hyväksymispäivä arvosana

arvostelija

2K-käyttöjärjestelmän reflektiiviset ominaisuudet

Sampsa Ojala

24.11.2003

Seminaaritutkielma
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Tiivistelmä

2K on reflektiivinen käyttöjärjestelmä, joka rakentuu reflektiivisen ORB:n päälle.

2K pyrkii tarjoamaan ratkaisuja ongelmiin, jotka liittyvät resurssien hallintaan heterogeenisissä tietoverkoissa, dynaamiseen sopeutuvuuteen ja komponenttipohjaisten hajautettujen sovellusten konfigurointiin.

2K:n pääominaisuuksia ovat hajautus, käyttäjäkeskeisyys, sopeutuvuus ja arkkitehtuuritietoisuus. Arkkitehtuurin perustana on sovellus- ja käyttäjäkeskeinen palvelumalli, jossa hajautettu järjestelmä muuttaa itseään, voidakseen paremmin täyttää käyttäjän ja sovellusten vaatimukset.

Tässä työssä kuvataan 2K käyttöjärjestelmää ja sen reflektiivisiä ominaisuuksia.

Sisältö

1 Johdanto

Vaikka aiempi hajautettujen käyttöjärjestelmien tutkimus on ratkaissut monia resurssien hallintaan liittyviä ongelmia, niissä harvoin kuitenkaan käsitellään heterogeenisyyteen ja dynaamiseen sopeutumiseen liittyviä ongelmia. CORBAn ja Java/JINI:n kaltaiset väliohjelmistot ratkaisevat osan heterogeenisyyteen liittyvistä ongelmista, mutta ne puolestaan eivät ota kantaa dynaamiseen resurssien hallintaan tai hajautettujen sovellusten sopeutuvuuteen. [KCM+00]

2K on järjestelmä-rajapinta, joka pyrkii ratkaisemaan kaikki edellä mainitut ongelmat samalla kertaa, yhdistämällä hajautettujen käyttöjärjestelmien ja CORBAn edut.

2K muodostaa ympäristön, joka pystyy sopeutumaan sekä hitaisiin että nopeisiin muutoksiin. Hitaisiin muutoksiin kuuluu laitteiston päivitykset ja uudet ohjelmaversiot. Nopeita muutoksia ovat esimerkiksi prosessorin kuorma, käytettävissä oleva muisti ja verkkoyhteys. Kaikissa tilanteissa järjestelmä pyrkii optimoimaan suorituskyvyn ottaen huomioon käytettävissä olevat resurssit ja ympäristön asettamat vaatimukset. Käytössä on aina ainoastaan ne komponentit, joita todella tarvitaan.

Järjestelmän komponentit liittyvät mm. automaattiseen konfigurointiin, dynaamiseen tietoturvaan ja QoS-tietoiseen hajautettujen resurssien hallintaan. [KCM+00]

Väliohjelmistojen toteuttajat pyrkivät tekemään tehokkaita toteutuksia ominaisuuksille, jotka ovat monille sovelluksille yhteisiä. Mikään tietty ominaisuusjoukko ei kuitenkaan ole riittävä kaikille sovelluksille. Niinpä sovellusten ohjelmoijien täytyy joko toteuttaa uusia ominaisuuksia väliohjelmiston päälle tai muokata väliohjelmiston toteutusta. Kumpikaan vaihtoehdoista ei ole tyydyttävä. Väliohjelmistoa ei voida myöskään optimoida tehokkaiksi kaikille sovelluksille ja alustoille. Näiden havaintojen pohjalta on argumentoitu muunneltavan väliohjelmiston puolesta. [SSC98]

Kun monimuotoisuus tietokonelaitteistossa ja käyttäjien vaatimuksissa kasvaa, tulee entistä vaikeammaksi tarjota jokaiseen eri tilanteeseen sopivaksi räätälöity käyttöjärjestelmä. Käyttäjä tarvitsee käyttöjärjestelmän, joka pystyy konfiguroimaan itsensä sopeutuakseen annettuun laitteistoarkkitehtuuriin sekä käyttäjä- ja sovellus-spesifeihin vaatimuksiin. [KCC99]

Olemassa olevat käyttöjärjestelmät eivät kykene sopeutumaan näihin erilaisiin arkkitehtuureihin ja palveluvaatimuksiin. Yleisesti ottaen käyttöjärjestelmien mukana on monia tarpeettomia komponentteja. Tämä johtaa resurssien tehottomaan käyttöön ja laskee suorituskykyä. Lisäksi käyttöjärjestelmän uudelleenkonfigurointi vaatii joko sen kääntämisen uudestaan, tai ainakin koneen uudelleenkäynnistämisen. [KCC99]

Mikrokernel-pohjaiset tai dynaamisesti laajennettavat järjestelmät mahdollistavat käyttöjärjestelmän uudelleenkonfiguroinnin ajon aikana. Silloin ongelmana ei ole enää *miten* konfiguroida järjestelmä, vaan *koska* ja *millä tavalla*. Käyttöjärjestelmäympäristön suorituskyvyn ja käytettävyyden vuoksi on ratkaisevaa pystyä tarkasti määrittelemään, mitkä ohjelmistomoduulit on asennettava.

