OULUN SEUDUN
AMMATTIKORKEAKOULU

Teemu Sirviö

PYTHON 3 JA WINDOWS-JÄRJESTELMÄN TIEDOT

Tilanseurantaohjelma asiakaskoneelle

PYTHON 3 JA WINDOWS-JÄRJESTELMÄN TIEDOT

Tilanseurantaohjelma asiakaskoneelle

Teemu Sirviö
Opinnäytetyö
Kevät 2011
Tietojenkäsittelyn koulutusohjelma
Oulun seudun ammattikorkeakoulu

TIIVISTELMÄ

Oulun seudun ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma

Tekijä: Teemu Sirviö
Opinnäytetyön nimi: Python 3 ja Windows-järjestelmän tiedot
Työn ohjaaja: Pekka Ojala
Työn valmistumislukukausi ja -vuosi: Kevät 2011 Sivumäärä: 32

Opinnäytetyön toimeksiantajana toimii yritys, jonka vastuulla on asiakkaittensa tietokoneiden ylläpitotyö. Työtä helpottamaan ja kustannuksia laskemaan toimeksiantaja päätti toteuttaa asiakkaiden tietokoneelle asennettavan valvontaohjelmiston. Toimeksiantaja halusi toteuttaa ohjelmiston opinnäytetyönä ja Python-ohjelmointikieltä käyttäen.

Työssä käytettiin Python-kielen versiota 3, jonka toimintaa laajennettiin PyWin32- ja Python WMI -moduuleilla. Näistä ensimmäinen tuo tulkin käyttöön Windows-spesifisiä toiminnallisuuksia ja jälkimmäinen yksinkertaistaa Microsoftin tekemän Windows Management Instrumentation -palvelun käyttöä Python-koodista käsin.

Ohjelmistotyö jaettiin vaatimusmäärittelyssä opinnäytetyön puitteissa toteutettaviin ja myöhemmin opinnäytetyöstä erillisenä työnä toteutettaviin toiminnallisuuksiin. Asiakaskoneen tiedot koostettiin XML-muotoiseen raporttiin, joka on jaettu kolmeen osaan: järjestelmän tietoihin, tietokoneelle asennettujen ohjelmistojen tietoihin sekä lokitietoihin.

Raportin tiedot haettiin käyttäen Python WMI:n luokkia ja niiden metodeja ja niistä koostettiin XML-muotoinen raportti käyttäen Pythonin xml.etree.ElementTree-moduulia. Ohjelma käyttää http.client-moduulia tiedon välittämiseksi verkon yli ja configparser-moduulia tarvittavien ohjelman asetusten lukuun ja tallennukseen. Valmis ohjelmisto paketoitiin cx_Freexe-ohjelmalla asiakaskoneella ajettavaksi tiedostoksi, joka sisältää tarvittavat Python-moduulit ja on käytettävissä ilman paikallista Python-tulkin asennusta.

Työn tuloksena saatiin aikaan toimeksiantajan tarvitsema ohjelmisto ja voitiin todeta Python-kielen olevan toimiva ohjelmointikieli myös Windows-ohjelmointiin, vaikkakin ohjelman muistinkäyttö todettiin melko korkeaksi. Lisäksi useiden eri laajennosten käyttäminen voi olla hankalaa mahdollisten yhteensopivuusongelmien johdosta ja siksi jatkokehityksessä voitaisiin harkita käytettäväksi jotain toista Python-toteutusta tai ohjelmointikieltä.

Asiasanat: Python, Windows 7, Windows XP, XML, tiedonhaku

ABSTRACT

Oulu University of Applied Sciences
Tietojenkäsittelyn koulutusohjelma

Author: Teemu Sirviö
Title of thesis: Python 3 and Windows system information
Supervisor: Pekka Ojala
Term and year when the thesis was submitted: Spring 2011 Number of pages: 32

Client of the thesis was a company whose responsibility is the maintenance of its customer's computers. To ease the work and to lower the expenses, it was decided to produce surveillance software to be installed on to the customer computers. The client wanted the software to be conducted as a thesis using the Python programming language.

The work was done using version 3 of the Python programming language, whose functionality was extended with the PyWin32- and Python WMI -modules. Former bringing Windows specific functionalities and latter simplifying the usage of Microsoft's Windows Management Instrumentation service from Python code.

In the requirements specification the thesis was divided into use-cases and functions to be produced within the thesis and those to be produces afterwards. The computer systems data was composed into a XML document that was in three parts: system information, installed programs information and log data.

All the data was fetched using the classes and methods in the Python WMI module. The data was then composed into XML using the xml.etree.ElementTree module found in the Python's standard library. The program uses http.client module to send the data to the server and configparser module to read and write user settings. cx_Freese program was then used to turn the finished software into an executable for Windows. The executable can then be used with out a Python interpreter and includes the necessary Python modules.

As a result the client got the software it needed and Python was found to be a good alternative for Windows based programming albeit the software was noted to be quite memory consuming. Also using a lot of different extensions can be difficult concerning the compatibility of different versions of the interpreter and the extensions themselves, therefore in further development another implementation of Python of another programming language could be considered.

Keywords: Python, Windows 7, Windows XP, XML, data collection

SISÄLLYS

1 JOHDANTO

TCO (Total Cost of Ownership) termillä tarkoitetaan tuotteen kokonaishintaa, joka saadaan laskemalla sen aiheuttamat välilliset ja välittömät kustannukset. Tietotekniikan alalla nämä kustannukset käsittävät mm. laitteiston ja ohjelmiston hankintakustannukset, tietohallinnon ja tuen aiheuttamat kustannukset ja tietoliikenteestä ja loppukäyttäjistä aiheutuvia kustannuksia. Kuluihin voidaan laskea myös työaika, joka menetetään esimerkiksi kun laitteisto ei ole toimintakunnossa tai kun työntekijöitä joudutaan kouluttamaan. (Gartner 2011, hakupäivä 6.5.2011.)

Yritysten pitää etsiä keinoja kulujen vähentämiseksi ja yksi keino niiden vähentämiseksi on tietokonelaitteiston ylläpitotyön ulkoistaminen. Tämän opinnäytetyön toimeksiantajana toimi yritys, jonka vastuulle on annettu asiakasyritysten tietokoneiden ja tietoverkkojen ylläpito. Toimenkuvaan kuuluu seurata laitteiston kuntoa säännöllisesti ja huoltaa niitä ongelmatilanteissa. Tämän toimenkuvan pohjalta syntyi ajatus asiakaskoneen seurantaohjelmistosta, joka voisi välittää tietoa tietokoneen toiminnasta toimeksiantajalle. Ohjelmisto voisi helpottaa ylläpidon työtä välittämällä tarpeellinen informaatio toimeksiantajan saataville ilman erillistä käyntiä asiakkaan tiloissa.

Ohjelmisto tulisi koostumaan kahdesta osasta: lähettävästä asiakasohjelmistosta ja vastaanottavasta palvelinohjelmistosta. Toimeksiantaja oli kiinnostunut Python-kielen mahdollisuuksista ja päätti teettää ohjelmiston opinnäytetyönä Python-ohjelmointikieltä käyttäen. Vaikka Python on verrattain uusi kieli (kehitys on alkanut vasta 90-luvulla), on se monipuolinen ja helposti opittava kieli (Kasurinen 2009, 7). Python-kielen kehitys on edennyt nopealla tahdilla, ja sen kehityksessä on vastikään siirrytty uudelle aikakaudelle, kun siitä julkaistiin versio numero 3. Uusi versio sisältää muutoksia kielen syntaksissa, ja se katkaisi kielestä taaksepäin yhteensopivuuden edellisten versioiden kanssa (Kasurinen 2009, 7-8). Michael Ogawa on tehnyt Python-kielen kehityksestä videon, joka on nähtävillä verkko-osoitteessa: http://www.vimeo.com/1093745. Video kuvaa kielen kehitystä vuodesta 1990 vuoteen 2006 saakka ja esittää kehityksen kulkua siihen luotujen tiedostojen, dokumenttien ja moduuleiden avulla.

Python-kielen syntyaikaan käynnistyi myös kehitys Windows Management Instrumentation (WMI) -palvelulle. WMI on Microsoft Windows -käyttöjärjestelmän ylläpitoa varten tehty palvelu, jonka

avulla voidaan tehdä ohjelmia ja komentosarjoja muun muassa hakemaan dataa käyttöjärjestelmän osista, ohjelmistoista ja laitteistosta. Palvelu tarjoaa yhtenäisen rajapinnan Windows-järjestelmän tietoihin ja sitä voidaan käyttää useista eri ohjelmointi- ja komentosarjakielistä käsin. (Microsoft 2011a, hakupäivä 16.4.2011; Microsoft 2011b, hakupäivä 1.5.2011.)

Opinnäytetyön tavoitteena oli perehtyä Python-ohjelmointikieleen, sen syntaksiin ja käyttömahdollisuuksiin. Tämän lisäksi tuli perehtyä tiedonkeruuseen Windows-järjestelmästä, sekä ottaa selvää, soveltuuko Python toimeksiantajan tarvitseman ohjelmiston toteutukseen. Opinnäytetyön tavoitteisiin kuului lisäksi tarvittavan ohjelmiston kehitystyö, mikäli Python-kieli katsottaisiin soveliaaksi tuotantovälineeksi.

