

Sovellusalustan integrointiominaisuudet

Tuukka Hastrup

Tuukka@iki.fi

24.3.2004

Tiivistelmä

Sovellusalusta tarjoaa valmiita toimintoja sovelluksille, ja sen ominaisuudet ovat oleellisia sovelluksista integroidussa kokonaisjärjestelmässä. Yhteisestä toimintaperustasta riippuu, millä tasolla toisistaan riippumattomat sovellukset toimivat yhdessä.

1 Johdanto

Haluttaessa laajoja muutoksia vallitsevien järjestelmien rakenteisiin saadaan suunnitella uusiksi isojakin osia tietokoneessa käytettävistä ohjelmitoista, jolloin muutoksille tarvitaan vankat perustelut. Valtavirrasta poikkeavaa järjestelmää voisi lähteä rakentamaan vaikkapa korvaamalla keskusmuistin kekomallin jonkinlaisella assosiaatioverkkorakenteella, mutta sellaisesta on pitkä matka käytännön vaikutuksiin. Toisaalta kolmiulotteinen käyttöliittymä poikkeaisi näkyvästi nykyisistä, mutta muutokset olisivat pintatasolla. Matalasta tasosta tai käyttöliittymästä huolimatta voidaan tarkastella sovellusalustan ominaisuuksia sovellusten vuorovaikutuksen kannalta.

Tässä raportissa pyritään kartoittamaan vertailukohtia yleiskäyttöisissä sovellusalustoissa juuri niiden integrointiominaisuuksista. Ensin motivoidaan sovellusalustaan ja integrointiin keskittymistä. Itse kartoitus koostuu kahden erilaisen sovellusalustan tarkastelusta: vallitsevat käyttöjärjestelmät siltä osin kuin niiden piirteet yhtenevät, ja Haystack, joka on asioiden merkityssuhteisiin perustuva tiedonhallintajärjestelmä.

2 Sovellusalusta

Tässä raportissa tarkoitetaan sovellusalustalla sitä yleistä ympäristöä, johon sovellukset ohjelmoidaan ja jossa ne toimivat. Erityisesti siis ei tarkoiteta

Taulukko 1: Ohjelmistojen yhteistoiminnan tasoja.

Laaajuus	Vaatimukset
Samassa koneessa	yhteinen käyttöjärjestelmä, ohjelmien käynnistys
Samalla näytöllä	moniajo, ikkunointi
Samaa tietoa	tiedostojärjestelmä, -formaatti
Samaan aikaan	versionhallinta, synkronointi, yhteinen tietomalli

jonkin tietyn sovellusalueen ohjelmien ympäristöä.

Käyttöjärjestelmä erotetaan sovellusalustasta, sillä myös käyttöjärjestelmän päälle voidaan toteuttaa ympäristöjä esimerkiksi kirjastoina ja protokollina – tuttu esimerkki on graafinen käyttöliittymä.

Sovellusalusta kattaa siis jossain määrin järjestelmän osat, jotka eivät ole sovelluskohtaisia. Tällaisia voivat olla laitteiden käsittely, sovellusten asennus, käyttöliittymäelementtien yhtenäinen ulkoasu ynnä muut sovelluksiin yleisimmin tarvittavat osat. Yksittäinen sovellus hyödyntää näitä valmiita palveluita ja keskittyy sovellusongelman ratkaisuun.

3 Integrointi

Sovellusalustan tarjoamat palvelut paitsi vähentävät sovelluksessa tarvittavia toimintoja myös mahdollistavat kahden toisistaan erillään kehitetyn ohjelmiston yhteistoiminnan. Käyttöjärjestelmän, kirjastojen ja kehysten tarjoaman yhteisen toimintaperustan ominaisuuksista riippuu, millä tasolla sovellukset integroituvat kokonaisjärjestelmään niiden ollessa silti toisistaan riippumattomat.

Suuren ja monimutkaisen järjestelmän rakenne on tärkeä sen laadullisille ominaisuuksille: tehokkuudelle, luotettavuudelle, skaalautuvuudelle [3]. Ohjelmiston rakenteesta poiketen sovellusalustan rakenne ei ole kokonaisuudessaan yhden tahon suunnittelema ja toteuttama, vaan jokainen sovellus vaikuttaa järjestelmään. Sovellusalustan tarjoamat rajapinnat lienevät keskeisessä osassa järjestelmän kokonaisrakenteessa, sillä jokaisen sovelluksen täytyy niitä käyttää.

Taulukossa 1 on tarkasteltu yhteistoiminnan tasoja. Yleisesti nykyiset ohjelmistot toimivat hyvin samassa koneessa ja tyydyttävästi myös samalla näytöllä. Tarpeellisimmissa tapauksissa ne käyttävät samaa tietoa, mutta harvoin silti samaan aikaan.

4 Vallitsevat käyttöjärjestelmät

Jo THE-järjestelmässä [1] oli *prosesseihin* perustuva, kerrosrakenteinen abstraktio sovellusohjelmien käyttöön. Tavoitteena silloin oli ajaa useaa ohjelmaa yhtä aikaa, jotta oheislaitteet tulivat tehokkaasti hyödynnetyiksi. Rakenne helpotti järjestelmän toiminnan päättelyä ja testausta.

Integroinnin kannalta prosessikäsite mahdollistaa ohjelmien käynnistämisen, suorituksen hallinnan ja lopetuksen ilman sovellusten yhteistyötä. Moniajon myötä ohjelmia voi olla käynnissä useampia yhtä aikaa.

