Teknillinen korkeakoulu
Tietoliikenneohjelmistojen ja multimedian laboratorio
TOVE-projekti

SSCOP-protokollan toteuttaminen

20.3.1997

Harri Sunila, Pasi Nummisalo, Vesa-Matti Puro, Jari Pusa, Timo Pärnänen

Protokollan toteuttaminen

Sisältö:

Lyhennelmä

0. Johdanto

1. SSCOP-protokollan kuvaus
1.1 SSCOP:n ominaisuudet
1.2 SSCOP:n implementointi
1.3 SSCOP conduitteina

2. SSCOP:n Protocol-conduit
2.1 Protocol-conduitin rakenne
2.2 Protocol-conduitin toteutus
2.2.1 Muodostimet ja tuhoaja
2.2.2 AA- ja MAA-primitiivien lähetys
2.2.3 PDU:den käsittely
2.2.4 Protokollan parametrien käsittely

3. SSCOP:n tilakone
3.1 Tilakoneen kantaluokka
3.2 Tilaluokat

4. Viestien toteutus
4.1 PDU:t
4.2 Sisäiset viestit
4.3 Ajastimien viestit

5. Toteutuksen puutteet
5.1. Puuttuvat ominaisuudet
5.2. Osittain toteutetut ominaisuudet
5.3. Virheet
5.4. Jatkokehitys

Lyhennelmä

Tietoliikenneprotokollan toteuttaminen on vaativa tehtävä. Protokolla on hyvin tarkasti määritelty, jotta se toimisi johdonmukaisesti tilanteessa kuin tilanteessa, eikä aiheuttaisi suuria yllätyksiä. Ohjelmoijien työtä ja varsinaisen protokollan ohjelmointia on helpotettu kehittämällä runkoja, framework:eja, joita käyttämällä protokollan toteuttaminen helpottuu. Eräs tällainen on OVOPS++, jota käyttäen on toteutettu useita protokollia, muun muassa Service-Specified Connection-Oriented Protocol, SSCOP.

OVOPS++:aa käyttäen voidaan protokolla koota hyvinmääritellyistä rakennuspalikoista, jolloin toteutus on usein hyvinkin suoraviivaista. OVOPS++:a käytettäessä framework:in luokista peritään protokollan luokat, joiden toiminnallisuus täydennetään framework:in pohjalta.

SSCOP:n rungon muodostaa yksi protokolla-conduit, joka peritään framework:in pfProtocol-luokasta. Johdettuun luokkaan lisätään protokollaspesifiset yksityiskohdat, ja oletustoteutuksia muokataan sopivasti

Protokollan varsinaisen toiminnallisuuden määrittelevä tilakone toteutetaan perimällä framework:sta pfState-luokasta protokollaspesifinen kantaluokka. Tästä kantaluokasta peritään jokaista tilakoneen tilaa vastaava luokka, jossa toteutetaan toiminnallisuus. Johdettuja luokkia joudutaan muokkaamaan protokollan toiminnan mukaan, mutta kuitenkin rakenne on varsin selkeä.

Protokollan käsittelemiin viesteihin käytetään frameworkissa tätä varten olevia luokkia. Frameworkin pfMessenger-, pfTimerMessenger- ja pfMsgTransporter-luokista peritään luokat viestejä varten. Näin saadaan toteutettua PDU:t, primitiivit ja muut tarvittaavat viestit.

Framework:in käyttö protokollan toteutuksen yhteydessä helpottaa itse toteuttamista, vaikka toisaalta sen opettelu saattaa olla suurikin urakka. Kun frameworkin käytön on oppinut, on protokollien toteutus helpompaa ja nopeampaa kuin jos niitä alkaisi koodata alusta pitäen.

0. Johdanto

Nykyaikaisessa tietoliikenteessä tietoliikenneprotokollat ovat usein kerrosrakenteisia perustuen Open Systems Interconnection- eli OSI-malliin tai muuhun, ehkä yksinkertaisempaan malliin. Protokollien toteuttamista varten on kehitetty menetelmiä, jotka helpottavat niiden toteuttamista.

Protokollien toteuttamisessa käytettävien menetelmien tulee olla helppokäyttöisiä ja kattavia, jotta yhdellä menetelmällä voidaan toteuttaa useita protokollia. Menetelmän kattavuus tarkoittaa sitä, että sen avulla on mahdollista toteuttaa rakenteeltaan erilaisia protokollia. Ohjelmoijan kannalta menetelmän tulee olla havainnollinen, jotta ohjelmoija sisäistää kyseisen menetelmän ja pystyy soveltamaan sitä toteuttaessaan protokollaa.