2 PYTHON-OHJELMOINTIKIELI

2.1 Yleisesti

Pythonia käytetään usein skripti- eli komentosarjakielenä erilaisissa web-sovelluksissa, kuten esimerkiksi Django-web-kehyksessä (Django Software Foundation 2011, Hakupäivä 6.4.2011). Se on lisäksi sisällytetty erillisenä komentosarjakielenä mukaan moniin työpöytäsovelluksiin, kuten esimerkiksi kuvankäsittelyohjelma Gimp:iin ja 3D-animaatio-ohjelmisto Maya:n (Henstridge 2006, hakupäivä 6.4.2011; Autodesk 2011, hakupäivä 6.4.2011). Pythonin käyttöä ei ole rajoitettu vain komentosarjakieleksi, vaan sitä voidaan käyttää myös perinteisempään ohjelmointiin.

Python on tulkattava ohjelmointikieli, mikä tarkoittaa sitä, että Python-kielistä ohjelmaa ajavalla laitteistolla tulee yleensä olla asennettuna erityinen Python-tulkki (Kasurinen 2009, 6). Aina se ei kuitenkaan ole pakollista, sillä Python-koodista on mahdollista paketoida erityinen kokonaisuus, jossa tulkki sisällytetään itse ohjelmaan. Tulkattavan ohjelmointikielen käyttö voi olla niin haitaksi kuin hyödyksikin. Tulkattavan ohjelman ajonaikainen suorituskyky ei välttämättä ole niin hyvä kuin valmiiksi konekielelle käännetyn ohjelman. Toisaalta kehitystyö ns. korkeamman tason kielellä voi olla helpompaa ja nopeampaa.

Pythonin voi ladata ja ottaa käyttöönsä ilmaiseksi osoitteesta www.python.org. Asennettava Python-tulkki on saatavilla usealle eri käyttöjärjestelmälle, kuten muun muassa Windowsille ja Mac OS X:lle. Yleisimmissä Linux-jakeluissa se asentuu käyttöjärjestelmän mukana. Se on saatavilla myös erikoisemmille alustoille, kuten Symbian S60:lle (Nokia 2011, hakupäivä 7.4.2011) ja jopa Applen iPod-soittimelle (Avariento 2011, hakupäivä 7.4.2011).

Kun Pythonista valmistui versio 3.0, se ei ollut enää yhteensopiva aiempien versioiden kanssa. Sen kielioppia yksinkertaistettiin ja yhdenmukaistettiin, jotta se olisi helpommin lähestyttävä myös vasta-alkajille (Kasurinen 2009, 6). Muutoksia tehtiin muun muassa luokkien ja merkkijonojen käsittelyyn, kokonaislukutyyppeihin, operandien vertailuun ja Pythonin peruskirjastoja järjesteltiin ja nimettiin uusiksi (Python Software Foundation 2010a, hakupäivä 7.5.2011).

2.2 Erilaisia Python-toteutuksia

Python-tulkki sisältää kattavan joukon tarpeellisia moduuleita, joita käyttämällä on mahdollista tehdä vaativiakin ohjelmistoja. Tulkin perusasennusta kutsutaan CPythoniksi sen C-kielisen toteutuksen takia, mutta sen lisäksi on olemassa myös muunkielisiä toteutuksia Pythonista. Jython on Java-alustan toteutus Python-ohjelmointikielestä. Sen avulla Python-kielistä ohjelmakoodia voidaan ajaa Javan virtuaalikoneessa ilman erikseen asennettua Python-tulkkia. Sen kehityksen aloitti vuonna 1997 Jim Hugunin, ja kehittäminen jatkuu edelleen. Kirjoitushetkellä Jython on versiossa 2.5, ja se sisältää myös Pythonista version 2.5. (Jython 2011a, hakupäivä 7.4.2011; Jython 2011b, hakupäivä 9.4.2011.)

Toinen hyvin yleisesti käytetty toteutus on nimeltään IronPython. Sen kehitys käynnistyi vuonna 2003 niin ikään ohjelmoija Jim Huguninin toimesta. Hugunin oli lukenut artikkelin, jossa väitettiin, ettei .NET-ohjelmistokehys ole tarpeeksi avoin dynaamisten kielten, kuten esimerkiksi Pythonin, käyttöön. Väitteen todenperäisyyttä testatakseen hän ohjelmoi prototyypin Python-kielisestä .NET-kehyksestä ja totesi CLR:n, (Common Language Runtime, ohjelmisto, joka suorittaa ajettavat ohjelmat) olevan avoin ja vakaa myös Pythonin kaltaiselle dynaamiselle kielelle. (Microsoft 2008, hakupäivä 9.4.2011.)

Microsoft palkkasi Huguninin kehittämään edelleen toteutusta Pythonin ajamiseksi .NET-ympäristössä. Vuonna 2005 PyCon-konferenssissa julkistettiin IronPython 0.7, ensimmäinen vakaa toteutus, joka oli täysin integroitu .NET:n kanssa. IronPython on avoimen lähdekoodin ohjelmisto, ja sen kehitys jatkuu edelleen. Kirjoitushetkellä IronPythonista on julkaistu versio numero 2.7 ja, kuten Jythonissakin, versionumero on sama kuin sen tukema Python. (Microsoft 2008, hakupäivä 9.4.2011.)

Toteutukset, kuten Jython ja IronPython, mahdollistavat Javan tai .NET-kirjastojen käytön Python-koodin sisällä. Myös ohjelmistojakelu voi helpottua, kun kohdelaitteistolla ei tarvitse olla erillistä Python-asennusta, vaan ohjelmaa ajettaisiin esimerkiksi Javan virtuaalikoneessa tai CLR-ympäristössä. Näiden toteutusten vaivana on kuitenkin niissä käytettävä Python-kielen versio. Toteutusten valmistuminen ja kehitys vaativat kuitenkin paljon työtä, eikä esimerkiksi IronPythonista ole todennäköisesti piankaan saatavilla Python 3 -yhteensopivaa versiota (Laird 2010, hakupäivä 19.3.2011).

3 WINDOWS MANAGEMENT INSTRUMENTATION

3.1 Standardien kehitys

Samaan aikaan, kun Python-kielen kehitys oli aluillaan, myös tietokoneiden määrä yrityksissä ylipäänsä oli kasvamassa suurta vauhtia. Niiden yleistyttyä suurissa yrityksissä kävi selväksi, että niiden hallintaa varten tarvittiin jokin yhteinen alusta. Vuonna 1996 BMC Software, Cisco Systems, Compaq, Intel ja Microsoft aloittivat yhteistyön Distributed Management Task Forcen (DMTF) kanssa. Tarkoituksena oli kehittää uusi, yhteinen rajapinta suurten tietojärjestelmien hallintaan. (Microsoft 2000, hakupäivä 12.2.2011.)

Työn tuloksena syntyi prototyyppi, josta myöhemmin kehittyi CIM (Common Information Model). CIM on DMTF:n ylläpitämä ja kehittämä avoin standardi. Se koostuu kahdesta osasta, joista CIM-spesifikaatio kuvailee tiedonkeruussa ja tiedon lähettämisessä käytettävien mallien kieltä, nimeämistä ja kartoitusta. CIM-skeema tarjoaa varsinaisen selvityksen mallista sekä tiedon viitekehyksestä. Se määrittelee tarvittavat luokat, niiden ominaisuudet ja metodit. Se ei ole sidottu yhteen tiettyyn toteutukseen, vaan on tarkoitettu määrittelemään, miten hallinnoitava informaatio esitetään. (Microsoft 2000, hakupäivä 12.2.2011.)

