assertion ::= true-expr

command ::= desc | pseudo-code

desc ::= { text }

pseudo-code ::= { code }
```

예시: 로그인 초간단 흐름

```
chain login_flow
{
    c1 = { "input from user" } login ! msg_req_user_login
    >>
    c2 = login ? msg_req_user_login { "login 대기 목록에 추가" } { waiting_users.has(m.user) } &&
    c2.1 = after 10s generate timeout message with a lambda action that has the user info &&
    c3 = db ! msg_req_user_info
```

```
    >>
    c4 = load_req_recv = db ? msg_req_user_info { "db에서 로딩" } &&
    c5 = login ! msg_res_user_info
    >>
    c6 = login ? msg_res_user_info &&
    c7 = client ! msg_res_user_login
}
```

alias는 각 메세지 송수신에 대한 설명을 위해 사용.

c4.comm_time, processing_time 등 속성에 접근 가능하게 한다. 위와 같이 성질을 줄 수 있으면 더 많은 이해를 할 수 있다.

group으로 묶인 부분은 동시에 처리된다. >>는 비동기로 처리된다.

위에서 waiting_users에 대한 타임아웃 처리가 필요하다는 걸 알 수 있다. 이를 어떻게 처리할 것인가? 지금까지 코딩을 통해 처리했다.

예시 : 사용자 로그아웃을 전체 서버에 통지

```
chain logout_notify {
    (z1, z2, ..., zn) ! msg_ntf_user_logout
    >>
    recvs = {
        (z1, ..., zn) ? msg_ntf_user_logout
    }
}
```

월드 서버와 연결이 일시적으로 끊어지고 다시 붙은 존 서버가 있다고 하자. 그러면 해당 서버에서 로그아웃 메세지를 못 받았을 가능성이 있는가? 클라이언트와 서버 간 연결은 하나만 있어야 한다는 점을 잘 보여준다.

일시적으로 두 개의 연결을 갖고 two-phase 커밋처럼 서버를 이동하면 어떻게 될까? 완전한가?

이것도 시간 등을 사용하여 검증할 수 있다.

의미

응용

자동화된 타임아웃 처리

메세지로 처리한다는 원칙 하에 어떻게 자동화할 것인가? 재미있고 검증 가능한 정도로 APL을 끌어 올린다.

- chain instance id (cid)
    - chain 별로 고유한 아이디를 갖는다.
    - 이를 메세지의 continuation 정보로 갖는다.

과제들:

- 어디에서 timeout을 등록할 것인가?
    - 채널이 대상이다.
    - actor나 service가 된다
    - 메세지를 받을 때 확인한다
- 클라이언트로는 전송되지 않아야 한다
    - msg_server_base와 같은 것?
    - 다른 방법은 없는가?
    - 각 핸들러에서 처리?

message 중심의 처리를 연결한 chain을 개념화한 것만으로도 위와 같은 문제에 대한 해결책을 생각할 수 있다. 일단, 이것만으로도 만족 스럽다.

일시적인 단선의 처리

하나의 체인에 있는 채널들의 순서가 $c_1, c_2, \ldots, c_n$ 이라고 할 때, c_k가 동작을 멈추면 두 가지 경우가 있다.

1) c_{k-1} 이 c_k의 단선을 알고 있을 경우

2) c_{k-1}이 c_k의 단선을 모르고 있을 경우

1)의 경우는 해당 채널에 접근하는 통신을 할 때 에러로 처리해야 한다. 이를 채널 에러라고 할 수 있다. 채널 에러를 감지할 경우 채널 에러 처리를 한다.

2)의 경우 c_k의 단선이 진행 중에 c_{k-1}에서 채널에 접근하는 경우이다. 몇 가지 경우들을 나눠서 살펴야 한다.

체인에 있는 노드들에 응답이 있는 경우, 즉, $j > i, c_j = c_i$ 일 때 응답을 받는 노드들은 타임아웃에 따른 에러 처리를 해야 한다.

모든 체인에 대해 단선 처리를 한다는 것은 물리적으로 불가능하다. 이를 처리할 일관되고 전체적인 방법이 있어야 한다.

- 단선은 무조건 장애로 처리한다
- 단선과 연결을 메세지로 본다
- 메세지의 시간이 중요하다