Usein käytettyjä menetelmiä protokollien toteutuksessa ovat erilaiset sovellusrungot, framework:it. Framework:ien ideana on koota sovelluksissa, tässä tapauksessa protokollissa, usein esiintyvistä samankaltaisuuksista eräänlainen perusta, jonka pohjalta ja jota täydentämällä saa usein helpostikin aikaiseksi valmiin protokollan. Vaikka sovellusrungon hahmottaminen saattaa ollakin suuri urakka, sen kerran hahmotettuaan saattaa ohjelmoija havaita protokollan ohjelmoimisen varsin suoraviivaiseksi toiminnaksi.

Tässä dokumentissa kerrotaan, kuinka OVOPS++:aa käyttäen toteutetaan todellinen protokolla, josta esimerkkinä on ATM-verkoissa käytettävä SSCOP, Service-Specific Connection-Oriented Protocol.

1. SSCOP-protokollan kuvaus

1.1 SSCOP:n ominaisuudet

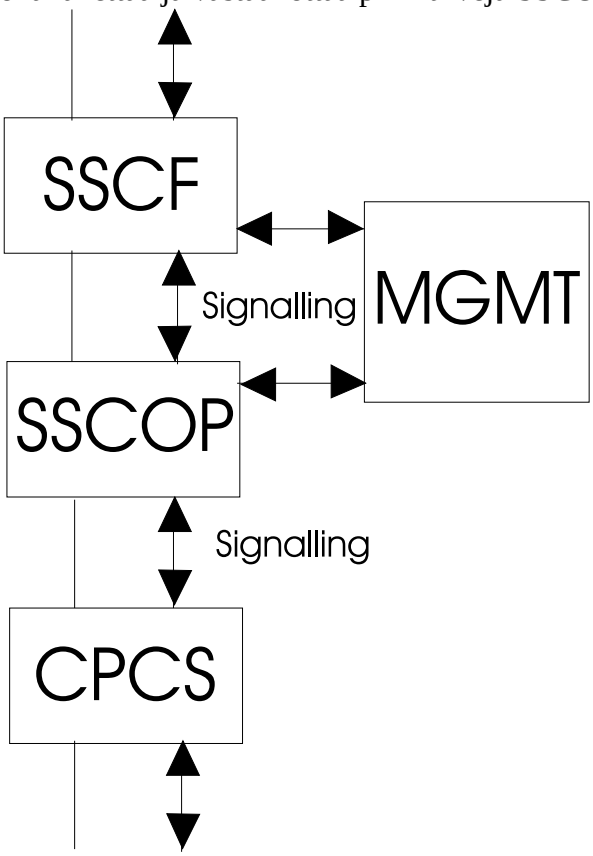
SSCOP on protokolla, jota käytetään ATM-verkossa ATM Adaption Layerissä (AAL). AAL on suunniteltu laajentamaan ATM-kerroksen tarjoamia palveluita, jotta seuraava ylempi kerros saa tarvitsemansa palvelut. Erilaisia palveluita varten on tehty useita AAL:eja, ja signalointia varten on kehitetty Signalling ATM Adaption Layer, SAAL.

SSCOP on suunniteltu tarjoamaan ne toiminnallisuudet, jota SAAL:ssa tarvitaan. SAAL muodostuu kahdesta alatasosta: yleisestä ja palvelukohtaisesta osasta. Palvelukohtainen osa tunnetaan Service-Specific Convergence Sub-layer:na, SSCS:nä, joka puolestaan muodostuu Service-Specific Coordination Function:ista, SSCF:stä ja SSCOP:sta. SSCF:n tehtävänä on sovittaa SSCOP:n palvelut SAAL:n käyttäjän tarpeisiin. Tällainen rakenne antaa eri SSCF:n määrittelemille AAL:lle mahdollisuuden käyttää yleistä yhteydellistä ja virheenkorjaavaa protokollaa yleiseen ja luotettavaan tiedonsiirtoon.