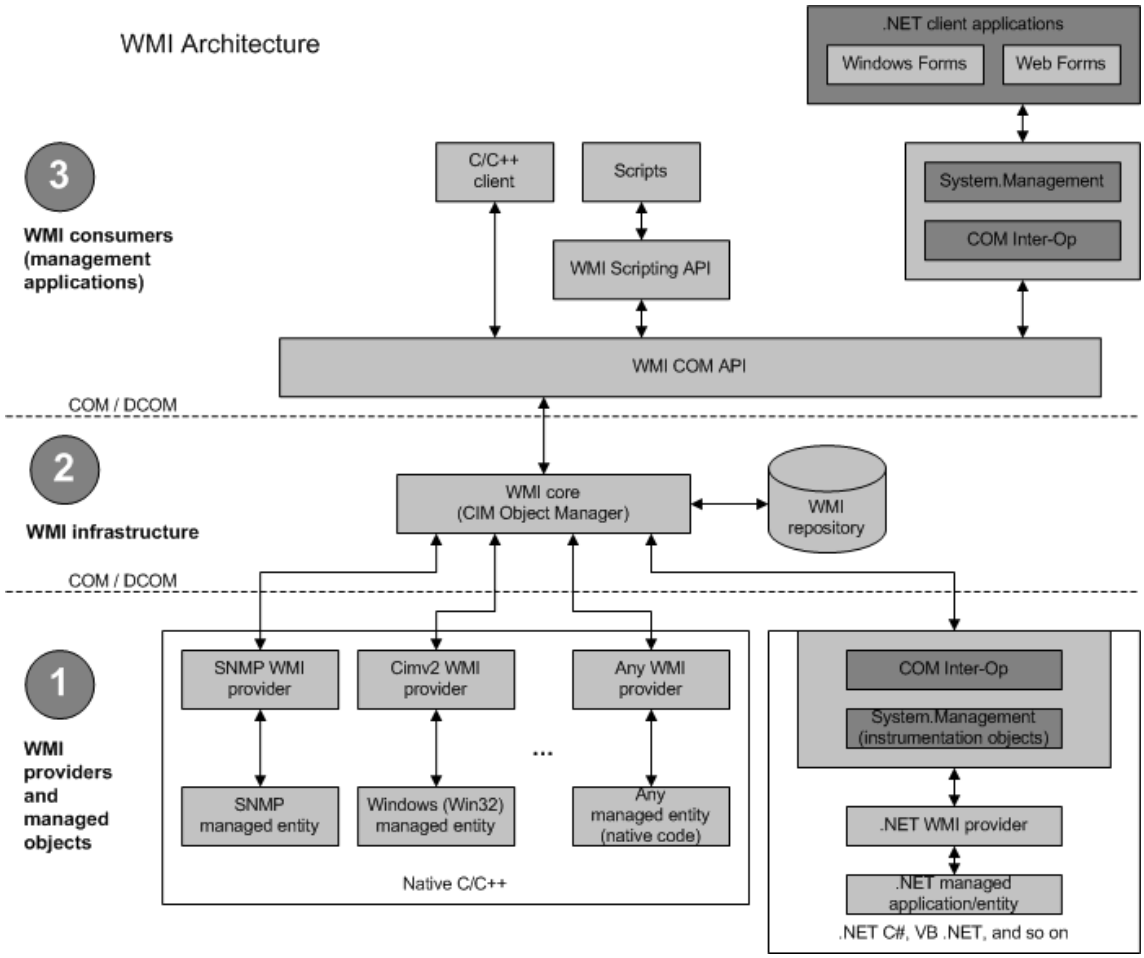
Saman yhteistyön tuloksena syntyi myös WBEM (Web-Based Enterprise Management). WBEM on kokoelma erilaisia tekniikoita, joiden tarkoituksena on luoda yhdenmukainen käytäntö hajautettujen tietojärjestelmien hallintaan (DMTF 2011a, hakupäivä 16.4.2011). Kun CIM-standardi on kuvaamassa dataa, WBEM määrittää sen välittämisessä käytettävää koodausta ja lähetystapaa (DMTF 2011b, hakupäivä 17.4.2011).

3.2 Microsoftin toteutus

WMI on Microsoftin tekemä toteutus WBEM- ja CIM-standardeista. WMI on työkalu, joka mahdollistaa pääsyn tietokoneen hallintadataan paikallisesti tai etänä ajettavan sovelluksen tai komentosarjan avulla. Ilman WMI:tä, saman tiedon saamiseksi jouduttaisiin käyttämään monimutkaisia käyttöjärjestelmäkohtaisia ohjelmointirajapintakutsuja, joita ei välttämättä voida käyttää ollenkaan komentosarjoissa. WMI esiteltiin Windows 2000 SP2 -käyttöjärjestelmän

mukana, mutta se on saatavilla myös aiemmille käyttöjärjestelmille. (Microsoft 2011a, hakupäivä 16.4.2011; Microsoft 2011b, hakupäivä 1.5.2011.)

WMI:n rakenne ja toiminta on esitetty kuvioissa 1 ja voidaan jakaa kolmeen osaan: oliot (kohta 1), infrastruktuuri (kohta 2) ja WMI-kuluttajat (kohta 3).



KUVIO 1: WMI:n arkkitehtuuri (Microsoft 2011a, hakupäivä 16.4.2011)

Kohdan 1 olioita ovat hallitut objektit (*managed objects*) ja WMI-tarjoajat (*WMI providers*). Hallittuja objekteja voivat olla kaikki laitteiston tai ohjelmiston osat, jotka voidaan esittää WMI-luokkana. Esimerkiksi tietokoneen kiintolevy, tietokanta tai käyttöjärjestelmä voi olla hallittu objekti. WMI-tarjoajia ovat COM-oliot (Component Object Model), jotka valvovat yhtä tai useampaa hallittua objektia. Tarjoajat välittävät WMI:lle datan hallittavasta objektista sekä kuljettavat WMI:ltä tulevat viestit takaisin hallitulle objektille. Kohdan 3 WMI-kuluttajia (*WMI consumers*) ovat kaikki hallintaohjelmistot ja komentosarjat, jotka ovat tekemisissä WMI-infrastruktuurin kanssa. Ohjelmat ja komentosarjat voivat käyttää hallitusta objektista tarjolla

olevia metodeja ja ominaisuuksia kutsumalla suoraan WMI COM API:a tai välillisesti WMI Scripting API:a. Kuvion kohdassa 2 on esitetty WMI-infrastruktuuri (*WMI infrastructure*), joka on osa Windows-käyttöjärjestelmää. Se voidaan jakaa edelleen kahteen osaan: WMI-säilöön (*WMI repository*) ja WMI-palveluun (*WMI service*), jonka osana on WMI-ydin (*WMI core*). WMI-säilö toimii WMI:n nimiavaruuksien mukaan järjestettynä alueena WMI-palvelua varten. Säilöön tallennetaan WMI-tarjoajien määrittelemät luokat, jotka ovat muuttumatonta dataa. Kaikki muuttuva data haetaan sitten suoraan WMI-tarjoajalta käyttäen hyväksi säilöön tallennettuja luokkia. WMI-palvelu toimii kuluttajien, tarjoajien ja säilön välisenä tiedon välittäjänä. (Microsoft 2011a, hakupäivä 16.4.2011.)

Kun Windows-käyttöjärjestelmä käynnistyy, luo WMI-palvelu WMI-säilöön oletuksena esimerkiksi Win32-nimisen WMI-tarjoajan luokat. Yksi Win32-tarjoajan luokista on Win32_LogicalDisk, joka sisältää metodeja ja ominaisuuksia Windows-järjestelmän paikallisista tallennusvälineistä. Kuluttajana toimiva hallintaohjelmisto voi sitten kutsua Win32_LogicalDisk-luokan ScheduleAutoChk-metodia ja antaa sille parametrina tiedon valitusta kiintolevystä. Tämä kutsu ajastaisi järjestelmän suorittamaan valitulle kiintolevylle levyntarkistuksen seuraavan käynnistyksen yhteydessä. (Microsoft 2011c, hakupäivä 17.4.2011.)

Tiedonhakuun WMI:stä on kehitetty erityinen WMI-kyselykieli, WMI Query Language (WQL). Se muistuttaa läheisesti SQL-kyselykieltä, ja sen avulla on mahdollista tehdä hakuja WMI:n tarjoamaan dataan, tapahtumiin ja rakenteeseen. Data-kyselyt ovat näistä tavallisimpia, ja niitä käytetään palauttamaan WMI-tarjoajien luokkien esiintymiä ja/tai tietoa niistä. Esimerkiksi Win32_LogicalDisk-luokasta voitaisiin hakea vain tiedot levyjen käyttämistä tiedostojärjestelmistä sen sijaan, että palautettaisiin suuri määrä tietoa, jota ei tarvittaisi. Tapahtumakyselyt hakevat järjestelmästä nimensä mukaan tietoa esiintyneistä tapahtumista järjestelmän lokitiedoista. Sitä voidaan myös käyttää ilmoittamaan, kun jokin tietyn tyyppinen tapahtuma kirjataan lokiin. Rakennekyselyt taas palauttavat tietoa luokkien rakenteesta ja niiden yhteyksistä. (Microsoft 2011d, hakupäivä 1.5.2011.)

4 PYTHON-LAAJENNOKSIA JA -MODUULEITA

4.1 Tarve ja yhteensopivuus

CPython on tarkoitettu alustariippumattomaksi ohjelmointikieleksi. Siksi ei olisi tarkoituksenmukaista, jos se sisältäisi paljon alustakohtaisia toiminnallisuuksia. Peruskirjastosta, erityisesti `os`- ja `sys`-moduuleista, löytyy joitain Unix- ja Windows-ympäristöön tarkoitettuja alustakohtaisia käskyjä, mutta niiden toiminnot keskittyvät lähinnä tiedostojen- ja prosessienhallintaan ja itse tulkin ympäristötietoihin (Python Software Foundation 2010b, hakupäivä 23.3.2011). On mahdollista ajaa Windowsin omia järjestelmäkutsuja Python-koodin sisältä, mutta se voi lisätä koodin määrää ja tehdä siitä vaikealukuista. Se ei myöskään toisi Windowsin omaa ohjelmointirajapintaa suoraan käyttöön.

Ei siis ole mielekästä lisätä tarkkoja alustakohtaisia toimintoja oletuksena CPythoniin, mutta näitä toiminnallisuuksia on mahdollista lisätä erillisinä laajennoksina (*extensions*). Laajennokset tuovat lisää moduuleita ohjelmoijan käyttöön ja ne asennetaan olemassa olevan Python-asennuksen päälle. Laajennosten ongelmana voi olla, että asennettava laajennos on tarkoitettu käytettäväksi vain jollain tietyllä Python-versiolla. Kun jokin laajennos halutaan ottaa käyttöön, pitää ensin tarkistaa, onko siitä tehty toimivaa versiota käytössä olevalle Python-asennukselle. Jos taas halutaan päivittää käytössä oleva CPython uudempaan versioon, tulee tarkistaa, löytyykö myös käytössä olevista laajennoksista tukea sille.

