L U K U 1

Ohjelmoinnin valmistelut

Tässä luvussa

Tässä luvussa opit kuinka suunnittelet Microsoft Visual C++ -sovelluksen hyödyntäen Microsoftin suunnittelumenetelmiä. Tutustut asioihin, jotka sinun tulisi huomioida tehdessäsi suunnittelua. Saat myös yleiskuvan Visual C++ -kehitystyökaluista asennuksen kuvauksen yhteydessä.

Ennen kuin aloitat

Tämän luvun lopussa olevan laboratorio 1:n suorittaminen edellyttää, että olet asentanut Microsoft SQL Server 7.0:n ja ajanut STUpload-tietokannan muodostamiseen tarvittavat skriptit edellisessä *Tästä kirjasta* -luvussa kuvatulla tavalla.

Oppitunti 1: Ohjelmiston suunnittelu Microsoft Solutions Framework -menetelmällä

Suuriin monimutkaisiin ohjelmistoprojekteihin sisältyy aina riskejä. Suurista tietojärjestelmäprojekteista tehdyt tutkimukset osoittavat, että merkittävä osa ohjelmistoprojekteista epäonnistuu. Syitä epäonnistumisiin ovat esimerkiksi:

- Jatkuvasti muuttuvat vaatimukset
- Liian väljät tai keskeneräiset määrittelyt
- Ohjelmoinnin huono laatu
- Liian laaja tehtäväkenttä
- Henkilöstöön liittyvät ongelmat
- Huono prosessin hallinta
- Epäselvät päämäärät

Ratkaistakseen näitä ohjelmistotuotannon ongelmia Microsoft Consulting Services -palvelu kehitti Microsoft Solutions Framework (MSF) -menetelmän. MSF perustuu Microsoftin projektiryhmien, teknisten yhteistyökumppaneiden ja suurimpien asiakkaiden parhaisiin käytäntöihin. Voit tehostaa omia ohjelmistoprojektejasi käyttämällä MSF-menetelmää ja hyödyntää sen käsitteistöä suunnitellessasi projekteja, joissa työskentelee useampia ohjelmoijia.

Tällä oppitunnilla kerrotaan MSF:n ominaisuuksista ja siitä, kuinka ne liittyvät sovellusten toteuttamiseen Visual C++:lla.

Tämän oppitunnin jälkeen:

- Tunnet MSF:n osat.
- Tunnet MFS:n roolin suunnittelu- ja kehitystyössä.

Oppitunnin arvioitu kesto: 30 minuuttia

Yleiskatsaus MSF:ään

MSF koostuu malleista, periaatteista ja käytänöistä, jotka auttavat organisaa-tiota tehokkaammin kehittämään ja hyödyntämään teknologiaa liiketoimintaansa liittyvien ongelmien ratkaisemisessa. MSF helpottaa edistymisen seurantaa ja ohjaa projektia tavalla, joka on riittävän joustava nykyaikaisen liiketoiminnan vaatimuksiin. MSF:n ytimen muodostavat seuraavat kuusi päämallia:

- MSF Enterprise Architecture Model
- MSF Risk Management Model

- MSF Team Model for Application Development
- MSF Process Model for Application Development
- MSF Application Model
- MSF Design Process Model

Kuvaamme aluksi jokaisen mallin lyhyesti ja palaamme niiden yksityiskohtiin tarkemmin myöhemmin tämän oppitunnin aikana.

MSF Enterprise Architecture Model

MSF Enterprise Architecture Model antaa yhdenmukaiset suuntaviivat hankkeen rakenteen nopeaan määrittelyyn vaiheittain tapahtuvan toimituksen avulla. Tämä malli sovittaa yhteen tietojenkäsittelyn ja liiketoiminnan vaatimukset lähtien neljästä eri näkökulmasta, jotka ovat: liiketoiminta, sovellus, tieto ja järjestelmä. Mallin käyttäminen auttaa lyhentämään hankkeen rakenteen määrittelyyn kuluvaa aikaa.

MSF Risk Management Model

MSF Risk Management Model mallin avulla voidaan projektin riskejä hallita järjestelmällisesti ja ennakoiden. Malli auttaa määrittelemään järjestelmällisesti ja ennakoiden mahdollisten riskien laajuuden, sekä löytämään ne riskit, jotka on erityisesti huomioitava ja auttaa sen jälkeen määrittelemään tavat, joilla nämä riskit käsitellään. Tämän mallin periaatteiden ja käytäntöjen hyödyntäminen auttaa kehitysryhmää keskittymään olennaiseen, tekemään oikeita päätöksiä ja olemaan valmiina tulevaan.

MSF Team Model for Application Development

MSF Team Model of Application Development -malli on joustava tapa kehitysryhmän organisoimiseksi. Se korostaa selkeästi roolien, vastuun ja päämäärien merkitystä ja lisää työryhmän jäsenten vastuunottoa tasa-arvoisen työryhmämallinsa avulla. Mallin joustavuus mahdollistaa sen sovittamisen kunkin kehitysprojektin tarpeita vastaavaksi. Tämän mallin periaatteiden ja käytäntöjen hyödyntäminen auttaa aikaansaamaan paremmin tehtävään sitoutuneen, tehokkaamman, joustavamman ja menestyksekkään työryhmän.

MSF Process Model for Application Development

MSF Process Model for Application Developement -malli perustuu välitavoitteisiin ja toistoon antaen ohjausta koko projektin elinkaaren ajan. Tässä mallissa kuvataan projektin vaiheet, välitavoitteet, toiminnot ja tuotteet, sekä näiden elementtien suhteet MSF Team Model of Application Developement -mallin työryhmän jäsenten rooleihin. Tämän mallin käyttäminen helpottaa projektinhallintaa, vähentää riskejä, parantaa laatua ja lyhentää toimitusaikoja.

MSF Application Model

MSF Application Model -sovellusmalli on looginen, monikerroksinen ja palveluihin perustuva lähestymistapa ohjelmistojen suunnitteluun ja kehittämiseen. Sovelluksen jakaminen käyttäjä-, liiketoiminta- ja tietopalveluihin mahdollistaa rinnakkaisen kehitystyön, teknologian paremman hyödyntämisen, helpottaa ylläpitoa ja tukea, sekä mahdollistaa toimituksissa suuren joustavuuden, koska palvelut, joista sovellus koostuu, voivat sijaita missä tahansa työasemassa, palvelimessa tai päätelaitteissa ympäri maailmaa.

MSF Design Process Model

MSF Design Process Model -malli on kolmivaiheinen, katkeamattomasti käyttäjäkeskeinen malli, joka mahdollistaa tehokkaan ja joustavan rinnakkaisen ja iteratiivisen suunnittelun. Mallin kolme vaihetta – käsitteellinen, looginen ja fyysinen – tarjoavat kolme erilaista näkökulmaa kolmelle eri ryhmälle – käyttäjille, kehitysryhmälle ja ohjelmoijille. Näiden kolmen vaiheen kautta jokainen sovelluksen ominaisuus voidaan johtaa suoraan käyttäjämäärittelyistä. Tämän mallin käyttäminen varmistaa sen, että sovellusta ei tehdä vain tekniikan takia, vaan suoraan käyttäjien ja liike-elämän tarpeita varten.

Presentation of MSF in This Book

Tässä luvussa esitellään joitakin MSF:n peruskäsitteitä, jotta saataisiin perusteet sovelluksen suunnittelu- ja toteutusvaiheiden käsittelyä varten. Keskitymme sovelluskehitysprosessin kannalta keskeisimpiin solution frameworkin piirteisiin – erityisesti Team Model of Application Developement, Process Model for Application Developement, Application Model ja Design Process Model -malleihin. MSF on käsitelty tarkemmin kirjassa *Analyzing Requirements and Defining Solution Architectures: MCSD Training Kit for Exam 70-100* (Microsoft Press, 1999).

MSF Development Team Model -mallin käyttäminen

Sen sijaan, että MSF olisi tiukka määrittely, se on ennemminkin runko, joka voidaan sovittaa kunkin organisaation tarpeisiin. Team model on yksi osa tätä kehystä. Mallissa kuvataan, kuinka työryhmien tulisi organisoida toimintansa ja mitä periaatteita niiden tulisi noudattaa ollakseen menestyksekkäitä.

MSF Development Team Model on luonteeltaan hyvin yksityiskohtainen, mutta frameworkin osana se tulisi nähdä ennemmmin lähtökohtana. Kehitysryhmät voivat toteuttaa mallin osia eri tavoin riippuen projektin tavoitteista, ryhmän koosta ja ryhmän jäsenten taidoista.