SSCOP mahdollistaa tiedonsiirron samalla taaten siirrettävän datan yhtenäisyyden, eli sen että lähetetyt Service Data Unit:it, SDU:t tulevat oikeassa järjestyksessä perille. Se tarjoaa virheenkorjauksen, joka perustuu uudelleenlähetykseen: vastaanottava SSCOP-

olio havaitsee puuttuvat SDU:t. SSCOP tarjoaa vuonvalvonnan sekä yhteyden valvonnan, eli yhteyden muodostuksen, purkamisen ja uudelleensynkronoinnin. SSCOP havaitsee toimintavirheensä sekä toipuu niistä. SSCOP:n ominaisuuksiin kuuluu, että yhteys kahden olion välillä pysyy kunnossa, vaikka niiden välillä ei olisi tiedonsiirtoa, kunnes yhteys puretaan.

SSCOP sijaitsee protokollapinossa Common Part Convergence Sub-layer:in, CPCS:n päällä. CPCS tarjoaa point-to-point -yhteyden, joten CPCS yhdessä SSCOP:n ja SSCF:n kanssa esittää yhtä signalointilinkkiä. SSCOP:n ja SSCF:n rinnalla on hallintaolio (kuvassa 1 MGMT), joka lähettää ja vastaanottaa primitiivejä SSCOP:ltä



Kuva 1. Osa protokollapinoa

1.2 SSCOP:n implementointi

SSCOP:ssa on toteutettu erikseen varsinainen protokollaolio ja palvelurajapinnat. Protokolla on toteutettu määrittelemällä protocol-conduit ja tähän liittyvä tilakone. Palvelurajapinta ylöspäin on totetutettu AAIF-modulissa, joka vastaa palveluprimitiiveistä SSCOP:n ja SSCF:n välillä. Palvelurajapintana alaspäin SSCOP käyttää CPCS:ää.

1.3 SSCOP conduitteina

SSCOP on rakenteltaan varsin yksinkertainen protokolla. Koska jokaista muodostettavaa signalointilinkkiä kohden on vain yksi SSCOP-protokollaolio, SSCOP:ta toteutettaessa ei tarvita lainkaan multiplekseriä. Koska SSCOP-protokollaolio sijaitsee protokollapinon sisällä, SSCOP:iin ei ole tarpeellista toteuttaa rajapintoja laitteisiin tai sovelluksiin, eikä siinä näinollen tarvita myöskään adaptereita. SSCOP:n muodostaakin vain yksi protocol-conduit ja tähän liittyvät State-luokat. Protokollaolion sisäiset viestit on toteutettu Transporter:ien avulla. Protokollan käsittelemät PDU:t on toteutettu Messenger:einä.

2. SSCOP:n Protocol-conduit

2.1 Protocol-conduitin rakenne

SSCOP:n Protocol-conduit toteutetaan perimällä framework:n pfProtocol luokasta sscopProtocol-luokka. SscopProtocol-luokassa määritellään tilakoneen luokat ystäviksi (friend), jolloin ne pääsevät käsittelemään protokollaolion tilamuuttujia suoraan. Ystävämäärittelyt eivät ole olio-ohjelmoinnin ideologian mukaisia, ja niitä tulisi välttää, mutta toteutuksessa on päädytty käyttämään näitä, jolloin luokkaan ei tarvitse määritellä tilamuuttujille käsittelymetodeja. Luokkaan määritellään luonnollisesti muodostin, kopiomuodostin ja tuhoaja, sekä clone-funktio, jolla olemassaolevasta protokollaoliosta saadaan aikaan kaksoiskappale.

```
class sscopProtocol : public pfProtocol
{
friend class sscopState;   // Base class for state
                           //classes in SSCOP
friend class sscopIdle;    // Derived class for
                           //an actual state
    ...

  public:
    sscopProtocol(void);
    sscopProtocol(const sscopProtocol &other_);
    virtual ~sscopProtocol(void);
    virtual sscopProtocol *clone(void) const;

}
```

Luokan asiakkaiden, eli käytännössä ulkopuolisen hallintaolion halutaan voivan asettaa protokollan parametrit ja ajastimet. Nämä metodit ovat julkisia juuri sen vuoksi, että myös luokan ulkopuolelta pääsee käsittelemään niitä.

```
class sscopProtocol : public pfProtocol
{
 ...
  public:

  ...
```

```
    void setMaxSDUsize(pfUlong SDUsize_);
    ...

    void setManager(SAALlayerManagement *manager_);
...}
```