Laajennusten ylläpito ja päivittäminen perustuu yleensä vapaaehtoisuuteen ja on harrastelijoiden, erilaisten säätiöiden tai ryhmien tekemää (Kasurinen 2009, 175). Tästä syystä niiden käyttöönotto voi jossain tapauksissa olla hankalaa. Kun Python päivitettiin versioon 3, poistui siitä taaksepäin yhteensopivuus edellisten versioiden kanssa. Samalla myös poistui tuki suurelta osalta olemassa olevia laajennuksia. Näiden päivittäminen yhteensopiviksi uuden version kanssa on edennyt vaihtelevaan tahtiin. Joitain moduuleita on suurempi tarve pitää mahdollisimman toimintakykyisinä, kuin päivittää ne yhteensopiviksi uuteen versioon. Joidenkin kääntäminen on myös suuritöisempää kuin toisten. Tästä syystä vielä kirjoitushetkelläkin on syytä harkita siirtymistä uuteen versioon, mikäli tietää tarvitsevansa jotain tiettyä moduulia, jota ei ole vielä päivitetty. (Python Software Foundation 2011, hakupäivä 16.4.2011.)

On myös mahdollista asentaa yhdelle tietokoneelle useita eri versioita Pythonista. Siinäkin tapauksessa tulee pitää huolta asennettavien laajennosten ja käytettävän Python versioiden yhteensopivuudesta. Lisäksi on huomioitava Windowsin käyttämät oletusohjelmat tietyn tyyppisille tiedostoille sekä halutun Python version lisääminen Windowsin polkuun.

4.2 Laajennokset Windows-järjestelmälle ja WMI:lle

VBScript- tai Perl-komentosarjakieliä, VisualBasic- tai C#-ohjelmointikieliä tai monia muita kieliä käyttämällä voitaisiin suoraan hakea ja käyttää WMI:n tuottamaa dataa (Microsoft 2011e, hakupäivä 21.4.2011). CPythonissa ei kuitenkaan tule oletuksena tukea Microsoftin WMI:lle. Käyttämällä PyWin32 (Python for Windows Extensions) ja Python WMI –laajennoksia voidaan tuki tuoda myös CPython-asennukseen.

PyWin32-moduuli on etupäässä Mark Hammondin kehittämä kokoelma, joka tuo CPythonin käyttöön Windows-spesifisiä toiminnallisuuksia. Hienoimpia esimerkkejä moduulin mahdollisuuksista on sen mukana tuleva PythonWin-ohjelmointiympäristö. Ohjelmointiympäristö on toteutettu käyttäen PyWin32-moduulin esiintuomaa Windowsin MFC-kirjastoa (Microsoft Foundation Class), joka on ohjelmistokehys muun muassa ikkunoinnin, valikoiden ja dialogien toteuttamiseen Windows-ympäristössä (Microsoft 2010, hakupäivä 19.3.2011). MFC:n lisäksi PyWin32-moduuli tuo käyttöön Windows API:n, Component Object Modelin (COM) ja Windowsin rekisterin ja tapahtuma-lokin. (Hammond 2009.)

Hammondin luoma moduuli riittäisi periaatteessa jo sellaisenaan tuomaan WMI:n CPythonin käyttöön, mutta se ei olisi vielä kovin käytännöllistä Python-kielen COM-toteutuksen hankaluuden vuoksi. Käyttöä helpottaakseen Tim Golden on luonut Python WMI -moduulin. Kaikki toiminnot on toki mahdollista tehdä käyttäen vain PyWin32-moduulia, mutta Python WMI -moduuli paketoi WMI Scripting API:n käyttäjäystävällisempään muotoon. Se helpottaa WMI:n käyttöä Python-koodista käsin vähentäen samalla kirjoitettavan koodin määrää. (Golden 2009a, hakupäivä 12.2.2011; Golden 2009b, hakupäivä 3.5.2011.)

Goldenin WMI-moduuli saadaan Python-koodissa käyttöön sisällyttämällä se mukaan, kuten mikä tahansa moduuli Python-kielessä, käyttäen `import`-toimintoa seuraavasti:

```
import wmi
```

Tämän jälkeen voidaan esimerkiksi tulostaa Windows Installerin järjestelmään asentamat ohjelmat ja niiden ominaisuudet aakkosjärjestyksessä käyttäen WMI:n Win32_Product-luokkaa seuraavalla tavalla:

```
c = wmi.WMI()
for product in c.Win32_Product():
    for property in product.properties:
        print(property, getattr(product, property))
```

Ensimmäisellä rivillä tehdään muuttujasta `c` WMI-luokan esiintymä, jonka jälkeen iteroidaan läpi kaikki järjestelmästä löytyvät Win32_Product-luokan esiintymät. Jokaisen löydetyn esiintymän yhteydessä käydään läpi myös kaikki esiintymän ominaisuudet. Viimeisellä rivillä tulostetaan ominaisuuden nimi, sekä `getattr`-metodilla haettu ominaisuudelle annettu arvo.

Edellinen esimerkki on myös mahdollista tehdä käyttäen WQL-kyselyä:

```
wql = "SELECT Caption, Description FROM Win32_Product"\
      "WHERE InstallDate2>\"20110101000000.000000-UUU\""
for product in c.query(wql):
    for property in product.properties:
        print(property, getattr(product, property))
```

Ensimmäisellä rivillä muodostetaan WQL-kysely, jossa määritetään, mitä tietoja ja minkä luokan esiintymästä tiedot haetaan. Esimerkissä käytetään samaista Win32_Product-luokkaa kuin aiemmassakin esimerkissä. Poikkeuksena on, että kaikkien ominaisuuksien sijaan haetaan luokasta vain Caption- ja Description-ominaisuudet, jotka palauttavat asennetun tuotteen lyhyen ja pidemmän kuvauksen (Microsoft 2011f, hakupäivä 3.5.2011).

Kyselyn where-ehtoon on kolmannella rivillä kirjattuna CIM DATETIME -muotoinen aikaleima, jota tarvitaan, kun käytetään WMI Scripting API:a. Aikaleimaa käytetään hakemaan tuotteita, joiden asennuspäivä on myöhäisempi kuin 1. tammikuuta 2011. Aikaleiman neljä ensimmäistä numeroa ovat vuosiluku, kaksi seuraavaa kuukausi, sen jälkeen edelleen kahdella merkitsevällä numerolla päivä, tunnit, minuutit ja sekunnit. Lopuksi aikaleimaan voidaan kirjata mikrosekunnit kuudella numerolla, minkä jälkeen neljä viimeistä merkkiä kertovat poikkeaman UTC-ajasta (Coordinated Universal Time). (Microsoft 2011g, hakupäivä 3.5.2011.) Kolmannella rivillä suoritetaan kysely komennolla `c.query(wql)`, jolle välitetään parametrina aiemmin merkkijonona muodostettu

kysely. Kysely palauttaa Win32_Product-luokan esiintymiä, ja ne voidaan iteroida läpi kuten edellisessäkin esimerkissä.

Ehtoja voidaan antaa myös ilman WQL-kyselyä. Edellisen esimerkin Win32_Product-luokkaan voidaan tehdä vastaavanlainen haku välittämällä sille alussa halutut parametrit:

```
c = wmi.WMI()
for product in c.Win32_Product(["Caption", "Description"],
InstallDate2>"20110101000000.000000-UUU"):
    for property in product.properties:
        print(property, getattr(product, property))
```

Toisella rivillä aloitetaan for-silmukka, joka iteroi läpi Win32_Product-luokan instanssit. Palautettavat luokan ominaisuudet on erotettu hakasulkeilla muista parametreista, ja viimeisenä parametrina on myös aiemmassa esimerkissä ollut InstallDate2.

Python WMI on siis varsin joustava ja helppo käyttää tiedon hakuun. Ohjelmoijan työksi jää kuitenkin tarvittavien luokkien löytäminen. WMI sisältää suuren määrä erilaisiin tarpeisiin suunniteltuja luokkia. Pelkästään Windows-järjestelmään asennettujen ohjelmien tarkastelua varten on olemassa yli 60 erilaista Win32-luokkaa (Microsoft 2011h, hakupäivä 3.5.2011).

4.3 XML-muotoisen datan käsittely

CPython sisältää `xml.etree.ElementTree`-moduulin XML-muotoisen tiedon käsittelyyn. Sitä voidaan käyttää niin XML-muotoisen datan kirjoittamiseen kuin lukemiseenkin.

```
import xml.etree.ElementTree as ET
juuri = ET.Element(“Juuri”)
oksa1 = ET.SubElement(juuri, “Oksa”)
oksa1.text = ”XML”
oksa2 = ET.SubElement(juuri, “Oksa”)
oksa2.text = ”puu”
oksa3 = ET.Element(“Oksa”)
oksa3.text = ”kasvaa”
juuri.append(oksa3)
puu = ET.ElementTree(juuri)
puu.write(”puu.xml”, encoding=”utf-8”)
```

Esimerkin ensimmäinen rivi sisällyttää `xml.etree.ElementTree`-moduulin mukaan koodiin. Se on tapana sisällyttää aliaksella `ET`, jotta koodi olisi helpompi kirjoittaa ja jotta moduulin toteutusta voitaisiin myöhemmin tarvittaessa vaihtaa. Moduulista on olemassa useita

toteutuksia ja käytettävän toteutuksen vaihtaminen onnistuu muuttamalla `import`-lausetta, eikä muualle koodiin tarvitse tehdä muutoksia. (Lundh 2011, hakupäivä 6.5.2011.)