Jotta projekti olisi onnistunut, täytyy tietyistä vastuista huolehtia ja tietyt päämäärät saavuttaa. Nämä vastuut ja tavoitteet auttavat ohjaamaan jokaisen ryhmän jäsenen työskentelyä. Tärkeimpiä päämääriä ja vastuita ovat:

- **Asiakastyytyväisyys** Projektin on täytettävä asiakkaiden ja käyttäjien tarpeet ollakseen menestyksekäs. On mahdollista, että aikataulussa ja alkuperäisen budjetin mukaan toimitettu tuote on epäonnistunut, jos se ei vastaa asiakkaan tarpeita.
- **Toiminta projektille asetetuissa rajoissa** Useimpien projektien onnistumista mitataan sen mukaan, onko pysytty aikataulussa ja budjetissa.
- **Käyttäjävaatimusten mukainen tuote** Toiminnallinen määrittely kertoo yksityiskohtaisesti asiakkaan tuotteelle asettamat vaatimukset. Määrittely on sopimus asiakkaan ja työryhmän välillä siitä, että ”me teemme sen, minkä sanomme tekevämme”.
- **Julkistus vasta kaikkien oleelisten seikkojen selvittämisen ja käsittelemisen jälkeen** Kaikki ohjelmistot ovat toimitusvaiheessa epätäydellisiä. Työryhmän tehtävänä on varmistaa, että kaikki epäkohdat on löydetty ja käsitelty ennen kuin tuote julkistetaan. Epäkohtien käsittely voi tarkoittaa mitä tahansa ongelman korjaamisesta ongelman kiertämisessä neuvovan ohjeistuksen tekemiseen. On parempi toimittaa epätäydellinen tuote, jonka puutteet on huomioitu ohjeistuksessa, kuin toimittaa tuote, jossa on tuntemattomia vikoja, jotka yllättävät kehitysryhmän ja myöhemmin asiakkaan.
- **Loppukäyttäjän työn tehostuminen** Menestyäkseen tuotteen täytyy helpottaa käyttäjiensä työtä. Ei ole järkevää tuottaa laajaa ja monipuolista sovellusta, jota ei kuitenkaan voida käyttää.
- **Sujuva jakelu ja jatkuva ylläpito** Jakelun tehokkuus vaikuttaa suoraan siihen, kuinka laadukkaana tuotetta pidetään. Esimerkiksi viallinen asennusohjelma aiheuttaa epäilyksen siitä, että koko ohjelmisto on vastaavasti viallinen. Ei myöskään riitä, että kehittäjät vain toimittavat tuotteet. Toimitusten on tapahduttava tyylikkäästi, minkä jälkeen on annettava tukea ja jatkettava tuotteen ylläpitoa.

MSF Team Model of Application Development huomioi nämä avainkohdat jakamalla tehtävät kuuden työryhmän toimenkuvan kesken: tuotteen hallinta (Product Management), ohjelman hallinta (Program management), tuotekehitys (Development), testaus (Testing), käyttäjäkoulutus (User Education) ja logistiikanhallinta (Logistics Management). Kunkin tavoitteen saavuttaminen on oma taiteen lajinsa, joten jokaiseen toimenkuvaan kuuluu huolehtiminen yhdestä tavoitteesta. Henkilöillä, joille toimenkuvat kuuluvat, täytyy olla näkemystä ja taitoa kunkin tavoitteen saavuttamiseksi.

Taulukossa 1.1 on kuvattu kuuden toimenkuvan ja kuuden tavoitteen väliset yhteydet. Koska jokainen näistä tavoitteista on lopputuloksen kannalta yhtä tärkeä, kaikilla niistä vastaavilla henkilöillä on yhtäläinen sananvalta päätöksenteossa.

Projektilla ei ole päällikköä, vaan ryhmä tietää, mitä on tehtävä ja sillä on kunnolliset välineet sen tekemiseen.

Taulukko 1.1 Päämäärät ja niistä vastaavat ryhmän jäsenet

Päämäärä	Ryhmän jäsen
Asiakastyytyväisyys	Product Management
Toiminta projektille asetetuissa rajoissa	Program Management
Käyttäjävaatimusten mukainen tuote	Development
Julkistus vasta kaikkien oleellisten seikkojen selvittämisen ja käsittelemisen jälkeen	Testing
Loppukäyttäjän työn tehostuminen	User Education
Sujuva jakelu ja jatkuva ylläpito	Logistics Management

MSF Development Process Model -mallin käyttäminen

MSF Process Model of Application Developement -malli sisältää projektinsuunnittelun tärkeimmät tehtävät. Siinä määritellään mitä, asioita pitää tehdä ja milloin ne on tehtävä. Malli huomioi lisäksi kaksi muuta tärkeää näkökohtaa: se on läheisessä yhteydessä MSF Development Team -malliin ja antaa näin organisaatiolle niiden yhteiskäytöstä koituvat edut, sekä Development Process Model mallin periaatteet ja käytännöt, joita ovat:

- Vaiheittainen julkistus.
- Ajantasaisen dokumetaation luominen.
- Epävarman tulevaisuuden aikatauluttaminen.
- Kompromissien hallinta.
- Riskien hallinta.
- Ajatus kiinteästä julkaisupäivästä.
- Suuren projektin jakaminen hallittaviin osiin.
- Edistymisen seuraaminen päivittäin.
- Alhaalta ylös -arvioinnin käyttäminen.

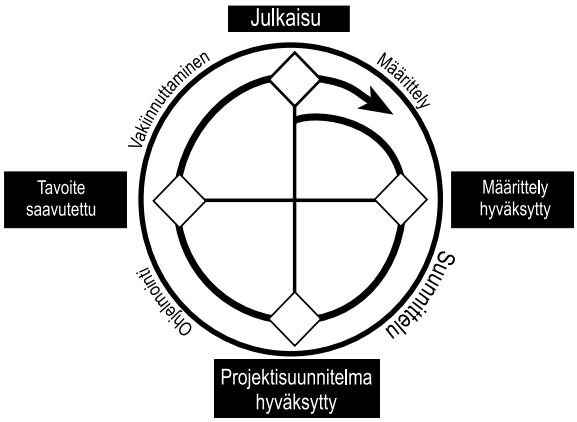
Perinteiset ohjelmistoprojektimallit, kuten vesiputousmalli ja spiraalimalli, eivät useinkaan vastaa nykyaikaisen sovelluskehityksen vaatimuksia.

Vesiputousmallissa projekti etenee vaiheittain määrittelystä testaukseen. Tämä malli määrittää välitavoitteet ja käyttää niitä siirtymä- ja arviointipisteinä. Malli toimii hyvin, jos projektin vaatimukset voidaan määritellä tarkasti jo alkuvaiheessa, mutta sopii huonosti monimutkaisiin projekteihin, joissa vaatimukset saattavat muuttua projektin aikana. Lisäksi tämän mallin käyttäjät luottavat vahvasti dokumentaation määrään ja yhteen tarkastukseen vaihetta kohden. Nämä kaksi vesiputousmallille ominaista piirrettä johtavat yleensä laajaan ”analyysihalvaukseen” ja kehittäjien, asiakkaan ja käyttäjien välien huononemiseen.

Spiraalimallissa sovellus kehitetään toistamalla kehitysprosessi useita kertoja. Ensimmäisillä kierroksilla spiraalimalli tuottaa tarkennuksia määrittelyynsä ja keskivaiheen ja lopun toistot lisäävät toimintoja ja toiminnallisuutta. Spiraalimallin tavoitteena on löytää projektin riskit varhaisessa vaiheessa ja käsitellä ne ensimmäisissä tuotteen julkaisuissa.

Iteratiivisesta luonteestaan johtuen spiraalimalli mahdollistaa kehitystyön tekemisen koko prosessin ajan ja toivottavasti parantaa näin tuotteen laatua. Toimiakseen tehokkaasti spiraalimalli edellyttää merkittävää määrää prosessointia ja kehittynyttä dokumentoinnin automatisointia. Käytännössä voi käydä niin, että asiakkaat ja käyttäjät pitävät järjestelmää epävakaana, koska muutokset tapahtu-vat nopeammin kuin he pystyvät niitä seuraamaan. Lisäksi monilta tällaisilta projekteilta puuttuu määrätty päätepiste, joten toistot jatkuvat ikuisesti, jos taloudelliset seikat eivät sitä estä.

Kuten kuvassa 1.1 on esitetty MSF-sovelluskehitysmalli yhdistää näiden kahden mallin parhaat ominaisuudet, hyödyntämällä vesiputousmallin välitavoitteet ja spiraalimallin toistuvat kehitysvaiheet.



Kuva 1.1 MSF Development Process Model -malli

MSF Development Process Model -mallilla on kolme tärkeää piirrettä:

- Vaiheittain etenevä prosessi (kuvassa 1.1 esitetyn kuvion neljännekset).
- Välitavoitteisiin perustuva eteneminen (vaiheita erottavat nelikulmiot).
- toistuva prosessointi (nuoli, joka osoittaa takaisin ensimmäiseen vaiheeseen).