Yhdessä ääripäässä on minimalistinen järjestelmä, jossa vain muutama erittäin tärkeä moduuli on ladattuna. Tämä konfiguraatio voi olla vakaa ja tehokas, mutta se ei todennäköisesti vastaa kaikkia sen käyttäjien tarpeita. Toisessa ääripäässä järjestelmä voidaan konfiguroida sisältämään kaikki mitä voitaisiin tarvita. Tämä on nykyisten käyttöjärjestelmien yleisimmin ottama lähestymistapa. Liian paljon ohjelmistopaketteja on suorituksessa, kuluttamassa resursseja ja laskemassa niiden sovellusten suorituskykyä, joita todella tarvittaisiin. [KCC99]

Väliohjelmiston muuntaminen sovelluskohtaisesti näyttäisi olevan tehokas lähestymistapa kestävien ja suorituskyvyltään tehokkaiden sovellusten rakentamiseen. Tutkimus on osoittanut, että yksittäisten komponenttien optimoinnin lisäksi, järjestelmän hyvä suorituskyky vaatii optimointeja päästä päähän -kommunikoinnissa, järjestelmässä itsessään ja eri alijärjestelmissä. [SSC98]

Tässä seminaarityössä kuvataan ensin 2K:n toimintaa yleisesti. Tämän jälkeen kuvataan tarkemmin riippuvuuksien hallintaa, joka on 2K:n ydintoimintoja. Lopuksi käsitellään vielä ympäristöjen hallintaa, joka mahdollistaa järjestelmän käytön laitteistosta ja sijainnista riippumatta.

2 2K-käyttöjärjestelmän yleiskuvaus

2K on reflektiivinen käyttöjärjestelmä, joka rakentuu reflektiivisen ORB:n (Object Request Broker) päälle. 2K:n pääominaisuuksia ovat hajautus, käyttäjäkeskeisyys, sopeutuvuus ja arkkitehtuuritietoisuus (architectural awareness). Arkkitehtuurin perustana on sovellus- ja käyttäjäkeskeinen palvelumalli, jossa hajautettu järjestelmä muuttaa itseään, voidakseen paremmin täyttää käyttäjän ja sovellusten vaatimukset. [CKB^{+}00]

Tehokas, reflektiivinen väliohjelmistokerros tukee dynaamisesti konfiguroitavaa CORBA-toteutusta. Lisäksi CORBAn päällä olevat ympäristöt tarjoavat korkean tason abstraktiot ja hajautetut palvelut. [KCC99]

Tavoitteena on toteuttaa 2K niin tehokkaasti, ja pitää sen muistijälki niin pienenä, että sitä voitaisiin ajaa periaatteessa missä tahansa laitteessa, jossa on riittävästi tehoa kevyen CORBA-yhteensopivan asiakkaan ajamiseen ja ajoittainen kommunikointimahdollisuus [CKB$^+$00]. Käytännön tavoitteena on ajaa 2K:ta yleisimpien käyttöjärjestelmien päällä, mm. PalmOS, UNIX-variantit ja eri Windows-versiot mukaanlukien Windows CE.

2K on "meta-käyttöjärjestelmä" siinä mielessä, että sitä ajetaan olemassa olevan käyttöjärjestelmän päällä [CKR$^+$00]. Voisi väittää, että 2K ei olekaan käyttöjärjestelmä, vaan väliohjelmistojärjestelmä. Projektissa sitä kutsutaan kuitenkin mieluummin käyttöjärjestelmäksi, koska se on vastuussa koneissa olevien laitteisto- ja ohjelmistoresurssien hallinnoimisesta [KCC99].

2K on komponentti-suuntautunut käyttöjärjestelmä. Toisin kuin olioilla, 2K:n komponenteilla on eksplisiittiset laitteisto- ja ohjelmisto-edellytykset, joiden täyttymisen takaa automaattinen konfigurointipalvelu. [CKB$^+$00]

2K tarjoamat mahdollisuudet esimerkiksi tietoturva-osajärjestelmän dynaamiseen konfigurointiin ovat Kon et al:n mukaan erittäin käyttökelpoisia monenlaisissa sovelluksissa ja useissa tilanteissa. Esimerkkinä artikkelissa [KCM$^+$00] on käytetty paljon laskentatehoa vaativaa sovellusta, jota aluksi suoritetaan yhdessä klusterissa, mutta myöhemmin se laajentaa suoritustaan Internetin välityksellä useampiin klustereihin. Kevyen salauksen ja kulunvalvonnan käyttö voi olla hyväksyttävää intranetissä, mutta Internetin välityksellä kommunikoitaessa voi olla tarpeen käyttää vahvempaa salausta ja kulunvalvontaa. Sama koskee kannettavia tietokoneita ja PDA-laitteita, kun se siirtyvät alueelta toiselle.

2.1 Resurssien hallinta

Sekä keskitetyn että hajautetun käyttöjärjestelmän perustehtävänä on hallita koneen (tai koneiden) resursseja. Perinteiset käyttöjärjestelmät eivät kuitenkaan kykene hallitsemaan heterogeenisten koneiden joukossa olevia resursseja. [KCM$^+$00]

Esimerkiksi CORBA ja JINI ovat muodostuneet tehokkaiksi teknologioiksi heterogeenisten ympäristöjen yhteen sovittamisessa. Niistä puuttuu kuitenkin "käyttäjän" käsite, eivätkä ne tue dynaamista resurssien hallintaa yksittäisessä koneessa tai hajautetussa ympäristössä. [KCM+00]

2K:ssa otettu lähestymistapa yhdistää CORBAn ja hajautettujen käyttöjärjestelmien edut. Se mahdollistaa hajautettujen resurssien hallinnan ja pystyy toimimaan eri laitteistoilla ja erilaisten käyttöjärjestelmien päällä. [KCM+00]

2K tehostaa resurssien hallintaa palvelulaatuun (Quality of Service, QoS) liittyvillä algoritmeilla. Sovellusohjelmoijilla on siten täysi pääsy järjestelmän dynaamiseen tilaan ja he voivat toteuttaa sovelluskohtaiset adaptiot samalla kun järjestelmä takaa, että haluttu palvelulaadun taso säilytetään.