Toisella rivillä kutsutaan `Element`-funktiota, jolle välitetään parametrina luotavan XML-tagin nimi. Funktio palauttaa `Element`-luokan instanssin, joka esimerkin tapauksessa toimii koko XML-dokumentin juurielementtinä, josta kaikki muut elementit periytyvät. Kolmannella rivillä kutsutaan moduulin funktiota `SubElement`, joka luo määrätyn elementin alle uuden elementin. Tässä tapauksessa `Ensimmäinen`-elementti saa alleen uuden `Toinen`-elementin. `Element`-luokalla on ominaisuus `text`, joka on XML-tagien välinen tekstimuotoinen informaatio. On myös mahdollista lisätä alielementti käyttäen `Element`-luokan `append`-metodia. Tällöin tulee ensin luoda `Element`-luokan instanssi ja sitten antaa tämä instanssi paramentrina halutun olion `append`-metodille. (Python Software Foundation 2010c, hakupäivä 6.5.2011.)

`xml.etree.ElementTree`-moduulissa on samanniminen `ElementTree`-luokka, joka pitää sisällään kokonaisen XML-puun rakenteen. Luokalle välitetään parametrina aiemmin luotu juurielementti. `ElementTree`-luokalla on `write`-metodi, jota voidaan käyttää luodun rakenteen tallentamiseksi tiedostoon. Parametreina metodille välitetään kirjoitettavan tiedoston nimi tai tiedosto-olio ja halutessa merkistökoodaus. (Python Software Foundation 2010c, hakupäivä 6.5.2011.) Tallennettuna esimerkin XML näyttäisi seuraavalta:

```
<Juuri>
    <Oksa>XML</Oksa>
    <Oksa>puu</Oksa>
    <Oksa>kasvaa</Oksa>
</Juuri>
```

Tiedostosta haettava XML voidaan tuoda ohjelmaan käyttämällä `xml.etree.ElementTree`-moduulin `parse`-funktiota. Funktio palauttaa `ElementTree`-luokan instanssin (Python Software Foundation 2010c, hakupäivä 6.5.2011).

```
puu = ET.parse(”puu.xml”)
oksat = puu.getroot().findall(”Oksa”)
for oksa in oksat:
     print(oksa.text)
```

Esimerkin `getroot`-metodi palauttaa juurielementin läpikäytävästä XML-hierarkiasta ja `findall`, jolle annetaan parametrina haettavan tagin nimi, palauttaa iteraattorin, joka sisältää kaikki ylätason `Element`-oliot, joilla on haettava nimi (Python Software Foundation 2010c, hakupäivä 6.5.2011). Esimerkin tapauksessa tulostettaisiin kaikkien Oksa-tagien sisällä oleva teksti.

4.4 Verkkoyhteyksien luonti

CPythonin `http.client`-moduulia voidaan käyttää verkon yli tiedon lähettämiseen. Sekin voidaan sisällyttää ohjelmaan käyttäen aliasta helpottamaan ohjelman kirjoittamista. Käytössä moduulin `HTTPConnection`-luokasta muodostetaan olio, jolle välitetään parametrina yhteyden isännän osoite. (Python Software Foundation 2010d, hakupäivä 6.5.2011.)

```
from http.client import HTTPConnection as HC
h = HC(www.testi.fi)
selector = ”/testi.php”
h.request('POST', selector, body, headers)
print(h.getresponse().read())
```

`HTTPConnection`-luokalla on metodi `request`, jota käytetään HTTP-pyyntöjen lähettämiseen. Parametreina metodille voidaan antaa käytettävä metodi, joka dataa lähetettäessä on ”POST”, URL, joka huolehtii lähetettävän datan käsittelystä vastaanottavassa päässä, body-elementti, joka sisältää lähetettävän datan sekä mahdolliset ylimääräiset headerit. (Python Software Foundation 2010d, hakupäivä 6.5.2011.)

`getresponse`-metodi, jota kutsutaan pyynnön jälkeen palauttaa `HTTPResponse`-luokan instanssin. Luokalla on metodi `read`, joka palauttaa palvelimelta saadun vastauksen aiemmin lähetettyyn pyyntöön. Saatu vastaus voidaan käsitellä tarpeen mukaan. (Python Software Foundation 2010d, hakupäivä 6.5.2011.)

4.5 Asetusten hakeminen tiedostosta

CPython sisältää myös valmiin moduulin, joka on suunniteltu ohjelmakohtaisten asetustiedostojen lukua ja kirjoitusta varten. `configparser`-moduuli mahdollistaa tekstimuotoisten asetustiedostojen lukemisen ja kirjoittamisen. Asetustiedosto jaetaan nimettyihin osiin, jotka voivat sisältää asetuksia ja niiden arvoja.

```
[server]
# Asetukset palvelinyhteydelle
host: www.testi.fi
selector: /testi.php
```

Asetustiedoston osat erotetaan muusta tiedosta hakasulkeilla, jonka jälkeen kerrotaan halutuista asetuksista ensin asetuksen nimi erotettuna kaksoispisteellä tai yhtäsuuruusmerkillä asetuksen arvosta. Esimerkin asetustiedoston osan nimi on server ja se sisältää kaksi asetusta: host ja selector, joilla on arvoina ”www.testi.fi” ja ”/testi.php”. Tiedostoon voidaan kirjoittaa vapaita kommentteja erotettuna muusta tekstistä rivin aloittavalla ristikkomerkillä. (Python Software Foundation 2010e, hakupäivä 6.5.2011.) Asetustiedostojen lukeminen voidaan Python-koodissa tehdä esimerkiksi seuraavalla tavalla:

```
from configparser import SafeConfigParser
conf = SafeConfigParser()
conf.read("asetukset")
host = conf.get("server", "host")
selector = conf.get("server", "selector")
```

Moduuli sisältää useita toteutuksia asetusten koostajasta: `RawConfigParser`, `ConfigParser` ja `SafeConfigParser`, joista viimeinen on suositeltavin käyttää (Python Software Foundation 2010e, hakupäivä 6.5.2011). Luokan `read`-metodi hakee muistiin parametrina annetun tiedoston sisältämät asetukset. Asetukset voidaan sitten tallentaa ohjelman käyttämiin muuttujiin hakemalla ne `get`-metodilla. Metodille tulee antaa parametreina asetustiedoston osan sekä haettavan asetuksen nimi. Osan kaikki asetukset voidaan myös palauttaa listaan käyttämällä `get`-metodin sijaan `items`-metodia, jolle annetaan parametrina vain osan nimi.

4.6 Python-ohjelman paketointi

Ohjelman kehitys aloitettiin käyttäen Python-tulkin versiota 3.1.2, josta se myöhemmin päivitettiin versioon 3.2. Jos ohjelmiston jakelu haluttaisiin suorittaa pelkän lähdekoodin kanssa, pitäisi ohjelman käyttäjillä olla asennettuna sama versio Pythonista kuin kehittäjälläkin. Tämän lisäksi käyttäjiltä vaadittaisiin vastaavien laajennosten asentamista kuin kehittäjälläkin. Tällaiset vaatimukset eivät ole erityisen käyttäjäystävällisiä ja vaikeuttavat ohjelman käyttöönottoa.

cx_Freeze on kokoelma erilaisia komentosarjoja ja moduuleita Python-komentosarjojen muuntamiseksi Windows-järjestelmällä ajettaviksi exe-tiedostoiksi. Se on saatavilla Pythonin versiolle 2.3 ja kaikille sitä uudemmille. Ohjelma paketoi halutun komentosarjan ja kaikki vaadittavat Python-moduulit yhdeksi toimivaksi ja helposti siirrettäväksi kokonaisuudeksi. (Tuininga 2011, hakupäivä 29.3.2011.)

Sen käyttö on suoraviivaista, ja asennuksen jälkeen sitä voidaan kutsua komentoriviltä komennolla:

```
cxfreeze testi.py --target-dir asennus
```

Tämä hakisi `testi.py`-nimisen komentosarjan ja sen jälkeen kaikki vaadittavat moduulit ja laajennokset ja paketoisi ne yhteen kansioon, joka on määrättynä komennossa `--target-dir`-vivun jälkeen eli `asennus`-nimiseen kansioon. (Tuininga 2011, hakupäivä 29.3.2011.)

5 OHJELMISTOKEHITYS PYTHONILLA

5.1 Ohjelmiston vaatimusmäärittely

Valmiin ohjelmistokokonaisuuden infrastruktuuri koostuu kahdesta osasta: asiakkaan tietokoneella toimivasta ohjelmasta ja sen lähettämää dataa käsittelevästä palvelinohjelmistosta. Ennen ohjelmointityötä tehtiin ohjelmistoa varten kirjallinen vaatimusmäärittely toimeksiantajan kanssa. Koska opinnäytetyönä toteutettavan ohjelmiston laajuus on rajallinen, vaatimusmäärittelyssä otettiin tämä huomioon. Ohjelmistotyö jaettiin opinnäytetyön puitteissa toteutettaviin ja myöhemmin opinnäytetyöstä erillisenä työnä toteutettaviin toiminnallisuuksiin. Asiakasohjelman osuus kirjattiin määrittelyyn ensisijaiseksi, kun taas palvelimen toimintojen toteutus jätettiin opinnäytetyön ulkopuolelle. Vaatimusmäärittelyssä otettiin huomioon myös asiakasohjelman todennäköinen jatkokehitys, eikä kaikkia ominaisuuksia katsottu järkeväksi ryhtyä toteuttamaan opinnäytetyön aikataulussa.

