Anssi Taskinen

Monialusta-mobiilisovelluksen koontiympäristö

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

10.4.2013

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Anssi Taskinen Monialusta-mobiilisovelluksen koontiympäristö 30 sivua + 2 liitettä 10.4.2013
Tutkinto	insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Yliopettaja Auvo Häkkinen Kehityspäälikkö Hanna Torri

Anygraaf Oy on graafisen alan ohjelmistoja valmistava yritys. Tässä insinöörityössä esitellään Anygraafin mobiilisovelluksen AnyReaderin automatisoidun koonti- ja konfigurointiympäristön toteuttaminen. AnyReader-sovelluksesta tuotetaan asiakkaille räätälöityjä versioita.

Sovelluksen konfiguroinnilla tarkoitetaan sovellusten räätälöintiin ja kääntämiseen vaikuttavien asetusten laittamista paikoilleen ennen varsinaista sovellusten kääntämistä. Tuotetun koontiympäristön tavoitteena oli automatisoida työläs monialustasovelluksen räätälöinti- ja käännösprosessi.

Jenkins CI on jatkuvan integroinnin koontipalvelin, jota Anygraaf käyttää entuudestaan muiden projektiensa koontiin. Toteutettu koontiympäristö on hajautettu useammalle koontipalvelimelle ja se liitettiin osaksi jo olemassa olevaa ympäristöä. AnyReaderin konfigurointiin käytetään Python-ohjelmointikielellä toteutettua Pynt-koontikirjastoa. Valmiit sovellusten asennuspaketit siirretään tiedostopalvelimelle, josta ne julkaistaan Apache HTTP-palvelimen päälle Web.py-ohjelmistokehyksellä toteutetulla Internet-sivustolla.

Toteutettua koonti- ja konfigurointiympäristöä käytetään asiakassovellusten koontityökaluna ja se vähentää AnyReaderin päivitysten yhteydessä tehtävää työtä. Tämän insinöörityöraportin lopussa esitellään mahdollisia jatkokehityksideoita koontiympäristön laajentamiseksi.

Avainsanat	jatkuva integrointi, koontiympäristö, konfiguraation hallinta

Author(s) Title Number of Pages Date	Anssi Taskinen Integration Environment for Multi-platform Mobile Application 30 pages + 2 appendices 10 April 2013
Degree	Bachelor of Engineering
Degree Programme	Information technology
Specialisation option	Software Engineering
Instructor(s)	Auvo Häkkinen, Principal Lecturer Hanna Torri, R & D Manager

Anygraaf is a software company specialized in the graphical field. Anygraaf's mobile software AnyReader is a multi-platform application designed for reading publications on a tablet and mobile platforms. This thesis introduces the implementation of AnyReader's continuous integration and configuration environment. Anygraaf produces customized versions of AnyReader. The configuration of the application means setting up the customization based parameters before the real building process. The main goal of this building environment is to automate tricky and laborious work that has to be done in the update and building process of the multi-platform mobile application.

Jenkins CI is a continuous integration server that is already being used for the building of Anygraaf's other development projects. The building and configuration system of AnyReader is decentralized to different servers and it is attached to the existing integration system. The configuration system is implemented by a Python programming language based Pynt configuration framework. At the end of the building process the installation packages of AnyReader are transferred to the file server where they get published on a website implemented by a Python based Web.py framework built on Apache HTTP server.

The configuration and building environment of AnyReader is used as a building tool of the client's AnyReader versions and it reduces the work that has to be done for publishing the new version of the application. At the end of the thesis, further development possibilities that expand the implemented environment are introduced.

Keywords	continuous integration, building environment, configuration management

Sisällys

Lyhenteet

CI	Continuous integration / Jatkuva koonti. Tarkoittaa sovelluksen jatkuvaa testaamista ja tuotantoon saattamista.
CIFS	Common Internet Filesystem. Microsoftin kehittämä verkkotiedostojärjestelmä-protokolla, joka osaa hyödyntää kehittyneitä verkkotekniikoita. Protokollaa käytetään myös erilaisten resurssien jakamiseen verkon yli.
DCOM	Distributed Component Object Model. Microsoftin patentoima Internet-yhteyden avulla toimiva sovellusosien välinen kommunikointiteknologia.
HG	Elohopean (engl. mercury) kemiallinen merkki, jota käytetään yleisesti lyhenteenä Mercurial-versionhallintajärjestelmästä.
HTTP	Hyper Text Transfer Protocol. TCP-yhteyteen perustuva siirtoprotokolla, jota esimerkiksi Internet-selaimet hyödyntävät kommunikoidessaan verkkopalvelimen kanssa.
IAP / IAB	In App Purchase / In App Billing. Älypuhelinvalmistajien ja mobiililaitteiden käyttöjärjestelmävalmistajien tarjoama laskutus- ja maksurajapinta, jonka avulla mobiilisovelluksen sisältä tehtävät ostokset saadaan laskutettua suoraan käyttäjän laitekohtaiselta tililtä.
IDE	Integrated development environment. Ohjelmoijalle sovelluskehitystyöhön kattavat työkalut tarjoava ohjelmointiympäristö.
iOS	iPhone Operating System. Applen valmistaman iPhone-puhelimen käyttöjärjestelmä.
JDK	Java Development Kit. Oraclen julkaisemat Java-ohjelmistokehitykseen tarvittavat työkalut.
JNLP	Java Network Launch Protocol. Java-liitännäinen, jonka avulla käynnistetään Java-sovelluksia Internet-sivuston kautta.
JSON	JavaScript Object Notation. Yksinkertainen tiedontallennusmuoto.

NDK	Native Development Kit. Kirjasto, joka mahdollistaa C- ja C++-kielellä kirjoitettujen sovellusosien liittämisen osaksi Androidin Java-sovellusta.
QT	Alun perin Nokian julkaisema sovelluskehys, jonka avulla kehittäjät voivat tehdä graafisia sovelluksia useille eri alustoille.
SDK	Software Development Kit. Joukko ohjelmistokehitykseen tarvittavia työkaluja, joiden avulla voidaan toteuttaa sovellus tietylle alustalle.
SSH	Secure Shell. Protokolla, joka on tarkoitettu turvalliseen tiedonsiirtoon turvattomassa verkossa kahden verkkoon liitetyn tietokoneen välillä.
TCP	Transmission Control Protocol. Tietoliikenneprotokolla, jolla luodaan yhteyksiä tietokoneiden välille.
WMI	Windows Management Instrumentation. Joukko Microsoft Windows -ajureiden laajennuksia, jotka mahdollistavat skriptipohjaisen etähallinnan Windows-tietokoneille.

1 Johdanto

Anygraaf on graafisen alan ohjelmistoja valmistava yritys, jonka asiakkaat koostuvat pääsääntöisesti lehtitaloista ja muista julkaisijoista. Yrityksen tuotteisiin kuuluva AnyReader on lehtien ja muiden julkaisujen lukemiseen suunniteltu monialusta-mobiili- ja tabletsovellus. AnyReader on toteutettu Android-, Symbian- ja iOS-alustoille ja sen toiminta perustuu C++-ohjelmointikielellä toteutettuun sovellusrunkoon, joka on liitetty osaksi laitekohtaisia toteutuksia eri alustojen tarjoamien natiivien liitäntärajapintojen avulla. Sovelluksesta on tarkoitus tuottaa sen tilanneille julkaisijoille räätälöityjä versioita. Yksittäiselle julkaisijalle räätälöidyt AnyReader-sovellukset julkaistaan alustakohtaisissa kaupoissa asiakkaan haluamilla nimillä ja asiakkaan omistamina sovelluksina. Räätälöityyn sovellukseen upotetaan asiakkaan oma verkkokauppa sisällön jakelua varten ja sovelluksen ulkoasu voidaan räätälöidä asiakkaan toiveiden mukaiseksi.

AnyReaderin vahvuus kilpaileviin tuotteisiin verrattuna on loppukäyttäjän laitteessa tapahtuva dynaaminen taitto, joka tekee sisällön tuottamisesta nopeaa ja helppoa. AnyReaderiin tuotettavat kuvat, videot ja tekstit hallinnoidaan Anygraafin tarjoamilla sisällönhallintajärjestelmillä. AnyReaderia ja sen toiminnallisuuksia kehitetään jatkuvasti ja uuden version valmistuttua jokaiselle asiakkaalle täytyy tällä hetkellä käsin rakentaa omat räätälöidyt versiot kullekin alustatoteutukselle. AnyReaderin kehitystyön pullonkaulaksi onkin muotoutunut työläs ja aikaa vievä sovelluksen päivitysprosessi.

Tässä insinöörityössä esitellään AnyReaderin automatisoidun koonti- ja konfigurointiympäristön toteutus. Ympäristö huolehtii asiakaskohtaisten sovellusten kustomoinnista sekä julkaisee valmiit asennuspaketit tiedostopalvelimelle. Anygraaf käyttää jo entuudestaan Jenkins CI -koontiympäristöä muiden projektiensa koontiin, joten tuotettava järjestelmä tulisi liittää osaksi jo olemassa olevaa ympäristöä. CI-lyhenne (Continuous Integration) tarkoittaa tuotettavan sovelluksen jatkuvan testauksen sekä tuotantoon saattamisen toteutusta. Mobiilisovellusten kääntäminen eri alustoille vaatii myös eri käyttöjärjestelmillä toimivien palvelinten keskinäisen kommunikoinnin, johon Jenkins tarjoaa kattavat työkalut. Useammalle eri alustalle tarkoitetun mobiilisovelluksen konfigurointi vaatii monialustaympäristön lisäksi melko paljon yhteisiä asetuksia sekä alustakohtaista konfigurointia. Tämän vuoksi pelkän Jenkins CI -koontiympäristön tarjoamien työkalujen käyttäminen koituisi työlääksi ja samoja asetuksia täytyisi määritellä useammalle alustalle moneen kertaan. Konfigurointiprosessi ja suoritettavat komennot

pyritään saamaan mahdollisimman yksinkertaisiksi ja yhtenäisiksi jokaisella alustalla käyttämällä alustariippumatonta konfigurointityökalua.

Konfigurointi- ja koontityökaluja on tarjolla useita kymmeniä, joista tunnettuja esimerkkejä ovat GNU Make, Apache Ant ja Apache Maven. Anygraafissa ei entuudestaan löytynyt juurikaan kokemusta näistä työkaluista, joten myös muita vaihtoehtoja koontityökalun valinnassa tuli harkita. Tarjolla olevia vaihtoehtoja tutkittaessa esille nousi Python-ohjelmointikielellä toteutettu Pynt-konfigurointityökalu, joka tarjoaa joustavan rajapinnan alustakohtaisten- sekä yhteisten konfigurointitehtävien toteuttamiselle. Tämä työkalu vaikutti lupaavalta, sillä Anygraafissa on jo entuudestaan kokemusta Python-ohjelmointikielestä. Lisäksi Jenkins tarjoaa kattavan käskyrajapinnan Python-skripteille. AnyReaderin konfigurointiympäristö päätettiin toteuttaa Pynt-koontikirjaston avulla.

Tämän insinöörityön luvussa 2 tutustutaan ensin mobiilisovellusten kyhitysympäristöön ja siihen kuuluviin osakokonaisuuksiin, jonka jälkeen esitellään AnyReader-sovelluksen nykyinen ja tuleva kehitysympäristö. Uutena osana AnyReaderin kehitysympäristöön liitetään automatisoitu koonti- ja konfigurointiympäristö. Luvussa 3 kerrotaan mitä tarkoitetaan mobiilisovellusten konfiguroinnilla sekä esitellään konfiguroitavan monialustasovelluksen konfiguroinnille asettamat vaatimukset. Luvussa 4 tarkastellaan koonti- ja konfigurointiympäristön toimintaa aikaan rinnastettuna sekä käydään läpi AnyReaderin koontiprosessin kulku lähdekoodista valmiiksi sovelluksen asennuspaketeiksi. Lopuksi tarkastellaan automatisoidun koontiympäristön Anygraafille tuomia etuja sekä esitellään muutamia jatkokehitysideoita.