Vaikka kuvassa 1.1 kaikki neljä vaihetta on kuvattu yhtä suuriksi, vaiheiden läpivieminen ei välttämättä vaadi ajallisesti samaa aikaa. Erilaiset toiminta-ympäristöt edellyttävät ajan ja muiden resurssien suhteuttamista eri tavoin.

Vaiheittain etenevä prosessi

MSF Development Process Model -malli koostuu neljästä toisiinsa liittyvästä vaiheesta. Jokainen näistä vaiheista edellyttää tiettyjen suoritteiden valmistumista ennen kuin voidaan siirtyä projektin seuraavaan vaiheeseen. Neljä vaihetta ja niiden tärkeimmät tehtävät ovat:

- Yleiskuvan muodostaminen, niin että yhteinen näkemys tuotteesta syntyy.
- Suunnittelu, jonka tuloksena täytyy saada aikaan tarkka projektisuunnitelma ja sovelluksen arkkitehtuuri.
- Kehitysvaihe, jonka tehtävänä on tuottaa hyvin rakennettu ja valmis tuote.
- Vakiinnuttaminen, jonka tuloksena syntyy vakaa ja jakeluun valmis tuote.

Välitavoitteisiin perustuva eteneminen

MSF Development Process Model malli perustuu välitavoitteisiin, joiden tarkoituksena on ennemmin uudelleen tarkastelu- ja tahdistuspisteinä toimiminen kuin sovelluksen tai määrittelyjen kiinnittäminen. Ne antavat kehitysryhmälle mahdollisuuden tarkastella projektin etenemistä ja tehdä muutoksia suunnitelmiin, esimerkiksi sovittamalla suunnitelmia paremmin asiakkaan muuttuneita tarpeita vastaaviksi tai reagoimalla riskeihin, jotka voisivat aiheuttaa ongelmia prosessin aikana.

MSF Development Process Model -mallissa käytetään kahden tyyppisiä merkkipaaluja: päätavoitteita (major milestones) ja alitavoitteita (interim milestones). Jokaisen merkkipaalun, niin pää- kuin alitavoitteen, saavuttaminen edellyttää yhtä tai useampaa tuotosta (deliverable). Tuotos on fyysinen todiste siitä, että ryhmä on saavuttanut tavoitteen.

Päätavoitteet

Jokainen kehitysprosessin vaihe päättyy julkiseen päätavoitteeseen. *Julkinen* tarkoittaa sitä, että päätavoite ja sen vaatimat tuotokset ovat myös projektiryhmän ulkopuolisten tahojen nähtävissä. Tällaisia ulkopuolisia tahoja ovat esimerkiksi asiakkaat ja tukihenkilöstö.

Päätavoite on piste, jolloin kaikki työryhmän jäsenet tahdistavat tuotoksensa. Lisäksi kaikille projektiin liittyville tahoille, kuten asiakkaille, käyttäjille, tukihenkilöille, jakelusta vastaaville (kaupallista sovellusta tehtäessä) ja kaikille projektin sidosryhmille, pitäisi kertoa projektin etenemisestä.

Yksi päätavoitteiden tärkeimmistä tehtävistä on antaa vaihe vaiheelta kuva projektin elinkykyisyydestä. Tarkasteltuaan nykyisen vaiheen tuotoksia asiakas ja työryhmä tekevät yhdessä päätöksen siitä, jatketaanko seuraavaan vaiheeseen vai ei. Näin ollen päätavoitteet huolehtivat projektin etenemisestä vaiheesta toiseen.

Alitavoitteet

Jokainen MSF-kehitysprosessin vaihe koostuu joukosta alitavoitteita, jotka kuten päätavoitteetkin tarjoavat mahdollisuuden uudelleen tarkasteluun ja työskentelyn tahdistamiseen. Tavoitteena ei myöskään alitavoitteiden kohdalla ole kiinnittää suunnitelmia tai kehitystyötä. Päätavoitteista poiketen alitavoitteet ovat julkisia vain *sisäisesti*, eli niiden tulokset ovat vain työryhmän jäsenten tarkasteltavissa.

Alitavoitteet osoittavat etenemistä ja jakavat suuret tehtäväkokonaisuudet helpommin hallittaviin osiin.

Vaiheittainen prosessi

Vaiheittaisuus tarkoittaa MSF Development Process Model mallissa sitä, että kehitysprosessia on tarkoitus toistaa koko tuotteen elinkaaren ajan. Jokainen valmistunut kehitysprosessin kierros tarjoaa uusia toimintoja ja ominaisuuksia tyydyttääkseen asiakkaiden muuttuvat tarpeet.

MSF Application Model -mallin käyttö

Application Model -sovellusmalli on käsitteellinen näkökulma sovellukseen. Se esittelee määritelmät, säännöt ja suhteet, joista sovellus rakentuu. Lisäksi sovellusmalli tarjoaa pohjan ajatusten vaihdolle loogisen (ennemmin kuin fyysisen) suunnittelun aikana. Application Model kuvaa sovelluksen rakenteen, ei sen toteutusta.

MSF Application Model -malli tarjoa monitasoisen palveluihin perustuvan lähestymistavan sovelluksen suunnitteluun ja toteuttamiseen. Ratkaisumallissa sovelluksen ajatellaan koostuvan loogisella tasolla yhteistyössä toimivista, jaetuista ja uudelleen käytettävistä palveluista, jotka tukevat kaupallista ratkaisua. Sovelluksen palvelut ovat loogisia yksiköitä, jotka sisältävät toiminnot jonkin operaation, funktion tai muunnoksen toteuttamiseen. Näitä palveluita tulisi käyttää

käyttöliittymämäärittelyssä kuvatun julkistetun käyttöliittymän kautta ja käsiteltävä tieto pitäisi sijoittaa palvelun tarjoavan osan sijasta asiakasosaan. Palveluiden tulisi liittyä suoraan käyttäjätoimintoihin.

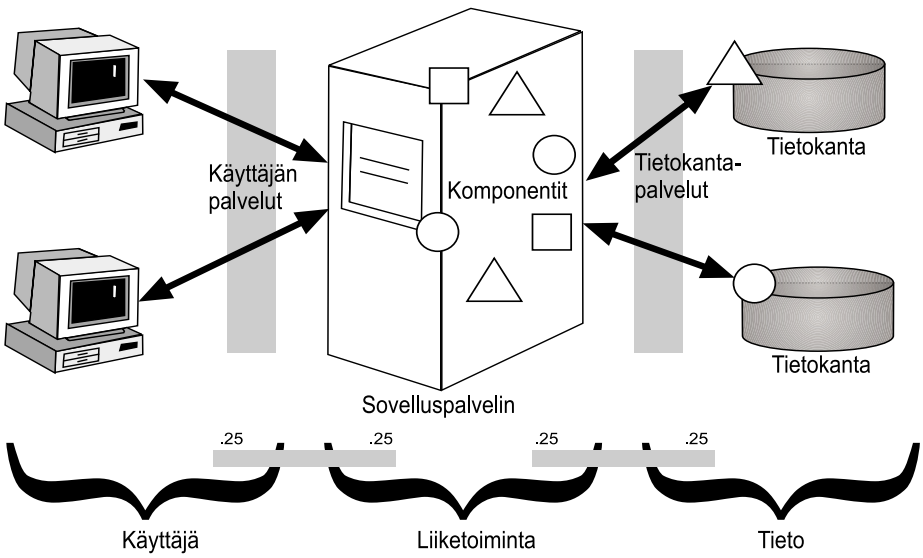
MSF Application Model -malli on Microsoftin suositus jaettujen, monitasoisten asiakas-palvelin-sovellusten toteuttamiseksi. Mallin tavoitteet ovat:

- Tarjota yhtenäinen lähestymistapa asiakas-palvelin-sovellusten suunnitteluun ja kehitykseen.
- Tarjota standardi joukko määrittelyjä monitasoisen sovelluksen loogisen rakenteen määrittelyyn.
- Helpottaa komponentteihin perustuvan tekniikan käyttämistä jaettujen, monitasoisten sovellusten toteuttamisessa, niin että sovellukset ovat riittävän joustavia, skaalautuvia ja ylläpidettäviä, jotta ne täyttäisivät kriittisille yritysten laajuisille sovelluksille asetut vaatimukset.
- Siirtyminen yksittäisistä mammuttiohjelmista pienten, yhtenäisistä osista koottujen, tiettyä tehtävää varten rakennettujen sovellusten yhteiskäyttöön.
- Kuvata tapa, jolla sovelluksia kehittävän organisaation taito ja muut resurssit voidaan jakaa samanaikaisesti useiden projektien käyttöön.
- Määritellä runko työryhmien organisaatiolle, esitellä rinnakkaisesti etenevä kehitysprosessi ja määritellä tapa, jolla prosessissa tarvittava osaaminen määritellään.

