VAASAN AMMATTIKORKEAKOULU
VASA YRKESHÖGSKOLA
UNIVERSITY OF APPLIED SCIENCES

Janne Säntti

RAJAPINNAN KEHITTÄMINEN SELAINPOHJAISELLE KASSAJÄRJESTELMÄLLE

Case: Laihia Data Oy

Tekniikka ja liikenne
2013

VAASAN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma

TIIVISTELMÄ

Tekijä	Janne Säntti
Opinnäytetyön nimi	Rajapinnan kehittäminen selainpohjaiselle kassajärjestelmälle
Vuosi	2013
Kieli	suomi
Sivumäärä	45 + 8 liitettä
Ohjaaja	Jukka Matila

Tässä opinnäytetyössä suunniteltiin ja kehitettiin rajapinta, joka on integroitavissa selainpohjaiseen kassajärjestelmään. Rajapinnan oli tarkoitus olla yhteensopiva toimeksiantajayrityksen, Laihia Data Oy:n, nykyisten ja tulevien ratkaisujen ja järjestelmien kanssa. Tämän johdosta rajapinta piti kehittää yksinomaan PHP:llä.

Työn alussa käydään hieman läpi Laihia Data Oy:n nykyistä järjestelmää ja sen toiminnallisuutta sekä rakennetta. Tätä kautta lukijan on tarkoitus saada yleispätevä kuva olemassa olevasta järjestelmästä ja sen toiminnasta. Tämän jälkeen tarkastellaan ja tunnistetaan kehitettävän rajapinnan tavoitteet, jonka jälkeen kuvaillaan maksuliikennestandardi EMV. Se on luotu erityisesti sirukorttimaksamista silmällä pitäen. EMV on hyvin oleellinen osa tätä työtä, koska kehitettävän rajapinnan on tarkoitus mukailla Luottokunnan tarjoamaa maksupäätelaiterajapintaa, EMVLumoa, joka on EMV-standardoitu. Kun EMV on yleispiirteittäin käyty läpi, siirrytään työssä käytettyjen tekniikoiden, ohjelmien ja ratkaisujen tarkasteluun.

Toteutusosuudessa käydään läpi, mitä rajapinnan suunnittelu ja kehitys pitivät sisällään. Kehityksen aikana oli kaksi pääongelmaa ratkaistavana: PHP:n käytön rajallisuus ja synkronisuus sekä EMVLumon käyttö TCP:tä käyttäen. Kehityksen vaihekerrontaa tuetaan koodikatkelmilla, jotka ovat joko suoraan kehitetystä rajapinnasta leikattuja tai osviittaa antavia. Rajapinnan testaus projektin aikana ja sen jälkeen kuvaillaan pääpiirteittäin omassa kappaleessaan. Kehitetyn rajapinnan tietoturva käydään myös lyhyesti läpi. Tietoturvaosuudessa myös kerrotaan hieman EMV-standardin tietoturvamäärityksistä.

Yhteenvedossa todetaan työn edenneen odotetusti, vaikka aika hyvin rajallinen kaikkine vaiheineen olikin. Ongelmitta ei täysin selvitty, mutta oleellisimmat niistä ehdittiin ratkaista. Rajapinta vaatii tuotantokäyttöön päätyäkseen vielä työstöä, mutta se toimii hyvänä lähtökohtana ja runkona, kun uusia maksupäätelaitteita liitetään nykyiseen tai vanhaan järjestelmään.

Opinnäytetyöstä laadittiin kaksi versiota: julkinen ja luottamuksellinen versio. Luottamukselliseen versioon sisältyvät toimeksiantajayrityksen sisäiseen käyttöön

tarkoitetut liitteet. Nämä liitteet ovat kokonaan poistettu julkisesta versiosta. Luottamuksellinen versio luovutettiin vain ohjaajalle ja toimeksiantajayritykselle.

Avainsanat EMV, Lumo, PHP, rajapinta, rahaliikenne, kassajärjestelmä

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tietotekniikan koulutusohjelma

ABSTRACT

Author	Janne Säntti
Title	Developing an Interface for a Browser-based Point of Sale System
Year	2013
Language	Finnish
Pages	45 + 8 Appendices
Name of Supervisor	Jukka Matila

This thesis is about designing and development of an application programming interface (API) which is capable of being integrated in a browser-based point of sale (POS) system. API was supposed to be compatible with the current and future systems and solutions of the contracting authority, Laihia Data Ltd. Because of this the API had to be developed solely with PHP.

In the beginning, thesis covers the current system of Laihia Data Ltd. and its functionality together with its structure. Thus the reader is supposed to get the universal picture of the current system and its functionality. After this the thesis observes and identifies the goals of the developed API, after which the payment standard EMV (Europay, Mastercard and Visa) is described. It is specifically created with a view to chip card payment. EMV is a very crucial part of this thesis because the developed API is supposed to adapt to the EMV certified payment terminal API, EMVLumo, which is provided by Luottokunta. After the EMV has been outlined the thesis will move on to the technology, software and solutions which were used during the process.

The implementation section describes what the designing and development consisted of. During the development there were two major problems to be solved: restrictions of PHP and its synchronicity together with the usage of EMVLumo with TCP. Narrative of the development is supported by code snippets which are either taken straight out of the developed API or written as guidance purposes only. Testing of the API during and after the project is described in its own chapter. Security of the developed API is also shortly documented. Also, the security section briefly describes the security definitions of the EMV standard.

In summary section it is stated that the development proceeded like planned, although the time was limited with all the needed phases. The project was not without its share of problems even though the most crucial ones were solved. For the API to be used in production it needs more work to be done but it works as a good starting point and base when new payment terminals are added to the current or future systems.

The thesis was prepared in two versions: a public and a confidential one. The confidential version contains appendices for the contracting authority's internal use only. These appendices are completely removed from the public version. Confidential version was exclusively handed over to the supervisor and the contracting authority.

Keywords EMV, Lumo, PHP, API, financial transactions, point of sale

LYHENTEET

API	Application programming interface, ohjelmointiarajapinta
C#	C Sharp, ohjelmointikieli
CLI	Command-line interface, komentoliittymä
CSS	Cascading Style Sheets, tyyliohjelaji
EMV	Europay, MasterCard and Visa
EMVLumo	Luottokunnan maksupääterajapinta
FKL	Finanssialan Keskusliitto
GPL	GNU General Public License, julkaisulisenssi
GPRS	General Packet Radio Service, pakettikytkentäinen tiedonsiirtopalvelu
GSM	Global System for Mobile, maailmanlaajuinen matkapuhelinjärjestelmä
HTML	HyperText Markup Language, kuvauskieli
HTTP	Hypertext Transfer Protocol, tiedonsiirtoprotokolla
IP	Internet Protocol, TCP/IP-mallin Internet-kerroksen protokolla
MIT	Massachusetts Institute of Technology, teknillinen korkeakoulu Yhdysvalloissa
NAT	Network address translation, osoitteenmuunnos
PHP	PHP: Hypertext Preprocessor, ohjelmointikieli
POS	Point of sale, kassajärjestelmä

SQL	Structured Query Language, kyselykieli
SSH	Secure Shell, tietoliikenneprotokolla
SSL	Secure Sockets Layer, tietoverkkosalausprotokolla
TCP	Transmission Control Protocol, tietoliikenneprotokolla
TCP/IP	Transmission Control Protocol / Internet Protocol, usean Internet-liikennöinnissä käytettävän tietoverkkoprotokollan yhdistelmä
UDP	User Datagram Protocol, yhteydetön protokolla
WLAN	Wireless local area network, langaton lähiverkkotekniikka
VPN	Virtual Private Network, virtuaalinen erillisverkko
XML	Extensible Markup Language, merkintäkieli

SISÄLLYS

KUVALUETTELO

LIITELUETTELO

1 JOHDANTO

Toimeksiantajana tälle opinnäytetyölle toimi laihialainen ohjelmistoalan yritys Laihia Data Oy. Laihia Data Oy suunnittelee ja valmistaa tietohallintojärjestelmiä erikoistavarakaupan toimialalle Suomessa. Opinnäytetyön tarkoitus on suunnitella ja kehittää PHP:llä rajapinta, jolla on mahdollista käyttää Luottokunnan tarjoamaa rahaliikenteeseen tarkoitettua EMVLumo-rajapintaa selainpohjaisesti. Selainpohjaisuudella haetaan pilvipalvelumalli, joka on hyvin yleinen nykyään. Ohjelmat ja palvelut tarjotaan verkon yli pilvestä, jolloin paikallisilla muuttujilla ei ole suurta merkitystä.

Työn päätavoitteena on selvittää, onko kyseessä olevan rajapinnan kehittäminen PHP:llä käytännössä mahdollista. Samalla tarkistellaan eri vaihtoehtojen implementoimista rajapintaa ja tutkitaan mikä niistä on käytännöllisin. Kehitettävän rajapinnan on tarkoitus tulla tuotantokäyttöön niin nykyiseen järjestelmään kuin tuleviinkin ratkaisuihin. Täydellisyyttä siltä ei tämän työn rajoituksissa voida vaatia, mutta tarkoitus on saada aikaan hyvä runko, jota jatkokehittää.

Rajapinnasta ei toistaiseksi laadita sen kummempaa vaatimusmäärittelyä, mutta koodia kommentoidaan siten, että kommenttisyntaksi on hyödynnettävissä phpDocumentor-työkalulla, joka luo HTML-pohjaisen dokumentaation kehitetystä rajapinnasta ja sen kommenteista. Opinnäytetyö suoritetaan suunnittelun ja kehityksen osalta toimeksiantajan tiloissa ja laitteilla, mutta kirjallinen osuus työstetään omalla ajalla, omissa oloissa.

2 TOIMEKSIANTAJAYRITYS

2.1 Laihia Data Oy

Opinnäytetyö tehdään laihialaisen Laihia Data Oy:n toimeksiantamana. Ensikosketuksen yritykseen sain hakemalla harjoittelupaikkaa heiltä. Sain paikan ja melko nopeasti olinkin jo mukana kehittämässä yrityksen asiakkailleen tarjoamaa verkkokaupparatkaisua.

Laihia Data Oy suunnittelee ja valmistaa tietohallintojärjestelmiä erikoistavarakaupan toimialalle Suomessa. Yritys on perustettu vuonna 1986 ja se on koko yli 25-vuotisen historiansa ajan toiminut yksityisellä omistuspohjalla. Toiminta-aikanaan Laihia Data Oy on parhaansa mukaan palvellut erikoistavarakaupan tarpeita silmälläpitäen sekä samalla kasvattanut omaa että asiakkaidensa osaamista kyseiseltä alalta. Yrityksen toiminnan peruspilarit ovat asiakkaan toimintaympäristön tunteminen, yrittäjyys ja asiakkaan liiketoiminnan tukena toimiminen.

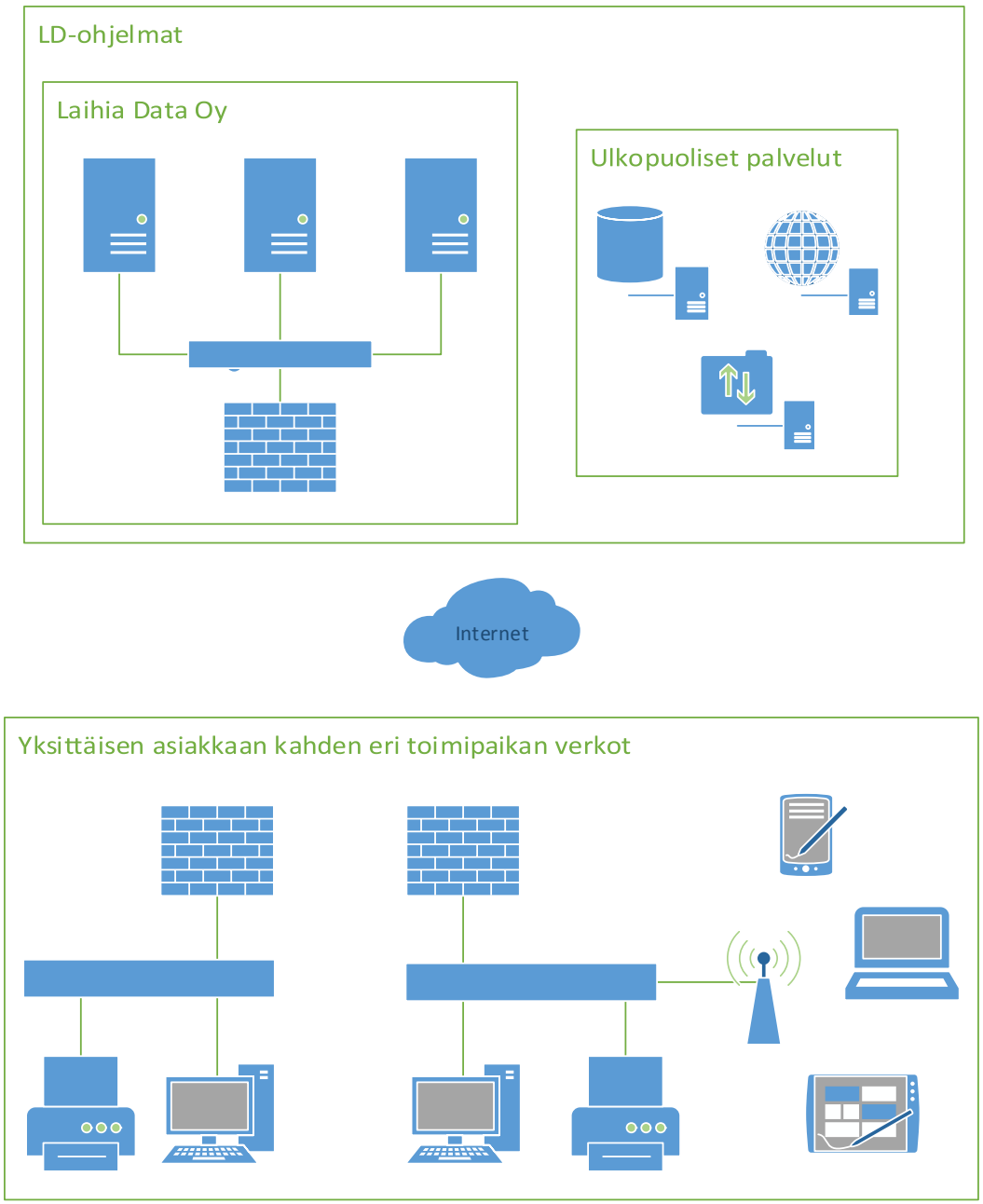
Laihia Data Oy suunnittelemme kestäviä ja innovatiivisia tietotekniikkaratkaisuja asiakkaidensa liiketoiminnan tavoitteiden ja toiveiden toteuttamiseksi ja automatisoimiseksi. Asiakkaiden tarpeiden tuntemisen ja yhteistyön ansiosta auttavat he asiakkaitaan johtamaan ja kehittämään liiketoimintaansa toimimaan yhä tehokkaammin. He pyrkivät toimimaan asiakasta lähellä ja olemaan kuunteleva osapuoli muutosherkkänä ja alati valmiina oppimaan uutta yhdessä asiakkaidensa kanssa.