2K:n resurssien hallinta-palvelu [Yam00] käyttää paikallisia resurssimanagereita (Local Resource Manager, LRM) jokaisessa 2K-klusterin solmussa. Niiden tarkoituksena on antaa kyseisen solmun laitteistoresurssit koko hajautetun järjestelmän käyttöön. LRM:t lähettävät säännölisesti tietoa resurssiensa tilasta globaalille resurssimanagerille (Global Resource manager, GRM). GRM on kopioitu palvelu, joka ylläpitää likimääräistä näkymää klusterin resurssien käyttöasteesta. GRM käyttää tätä tietoa avukseen suorittaessaan QoS-tietoista kuormantasausta klusterin sisällä. [KCM+00]

2.2 Reflektiiviset ORB:t

2K:ssa on käytössä *verkkokeskeinen* malli, jossa kaikki entiteetit (käyttäjät, ohjelmakomponentit ja laitteet) ovat verkossa ja ne on kuvattu CORBA olioina. Jokaisella entiteetillä on koko verkon laajuinen identiteetti ja profiili sekä riippuvuudet muista verkon entiteeteistä. Kun palvelu instantioidaan, palvelun muodostavat entiteetit kootaan. [KCM+00]

Jotta dynaamisessa, hajautettuja resursseja sisältävässä ympäristössä saavutettaisiin sovelluksissa optimaalinen suorituskyky, täytyy väliohjelmiston olla konfiguroitava ja sopeutua muutoksiin sekä resurssien saatavuudessa että ohjelmiston ja laitteiston infrastruktuurissa. Tähän 2K käyttää dynaamisesti konfiguroitavaa reflektiivistä ORB:ia, jonka avulla sovellukset voivat hyötyä korkeasta joustavuudesta säätämällä CORBA implementaatiota omiin spesifeihin tarpeisiinsa.

Reflektiivinen ORB vastaa kysymykseen miten järjestelmä voi sopeutua sovelluksen tarpeisiin, mutta se ei vastaa kysymyksiin miten ja milloin sopeutua. 2K:ssa nämä ongelmat pyritään ratkaisemaan ylläpitämällä eksplisiittistä kuvausta järjestelmän ja sovellusten kesken olevista dynaamisista riippuvuuksista. Lisäksi tätä dynaamista tilaa voidaan tutkia ja seurata. [KCM+00]

Yksi 2K:n olennaisista osista on *dynamicTAO* [KRL+00], COR-BA-yhteensopiva reflektiivinen ORB. Se on TAO ORB:n laajennus, joka mahdollistaa lennosta tapahtuvan uudelleenkonfiguroinnin ORB:n sisällä ja sen päällä toimivissa sovelluksissa. dynamicTAO:ssa käytetään komponenttikonfiguroijia kuvaamaan riippuvuuksia eri ORB-komponenttejen sekä ORB- ja sovelluskomponenttejen välillä. [KCM+00] Komponenttikonfiguroijia käsitellään tarkemmin myöhempänä luvussa 3.2.2.

Tyypillisesti sovellukset käyttävät vain hyvin pientä osaa tavallisten ORB:ien tarjoamista palveluista ja toiminnoista. Lisäksi yksi CORBAan usein kohdistuvista kritiikeistä on, että se on liian raskas käytettäväksi pienissä laitteissa ja sulautetuissa järjestelmissä. Vaikka dynamicTAO on dynaamisesti konfiguroitava, sen muistijälki ei ole koskaan alle muutaman megatavun, mikä tekee sen sopimattomaksi ympäristöihin joissa on rajoitetusti resursseja ja sovelluksiin joilla on tiukat resurssivaatimukset. Tämä on motivoinut ryhmän kehittämään uuden ORB-arkkitehtuurin nimeltä *LegORB* [RMKC00]. Se voidaan dynaamisesti mukauttaa sopeutumaan resurssien saatavuuteen ja ottamaan huomioon eri sovellusten ja laitteiden vaatimukset eri hetkillä. [KCM+00]

Toisin kuin TAO, LegORB on suunniteltu komponentisointi ja dynaaminen uudelleenkonfigurointi perusedellytyksinä. LegORB:n minimaalinen konfigurointi, joka kykenee lähettämään yksinkertaisia CORBA-kutsuja standardeille ORB:lle vie vain n. 6 kilotavua muistia PalmOS:ssa. Alustavien tulosten mukaan näyttäisi siltä, että LegORB voisi olla sulautettujen järjestelmien ja PDA-laitteiden lisäksi hyvä valinta myös työasemiin, joissa LegORB voi olla jopa nopeampi kuin korkeasti optimoidut kaupalliset ORB:t.

3 Riippuvuuksien hallinta

Toisin kuin nykyisissä järjestelmissä, missä järjestelmän asennuksessa mukana tulee monia tarpeettomia moduuleita, 2K:n filosofia perustuu "What You Nees Is What You Get" (WYNIWYG) -malliin. Järjestelmä konfiguroi itsensä automaattisesti ja lataa minimaalisen joukon komponentteja, jotka tarvitaan käyttäjän sovellusten suorittamiseksi mahdollisimman tehokkaasti. Tämä toteutetaan standardien CORBA-palveluiden (Naming, Trading, Security, Persistence) avulla ja lisäksi lisäämällä omat palvelut QoS-tietoiseen resurssien hallintaan, automaattiseen konfigurointiin ja koodin jakeluun. [KCM+00]

3.1 Riippuvuusongelmia

Viimeaikaikoina kehitetyt komponettiarkkitehtuurit (esim EJB, ActiveX Controls ja CORBA Component Model) mahdollistavat hienostuneiden järjestelmien rakentamisen kokoamalla valmiista ohjelmakomponenteista joko visuaalisia työkaluja tai ohjelmallisia rajapintoja käyttäen. Komponenttien välisten riippuvuuksien hallintaa on kuitenkin tuettu hyvin vähän.