MSF Application Model -mallin sovellukset esitetään kolmen palvelutyypin avulla: käyttäjä-, liiketoiminta- ja datapalvelut. Palvelu on sovelluksen yksikkö, joka huolehtii operaatioista, toiminnoista ja muunnoksista, jotka on määrätty sen tehtäväksi. Palvelut voivat toteuttaa liiketoiminnan sääntöjä, suorittaa laskutoimituksia ja muunnoksia tai tarjota ominaisuuksia tietojen syöttämistä, tutkimista ja muuttamista varten. Nämä palvelut mahdollistavat rinnakkain tapahtuvan kehitystyön, tehostavat tekniikan hyödyntämistä, helpottavat ylläpitoa ja parantavat joustavuutta sovelluksen palveluiden jakamisessa. Lisäksi palvelut voidaan sijoittaa mihin tahansa ympäristöön, yksittäisestä työasemasta palvelimiin ja asiakaskoneisiin, jotka sijaitsevat eri puolilla maailmaa.

Sovellusarkkitehtuurit

Sovelluksen arkkitehtuuri on käsitteellinen näkökulma sovelluksen rakenteeseen. Kuten kuvasta 1.2 nähdään, jokaisessa sovelluksessa on kolme päätasoa: käyttäjäkerros, liiketomintakerros ja datakerros. Jokaiseen sovellukseen sisältyy lisäksi kolmenlaista ohjelmakoodia: esittämiseen, liiketoiminnan sääntöjen käsittelyyn sekä tietojen käsittelyyn ja tallentamiseen liittyvää koodia. Sovellusarkkitehtuurit eroavat toisistaan siinä, kuinka näistä osista muodostetaan tietty toimiva ohjelma.



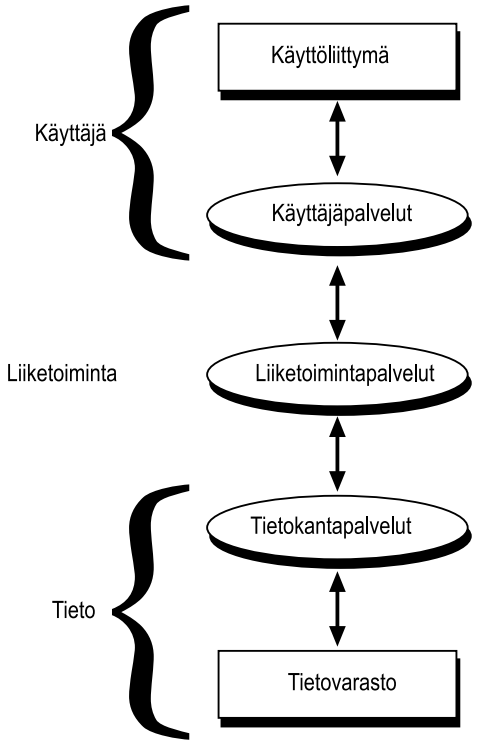
Kuva 1.2 Application Model ja sen sovelluskerrokset

Monien nykyisten sovellusten arkkitehtuuri on kaksikerroksinen (two-tier architecture). Arkkitehtuuri tunnetaan myös nimellä *asiakas/palvelinarkkitehtuuri* (client/server architecture). Kaksikerroksisessa arkkitehtuurissa asiakasosa sisältää tiedon käsittelyyn ja esittämiseen liittyvät osat. Tieto varastoidaan keskitetysti palvelin koneille. Asiakas muodostaa palvelimelle yhteyden, jota se usein tarvitsee sovelluksen koko käytössäoloajan.

Asiakas/palvelin-sovellukset toimivat hyvin säädellyssä ympäristössä, jossa käyt-täjien määrä voidaan arvioida ennalta ja hallita, ja jossa resurssit voidaan jakaa sen mukaisesti. Kun käyttäjien määrää ei tunneta tai se on hyvin suuri, asiakas/palvelinarkkitehtuuri kuitenkin romahtaa. Koska jokainen asiakas kytkeytyy suoraan palvelimeen, yhtäaikaisten yhteyksien määrä rajoittaa skaalautuvuutta. Uudelleenkäytön mahdollisuudet ovat rajoitettuja, koska asiakkaat on sidottu tietokantaformaatteihin. Jokainen asiakasosa sisältää tietojen käsittelyssä tarvit-tavan logiikan ja sovelluksesta tulee näin suhteellisen suuri. (Tällaisesta asiak-kaasta käytetään joskus nimitystä fat client.) Jos tietojenkäsittelyyn liittyvää logiikkaa täytyy joskus muuttaa, uusi sovellus täytyy toimittaa kaikkiin asiakas- tietokoneisiin.

Pienehkö parannus saadaan aikaan siirtämällä osa tietojenkäsittelystä palvelimen tehtäväksi, esimerkiksi käyttämällä SQL Serveriin talletettuja toimintoja. Tällaista arkkitehtuuria kutustaan joskus kaksi ja puolitasoiseksi ("two-and-a-half tier"). Tähän malliin perustuvat sovellukset ovat tietyllä tavalla paremmin skaalautuvia, mutta eivät tarpeeksi skaalautuvia laajasti levitettyjen sovellusten tarpeita ajatellen. Lisäksi uudelleenkäytön mahdollisuudet ovat rajoitetut.

Skaalautuvuutta ja uudelleenkäytettävyyttä voidaan merkittävästi parantaa ottamalla sovelluksen arkkitehtuurissa käyttöön kolmas kerros. *Monikerroksisessa* arkkitehtuurissa (multi-layer architecture), joka tunnetaan myös *N-kerroksisena* arkkitehtuurina (N-layer architecture), käyttäjä-, liiketoiminta- ja datatasot on loogisesti erotettu toisistaan, kuten kuvassa 1.3 on esitetty.



Kuva 1.3 Monitasoinen sovellusarkkitehtuuri

Nämä kolme kerrosta suorittavat seuraavat toiminnot:

- **Käyttäjäkerros** (User layer) esittää tiedot käyttäjälle ja antaa mahdollisesti käyttäjän muuttaa tietoja. Henkilökohtaisten tietokoneiden sovellusten käyttöliittymät (UI) voidaan jakaa kahteen pääryhmään, alkuperäisiin ja Web-pohjaisiin. Alkuperäinen käyttöliittymä käyttää alla olevan käyttöjärjestelmän palveluja. Esimerkiksi Microsoft Windows -ympäristössä alkuperäinen käyttö-liittymä käyttää Win32 APIa ja Windows-kontrolleja. Web-pohjainen käyttö-liittymä perustuu HTML:ään ja Extentensible Markup Language (XML) kie- leen, jota voidaan käyttäää Web-selaimella riippumatta käyttöjärjestelmästä.
- **Liiketoimintakerros** (Business layer) huolehtii liiketoiminta- ja tietojenkäsittelysääntöjen käyttöönottamisesta. Esityskerros käyttää liiketoimintakerroksen palveluita. Liiketoimintakerros ei kuitenkaan ole sidottu mihin-

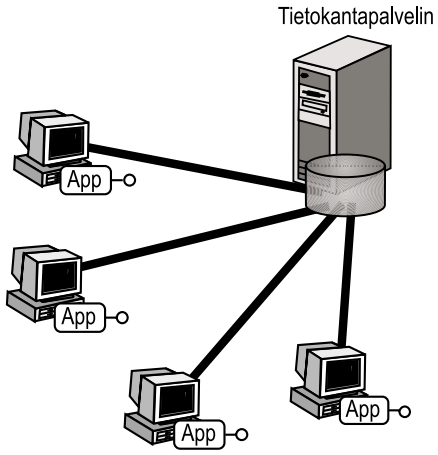
kään tiettyyn asiakkaaseen; sen palvelut ovat kaikkien sovellusten käytettävis-sä. Liiketoimintasäännöt voivat olla liiketoiminta-algoritmeja, liiketoiminta- käytäntöjä, lain vaatimia käytäntöjä ja niin edelleen — esimerkiksi "Käyttäjät saavat 10 prosentin alennuksen ilmoituksista, jotka on jätetty ennen tiistai-il-taa" tai "Kaikista Viroon viedyistä tilauksista on kerättävä kuuden prosentin myyntivero". Tietojenkäsittelysäännöt auttavat varmistamaan tietojen oi-keellisuuden ja toisinaan tallennetun tiedon eheyden — esimerkiksi "Tilauk-sessa täytyy olla ainakin yksi yksilöivä tieto" tai "Rahaa ei saa kadota eikä tulla lisää siirrettäessä maksuja tililtä toiselle". Liiketoimintasäännöt toteute-taan normaalisti erillisinä koodimoduuleina, jotka on yleensä sijoitettu kes-kitettyyn paikkaan, niin että useat sovellukset voivat käyttää niitä.