LD-ohjelmistot, joiksi yrityksen ohjelmistoja kutsutaan, skaalautuvat asiakasyrityksen liiketoiminnan tarpeiden ja koon mukaan. Näin täytetään niin pienen kuin suurenkin yrityksen tietohallintojärjestelmän erilaiset vaatimukset. Laihia Data Oy on keskittynyt niille yritystoiminnan alueille Suomessa, joista heillä on vahvin asiantuntemus ja osaaminen, kuten auto-, matkailuvaunu- ja maatalouskonekauppa; hydrauliikka- ja työkalukauppa; varaosamyynti ja korjaamotoiminta; rengasmyynti- ja pinnoitustoiminta sekä pientarvike- ja urheiluvälinekauppa. /24/

2.2 Nykyinen järjestelmä

Yrityksen osaamisen pääpaino on kassajärjestelmäohjelmistossa, joka on toteutettu Progress 4GL:llä (nyk. OpenEdge Advanced Business Language /18/). Järjestelmä **(Kuva 1.)** toimii pääasiallisesti SSH-asiakasohjelmaa hyödyntäen. Tähän LD-ohjelmaan on ajan myötä kehitetty lisäominaisuuksia ja -osia, jotka ovat pitkälti selainpohjaisia. Hyödynnettyjä tekniikoita ovat tällöin mm. PHP, HTML, JavaScript, jQuery, AJAX, MySQL jne. Vanhaa LD-ohjelmaa pyritään palapalalta nykyaikaistamaan ja tätä kautta tuomaan selainpohjaiseksi eli ts. pilvipalvelumaiseksi. Kappaleen 2.1 alussa mainittu verkkokaupparatkaisu on osa tätä kehityskaarta. Nykytrendeihin räätälöitävissä oleva verkkokauppa mahdollistaa Laihia Data Oy:n asiakkaille sen, että he voivat tarjota tuotteitaan omille asiakkailleen kätevästi Internetin välityksellä.

Maksupäätejärjestelmän tarjoaa ja ylläpitää Luottokunta, joka on nykyään osa Nets-konsernia. Rajapinta maksupääterajapinnan ja LD-ohjelman välillä on Laihia Data Oy:n kehittämä. Itse standardeihin paneudutaan kappaleessa 4.1, mutta tässä kohtaa on hyvä kuvailla hivenen nykyistä rajapintaa. Tällä hetkellä käytössä olevalla rajapinnalla, joka on maksupääterajapinnan ja LD-ohjelman välissä, pystytään tekemään maksusuorituksia. Se ei kuitenkaan ole tarpeeksi vikasietoinen nykyisellään ja se pohjautuu vanhaan Luottokunnan tarjoamaan DoPay-viestijärjestelmään, kun taas uusi suositeltu rajapinta perustuu XML-viesteihin. Luottokunnan tarjoamia DoPay- ja XML-viestijärjestelmiä käsitellään myös paremmin kappaleessa 4.1. Edellä mainituista syistä Luottokunta kehottaakin maksupääteasiakkaitaan siirtymään uudempaan XML-pohjaiseen EMVLumo-rajapintaan. /7/



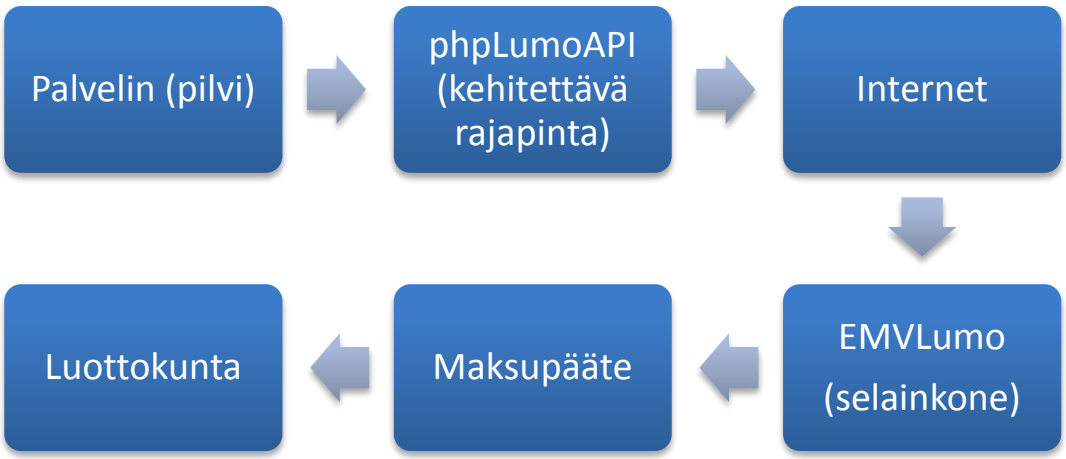
Kuva 1. Nykyinen järjestelmä.

3 MAKSUJÄRJESTELMÄN KEHITTÄMINEN

Tässä kappaleessa on tarkoitus kartoittaa tämän opinnäytetyön vaatimuksia ja tavoitteita. Tavoitteet pohjautuvat pääasiallisesti keskusteluihin ja palavereihin, joita käytiin toimeksiantajayrityksen kanssa. Vaatimuksia ohjasivat hyvin pitkälti nykyiset, jo käytössä olevat ratkaisut, sekä mahdollinen jatkokehitys rajapinnan ympärille. Kappaleen lopussa on tiivistelmä tehdyistä havainnoista ja päätöksistä.

3.1 Yleistä kehitettävästä rajapinnasta

Aivan ensimmäisenä on oleellista selvittää konkreettisesti, millaiseen järjestelmään ollaan rajapintaa kehittämässä, ja mikä sen tarkoitus on. Tulevaisuuden visio on, että Laihia Data Oy:n nykyinen SSH-pohjainen kassajärjestelmä korvattaisiin selainpohjaisella kassajärjestelmällä. Tällöin kaikki olemassa olevat ratkaisut tulee muuntaa selainpohjaisiksi. Kassajärjestelmän olennainen osa, maksupääte, ei ole poikkeus. Se on kuitenkin erikoistapaus, koska Luottokunta tarjoaa itse maksupäätelaitteen ja rajapinnan tietokoneelle, johon se kytketään. Siitä eteenpäin vastuu siirtyy kassajärjestelmän kehittäjälle. Ajatushan oli, että järjestelmä toimii pilvipalvelun tavoin **(Kuva 2.)**, jolloin tämän opinnäytetyön aihe, eli rajapinta, astuu mukaan. Tuon rajapinnan pitäisi pystyä lähettämään ja vastaanottamaan palvelinpäässä viestejä, jotka tietokone, johon maksupäätelaite on kytketty, lähettää tai vastaanottaa.



Kuva 2. Visioitu kassajärjestelmä rajapinnan kera.

3.2 Rajapinnan vaatimukset

Seuraavaksi on hyvä selvittää, mikä ohjelmointirajapinta (engl. application programming interface, API) on. Yleensä, kun puhutaan rajapinnasta, niin tarkoitetaan juuri ohjelmointirajapintaa. Toisinaan ohjelmointirajapinta ja luokkarajapinta (engl. class interface) voivat sekoittua, jos niitä ei erotella selkeästi. Ohjelmointirajapinta on kokoelma funktiota, proseduureja, metodeja tai luokkia, joita ohjelma käyttää kutsuakseen palveluita käyttöjärjestelmältä, ohjelmistokirjastoilta tai miltä tahansa muulta tietokoneella olevalta instanssilta. Ohjelmointirajapinnat eivät siis ole kokonaisia ohjelmia, vaan niitä hyödynnetään isommissa kokonaisuuksissa. /3/

Ohjelmointirajapinnat ovat käytännössä pakollisia, jos halutaan, että kolmasosapuoli voi kehittää omia sovelluksiaan, jotka hyödyntävät emosovellusta. Esimerkiksi suositusta sosiaalisesta Twitter-mikroblogipalvelusta voidaan hakea haluttuja viestejä heidän tarjoamallaan ohjelmointirajapinnallaan, eikä käyttäjän tarvitse tietää, miten itse palvelu toimii. /6/

Luokkarajapinnalla kerrotaan, mitkä metodit luokka implementoi. Näin ei erikseen tarvitse määritellä, miten kyseiset metodit käsitellään. Kaikki luokkarajapinnassa julistetut metodit täytyy olla määriteltynä julkisiksi. Samaa kaavaa käyttämällä luokkarajapinta voidaan implementoida uuteen luokkaan, jolloin sen ominaisuuksia voidaan jatkaa tietämättä, miten alkuperäiset metodit toimivat. /16/, /21/

Tämän opinnäytetyön tarkoitus on kehittää ohjelmointirajapinta eli API. Mitkä ovat ne tavoitteet, jotka kehitettävän rajapinnan tulisi sitten täyttää? Asiasta keskusteltiin useasti jo harjoitteluaikanani toimeksiantajayrityksessä, kun mahdollista opinnäytetyötä suunniteltiin. Opinnäytetyön aihetta alettiin suunnittelemaan suullisella tasolla jo varhaisessa vaiheessa. Selvää oli, että rajapinta kehitetään Laihia Data Oy:n järjestelmien ja Luottokunnan tarjoaman EMVLumo-ohjelmointirajapinnan välille. EMVLumo mahdollistaa kommunikoinnin maksupäätelaitteen kanssa.

Aluksi todettiin, että nykypäivän suuntaus ja trendi on selkeästi selainpohjaiset palvelut, joita usein myös pilvipalveluiksi kutsutaan. Tämän pohjalta tehtiin päätös, että kehitettävän rajapinnan täytyy olla helposti ja vaivattomasti hyödynnettävissä

selainpohjaisessa järjestelmässä. Kriteerinä oli myös, että rajapinta tulee olla suhteellisen helposti liitettävissä nykyiseen järjestelmään, joka kuvattiin kappaleessa 2.2. Toisin sanoen nykyisellä järjestelmällä tarkoitetaan olemassa olevaa maksupääterajapintaa sekä verkkokauppapohjaa, jotka ovat molemmat toteutettu PHP:llä. Päädyttiin siihen, että rajapinta tulee kehittää PHP:llä. Työn tärkein tehtävä olikin selvittää, onko tämä ylipäätään edes mahdollista.

Toiveena oli, että kehitettävää rajapintaa voi helposti hyödyntää myös sellainen henkilö, joka ei sen kehityksessä ole mukana, joten dokumentointi ja koodin kommentointi nousivat suureen rooliin. Kehityksen loppupuolella päädyttiinkin hyödyntämään phpDocumentoria, josta lisää kappaleessa 5.3.

Toinen pääkriteeri heti PHP:n jälkeen oli, että rajapinta toteuttaa EMVLumon XML-viestejä käyttävän vaihtoehdon, koska nykyinen käytössä oleva DoPay-viestejä hyödyntävä ratkaisu on vanhenemassa kovaa vauhtia, kuten Luottokunta on ilmoittanut. XML-viestit lähetetään TCP-protokollaa käyttäen. /7/

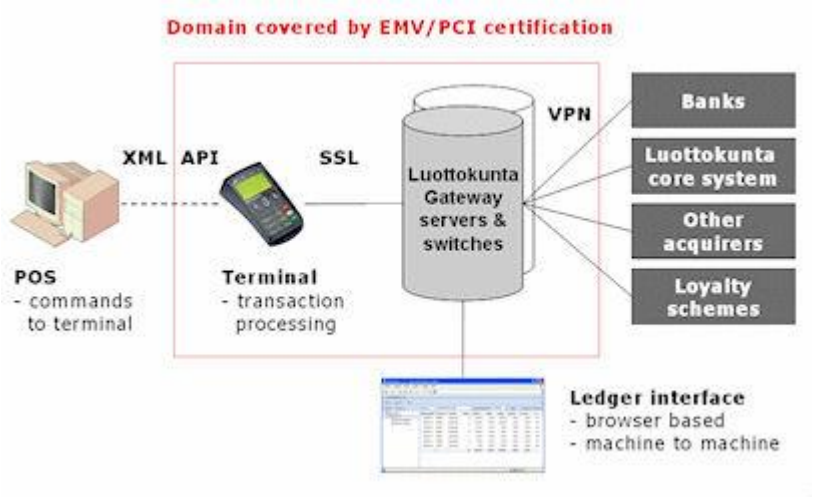
3.3 Opinnäytetyön tavoite

Edellisen kappaleen pohdiskelun ja vaateiden pohjalta voidaan todeta, että opinnäytetyön tavoitteeksi asetettiin helppokäyttöisen, jatkokehityskykyisen, EMVLumon XML-viestejä tukevan ja PHP:tä hyödyntävän rajapinnan kehittäminen. Tämän lisäksi haluttiin, että kehitys ja koodi ovat hyvin kommentoitu ja dokumentoitu mahdollista jatkokehitystä silmällä pitäen. Lisättäköön vielä, että rajapinnan pitäisi olla mahdollisimman vikasietoinen.