Eri ohjelmoijat luovat komponentteja. Usein he työskentelevät eri ryhmissä ja käyttävät erilaisia menetelmiä. On vaikeaa luoda kestäviä ja tehokkaita järjestelmiä, ellei ymmärrä komponenttien välisiä dynaamisia riippuvuuksia. Niinpä onkin hyvin yleistä löytää tapauksia, niin perinteisissä kuin komponenttipohjaisissa järjestelmissä, joissa moduuli epäonnistuu tehtävänsä suorittamisessa, koska järjestelmä ei pysty ratkaisemaan jotain yksittäistä riippuvuutta. Joskus muut moduulit eivät kunnolla havaitse pientä virhetilannetta yhdessä moduulissa, mikä johtaa totaaliseen järjestelmävirheeseen. [KC00]

Käyttöjärjestelmän ylläpitäjien täytyy seurata tietoturvailmoituksia ja olla valmiita päivittämään järjestelmiä päivittäin. Myös käyttäjät vaativat uusia versioita sovelluksista. Monesti uuden ohjelmistopaketin asentaminen vaatii päivityksiä myös muihin työkaluihin. [KC00]

Edellä kuvattujen vaikeuksien takana on se, että ei ole sellaista mallia, joka kuvaisi riippuvuudet järjestelmän ja sovellusten komponenttien välillä eikä työkaluja näiden riippuvuuksien hallintaan. Kon ja Campbell ovat sitä mieltä, että käyttöjärjestelmän ja väliohjelmistoympäristöjen täytyy eksplisiittisesti kuvata riippuvuudet ohjelmakomponettien välillä [KC00]. Silloin kuvausta muokkaamalla on mahdollista toteuttaa ohjelmakomponetteja, jotka voivat konfiguroida itseään ja sopeutua jatkuvasti muuttuviin dynaamisiin ympäristöihin.

Järjestelmän ja sovelluskomponenttien välisten vuorovaikutusten kuvaaminen (reification) antaa järjestelmäohjelmiston tunnistaa uudelleenkonfigurointitarpeen, ja niin ollen mahdollistaa paremman tuen palveluille (vikasietoisuus, tietoturva, QoS ja optimointi). Lisäksi kuvaaminen antaa järjestelmäohjelmistolle mahdollisuuden uudellenkonfiguroitua, tinkimättä järjestelmän vakaudesta ja luotettavuudesta sekä vaikuttaen vain minimaalisesti järjestelmän suorituskykyyn. [KC00]

Projektissa otettu lähestymistapa erottaa komponenttienvälisen kommunikaation komponenttienvälisestä riippuvuudesta. Näin 2K:n kehystä voidaan käyttää erilaisten kommunikointimenetelmien kanssa. Kommunikointi ja riippuvuus liittyvät monesti läheisesti toisiinsa, mutta usein niiden erottamisesta on kuitenkin hyötyä. Esimerkiksi multimediasovelluksen palvelulaatu riippuu pitkälti alla olevista käyttöjärjestelmän palveluista kuten virtuaalimuisti, skedulointi ja muistin varaus. Näiden palvelujen ja sovelluksen välinen vuorovaikutus on monesti implisiittistä, eli järjestelmäkutsuja tai muuta suoraa kommunikointia ei tapahdu. Mutta mikäli järjestelmän infrastuktuuri kuitenkin antaa sovelluskehittäjälle mahdollisuuden määrittää ja muokata sovelluksen ja näiden palvelujen välisiä riippuvuussuhteita, sovellusta voidaan informoida näiden palvelujen tilassa tai konfiguraatiossa tapahtuvista merkittävistä muutoksista, joilla saattaa olla vaikutusta sen suorituskykyyn.

Sen sijaan että tarkasteltaisiin yhden sovelluksen komponenttien välillä olevia suhteita, tässä lähestymistavassa otetaan huomioon kaikki mahdolliset riippuvuudet, jotka sitovat kunkin komponentin muiden sovellusten, väliohjelmiston ja järjestelmän komponentteihin. [KC00]

3.2 Riippuvuudet komponenttien välillä

Komponenttien välillä olevat riippuvuudet voidaan jakaa edellytyksiin ja dynaamisiin riippuvuuksiin. Edellytyksillä tarkoitetaan vaatimuksia komponentin lataamiseksi ajonaikaiseen ympäristöön. Dynaamiset riippuvuudet puolestaan ovat suorituksessa olevien komponenttien välisiä. [KC00]

3.2.1 Edellytykset

Komponentin edellytysten täytyy määritellä kaikki erityisvaatimukset, mitä edellytetään komponentin lataamiseksi, konfiguroinniksi tai suorittamiseksi. Edellytyslistaan voi kuulua kolmea erilaista informaatiota: Minkälaisia laitteistoresursseja komponentti tarvitsee, tarvittavan laitteistoresurssin kapasitetti ja ohjelmistopalvelut, jotka se vaatii. [KC00]

Esimerkiksi hajautettujen resurssien hallinta-palvelu voisi käyttää kahta ensiksi mainittua selvittääkseen missä, miten ja koska suorittaa komponentteja. QoS-tietoiset järjestelmät voivat käyttää tätä tietoa mahdollistaakseen asianmukaisen kulunvalvonnan, resurssien neuvottelun ja resurssien varaamisen. Viimeksimainittu määrää, mitä apukomponentteja täytyy ladata ja minkälaisessa ympäristössä ne suoritetaan.