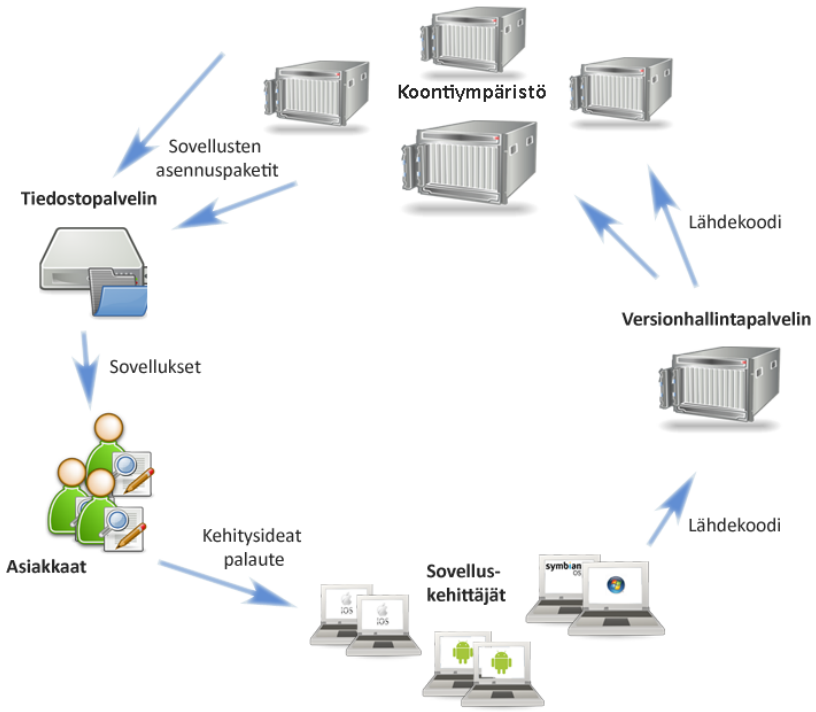
2 Mobiilisovelluksen kehitysympäristö

Mobiilisovellusten kehitysympäristöihin liittyy nykyään monia osakokonaisuuksia, varsinkin jos kyseessä on monelle eri alustalle tuotettava sovellus. Tarvittavat osat koostuvat vähintään kehittäjien työasemista, mutta nykyään lähes jokainen sovelluksia kehittävä yhteisö hyödyntää jotakin lähdekoodin versionhallintajärjestelmää. Etenkin iteratiivisten kehitysmenetelmien ansiosta myös koontiympäristöt ovat yleistyneet yhä keskeisemmäksi osaksi kehitysympäristöjä.

Anygraafin mobiilisovelluksen AnyReaderin kehitysympäristössä sovellusta kehitetään kehittäjien työasemissa, joista lähdekoodiin tehdyt muutokset viedään versionhallintaan jaettaviksi muille kehittäjille. Uuden sovellusversion valmistuessa AnyReader kasataan kehittäjien työasemilla ja valmiit sovelluspaketit siirretään manuaalisesti tiedostopalvelimelle jaettavaksi. Kehitysympäristöön kuuluva tiedostopalvelin toimii sovelluspakettien latauspaikkana testikäyttäjille ja sieltä ladataan myös alustakohtaisiin kauppoihin ladattavaksi vietävät asennuspaketit. Ennen koontiympäristön liittämistä osaksi kehitysympäristöä uuden sovellusversion aikaansaamiseksi vaaditaan useamman kehittäjän samanaikainen panostus, sillä sovelluksen alustakohtaiset toteutukset halutaan lähes poikkeuksetta julkaista samanaikaisesti.

2.1 Koontiympäristön liittäminen osaksi kehitysympäristöä

Kuvassa 1 on AnyReader-sovelluksen tuleva sovelluskehitysympäristö sekä kehityksessä käytettävän kehitysmenetelmän työvaiheet. Kehitysympäristön tuotannollinen sydän on edelleen kehittäjien työasemat ja niille asennetut kehitysympäristöt (IDE). IDE-lyhenne (Integrated Development Environment) tarkoittaa kehittäjien työasemille asennettua lähdekoodin kehittämiseen käytettävistä työkaluista muodostuvaa kehitysympäristöä [1]. Työasemilta tuotettu lähdekoodi siirretään versionhallintaan, josta koontipalvelimet kokoavat valmiit sovellukset tiedostopalvelimelle jaettavaksi. Asiakkaat saavat nopeasti ladattua uuden sovellusversion käyttöönsä tiedostopalvelimelta, jolloin myös sovellusversioon tehdyistä muutoksista saadaan palautetta nopeammin.



Kuva 1. AnyReaderin kehitysympäristö ja kehitysmenetelmän työvaiheet

Uutena osana AnyReaderin kehitysympäristöön liitetään koontipalvelin, jota käytetään jo Anygraafin muiden sovellusten kokoamisessa. AnyReaderin koontimääritys eroaa Jenkins CI -koontipalvelimelle aiemmin määritetyistä koonneista ainakin siten, että koontiprosessi täytyy suorittaa usealla eri käyttöjärjestelmällä, jotta AnyReaderin kaikkien alustaversioiden kokoamisprosessit saadaan automaattisiksi. Alustakohtaisten rajoitusten takia kaikkia sovelluksen alustaversioita ei siis voida kääntää samalla palvelimella.

Esimerkiksi Apple rajoittaa kehitysympäristönsä asennusmahdollisuuksia niin, että se voidaan asentaa ainoastaan Applen valmistamiin laitteisiin [2]. iOS-käyttöjärjestelmää (iPhone Operating System) käyttävien alustojen kokoamisprosessi halutaan saada myös automaattiseksi, joten kehitysympäristöön täytyy liittää Applen fyysinen tietokone. Jenkins CI -koontipalvelimeen on liitetty Applen palvelimen lisäksi ikään kuin orjapalvelimeksi Windows 7 -käyttöjärjestelmällä varustettu palvelin, joka vastaa Symbian- ja Windows-alustalle tarkoitettujen sovellusten koonnista. Lisäksi koontiympäristään kuuluu Windows 8 -käyttöjärjestelmällä toimiva palvelin, jolla kääntyvät Anygraafin muut projektit. Anygraafin kehitysympäristöön kuuluu kokonaisuudessaan neljä koontipalvelinta, joilla kaikilla on omat määrätyt tehtävänsä.

Android-sovellusten kehittäminen eroaa muusta mobiilikehityksestä ainakin kehitysprosessin alustariippumattomuuden ansiosta. Lisäksi Android-sovellukset voidaan kääntää millä tahansa alustalla. Kehittäjien työasemilta tuotettu koodi viedään versionhallintaan. Jenkins CI -koontipalvelin käy tutkimassa määrätyin väliajoin versionhallinnan tietyn kehityshaaran muutoksia. Koontipalvelin aloittaa uuden AnyReader-version konfigurointi- ja paketointiprosessin, kun se havaitsee muutoksen kehityshaarassa. Myös tiedostopalvelimella tapahtuva AnyReaderin asennuspakettien julkaisu automatisoidaan koontiympäristön käyttöönoton yhteydessä. Lisäksi tiedostopalvelimelle tehdään verkkosovellus, jonka kautta asennuspaketit saadaan ladatuksi Internet-selaimen läpi.

2.2 Työasemat ja sovelluskehitys

AnyReaderia kehitetään tiimissä, jonka henkilömäärä vaihtelee projektin vaiheesta riippuen. Sovelluskehittäjien työasemille on asennettu kehitettävän alustaversion vaatimat kehitysympäristöt. Android-sovelluskehitykseen käytetään Eclipse-kehitysympäristöä, joka kääntää Android-projektit käyttäen Ant-koontiskriptiä. Java-ohjelmointikielellä toteutettuun Android-sovellukseen on liitetty kaikille alustoille yhteinen C++-ohjelmointikielellä toteutettu sovellusrunko NDK-liitäntärajapinnan avulla. AnyReaderin iOS-version kehittäjät käyttävät xCode-kehitysympäristöä ja Symbian- sekä Windows-versioita kehitetään QT Creator -ohjelmistolla. QT Creator -sovellusta käytetään myös kaikille alustoille yhteisen sovellusrungon kehittämiseen. QT on alun perin Nokian julkaisema sovelluskehys, jonka avulla voidaan tehdä graafisia sovelluksia useille alustoille.

AnyReaderin yhteinen sovellusrunko hyödyntää useita kolmansien osapuolten tuottamia sovelluskirjastoja. Esimerkiksi Boost-kirjastoa [3] käytetään AnyReaderin säikeistämiseen. Libcurl-kirjasto [4] on asiakaspäähän tarkoitettu tiedonsiirtokirjasto, joka tarjoaa kommunikointirajapinnan Internet-yhteyden läpi palvelimelle yleisimpiä protokollia hyödyntäen. AnyReader lataa julkaisut, niihin liittyvät metatiedot ja kuvat palvelimilta Libcurl-kirjaston avulla. JsonCpp-kirjasto [5] tarjoaa JSON-muotoisen datan lukemiseen ja kirjoittamiseen tarvittavat työkalut C++-ohjelmointikielellä. JSON-lyhenne (JavaScript Object Notation) tarkoittaa alun perin JavaScript-ohjelmointikielessä käytettyä yksinkertaista tiedon mallinnusmuotoa. Minizip-kirjaston avulla puretaan palvelimelta ladatut julkaisupakkaukset.

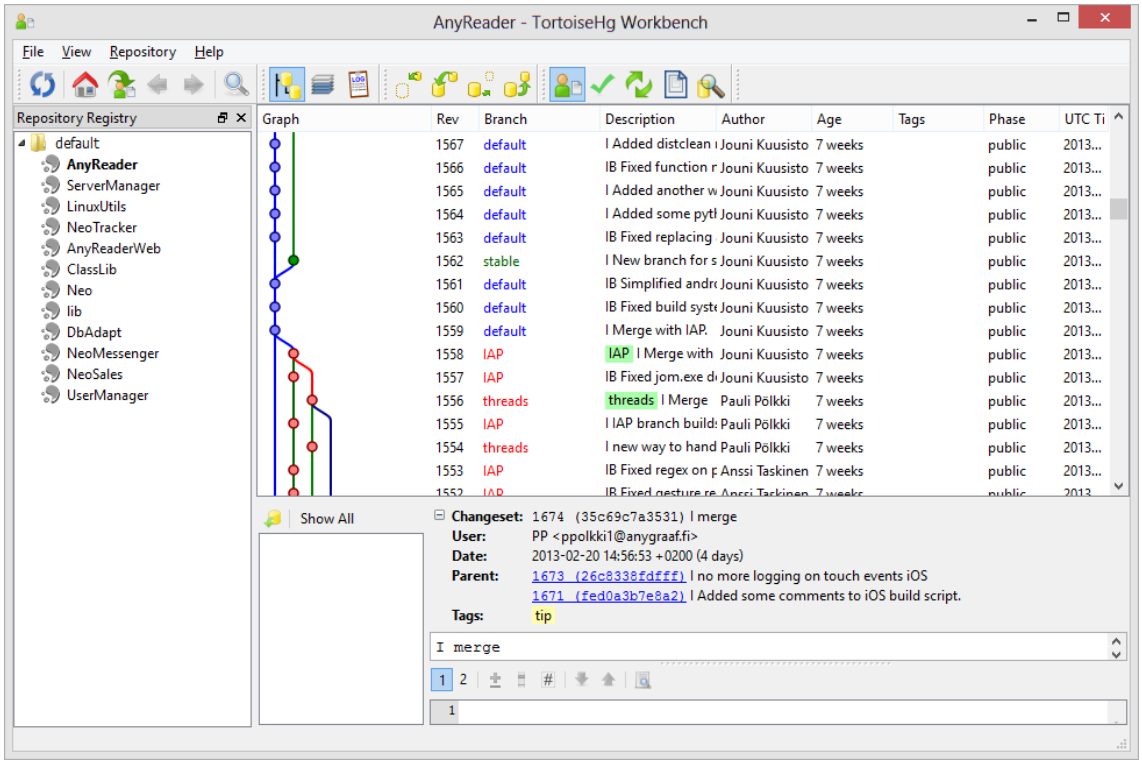
AnyReaderia kehitetään iteratiivisella kehitysmenetelmällä. Aina noin kaksi viikkoa kestävän sprintin päätteeksi saadaan uudet asennuspaketit tiedostopalvelimelle jaettaviksi. Tämän vuoksi projektin kehitystyössä on luontevaa alkaa hyödyntämään koontijärjestelmän tuomia etuja. Iteratiivisten kehitysmenetelmien perusidea on saada aikaan mahdollisimman nopeasti jotakin näkyvää. Sovelluskehityksen osalta se yleensä tarkoittaa uutta versiota kehitettävästä tuotteesta. Suhteellisen lyhyiden sprinttien takia versioiden kääntämistä ja testaamista tulisi siis erittäin paljon. Koontiympäristöjen avulla on mahdollista automatisoida suurin osa koonti- ja käännösprosessista, jolloin kehittäjien aikaa voidaan hyödyntää muuhun.