4 STANDARDISOITU MAKSULIIKENNE

Mikä on EMV, entä EMVLumo? Miten ne liittyvät standardoituun maksuliikenteeseen? Tämä kappale valaisee hieman, mitkä ovat EMV ja EMVLumo sekä mihin niitä tarvitaan. Sen verran tässä vaiheessa voi jo paljastaa, että EMV ja EMVLumo ovat kaksi eri asiaa.

Rajapinta tullaan toteuttamaan siten, ettei itse standardia tarvitse juuri ottaa huomioon. Tämä siksi, että varsinainen rahaliikenne kulkee maksupäätelaitteen ja Luottokunnan välillä **(Kuva 3.)**, ei siis tämän opinnäytetyön aiheena olevan rajapinnan läpi. Totuus on kuitenkin se, että kehitettävä rajapinta rakentuu EMV-standardin päälle, joten on se hyvä selittää hivenen yksityiskohtaisemmin. Tässä kappaleessa käydyt asiat perustuvat pääasiallisesti Luottokunnan ja Finanssialan Keskusliiton tarjoamiin materiaaleihin.



Kuva 3. EMV-sertifioitu maksujärjestelmä. /14/

4.1 EMV

Lyhenne EMV tulee sen perustaneiden organisaatioiden nimistä: Europay, Mastercard ja Visa. Se on maailmanlaajuinen standardi sirukorttimaksamista varten. Standardilla pyritään varmistamaan yhteensopivuus ja turvallisuus, kun käytetään sirukorttia, kassapäätejärjestelmää tai raha-automaattia rahaliikenteeseen. EMV on nimensä veroisesti kolmen organisaation Europayn, Mastercardin ja Visan yhteinen

pyrkimys varmistaa turvallinen rahaliikenne niin, että heidän sirukorttejaan voidaan käyttää kaikkialla. /1/, /5/, /6/, /11/, /22/

Sirukortti ei rakenteeltaan ja ulkonäöltään muuten eroa vanhasta magneettijuovakortista muutoin kuin että siinä on pieni mikropiiri. Mikropiirinsä ansiosta sirukortti tarjoaa mahdollisuuden tunnistautumiseen, varmentamiseen, tiedontallentamiseen ja pienelle prosessoinnille. Sirukorteista löytyy mm. kontaktiton, kontaktillinen ja hybridimalli. EMV-standardi määrittelee toiminnot sirukortin ja sitä lukevan laitteen väliselle fyysiselle ja sähköiselle tasolle sekä data- ja sovellustasolle, kun kyseessä on rahaliikenne. Kontaktikorteille on määritelty standardi ISO/IEC 7816:n pohjalta ja kontaktittomille korteille ISO/IEC 14443:n pohjalta. /1/, /5/, /6/, /11/, /22/

Sirukortissa on neljä ehdotonta etua vanhaan magneettijuovakorttiin nähden: sirukortin varmentaminen, riskinhallintaparametrit, digitaalinen allekirjoitus, parempi kortinhaltijan varmennus. Kuten edellä olevista sysitä voi päätellä, suurimmat edut ilmenevät turvallisuutena. Tämä ei toki ole mikään huono asia kun ajatellaan ketkä kärsivät väärinkäytöksien yhteydessä, yleensä kaikki muut, paitsi väärinkäyttäjä. Edut eivät rajoitu turvallisuuteen, myös käytännöllisyys kohentuu huomattavasti. EMV:n ansiosta kortinhaltijan ei tarvitse enää antaa allekirjoitusta kuitille vaan PIN-tunnus ajaa tämän asian, mikä jouhentaa maksuprosessia huomattavasti. /1/, /5/, /6/, /11/, /22/

Kappaleen alussa mainittu EMVLumo taas on Luottokunnan tarjoama rajapinta kassajärjestelmän ja maksupäätelaitteen välille. Se myös toteuttaa EMV:hen liittyvät standardimääritykset lähettämällä maksupäätelaitteeseen tallennetut maksutapahtumat eteenpäin Luottokunnalle. EMVLumo on uuden sukupolven rajapintaratkaisu, joka hyödyntää nykyajan henkeen kuuluvasti XML-viestejä liikennöintiin. Vanhemman sukupolven rajapinta, EMVCard, käyttää Luottokunnan omaa DoPay-viestiä, joka pyritään korvaamaan uusilla XML-viesteillä, vaikka EMVLumo DoPay-viestejä taaksepäin yhteensopivasti tukeekin. /7/

4.2 Maksupäätejärjestelmä

FKL kuvailee maksupäätejärjestelmän **(Kuva 4.)** seuraavasti:

> ”EMV-maksupäätejärjestelmä on maksunsaajalla oleva, korttitapahtumien tilittäjän hyväksymän tahon sertifioima laite tai laitteisto. Maksupääte automatisoi avain- ja parametriaineistojen noudon, kortilla maksamisen, varoitustietojen tarkistamisen ja tapahtumien pankkiin lähettämisen sekä tapahtumien varmentamisen. Kortin tiedot luetaan maksupäätteellä joko sirulta tai magneettijuovalta ja maksupääte tekee tarvittavat tarkistukset, taltioi ostotapahtuman sekä lähettää tapahtumat.” /9/

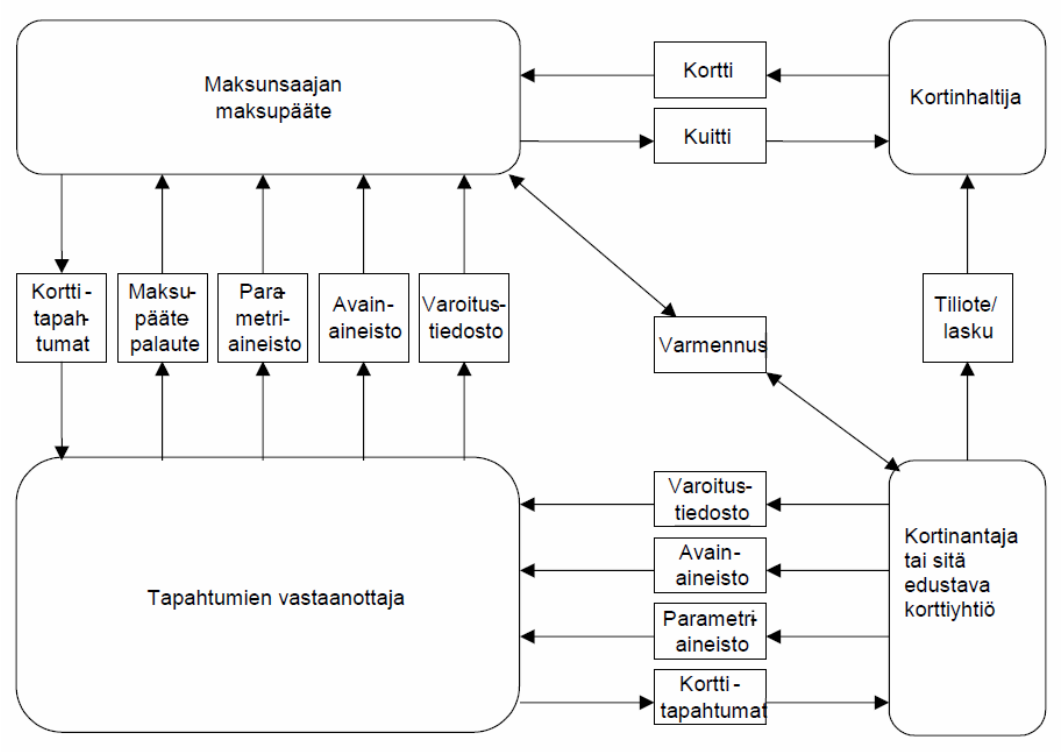
Maksupäätejärjestelmä on se kokonaisuus esimerkiksi kaupan kassalla, joka kattaa seuraavat asiat:

- siru- ja magneettijuovakortin lukija
- tunnuslukunäppäimistö (PIN-pad)
- näyttö
- näppäimistö
- kuittitulostin
- maksupääteohjelmisto
- linjayhteys. /9/

Laihia Data Oy tarjoaa näistä yleensä asiakkailleen kaiken muun paitsi linjayhteyden. Laitteet tulevat eri laitevalmistajilta, mutta kassajärjestelmäohjelmisto on Laihia Data Oy:n kehittämä. Tällöin kaikki laitteet ovat helposti konfiguroitavissa yhteensopiviksi ja liitettävissä LD-ohjelmistoverkkoon VPN-tunnelia hyväksikäyttäen. Laitteet yleensä konfiguroidaankin valmiiksi ennen kuin ne lähetetään tai viedään asiakkaalle. Asiakkaan tarvitsee siis vain omatoimisesti kytkeä laitteet toimipisteeseensä ja yhdistää verkkoon. Tarvittaessa voidaan ottaa etäyhteys.

Maksupäätejärjestelmään liittyy useita eri toimintoja ja ominaisuuksia, kuten osto- ja käteisnostotapahtumat sekä niiden välitysmenettelyt; maksupäätevarmennus ja

sen tekeminen; palaute ja kauppiashyvitys; avain- ja parametritietojen välitys; varoitustietojen välitysmenettelyt. /9/



Kuva 4. FKL:n kuvaus maksupäätejärjestelmästä. /9/

5 KEHITYSYMPÄRISTÖ JA -TYÖKALUT

Kehityksen tukena oli muutamia eri ohjelmia, lähdekoodeja sekä laitteita. Tämän kappaleen tarkoitus on perehtyä näistä ainakin tärkeimpiin ja oleellisimpiin.

5.1 PHP

Kehitettävän rajapinnan ohjelmointikieleksi valittiin PHP **(Kuva 5.)**. Vaikka valinta oli sinänsä selvä heti alusta lähtien, oli sillä myös syvempi tarkoitus. Opinnäytetyön tavoitteissa ja vaatimuksista kerrottiin kappaleessa 3.2. Kappaleesta selviää, että työn tarkoitus on myös samalla tutkia, onko kehitettävän rajapinnan toteuttaminen PHP:llä edes mahdollista. Samaisesta kappaleesta selviää myös miksi rajapinta halutaan toteuttaa PHP:llä. Lyhyesti: iso osa nykyisestä järjestelmästä on toteutettu sillä ja lisää tullaan toteuttamaan.

PHP on laajalti käytetty avoimen lähdekoodin lisenssiä käyttävä yleispätevä skriptausohjelmointikieli, joka soveltuu erityisesti verkko-ohjelmointiin sekä sitä voi upottaa HTML:ään. Suurin osa sen syntaksista on lainattu C:stä, Javasta ja Perlistä. Sitä on myös helppo oppia. Kielen päämääränä on dynaamisten verkkosivujen kirjoittaminen nopeasti ja vaivattomasti. PHP:n käyttö ei rajoitu vain verkkosivuihin, vaan sillä voi tehdä paljon muutakin. /17/

```
/**
 * Getter for boolen $connected.
 *
 * @return  boolean
 */
public function getConnected() {
    return $this->connected;
}
```

Kuva 5. Esimerkkikatkelma kehitetystä rajapinnasta. getConnected-funktio.

Omalta kannaltani pidän PHP:tä hyvänä vaihtoehtona, koska hallitsen mielestäni hyvin sen perusteet ja joitain vaikeampiakin osa-alueita. Olen osallistunut korkea-

koulutasoiselle kurssille, joka käsitteli PHP:tä. Myös Laihia Data Oy:ssä työskennellessäni olen joutunut turvautumaan usein kieleen. PHP:n käyttö ei siis kielenä osoittautunut vaikeaksi, mutta sen rajoitteet kyllä. Niistä enemmän kappaleessa 6.5.

Rajapinnan kehityksessä käytettiin PHP:n versio 5:sta, koska kyseisessä versiossa esiteltiin uusia funktioita ja ominaisuuksia, jotka auttoivat huomattavasti kehitystyössä. Näistä esimerkkinä mainittakoon socket_stream_client-funktio, joka yksinkertaisti TCP-yhteyden käyttöä paljon. Funktio avaa yhteyden ja käsittelee sitä, kuten tiedostovirtaa (engl. file stream), jolloin tiedon lähettäminen ja vastaanottaminen ovat helpompaa. Uudessa versiossa parannettiin myös oliopohjaista ohjelmointimallia, mutta sen edut eivät työstä tule juuri esiin. /17/

5.2 KLogger

KLogger on pieni yhden miehen avoin projekti, joka mahdollistaa PHP-ohjelmalle kätevän ja yksinkertaisen lokituksen eli tapahtumien kirjaamisen. Käytännössä se on vain yksi luokka, joka täytyy lisätä ohjelmakoodiin, jonka jälkeen sen lokitusfunktioita **(Kuva 6.)** voi kutsua. Lokimerkinnät kirjataan tekstitiedostoon. KLoggeria jaetaan MIT-lisenssin alaisuudessa.

```
// Log current action.
$this->KLogger->logInfo("Trying to establish connection to " . $this->host . "
in port " . $this->port . ".");
```

Kuva 6. Esimerkkilokitusfunktio.