- **Datakerros** (Data layer) Liiketoimintakerros ei tiedä, kuinka sen käsittelemä tieto on varastoitu. Sen sijaan se luottaa tiedonsaantipalveluihin, jotka suorittavat varsinaisen tiedon varastointiin ja hakemiseen liittyvän työn. Tiedonsaantipalvelut on myöskin usein toteutettu erillisinä moduuleina, jotka kapseloivat allaolevaan tietovarastoon liittyvän tietouden. Jos tietovarasto siir-tyy tai sen formaatti muuttuu, tarvitsee vain päivittää tiedonsaantipalvelu. Jokainen tiedonsaantipalvelu on tyypillisesti vastuussa yhden tietojoukon datasta — esimerkiksi relaatiotaulukoihin sijoitetusta tiedosta tai Microsoft Data Objects (ADO) ja OLE DB -tekniikasta. N-kerroksisessa suunnittelussa tieto-varasto on yksinkertaisesti DBMS — järjestelmä, joka tarvitaan tietojen hake-miseen taulukoista ja tiedon käsittelyn optimointiin, kuten tietokannan in- deksointiin. Tietovarastoja ovat esimerkiksi SQL Server, Microsoft Exchange Server, Microsoft Database Engine (MSDE), Microsoft FoxPro, Microsoft Access ja Microsoft Jet.

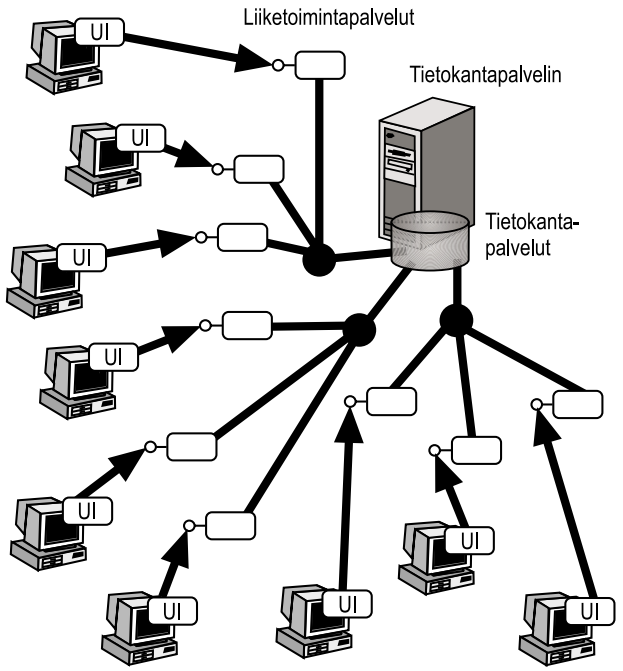
On ehkä hieman hämmentävää puhua loogisesta arkkitehtuurista osana fyysistä mallia. On tärkeää ymmärtää, että looginen arkkitehtuuri pitää sisällään suuren osan fyysisestä arkkitehtuurista ja toteutuksesta, joka määrää, minne palvelut sijoitetaan. Toisin sanoen sovellukset koostuvat palvelujen käyttäjien ja tarjoajien verkostoista. Palvelut ovat pelkästään sovelluksen loogisia yksiköitä, jotka tarjoavat tarkasti määriteltyjä palveluita.

Termit *monikerroksinen* ja *n-kerroksinen* eivät viittaa erillisiin tietokoneisiin. N-kerroksinen sovellusarkkitehtuuri parantaa sovellusten skaalautuvuutta. Jotta voitaisiin luoda hyvin skaalautuvia sovelluksia, resurssien kuten tietokantayhteyksien tulee olla jaettuja. Sen sijaan, että jokainen sovellus olisi resursseja kuluttaen yhteydessä suoraan palvelimeen, asiakassovellukset kommunikoivat liiketoimintapalveluiden kanssa. Yksi liiketoimintapalvelun ilmentymä voi huolehtia useista asiakkaista, vähentäen resurssien kulutusta ja parantaen skaalautuvuutta, kuten seuraavan sivun kuvista 1.4 ja 1.5 nähdään. Koska liiketoimintapalvelut eivät suoranaisesti käsittele tietoa, näitä palveluita voidaan helposti kopioida tukemaan vielä useampia asiakkaita. Palvelut voidaan usein suunnitella ja toteuttaa erillään kaikista asiakassovelluksista, mikä lisää joustavuutta ja mahdollistaa uudelleenkäytön monissa sovelluksissa. Kätkemällä sovelluksen lo-

giikan hyvin määritellyn käyttöliittymän taa ohjelmoija voi luoda uudelleenkäytettävistä palveluista rungon, joka voidaan helposti järjestää uudelleen uusia sovelluksia kehitettäessä. Lisäksi toiminnot voidaan helposti päivittää vastaamaan liike-elämän muuttuvia vaatimuksia, ilman vaikutuksia asiakassovelluksiin, jotka perustuvat tähän toimintoon. Tämä lähestymistapa vähentää hallinnoinnin ja jakelun kustannuksia vaatimusten muuttuessa.



Kuva 1.4 Asiakas/palvelinjärjestelmä



Kuva 1.5 N-tasoinen järjestelmä

Monikerroksinen sovellusarkkitehtuuri voi myös auttaa ohjelmoijaa nykyisten ja aikaisempien järjestelmien huomioimisessa. Ohjelmoijat voivat "kietaista" yhteyden vanhoihin järjestelmiin liiketoimintalogiikka-, tiedonsaanti- tai tietovarastopalveluihin. Asiakassovellusten tarvitsee huolehtia vain siitä, kuinka ne ovat yhteydessä liiketoimintalogiikkapalveluun, ei siitä kuinka olla yhteydessä kaikkiin eri järjestelmiin, joihin niiden toiminta perustuu. Jos allaoleva järjestelmä muuttuu tai korvataan toisella, täytyy vain välikerros uusia.

MSF Design Process Model -mallin käyttäminen

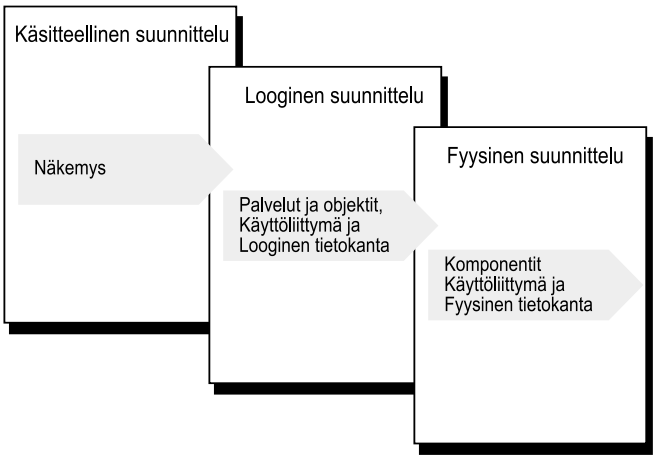
MSF Design Process -prosessi koostuu kolmesta helposti toisistaan erotettavasta osasta: käsiteosa (conseptual), looginen osa (logical) ja fyysinen osa (physical). Jokainen näistä muodostaa saman nimisen mallin: käsitemalli (Conseptual Design), looginen malli (Logical Design) ja fyysinen malli (Physical Design).

Jokainen prosessin osa lähestyy suunnittelutehtävää eri näkökulmasta ja määrittelee ratkaisun eri tavalla, kuten taulukossa 1.2 esitetään.

Taulukko 1.2 MSF:n suunnitteluprosessin osat

Suunnittelutehtävä	Näkökulma	Toiminta
Käsitteellinen	Tarkastelee ongelmaa käyttäjän ja liike-elämän kannalta.	Määrittää ratkaisun ongelmaan kuvauksina.
Looginen	Ratkaisua tarkastellaan kehitysryhmän näkökulmasta.	Määrittelee ratkaisun joukkona yhteistyössä toimivia palveluja.
Fyysinen	Tarkastelee ratkaisua ohjelmoijien näkökulmasta.	Määrittelee ratkaisun palveluina ja tekniikoina.

Edellisen osan tuloksia käytetään aina seuraavan osan lähtökohtana, kuten kuviossa 1.6 esitetään.



Kuva 1.6 Suunniteluprosessin kolmen itsenäisen vaiheen tuotokset

Osien tavoitteet ovat:

- **Käsitteellinen suunnittelu** Määritellään liiketoiminnan tarpeet ja selvitetään, mitä käyttäjät tekevät ja mitä vaatimuksia heillä on. Käsitteellinen suunnit-telu tuottaa kuvauksia, jotka heijastavat asiakkaiden, käyttäjien ja muiden asianosaisten vaatimuksia.
- **Looginen suunnittelu** Organisoi ratkaisun ja kommunikaation sen osien välillä. Loogisessa suunnittelussa muutetaan käsitteellisen suunnittelun tuottamat kuvaukset ratkaisun abstraktiksi malliksi. Kuvion 1.6 työnkulkua seuraten looginen suunnittelu muuttaa käsitteellisen suunnittelun kuvaukset objekteiksi ja palveluiksi, käyttöliittymäprototyypeiksi ja suunnitelmiksi tietokantojen loogisesta rakenteesta.
- **Fyysinen suunnittelu** Sovelluksen yksityiskohdat selvitetään soveltamalla tosielämän teknisiä rajoituksia, mukaan lukien toteutukseen ja tehokkuuteen liittyvät, loogisen suunnittelun tuloksiin. Loogisen suunnittelun lopputuloksia käytetään komponenttien, käyttöliittymämäärittelyjen ja tietokannan fyysiseen suunnitteluun.