2.3 Hajautetun versionhallintajärjestelmän hyödyntäminen

AnyReader-kehityksessä käytössä ollut Subversion-versionhallintajärjestelmä osoittautui kehitystarpeisiin riittämättömäksi, koska useampi kehittäjä työsti samanaikaisesti samaa uutta ominaisuutta. Tuotettu lähdekoodi oli tärkeä saada kaikkien käyttöön mahdollisimman nopeasti, mutta samalla piti säilyttää toimiva versio lähdekoodeista kaikkien saatavilla. AnyReaderin kehitysympäristöön liitettävä koontiympäristö on tarkoitus määrittää julkaisemaan uudet asennuspaketit tiedostopalvelimelle jokaisen versionhallintaan viedyn muutoksen jälkeen. Versionhallintaan viedään muutoksia suhteellisen usein, jonka takia koontiympäristö kokosi käytännössä jatkuvasti uusia AnyReader-versioita. AnyReaderin koontiprosessi voitaisiin ajastaa koontipalvelimelle, jotta uusia käännöksiä ei syntyisi niin usein, mutta samalla menetettäisiin kehittäjillä oleva koontiprosessin kontrolli. Toinen vaihtoehto ongelman ratkaisemiseksi oli käytössä olevan Subversion-versionhallintajärjestelmän kehityshaarojen hyödyntäminen, joka kuitenkin koettiin erittäin hankalaksi.

Hajautetut versionhallintajärjestelmät [6] tarjoavat Subversion-versionhallintajärjestelmää joustavammat ja monipuolisemmat ominaisuudet kehityshaarojen hyödyntämiseen. Hajauttamisella tarkoitetaan sitä, että jokaisella versionhallinnan käyttäjällä on oma täydellinen kopio koko projektin repositorystä omalla työasemallaan. Tämä mahdollistaa kehityshaarojen ja sitoutusten (commit) tekemisen paikallisena operaationa. Palvelimelle voidaan viedä useampi sitoutus kerrallaan ja kehittäjät voivat päivittää oman paikallisen kopion sovelluksesta mihin tahansa kehityshaaraan tai revisioon paikallisena operaationa ilman Internet-yhteyttä. Tunnetuimpia hajautettuja versionhallintajärjestelmiä ovat Git ja Mercurial (HG). Mercurial-versionhallintajärjestelmästä käytet-

tävä lyhenne HG tulee elohopean (engl. mercury) kemiallisesta merkistä. Molemmissa versionhallintajärjestelmissä on omat hyvät puolensa, mutta Anygraafin käyttötarkoitukseen jälkimmäinen tarjoaa enemmän mahdollisuuksia. Mercurial on helposti laajennettavissa modulaarisen rakenteensa ansiosta. Lisäksi se on toteutettu Python-ohjelmointikielellä, jota käytetään muissakin Anygraafin järjestelmissä. Haittapuolena hajautetussa versionhallintajärjestelmässä on ainakin se, että Subversion-versionhallintajärjestelmästä poiketen revisionumerot vaihtelevat repositoryn kopiosta riippuen. Revisionumero 1600 saattaa toisessa kopiossa olla 1595 riippuen paikallisten sitoutusten ajankohdista. Tämä ominaisuus aiheuttaa ongelman AnyReaderin nykyiseen versionumerointiin, joka pohjautuu osittain myös revisionumeroihin. Lisää ongelmasta ja sen ratkaisusta on kerrottu luvussa 4.1. Kuvassa 2 on ruutukaappaus AnyReader-projektin HG Workbench -näkymästä, josta erottuvat selkeästi eri kehityshaarat ja niiden hyödyntäminen AnyReaderin sovelluskehityksessä.



Kuva 2. Mercurial-versionhallintajärjestelmän HG Workbench -näkymä

Kuvan tilanteessa on alettu kehittää kahta uutta ominaisuutta *IAP*- ja *threads*-kehityshaaroissa. IAP-kehityshaarassa kehitetään AnyReader-sovelluksen sisältä tapahtuvaan julkaisujen myyntiin tarvittavaa rajapintaa. Lyhenne IAP (In App Purchase) tarkoittaa mobiililaitteiden käyttöjärjestelmävalmistajien tarjoamaa laskutus- ja maksu-

rajapintaa, jonka avulta mobiilisovellusten sisältä myydyt tuotteet saadaan laskutetuksi suoraan käyttäjän laitekohtaiselta tililtä. Threads-kehityshaarassa kehitetään AnyReader-sovelluksen säikeistämiseen käytettävää tekniikkaa. Ominaisuudet liitetään takaisin *default*-kehityshaaraan, kun ne tulevat valmiiksi. *Default*-kehityshaara on tarkoitettu kehittäjien käyttöön ja sieltä pitäisi aina pystyä kääntämään uusi versio testejä varten. Testien ja niiden esille tuomien korjaustoimenpiteiden suorittamisen jälkeen muutokset viedään *stable*-kehityshaaraan. Jenkins CI -koontiympäristö kuuntelee *stable*-kehityshaaran muutoksia, jotka laukaisevat AnyReaderin koonti- ja konfigurointiprosessin koontipalvelimilla. Kehityshaarojen avulla saadaan säilytettyä AnyReader-koonnin kontrolli kehittäjillä, mutta vähennettyä turhia koonteja koontipalvelimella.

Anygraafissa otettiin Mercurial-versionhallinta käyttöön ensimmäisenä AnyReader-projektissa. Sen tuomat edut verrattuna aiemmin käytössä olleeseen Subversion-versionhallintajärjestelmään koettiin niin suureksi edistykseksi, että Mercurial otettiin käyttöön myöhemmin keväällä 2013 kaikissa Anygraafin kehitysprojekteissa.

2.4 Konfigurointi- ja käännösympäristö

Anygraaf käyttää jo entuudestaan projektiensa koontiin Jenkins CI -koontipalvelinta, joten oli luontevaa liittää toteutettava mobiilisovellusten koontityökalu osaksi jo olemassa olevaa järjestelmää. Mobiilisovellukset vaativat kuitenkin eri alustojen kääntämistä varten monialustaympäristön tuen, mihin Jenkins-palvelin tarjoaa kattavat työkalut. Windows-palvelimet liitettiin osaksi Jenkins CI -koontipalvelinta sen tarjoaman Slave agent- ja liitäntäsovelluksen avulla. Liitäntäsovellusta ajetaan palveluna Windows-palvelimilla. iOS-palvelimen kanssa tämä menetelmä osoittautui ainakin toistaiseksi mahdottomaksi, joten iOS-palvelimen liittäminen osaksi koontiympäristöä tehtiin SSH-yhteyden avulla. SSH-protokolla (Secure Shell) mahdollistaa turvallisen tiedonsiirron turvattomassa verkossa kahden verkkoon liitetyn tietokoneen välillä. Lisää palvelinten liittämisestä Jenkins-ympäristöön on kerrottu luvussa 2.5.

Jenkins-palvelimeen määritetyn AnyReader-koonnin komennot ajetaan kaikilla Jenkins-koontipalvelimeen liitetyillä orjapalvelimilla samanaikaisesti, jolloin palvelimen käyttöjärjestelmä ja siihen asennetut kehitysympäristöt määrittävät käännettävän AnyReader-sovelluksen alustaversion. Ensimmäiseksi ajettava komento testaa, mitkä alustaversiot sovelluksesta on mahdollista kääntää kyseisellä palvelimella. Esimerkiksi Android-

sovelluksen konfiguroinnin ensimmäinen tehtävä testaa, onko kyseiselle palvelimelle asennettu NDK-liitäntärajapinta sovellusrungon liittämiseksi osaksi Android-sovellusta. NDK-liitäntärajapinnan (Native Development Kit) avulla Java-ohjelmointikielellä toteutettuun Android-sovellukseen saadaan liitettyä C- ja C++-ohjelmointikielellä toteutettuja sovellusosia. *Test*-tehtävä tarkistaa myös, että palvelimelle on asennettu Android SDK (Software Development Kit), joka tarjoaa Android-sovelluskehitykseen tarvittavat työkalut [7]. Tehtävässä tarkistetaan myös, että palvelimelta löytyy Java-ohjelmistokehitykseen tarvittavat JDK-työkalut (Java Development Kit) [8] sekä muut Android-sovellusten kääntämiseen tarvittavat työkalut. Jos kaikki on kunnossa, konfiguraatio-tehtävä merkitsee Android-käännöksen tulevat tehtävät mahdollisiksi.

AnyReaderin konfigurointi tarkoittaa käytännössä asetusten määrittämistä paikoilleen ennen käännösprosessin alkua. Sovellus käännetään useamman kerran eri asetuksia käyttäen. Näin saadaan aikaan räätälöityjä asennuspaketteja, joista jokainen julkaistaan tiedostopalvelimelle koontiprosessin päätteeksi. Konfiguroinnissa käytettävät asetukset on tallennettu JSON-muodossa tiedostoon, joka on viety versionhallintaan. Konfigurointi on toteutettu Python-ohjelmointikielellä toteutetulla Pynt-kirjastolla. Kirjasto tarjoaa rajapinnan koontitehtävien (Task) suorittamiseksi määrätyssä järjestyksessä. AnyReaderin konfigurointi ja siihen käytetyt työkalut on esitetty tarkemmin luvussa 3. Sovelluksen konfigurointi ja kääntäminen tapahtuvat samanaikaisesti useammalla koontipalvelimella, joilta kultakin on kahdensuuntainen tiedonkulku isäntäpalvelimelle. Jenkins CI -palvelin käskyttää orjapalvelimia ja orjapalvelimet viestivät Jenkins-palvelimelle konfigurointi- ja käännösprosessin tilasta jatkuvasti.

Kuvassa 3 on konfigurointi- ja käännösympäristöön kuuluva seurantamonitori, josta voi tarkkailla käännösprosessin etenemistä. Seurantamonitorista näkee nopeasti epäonnistuneet ja epävakaat käännökset, jolloin niiden huomaaminen ei välttämättä vaadi ihmisen tekemää tuotetestausta.

Kuva 3. Anygraafin seinällä oleva seurantamonitori

Anygraafin seurantamonitoriin on liitetty koontiseurannan lisäksi kehityksessä olevien projektien sprinttien tilanteen kuvaajat sekä asiakaspalvelinten toimivuuden seuranta. Seurantamonitori on sijoitettu keskeiselle paikalle kahvikoneen läheisyyteen Anygraafin toimistolla Helsingissä.

2.5 Jenkins CI -koontipalvelin

Jenkins CI on jatkuvan integroinnin koontipalvelin, joka on toteutettu Java-ohjelmointikielellä. Jenkins voidaan asentaa lähes kaikille alustoille ja sitä voidaan laajentaa monipuolisesti tarjolla olevien yli kuudensadan moduulin avulla [9]. Anygraafin Jenkins CI -koontipalvelin on asennettu virtualisoidulle palvelimelle Ubuntu 12 -käyttöjärjestelmän päälle. Jenkins CI -koontipalvelimeen on sisäänrakennettu isäntä-orja-periaatteella toimiva ominaisuus, jonka avulla voidaan jakaa yksittäisen koonnin tehtäviä usealle orjapalvelimelle. Ominaisuuden avulla voidaan esimerkiksi vähentää yksittäisen Jenkins CI -asennuksen kuormaa, jos käännettävänä on suuri määrä projekteja [10]. AnyReader-projektin osalta ominaisuus mahdollistaa saman koonnin suo-

rittamisen useassa ympäristössä, jolloin sovelluksen alustaversiot voidaan kääntää yhdellä Jenkins CI -koontipalvelimen koontimäärityksellä.

Jenkins-palvelimeen liitetyt Windows-koontipalvelimet asennettiin virtuaalisesti samaan palvelinkaappiin Jenkins-palvelimen kanssa. Nämä orjapalvelimet on konfiguroitu kääntämään projekteja isäntäpalvelimelle Jenkins-palvelimen tarjoamien liitäntämenetelmien avulla. Jenkins tarjoaa useita tapoja orjapalvelimen liittämiseksi, mutta kaikki tavat perustuvat pohjimmiltaan kahdensuuntaiseen tietovirtaan isäntä- ja orjapalvelimen välillä [11]. Jenkins-sovelluksen Internet-sivustolla suositellaan Windows-palvelimen liittämistavaksi etähallintaa (Remote Management Facility), joka on kaikissa Windows 2000- ja sitä uudemmissa Windows-versioissa. Tämä liitäntätapa hyödyntää Microsoftin DCOM- [12] ja WMI-rajapintoja [13], jotka ovat molemmat Microsoftin toteutuksia verkon yli tapahtuvaan sovellusosien väliseen kommunikointiin. Näiden tekniikoiden avulla voidaan käskyttää ja hallita Windows-palvelinta eri skriptikieliä hyödyntäen, mutta esimerkiksi käyttöliittymän testaaminen orjapalvelimessa olisi lähes mahdoton toteuttaa. Lisäksi tämä liitäntätekniikka vaatii orjapalvelimen järjestelmänvalvojan oikeudet omaavan käyttäjän käyttäjätunnuksen ja salasanan tallentamisen isäntäpalvelimelle. Tämä aiheuttaa tietoturvariskin domainiin liitetyssä ympäristössä.