5.3 phpDocumentor

phpDocumentor on työkalu, jolla voi generoida dokumentaation kirjoitetusta PHP-lähdekoodista. Se tunnistaa lähdekoodista tiettyä syntaksia noudattavat kommentit **(Kuva 7.)** ja muodostaa niiden pohjalta HTML-pohjaisen dokumentaation. Dokumentaation avulla on helpompi jälkikäteen esimerkiksi tarkistaa, miten joku tietty luokka toimi ja mitkä ovat sen funktiot. Tällainen dokumentaatio on oleellinen, varsinkin silloin, jos jatkokehittäjä on joku toinen kuin alkuperäinen tekijä. Varsinkin tämän projektin kohdalla phpDocumentorin käyttö oli vähintäänkin pakollinen, koska kyseessä on rajapinta. Rajapinnasta saa huomattavasti paremmin kiinni, kun

tutkii sille hyvin tehtyä dokumentointia. Lisäksi phpDocumentor on avoin ja lisenssinä toimii LGPL.

```
/**
 * Creates connection to the Lumo API.
 *
 * Connection will be created with the parameters that were initialized in the
constructor.
 *
 * @return  boolean
 */
```

Kuva 7. Esimerkkikommentti, jonka phpDocumentor ymmärtää.

5.4 EMVLumo ja XML

Tietokone, johon maksupäätelaite liitetään, asennetaan Luottokunnan tarjoama EMVLumoAgent-ohjelma. Ohjelma sisältää TCP/IP Socket-palvelimen, joka kuuntelee XML-viestejä. Kuten kappaleissa 3.1 ja 4.1 mainitaan, perustuu EMVLumon kommunikointi kehitettävän rajapinnan kanssa täysin XML-viesteihin. XML-merkintäkieli on siis hyvin oleellinen osa kehitettävää rajapintaa. Kaikki viestit, joita EMVLumo lähettää ja vastaanottaa ovat XML-muotoisia **(Kuva 8.)**. /8/

```
<EMVLumo xmlns="http://www.luottokunta.fi/EMVLumo">
    <InitializeSystem />
</EMVLumo>
```

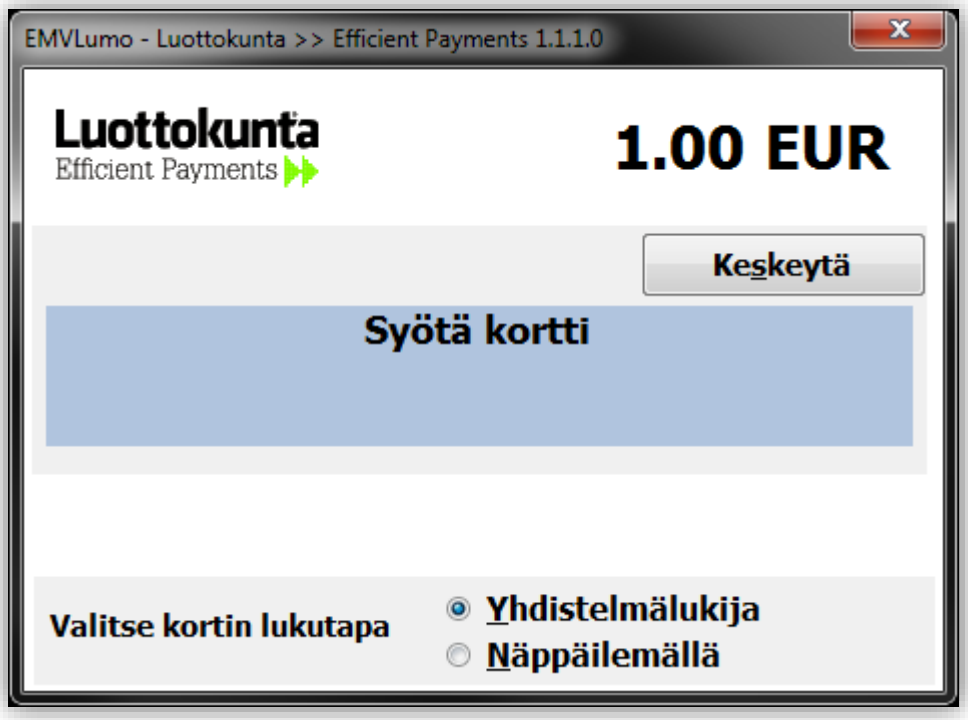
Kuva 8. EMVLumon XML-esimerkkiviesti.

XML eli Extensible Markup Language on merkkikieli, joka määrittelee nipun sääntöjä dokumenttien koodaamiseksi, jotka niin ihminen kuin konekin voi lukea. Se on määriteltynä XML 1.0 -spesifikaatiossa, jonka on tuottanut W3C-organisaatio. Standardi on avoin kaikille. XML:n tavoite on olla yksinkertaista, yleispätevää ja helposti käytettävissä Internetin välityksellä. /10/

EMVLumon toiminta perustuu metodien, ominaisuuksien ja tapahtumien lähettämiseen XML-viestin rakenteessa. Esimerkiksi, jos halutaan alustaa maksupäätelaite, lähetetään InitializeSystem-metodi XML-viestissä **(Kuva 8.)**. Metodeihin vastataan aina arvolla True tai False riippuen, onnistuiko se. Jos EMVLumolla on

jotain palautettavaa, se myös palautetaan samassa viestissä. EMVLumo-rajapinta toimii asynkronisesti, joka muodostikin yhden tämän opinnäytetyön suurimmista haasteista. Tästä lisää kappaleessa 6.5. /8/

Maksutapahtuma voidaan suorittaa kahdella tavalla EMVLumon näkökulmasta: MakeTransaction-metodilla, joka käyttää EMVLumoAgentin dialogeja **(Kuva 9.)** hyväkseen. Tällöin kehitettävässä rajapinnassa ei tarvitse ottaa huomioon koko maksutapahtuman etenemistä, vain aloitus ja lopetus. Maksutapahtuma on mahdollista suorittaa myös StartTransaction-metodilla, jolloin koko maksutapahtumaprosessi pitää käydä vaiheittain läpi ulkopuolisessa ohjelmassa, joka tässä tapauksessa olisi kehitettävä rajapinta. Tämän opinnäytetyön puitteissa käytetäänkin ensin mainittua MakeTransaction-metodia. /8/



Kuva 9. EMVLumoAgentin dialog-ikkuna.

EMVLumo tietää, että XML-viesti on tullut perille, kun se loppuu tyhjään merkkiin (engl. null mark) eli ”\0”. Tämä merkki pitää löytyä jokaisen EMVLumo-elementin jälkeen. EMVLumo myös lähettää viesteissään samaisen merkin, joten se täytyi ottaa huomioon kehitettävässä rajapinnassa.

5.5 Muut kielet

Kehitettävän rajapinnan toiminnan demonstroimiseksi käytännössä tein suhteellisen kevyen ja yksinkertaisen verkkosivun hyväksikäyttäen PHP-, HTML-, jQuery-, CSS-ohjelmointikieliä. PHP:n ja HTML:n avulla voidaan tulostaa selaimeen käyttöliittymä, jonka avulla käyttäjä voi testata rajapintaa ja maksupäätettä. jQuery on JavaScriptin laajennus, joka helpottaa JavaScriptin käyttöä. Testikäyttöliittymään jQuery integroituu siten, että se suorittaa AJAX:ia käyttäen kaikki pyynnöt selaimesta palvelimelle, jolloin koko sivua ei tarvitse ladata joka kerta uudelleen. CSS mahdollistaa HTML-sivun helpon ja rajattoman muokkauksen. Sitä käytettiin testikäyttöliittymän siistimiseen ja ulkoasun kohentamiseen. Käyttöliittymän tekoon ei syvennytä kovin, koska sen on tarkoitus toimia vain demonstraationa ja täten ei ole osa varsinaista kehitettyä rajapintaa, jota myöhemmin voi hyödyntää.

5.6 Luottokunnan mallikassaohjelma

Kehitystyön alkumetreillä Luottokunta lähetti lähdekoodin heidän kehittämästään malliohjelmasta, joka hyödyntää EMVLumoa. Malliohjelma toteuttaa pääpiirteittäin kaikki EMVLumon ominaisuudet. Ohjelmointikielenä malliohjelmassa on käytetty C#:ia. Kielenä C# ei ole minulle tuntematon, joten malliohjelman tulkitsemisessa ja valjastamisessa hyödykseni ei ollut suurempia ongelmia. Lähdekoodin mukana toimitettiin myös valmiiksi käännetty ohjelma, jolloin lähdekoodia ei tarvinnut erikseen kääntää. Lähdekoodit avattiin luettavaksi tekstieditorilla.

5.7 Wireshark

Luottokunta tarjoaa melko kattavan dokumentaation EMVLumon käytöstä, mutta varsinkin aluksi meinasi olla hankalaa saada XML-viestit oikeaan muotoon. Tämän takia käytettiinkin melko paljon Wireshark-pakettihaistelija. Wiresharkilla pystyttiin nappaamaan lähetetyt ja vastaanotetut TCP-paketit ja tutkimaan niiden rakennetta tarkemmin. Wiresharkin ansiosta saatiin tulostettua maksupäätelaitteen ruudulle ensimmäinen ”Hello World!”-teksti. Käytetty versio oli 1.8.6.

5.8 Käyttöjärjestelmät ja muut ohjelmat

Työpisteenä koko rajapinnan kehityksen aikana toimi PC-tietokone, johon oli asennettu Microsoft Windows 7 -käyttöjärjestelmä. Kaikki kehitettävään rajapintaan liittyvät tiedostot olivat Linux-palvelimella, jossa itse rajapintaa myös käytettiin. Aluksi testaaminen suoritettiin pääasiassa CLI-pohjaisesti eli komentorivillä. CLI:n käytön mahdollisti PuTTY-ohjelma, jolla voi ottaa SSH-yhteyden sitä tukevaan käyttöjärjestelmään. Ohjelmakoodien muokkaamiseen käytettiin pääasiassa maksullista UltraEdit-tekstieditoria, jonka käyttöön LaihiaData Oy:llä on lisenssi. Kotona koodien muokkaaminen tapahtui pääasiallisesti ilmaista Notepad++:aa käyttäen, joka soveltuu mainiosti kevyeen ohjelmointiin. Kyseiselle tekstieditorille on paljon lisäosia, jotka helpottavat työskentelyä. Esimerkiksi NppExport-lisäosa mahdollisti koodin kopioinnin muotoiluineen liitedokumenttiin.

5.9 Maksupäätelaite

Kehitysympäristössä käytetty maksupäätelaite oli mallia Ingenico ML-30 Contactless **(Kuva 10.)**. Laitteella pystyy käyttämään kontaktikortteja sekä kontaktittomia. Käytössä oli Nordean toimittamat testikortit, jotka olivat kaikki kontaktikortteja eli ne piti työntää maksupäätteen sisään. Luottokunta on pannut laitteen asetukset ja parametrit valmiiksi jo ennen toimitusta, joten sen käyttö ei vaadi muuta kuin oikeiden ajureiden asentamisen kassajärjestelmään.

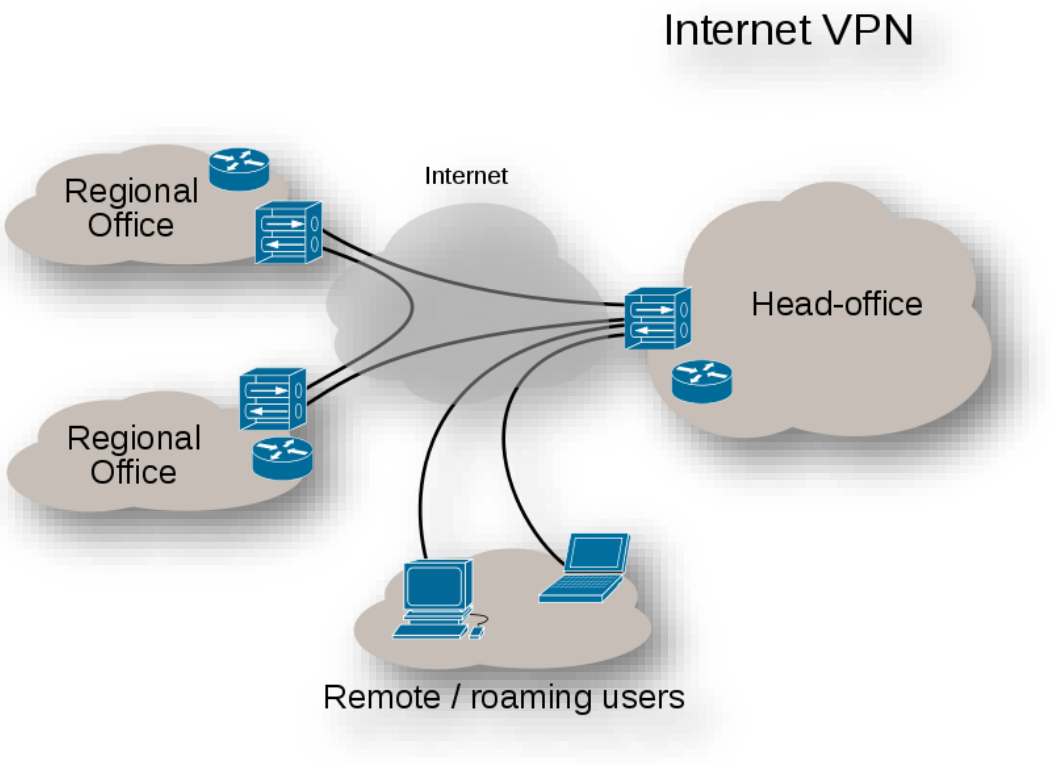
Kuva 10. Ingenico ML30 -maksupäätelaite. /13/

5.10 EGroupware

Toimeksiantajayrityksellä on käytössään avoin EGroupware-ryhmätyösovellus, joka mahdollistaa monien käytännöllisten työkalujen käytön organisaation sisäisesti selainta hyödyntäen. EGroupwaresta löytyy esimerkiksi kalenteri, sähköposti, yhteystietoluettelo, wiki ja monia muita. Kovimmassa käytössä sovelluksesta yrityksen sisäisestikin on InfoLog, joka mahdollistaa muistiinpanojen ja tehtävien ylöskirjaamisen sekä ylläpidon. Yhteisöversion lisenssi on GPL2.