Asiakasohjelman tulee hakea haluttuja tietoja paikallisesta Windows-käyttöjärjestelmästä. Tarkemmin rajattuna ohjelman tuli toimia Microsoft Windows XP- ja Windows 7 -käyttöjärjestelmien 32- ja 64-bittisissä versioissa. Windows Vista -käyttöjärjestelmä jätettiin kehityksen ja testauksen ulkopuolelle, mutta on todennäköistä olettaa ohjelmiston toimivan myös sillä.

Järjestelmästä haettavat tiedot jaettiin kolmeen osaan: järjestelmän tietoihin, tietokoneelle asennettujen ohjelmistojen tietoihin sekä lokitietoihin. Järjestelmän tietoihin katsottiin tarvittavan tietokoneella olevan käyttöjärjestelmän tiedot, paikallisten käyttäjätilien tiedot, näytönohjaimen ja BIOS:in tiedot. Tämä lisäksi tarvittiin järjestelmän fyysiset tiedot kuten asennetun suorittimen tiedot, muistin määrä, loogiset asemat ja kiintolevyjen osiot ja mahdollisesti käytössä olevan tietokoneen akun tiedot. Ohjelmiston haluttiin hakevan myös tiedot tietokoneelle asennetuista ohjelmista. Erityisesti kiinnostuksen kohteena olivat ohjelmistojen versionumerot, joista on usein pääteltävissä mahdollisten toimintahäiriöiden syy. Windows-järjestelmän tapahtumaloki jaetaan system-, security- ja application- eli järjestelmä-, suojaus- ja ohjelmistolokeihin. Näistä lokitiedoista haluttiin hakea mahdolliset virheet ja varoitukset.

Kaikki kerätty data tuli koostaa järkevään muotoon ja lähettää edelleen verkon yli palvelimen käsiteltäväksi. XML (eXtensible Markup Language) nähtiin sopivana välineenä tiedon muodostamiseen, joten kerätystä datasta päätettiin luoda yksi XML-muotoinen raportti, joka voidaan lähettää sellaisenaan tai osittain palvelimelle.

5.2 Raportin muodostaminen

Asiakasohjelman rakenne jaettiin kolmeen moduuliin: järjestelmätiedot hakevaan ja raportin muodostavaan moduuliin, koostetun datan verkon yli lähettävään moduuliin sekä varsinaista ohjelmasilmukkaa läpikäyvään moduuliin. Python mahdollistaisi kaikkien ohjelman funktioiden, luokkien ja niiden metodien ja muun koodi tekemisen vain yhteen tiedostoon, mutta katsottiin kuitenkin järkevämmäksi jakaa koodi omiin luonnollisiin kokonaisuuksiinsa. Näin koodista saadaan helpommin ylläpidettävää.

Ohjelma hakee käynnistyessään tiedon siitä, ajetaanko sitä tuetulla käyttöjärjestelmällä. Järjestelmätietojen hakemista varten luotu moduuli sisältää luokan, josta voidaan pääohjelmassa tarvittaessa luoda olio. Tämän olion metodit ja ominaisuudet huolehtivat ohjelman varsinaisesta toiminnasta. Mikäli käytettävä järjestelmä on oikea, jatketaan ohjelman suorittamista ja luodaan tiedonkeruuta varten sopiva olio. Käytössä olevan järjestelmän tiedot haettiin `platform`-moduulin `system`-metodilla, joka on käytettävissä CPython-asennuksessa. Metodi palauttaa käyttöjärjestelmästä riippuen merkkijonon *Linux*, *Windows* tai *Java* tai tyhjän merkkijonon mikäli järjestelmää ei pystytä tunnistamaan (Python Software Foundation 2010f, hakupäivä 6.5.2011).

Oliota luotaessa haetaan raporttia varten tarvittavat asetukset erillisestä asetustiedostosta käyttäen `configparser`-moduulia. Asetukset sisältävät tiedon käytössä olevista raportin osista sekä niiden päivitysvälistä. Asetusten luvun lisäksi Python-tulkki kutsuu luokan muodostinta eli `__init__`-metodia. Metodissa tarkistetaan, löytyykö aiemmin muodostettua raporttia paikallisesti tallennettuna. Jos tiedosto löytyy, ohjelma yrittää koostaa siitä `ElementTree`-luokan instanssin. Jos XML:n koostaminen epäonnistuu ja löydetty tiedosto on korruptoitunut tai tiedostoa ei ole lainkaan, kutsutaan uuden raportin koostavaa metodia.

Raportin koostaminen ensimmäisellä kerralla toimii lähes samalla tavalla kuin myöhemmin esitelty päivittäminenkin. Poikkeuksena on, että raportin kaikki osat haetaan yhtä aikaa. Ohjelma

lukee asetustiedostosta käytössä olevat raportin osat ja kutsuu peräjälkeen niitä vastaavia metodeita. Tämä jälkeen kerätty data koostetaan yhteen XML-puuhun ja tallennetaan tiedostoon.

Kun `ElementTree`-luokan olio saadaan luotua, voidaan sen sisältämät raportin osat iteroida läpi ja tallentaa jokaisen osan luontiaika erilliseen tauluun. Ohjelma vertaa raportin osien luontiaikoja ja niille määrättyjä asetustiedostosta luettuja päivitysvälien pituuksia keskenään ja palauttaa vertailun perusteella tiedon seuraavaksi päivitettävästä raportin osasta ja seuraavaan päivitysajankohtaan kuluvan ajan pituuden. Joitain raportin osia päätettiin päivittää tiuhempaan kuin toisia. Tämä todettiin järkeväksi, sillä, toisin kuin tapahtumalokin tiedot, esimerkiksi järjestelmän näytönohjaimen tietojen katsottiin pysyvän muuttumattomina lyhyellä aikavälillä. Näin ollen päätettiin määritellä erilliseen asetustiedostoon raportin osien päivitysvälit, mikä helpottaa mahdollisten muutosten tekemistä jälkikäteen.

Kun ohjelma saa tiedon seuraavaksi päivitysvuorossa olevasta raportin osasta, se myös vastaanottaa odotusajan seuraavaan päivitykseen. Tämä odotusaika välitetään `time`-moduulin `sleep`-funktiolle, joka pysäyttää ohjelman suorituksen annetun pituiseksi ajaksi. Jos odotusaika on yli vuorokausi, ei seuraavaksi päivitysvuorossa olevasta raportin osasta välitetä tietoa, vaan edellinen aikavertailu suoritetaan odotusajan jälkeen uudestaan.

Kun kutsutaan raportin osan päivittävää metodia, haetaan raportin XML-koosteesta päivitettävän osan tiedot. Ohjelma kutsuu sitten päivitettävää osaa vastaavaa metodia, joka hakee `wmi`-moduulia käyttäen tarvittavan datan järjestelmästä. Kaikki järjestelmästä tietoa hakevat metodit käyttävät tiedonhakuun WMI:tä, joka palauttaa pyydetyn datan ohjelman käyttöön. Metodin sisällä palautunut data iteroidaan läpi ja siitä muodostetaan ja palautetaan `xml.etree.ElementTree.Element`-luokan olio. Kun uusi data palautuu päivitystä tekevälle metodille, poistetaan vanha data raportista `Element`-luokan `remove`-metodilla ja uusi data lisätään poistetun tilalle.

5.3 Raportin tietojen lähettäminen

Kun raportti koostetaan ensimmäisen kerran, se lähetetään kokonaisuudessaan verkon yli palvelimen käsiteltäväksi. Kun taas päivitetään vain yhtä raportin osaa, lähetetään vain tuo päivitetty osa palvelimelle. Lähetysyrityksen jälkeen raporttiin kirjataan merkintä lähetysyrityksen

tuloksesta. Jos lähetys kirjataan epäonnistuneeksi, ei raportin tietoja tai raportin jonkin osan tietoja päivitetä, ennen kuin lähetys on toistettu onnistuneesti.

Raportin lähetyksestä huolehtivalle metodille voidaan välittää parametrina lähetettävän raportin osan nimi, tai jos parametri jätetään tyhjäksi, lähetetään koko raportti. Yksittäistä raportin osaa lähetettäessä haetaan se koostetusta XML-puusta ja muunnetaan merkkijonoksi, joka välitetään verkkoliikenteestä huolehtivalle moduulille. Moduuli käyttää `configparser`-moduulia lukemaan tarvittavat asetukset lähetystä varten. Lähetysmetodi vastaanottaa merkkijonon tai, parametrin ollessa tyhjänä, lukee raportin tiedostosta ja muuntaa sen merkkijonoksi lähetystä varten. Lähetys toteutetaan `http.client`-moduulia käyttäen POST-tyyppisenä lähetyksenä, jonka jälkeen metodi palauttaa tiedon lähetyksen onnistumisesta.