Protocol-conduitin yksityiseen osaan toteutetaan metodit, joiden avulla se viestii ulkomaailman kanssa. Protokollan pitää pystyä lähettämään signaaleita yläpuolella olevalle AAIF-modulille. Samoin täällä määritellään metodit, joilla protokolla-olio saa lähetettyä primitiivejä hallintaoliolle. Metodit PDU:iden luomiseen, koodaamiseen ja lähettämiseen määritellään myös yksityisiksi. Tämä onkin oliopohjaisen ajattelun perusteella aivan luonnollista, koska näitä metodeja kutsutaan vain luokan sisältä.

```
class sscopProtocol : public pfProtocol
{
 ...
  private:
// Methods to create and send AA-signals upwards to
// conduit side A

    void sendAaESTABLISHind(const pfFrame &sscop_uu_);
    ...

// Methods to send MAA-indications to the layer
// management

    void sendMaaERRORind(sscopError code_, pfUlong
    count_ = 0);
    void sendMaaUNITDATAind(const pfFrame &frame_);

// Methods to create, encode and send SSCOP PDUs to
// the peer protocol

    void sendBGNpdu(pfFrame &frame_);
    ...

// Method to send a frame (encoded PDU) to CPCS
// protocol (on side B)
// used by sendXXpdu() methods

    void sendCpcsUNITDATAreq(const pfFrame &frame_);

// Method to create a "SD PDU Queued Up" internal
// event

    void createSDpduQueuedUp(void);

// Some helpful PDU encoding routines used by
// sendXXpdu() methods
    pfUlong padPDUInformation(pfFrame &frame_);
    ...

...}
```

Protokollan parametrit, tilamuuttujat, ajastimet sekä muuttujia joihin tallennetaan ajastinten kestot ja lähetetyt PDU:t laitetaan yksityiseen osaan. Ajastinten kestot tallennetaan, koska kopiomuodostin tarvitsee niitä, ja PDU:t tallennetaan uudelleenlähetyksen varalta. Lisäksi pitää varata puskurit tulevia ja lähteviä PDU:ita varten, sekä asettaa osoitin hallintaolioon. OVOPS++ tarjoaa myös mahdollisuuden määritellä ajastimet erilliseen pfTimers-moduliin, jossa oleviin ajastimiin on mahdollista viitata ajastimen nimellä. Niinikään parametrit olisi mahdollista määritellä pfStorage-modulissa, jolloin niihin voisi viitata nimellä.

```
class sscopProtocol : public pfProtocol
{
 ...
  private:
  ...
// SSCOP parameters
    pfUlong _maxSDUsize;
    ...
// State variables
    moduloInteger _VT_S;
    ...
// SSCOP timers
    pfTimer _timerCC;
    ...
// Timer timeout values are saved here for use in copy
// constructor
    pfUlong _saveCCtimeout;
    ...
// These are to store some previously sent PDUs for
// resending
    pfFrame _lastBGNframe;
    ...
// Transmitter/receiver buffers and queues
    typedef deque<pfFrame> frameQueue;
    typedef map<pfUlong, sscopSD_PDU *, less<pfUlong>
      pduMap;
    frameQueue _transmissionQueue;
    pduMap _transmissionBuffer;
    ...
// Pointer to the layer management entity of this
// protocol instance

    SAALlayerManagement *_manager;
};
```

2.2 Protocol-conduitin toteutus

2.2.1 Muodostimet ja tuhoaja

sscopProtocol-luokan muodostin yksinkertaisesti alustaa tilamuuttujat, protokollan parametrit, puskurit ja ajastimet, sekä asettaa protokollan alkutilaansa.

```
sscopProtocol :: sscopProtocol(void) : pfProtocol(),
```

```
                    _maxSDUsize(NNI_SSCOP_MAX_SDU_SIZE),
    ...
{
  pfTimerMessenger *messenger = 0;

  _timerCC.setHost(this);        // Initialize timers
  messenger = new sscopCCtimeout;
  assert(messenger != 0);
  _timerCC.setMessenger(messenger);
  setCCtimeout(NNI_SSCOP_TIMER_CC_TIMEOUT);
  ...
  _VR_SQ = 0;         // Initialize state variables
  _clearBuffers = 1;             // ... and parameters

  changeState(sscopIdle::instance()); // Set new state
  return;
}
```

Kopiomuodostin ei eroa muodostimesta paljoakaan. Se kopioi parametrit toisesta sscopProtocol-instanssista ja asettaa uuden instanssin alkutilaan. SscopProtocol-luokan tuhojan toteutus on yksinkertainen: se on tyhjä. Clone()-metodi palauttaa osoittimen uuteen sscopProtocol-olioon, joka muodostetaan kopiomuodostimella oliosta, jolle metodia kutsutaan.