5.11 VPN

Kaikki Laihia Data Oy:n asiakkaat, joiden palvelut tuotetaan yrityksen palvelinsalista, ovat VPN-tunnelointien takana. Eli asiakkaille päin näyttää, että he olisivat Laihia Data Oy:n kanssa samassa sisäverkossa **(Kuva 11.)**. Eri asiakkailla ja itse Laihia Data Oy:llä ovat tietenkin eri aliverkot, mutta näitä on mahdollista yhdistää ja muokata. VPN-tunneli muodostetaan laitepalomuurien läpi, jolloin tietoturvan pitäisi olla taattu. Kehitettävä rajapinta käyttää samaa ajatusmallia, jolloin rajapinnan liikennöintiä ei tarvitse erikseen suojata esimerkiksi SSH:lla, koska se tehdään jo VPN-tunnelille. VPN on käytännössä pakollinen kehitettävän rajapinnan käytön kanssa. Siksi pakollinen, että maksutapahtumat ovat helpommin ohjattavissa omille maksupäätteilleen, kun tiedetään sisäverkon IP-osoitteet. Tällöin ei tarvitse erikseen asettaa NAT-ohjauksia reitittimelle.



Kuva 11. Malliesimerkki VPN:stä. /23/

6 RAJAPINNAN TOTEUTUS

Tässä kappaleessa käydään läpi kaikki rajapinnan toteutukseen tarvitut vaiheet sekä niiden tulokset. Tästä eteenpäin lukijalla on hyvä olla perustason käsitys PHP:stä ja olio-ohjelmoinnista. Toimeksiantaja pyysi, ettei lähdekoodeja julkaistaisi sellaisenaan, joten ne on toimitettu vain työn tarkistajalle liitteinä. Tämän dokumentin kannalta on tietenkin tärkeää, että tehty työ tulee ilmi, joten kirjoitusta tuetaan koodikatkelmilla, jotka eivät kuitenkaan paljasta koko kuvaa rajapinnan toiminnasta.

6.1 Pohjustus

Kehitettävän rajapinnan suunnitteluvaiheet käytiin jo osittain läpi kappaleessa 3, mutta aiheeseen syvennytään vielä hieman erikseen. Mielestäni suunnittelu on kuitenkin iso osa tällaista ohjelmistokehitysprojektia. Tätä mieltä oli myös toimeksiantajayritys.

Kuten kappaleessa 3 mainitaan, ennen työn aloittamista aiheesta käytiin useampia pienempiä keskusteluja ja yksi suurempi palaveri. Näiden pohjalta luotiin tavoitteet ja suunnitelma opinnäytetyön etenemiselle ja työstämiselle. Ensimmäisessä palaverissa asetettiin myös alustava aikataulu ja jaksotus rajapinnan kehityksen eri osa-alueille. Aikataulusta tehtiin kevyt Gantt-kaavio, mutta se jääköön liittämättä yksinkertaisuutensa johdosta.

Sovittiin, että joka viikon alussa pidetään välipalaverin, jossa käydään edellisen viikon aikaansaannokset läpi sekä tunnistetaan sen hetkiset ongelmat ja yritetään kehitellä niihin ratkaisu, jota voi sitten työstää. Näin koko kehitys etenikin. Toisinaan oli useita asioita läpikäytävänä ja toisinaan palaveri oli ohi vartissa. Nämä välipalaverit olivat kuitenkin hyvin oleellinen osa kehitysprosessia. Palavereiden avulla pysyttiin toimeksiantajan kanssa koko ajan samalla sivulla, muuten kehitys olisi voinut esimerkiksi rönsyillä väärään suuntaan tai polkea paikallaan.

Koko kehitysprosessin ajan pidettiin päiväkirjaa kappaleessa 5.10 mainitun EGroupwaren InfoLog-sovelluksen avulla. Sinne kirjattiin viikon tapahtumat ja ilmenneet sekä ratkenneet ongelmat. Näin kaikki projektia seuranneet pystyivät helposti

tarkistamaan työn tilanteen. Se toimi samalla henkilökohtaisena tehtävä- ja muistilistana.

Aluksi tutustuttiin FKL:n tarjoamaan toiminnalliseen kuvaukseen EMV-maksupäätejärjestelmästä (/9/), joka antaa suhteellisen selkeän kuvan maksupäätejärjestelmän toiminnallisuudesta. Dokumentin tutkiminen oli pakollista, koska tietämykseni ennen työn aloittamista maksujärjestelmien suhteen oli hyvin minimaalinen. Seuraavaksi tarkasteltiin ja tutkittiin EMVLumo-rajapinnan määritelmädokumenttia (/8/). Dokumentissa kerrotaan, miten EMVLumo toimii ja miten sitä käytetään. Tämä kaikki oli hyvin oleellista tietoa kehitettävän rajapinnan suhteen.

6.2 Käyttötapaukset

Dokumenttien tutkimisen jälkeen ajatukset siirrettiin kehitettävään rajapintaan ja sen varsinaiseen kehittämiseen. Neuvoksi saatiin, että aluksi olisi hyvä tehdä lista oleellisimmista käyttötapauksista (engl. use cases) kehitettävän rajapinnan suhteen. Näiden käyttötapausten perusteella olisi sitten helppo kehittää rajapintaa. Ajatuksen oli, että kun yhden oleellisen käyttötapauksen saa valmiiksi, voi sitä hyödyntää sitten muissa. Ensimmäiseksi käyttötapaukseksi valikoitu perusmaksutapahtuma, jossa kortinhaltija suorittaa määritellyn summan sirukortilla. Tämä ei tietenkään ainut käyttötapaus ollut, joka otettiin huomioon, mutta sen ympärille rajapinta rakennettiin.

Toimeksiantajayrityksen kanssa sovittiin, että työ rajataan aluksi juuri tähän maksutapahtumaan, josta on sitten mahdollista laajentaa muihinkin käyttötapauksiin opinnäytetyön teon jälkeen. Tämä kaikki kuitenkin niin, että virhetapaukset maksutapahtuman yhteydessä on otettava huomioon jo kehitysvaiheessa. Tässä kohtaa on hyvä muistaa, että kehitettävä rajapinta käsittelee käyttötapaukset maksupäätteestä rajapintaan, eli EMV-standardin rajoituksia ja määräyksiä ei oteta käyttötapauksissa huomioon.

6.3 Hello World!

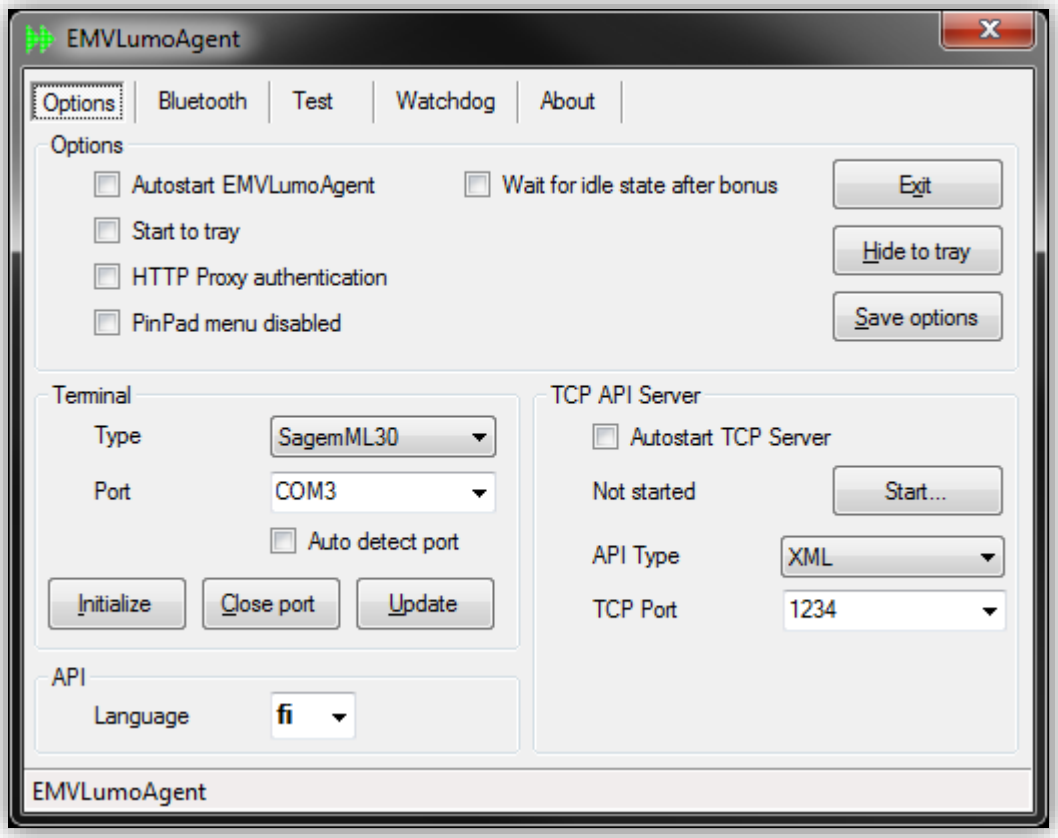
Toimeksiantajan kanssa päätettiin, että alkututkimuksen jälkeen pyritään testaamaan yhteyden muodostaminen ja XML-viestin lähettäminen EMVLumolle perinteisellä ”Hello World!” -viestillä **(Kuva 14.)**. Näin ollen varsinaisen rajapinnan luokkarakenne, käytettävät tekniikat ja muut jätettiin syrjään. Haluttiin vain tietää onnistuuko EMVLumon käyttö PHP:llä alkuunkaan.

Testaaminen lähti käyntiin siten, että otettiin testikoodin pohjaksi käytössä oleva socket-rajapinta, joka toteuttaa käytössä olevan, mutta vanhan DoPay-viestien käytön. DoPay-viestit käyttäytyvät hyvin vastaavalla tavalla kuin uudet XML-viestit EMVLumossa. DoPay-viestin lähetysketju on kuitenkin paljon yksinkertaisempi ja toimii synkronisesti. Synkroninen tarkoittaa sitä, että viestit kulkevat vuoronperään, eivätkä päällekkäin. Testikoodia **(Kuva 12.)** ei sen tarkemmin käydä tässä työssä läpi, koska se oli valmis ja vain testikäytössä työn alkuvaiheessa.

```
$socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
socket_connect($socket, $address, $port);
$message = file_get_contents('HelloWorld.xml');
socket_write($socket, $message, strlen($message));
while ($temp = socket_read($socket, 2048)) {
    echo $temp;
}
socket_close($socket);
```

Kuva 12. Referoitu logiikka alkuperäisestä testiskriptistä.

Työasemalle asennettiin EMVLumoAgent-ohjelma **(Kuva 13.)** ja maksupäätelaitte liitettiin USB-porttiin. Laitteen käyttöön vaadittavat ajurit asennettiin Windowsiin. EMVLumoAgent toimii siten, että se kuuntelee ennalta määritettyä TCP-porttia ja poimii sieltä EMVLumo-muotoiset XML-viestit. TCP-portin kuuntelu tapahtuu käynnistämällä EMVLumoAgentista löytyvä palvelin. Varsinkin alkuvaiheessa palvelinta joutui uudelleen käynnistämään usein virheellisten viestien ynnä muiden ongelmien takia. Ohjelma tekee sen onneksi vaivattomaksi. XML-viesteille oletusportti on 1234, jota myös työssä käytettiin.



Kuva 13. EMVLumoAgent-ohjelman oletusnäkymä.

Maksupäätelaite pitää joka kerta kun EMVLumoAgent-ohjelma avataan, alustaa. Tätä varten ohjelmassa on oma paneelinsa, josta voi myös hakea laitteelle päivitystä ja tarvittaessa sulkea yhteyden siihen. Myöhemmin selvisi, että tämä samainen alustaminen on hyvä tehdä välillä myös laitteen käytössä olon aikana. Siitä lisää myöhemmin. EMVLumoAgentin käynnistymisen yhteyteen on mahdollista määritellä parametreja, jolloin se esimerkiksi ottaa automaattisesti yhteyden maksupäätelaitteeseen käynnistymisensä yhteydessä sekä alustaa laitteen. Tämän ominaisuuden käyttö jätettiin testaamatta ja käyttämättä, ne soveltuvat paremmin tuotantokäyttöön.

”Hello World!” -testin tarkoituksena oli saada maksupäätelaitteen näytölle tulostumaan kyseinen teksti. Tämän piti olla mahdollista Luottokunnan dokumentaation mukaan funktiolla DoDisplay eli XML-muodossa: <DoDisplay>.

```
<EMVLumo xmlns="http://www.luottokunta.fi/EMVLumo">
    <DoDisplay>
        <Clear>true</Clear>
        <RowText>
            <Row>0</Row>
            <Inverse>True</Inverse>
            <Alignment>Left</Alignment>
            <Text>Hello</Text>
        </RowText>
        <RowText>
            <Row>0</Row>
            <Inverse>False</Inverse>
            <Alignment>Right</Alignment>
            <Text>World!</Text>
        </RowText>
    </DoDisplay>
</EMVLumo>
```

Kuva 14. XML-viesti, joka sisältää DoDisplay-funktion parametreineen.