Toinen mahdollinen tapa liittää Windows-palvelin orjapalvelimeksi Jenkins-koontiympäristöön on käynnistää Jenkins-palvelimen liitäntäsovellus orjapalvelimella Java Web Start -liitännäisen avulla. Java Web Start -liitännäinen hyödyntää JNLP-protokollaa (Java Network Launch Protocol), jonka avulla Java-sovelluksia voidaan käynnistää Internet-selaimen kautta asiakkaan laitteessa. Käytännössä liittäminen tapahtuu navigoimalla orjapalvelimen selaimella Jenkins-palvelimen hallintasivuston slave-osioon. Sivulla on Java Web Start -painike, josta painamalla liitäntäsovellus käynnistyy orjapalvelimella. Tämä tapa mahdollistaa esimerkiksi eri verkoissa sijaitsevien palvelinten välisen kommunikoinnin, vaikka niiden välissä olisi palomuuri ja normaalisti suora kommunikointi ei olisi mahdollista. Lisäksi tällä liitäntätekniikalla käyttöliittymään liittyvien testien toteuttaminen orjapalvelimella on mahdollista. Huonona puolena liitäntämenetelmässä on se, että orjapalvelimen uudelleenkäynnistyksen yhteydessä Jenkins CI -koontipalvelimella ei ole mitään mahdollisuutta muodostaa yhteyttä orjapalvelimeen uudelleen. Lisäksi liitäntäsovellus voidaan käynnistää selaimen kautta ainoastaan yhden kerran.

Windows-palvelin voidaan liittää Jenkins CI -palvelimen orjapalvelimeksi myös ajamalla Jenkins-liitäntäsovellusta palveluna orjapalvelimella. Tällä liitäntämenetelmällä Jenkins-palvelu voidaan käynnistää automaattisesti orjapalvelimen käynnistymisen yhteydessä. Palvelu käy käynnistyessään ilmoittautumassa isäntäpalvelimelle, jolloin muodostetaan kahdensuuntainen tietovirta palvelinten välille. Windows-orjapalvelinten liitäntämenetelmäksi valittiin tämä menetelmä siksi, että se toimii täysin automaattisesti ja mahdollistaa myöhemmin myös käyttöliittymätestien liittämisen osaksi koontiympäristöä.

Apple-palvelimen liittäminen isäntäpalvelimeen Jenkins-palvelimen tarjoaman Java-liitäntäsovelluksen avulla ei onnistunut, joten jäljelle jäi kaksi liitäntätapaa. Ensimmäinen on kirjoittaa oma liitäntäsovellus, johon löytyy ohjeistus Jenkins-sovellusten Internet-sivuilta [14]. Tämän liitäntämenetelmän vaatima työmäärä verrattuna saavutettavaan hyötyyn on kuitenkin melko suuri. Ainoa konkreettisesti saavutettava hyöty olisi AnyReader-koontiin myöhemmin lisättävän käyttöliittymätestauksen iOS-alustan automatisoinnin helpottuminen.

Toinen tapa liittää Apple-orjapalvelin on hyödyntää Jenkins-palvelimeen sisäänrakennettua SSH-toteutusta. Jenkins-palvelimen ja orjapalvelimen välisen SSH-yhteyden autentikointiin voidaan käyttää joko käyttäjätunnusta ja salasanaa tai käyttää orjapalvelimella generoituja SSH-avaimia. Tietoturvallisuuteen liittyvistä syistä jälkimmäinen tapa oli tässä tapauksessa varmempi, koska isäntäpalvelimen ei tarvitse tuntea orjapalvelimen käyttäjätunnuksia. Apple-palvelimella generoitiin julkinen ja yksityinen avain sekä niiden yhdistämisestä salausalgoritmilla laskettu salausavain. Julkinen avain sekä salausavain siirrettiin isäntäpalvelimelle, joka lähettää ne yhteyttä muodostettaessa orjapalvelimelle. Orjapalvelin testaa vastaako isäntäpalvelimen lähettämän avaimen ja orjapalvelimella olevan yksityisen avaimen summa isäntäpalvelimen lähettämää salausavainta. Tämän liitäntämenetelmän avulla Jenkins CI -koontipalvelin voi käskyttää orjapalvelinta skriptikielien ja komentoriviltä ajettavien komentojen avulla. Jenkins-palvelin saa orjapalvelimella suoritettavista komennoista vastaukseksi komennon tulosteen.

Jenkins CI -koontipalvelimeen liitettyihin orjapalvelimiin voidaan määrittää etikettejä (Label), joiden avulla voidaan kuvata esimerkiksi orjapalvelimen tarjoamia palveluita. Anygraafin Jenkins CI -asennuksessa hyödynnetään etikettien nimeämiskäytäntöä, jossa listataan kullakin palvelimella käännettävien alustojen nimet.

Taulukko 1. Anygraafin koontiympäristön palvelinten nimet, käyttöjärjestelmät ja etiketit.

Palvelimen nimi	**Käyttöjärjestelmä**	**Etiketit**
agdevlinux	Ubuntu 12	linux, android
agmacbuilder	osx	osx, ios
agdevwin7	MS Windows 7	windows7, symbian, qt
agdevwin8	MS Windows 8	windows8

Jenkins-palvelimen AnyReader-koontimäärityksessä valitaan, millä kaikilla etiketeillä kyseinen koonti suoritetaan. Konfiguraation jokainen käsky suoritetaan eri ympäristöissä samanaikaisesti juuri sellaisenaan ympäristöstä riippumatta.

Kaikki AnyReaderin alustaversiot voidaan kääntää komentoriviltä annettujen komentojen avulla, mutta alustakohtaiset komennot ja tarvittavat operaatiot poikkeavat toisistaan. Tavoitteena on kuitenkin saada aikaiseksi onnistunut käännös AnyReaderista kullekin alustalle, joten Jenkins-palvelimen ja orjapalvelimen väliin tarvittiin sovittimeksi yhtenäinen käännösrajapinta. Lisäksi ennen kustomoitujen alustakohtaisten versioden kääntämistä pitää siirrellä sovelluspakettien allekirjoittamiseen tarvittavia avaimia, sovellukseen käytettäviä ikoneita ja asettaa käännökseen vaikuttavat parametrit paikoilleen. Suoraan Jenkins-koontiin tehtynä tällainen määritys olisi ollut työläs, eikä samaa määritystä olisi voitu hyödyntää kehittäjien työasemilta testisovelluksia käännettäessä. Tämä vahvisti tarpeen alustoille yhteisen konfigurointi- ja käännösrajapinnan käyttöönotolle.

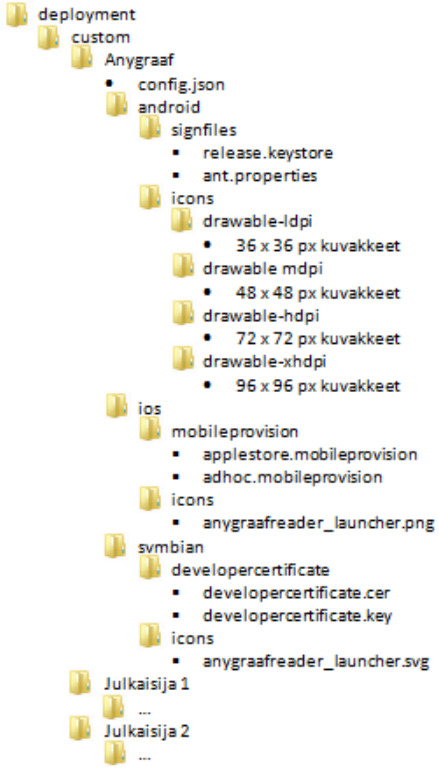
3 Mobiilisovelluksen konfigurointi

Anygraafin mobiilisovelluksen AnyReaderin konfiguroinnilla tarkoitetaan käännökseen vaikuttavien asetusten muuttamista ennen varsinaisten sovellusten kääntämistä. Asetuksissa määritetään sovelluksessa käytettävät ikonit, julkaisijan verkkokaupan tiedot, julkaisupalvelinten osoitteet sekä tuotettavan sovelluksen nimi. Konfigurointiin tarvittavat asetukset on tallennettu konfiguraatiotiedostoihin ja ne asetetaan paikoilleen konfigurointityökalun avulla räätälöidyn AnyReader-sovelluksen käännösprosessin alussa.

3.1 Konfiguraatiotiedosto

Kullekin tuotettavalle räätälöidylle sovellukselle on tehty omat konfigurointitiedostot, jotka luetaan muistiin käännösprosessin alussa. Käännösprosessi suoritetaan kerran kutakin konfiguraatiota käyttäen, jolloin saadaan aikaan räätälöityjä versioita AnyReaderista. Liitteessä 1 on AnyReaderin testisovelluksen AnygraafReaderin konfiguraatiotiedosto. Tiedostossa on määritetty alustoille yhteiset asetukset, joita ovat sovelluksen nimi, verkkokauppojen url-osoitteet ja tiedot palvelimista, joilta julkaisut ladataan sovellukseen. Laitekohtaisissa asetuksissa määritetään tuotettavan asennustiedoston allekirjoittamiseen tarvittavat tiedot, tarvittaessa lisenssiteksti ja alustakohtaiset kuvakkeet.

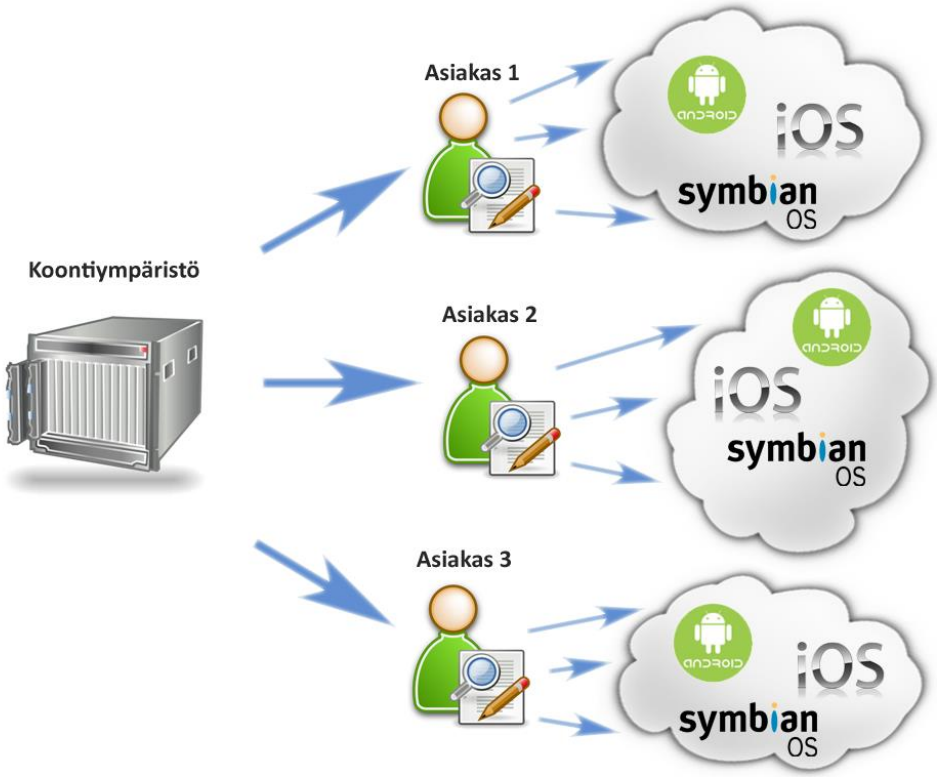
Kuvassa 4 on esitetty AnyReaderin konfiguraatiotiedostot sisältävän hakemiston rakenne. Hakemisto sisältää kullekin julkaisijalle oman konfiguraatiohakemiston, jonka sisältä löytyvät alustakohtaiset sovelluspakettien allekirjoittamiseen tarvittavat tiedostot ja räätälöidyssä sovelluksessa käytettävät kuvakkeet.



Kuva 4. AnyReaderin konfiguraatiohakemiston rakenne

Kussakin julkaisijan hakemistossa olevat config.json-tiedostot vastaavat liitteen 1 testisovelluksen konfigurointitiedostoa. Konfiguraatiohakemisto on AnyReaderin repositoryssä versionhallinnassa. Näin sen sisältämiä tiedostoja päästään muuttamaan helposti. Räätälöityjä sovelluksia voidaan kääntää koontiympäristön lisäksi myös kehittäjien työasemilta, jolloin räätälöityjen sovellusten testaaminen sovelluksen kehitysvaiheessa on nopeampaa.