Edellytyksiin voi kuulua määrällinen tai laadullinen kuvaus laitteistoresursseista jotka komponentti vaatii (esim CPU, verkko, muisti ja video enkooderit), lista muista vaadituista ohjelmistokomponenteista (esim kirjastot, paikalliset ja hajautetut järjestelmäpalvelut) ja palvelulaadun kuvauksia. [KCC99]

Projektissa on päätetty kehittää kehys, jossa voitaisiin käyttää erilaisia edellytysten kuvauksia. Näin vältetään rajoittumasta tiettyyn kuvauskieleen. Projektissa on toteutettu hyvin yksinkertainen tekstipohjainen kuvauskieli (SPDF). Myöhemmin on tarkoitus tehdä kokeiluja kehittyneemmillä QoS-kuvauksilla esim QML ja QoS-aspektikielillä.

Edellytykset voivat olla myös CORBAn IDL-kuvauksia. Vaikka IDL-lähestymistapa on hyvin joustava, siinä on kirjoitettava suoritettavaa koodia, että voitaisiin generoida CORBAn tietorakenteita. Esim kovakoodaamalla esiehtospesifikaatio komponentin toteutukseen. Tämä rajoitus sai projektin etsimään ratkaisua, jossa ehdot voitaisiin kuvata yksinkertaisesti vain muokkaamalla tekstitiedostoa. Tähän päästiin SPDF:n avulla. [KCC99]

3.2.2 Dynaamiset riippuvuudet

2K:n käyttämässa mallissa *komponenttikonfiguroija* (component configurator) hallinnoi jokaista komponenttia. Se on vastuussa kyseisen komponentin ja muiden järjestelmä- ja sovelluskomponenttien välisten ajonaikaisten riippuvuuksien varastoinnista.

Jokaisella komponentilla C on *koukkuja* (hooks), joihin muut komponentit voivat kiinnittyä. Nämä koukutetut komponentit ovat komponentteja, joista C riippuu. Lisäksi voi olla muita komponentteja, *asiakkaita*, jotka riippuvat C:stä. [KC00]

Komponenttikonfiguroija ylläpitää listaa koukuista. Riippuvuudet tallennetaan CORBA IOR:inä toisiin komponenttikonfiguroijiin ja ne muodostavat hajautettujen komponenttien verkon. [KCC99]

Esim Web-selain voisi vaatia toimiakseen TCP/IP-palvelun, ikkunamanagerin, paikallisen tiedostopalvelun ja Java virtuaalikoneen toteutuksen. Sen komponenttikonfiguroijan pitäisi ylläpitää koukkua jokaista edellämainittua palvelua kohti. Kun selain ladataan, järjestelmän täytyy selvittää ovatko nämä palvelut käytettävissä paikallisessa ympäristössä. Jos eivät, järjestelmän täytyy luoda niistä uudet instanssit. Joka tapauksessa järjestelmä tallentaa viitteet palveluihin selaimen konfiguroijan koukkuihin, ja voi myöhemmin noutaa ne tai päivittää niitä. [KC00] Osa vaatimuksista voi olla valinnaisia, esimerkiksi yllä olevassa esimerkissä selain voisi toimia vaikka tiedostopalvelua ja Java virtuaalikonetta ei olisi käytettävissä.

Tieto ajon aikaisista riippuvuuksista antaa sovellukselle mahdollisuuden viitata omiin riippuvuuksiinsa ja valita eri komponentit täyttämään sen tarpeet eri ympäristöissä ja eri hetkinä. [KCC99]

Esimerkiksi kun kannettava tietokone ei ole kytkettynä lähiverkkoon, järjestelmä voi automaattisesti havaita, että langallisen verkkoyhteyden toteuttava palvelu ei toimi ja sovellus ohjataan automaattisesti käyttämään langatonta yhteyttä. [KCC99]

3.3 Mobiilit konfigurointiagentit

Luvussa 3.2.1 on käytössä *pull*-pohjainen lähestymistapa koodin jakeluun, missä järjestelmä noutaa komponentit komponenttivarastosta tarvittaessa. Tämän lisäksi 2K tukee myös *push*-mekanismia, joka pohjautuu mobiilien konfigurointiagenttien käyttöön. Monissa tapauksissa tämä vaihtoehtoinen mekanismi voi parantaa järjestelmän suorituskykyä parantamalla koodipäivityksiä skaalautuvalla tavalla. [KCM+00]

Reflektiiviset ORB:t muodostavat jakeluverkoston, jonka kautta järjestelmän ylläpitäjät tai sovellukset voivat lähettää konfigurointi ja tutkimusagentteja. Agentit voivat sisältää konfigurointikomentoja ja uusia implementaatioita järjestelmän tai sovellusten komponenteista (esim dynaamisten kirjastojen tai Javan tavukoodin muodossa). Näiden mekanismejen kombinaationa saadaan joustava infrastruktuuri dynaamisille ohjelmistopäivityksille. [KCM+00]

3.4 Automaattinen konfigurointi

Kuvatut komponenttienväliset riippuvuudet voivat auttaa konfigurointiprosessin automatisoinnissa. Käyttöjärjestelmä tai väliohjelmisto voi tutkia esiehtolistan varmistaakseen, että kaikki laitteisto- ja ohjelmistovaatimukset tietyn komponentin suorittamiseksi täyttyvät, ennen kuin komponentin suoritus alkaa. Tämä voi ennaltaehkäistä monia nykyjärjestelmissä tavallisesti esiintyviä ongelmia, jotka aiheutuvat siitä, että tietyn komponentin tai resurssin puuttuminen havaitaan vasta silloin, kun sovellus on jo suorituksessa. [KC00]

Dynaaminen riippuvuusinformaatio puolestaan mahdollistaa jo suorituksessa olevien komponenttien uudelleenkonfiguroinnin. Vaikka 2K:n infrastruktuuri itsessään ei takaa turvallista uudelleenkonfigurointia, se tarjoaa hyödyllisen kehyksen jonka avulla ohjelmoijat voivat toteuttaa turvallisen uudelleenkonfiguroinnin helpommin ja yhdenmukaisemmin. [KC00]