Tämän kaiken alustustyön jälkeen päästiin koettamaan järjestelmää ensi kertaa. Testiskriptin ajoon käytettiin Linux-palvelinta, johon oli asennettuna PHP. Palvelimen komentoliittymään otettiin yhteys PuTTYllä. Testiskriptiä, kuten myös lopullista rajapintaa, käytetään komentoliittymän avulla. Tämä siksi, että rajapinnan virhetilanteet ovat helppo selvittää sellaisten sattuessa, kun ei tarvitse luottaa selaimeen. Asiasta sovittiin toimeksiantajan kanssa jo suunnitteluvaiheessa. Valitettavasti testiskripti ei tuntunut toimivan aluksi ollenkaan.

Maksupäätteen näytölle ei tulostunut mitään eikä testiskripti saanut luettua mitään vastausviestejä TCP-socketista. Testiskriptiin tehtiin pieniä muunnoksia, mutta tuloksetta. Lopulta turvauduttiin jälleen Luottokunnan rajapintadokumentaatioon, josta selvisi, että jokaisen yhteyden muodostuksen jälkeen maksupäätteelle pitää lähettää muuttuja-asetus, joka määrittää, katkaistaanko yhteys jokaisen viestin jälkeen vai ei. Muuttujan nimi on TCPDisconnect. Sen arvo on joko True, jolloin yhteys katkaistaan joka kerta tai False, jolloin yhteys pidetään päällä niin kauan kuin toisin sanotaan. Kyseinen muuttuja asetetaan lähettämällä SetTCPDisconnect-parametri XML-viestissä ennen minkään muun viestin lähettämistä **(Kuva 15.)**.

```
<EMVLumo xmlns="http://www.luottokunta.fi/EMVLumo">
    <SetTCPDisconnect />
    <Value>False</Value>
</EMVLumo>
```

Kuva 15. XML-viesti TCPDisconnect-muuttujan asettamiseksi.

Maksupäätteen näytöllä ei vieläkään näkynyt ”Hello World!” -tekstiä. Seuraavaksi otettiin käyttöön Wireshark-pakettihaistelijan, joka esiteltiin lyhyesti kappaleessa 5.7. Wiresharkin avulla saatiin selville, että TCP-paketit kyllä tulevat EMVLumoAgentille, mutta mitään ei silti tapahdu. Samoin selvisi, että maksupääte lähettää kyllä viestejä takaisin palvelimella olevalle testiskriptille, mutta se ei kuitenkaan niitä koskaan vastaanota. Tälle ei koskaan täysin selitystä keksitty, mutta vaikuttaisi siltä, että juuri tuon synkronisen socketin käyttö vaikutti asiaan. Liikenne ei toiminut vieläkään halutulla tavalla.

Luottokuntaan yritettiin olla yhteydessä useasti, mutta heillä tuntui olevan melkoinen ruuhka niin puhelimessa kuin sähköpostissakin. Lopulta heiltä saatiin kuitenkin kappaleessa 5.6 kuvailtu malliohjelma ja sen lähdekoodi. Malliohjelmasta löytyy graafinen käyttöliittymä, jossa eri ominaisuudet on jaettu omiin välilehtiinsä. Ohjelmalla otetaan ensin yhteys maksupäätteeseen, jonka jälkeen muita toimintoja voi ruveta käyttämään. Malliohjelmalla tehtiin täsmälleen sama ”Hello World!” -tekstin näytölle tulostaminen ja kaapattiin paketit jälleen Wiresharkilla. Ohjelma lähetti täysin saman viestin kuin testiskriptikin, mutta DoDisplay-tagien välissä olevat erikoismerkit olivatkin merkkikoodattu niin kutsutuiksi entiteeteiksi (engl. entities) **(Kuva 16.)**. Tehty XML-viesti muokattiin vastaavaksi ja testattiin uudelleen testiskriptillä, joka tehdyillä muutoksilla alkoi toimimaan. Viestit menivät perille ja maksupäätteen näytön ”Tervetuloa! Syötä kortti.”-tekstin tilalle ilmestyi ”Hello World!”. Myös palvelin päässä saatiin vastaanotettua viestejä, mutta ne olivat edelleen ennalta arvaamattomia ja epäjohdonmukaisia. Tämä onnistuminen kuitenkin todisti sen, että PHP:llä pystyy ainakin tietyin rajoituksin EMVLumoa käyttämään.

```
<EMVLumo xmlns="http://www.luottokunta.fi/EMVLumo">
    <DoDisplay>
        &lt;Clear&gt;true&lt;/Clear&gt;
        &lt;RowText&gt;
            &lt;Row&gt;0&lt;/Row&gt;
            &lt;Inverse&gt;True&lt;/Inverse&gt;
            &lt;Alignment&gt;Left&lt;/Alignment&gt;
            &lt;Text&gt;Hello&lt;/Text&gt;
        &lt;/RowText&gt;
        &lt;RowText&gt;
            &lt;Row&gt;0&lt;/Row&gt;
            &lt;Inverse&gt;False&lt;/Inverse&gt;
            &lt;Alignment&gt;Right&lt;/Alignment&gt;
            &lt;Text&gt;World!&lt;/Text&gt;
        &lt;/RowText&gt;
    </DoDisplay>
</EMVLumo>
```

Kuva 16. Uudelleen muotoiltu DoDisplay-viesti.

6.4 Luokkarakenne

Edellisen kappaleen lopussa saavutettujen onnistumisien myötä päästiin työstämään itse tämän opinnäytetyön aihetta eli uutta PHP-rajapintaa, joka mahdollistaa EMVLumon XML-viestin käytön nyt ja tulevaisuudessa. Nyt kun oli todettu, että rajapinnan kehitys PHP:llä pitäisi olla mahdollinen, ruvettiin hahmottelemaan itse rajapintaa. Päädyttiin oliopohjoiseen malliin, jossa jokainen rajapinnan osa-alue tulee omaan luokkaansa **(Kuva 17.)**. Oliopohjaisuus mahdollistaa modulaarisuuden, jolloin kehitettäviä luokkia on helppo käyttää tulevissa ja nykyisissä ratkaisuissa saumattomasti. Luokkien päälle oli tarkoitus kehittää ohjelmaskripti, jolla voi testata ja demonstroida rajapinnan toimintaa joko komentoliittymällä tai selaimella.

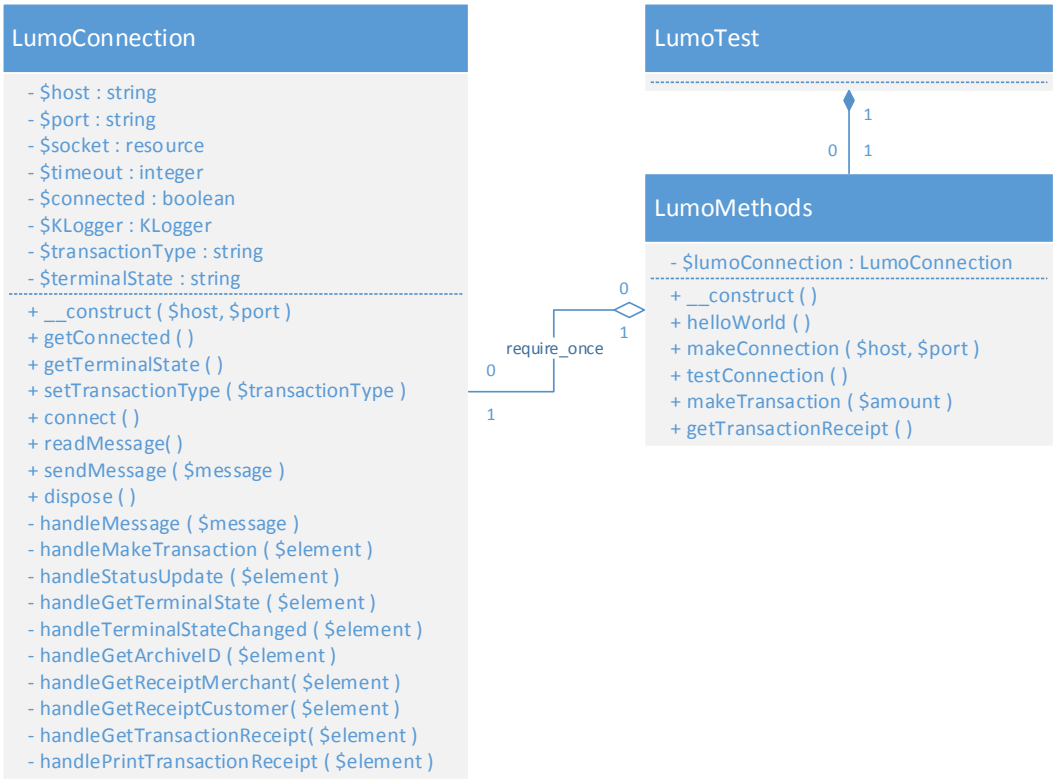
Vaikka minulla suhteellisen hyvä käsitys olikin PHP:n taipumisesta oliopohjaiseksi, tutkin silti aihetta ennen kuin aloin mitään ohjelmoimaan. PHP.netin artikkeleiden pohjalta päädyttiin sitten suunnittelemaan kehitettävän rajapinnan luokkarakennetta, joka tulisi olemaan oliopohjainen. Olio-ohjelmointia uudistettiin todella paljon PHP:n versioon 5, mutta kaikkia sen tuomia ominaisuuksia en kuitenkaan lähdetty hyödyntämään. Näihin kuuluu muun muassa nimiavaruudet, jotka ovat hyvin oleellisia esimerkiksi Java-ohjelmoinnissa. Kehitettävän rajapinnan laajuus oli kuitenkin sitä luokkaa, että päätettiin vielä jättää uutuudet pois ja keskittyä olennaiseen.

Luokkajakoon hyödynnettiin jonkun verran Luottokunnan tarjoaman malliohjelman lähdekoodin rakennetta, koska se vaikutti olevan looginen ja hyvin sovellettavissa myös kehityksen tarkoituksiin. Suoraa kopiointia ei voitu lähteä tekemään, koska malliohjelma on tehty C#:lla. Rajapinta päädyttiin jakamaan kahteen luokkaan sekä ohjelmaskriptiin, joka toteuttaa kaikki tarvittavat toiminnot joko komentoliittymällä tai selaimella:

- **LumoConnection**-luokka sisältää kaiken tarvittavan yhteyden muodostamiseen rajapinnalta EMVLumoon. Tämän lisäksi luokasta löytyvät funktiot viestin lähettämiseen ja vastaanottamiseen. Muodostetun yhteyden tuhoamiseen liittyvä funktio on myös tässä luokassa. Luokasta löytyy myös käsittelijäfunktiot vastaanotetuille viesteille. Esimerkiksi, jos vastaanotettu viesti on StatusUpdate-viesti, luetaan ja lokitetaan se, jonka jälkeen jatketaan portin kuuntelua.

- **LumoMethods**-luokassa ovat funktiot, joita kutsutaan päällimmäisestä ohjelmaskriptistä. Luokan funktiot ovat käytännössä kirjoitus- ja lukumetodien vuorotteluparejа. Kaikki XML-viestit ovat staattisesti määriteltyinä parametreiksi, joten niitä ei tarvitse tietää ylemmällä tasolla.

- **LumoTest**-ohjelmaskripti ei ole varsinainen luokka. Se on rajapintaa hyödyntävä ohjelmaskripti, jonka voi korvata toisella luokalla tai skriptillä. Sen päällimmäinen tarkoitus on kutsua tarvittavia funktioita LumoMethods-luokasta joko komentoliittymästä tai selaimesta käytettynä. Selainta käytettäessä hyödynnetään erillisiä lisäresursseja, jotka mahdollistavat toimivan ja käyttäjäystävällisen käyttöliittymän. Tämä skripti ei ole varsinaisesti osa rajapintaa vaan se demonstroi, miten kehitettyä rajapintaa voi hyödyntää.



Kuva 17. Luokkakaavio rajapinnan luokista ja ohjelmaskriptistä.

6.5 Toteutus

Kun rajapinnan luokkia lähdettiin kirjoittamaan, täytyi vielä pohtia ja tutkia, miten socketit toimivat PHP:llä. Alkuperäinen ratkaisu **(Kuva 12.)** tuotti edelleen ongelmia viestin vastaanottamisessa ja lähettämisessä asynkronisesti eli rinnakkain. Samoin itse PHP:n toimivuus asynkronisesti täytyi tutkia vielä tarkemmin. Tämä kaikki asynkronisuus siksi, että EMVLumo on tarkoitettu toimivaksi siten, että se lähettää viestejä kyselemättä, mutta myös vastaanottaa viestejä samaan aikaan ja toisin päin. Luottokunnan toimittama malliohjelma toimii moitteetta näin, koska C# mahdollistaa toimintojen jakamisen moneen säikeeseen (engl. threads), jolloin esimerkiksi yksi säie kuuntelee, toinen lähettää ja kolmas päivittää käyttöliittymää. Tämä oli kärjistetty esimerkki malliohjelman toiminnasta, mutta sen olikin tarkoitus antaa vain yleiskuva sen toiminnasta. Vastaavanlainen säietys ei ole mahdollista PHP:llä, joten rajapinnan kehitykseen täytyi keksiä jokin toinen ratkaisu.