Kuvassa 5 on esitetty miten koontiympäristö hyödyntää asiakaskohtaisia konfiguraatiotiedostoja. Koontiympäristö julkaisee sovelluksen kaikki alustaversiot asiakaskohtaisissa konfiguraatioissa määritettyjä asetuksia käyttäen.



Kuva 5. AnyReaderin asiakaskohtainen konfigurointi

AnyReader-sovelluksen koonnissa alustakohtaiset käännösprosessit suoritetaan useamman kerran peräkkäin. Jokainen käännös tuottaa asiakkaalle räätälöidyn sovelluksen. Käännökseen vaikuttavat asetukset on määritetty asiakaskohtaisessa konfiguraatiotiedostossa. Yksittäisessä koonnissa käännettävien sovellusten määrä riippuu asiakaskohtaisten konfiguraatioiden määrästä.

3.2 Pynt-konfigurointityökalu

Koontityökalun käyttöönoton suurin syy oli tarve yhtenäistää alustakohtaisten kääntöprosessien käskyttämiseen käytettävät komennot. Koontikirjaston avulla koonnin määrittäminen ja hajauttaminen koontiympäristöön liitetyille palvelimille sekä sen ylläpito Jenkins CI -ympäristössä ovat huomattavasti helpompia toteuttaa verrattuna tilanteeseen, jossa kaikki tehtävät määritettäisiin suoraan Jenkins-palvelimen koontiin.

Pynt-koontityökalun toiminta eroaa Ant- ja Make-koontiskriptien toiminnasta, koska Pynt-kirjastolla voidaan suorittaa ulkoisten komentojen lisäksi myös Python-ohjelmointikielellä toteutettavia toimintoja. Python-ohjelmointikielen edut antavat konfigurointikirjastolle lisäarvoa muihin verrattuna. Esimerkiksi Python-ohjelmointikieleen sisäänrakennetut teksti- ja XML-tiedostojen muokkaamiseen käytettävät työkalut toimivat hyvin työkaluina Androidin Ant-konfiguraatiotiedostoja muokattaessa. Pynt-kirjaston avulla määritellään komentoriviltä suoritettavia tehtäviä vastaavalla tavalla, kun Ant- tai Make-koontikirjastoihinkin. Tehtävät voidaan määrittää myös riippuvaiseksi toisistaan.

AnyReaderin koontiin käytettävää Pynt-koontityökalua on laajennettu niin, että tehtävä voidaan asettaa joko yleiseksi tai alustakohtaiseksi. Tämän laajennuksen avulla on saatu siirrettyä vastuu käännettävien sovellusten alustaversioiden valinnasta käyttäjältä käännöksen suoritusalustalle.

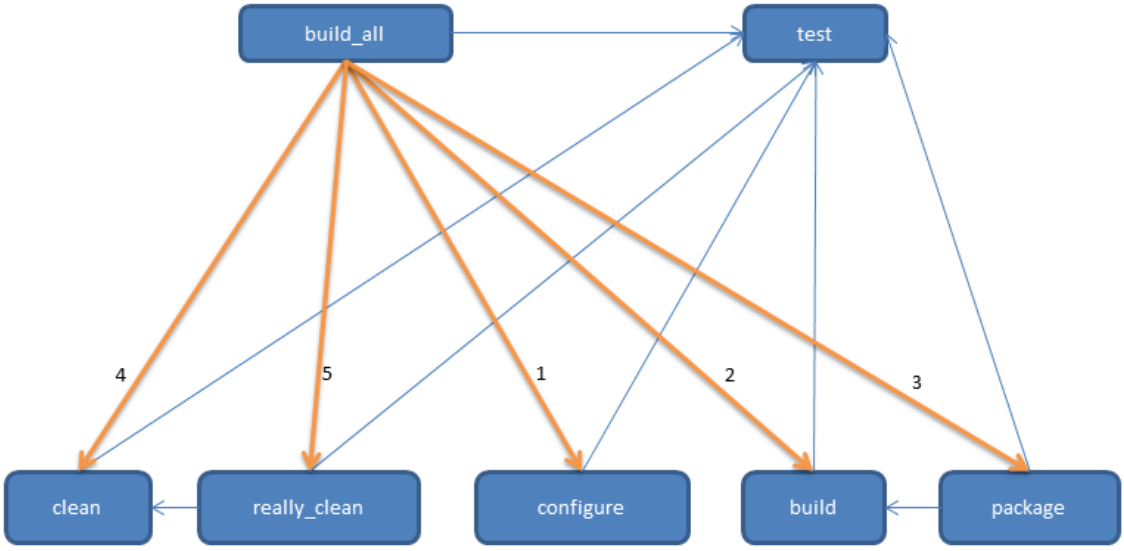
Liitteessä 2 on AnyReaderin Android-alustan Pynt-koontiskripti. Koontiskriptin tehtävät määritetään *@task()*- ja *@platform_task(test)*-annotaatioilla, joille voidaan antaa parametrinä tehtävän riippuvuudet. Näin määritetyistä tehtävistä saadaan rakenteeltaan puumainen tehtävälista. Alustakohtaiset tehtävät suoritetaan yleistä tehtävää kutsuttaessa riippuen alustakohtaisten *test*-tehtävien lopputuloksesta. Pynt-koontikirjastoon määritetyt alustoille yhteiset tehtävät, tehtävien riippuvuudet ja niiden käyttötarkoitukset on esitetty taulukossa 2.

Taulukko 2. AnyReaderin koonnin yhteiset tehtävät

Tehtävä	**Riippuvuudet**	**Kuvaus**
configure	test, common.scm_info	Lukee asetukset määrätystä konfiguraatiotiedostosta ja suorittaa alustakohtaiset konfiguraatiot. Parametrinä JSON-konfiguraatiotiedoston sisältö.
build	test	Suorittaa alustakohtaiset käännökset.
clean	test	Nollaa asetetut alustakohtaiset konfiguraatiot.
really-clean	test, clean	Poistaa käännetyt kirjastot ja sovelluspaketit sekä suorittaa *clean*-tehtävät.
build_all	test	Suorittaa peräkkäin *configure*-, *build*- ja *clean*-tehtävät kullakin konfiguraatiohakemistosta löytyvällä konfiguraatiolla.
package	test, build	Allekirjoittaa tuotetun sovelluspaketin ja siirtää sen määritettyyn output-hakemistoon.

AnyReaderin Pynt-koontiskriptin laajennuksen avulla voidaan kääntää kaikkien alustojen sovellusversiot samoja komentoja käyttäen. Koontiskriptin suoritusalusta määrää, mitkä alustaversiot sovelluksesta todellisuudessa käännetään. Ennen kunkin tehtävän suoritusta ajetaan alustakohtaiset *test*-tehtävät, jotka merkitsevät muut alustakohtaiset tehtävät joko mahdollisiksi tai mahdottomiksi.

Perinteinen AnyReaderin versio saadaan aikaiseksi ilman erillistä konfigurointia pelkällä *build*-komennolla. Onnistuneen käännöksen jälkeen tai sen yhteydessä tuotettu asennuspaketti allekirjoitetaan ja siirretään määritettyyn output-hakemistoon *package*-komennolla. *Build_all*-komento lukee kaikki konfiguraatiotiedostot taulukkoon ja suorittaa kullakin konfiguraatiolla *configure*-, *build*-, *package*- ja *clean*-komennot. Prosessin lopuksi suoritetaan *really_clean*-tehtävä, joka nollaa käytettävän kopion repositorystä vastaamaan tilannetta ennen käännösprosessin aloittamista. Kuvassa 6 on esitetty AnyReader-konfiguroinnin yhteiset tehtävät ja niiden riippuvuudet.



Kuva 6. Pynt-koontiskriptissä määritetyt yhteiset tehtävät ja niiden riippuvuudet.

Kuvan siniset nuolet tarkoittavat yhteisten tehtävien riippuvuuksia. *Build_all*-tehtävästä kutsutaan oranssien nuolien osoittamia tehtäviä määrätyssä järjestyksessä. Jenkins-palvelimelle määritetty koonti hyödyntää *build_all*-tehtävää kääntäessään tuotantoon meneviä räätälöityjä sovelluspaketteja. Luvussa 2 esiteltiin AnyReaderin koontiympäristö, johon on liitetty useita koontipalvelimia. *Build_all*-tehtävä suoritetaan Jenkins-palvelimen AnyReader-koonnissa samanaikaisesti usealla orjapalvelimella, jolloin sovelluksen alustaversiot saadaan tuotettua hajautetusti ja orjapalvelin määrää käännettävän sovelluksen alustaversion.

Yksittäisen julkaisijan yksittäinen sovellus saadaan käännetyksi kehittäjien työasemilla antamalla ensin *configure*-tehtävälle polku konfiguraatiotiedostoon parametriksi ja kutsumalla onnistuneen konfigurointitehtävän jälkeen *build*- ja *package*-tehtäviä.

Kehittäjien työasemilta voidaan kääntää sovellusversioita samoilla komennoilla, mutta näin käännettyjä sovelluksia ei voida kuitenkaan julkaista AnyReaderiin versionhallinnan revisionumeroihin perustuvan versionumeroinnin takia. Kuten luvussa 2 todettiin, hajautetun versionhallintajärjestelmän revisionumerot poikkeavat toisistaan repositoryn kopiosta riippuen ja jos työasemilta käännettyjä sovelluksia julkaistaisiin, olisi mahdotonta jäljittää, mitkä muutokset ovat mukana kyseisessä versiossa.

3.3 Alustakohtainen konfigurointi

AnyReaderin alustakohtaiset konfiguroinnit poikkeavat hieman toisistaan, vaikka määritettyjen alustakohtaisten tehtävien tavoitteet ovat samat. Tarkastellaan konfigurointia liitteessä 2 olevan AnyReaderin Android-version Pynt-koontiskriptin avulla. Skriptistä löytyvät Android-koonnin alustakohtaiset tehtävät sekä niiden käyttämät apufunktiot.

Alustakohtaiset tehtävät suoritetaan, mikäli tiedostossa määritetty *test*-tehtävä merkitsee alustakohtaiset tehtävät mahdolliseksi. Alustakohtaisia *test*-tehtäviä ei kutsuta erikseen, vaan ne suoritetaan automaattisesti, sillä ne on merkitty riippuvuudeksi kaikille muille alustakohtaisille tehtäville. *Test*-tehtävä tietää oman tilansa koko käännösprosessin ajan, jolloin sitä ei tarvitse suorittaa useasti jokaisen tehtävän alussa.

Android-alustan koontiin käytettävän Pynt-skriptin ensimmäinen tarkasteltava tehtävä on *configure*. Tehtävän tarkoitus on kopioida tarvittavat kuvakkeet konfigurointihakemistosta Androidin resurssihakemistoon. Lisäksi tehtävässä muutetaan sovelluksen rakennetta niin, että Ant-koontiskriptissä esitellyn Java-pakkauksen Activity-luokkien [15] määritykset vastaavat muutettua Java-lähdekoodihakemiston rakennetta. Androidin Java-pakkauksia joudutaan muokkaamaan esimerkiksi tilanteessa, jossa julkaisija haluaa tuottaa oman AnyReader-version jokaista tuottamaansa julkaisua varten. Kustomoituja sovelluksia täytyy tässä tapauksessa julkaista alustakohtaiseen kauppaan useampia saman julkaisijan tunnuksilla. Android-sovellusten jakeluun tarkoitettu Google play -sovelluskauppa ei hyväksy kahta samaan Java-pakkaukseen viittaavaa sovellusta yhdeltä julkaisijalta. Pynt-koontiskriptin *configure*-tehtävä luo määrätyn Java-pakkauksen [16] sisään uuden Activity-luokan, joka paketoi sisäänsä todellisuudessa kutsuttavat sovelluksen alkuperäisen Activity-luokan operaatiot. Lisäksi Androidin Ant-koontiskriptin Activity-luokan määritys muutetaan viittaamaan uuteen Activity-luokkaan. *Configure*-tehtävä kysyy myös versionhallinnasta revisionumeron ja asettaa sen AnyReaderin versionumeroksi.