5.4 Ohjelmiston toiminta Python-tulkilla ja paketoituna

Ohjelmiston tarkoituksena on toimia tietokoneen muun käytön taustalla, eikä se saa häiritä normaalia käyttöä liiaksi. Tästä syystä ohjelman resurssienkäyttöä tarkkailtiin työn edetessä. Ohjelmisto, johon kaikki vaaditut toiminnallisuudet oli toteutettuna, kulutti muistia noin 30 Mt. Vaaditun muistin määrää saatiin laskettua muuttamalla joitain yksinkertaisella `import`-komennolla sisällytettyjä moduuleita muotoon:

```
from <moduulin nimi> import <tarvittava moduulin osa>
```

Näin ohjelmaan ei sisällytetä kokonaista moduulia vaan ainoastaan se osa moduulista, joka on tarpeen ohjelman toiminnan kannalta. Tällä keinoin saatiin hieman vähennettyä ohjelman tarvitsemaa muistin määrää, ja muunnon jälkeen kulutus oli hieman yli 20 Mt. Tämän lisäksi ohjelmakoodiin lisättiin mahdollisiin kohtiin `del`-komento poistamaan tarpeettomia muuttujia. Yleensä tämä on tarpeetonta, sillä Python-tulkki sisältää oman roskienkeruujärjestelmän, joka huolehtii muistin vapautuksesta. Muuttujien viittausten poistamisella ei kuitenkaan havaittu yhtä merkittävää vaikutusta muistin kulutukseen kuin sisällytysten muutoksilla.

Paketointi ei varsinaisesti kuulunut osaksi ohjelmistolle tehtyä vaatimusmäärittelyä, mutta se koettiin tärkeäksi, ja siksi päätettiin myös kokeilla sen toteuttamista. cx_Freezen käyttö havaittiin yksinkertaiseksi ja helpoksi, mutta sen käytössä tuli ottaa huomioon sitä ajavan

käyttöjärjestelmän bittisyys. 64-bittisellä käyttöjärjestelmällä paketoitua ohjelmaa ei voida ajaa 32-bittisessä käyttöjärjestelmässä.

Paketointi suoritettiin lopulta käyttäen 32-bittistä Windows XP -käyttöjärjestelmää. Paketointi tuotti ohjelman, jota voitiin ajaa niin 64-bittisessä kuin 32-bittisessäkin ympäristössä käyttäen Windows XP- tai Windows 7 -käyttöjärjestelmää. Paketointiin suhtauduttiin aluksi varauksella mahdollisen muistinkäytön kasvun takia. Muistinkulutuksessa havaittiin kuitenkin vain pieni kasvu, eikä sitä pidetty merkityksellisenä toiminnan kannalta.

6 TULOKSET JA JOHTOPÄÄTÖKSET

Yhtenä kehitystehtävänä perehdyttiin Python-ohjelmointiin ja tutkittiin, voidaanko sitä käyttää toimeksiantajan tarvitseman asiakasohjelmiston tuottamiseen. Opitun perusteella voidaan todeta, että Python on toimiva myös Windows-alustan ohjelmointiin, vaikkakin erityisiä kolmannen osapuolen laajennoksia käyttäen. Toimeksiantajan tarvitsema ohjelmisto nähtiin mahdollisena toteuttaa valitulla ohjelmointikielellä ja ohjelmiston kehitys myös saatiin hyvälle alulle.

Vaatimusmäärittelyyn kirjatut toiminnallisuudet ja hieman enemmänkin saatiin opinnäytetyön aikataulussa toteutettua, ja ohjelmointityön lopputuloksena oli toimiva ohjelmisto asiakaskoneen tilanseurantaan. Ohjelmistolla saadaan haettua järjestelmästä haluttu data ja se voidaan välittää edelleen verkon yli palvelimelle.

Ei ole kuitenkaan varmaa, onko Python lopulta paras mahdollinen valinta toteutukselle. IBM:n (2010, 44) mukaan yksi ohjelmistosuunnittelijan tehtävistä on tarkoin harkita, mikä ohjelmointikieli palvelee parhaiten tietyn ohjelmiston toimintaa. Ohjelmiston toimintaan liittyvät tarpeet ovat hyvin Microsoft Windows -käyttöjärjestelmäspesifisiä, ja käytettävää kieltä voitaisiin vielä harkita tämän tarpeen mukaan. On kuitenkin huomioitava, että jos eri ohjelmointikielistä ei ole osaamista, ei niiden käytettävyyttäkään voida vertailla.

Tarvittavien toiminnallisuuksien tuottamiseen vaadittiin Windows-kohtaisten rajapintojen (erityisesti WMI) käyttöä. Jos alussa olisi mietitty, miten tämä kannattaa tehdä, sen sijaan että mietittiin, miten tämä voidaan tehdä tällä ohjelmointikielellä, olisi voitu päätyä käyttämään jotain muuta kuin Pythonia. Toisaalta, kun on tarpeen käyttää erityisiä Microsoftin luomia Windows-rajapintoja, voidaan ehkä kyseenalaistaa minkä tahansa muun, kuin erityisesti Windowsin natiiveille ohjelmistoille suunnatun ohjelmointikielen käyttö.

Vaikka Python on alustariippumaton, vaatii se tarkkaa huolta käytettävän tulkin versiosta. Työn aikana havaittiin ongelmia ohjelmiston toiminnassa jo pienenkin (3.1.2 → 3.2) versiomuutoksen kanssa. Jos erityistä paketointia ei tehdä, tulee asiakasympäristön tulkin yhteensopivuudesta pitää tarkkaan huolta. Kehitysympäristössä käytettävän tulkin on myös oltava sama kaikilla ohjelmointiin osallistuvilla. Mikäli tuotantoa halutaan jatkaa käyttäen Pythonia, voitaisiin harkita siirtymistä CPythonista IronPythonin käyttöön. Sen eduiksi voidaan katsoa, että se on lähempänä

kohdealustaansa. CPython sellaisenaan on suunnattu laajalle joukolle erilaisia järjestelmiä, eikä siitä syystä mahdollista yhtä suoraviivaista kehitystyötä yhdelle tietylle alustalle.

Myös ohjelman vaatimat fyysiset resurssit ovat tärkeä puoli tuotantokielen valinnassa. Kun ohjelmisto halutaan toteuttaa mahdollisimman vähän tietokoneen muistia ja suoritinaikaa käyttävänä, voitaisiin ohjelmointikieli valita näiden vaatimusten mukaan. Toisaalta modernit tietokoneet sisältävät useita gigatavuja muistia, eikä Python-sovelluksen 20-30 Mt muistinkäyttö siihen verrattuna välttämättä ole paljon.

Käytettävää ohjelmointikieltä voidaan harkita myös ohjelmistonlevityksen kannalta, sillä ilman erillistä paketointia on Python-kielisessä toteutuksella käytössä vain irrallisia komentosarjatiedostoja. Opinnäytteen asiakasohjelmisto on toistaiseksi tarkoitettu käytettäväksi Windows-käyttöjärjestelmällä ja se käyttää Microsoftin erityisesti Windows-alustalle kehittämää WMI:tä. On ehkä mahdollista, että ohjelmiston toimintaa laajennetaan myöhemmin koskemaan muitakin käyttöjärjestelmiä. Vaikka järjestelmällä olisikin sopiva Python-tulkki, jouduttaisiin ohjelmiston toiminta siitä huolimatta toteuttamaan uudestaan, sillä WMI ei ole käytettävissä kuin Microsoftin käyttöjärjestelmissä. Asiakasohjelmiston toiminta voi mahdollisesti olla triviaalia kääntää toiselle alustalle, mutta tiedonkeruu joudutaan aina toteuttamaan järjestelmäkohtaisesti.

7 POHDINTA

Työ toteutui omasta mielestäni pitkähköllä aikajänteellä. Uuden ohjelmointikielen opetteluun sai käytettyä paljon aikaa, ennen kuin omat taidot tuntuivat riittäviltä varsinaiseen ohjelmointityöhön. Python-kielen syntaksin tullessa tutummaksi, sai myös perehtyä Windows-järjestelmän tiedonhakuun ja WMI:n toimintaan. Näiden käyttöä varten taas tuli tutkia ja opetella käyttämään Pythonissa tarvittavia laajennoksia PyWin32 ja Python WMI. Myös vaatimusmäärittelyn miettiminen ja kirjoittaminen vei oman aikansa.

Esitysseminaarissa sain jonkin verran palautetta koskien työn laajuutta. Työmäärää pidettiin suurena ja tästä syystä työn ehkä pelättiin jäävän kesken. Mielestäni selvisin kuitenkin hyvin, ja työ säilytti mielenkiintonsa loppuun saakka ja pidemmällekin. Vaatimusmäärittelyyn kirjatut toiminnallisuudet ja hieman enemmänkin saatiin opinnäytetyön aikataulussa toteutettua ja ohjelmointityön lopputuloksena oli toimiva ohjelmisto asiakaskoneen tilanseurantaan. Ohjelmistolla saadaan haettua järjestelmästä haluttu data, ja se voidaan välittää edelleen verkon yli palvelimelle. Työ oli myös ilmeisesti ensimmäinen Python-kieltä käsittelevä opinnäytetyö OAMK:n Liiketalouden yksikössä.