2.2.2 AA- ja MAA-primitiivien lähetys

Metodit, jotka lähettävät AA-primitiivejä, luovat ensin uuden messengerin. Metodit asettavat niiden parametrit ja oletusarvot ja lähettävät ne B-puolella olevalle conduitille.

```
void sscopProtocol :: sendAaESTABLISHind(const pfFrame
                      &sscop_uu_)
{
  aaESTABLISHind *message = new aaESTABLISHind;
  assert(message != 0);
  message->setSSCOP_UU(sscop_uu_);
  toB(message);
  return;
}
```

Samoin toimivat myös metodit, jotka käsittelevät hallintaoliolle lähetettäviä primitiivejä. Koska sscopProtocol-oliossa on vain osoitin hallintaolioon, eikä sitä ole liitetty conduittina Protocol:iin, ei lähetykseen voida käyttää Conduit-mallin mukaisia rajapintametodeja samoin kuin AA-primitiiveille, vaan viesti välitetään perille hallintaolion osoitteen perusteella.

2.2.3 PDU:den käsittely

PDU:den lähettämistä varten olevat metodit koodaavat PDU:t frameihin ja lähettävät PDU:t edelleen SSCOP:n vastinoliolle. Lähetykseen käytetään CPCS-protokollaa, joka on sscopProtocol-conduitin A-puolella. PDU koodataan tietyn

muotoiseksi. Joitain PDU:ita pitää mahdollisesti lähettää uudelleen, joten niitä varten on omat metodit, ja tällaiset PDU:t talletetaan niitä varten varattuihin muuttujiin.

```
void sscopProtocol :: sendBGNpdu(pfFrame &frame_)
{
  _lastBGNsscop_uu = frame_;
  pfUlong pads = padPDUInformation(frame_);
  appendReserved(frame_, 3);
  append8bitInt(frame_, _VT_SQ);
  appendTypeOctet(frame_, sscopBGNpdu, pads);
  append24bitInt(frame_, _VR_MR);
  _lastBGNframe = frame_;
  sendCpcsUNITDATAreq(frame_);
  return;
}


void sscopProtocol :: resendLastBGNpdu(void)
{
  sendCpcsUNITDATAreq(_lastBGNframe);
  return;
}
```

Koodissa esiintyvä sendCpcsUNITDATAreq()-metodi lähettää frame:n conduitin A-puolella olevalle CPCS-protokollaoliolle, joten se on toteutukseltaan vastaava kuin AA-primitiivien käsittelyn hoitavat metodit. Muut metodit hoitavat PDU:n koodauksen.

Protokollan sisäistä tapahtumaa "SD PDU Queued Up", varten on metodi, joka luo tämän tapahtuman ja laittaa sen protokollan viestijonoon.

```
void sscopProtocol :: createSDpduQueuedUp(void)
{
  sscopSDpduQueuedUp *messenger = new sscopSDpduQueuedUp;
  assert(messenger != 0);
  accept(messenger);
  return;
}
```

2.2.4 Protokollan parametrien käsittely

Protokollan parametrien käsittelyä varten on joukko metodeja, jotka yksinkertaisesti asettavat argumenttinsa parametrin arvoksi. Nämä metodit ovat sitä varten, että SSCOP:n ulkopuolelta voidaan asettaa parametrien arvot. State-luokat pääsevät suoraan käsiksi tilamuuttujiin, joten tilamuuttujia varten ei ole käsittelymetodeja. Mikäli tilaluokkia ei olisi määritelty ystäviksi, pitäisi kyseisten metodien olla julkisia.

```
void sscopProtocol :: setMaxSDUsize(pfUlong SDUsize_)
{
  _maxSDUsize = SDUsize_;
```

```
  return;
}
```

Ajastimien kestot pitää myös voida asettaa protokollan ulkopuolelta. Tätä varten on omat käsittelymetodit, jotka asettavat ajastimien pituudet. Ajastimien pituudet ovat parametreja, joten niiden asettaminen on käytännössä samanlaista kuin muidenkin parametrien. Ajastimien käynnistäminen tapahtuu aina tilakoneen sisältä käsin.

```
void sscopProtocol :: setCCtimeout(pfUlong msec_)
{
  pfUlong usec = (msec_ % 1000) * 1000;
  pfUlong sec = (msec_ - (usec / 1000)) / 1000;
  _timerCC.setTimeout(sec, usec);
  _saveCCtimeout = msec_;
  return;
}
```