Tutkimalla useita artikkeleita (/2/, /4/, /12/, /15/, /19/) PHP:n socketien toiminnasta asynkronisesti, päädyttiin siihen tulokseen, että helpoiten ja parhaiten se onnistuisi

stream_socket_client-funktiolla **(Kuva 18.)**, joka löytyy PHP:n versiosta 5. Kyseisellä funktiolla on mahdollista luoda socket-resurssi, jonka voi asettaa ei-estäväksi (engl. non-blocking), jolloin se osaa lähettää ja vastaanottaa TCP-paketteja asynkronisesti. Tämä ratkaisi ongelman, jossa viestit hukkuivat TCP-puskuriin. Ratkaisun kehittäminen vei oman aikansa ja testaamissyklinsä, mutta se lopulta toimi halutulla tavalla, mikä oli helpotus.

```
// Try to declare new stream socket client resource using tcp with given host
address and port number.
$this->socket = stream_socket_client("tcp://" . $this->host . ":" . $this->port,
$errno, $errstr, ini_get("default_socket_timeout"), STREAM_CLIENT_CONNECT |
STREAM_CLIENT_ASYNC_CONNECT);
```

Kuva 18. Asynkronisen socket-resurssin luontifunktio.

Kun oli onnistuttu yhteydenmuodostamisen, viestien lähettämisen ja vastaanottamisen kanssa, tehtiin funktio, joka osaa pilkkoa vastaanotetut viestit kappaleessa 5.4 mainitun tyhjän merkin perusteella. Tämän jälkeen viestit ohjataan yksitellen viestikäsittelijälle, joka jälleenohjaa viestit vielä erillisille käsittelyfunktioille **(Kuva 19.)**, jotka kirjoittavat tapahtuman lokiin sekä muuttavat globaalin keepReading-muuttujan trueksi tai falseksi riippuen siitä, pitääkö kuuntelua jatkaa. Kuuntelun jatkamisen logiikka on ennalta määrätty ja seuraavaksi kerrotaankin, miksi näin on.

```
/**
 * Handler function for StatusUpdate message.
 *
 * @param       SimpleXMLElement    $element    A XML element which has
StatusCode and StatusInfo elements.
 */
private function handleStatusUpdate($element) {
    $this->keepReading = true;
    try {
        // Fetch the status code from the read XML.
        $statusCode = $element->StatusCode;
        // Fetch the status info from the read XML.
        $statusInfo = $element->StatusInfo;
        // Checks if the $terminalState is null or empty.
        if ($statusCode != null && $statusInfo != null) {
            //echo "StatusInfo: ".$statusInfo."\n";
        } else {
            // Throw an exception because the message is either null or empty.
            throw new Exception("Message was undefined.");
        }
    } catch (Exception $e) {
        // Log the catched exception.
        $this->KLogger->logError("Error occurred while handling the StatusUpdate
message. " . $e->getMessage());
    }
}
```

Kuva 19. StatusUpdate-viestin käsittelyfunktio.

Toinen oleellinen ongelma, PHP:n synkronisuus, oli vielä ratkaisematta. Tämän kappaleen alussa mainittiin, miten Luottokunnan malliohjelma toimii hienosti asynkronisesti tehden kaikkea yhtäaikaisesti, mutta vastaava ei ole mahdollista PHP:llä suoraan. Asiaa tutkittiin jonkin verran ja käytännössä on mahdollista ajaa esimerkiksi useita shell-komentoja palvelimella, jotka toteuttaisivat eri PHP:n funktioita. Tämä ajatus kuitenkin rajattiin opinnäytetyön alueesta pois nopeasti ja tyydyttiin yrittämään jotain muuta. PHP ei ehkä ole kaikkein monipuolisin kieli, mutta ainakin se pakottaa käyttäjänsä luovuuteen, kuten tämän työn kohdalla kävi. EMVLumon viestikeskustelua tarkasteltiin tarkemmin ja huomattiin, että se lähettää aina toimintojen suorituksen lopuksi TerminalState-viestin, joka kertoo missä tilassa maksupäätelaite on. Maksupäätteen eri tiloja on kahdeksan:

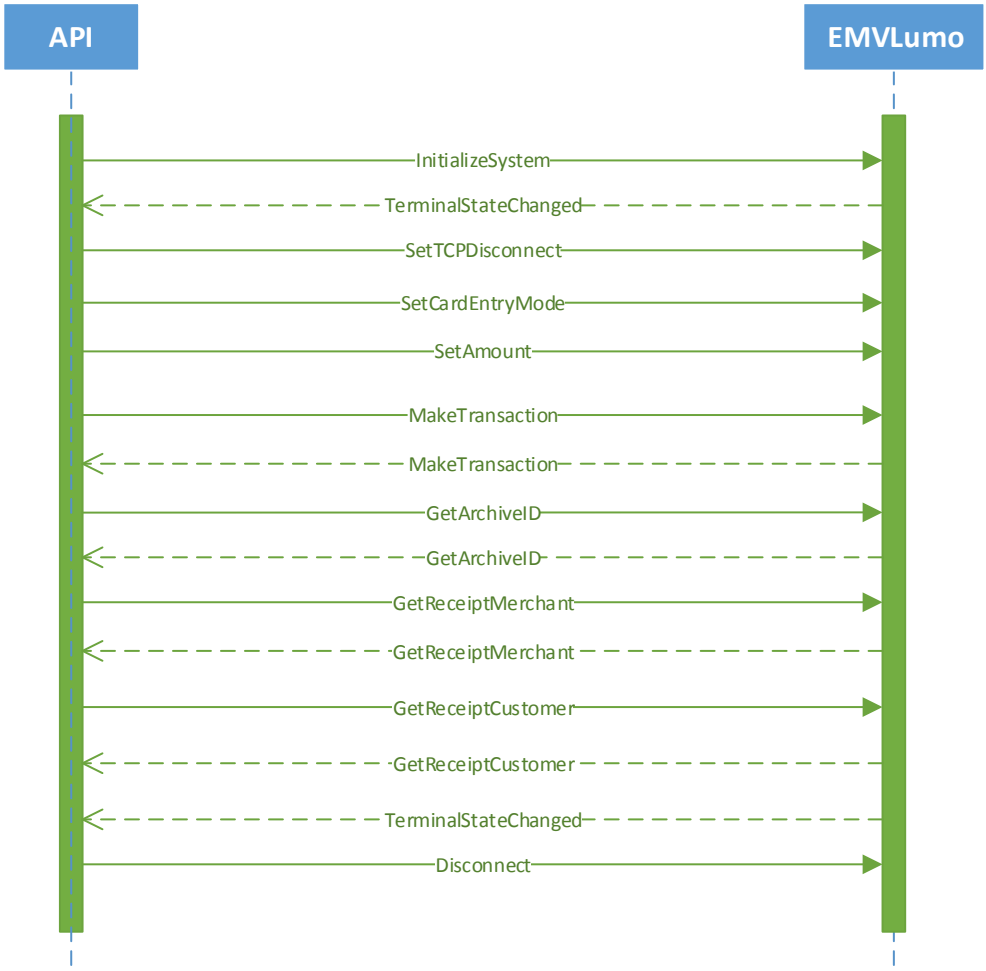
- NotInitialized = 0, maksupäätelaitetta ei ole alustettu eikä se ole valmis käytettäväksi.
- TerminalRebooting = 1, maksupäätelaite käynnistyy uudelleen.
- TerminalReady = 2, maksupäätelaite on alustettu ja valmiina käyttöön.

- TerminalSendingTransactions = 3, maksupäätelaite lähettää maksutapahtumia.
- TerminalInIdleState = 4, maksupäätelaite on valmis aloittamaan maksutapahtuman.
- TerminalSendingTransactionsDone = 5, maksupäätelaite on lähettänyt maksutapahtumat.
- TerminalInitialized = 6, maksupäätelaite on alustettu ja valmiina käyttöön.
- ProcessingTransaction = 7, maksupäätelaite käsittelee maksutapahtumaa. /8/

Nämä tilaviestit keksittiin valjastaa rajapinnan käyttöö. PHP:tä ei voi käyttää asynkronisesti, mutta jotenkin oli tiedettävä, mitä tehdään seuraavaksi ja mitä viimeksi on tapahtunut. Nyt tiedettiin, että voi lähettää ja vastaanottaa viestejä esimerkiksi maksutapahtuman aikana niin kauan, että maksupäätelaite ilmoittaa olevansa joko tilassa 2 tai 4. Tämän kaiken perusteella rakennettiin globaaliin keepReading-muuttujaan perustuva logiikka. Toimeksiantajan kanssa keskusteltiin kyseisen muuttujan käytöstä ja sen sekavuudesta. Loppujenlopuksi luokan rakennetta saatiin selvennettyä hieman, vaikka keepReading-muuttuja jouduttiinkin säilyttämään.

Rajapinnan perustoiminnallisuus oli valmis. Testeissä kuitenkin ilmeni jonkin asteisia viiveitä rajapinnan ja maksupäätteen välisessä käytössä. Välillä saattoi kestää viidestä sekunnista minuuttiin ennen kuin maksupäätelaite reagoi rajapinnan lähettämiin kutsuihin. Aluksi tämän epäiltiin johtuvan rajapinnan synkronisuudesta tai viestien hukkumisesta TCP-puskuriin. Sitkeän testaamisen seurauksena selvisikin, että vaikka yhteys katkaistaan rajapinnalla ja socket-resurssi tuhotaan, jää EMVLumo edelleen kuuntelemaan luotua yhteyttä. Yhteyden sulkemisfunktioon lisättiin vielä yksi viestilähetys ennen yhteysresurssin tuhoamista. Tuo viesti pitää sisällään komennon, jolla EMVLumo sulkee nykyisen yhteyden sen ja rajapinnan väliltä. Komento on yksinkertaisesti Disconnect. Rajapinta toimi lisäyksen jälkeen moitteetta. Varmuudeksi kehitetyn rajapinnan yhdistämisfunktioon lisättiin vielä InitializeSystem-komento, jolla maksupäätelaite aina yhteydenmuodostuksen jäl-

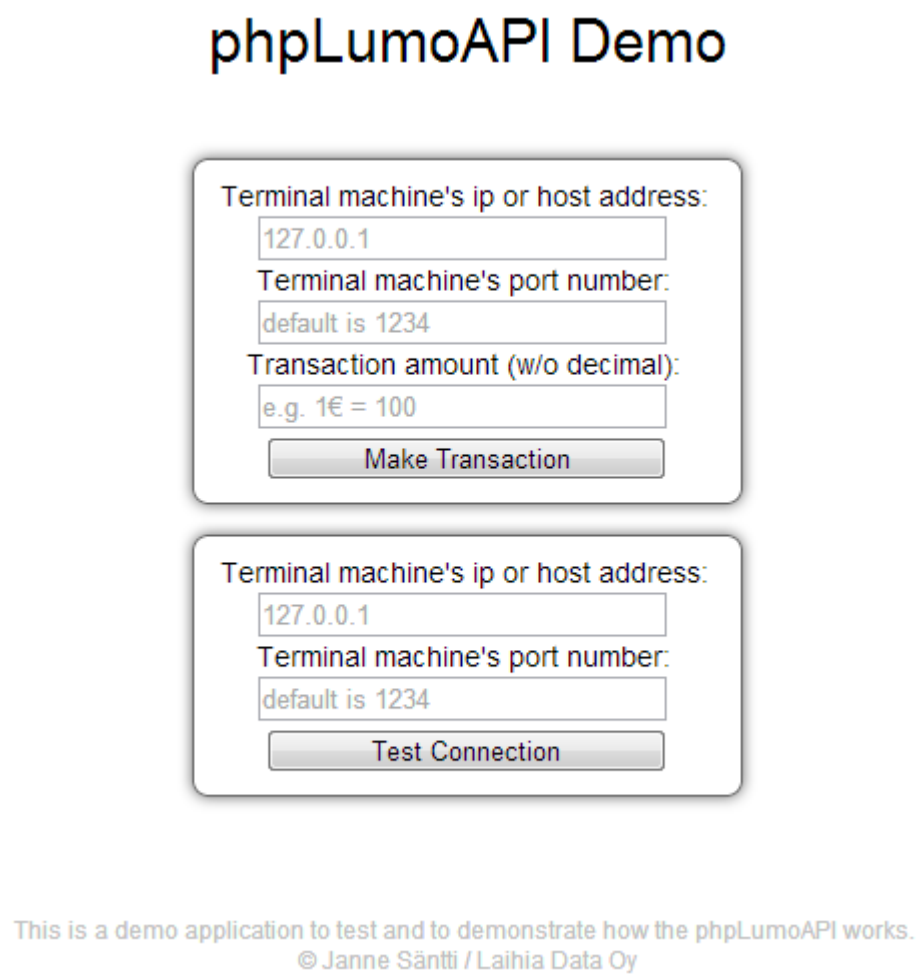
keen alustetaan. Seuraavalla sivulla olevassa kuvassa **(Kuva 20.)** näkyy maksutapahtuman sekvenssikaavio kehitetyllä rajapinnalla silloin, kun tapahtuman aikana ei ilmene ongelmia.



Kuva 20. Täydellisen maksutapahtuman sekvenssikaavio EMVLumo-viestien muodossa.

Kehitettävän oli enää selaindemo. Toimeksiantajan kanssa sovittiin, että komentonäkymädemon lisäksi tehdään yksinkertainen selaimessa toimiva demo. Tämän avulla rajapintaa pystyisi esittelemään ympäristössä ja käyttöliittymällä, jotka muistuttaisivat hyvin paljon potentiaalista tuotantovastinettaan. Tähän tarkoitukseen kehitettiin yksinkertainen HTML-sivu, joka muodostetaan LumoTest-ohjelmaskriptiä apuna käyttäen. HTML-sivu toteuttaa kaksi kevyttä lomaketta **(Kuva**

21.), joihin täytetään tietokoneen IP ja porttinumero, johon maksupäätelaite on liitettynä. Jos kyseessä on maksutapahtuma, annetaan myös maksun summa. Sivu lähettää AJAX-pyyntönä halutun toiminnon parametreineen ja ilmoittaa lopputuloksen lomakkeen alapuolelle.