MSF:n suunniteluprosessi etenee käsin kosketeltavista vaatimuksista (käyttötapaus), abstraktiin sovellussuunnitelmaan (Conseptual Design), edelleen rationalisoituun sovellussuunnitelmaan (Logical Design), sitten konkreettiseen sovellussuunnitelmaan (Physical Design) ja lopuksi käsin kosketeltavaksi sovellukseksi. Käsitteellinen suunnittelu on ensimmäinen käännös käyttötapauksista (käyt-täjän kieleltä), sovellus suunnitelmaksi (sovelluksenkehittäjien kielelle). Loogisen suunnittelun tuotos edustaa ohjelmoijan näkökulmaa ja toimii välittäjänä so- velluksen korkean tason käsitteistön — liike-elämän ja käyttäjän vaatimukset — ja yksityiskohtaisen sovellussuunnitelman välillä.

Käsitteellisessä suunnittelussa ei oteta kantaa tekniseen lähestymistapaan eikä toteutukseen. Käsitteellinen suunnitelma vastaa ennemmin karkeita luonnoksia ja kuvauksia rakennettavasta talosta. Suunnittelun elementit ovat helposti ymmärrettäviä ja asiakkaiden, käyttäjien ja arkkitehtuurin suunnittelijan yhdessä laatimia. Käsitteellisen suunnitelun tuotos on liike-elämän ja käyttäjien tekemien määrittelyjen käännös käyttäjien kielestä suunnittelijoiden kielelle.

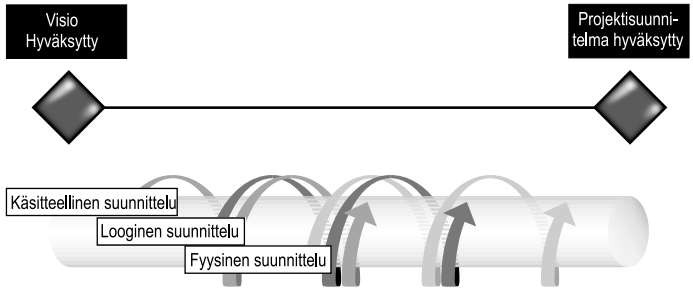
Looginen suunnittelu alkaa pureutua sovelluksen yksityiskohtiin, jotka työryhmän on toteutettava täyttääkseen liiketoiminnan tarpeet ja käyttäjien vaatimukset. Looginen suunnittelu vastaa rakennuksen pohjaratkaisua ja talon pystyttämistä, jolloin elementit kuten tilasuhteet organisoidaan. Prosessin tässä vaiheessa arkkitehdin ryhmä määrittää suunnitelman suhteet ja tietoliikenneväylät.

Fyysinen suunnittelu tarkentaa käytettävän teknologian. Tämä vaihe lisää arkkitehtuuriin yksityiskohdat ja huomioi tosimaailman toteutukseen tuomat rajoitukset.

Pääasiassa MSF:n suunnitteluprosessin vaiheet palvelevat erilaisten "lukijoiden" tarpeita, vaikka ne samalla ovatkin toimivan kokonaisuuden osia. Sen sijaan, että pakottaisi lukijan käymään läpi erittäin laajan ja yksityiskohtaisen dokumen-tin, malli mahdollistaa keskittymisen vain lukijalle itselleen oleelliseen tietoon. Malli sallii myös muodollisten ja epämuodollisten tekniikoiden käyttämisen tarpeen mukaan.

Suunnitelma kehittyy vaihe vaiheelta kohti toiminnallista määrittelyä. Ja samalla kun sovellus kehittyy, jokainen yksityiskohta on milloin tahansa jäljitettävissä takaisin alkuperäiseen ongelmaan, tarkistamista ja testausta varten.

Vaikka MSF:n suunnitteluprosessi sijoittuukin pääasiassa MSF Development Process Model -mallin suunnitteluvaiheeseen, suunnittelutyö aloitetaan usein ennen kuin suunnitteluvaihe virallisesti alkaa ja jatkuu jossain laajuudessa siihen saakka, kunnes koodi jäädytetään (kehitysprosessin loppuvaiheessa). Jokaisen kolmen vaiheen peruslinjat vakiinnutetaan suunnitteluvaiheessa. Kuten kuvasta 1.7 havaitaan, nämä suunnittelutoiminnot etenevät portaittaisesti ja rinnakkaisesti, niin että yhden toiminnon varhaisimmat tulokset ilmaisevat, että seuraavan vaiheen tulee alkaa.



Kuva 1.7 MSF:n suunnitteluprosessin limittäisyys, iteratiivisuus ja spiraalimaisuus

Käsitteellinen suunnittelu alkaa, kun ryhmä sopii vision määritelmästä, mikä voi tapahtua ennen kuin muodollinen Vision Approved Millestone -välitavoite on saa-vutettu. Looginen ja fyysinen suunnittelu ovat rinnakkaisia käsitteellisen suun-nittelun kanssa ja kaikki kolme sijoittuvat kokonaan, mutta eivät rajoitu MSF Development Process Model -mallin suunnitteluvaiheeseen.

Kaikkien kolmen vaiheen yhtäaikainen suorittaminen tuo käsitteelliseen, loogiseen ja fyysiseen suunnitteluun seuraavat ominaisuudet:

- **Limittäisyys** (Overlapping) Toiminnot ovat rinnakkaisia siten, että käsit-teellinen suunnittelu alkaa tyypillisesti ensimmäisenä ja sitä seuraavat loo-ginen ja sitten fyysinen suunnittelu. Ne eivät ole peräkkäisiä, joten edellisen vaiheen ei tarvitse päättyä ennen kuin seuraava alkaa. Itse asiassa looginen ja fyysinen suunnittelu voivat alkaa jopa ennen kuin käsitteellisen suunnittelun peruslinjauksiakaan on tehty.
- **Iteratiivisuus** Toiminnoilla on omat tarkistus-, suunnitelman testaus-(vastakohtana koodin testaamiselle) ja uudelleensuunnittelusyklinsä.
- **Spiraalimaisuus** Jokainen kaikkien kolmen toiminnon toisto edustaa kehittymistä kohti suunnitteluvaiheen Project Plan Approved välitavoitetta.

Selvää on, että käytännössä tulee esiintyä mahdollisimman vähän sekvensointia. Käsitteellisen suunnittelun tulee käynnistyä ennen loogista suunnittelua, jonka tulee alkaa ennen fyysistä suunnittelua. Vastaavasti, käsitteellisen suunnittelun peruslinjat täytyy määritellä ennen loogisen suunnittelun peruslinjoja ja ne taas ennen fyysisen suunnittelun peruslinjoja.

Suunnittelunäkökohtia

Ymmärryksen lisääminen projektityöryhmän sisällä on tärkein huomioon otetta-va asia pohdittaessa, kuinka laajaan liiketoimintasovellukseen luonnostaan kuu-luvaa *monimutkaisuutta* hallitaan ja kun pyritään arvioimaan, miten suurella tarkkuudella suunnitelma tulisi laatia. Jos yksityiskohtia on liian vähän, on ole-massa vaara, että työryhmältä jää huomioimatta tärkeitä vuorovaikutussuhteita. Toisaalta, jos mukana on liikaa yksityiskohtia, suunnitelmasta voi tulla liian mo-nimutkainen.

C++-ohjelmoijien kokemus ja taidot ovat erityisen arvokkaita loogisessa suun-nitteluprosessissa, sillä objektiorientoituneen suunnittelun periaatteet ovat hyvin samanlaisia kuin loogiseen suunniteluun vakiintuneet menetelmät. Tämä jakso hahmottelee muutamia loogisen suunnitteluprosessin avainkäsitteitä — käsitteitä, jotka ovat tuttuja kaikille, jotka ovat hankkineet kokemusta objektiorientoituneesta sovellussuunnittelusta.

Modulaarisuus

Työryhmä aloittaa loogisen suunnitteluprosessin määrittämällä järjestelmän tär-keimmät palvelut. Palvelu edustaa kokoelmaa prosesseja, jotka toimivat yhdessä suorittaakseen jonkin tehtävän. Modulaarisen suunnittelun pääajatus on se, että

lopullinen tuote koostuu pääpalvelujen yhdistelmästä. Jokainen palvelu on suunniteltu toteuttamaan tietty tehtävä itsenäisenä yksikkönä, jonka täytyy toimia yhdessä muiden järjestelmän osien kanssa.