3. SSCOP:n tilakone

3.1 Tilakoneen kantaluokka

Protokollan varsinainen toiminnallisuus toteutetaan tilakoneessa. SSCOP:n tilakone on toteutettu perimällä pfState:sta tilakoneen kaikille tiloille yhteinen kantaluokka sscopState. SSCOP:n tilakoneessa on kymmenen tilaa, joita kaikkia varten on oma luokka. Kantaluokka perii myös metodit, joilla se käsittelee protokollaan tulevat palveluprimitiivit. Nämä metodit määritellään luokissa aaDownInputs ja cpcsUpInputs. Kantaluokassa on metodit käsitellä protokollaan tulevia PDU:t. Metodit määrittelevät tilakoneen toiminnallisuuden, eli tilasiirtymät ja protokollaolion jäsenmuuttujien asetukset. Metodit määritellään virtuaalisiksi, ja niille annetaan oletustoteutus, jota muutetaan johdetussa luokassa sen mukaan, miten tilakoneen on suunniteltu toimivan.

```
class sscopState : public pfState, public
                   aaDownInputs, public cpcsUpInputs
{
  public:
// Actions for incoming AA-signals from above layer
    virtual void aaESTABLISHreqAct(aaESTABLISHreq
            *messenger_, pfProtocol *protocol_) const;
    ...

// Actions for incoming CPCS-signals from lower layer
    virtual void cpcsUNITDATAindAct(cpcsUNITDATAind
            *messenger_, pfProtocol *protocol_) const;
    ...

// Actions for incoming MAA-primitives from management
// plane
    virtual void maaUNITDATAreqAct(maaUNITDATAreq
```

```
                *messenger_, pfProtocol *protocol_) const;
        ...

    // Actions for incoming PDUs
        virtual void sscopBGNpduAct(sscopBGN_PDU
          *messenger_, sscopProtocol *protocol_) const;
        ...

    }
```

Kantaluokassa määritellään myös joukko metodeja, jotka käsittelevät protokollan sisäiset tapahtumat ja toiminnat ajastimien lauetessa. Metodit ovat niinikään virtuaalisia, koska tapahtumat ovat luonnollisesti riippuvia protokollan senhetkisestä tilasta, ja ne toteutetaan yksityiskohtaisesti joka tilaluokassa. Virtuaalimetodeille annetaan hyvin yksinkertainen toteutus, tyypillisesti SSCOP:tä totetutettaessa nämä virtuaalimetodit vain kertovat protokollan senhetkisen tilan ja toiminnon.

```
    class sscopState : public pfState, public
                    aaDownInputs, public cpcsUpInputs
    {
      public:
      ...
    // Actions for timer timeouts
        virtual void sscopCCtimeoutAct(sscopProtocol
                                       *protocol_) const;
        ...

    // Actions for internal events
        virtual void sscopSDpduQueuedUpAct(sscopProtocol
                                       *protocol_) const;
        ...
    }
```

Kantaluokkaan määritellään myös muutamia metodeja, jotka toteuttavat protokollan tilakaaviossa määritellyt SDL-makrot., jotka toistuvat säännöllisesti varsinaisissa luokissa. Tällaiset metodit määritellään suojatuiksi, jotta perivä luokka saa ne käyttöönsä. Suojatut jäsenet eivät muuten näy luokan ulkopuolelle.

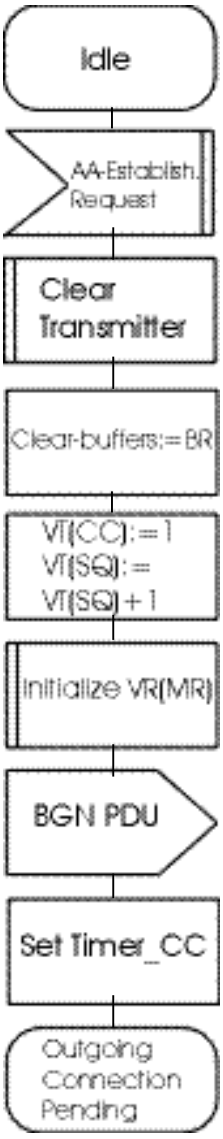
```
    class sscopState : public pfState, public
                    aaDownInputs, public cpcsUpInputs
    {
      ...

      protected:
    // Some SDL macros for state machine implementation
        void releaseBuffers(sscopProtocol *protocol_)
                                                const;
        ...

    }
```