Kuva 21. Demosivu.

6.6 Tietoturva

Tietoturva nousee yleensä suureksi kysymykseksi, oli kyseessä sitten verkkosovellus tai ei. Kun kyseessä on verkkosovellus, kuten kehitetyn rajapinnan tapauksessa, tietoturvan rooli korostuu entisestään. Yleensä kysymyksessä ovat käyttäjien henkilökohtaiset tiedot ja maksutapahtuman tapauksessa rahat, jotka ovat vaarassa jou-

tua vääriin käsiin. Tästä kaikesta huolimatta kehitetyn rajapinnan kohdalla tietoturva-ajattelu jäi pitkälti taka-alalle. Tarkoitus ei ollut tehdä rajapintaa, joka ”toivottavasti kestää”, vaan tietoturva tehdään jo alemmalla tasolla, kuten kappaleessa 5.11 kerrottiin. Kaikki Laihia Data Oy:n yritysyhteydet käyttävät palomuuria ja VPN-tunnelointia. Jos palvelin on asiakkaan omissa tiloissa, tapahtuu liikennöinti lähiverkossa.

EMV-standardi määrittelee oman tietoturvan, joka täytyy toteutua maksutapahtumien ja raporttien lähettämisessä Luottokunnalle maksupäätelaitteesta. Tämä liikennöinti tapahtuu kuitenkin siten, ettei kehitetty rajapinta ole kytköksissä näihin siirtoihin millään tavalla. EMV:n tietoturvasta FKL kirjoittaa seuraavasti:

> ”Tapahtumien lähetys tulee suojata Finanssialan Keskusliiton pankkiturvastandardin (PATU) määrittämällä tavalla. Maksunsaaja saa suojauksessa käytettävät siirtoavaimet omasta pankistaan.
>
> Vaatimuksena on, että kaikki pankkiaineistoja sisältävä tietoliikenne langattomissa ja julkisissa Internet verkoissa sekä ulkomaan yhteyksillä salakirjoitettaan. Salakirjoitus on toteutettava vahvalla tekniikalla. WLAN-, GSM- ja GPRS-järjestelmien käyttämät salausmenetelmät eivät sellaisenaan ole riittäviä ja salauksen on ulotuttava siirtoyhteyden päästä päähän.” /9/

Kuten lainauksesta ilmenee, ovat määritykset itse rahaliikenteelle hyvin tiukat. Aiheeseen ei kuitenkaan tämän syvemmin paneuduta tämän opinnäytetyön rajoissa.

Kehitettävää rajapintaa on mahdollista muokata ja parantaa siten, että TCP-protokollan käyttö korvataan SSH:lla, jolloin rajapinnan käyttöönkin saataisiin haluttua turvallisuutta. SSH kuitenkin toistaiseksi jätettiin pois työstä, koska se olisi aiheuttanut suuren määrän lisätutkimusta alkutaipaleella, joka oli jo valmiiksi haastava alkuperäisten haasteidensa johdosta.

7 TESTAUS

Rajapinnan suunnitteluvaiheessa keskusteltiin alustavasti komentonäkymäliittymästä, jolla rajapintaa ja sen tilaa voi testata. Demosivusta alettiin puhua vakavasti vasta kehityksen loppuvaiheessa. Komentonäkymää käytetään esimerkiksi PuTTYllä. Parametreina rajapinnalle täytyy antaa tietokoneen IP, jossa maksupäätelaite on kiinni; samaisen koneen porttinumero; komento, joka halutaan tehdä, esimerkiksi testConnection. Jos komento on maksutapahtuma, kysyy ohjelmaskripti erikseen maksutapahtuman summaa. Tätä tekniikka käytettiin käytännössä kaikkeen rajapintaan liittyvään testaamiseen alusta lähtien aina projektin loppuun saakka. Skripti on virhetilanteessa helppo lopettaa komentonäkymällä eikä tarvitse huolehtia selaimen tuomista haasteista.

LumoConnection-luokassa tapahtuvat tapahtumat lokitetaan kattavasti tekstitiedostoihin päivämäärän mukaan, jolloin on helppo käydä tarkistamassa missä vika on. Tämän lisäksi PHP tarjoaa itsessään todella kattavan, muokattavan ja yksityiskohtaisen virheraportoinnin, jonka avulla esimerkiksi moni syntaksivirhe löytyi ja korjaantui helposti. PHP:lle olisi olemassa esimerkiksi PHPUnit-yksikkötestausrunko (engl. unit testing software framework), jolla testataan koko kehityksen ajan ohjelmakoodin eri osia omina yksikköinään. PHPUnitin avulla on mahdollista havaita virheet koodissa jo varhaisessa vaiheessa. Mutta kuitenkin rajallisen ajan ja aiemman kokemuksen puutteesta, en kyseistä tekniikka ei lähdetty kehityksessä soveltamaan.

Ennen kuin rajapinta voidaan tuoda tuotantokäyttöön, täytyy sitä vielä rutkasti testata ja kehittää. Pitää pystyä varmistamaan, että käyttäjä saa aina tiedon suoraan selaimeensa, jos jotain ongelmia ilmenee. Myös niin kutsuttu rinnakkaiskäyttö täytyy tutkia ja implementoida paremmin. Rinnakkaiskäytöllä tarkoitetaan tilannetta, jossa kaksi eri käyttäjää käyttävät samanaikaisesti yhtä maksupäätelaitetta eri selaimista. Kehitysvaiheessa tämäkin testattiin Laihia Data Oy:n omassa verkossa ja se toimi yllättävänkin moitteetta ottaen huomioon, että tällainen ominaisuus ei ollut kehityksen pääpainona.

Itse maksutapahtumien testaaminen maksupäätelaitetta käyttäen vaatii tietenkin jonkin näköisen sirukortin. Testaamiseen käytettiin Nordea-pankin toimittamia testikortteja, joita oli useaa eri tyyppiä: debit, credit ja näiden yhdistelmiä sekä variaatioita. Näiden korttien joukossa on myös niin kutsuttuja virhekortteja, jotka tuottavat erinäisiä virhetilanteita maksutapahtuman aikana. Pääasiallisesti kehitetyn rajapinnan testaamisen yhteydessä käytettiin vain muutamaa näistä korteista. Korteilla, jotka eivät tuota virhetilanteita muutoin kuin käyttäjän virheestä. Tämä siksi, että korttien virhetilanteet käsitellään EMVLumoAgentin dialogeilla eli kehitetty rajapinta ei ota niihin kantaa.

8 YHTEENVETO

Opinnäytetyön tuloksena tuotettu rajapinta suunniteltiin ja kehitettiin suhteellisen nopealla aikataululla ottaen huomioon, että vastaavaa ratkaisua en ainakaan henkilökohtaisesti ole kuullut käytettävän. Kehitetty rajapinta mukailee nykytrendin mukaista ajatusta siitä, miten moni palvelu ja sovellus siirtyvät pilvipalveluihin ja verkon yli käytettäviksi. Vain tulevaisuus näyttää, mitkä kaikki palvelut sinne täysin siirtyvät.

Kehityksessä oli omat haasteensa ja ongelmansa, kuten kunnollisessa ohjelmistoprojektissa kuuluukin olla. Varsinkin PHP:n rajoittuneisuus moniajon suhteen tuotti ongelmia. Myös Luottokunnan EMVLumo vaati totuttelua ja suuren määrän yrityksiä toimiakseen halutulla tavalla. Kehitystä helpotti se, ettei tietoturva ollut isoin kysymys rajapintaa kehitettäessä, koska se toteutetaan alemmalla kerroksella.

Rajapinta toteutettiin siten, että se käyttää hyväkseen Luottokunnan EMVLumoAgent-palvelinohjelman dialog-rutiinia, jolloin maksupäätelaitteen käyttö rajapinnan näkökulmasta helpottui huomattavasti. Rajapinta olisi mahdollista myös toteuttaa siten, ettei dialogeja käytettäisi vaan jokainen tapahtuma kierrätettäisiin rajapinnan kautta, jolloin esimerkiksi käyttäjän selaimelle voisi tulostaa samat vaiheet, jotka näkyisivät dialogeilla. Jääköön ajatus vielä toistaiseksi muhimaan ja odottamaan päivää, jolloin selain on se ainut oikea ratkaisu.

Kehitettävää jäi vielä paljon ennen kuin rajapinta on täysin tuotantokäyttöön soveltuva, mutta se tarjoaa sellaisenaan jo oivan alustan jatkokehitykselle. Nykyisiä käyttötapauksia on helppo mukauttaa uusiin vastaaviin. Myös virheenkäsittelyä ja -sietokykyä on vielä parannettava. Rajapinta kuitenkin toimii PHP:llä, mikä oli se alkuperäinen suurin kysymys.

Kaiken kaikkiaan kehitysprojekti oli vaiheittain todella haastava, mutta niistä opittiin paljon kaikkea uutta - varsinkin PHP:n saralta. EMV myös standardina oli tuntematon, mutta nyt siitäkin on suhteellisen kattava käsitys. Rajapintaa tullaan varmasti käyttämään tavalla tai toisella tulevissa ja nykyisissäkin ratkaisuissa. Parannettavaa omasta mielestäni jäi koodin kommentoinnin suhteen. Lopputulokseen

olin tyytyväinen phpDocumentorin implementoinnin jälkeen. Suurin osa kommentoinnista tehtiin vasta jälkikäteen, mikä oli melko työlästä.

LÄHTEET

/1/ A Guide to EMV. 5/2011. Viitattu 23.4.2013. EMVCo. http://www.emvco.com/best_practices.aspx?id=217

/2/ Asynchronous PHP calls? Viitattu 2.4.2013. Stack Overflow, Stack Exchange Network. http://stackoverflow.com/questions/124462/asynchronous-php-calls

/3/ Bloch, J. 2008. Effective Java (2nd edition). Addison-Wesley.

/4/ Calvin. 2013. How to Make Async Requests in PHP. Viitattu 2.4.2013. Segment.io. https://segment.io/blog/how-to-make-async-requests-in-php/

/5/ EMV Integrated Circuit Card Specifications for Payment Systems. Version 4.3. 11/2011. Viitattu 23.4.2013. http://www.emvco.com/specifications.aspx?id=223

/6/ EMV: FAQ. Viitattu 27.4.2013. Smart Card Alliance. http://www.smartcardalliance.org/pages/publications-emv-faq

/7/ EMVLumo – Usein kysyttyjä kysymyksiä. Viitattu 18.4.2013. Luottokunta, Nets Oy. http://www.screenway.com/emv/emvlumo/Usein%20Kysyttyja%20Kysymyksia.pdf

/8/ EMVLumo API Specification Document. 03/2013. Viitattu 29.4.2013. Luottokunta, Nets Oy. http://www.screenway.com/emv/emvlumo/EMVLumo%20API.pdf

/9/ EMV-maksupäätejärjestelmä – toiminnallinen kuvaus. Versio 4.2. 21.10.2011. Viitattu 18.4.2013. Finanssialan Keskusliitto ry. http://www.fkl.fi/teemasivut/sepa/tekninen_dokumentaatio/Dokumentit/EMV_maksupaatejarjestelma.pdf

/10/ Extensible Markup Language (XML). Viitattu 29.4.2013. World Wide Web Consortium (W3C). http://www.w3.org/XML/

/11/ FAQ. Viitattu 23.4.2013. EMVco. http://www.emvco.com/faq.aspx

/12/ Furlong, F. 2005. Guru – Multiplexing. Viitattu 2.4.2013. http://wezfurlong.org/blog/2005/may/guru-multiplexing/

/13/ Maksupäätteet PC-kassaympäristöön. Viitattu 9.5.2013. Luottokunta, Nets Oy. http://www.luottokunta.fi/Tuotteet/Maksupaatteet/Maskuspaatteet-PC-ymparistoon/

/14/ Nets järjestelmätoimittajien kumppanina. Viitattu 18.4.2013. Luottokunta, Nets Oy. http://www.luottokunta.fi/Tuotteet/Maksupaatteet/Jarjestelmatoimittajille/

/15/ Newson D. 2012. Methods for asynchronous processes in PHP. Viitattu 2.4.2013. http://davenewson.com/dev/methods-for-asynchronous-processes-in-php

/16/ Object Interfaces. Viitattu 21.4.2013. PHP.net. http://php.net/manual/en/language.oop5.interfaces.php

/17/ PHP Manual. Viitattu 28.4.2013. PHP.net. http://www.php.net/manual/en/index.php

/18/ Progress Software History. 2013. Viitattu 18.4.2013. Progress Software Corporation. http://www.progress.com/en/whoweare/history.html

/19/ stream_socket_client-funktion manuaalisivu. Viitattu 2.4.2013. PHP.net. http://php.net/manual/en/function.stream-socket-client.php

/20/ Twitter Developers – Documentation. Viitattu 24.4.2013. Twitter, Inc. https://dev.twitter.com/docs

/21/ What Is an Interface. Viitattu 21.4.2013. The Java Tutorials. Oracle. http://docs.oracle.com/javase/tutorial/java/concepts/interface.html

/22/ What is EMV? Viitattu 23.4.2013. CreditCall Ltd. http://www.level2kernel.com/emv_guide.html

/23/ Virtual private network -kuva. Viitattu 3.5.2013. Wikipedia. http://en.wikipedia.org/wiki/File:Virtual_Private_Network_overview.svg

/24/ Yritysesittely Lahia Data Oy:n verkkosivuilla. Viitattu 18.4.2013. http://www.laihiadata.fi/