sipoks05 – Core Konzept

Statemachine-Realisierung

Enrico Hartung
Matthias Liebig

07.07.05
Update 13.07.05

Core: Event Dispatcher

- event_dispatch(char* event, void** params)
  - event: “GUI.MAKE_CALL”, “SIPLISTENER.INCOMING_CALL”, “SIPLISTENER.THROW”, ..
  - params: die jeweiligen Parameter (wie im Klassendiagramm) in einem Array
  - dient der Verteilung von Events und dem Starten von Statemachines

Statemachine starten

- Statemachine für EventListener wird mit Programmstart in einem Thread gestartet
  - eigene Queue (s.u.) und condition variable
- Beim Event “GUI.MAKE_CALL” oder “SIPLISTENER.INCOMING_CALL” wird die Haupt-SM gestartet
  - eine Queue wird angelegt; Queue = Array von struct mit event und params

Statemachine starten

- Queue wird mit dem aktuellen Event gefüllt
- eine pthread condition variable wird angelegt
- Thread für die “normale” Statemachine und das für den Eventlistener bekommen ihre jeweilige Event-Queue-ID und die conditional variable übergeben
- CallID wird generiert und mit Bezug zu der conditional variable gespeichert
- CallID wird zurückgegeben

Statemachine-Threads

- State wird als lokale Variable gespeichert (enum { CHECKING, INVITING, ... } state;)
- Einfache case-Fallunterscheidung wenn Transitionen geprüft werden:
  case *state*: switch (*trigger*)
- Wenn ein Event abgearbeitet ist, wird pthread_cond_wait aufgerufen

Event Weiterleitung

- Event landet durch GUI oder SipListener bei Event Dispatcher
- Event wird auf die richtige Queue gepackt
  - Auswahl ob für EventListener oder “normale” Statemachine
  - Bei letzterem: Auswahl nach CallID
- Entsprechendes Thread wird über pthread_cond_signal aufgeweckt

Event Weiterleitung

- Zweifaches Aufwecken wird ausgeschlossen
  - Nur möglich wenn das Thread wartet, d.h. eine Transition wird nicht unterbrochen
  - Falls Thread arbeitet, wird gewartet bis das Thread inaktiv ist, und dann vorsichtshalber wieder geweckt (falls Queueveränderung unbemerkt blieb)

Event Queue

- Thread wacht auf: Queue anschauen
  - Event passt zu einer möglichen Transition
    - eventuell Aktion ausführen und Event von der Queue löschen
    - State wechseln
    - Queue anschauen
  - Event passt nicht
    - Event von der Queue löschen
    - Queue anschauen

Event Queue

  - Queue ist leer
    - auf neues Event warten
- Queue wird über Mutex (Lock) vor konkurrierenden Zugriffen geschützt

SipListener

- Wandelt Funktionsaufrufe in Events um
- z.B. incomingCall() ruft event_dispatch (“SIPLISTENER.INCOMING_CALL”, ...) auf