3.5 Mallin heikkoudet

Kun 2K:n mallia käytetään C++:n kaltaisessa ohjelmointikielessä, komponentin kehittäjän on ehdottomasti noudatettava annettuja ohjesääntöjä. Tärkeää on myös, että kaikki kommunikointi komponenttien välillä tapahtuu kontrolloidusti rajapintojen kautta. Jotta voitaisiin kuitenkin estää kuvauksiin liittyvien ohjelmointivirheiden lisääntyminen, on kehittävä erityisiä kieliä, kääntäjiä ja runtime-järjestelmiä, joilla komponenttien suorituksen ja uudelleenkonfiguroinnin turvallisuus voidaan taata. [KC00]

Siistimpi ratkaisu olisi käyttää olemassaolevia reflektiivisiä kieliä ja ympäristöjä[1]. 2K:n päätavoitteena on kuitenkin käyttää laajasti hyväksyttyjä standardeja, rajoittumatta tiettyyn ohjelmointikieleen.

Toinen mahdollinen ratkaisu olisi sitoa yhteen mekanismit kommunikointiin ja riippuvuuksien kuvaamiseen, käyttämällä esim. rakenteellisia liittimiä (architectural connectors). 2K:n tavoitteena ei kuitenkaan ole rajoittaa mallin ilmaisuvoimaa, vaan kehittää yleinen metodologia, jota voidaan käyttää useissa heterogeenisissä ympäristöissä. Nämä vaatimukset voidaan täyttää ainoastaan käyttämällä jotain standardia arkkitehtuuria, kuten esimerkiksi CORBAa. [KC00]

3.6 Arkkitehtuuritietoisuus

Arkkitehtuuritietoisuus saavutetaan kuvaamalla laitteiston ja ohjelmiston edellytykset jokaiselle ladattavalle komponentille. Käyttöjärjestelmä käyttää tätä tietoa varmistaakseen, että kaikki komponentin vaatimat peruspalvelut ovat käytettävissä ennen kuin komponentti ladataan. Lisäksi hajauttujen resurssien manageri käyttää komponentin laitteistovaatimusten määrityksiä päättäessään mihin koneeseen komponentti ladataan ja lisäksi suorittaa kulunvalvonnan ja resurssien varaamisen. [KC00]

Kun 2K-komponentissa tapahtuu virhe, järjestelmä tutkii sen riippuvuudet ja

[1]Esim. Iguana, C++:n laajennus, joka reifioi useita kielen ominaisuuksia ja antaa muokata niiden toteutusta dynaamisesti.

tiedottaa asianomaisia komponentteja virheestä. Vaihtoehtoisesti järjestelmä voisi toipua virheestä korvaamalla viallisen komponentin uudella. [KC00]

4 Ympäristöjen hallinta

2K:ssa suoritus tapahtuu käyttäjän tarpeisiin muunneltavissa ympäristöissä. 2K-ympäristö on säiliö, joka sisältää komponentit, laitteet ja konfigurointiparametrit ja tarjoaa suoritusympäristön hajautetussa järjestelmässä oleville käyttäjille. Suoritusympäristöjen hallinta edistää järjestelmän käyttäjäkeskeisyyttä, keräämällä yhteen kaikki käyttäjän kontekstit yhdistelmärakenteeseen joka on saatavissa mistä tahansa 2K-järjestelmän aluella. yhdistelmäympäristöjen rakenteet vuorovaikuttavat itsensä ja toistensa kanssa ja ovat korreloivia, sillä tietyssä ympäristössä tehdyt toiminnot aiheuttavat toimintoja muissa ympäristöissä. [CKR+00]

Tietokonejärjestelmät ovat tulossa yhä monimutkaisemmiksi. Käyttäjillä on tavallisesti käytössään monia laitteita. Eri käyttäjät eivät välttämättä jaa samaa näkymää järjestelmään. Tämän lisäksi käyttäjä ei tarvitse koko ajan samaa näkymää järjestelmään. Tällaisiin tarpeisiin LAN-pohjaisten hajautetujen järjestelmien verkkokeskeinen näkymä ei ole riittävä. Projektissa halutaan tarjota käyttäjille yhtenäinen näkymä järjestelmään, riippumatta käytetystä laitteesta tai käyttäjän tai resurssien sijainnista.

Esimerkiksi samalla, kun käyttäjä matkustaisi konferenssiin maapallon toiselle puolelle, hän ehkä haluaisi samalla työskennellä toisen, kotitoimistossa käynnissä olevan, projektin parissa. Hän voisi yksinkertaisesti jäädyttää istunnon toimistosta lähtiessään, ja matkan aikana tai perillä palauttaa istunnon vaikkapa kannettavalla tietokoneella tai kämmenmikrolla. [CKB+00]

Tämä on mahdollista nykypäivänä, mutta prosessi on harmillisen kömpelö käyttäjälle. Sen lisäksi on lukuisia esiehtoja, jotka täytyy asettaa kohdalleen etukäteen, kuten tunnukset, pääsy jne, jotka yleisesti eivät ole helposti käsillä. [CKB+00]

2K:n suoritusympäristöt toteuttavat mekanismit, joilla voidaan tarjota käyttökelpoinen, liikkuva, yhtenäinen ja tehokas työtila, joka voi sopeutua laitteisto- ja ohjelmistokomponenttien evoluutioon ja korvaamiseen. 2K:ssa jokaista käyttäjän istuntoa pidetään ympäristönä, joka sisältää vuorovaikuttavat komponentit, konfigurointiparametrit allaoleville alustoille ja käyttäjän profiilin kyseisessä istunnossa. Osapuolta kuvaa hänen globaali ympäristönsä, joka on yhdiste kaikista hänen sillä hetkellä järjestelmässä olevista ympäristöistään. [CKR+00]