Tässä kohdassa Symbian-sovelluksen konfiguroiminen eroaa hieman muista alustoista, sillä Symbian-käyttöjärjestelmän versionumeroinnin pisteellä erotetut numerot eivät saa ylittää lukua 255 [17]. Tämä ongelma on ratkaistu Symbian-alustan Pynt-skriptissä laskemalla Symbian-sovelluksen eniten merkitsevä versionumero jakamalla revision numero luvulla 255. Symbian-sovellusten vähiten merkitsevä numero merkitään revisionumerosta vastaavalla luvulla laskettavan jakojäännöksen perusteella. Esimerkiksi

revisiosta numero 1690 luodun AnyReader-version Android- ja iOS-alustan versionumeroiden ollessa 1.1690 Symbian-sovelluksen versionumero on 1.6.160. Versionumeroiden alussa oleva numero merkitsee AnyReaderin todellista sovellusversiota, joka on esimerkkitapauksissa 1.

Alustakohtaiset *build*-tehtävät kääntävät sovellukset käyttäen edellä esiteltyjä konfiguraatioita. *Build*-tehtävissä kutsutaan aliprosessina alustakohtaisia käännöskomentoja. Android-alustan *build*-tehtävä kutsuu Ant-koontiskriptin build-operaatiota. Tämä on toteutettu kuvassa 7 esitellyssä Python-ohjelmointikielellä toteutetussa build-tehtävässä.

```
@platform_task(test)
def build(custom, debug=False):
    '''Builds android application'''
    cmd = [ndk(), '-j%d' % multiprocessing.cpu_count()]
    if debug: cmd.append('APP_OPTIM=debug')
    else: cmd.append('STORE=true')
    if custom != None:
        defines = 'DEFINES='
        for define in utils.get_defines(custom):
            defines = defines + ' -D%s' % define
        cmd.append(defines)
    subprocess.check_call(cmd, cwd=android_root())
    subprocess.check_call([ant(), 'release'], cwd=android_root())
```

Kuva 7. Android-alustan Pynt-koontiskriptin build-tehtävä

Python-ohjelmointikielellä aliprosesseja voidaan suorittaa kutsulla subprocess.check_call(cmd, cwd=android_root()). Ennen Ant-koontiskriptin build-kutsun suorittamista Androidin *build*-tehtävässä käännetään C++-ohjelmointikielellä toteutettu sovellusrunko erillisenä aliprosessina. Sovellusrungon kääntämiseen käytetään NDK-rajapinnan tarjoamaa ndk-build-komentoa. NDK kääntää C++-tiedostot kirjastoiksi, jotka liitetään osaksi tuotettavaa Android-sovellusta sovelluksen linkkaamisvaiheessa. Symbian- ja Windows-alustojen build-tehtävässä kutsutaan GNU Make -koontiskriptin make-operaatiota. Applen iOS-sovellusversiota varten määritetyssä Pynt-koontikirjaston *build*-tehtävässä kutsutaan xcode-kääntäjän xcodebuild-operaatiota. Alustakohtaisen *build*-tehtävän lopputuloksena saadaan valmis AnyReaderin asennuspaketti.

Android-koontiin käytettävä Ant-koontiskripti allekirjoittaa (sign) asennuspaketin automaattisesti käännöksen jälkeen, mikäli allekirjoittamiseen vaadittavat avaimet on kopioitu onnistuneesti oikeaan hakemistoon ennen käännöksen suorittamista. Tehtävän

päätteeksi Androidin asennuspaketti on julkaisuvalmis, kun taas Symbian- ja iOS-asennuspaketit tarvitsevat vielä allekirjoittamisen. Android-sovelluksen asennuspaketin allekirjoittaminen erillisenä operaationa todettiin hankalaksi ja sitä varten koontipalvelimille olisi jouduttu asentamaan erillisiä työkaluja. Muiden alustojen asennuspaketit pitää allekirjoittaa erillisenä operaationa. Tämän takia *build*-tehtävän päätteeksi alustakohtaiset lopputulokset eroavat toisistaan. Tätä ei kuitenkaan koettu ongelmaksi, sillä kaikki asennuspaketit joudutaan kuitenkin nimeämään oikein ja siirtämään output-hakemistoon odottamaan julkaisua. Symbian- ja iOS-asennuspakettien allekirjoitus päätettiin tehdä saman tehtävän yhteydessä.

Package-tehtävän tarkoitus on paketoida ja allekirjoittaa tuotetut asennuspaketit sekä siirtää ne output-hakemistoon odottamaan siirtoa tiedostopalvelimelle. AnyReaderin asennuspaketit nimetään tuotettavan sovelluksen nimisiksi ja perään liitetään käännöksen revisionumero. Asennuspakettien paketoinnilla tarkoitetaan niiden pakkaamista, jonka avulla säästetään levytilaa ja optimoidaan pääsääntöisesti langattomasta mobiiliverkosta ladattavan sovelluksen latausaika.

Alustakohtaisia puhdistustehtäviä on kaksi. *Clean*-tehtävä on tarkoitettu suoritettavaksi kustomoitujen sovellusten koontien välissä ja *really_clean*-tehtävä koko prosessin päätteeksi. Ensimmäinen palauttaa *configure*-tehtävässä asetetut asetukset ennalleen sekä poistaa konfiguroinnista riippuvaisten kirjastojen käännökset. *Really_clean*-tehtävä palauttaa koko repositoryn samaan tilaan, jossa se oli ennen koontiprosessin alkamista. Androidin puhdistustehtävien määritykset löytyvät liitteen lopusta.

3.4 Alustakohtaiset rajoitteet ja kohdatut haasteet

Alun perin Android-sovelluksen lähdekoodeja ei ollut tarkoitus muokata koonnin yhteydessä, mutta ensimmäisen räätälöidyn testisovelluksen IAP-ominaisuutta testattaessa huomattiin, ettei samoilla Google Play -tunnuksilla voi julkaista samaan Androidin Activity-luokan sisältävään Java-pakkaukseen viittaavaa sovellusta. Ongelman ratkaisuksi toteutettu eri pakkaukseen viittaava rajapinta, joka kutsuu sisäisesti AnyReaderActivity-luokan operaatioita, osoittautui toimivaksi. Androidin osalta päädyttiin asettamaan kaikki tuotantoon menevät räätälöidyt sovellukset omiin pakkauksiinsa.

AnyReaderin koonnissa tehdyt *clean*-operaatiot tekevät repositoryssä revert-operaation, jonka on tarkoitus kumota sitouttamattomat tiedostoihin tehdyt muutokset. Versionhallinnan revert-operaatiot jättivät kuitenkin varmuuskopioita muutetuista Android-sovelluksessa käytettävien tekstien määrityksistä. XML-tiedostoja käytetään esimerkiksi Android-sovelluksessa käytettävien tekstien määrityksissä. Ant-koontiskripti menee sekaisin, jos XML-tiedoston rinnalla on samanniminen bak-päätteinen tiedosto. Pitkän tutkimisen jälkeen vaikuttaa siltä, että Ant-koontiskripti käyttää välillä oikeaa tiedostoa ja välillä tiedoston varmuuskopiota. Jos käännöksen yhteydessä on viitattu väärään määritystiedostoon, sovelluksen koonti saattaa vaikuttaa onnistuneelta, mutta todellisuudessa tuotettu sovellus saattaa sisältää vääriä tekstimäärityksiä tai pahimmassa tapauksessa ei käynnisty ollenkaan. Tällainen ongelma on erittäin hankala havaita. Ongelma ratkaistiin suorittamalla versionhallintajärjestelmän revert-operaatiot pakottamalla varmuuskopioiden luonti pois käytöstä. Ongelma havaittiin jo testeissä hyvissä ajoin ja sen suurempaa vahinkoa ei ehtinyt tapahtua. Koontiskriptissä tehdyt versionhallinnan revert-operaatiot ovat vaarallisia myös siinä mielessä, että työasemalta suoritettuna ne saattavat tuhota kehittäjän lähdekoodeihin tekemiä muutoksia. Tämän takia Pynt-koontiskriptissä määritetyt revert-operaatiot suoritetaan aina mahdollisimman tarkasti kohdistettuna joko yksittäiseen tiedostoon tai koonnin yhteydessä luotuun hakemistoon.

Anygraafin muiden projektien koonnit suoritetaan pääasiassa Windows 8 -käyttöjärjestelmällä varustetulla palvelimella Visual Studio 2012 -kehitysympäristössä. AnyReaderin QT-sovelluskehyksellä toteutettu Symbian-käännös oli tarkoitus määrittää suoritettavaksi tässä ympäristössä. Symbian-sovelluksen kääntämiseen käytettävät QT-kirjastot eivät kuitenkaan ainakaan vielä ole yhteensopivia uusimman Microsoftin ympäristön kanssa. Ongelma ratkaistiin väliaikaisesti liittämällä koontiympäristöön vanhempaa Visual Studio 2008 -kehitysympäristöä käyttävä virtuaalipalvelin. Tulevaisuudessa Symbian- ja Windows-käännöksiä tullaan muuttamaan niin, että ne käännetään samalla palvelimella muiden sovellusten kanssa.

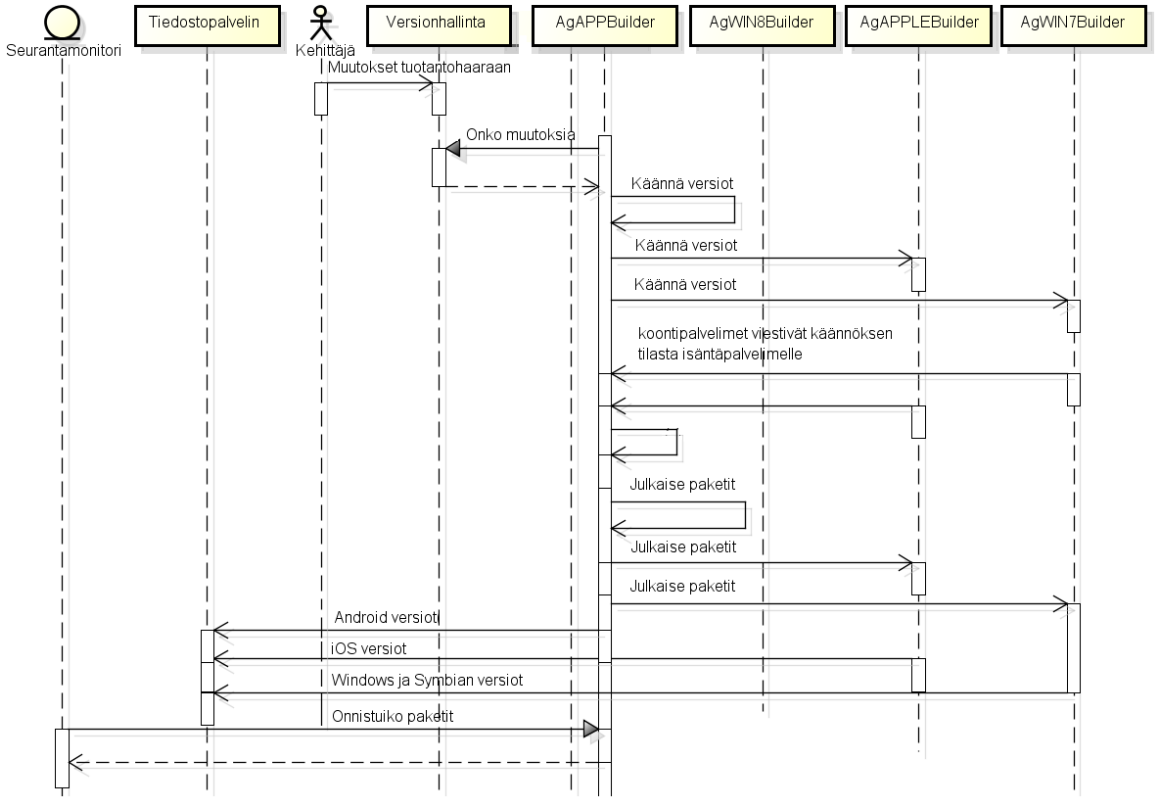
Apple rajoittaa omien sovellustensa, mukaan lukien kehitysympäristöjensä asentamisen ainoastaan omiin fyysisiin laitteisiinsa. Siksi esimerkiksi järjestelmään liitetty Apple-palvelin on oma fyysinen rautansa. Tämä rajoitus oli otettu huomioon jo ennen koontiympäristön toteuttamista, joten se ei aiheuttanut suurempaa päänvaivaa.

4 Mobiilisovelluksen automatisoitu koonti

AnyReaderin koontiprosessi haluttiin automatisoida, koska se vaati samanaikaisesti paljon usean kehittäjän työaikaa. Koontiprosessin kontrollointi haluttiin kuitenkin säilyttää yhä kehittäjillä. Siksi päädyttiin ratkaisuun, jossa koontipalvelin kuuntelee ainoastaan sovelluksen julkaisuun tarkoitetun versionhallinnan kehityshaaran muutoksia. AnyReaderin kehityssprintin päätteeksi kehittäjät vievät uudet lähdekoodit kehityshaarasta julkaisuhaaraan, jolloin koontiprosessi käynnistyy koontipalvelimella.