Pythonin opettelu oli mielenkiintoista. Se on ohjelmointikielenä monipuolinen, mutta silti helppo oppia ja ottaa käyttöön. Työn edetessä se myös osoittautui pystyväksi Windows-alustan ohjelmistokehitykseen. Ennen kuin yhtään riviä koodia voitiin kirjoittaa tuli opetella käytettävän kielen syntaksi ja tärkeimmät moduulit. Hyviksi tiedonlähteiksi oppimisen aikana osoittautuivat J. Kasurisen kirja *Python 3 Ohjelmointi* sekä Mark Pilgrimin kirjoittama verkko-opas *Dive Into Python 3*, joka on luettavissa osoitteessa http://diveintopython3.org/. Tärkeimmäksi lähteeksi varsinaisen kehityksen aikana osoittautuivat kuitenkin Python-kieltä kehittävän Python Software Foundationin omat dokumentaatiot.

Jotain kehittämisen varaa virallisillakin dokumenteilla on. XML-elementtien merkkijonoksi muuntaminen tehdään käyttäen `xml.etree.ElementTree`-moduulin `tostring`-funktiota. Kehitystyön aikana havaittiin kuitenkin ongelma, kun kehitysalustan Python-asennus päivitettiin versiosta 3.1.2 versioon 3.2. Aiemmassa versiossa `tostring`-funktio palauttaa nimensä mukaisesti Pythonin merkkijonoja käsittävän `str`-olion, kun taas uudempi 3.2-asennus

palauttaa samaa funktiota käyttäen `bytes`-luokan olion, joka pitää erikseen muuntaa merkkijonoksi eksplisiittisellä muunnoksella. Tätä muutosta ei ollut vielä raportin kirjoitushetkellä merkitty virallisiin dokumentaatioihin.

Työn alussa jouduin keskittymään etupäässä Python-kielen opetteluun. Jos Pythonista olisi ollut jo alussa vahva osaaminen, olisi tällöin varmaankin päädytty käyttämään IronPython-toteutusta tai jopa täysin eri ohjelmointikieltä. Tästä huolimatta, tai ehkä juuri siksi, työ oli opettavaista ja uskoisin, että uuden ohjelmointikielen opettelu parantaa ohjelmointitaitoja yleisestikin. Osaamisen kasvaessa onkin helpompi tehdä ohjelmointityöhön liittyviä käyttökielen valintoja. Pythonin lisäksi työssä pääsi tutustumaan Windows-järjestelmän tietorakenteeseen ja etenkin WMI:n toimintaan. Aiheena se ei ollut etukäteen kovinkaan tuttu, joten uutta opittiin paljon.

Muiden työkiireiden ohella toimeksiantajan henkilöstö oli tarvittaessa valmis antamaan palautetta toimeksiannosta ja lisäselvitystä sellaista vaativista kohdista. Ohjelmisto suunniteltiin täyttämään toimeksiantajan omia tarpeita ylläpitotyössä. Tämän lisäksi sovelluksella voi olla muutakin kaupallista potentiaalia. Toimeksiantajan kanssa on kehitystyön jälkeen käyty keskustelua ohjelmiston jatkokehityksestä ja saatu paljon hyviä ideoita tulevasta kehityksestä. Jos kehitystä jatketaan vielä eteenpäin, uskoisin ohjelmiston tulevan hyväksi lisäksi toimeksiantajan palveluihin.

LÄHTEET

Autodesk. 2011. Maya 2012 Frequently Asked Questions. Hakupäivä 6.4..2011 http://images.autodesk.com/adsk/files/maya_2012_frequently_asked_questions_us.pdf

Avariento, J. 2011. Python for iPod. Hakupäivä 7.4.2011 http://ciberjacobo.com/python-ipod

Django Software Foundation. 2011. The Web framework for perfectionists with deadlines. Hakupäivä 6.4.2011 http://www.djangoproject.com/

DMTF. 2011a. Web-Based Enterprise Management. Hakupäivä 16.4.2011 http://www.dmtf.org/standards/wbem

DMTF. 2011b. CIM FAQ. Hakupäivä 17.4.2011 http://dmtf.org/about/faq/cim_faq

Gartner. 2011. IT Definitions and Glossary. Hakupäivä 6.5.2011 http://www.gartner.com/technology/research/it-glossary/

Golden, T. 2009a. WMI v1.4.7 documentation. Hakupäivä 12.2.2011 http://timgolden.me.uk/python/wmi/index.html

Golden, T. 2009b. wmi Tutorial. Hakupäivä 3.5.2011 http://timgolden.me.uk/python/wmi/tutorial.html

Hammond, M. 2009. Python for Win32 Extensions Help. Elektroninen ohjetiedosto.

Henstridge, J. 2006. GIMP Python Documentation. Hakupäivä 6.4.2011 http://www.gimp.org/docs/python/index.html

IBM z/OS Basic Skills Information Center. 2010. Application programming on z/OS. Hakupäivä 19.3.2011 http://publib.boulder.ibm.com/infocenter/zos/basics/topic/com.ibm.zos.zappldev/zappldev_book.pdf

Jython. 2011a. JythonFaq. Hakupäivä 7.4.2011 http://wiki.python.org/jython/JythonFaq/GeneralInfo

Jython. 2011b. HISTORY OF THE SOFTWARE. Hakupäivä 9.4.2011 http://www.jython.org/archive/22/history.html

Kasurinen, J. 2009. Python 3 ohjelmointi. Jyväskylä. Docendo.

Laird, C.2010. What an IronPython user should know about Python 3. Hakupäivä 19.3.2011 http://www.itworld.com/development/104506/python-3-and-ironpython

Lundh, F. 2011. ElementTree Overview. Hakupäivä 6.5.2011 http://effbot.org/zone/element-index.htm

Microsoft. 2000. Windows Management Instrumentation: Background and Overview. Hakupäivä 12.2.2011 http://msdn.microsoft.com/en-us/library/ms811553.aspx

Microsoft. 2008. Open Source at Microsoft. IronPython: Engaging the Python Community in Its Own Language. Hakupäivä 9.4.2011 http://download.microsoft.com/download/8/2/f/82f65f5e-d791-4199-9ce4-25e772682731/ironpython_final.pdf

Microsoft. 2010. General MFC Topics. Hakupäivä 19.3.2011 http://msdn.microsoft.com/en-us/library/583ya1kc.aspx

Microsoft. 2011a. WMI Architecture. Hakupäivä 16.4.2011 http://msdn.microsoft.com/en-us/library/aa394553(v=VS.85).aspx

Microsoft. 2011b. Windows Management Instrumentation. Hakupäivä 1.5.2011 http://msdn.microsoft.com/en-us/library/aa394582(v=VS.85).aspx

Microsoft. 2011c. Win32_LogicalDisk Class. Hakupäivä 17.4.2011 http://msdn.microsoft.com/en-us/library/aa394173(v=VS.85).aspx

Microsoft. 2011d. Querying with WQL Hakupäivä 1.5.2011 http://msdn.microsoft.com/en-us/library/aa392902(v=VS.85).aspx

Microsoft. 2011e. Using WMI. Hakupäivä 21.4.2011 http://msdn.microsoft.com/en-us/library/aa393964(v=VS.85).aspx

Microsoft. 2011f. Win32_Product Class. Hakupäivä 3.5.2011 http://msdn.microsoft.com/en-us/library/aa394378(v=VS.85).aspx

Microsoft. 2011g. CIM_DATETIME. Hakupäivä 3.5.2011 http://msdn.microsoft.com/en-us/library/aa387237(v=VS.85).aspx

Microsoft. 2011h. Installed Applications Classes. Hakupäivä 3.5.2011 http://msdn.microsoft.com/en-us/library/aa390887(v=VS.85).aspx

Nokia. 2011. Nokia Open Source. Hakupäivä 7.4.2011 http://wiki.forum.nokia.com/index.php/Nokia_Open_Source

Python Software Foundation. 2010a. What's New In Python 3.0. Hakupäivä 7.5.2011 http://docs.python.org/release/3.0.1/whatsnew/3.0.html

Python Software Foundation. 2010b. The Python Standard Library. Hakupäivä 23.3.2011 http://docs.python.org/release/3.1.2/library/index.html

Python Software Foundation. 2010c. xml.etree.ElementTree — The ElementTree XML API. Hakupäivä 6.5.2011 http://docs.python.org/release/3.1.2/library/xml.etree.elementtree.html

Python Software Foundation. 2010d. http.client — HTTP protocol client. Hakupäivä 6.5.2011 http://docs.python.org/release/3.1.2/library/http.client.html

Python Software Foundation. 2010e. configparser — Configuration file parser. Hakupäivä 6.5.2011 http://docs.python.org/release/3.1.2/library/configparser.html

Python Software Foundation. 2010f. platform — Access to underlying platform's identifying data. Hakupäivä 6.5.2011 http://docs.python.org/release/3.1.2/library/platform.html

Python Software Foundation. 2011. Should I use Python 2 or Python 3 for my development activity? Hakupäivä 16.4.2011 http://wiki.python.org/moin/Python2orPython3

Tuininga, A. 2011. cx_Freeze. Hakupäivä 29.3.2011 http://cx-freeze.sourceforge.net/cx_Freeze.html