Käyttäjän käytössä oleva hajautettu suoritusympäristö koostuu useammista aliympäristöistä, jotka toimivat eri alustoilla. Ne voivat olla väliaikaisesti keskeytettynä, valmiina palautettavaksi suoritukseen myöhemmin. Suoritusympäristöjen hallinta on suunniteltu tarjoamaan käyttäjäkeskeinen näkymä järjestelmään. Se helpottaa käyttäjän liikkuvuutta vapauttamalla käyttäjän olemasta sidottuna tiettyihin alustoihin ja ottamalla resursseja käyttöön sieltä mistä niitä on saatavissa. [CKB+00]

4.1 Hajautettu ympäristöpalvelu

Ympäristöpalvelu on ympäristötehdas, joka pitää kirjaa järjestelmässä aktiivisina olevista ympäristöistä. [CKB+00]

Osapuoli on entiteetti, ihminen tai komponentti, joka voi autentikoida itsensä järjestelmään, jota voidaan pitää vastuussa toiminnoistaan ja jonka käyttöön voidaan varata resursseja. Esimerkkejä osapuolista on: käyttäjät, aktiiviset sovellus- tai järjestelmäkomponentit, huoneet, rakennukset ja verkot. Osapuolet on rekisteröity 2K-järjestelmään, ne tunnetaan uniikista tunnisteestaan ja ne on karakterisoitu profiilinsa perusteella. [CKR+00]

Ympäristön luomista pyydettäessä, osapuoli tunnistaa itsensä millä tahansa 2K:n käyttöalueella olevalla tietokoneella, terminaalilla, PDA-laitteella tms. Ympäristöpalvelu noutaa osapuolen ympäristörakenteen, luo sen uudelleen ja aktivoi ympäristön osapuolelle. Luontioperaation tyyppi (*start* tai *resume*), määrittelee luodaanko ympäristö alusta vai jatketaanko keskeytettyä istuntoa. [CKB+00]

Ympäristöä suljettaessa, sulkemisoperaation tyyppi määrittelee keskeytetäänkö (*suspend*) vai päätetäänkö (*terminate*) istunto. Keskeytettäessä ympäristön tila talletetaan, jolloin sitä voidaan myöhemmin jatkaa. Ympäristön uudellensijoitus vastaa sen keskeytystä ja jatkamista toisessa paikassa.

4.2 Ympäristöjen käyttäytyminen 2K:ssa

2K on organisoitu palvelinten ja asiakkaiden joukkona, joihin kuuluu: automaattinen konfigurointipalvelu, ympäristöpalvelu, tietoturvapalvelu, nimipalvelu, QoS-tietoinen resurssien hallintapalvelu ja tiedostopalvelu. 2K:n sovellukset, kuten sisäänkirjautumis-sovellus ja Video-On-Demand-palvelu, ovat väliohjelmistopalvelujen asiakkaita. [CKB+00]

Minimissään koneelta vaaditaan 2K:n ajamiseen esilatausympäristö (bootstrap environment) ja ORB:n instanssi. Automaattisen konfigurointipalvelun instanssi on toivottava, komponenttien lataamisen automatisoimiseksi. Perussyinä on, että ORB ja ympäristö tarjoavat minimaalisen tuen 2K:lle jopa silloin kun se ei ole kytkettynä 2K:n hajautettujen palvelujen verkkoon. Järjestelmän ylläpitäjä käynnistää käynnistää alkulatausympäristön ja ORB:n. Käsitteellisesti alkulatausympäristö on järjestelmäkomponentti, jolla on profiili, joka puolestaan määrittelee järjestelmän ajamiseksi tarvittavat komponentit. Alkulatausympäristö toteuttaa laiteriippuvan toiminnallisuuden, joka on tarpeen muiden ympäristöjen käynnistämiseksi. Alkulatausympäristöä käynnistettäessä sen tila alustetaan ja loput palvelut luodaan alkulatausympäristön komponentteina. Automaattinen konfigurointipalvelu selvittää niiden riippuvuudet, kun ne ladataan ja käynnistetään. [CKB+00]

2K-asema on kone, joka antaa osapuolten ottaa yhteyden ympäristöönsä ja suorittaa tehtäviä 2K:ssa. 2K-asema suorittaa alkulatauksen rekisteröimällä itsensä 2K-väliohjelmiston kanssa ja selvittämällä järjestelmän ajamiseen tarvittavat komponentit. Jotkut komponentit, kuten sisäänkirjautumis-sovellus ja paikallinen resurssimanageri määrittelevät, että ne täytyy ajaa paikallisesti, toisia voidaan suorittaa etäkomponentteina. [CKB+00]

4.3 Ympäristöjen yhtenäisyys

Ympäristöpalvelu säilyttää tietoa järjestelmässä olevien osapuolten ympäristörakenteista. Yksinkertaisimmillaan varasto on organisoitu hierarkisiksi alueiksi. Saatavuus-, skaalattavuus- ja luotettavuus-syiden vuoksi tästä palvelusta on useampi instanssi samanaikaisesti suorituksessa. Osapuolten yhdistetyn ympäristörakenteen tila on replikoitu näiden instanssien kesken. Esim kiertelevällä (nomadic) osapuolella voi olla ympäristörakenteensa osittain yhdistettynä tai kopioituna eri alueilla. [CKR+00]

Koska replikointia tapahtuu 2K:n hajautetun ympäristökehyksen useilla tasoilla, tarvitaan yhtenäisyysprotokolla (consistence protocol) yhtenäistämään kopiojoukon tila, kun muutos tapahtuu. [CKR+00]