4.1 Automatisoidun koonnin kulku

Koontiprosessi käynnistyy, kun kehittäjä vie kehityshaarassa tehdyt muutokset stable-haaraan. Kuvassa 8 on esitetty AnyReaderin koontiprosessin eteneminen sekvenssikaaviona.



Kuva 8. AnyReaderin koontiympäristön tehtävät suoritusjärjestyksessä

Isäntäpalvelin käy AnyReaderin Jenkins CI -koonnissa määritetyn väliajoin kysymässä versionhallinnasta mahdollisia muutoksia. Isäntäpalvelimella ajettava Jenkins CI käskyttää käännökset käyntiin orjapalvelimille, kun stable-haarassa havaitaan uusia muutoksia. Kaikki kehitysympäristöön liitetyt koontipalvelimet käyvät lataamassa versionhallintapalvelimelta uusimman version lähdekoodeista, jonka jälkeen Jenkins käskee orjapalvelimia suorittamaan *build_all*-tehtävän.

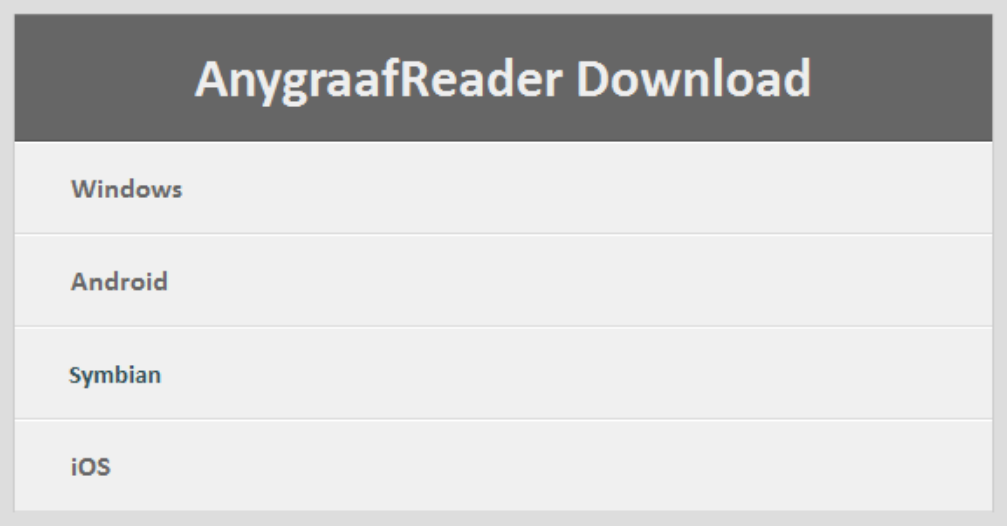
Jenkins CI -palvelin saa jatkuvasti paluuviestinä tietoa käännösten tilasta ja koontiympäristöön liitetty seurantamonitori käy kyselemässä Jenkins-palvelimelta koonnin tilannetta. Seurantamonitorista havaitaan nopeasti mahdolliset virheet, jonka ansiosta virheisiin pystytään reagoimaan nopeasti. Jenkins CI -koontipalvelin käskee orjapalvelimia julkaisemaan kaikki output-hakemistoihin syntyneet asennuspaketit tiedostopalvelimelle, kun kaikki *build_all*-tehtävässä suoritetut tehtävät on saatu valmiiksi. Tiedostopalvelin ja sen toiminta on esitelty luvussa 4.2.

Jenkins CI -koontipalvelimelle on määritetty CIFS-protokollaa [18] käyttävä yhteys verkkotiedostojärjestelmään. Valmiit AnyReaderin asennuspaketit siirretään protokollan avulla tiedostopalvelimelle jaettavaksi. CIFS-protokolla on Microsoftin kehittämä verkkotiedostojärjestelmä-protokolla, joka osaa hyödyntää kehittyneitä verkkotekniikoita kuten tiedostojen lukitsemista. Lukitsemisella varmistetaan, ettei sovelluspaketteja käytetä tiedostopalvelimelta ennen kuin ne ovat siirtyneet kokonaisuudessaan. Isäntäpalvelin käskee orjapalvelimia julkaisemaan asennuspaketit verkkotiedostojärjestelmään, kun kaikki kustomoidut sovellusversiot on käännetty. Koontiprosessin päätteeksi isäntäpalvelin käskee *really_clean*-tehtävät suoritettavaksi koontipalvelimilla. *Really_clean*-tehtävä palauttaa koontipalvelinten kopiot AnyReaderin repositorystä samaan tilaan, jossa ne olivat ennen koontiprosessin alkua. Koonnin jälkeen Jenkins-koontipalvelin alkaa taas kuunnella mahdollisia muutoksia stable-kehityshaarasta ja prosessi alkaa alusta.

4.2 Tiedostopalvelin ja asennuspakettien jakaminen

AnyReaderin kehitysympäristön tueksi on liitetty Ubuntu 12 -käyttöjärjestelmällä toimiva tiedostopalvelin, jonne koontiympäristö siirtää valmiit asennuspaketit. Tiedostojaon lisäksi palvelimelle on asennettu Apache HTTP -palvelin [19]. HTTP on tiedonsiirtoprotokolla, jota käytetään esimerkiksi Internet-selainten ja verkkopalvelinten väliseen

kommunikointiin. Tiedostopalvelimelle asennetun Apache-palvelimen päälle on toteutettu Web.py-ohjelmointikehyksen avulla verkkosovellus, jonka tarkoitus on huolehtia uusimpien asennuspakettien jakelusta Internet-selaimen läpi ladattaessa. Web.py [20] on erittäin kevyt, mutta tehokas työkalu pienten verkkosovellusten toteutukselle Python-ohjelmointikielellä. Kehys tarjoaa rajapinnat kaikkien yleisimpien verkkosovelluksissa käytettävien ominaisuuksien toteutukselle. Web.py-sovelluskehyksellä toteutettu sovellus lukee palvelimella olevasta tiedostojaosta asennuspakettien tiedot ja luo niiden perusteella asennuspakettien lataussivuston. Kuvassa 9 on kuvakaappaus AnyReaderin testisovelluksen lataussivustosta. Sivuston ulkoasu on yksinkertainen, jotta sen käyttö olisi helppoa myös pieninäyttöisillä mobiililaitteilla.



Kuva 9. Kuvakaappaus AnygraafReaderin lataussivustolta.

Sivusto toimii julkaistavien asennuspakettien jakelijana sovelluksen tilanneille julkaisijoille. Lisäksi sivustoa käytetään testisovellusten jakamiseen testikäyttäjille.

5 Jatkokehitysideoita

AnyReaderin koonnin yhteydessä tehdään staattista analyysia lähdekoodeille. Analyysin lopputulos vaikuttaa seurantamonitorissa olevaan AnyReader-koonnin kuvaajan taustaväriin. Jo pelkän staattisen analyysin avulla on löytynyt muutamia selkeitä virheitä lähdekoodista.

Myös erilaisten testien liittäminen osaksi AnyReaderin koontia toisi paljon lisäarvoa automatisoidulle koontiympäristölle. Anygraafissa on alustavasti suunniteltu käyttöliittymän automatisoitua testiympäristöä, jota voisi mahdollisesti käyttää myös AnyReader-projektin koonnin yhteydessä. Etenkin sovelluksen eri osia testaavien Unittestien liittäminen osaksi koontia olisi melko yksinkertaista ja siitä saatava hyöty joissakin tapauksissa jopa korvaamatonta. Tulevaisuudessa koontiympäristöön tullaan liittämään myös jonkinlainen testiympäristö.

AnyReader-projektin koonti on automatisoitu siihen vaiheeseen, että sovellusten asennuspaketit syntyvät automaattisesti. Kehittäjät joutuvat yhä siirtämään manuaalisesti asennuspaketit laitekohtaisiin kauppoihin loppukäyttäjien ladattaviksi. Mikäli koontiympäristöön saadaan liitettyä tarpeeksi kattavat testit, niin myös sovellusten asennuspakettien siirtyminen laitekohtaisiin kauppoihin voitaisiin automatisoida. Tämä idea saattaa olla mahdoton toteuttaa laitekohtaisten kauppojen liitäntärajapintojen puutteellisuuden takia, mutta testien toteuttamisen jälkeen liitäntämahdollisuuksia kannattaa ainakin tutkia. Uusi versio AnyReader-sovelluksesta siirtyisi tämän ominaisuuden avulla täysin automaattisesti suoraan loppukäyttäjän käytettäväksi.

iOS-alustan koonti suoritetaan orjapalvelimella, joka on liitetty isäntäpalvelimeen SSH-yhteyden avulla. Varsinkin ennen käyttöliittymätestien liittämistä osaksi koontiympäristöä myös koontiympäristöön liitetty Applen fyysinen palvelin täytyy saada liitettyä isäntäpalvelimeen vastaavalla tavalla, kuin muutkin koontiympäristöön liitetyt koontipalvelimet.

6 Yhteenveto

Tässä insinöörityössä esiteltiin Anygraaf Oy:n mobiilisovelluksen AnyReaderin automatisoidun koonti- ja konfigurointiympäristön toteuttaminen. Työssä kerrottiin, minkälaista konfigurointia räätälöidyn mobiilisovelluksen automatisoitu koonti vaatii sekä mitä vaatimuksia monialustatuki asetti toteutetulle konfigurointiympäristölle. Tuotettu koonti- ja konfigurointiympäristö laajensi jo olemassa olevaa koontiympäristöä, joka huolehtii Anygraafin muiden projektien koonnista.

Automatisoidut koontijärjestelmät ovat Anygraafille ja minulle itsellenikin vielä melko uusi tuttavuus. Monialustasovelluksen hajautetun koontiympäristön toteuttaminen on

varmasti monipuolisuudeltaan ja laajuudeltaan suurimpia esimerkkejä koontiympäristöistä, joita sovelluskehitysympäristöön tänäpäivänä voidaan liittää. Projektin eri vaiheissa eniten yllättivät alustatoteutusten väliset erot sekä niiden koontiympäristön toteutukselle aiheuttamat haasteet. Jenkins CI -koontiympäristö vaikuttaa todella onnistuneelta valinnalta Anygraafin koontiympäristön toteutusalustaksi. Monet muut yhteisöt ovat päätyneet samaan ratkaisuun, mistä kertoo tiuhaan tahtiin ilmestyvät päivitykset kyseiseen järjestelmään.

Toteutettu koontiympäristö on ollut tähän asti lähinnä testikäytössä, mutta ensimmäiset räätälöidyt asiakassovellukset tullaan julkaisemaan sen kautta ennen kesää 2013. Koontiympäristön koko potentiaalista hyötyä Anygraafille on tässä vaiheessa hieman hankala arvioida, sillä siihen vaikuttaa etenkin AnyReader-tuotteen menestys tulevaisuudessa.

Alkuvuodesta 2013 vietiin läpi testiprojekti yhden julkaisijan kustomoidun AnyReader-sovelluksen käyttöönotosta. Tällöin koontiympäristöä päästiin testaamaan Android-alustatoteutuksen osalta. Android-sovelluksen julkaisuun kului yhden kehittäjän työaikaa alle minuutti. Ainoa julkaisun vaatima työtehtävä oli viedä uudet muutokset versionhallinnan stable-kehityshaaraan. Tässä vaiheessa iOS- ja Symbian-alustojen koontien toteutukset eivät olleet vielä valmiit, joten kehittäjät joutuivat julkaisemaan asennuspaketit tiedostopalvelimelle käsin omilta työasemiltaan. Tuntikirjanpidon mukaan asennuspakettien valmistukseen ja julkaisuun kului työaikaa kahdelta kehittäjältä yhteensä hieman yli kolme tuntia. Jos kustomoitavia sovelluksia olisi ollut useampi, tämä aika voitaisiin kertoa räätälöitävien sovellusten määrällä. Jo tämän testiluontoisen projektin perusteella voidaan arvioida koontiympäristön olevan hyödyllinen ja tuotannollisesti jopa välttämätön työkalu. Testiprojektin ajankäyttöä arvioitaessa on toki otettava huomioon, että prosessi vietiin läpi ensimmäistä kertaa kokonaisuudessaan alusta loppuun. Seuraavalla kerralla prosessi saataisiin varmasti suoritettua nopeammin myös ihmisten toimesta.