Prosessin lisäksi nykyisissä käyttöjärjestelmissä keskeinen käsite on *tiedosto*. Tiedostojärjestelmän kautta käsiteltynä sovellusten tieto voi olla samalla lohkolaitteella. Jos sovelluksilla on lisäksi yhteinen tiedostomuoto, ne voivat käsitellä samaa tietoa. *Samanaikainen* käsittely on kuitenkin vaara: toisen sovelluksen tekemät muutokset tiedostoon eivät näy toisen sovelluksen toiminnassa.

Unix-tyylisessä tiedostomallissa lohkolaitteiden tiedostojärjestelmiä liitetään osaksi hakemistopuuta, ja alkuperäiseen Unix-filosofiaan kuuluu, että kaikki on käsiteltävissä tiedostoina. Nykyisin kuitenkin tietoverkot ovat sovelluksille ja käyttäjille tärkeitä, mutta niitä ei esitetä tiedostoina.

5 Haystack

Haystack [2] on tiedonhallintajärjestelmä, jossa merkityksellisiä ovat asioiden väliset suhteet. Tiedostojen ja sovellusprosessien sijaan siinä on *RDF-kanta* (Resource Description Framework), jota erilaiset *palvelut* käsittelevät ja *näkymät* esittävät käyttäjälle. Kannan muutokset välittyvät suoraan näkymille ja muutosilmoituksen tilanneille palveluille, jolloin kaikki taulukossa 1 mainitut integrointitasot saavutetaan.

Nykyisessä toteutuksessa palveluita hallitsee Java-prosessi, joka myös lataa RDF-kannan ja toteuttaa palveluiden välisen viestinnän, mutta palvelut voivat toimia omina säikeinään.

Käyttöliittymä on WWW-sivun tai portaalin oloinen: suurimman osan ikkunasta kattaa valitun *kohteen* näkymä, reunoilla on yleisiä valikkoja, muistutuksia ja lisätietoja kohteesta. Järjestelmä valitsee kohteille sopivat näkymät kohteen tyyppin ja esitysyhteyden mukaan, esimerkiksi tehtävälistan näkymä voi hakea rivinäkymiä listankohtia varten, jolloin tehtävälistaan voi laittaa tekstinpätkien lisäksi vaikka sähköpostiviestejä.

Järjestelmä tietää, mitä kohdetta näkymät näyttävät, jolloin sen on mahdollista toteuttaa kohteiden suoramanipulointi ilman näkymien apua. Kontekstivalikossa on kohteelle sopivia toimintoja eri tasoilla (tämä lista, tämä

alkio, tämä teksti). Kohteella voi olla toiminto, joka suoritetaan raahaamalla toinen kohde päälle.

Käyttöliittymässä ei ole ikkunoita tai muuta samanaikaisten sovellusten käsitettä. Ajankohtaisia kohteita voi pitää muistilistalla. Tiedostonvalintaikkunat ja muut kyselyt on korvattu *käyttöliittymäjatkeilla*, jotka ovat osittain täytettyjä toimintoja, joiden toteuttamiseen voi palata myöhemmin.

Sovellus tällaisessa järjestelmässä on kokonaisuus näkymiä, palveluita, valmiita kohteita ja käyttöliittymäjatkeita. Sovellukset lomittuvat luonnostaan tarkasteltavan tiedon mukaan, kun käyttäjä yhdistelee useammantyyppisiä kohteita.

6 Yhteenveto

Järjestelmää voidaan tarkastella matalasta tasosta ja käyttöliittymästä riippumatta sen sovelluksia integroivien ominaisuuksien perusteella. Sovelluslusta tarjoaa valmiita toimintoja sovelluksia varten, mutta alustan laadullisten ominaisuuksien kannalta on oleellista, että sen ominaisuudet vaikuttavat sovellusten integroinnissa kokonaisjärjestelmään. Yhteisestä toimintaperustasta riippuu, millä tasolla riippumattomat sovellukset toimivat yhdessä.

Vallitsevissa käyttöjärjestelmissä keskeisiä käsitteitä ovat prosessi ja tiedosto. RDF-kantaa tiedostojen sijaan käyttävässä Haystack-järjestelmässä useat riippumattomat sovellukset voivat käsitellä samaa tietoa yhtä aikaa ongelmitta. Edelleen sovellukset eivät ole selvästi erillisiä, kun käyttäjä yhdistelee RDF-kannassa erityyppisiä kohteita.

Jatkossa voisi kartoittaa sovelluslustoja kattavammin ja pyrkiä vertailemaan niitä erilaisten sovellustyyppien kannalta – toiselle alustalle sopiva sovellustyyppi on ehkä sopimaton tai jopa mahdoton toisella alustalla. Samalla voisi selvittää, mihin asti nykyisten alustojen vähittäinen kehitys voi yltää.

Viitteet

- [1] Edsger W. Dijkstra, *The Structure of the "THE"-Multiprogramming System*, Commun. ACM, 11(5), s. 341–346, 1968.
- [2] Dennis Quan, David Huynh ja David R. Karger, *Haystack: A Platform for Authoring End User Semantic Web Applications*, teoksessa *The Semantic Web – ISWC 2003*, s. 738–753, Springer-Verlag, Heidelberg, 2003.
- [3] David Garlan, *Software architecture: a roadmap*, teoksessa *The Future of Software Engineering*, ACM Press, 2000.