3.2 Tilaluokat

Tilaluokat siis peritään kantaluokasta sscopState. Jokaisesta tilaluokasta halutaan luoda vain yksi instanssi, mikä toteutetaan käyttämällä Singleton-hahmoa (design pattern). Luokkaan toteutetaan toiminnot kaikille mahdollisille syötteille, joita protokolla voi saada ollessaan tässä tilassa. Protokollan toiminnallisuus voidaan havainnollistaa esimerkiksi käyttämällä SDL-kaavioita. Varsinainen ohjelmakoodi on tämän perusteella helposti hahmotettavissa, ja yhtä SDL-kaavion tapahtumaa vastaakin jokseenkin tarkasti rivillinen koodia.




```
void sscopIdle :: aaESTABLISHreqAct(aaESTABLISHreq
            *messenger_, pfProtocol *protocol_) const
{
  TRACE_STATE(protocol_, "State = SSCOP/1, Input =
                                  aaESTABLISHreq");

  sscopProtocol *sscop = (sscopProtocol *) protocol_;
  clearTransmitter(sscop);
  sscop->_clearBuffers = messenger_
                                ->getBufferRelease();
```

```
    sscop->_VT_CC = 1;
    sscop->_VT_SQ++;
    initializeVR_MR(sscop);
    sscop->sendBGNpdu(messenger_->getSSCOP_UU());
    sscop->_timerCC.start();
    changeState(protocol_,
        sscopOutgoingConnectionPending::instance());
    return;
}
```

4. Viestien toteutus

Protokollan viestit on toteutettu käyttämällä framework:in tarjoamia Messenger- ja Transporter-luokkia. PDU:den toteutuksessa on käytetty hyväksi pfMessenger-luokkaa, josta on peritty sscopPDUmessenger-luokka. sscopPDUmessenger-luokasta peritään jokainen todellinen PDU.

SSCOP-protokollaolion sisäiset viestit on toteutettu perimällä frameworkin tarjoamasta pfMsgTransporter-luokasta kutakin viestiä vastaava luokka. pfMsgTransporter on puolestaan johdettu frameworkin Transporter-luokasta, jonka instansseja käytetään kuljettamaan Messenger-olioita conduitgraafissa.

Ajastimien käsittelyssä käytettävät viestit on peritty frameworkin pfTimerMessenger-luokasta, jonka kantaluokkana on pfMessenger-luokka.

4.1 PDU:t

sscopPDUmessenger luokalla on create()-metodi, jolla tulevasta kehyksestä (frame) luodaan oikeanlainen PDU. Lisäksi luokassa on metodit, jotka palauttavat PDU:den yhteiset parametrit. PDU:den parametrit ovat suojattuja jäseniä, joten varsinaiset PDU-luokat käsittelevät niitä ilman erillisiä käsittelymetodeja. sscopPDUmessenger-luokalla on kehyksen koodaamista ja purkua varten yksityiset encode()- ja decode()-metodit, jotka ovat puhtaasti virtuaalisia (pure virtual). Tämän vuoksi nämä metodit on määriteltävä jokaisessa johdetussa PDU-luokassa.

```
class sscopPDUmessenger : public pfMessenger
{
  public:
    sscopPDUmessenger(void);
    sscopPDUmessenger(const sscopPDUmessenger
                                        &other_);
    virtual ~sscopPDUmessenger(void);
    static sscopPDUmessenger *create(const pfFrame
                                        &frame_);
    const sscopPDUmessenger &operator=(const
                          sscopPDUmessenger &other_);
    pfFrame &getInformation(void);
    ...
```

```
  protected:
    pfByte readPadLength(pfByte typeOctet_) const;
    void removeReserved(pfFrame &frame_, pfUlong n_)
                                            const;

    pfFrame _frame;
    pfUlong _N_S;
    pfUlong _N_R;
    pfUlong _N_PS;
    pfUlong _N_SQ;
    pfUlong _N_MR;

  private:
    virtual pfUlong decode(void) = 0;
    pfUlong _errorCode;
};
```