Abstraktio

Abstraktio tarkoittaa menetelmiä, joilla tutkimme ja suodatamme yksittäisen asian ominaisuuksia ja erikoisuuksia ja luokittelemme asioita olennaisilta ominaisuuksiltaan samanlaisten asioiden ryhmiksi. Abstraktion avulla voimme selkiyttää ja järjestää jokapäiväistä ympäristöämme keskittymällä kohteiden määrääviin piirteisiin ja käyttäytymismalleihin ja jättämällä epäolennaiset yksityiskohdat huomioimatta. Tämä prosessi auttaa meitä käsittelemään asioita, jotka muuten olisivat ylipääsemättömän monimutkaisia.

C++-ohjelmoijat ovat tyypillisesti tottuneet abstraktioimaan ympäröivän maailman kohteita ja abstraktimpia käsitteitä *luokiksi*, jotka määrittelevät ohjelmiston osien ominaisuudet sovelluksessa. Esimerkiksi postin jakelujärjestelmän suunnittelija saattaisi suunnitella luokat edustamaan kirjeitä, paketteja, kuljetusautojen reittejä ja lentoaikatauluja samoin kuin ohjelmiston sisäisiä yksiköitä, kuten cachea, tapahtumien aikataulutusta ja viivakoodin käsittelyä varten.

Kapselointi

Kapselointi (encapsulation) on rakenteellisen suunnittelun, objektiorientoituneen suunnittelun ja objektiperustaisen komponenttisuunnittelun perusperiaatteita. Rakenteellisessa suunnittelussa kapselointi tarjoaa käytettäväksi käsitteen "black box", joka tarkoittaa käännettyä, yksikkötestattua moduulia, jonka voidaan olettaa suorittavan rajoitetun, tarkkaan määritellyn toiminnon oikein. Objektiorientoituneessa suunnittelussa se tarjoaa perustan abstraktiolle. Abstraktio keskittyy jonkin kohteen oleellisimpiin ominaisuuksiin suhteessa tarkastelijan näkökulmaan. Ohjelmiston komponentit kapseloivat komponenttien käyttäytymismalleja, jotka ne tarjoavat palveluina asiakassovelluksille.

Kapselointi on tapa pakata informaatiota (tietoa ja prosesseja) niin, että se minkä kuuluu olla piilossa on kätketty, ja se minkä tulee olla näkyvää on näkyvissä. Komponentista tulee määritellä se, minkä palvelun se tarjoaa ja minkä tyyppisiä sen syötteet ja palautteet ovat. Sen ei tule näyttää ulospäin, kuinka se tarjoamansa toiminnot toteuttaa — erityisesti sen ei tule paljastaa sisäistä tilaansa niin, että ul-koiset tekijät voisivat sotkea sen toimintoja.

Kun ohjelmoija päättää paljastaa objektin jonkin ominaisuuden, hän käyttää rajapintaa. Rajapinta määrittelee palvelut, jotka komponentti voi tarjota asiakkaalle. Jotta varmistuttaisiin siitä, että komponentti on mahdollisimman helppokäyttöinen ja että sitä on mahdollisimman vaikea väärinkäyttää, sen sisäisestä toiminnasta tulisi piilottaa niin suuri osa kuin mahdollista.

Kapselointi mahdollistaa järjestelmän kokoamisen paloista. Kapseloinnin idea on sovellettavissa kaikille suunnittelun tasoille — laajimmista alijärjestelmistä pienimpiin rutiineihin ja toimintoihin saakka.

Kiinteys ja kytkentä

Sisäinen *kiinteys* (cohesion) viittaa siihen, kuinka läheisesti palvelun toiminnot liittyvät toisiinsa. Kaiken suunnittelun tavoitteena on maksimoida hyödyllinen kiinteys rakenneosien välillä. Suuri yhteneväisyys helpottaa moduulin tarkoituksen tai toiminnan abstraktointia niin, että sen toimintatavat ovat helpommin ja tarkemmin määriteltävissä.

Osien välinen *kytkentä* (coupling) viittaa palvelujen välisiin suhteisiin. Mitä pienempi määrä informaatiota liittää modulit toisiinsa, sen vapaammin suunnittelija voi koota uusia ratkaisuja aiheuttamatta häiriöitä palveluihin itseensä tai palvelujen olemassaolevaan yhteistoimintaan. Palveluiden tulisi olla mahdollisimman vähän riippuvaisia toisistaan. Missä riippuvuutta kuitenkin esiintyy, yhteyksien tulisi olla niin suoraviivaisia ja selkeitä kuin mahdollista. Tällä pyritään helpottamaan määrittelyä ja suurempaan yksinkertaisuuteen rajapintoja määriteltäessä.

Hyvin suunnitellun palvelun muodostavilla yksiköillä on heikko yksiköiden välinen kytkentä ja voimakas sisäinen kiinteys. Ajatellaan esimerkkinä hifijärjestelmää, joka koostuu kolmesta eri palvelusta — vahvistimesta, CD-soittimesta ja kasettidekistä. Vahvistimen sisäiset osat, kuten transistorit ja vastukset, eivät yksinään tee juuri mitään. Niiden toiminta perustuu suureen sisäiseen kytkentään muiden oman tehtävänsä tekevien vahvistimen moduulien kanssa. Toisaalta hifijärjestelmän modulaariset osat ovat malli heikoista yksiköiden välisistä yhteyksistä — ne on yleensä yhdistetty muutamalla kaapelilla ja ne voidaan helposti vaihtaa.

Rajapinnan ja toteutuksen eriyttäminen

Palvelut paljastavat ulkoisia piirteitään, jotka mahdollistavat niiden käytön rakennuspalikoina. Jokaisella palvelulla on rajapinta, joka mahdollistaa sen käyttämisen, mutta joka kätkee sen sisäisen toiminnan niin, että sisäistä toteutusta voidaan muuttaa ilman että vaikutetaan muuhun järjestelmään. Jotta tämä lähestymistapa toimisi tehokkaasti, täytyy ohjelmoijien ymmärtää, että jokaisella palvelulla on toiminnallisuutta, joka täytyy saada muun maailman käytettäväksi. Tämän toiminnallisuuden kuvausta voidaan käyttää sopimuksena muiden moduuleiden kanssa. Mikä tahansa muu palvelu, joka mukautuu tähän sopimukseen, voi hyödyntää toiminnallisuutta.

Sopimus määrittelee kuluttajan ja palvelun tarjoajan välisen vuorovaikutuksen säännöt. Esimerkiksi tiedämme, että jos haluamme käyttää puhelinkioskia, meidän täytyy laittaa ensin muutamia kolikoita automaattiin ja valita numero. Me luotamme siihen, että jos teemme tämän oikein, soitto yhdistetään tai jos yhdistäminen epäonnistuu, rahat palautetaan. Vastaavasti, ohjelman komponentin käyttäjät tietävät, että niin kauan kuin käyttöliittymäsopimuksen ehtoja noudatetaan, kuluttaja voi ottaa yhteyttä palvelun tarjoajaan ja saada määrätyn palvelun. Tätä kutsutaan avoimeksi sopimukseksi. On tärkeää, että käyttöliittymä suunnitellaan niin avoimeksi kuin mahdollista, jolloin saavutetaan hyvä modulaarisuus ja uudelleenkäytettävyys.

Rajapinta määritellään loogisesti, erillään toteutuksesta, niin että kaikilla mahdollisilla käyttäjillä on yhtäläinen pääsy palveluihin. Käyttöliittymän sopimuksenluontoisuus jättää vastuun toiminnallisuuden toteuttamisesta palveluntarjoajan vastuulle. Jos valitsemme numeron oikein, on puhelinyhtiön vastuulla huolehtia tarvittavista yksityiskohdista niin, että puhelu yhdistetään oikeaan paikkaan. Käyttöliittymän toimintojen toteutus on muutettavissa, kun vain huolehditaan siitä, että uusi toteutus pystyy täyttämään samat käyttöliittymäpalvelut. Tämä mahdollistaa palvelun kehittämisen olosuhteiden tai vaatimusten muuttuessa sekä parannuksien tekemisen. Esimerkiksi puhelinyhtiö voi halutessaan vaihtaa puhelinkioskin puhetorven. Niin kauan kuin tuttu valintanäppäimistö ja kolikkoaukko pysyvät muuttumattomina, voimme jatkaa puhelimen käyttämistä ilman ongelmia.