Vahvaa yhtenäisyyttä käytettäessä lukemiset onnistuvat aina, mutta kirjoittamiset voivat kestää kauemmin. *Heikossa* yhtenäisyydessä kirjoitukset ovat välittömiä, mutta lukemiset voivat vaatia epäkelvon komponentin päivittämisen. Tulevaisuudessa projektissa aiotaan näiden kahden tason lisäksi suorittaa kokeiluja laajemmalla yhtenäisyystasojen kirjolla. Komponentit määrittelevät osana palvelulaadun kuvausta tarvitun kopio-yhtenäisyyden. Eri strategiat sopivat eri tapauksiin. [CKR+00]

5 Yhteenveto

2K on järjestelmä-rajapinta, joka pyrkii ratkaisemaan ongelmat, jotka liittyvät resurssien hallintaan heterogeenisissä tietoverkoissa, dynaamiseen sopeutuvuuteen ja komponenttipohjaisten hajautettujen sovellusten konfigurointiin.

Käyttäjä näkee verkon virtuaalikoneena, jolta hän pyytää palveluita. Ympäristön hallinnan tehtävänä on tehdä ympäristö saavutettavaksi riippumatta käyttäjän sijainnista ja käytetyn koneen tyypistä.

2K:n joustavuus perustuu sen käyttämään reflektiiviseen väliohjelmistokerrokseen, joka mahdollistaa reflektiivisten ominaisuuksien käyttämisen myös järjestelmän palveluissa ja sovelluksissa.

Reflektiivisen väliohjelmistokerroksen tarjoamien etujen lisäksi 2K ylläpitää kuvausta järjestelmän komponenttien välillä olevista riippuvuuksista. Se puolestaan mahdollistaa automaattisen uudelleenkonfiguroinnin ja voi parantaa mm. järjestelmän vikasietoisuutta, tietoturvaa, palvelulaatua ja muita palveluita. Periaatteessa kaikki palvelut voivat käyttää järjestelmän reflektiivisiä ominaisuuksia hyödykseen.

Projektissa halutaan kyetä käyttämään standardeja ja yleisesti käytössä olevia ohjelmointikieliä ja menetelmiä. Esim C++:n käyttö tuo toteutukseen hieman ylimääräistä lisätyötä verrattuna reflektiivisiin ohjelmointikieliin, mutta esim olemassa olevan ohjelmakoodin tai komponenttien käyttö on helpompaa eikä komponenttien tai sovellusten toteutusta tarvitse rajoittaa esim tiettyyn ohjelmointikieleen.

Väliohjelmisto on CORBA-yhteensopiva, minkä ansiosta voidaan käyttää hyödyksi standardeja CORBA-komponentteja ja toisaalta myös kommunikoida helposti olemassaolevien CORBA-toteutusten kanssa.

2K:ssa on kiinnitetty huomiota erityisesti suorituskykyyn. Reflektiivinen väliohjelmisto mahdollistaa resurssien käytön huomattavasti tehokkaammine kuin perinteiset väliohjelmistot, koska komponentti voi käyttää hyödykseen reflektiolla saatavaa lisäinformaatiota ja lisäksi tarvittaessa konfiguroida ja optimoida väliohjelmiston toteutuksen omiin erityistarpeisiinsa sopivaksi.

Viitteet

[CKB+00] Dulcineia Carvalho, Fabio Kon, Francisco Ballesteros, Manuel Román, Roy Campbell, and Dennis Mickunas. Management of Execution Environments in 2K. In *Proceedings of the Seventh International Conference on Parallel and Distributed Systems (ICPADS'2000)*, pages 479–485. IEEE Computer Society, July 2000.

[CKR+00] Dulcineia Carvalho, Fabio Kon, Manuel Roman, Francisco Ballesteros, Christopher Hess, Roy Campbell, and Dennis Mickunas. Interaction and distribution of execution environments in 2k. In *Fourth World Multiconference on Systemics, Cybernetics and Informatics, SCI 2000*, volume II - Information Systems Development, pages 491–496. International Institute of Informatics and Systemics, July 2000.

[KC00] Fabio Kon and Roy H. Campbell. Dependence Management in Component-Based Distributed Systems. *IEEE Concurrency*, 8(1):26–36, January-March 2000.

[KCC99] Fabio Kon, Dulcineia Carvalho, and Roy Campbell. Automatic Configuration in the 2K Operating System. In *Proceedings of the ECOOP'99 Workshop on Object Orientation and Operating Systems*, pages 10–14, Lisbon, June 1999.

[KCM+00] Fabio Kon, Roy H. Campbell, M. Dennis Mickunas, Klara Nahrstedt, and Francisco J. Ballesteros. 2K: A Distributed Operating System for Dynamic Heterogeneous Environments. In *Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing (HPDC'9)*, pages 201–208, Pittsburgh, August 2000.

[KRL+00] Fabio Kon, Manuel Román, Ping Liu, Jina Mao, Tomonori Yamane, Luiz Claudio Magalhães, and Roy H. Campbell. Monitoring, Security, and Dynamic Configuration with the dynamicTAO Reflective ORB. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'2000)*, number 1795 in LNCS, pages 121–143, New York, April 2000. Springer-Verlag.

[RMKC00] Manuel Román, Dennis Mickunas, Fabio Kon, and Roy H. Campbell. LegORB and Ubiquitous CORBA. In *Proceedings of the*

IFIP/ACM Middleware'2000 Workshop on Reflective Middleware, pages 1–2, Palisades, NY, April 2000.

[SSC98] Ashish Singhai, Aamod Sane, and Roy Campbell. Quarterware for Middleware. In *Proc. 18th International Conference on Distributed Computing Systems (ICDCS)*, pages 192–201. IEEE, May 1998.

[Yam00] Tomonori Yamane. The Design and Implementation of the 2K Resource Management Service. Master's thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, February 2000.