Jokaisessa konkreettisessa PDU-luokassa on toteutettu apply()-metodi, jolla on parametreinä tila ja protokolla. apply()-metodin ansiosta tilakoneen luokat saavat suoritettua kustakin PDU:sta riippuvat toiminnot.

```
void sscopBGN_PDU :: apply(pfState *state_,
                           pfProtocol *protocol_)
{
  dynamic_cast<sscopState *>(state_)->sscopBGNpduAct(
    this, dynamic_cast<sscopProtocol *>(protocol_);
  return;
}
```

decode()-metodi purkaa framesta PDU:n kentät

```
pfUlong sscopBGN_PDU :: decode(void)
{
  pfUlong error = SSCOP_PDU_OK;
  if (_frame.length() < sscopBGNpdu_MINLEN)
  {
    error = SSCOP_PDU_INVALID_LENGTH;
  }
  else
  {
    _N_MR = _frame.getLast24bit();
    pfByte pads = readPadLength(_frame.getLast());
    _N_SQ = (pfUlong) _frame.getLast();
    removeReserved(_frame, 3 + pads);
  }
  return error;
}
```

4.2 Sisäiset viestit

Sisäiset viestit on toteutettu perimällä pfMsgTransporterista oikea viestiluokka. Näillä luokilla on metodinaan atProtocol(), jonka parametreina on osoittimet protocol-olioon ja tilaan. atProtocol() huolehtii siitä, että tilaolion sopivaa metodia kutsutaan. Tämä tapahtuu siten, että MsgTransporter suorittaa apply()-

metodikutsun parametreinaan tila ja protokolla. Tämän perusteella tilakoneen toimintaa ohjataan.

```
void sscopSDpduQueuedUp :: atProtocol(pfProtocol
                        *protocol_, pfState *state_)
{
  dynamic_cast<sscopState *>(state_)->
    sscopSDpduQueuedUpAct(
      dynamic_cast<sscopProtocol *>(protocol_);
    return;
}
```

4.3 Ajastimien viestit

Nämä viestit on toteutettu johtamalla timerMessenger-luokasta sopivat luokat. Luokissa on jäsenenä apply()-metodi, joka on toteutukseltaan samankaltainen PDU-luokissa olevan apply()-metodin kanssa.

5. Toteutuksen puutteet

5.1 Puuttuvat ominaisuudet

Toteutetussa SSCOP:n versiossa ei ole toteutettu MAA-UNITDATA.request:ien käsittelyä. Samoin MD PDU:den käsittelyä ja MD PDU Queued Up -tapahtuman käsittelyä ei ole implementoitu. SD PDU Queued Up -tapahtumien puskurointia ei ole toteutettu tilakoneen tiloissa 7, 8 ja 9. Tilakoneen tilassa 10 ei tarkisteta, onko alempi kerros (CPCS) kiireinen.

5.2 Osittain toteutetut ominaisuudet

Tällä hetkellä VR_MR-tilamuuttujan päivitys ei ehkä ole oikein. VR_MR muuttujaa, joka määrää vastaanottajan ikkunakoon, kasvatetaan aina, kun vastaanotettu PDU on lähetetty SSCF:lle. Tämä on toiminut testeissä, muttei ominaisuutta ole tutkittu vielä kunnolla

Lyhenteet

AA	ATM Adaptation
AAIF	ATM Adaptation InterFace
AAL	ATM Adaptation Layer
ATM	Asynchronous Transfer Mode
CPCS	Common Part Convergence Sublayer
MD	Management Data
OSI	Open Systems Interconnectioin
PDU	Protocol Data Unit
SAAL	Signalling ATM Adaptation Layer
SD	Sequence Data
SDL	Specification and Description Language
SDU	Service Data Unit
SN	Sequence Number
SSCF	Service Specific Coordination Function
SSCOP	Service Specific Connection-Oriented Protocol
STL	Standard Template Library
VR_MR	Maximum acceptable Receiver state Variable

Lähdeluettelo

Hüni - Johnson - Engel: A Framework for Network Protocol Software

Engel - Bieri - Keller: Signalling in ATM Networks: Experiences With an Object-Oriented Solution

ITU-T Recommendations Q.2110, B-ISDN ATM Adaption Layer - Service Specified Connection Oriented Protocol (SSCOP), July 1994

Martikainen - Puro - Räsänen - Pärnänen - Nummisalo - Koponen: TOVE Project 1996 Deliverables

Juhana Räsänen: SSCOP:n lähdekoodi

Juhana Räsänen, haastattelu