Koontiympäristön toteuttamiseen kului yllättävän paljon työaikaa lähinnä projektin aikana esille tulleiden alustatoteutusten eroavaisuuksien takia. Esimerkiksi liitteessä 2 olevasta Android-alustan Pynt-koontiskriptistä piti alun perin tulla huomattavasti yksinkertaisempi. Onni onnettomuudessa oli onnistunut koontityökalun valinta. Python-ohjelmointikielen laajat sisäänrakennetut kirjastot tarjosivat aina poikkeuksetta ratkai-

sun ongelmiin, joiden ratkaisemiseen muiden koontikirjastojen kanssa olisi tarvittu ulkoisia sovelluksia.

Toteutettu koontiympäristö on toimiva osa AnyReaderin kehitysympäristöä ja uskon, että sen eteen nähty vaiva maksaa vielä itsensä takaisin. Potentiaalisia jatkokehitysideoita järjestelmän laajentamiseksi löytyy paljon ja niiden toteuttamisen tarve ratkeaa tulevaisuudessa koontiympäristön käyttömäärän perusteella.

Lähteet

1 Integrated Development Environment. 2013. Verkkotietojärjestelmä. <http://en.wikipedia.org/wiki/Integrated_development_environment>. Luettu 21.3.2013.

2 Apple rajoittaa OS X -lisenssiteksti. Verkkodokumentti. <http://images.apple.com/legal/sla/docs/macosx107.pdf>. Luettu 17.3.2013.

3 Boost C++ kirjasto. Verkkodokumentti. 2013. <http://www.boost.org/>. Luettu 17.2.2013.

4 Libcurl the multiprotocol file transfer library. 2013. <http://curl.haxx.se/libcurl/>. Luettu 11.3.2013.

5 JsonCpp is an implementation of a JSON reader and writer in C++. 2013. Verkkodokumentti. <http://sourceforge.net/projects/jsoncpp/>. Luettu 17.3.2013.

6 Hajautettu versionhallinta. 2011. Verkkodokumentti. <http://reaktor.fi/osaaminen/hajautettu_versionhallinta/>. Luettu 21.3.2013.

7 Android SDK (Software Development Kit). Verkkodokumentti. <http://developer.android.com/sdk/index.html>. Luettu 22.3.2013.

8 Java Development Kit. 2013. Verkkotietojärjestelmä. <http://en.wikipedia.org/wiki/JDK>. Luettu 22.3.2013.

9 Jenkins CI Plugins. 2012. Verkkodokumentti. <https://wiki.jenkins-ci.org/display/JENKINS/Before+starting+a+new+plugin>. Luettu 11.3.2013.

10 Jenkins CI Distributed builds. 2012. Verkkodokumentti. <https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds>. Luettu 11.3.2013.

11 Jenkins CI Distributed builds – How does it works. 2013. Verkkodokumentti. <https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds#Distributedbuilds-Howdoesthiswork%3F>. Luettu 11.3.2013.

12 Understanding the DCOM Wire Protocol by Analyzing Network Data Packets. 1998. Verkkodokumentti. <http://www.microsoft.com/msj/0398/dcom.aspx>. Luettu 3.3.2013.

13 Windows Management Instrumentatium. 2012. Verkkodokumentti. <http://msdn.microsoft.com/en-us/library/windows/desktop/aa394582%28v=vs.85%29.aspx>. Luettu 3.3.2013.

14 Write your own script to launch Jenkins slaves 2013. Verkkodokumentti. <https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds#Distributedbuilds-WriteyourownscripttolaunchJenkinsslaves>. Luettu 4.3.2013.

15 Android activity class reference. 2013. Verkkodokumentti. <http://developer.android.com/reference/android/app/Activity.html>. Luettu 21.3.2013.

16 Creating and using Java packages. 2013. Verkkodokumentti. <http://docs.oracle.com/javase/tutorial/java/package/packages.html>. Luettu 21.3.2013.

17 Symbian pkg version number limits. 2012. Verkkodokumentti. <http://symbiangeek.flaminghorns.com/tag/pkg-version-number-limits>. Luettu 17.3.2013.

18 CIFS –verkkotiedostojärjestelmä. 2008. Verkkotietojärjestelmä. <https://jop.cs.tut.fi/twiki/bin/view/Tietoturva/Tutkielmat/2008-33>. Luettu 17.3.2013.

19 Apache HTTP server project. 2012. Verkkodokumentti <http://httpd.apache.org/ABOUT_APACHE.html>. Luettu 17.3.2013.

20 Web.py -sovelluskehyksen dokumentaatio ja kotisivut. 2013. Verkkodokumentti. <http://webpy.org/docs/0.3/api>. Luettu 21.3.2013.

AnygraafReader-testisovelluksen konfiguraatiotiedosto


```
{
    "name": "AnygraafReader",
    "stores": {
        "prod": "http://****.****.fi/subscriptions",
        "test": "http://****.***.****.fi/store/"
    },
    "servers": {
        "prod": "http://**.**.**/edoris?app=AnyReaderweb&com=service",
        "test": "http://**.**.**/**/**.*?app=AnyReaderweb&com=service"
    },
    "ios": {
        "sign_key": "********"
    },
    "android": {
        "package": "com.anygraaf.anygraafreader",
        "eula": "LICENSE AGREEMENT AND LIMITED WARRANTY\n\n......."
    },
    "symbian": {
        "target_uid": "0x20028100"
    }
}
```

Android-sovelluksen Pynt-konfigurointiskripti

```
#!/usr/bin/python

import os, shutil
from pynt import task, platform_task
import utils, subprocess, multiprocessing
from distutils import dir_util
from platform import system
def get_os():
    return system()

def android_root():
    '''Get qt source root.'''
    return utils.get_dir('src', 'android')

def ndk():
    '''Get OS specific ndk-build command.'''
    if get_os() == 'Windows':
        return 'ndk-build.cmd'
    return 'ndk-build'

def ant():
    '''Get OS specific ant command.'''
    if get_os() == 'Windows':
        return 'ant.bat'
    return 'ant'

def get_signed_apk_file(config):
    '''Guess output file name for signed apk-file.'''
    if config != None:
        return os.path.join(android_root(), 'bin', '%s-release.apk' %
config['name'])
    return os.path.join(android_root(), 'bin', 'AnyReader-
release.apk')

_test_ok = None
def create_icons(config):
    dir_util.copy_tree(
        config.get_path('android', 'icons'),
        os.path.join(android_root(), "res")
    )

@task()
def make_android_jsapi():
    '''Generate android.js and copy html resources to andoid assets
directory.
    Also creates platform-specific css file for Android devices.'''
    assets = os.path.join(android_root(), 'assets')
    if os.path.exists(assets):
        shutil.rmtree(assets)
    shutil.copytree(utils.get_dir('res', 'html'), assets)
    utils.make_AnyReader_css("android.css", os.path.join(assets, 'Any-
Reader.css'));
    utils.make_jsapi('android.js', os.path.join(assets, 'AnyRead-
er.js'))
```

```
@platform_task(test, make_android_jsapi)
def configure(custom, debug=False):
    # Update revision to manifest file
    utils.replace_content(
        os.path.join(android_root(), "AndroidManifest.xml"),
        [
            (r'versionCode=".*"', r'versionCode="%s"' %
utils.scm_rev())
        ]
    )
    if custom != None:
        # Set custom app package (identifies app in store& phone)
        utils.replace_content(
            os.path.join(android_root(), "AndroidManifest.xml"),
            [(r'package=".*"', r'package="%s"' % cus-
tom['android']['package'])]
        )
        # Set custom app name (and eula)
        utils.replace_content(
            os.path.join(android_root(), "build.xml"),
            [(r'<project name=".*"', r'<project name="%s"' % cus-
tom['name'])]
        )
        utils.replace_content(
            os.path.join(android_root(), "res", "values",
"strings.xml"),
            [
                (r'<string name="eula">.*</string>', r'<string
name="eula">%s</string>' % custom['android']['eula']),
                (r'<string name="app_name">.*</string>', r'<string
name="app_name">%s</string>' % custom['name'])
            ]
        )
        # copy_signature_files
        dir_util.copy_tree(
            custom.get_path('android', 'signfiles'),
            android_root()
        )
        # copy custom icons
        dir_util.copy_tree(
            custom.get_path('android', 'icons'),
            os.path.join(android_root(), "res")
        )
        create_custom_activity(custom['android']['package'])
def create_custom_activity(package):
    '''Move ApplicationActivity in to correct package
    Import correct package in to activity class'''
    activity = os.path.join(android_root(), 'src', 'com', 'anygraaf',
'AnyReader', 'ApplicationActivity.java')
    new_activity = [android_root(), 'src']
    new_activity.extend(package.split('.'))
    new_activity.append('ApplicationActivity.java')
    utils.replace_content(
        activity,
        [(r'package com.anygraaf.AnyReader;', r'package %s;' % pack-
age)]
    )
    if not
os.path.exists(os.path.dirname(os.path.join(*new_activity))):
```

```
        os.makedirs(os.path.dirname(os.path.join(*new_activity)))
    shutil.move(activity, os.path.join(*new_activity))
def remove_custom_activity(package):
    '''Delete custom activity and revert it back to default loca-
tion'''
    if package:
        #Remove package folder
        new_package = package.split('.')
        for i in range(len(new_package)):
            folder = new_package[0:len(new_package) - i]
            if not folder[len(folder)-1] in ['com', 'anygraaf', 'Any-
Reader']:
                dir_to_remove = os.path.join(android_root(), 'src',
*folder)
                if os.path.isdir(dir_to_remove):
                    shutil.rmtree(dir_to_remove)


@platform_task(test)
def build(custom, debug=False):
    '''Builds android application'''
    cmd = [ndk(), '-j%d' % multiprocessing.cpu_count()]
    if debug: cmd.append('APP_OPTIM=debug')
    else: cmd.append('STORE=true')
    if custom != None:
        defines = 'DEFINES='
        for define in utils.get_defines(custom):
            defines = defines + ' -D%s' % define
        cmd.append(defines)
    subprocess.check_call(cmd, cwd=android_root())
    subprocess.check_call([ant(), 'release'], cwd=android_root())

@platform_task(test)
def package(custom):
    '''Package android application with jarsigner.'''
    if custom != None:
        shutil.move(
            os.path.join(android_root(), 'bin',
get_signed_apk_file(custom)),
            utils.get_output_path(custom['name'] + '_r' +
utils.scm_rev() + '.apk')
        )
    else:
        shutil.move(
            os.path.join(android_root(), 'bin', 'AnyReader-
release.apk'),
            utils.get_output_path('AnyReader_r' + utils.scm_rev() +
'.apk')
        )

@platform_task(test)
def clean(custom):
    subprocess.check_call([ant(), 'clean'], cwd=android_root())
    subprocess.check_call([ndk(), 'clean'], cwd=android_root())

    if custom and custom.has_key('android') and cus-
tom['android'].has_key('package'):
        remove_custom_activity(custom['android']['package'])
        #revert images and modified files
```

```
        utils.hg_revert(utils.get_root(), 'src/android/res');
        utils.hg_revert(utils.get_root(),
'src/android/AndroidManifest.xml');
        utils.hg_revert(utils.get_root(), 'src/android/build.xml');
        utils.hg_revert(utils.get_root(), 'src/android/strings.xml');
        utils.hg_revert(utils.get_root(), 'src/android/src');
        if os.path.isdir(os.path.join(android_root(), 'obj', 'local')):
                shutil.rmtree(os.path.join(android_root(), 'obj', 'lo-
cal'))
        if os.path.isdir(os.path.join(android_root(), 'libs', 'armea-
bi')):
                shutil.rmtree(os.path.join(android_root(), 'libs', 'ar-
meabi'))
        if os.path.isdir(os.path.join(android_root(), 'libs', 'armeabi-
v7a')):
                shutil.rmtree(os.path.join(android_root(), 'libs', 'ar-
meabi-v7a'))


@platform_task(test)
def reallyclean(custom):
    '''Clean AnyReader and libs'''
    dir_path = os.path.join(android_root(), 'obj', 'local')
    if os.path.isdir(dir_path):
        for filename in os.listdir(dir_path):
            filepath = os.path.join(dir_path, filename)
            try:
                shutil.rmtree(filepath)
            except OSError:
                os.remove(filepath)
    dir_path = os.path.join(android_root(), 'libs')
    if os.path.isdir(dir_path):
        for filename in os.listdir(dir_path):
            filepath = os.path.join(dir_path, filename)
            try:
                shutil.rmtree(filepath)
            except OSError:
                os.remove(filepath)
    subprocess.check_call([ant(), 'clean'], cwd=android_root())
    subprocess.check_call([ndk(), 'clean'], cwd=android_root())
```