Käyttöliittymien tarkka määritteleminen on välttämätöntä suunniteltaessa ja ohjelmoitaessa palveluja erillään toisistaan. Kun käyttöliittymät ovat hyvin määriteltyjä, nämä erillään tehdyt palvelut voidaan luotettavasti yhdistää täysin toimivaksi sovellukseksi. Selvästi määriteltyjen käyttöliittymien avulla saavutettu palvelujen välinen riippumattomuus helpottaa ylläpitoa ja ylläpito on tulevia päivityksiä ajatellen helpompaa.

Sovellusten ohjelmointi

Tällä kurssilla käsittelemme sovellusrungon osuutta työasemasovellusten ohjelmoinnissa. "Työasemasovelluksilla" emme tarkoita vain Microsoft Wordin tapaisia sovelluksia, jotka toimivat itsenäisesti yksittäisessä tietokoneessa. Määrittelemme työasemasovelluksiksi myös asiakas/palvelinsovellukset — erityisesti niin sanottuun fat-client-malliin perustuvat, joissa liiketoiminta- ja esityspalvelut on toteutettu yhdessä paikallisena yksikkönä, joka vastaanottaa tietoja etätietolähteestä. Tämä malli perustuu ajatukseen suhteellisen yksinkertaisista liiketoimintasäännöistä ja rajoitetusta jakelusta.

Työasemasovelluksen tulisi hyödyntää sovellusmallin kuvaamaa komponenttipohjaista suunnittelumallia aina kun se on tarkoituksenmukaista, jotta saataisiin hyödynnettyä komponenttipohjaisen arkkitehtuurin joustavuus, skaalautuvuus ja luotettavuus.

Oppitunnin yhteenveto

MSF on kokoelma malleja, periaatteita ja ohjeita ohjelmistojen tekemisestä ja jakelusta. MSF on kokoelma Microsoftin tuotantoryhmien ja Microsoft Consulting Services -palvelun parhaita toimintatapoja.

MSF sisältää seuraavat mallit ja näkökulmat, jotka osaltaan vaikuttavat kehitystyöhön:

- **MSF Development Team Model** Määrittelee tasavertaisista itsenäisesti yhteistyössä toimivista rooleista muodostuvan työryhmän.

- **MSF Development Process Model** Auttaa työryhmää vakiinnuttamaan suunnittelun ja kontrolloinnin suuntaviivat tuloslähtöisissä projekteissa sen soveltamisalueeseen, saatavissa oleviin resursseihin ja aikatauluun perustuen.
- **MSF Application Model** Auttaa työryhmää hajautetun sovelluksen tekemisessä hyödyntäen parhaalla mahdollisella tavalla komponenttien uudelleenkäytön. Se asettaa standardit ja suuntaviivat jaettujen asiakas/palvelin- ja monitasoisten sovellusten suunniteluun, jotka voidaan toteuttaa käyttämällä Microsoftin komponentteihin perustuvia työkaluja ja tekniikoita.
- **MSF Design Process Model** Kertoo, kuinka sovellus tulee suunnitella sekä käyttäjän että liiketoiminnan näkökulmista. Se määrittelee kolme toistuvaa suunnitteluvaihetta, jotka sitovat yhteen Team, Process ja Apllication -mallit ratkaisulähtöisessä asiayhteydessä.
- **Käsitteellisen suunnittelun näkökulma** Keskittyy liiketoiminnan ja käyttäjien tarpeisiin ja teknisiin mahdollisuuksiin. Käsitteellisen suunnittelun tuloksena syntyy kuvaus näkemyksistä.
- **Loogisen suunnittelun näkökulma** Keskittyy siihen, että projektityöryhmä ymmärtää tehtävän. Ratkaisu kuvataan järjestelmän ainesosina ja niiden välisenä yhteistyönä.
- **Fyysisen suunnittelun näkökulma** Keskittyy ohjelmoijien toteutusvaiheessa tarvitsemien tietojen tuottamiseen. Fyysinen suunnittelu keskittyy ratkaisun jäsentelyyn, rakenteeseen ja tekniikkaan. Fyysisen suunnitteluvaiheen tuloksena syntyy järjestelmästä matalan tason kuvauksia, kuten vuokaavioita, objektimalleja ja pseudokoodia.

Joskus ohjelmoijat osallistuvat vain ohjelman fyysiseen suunniteluun. Loogista suunnittelua ei kuitenkaan tulisi jättää hunningolle, koska sillä on tärkeä rooli sovelluksen kehittämisessä. Loogisen suunnitteluprosessin tarkoituksena on lisätä koko kehitysryhmän syvempää ymmärtämystä järjestelämästä ja auttaa työryhmää määrittelemään, kuinka suuriin liiketoimintasovelluksiin luonnollisesti liittyvää monimutkaisuutta hallitaan. Monimutkaisuuden hallintaan päästään käyttämällä samanlaisia suunnittelumenetelmiä kuin oliopohjaisessa sovellussuunnittelussa.

Oppitunti 2: Visual C++:n asennus

Tässä luvussa tutkitaan niitä vaihtoehtoja, joita Microsoft Visual Studio 6.0:n mukautettu asennus tarjoaa Visual C++:n Enterprise-versiota asennettaessa. Asennusohjelman avulla saadaan lisäksi hyvä yleiskuva Visual C++:n ominaisuuksista, koska siinä on koottu yhteen loogiseen ja helposti selattavaan pakettiin kaikki mahdolliset asennettavat osat. Jos olet jo asentanut Visual C++:n, voit silti käydä tämän luvun läpi ymmärtääksesi paremmin asennuksen tarjoamat vaihtoehdot.

Tämän oppitunnin jälkeen:

- Tunnet vaihtoehdot, jotka ovat valittavissa Visual C++ 6.0:aa asennettaessa.

Oppitunnin arvioitu kesto: 30 minuuttia

Microsoft Visual C++ -asennus

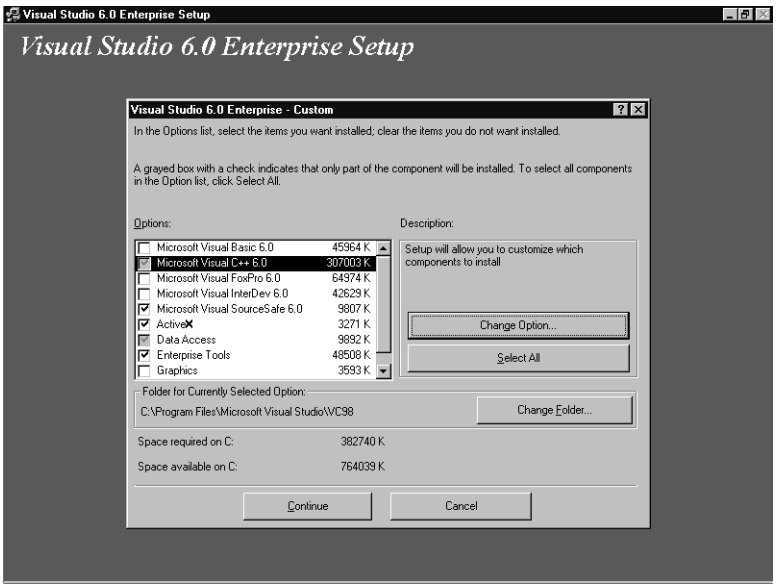
Visual C++ on saatavana sekä osana Visual Studiota että itsenäisenä tuotteena. Molemmissa tapauksissa asennus aloitetaan käynnistämällä Setup.exe-ohjelma CD-levyn juurihakemistosta.

Huomio Jos käyttämäsi käyttöjärjestelmä on Windows NT 4.0, on vähintään Service Pack 3:n oltava asennettuna ennen Visual C++:n asennusta.

Ensimmäiseksi avautuu ikkuna, joka tarjoaa sinulle mahdollisuuden tutustua Readme-tiedostoon. Tämä tiedosto sisältää tärkeitä tietoja, joilla on todennäköisesti vaikutusta työskentelyysi, joten kannattaa ainakin silmäillä sen sisältö läpi.

Seuraavaksi saat luettavaksesi käyttöoikeussopimuksen. Kun olet lukenut ja hyväksynyt sen ja lisäksi syöttänyt tuotteen tunnistenumeron sekä käyttäjätunnisteen, avautuu ikkuna, josta näet mahdolliset asennusvaihtoehdot. Valitse **Custom** ja napauta **Next**-painiketta.

Seuraavaksi hyväksytään asennuksen oletussijainti napauttamalla **Next**-painiketta. Asennusprosessi alkaa. Lue informaatioruudut ja paina **Next**-painiketta, kunnes näytölle ilmestyy **Custom Setup** -ikkuna, jonka näet seuraavalla sivulla kuvassa 1.8. (Kuvassa oleva ikkuna on Visual Studio -paketista, joten siinä näkyvät myös muiden ohjelmien asennusvaihtoehdot.)



Kuva 1.8 Visual Studion Custom Setup -näyttö

Jos olet jo asentanut Visual C++:n, mene ylläpitoikkunaan, jossa voit lisätä ja poistaa nykyisen asennuksen osia. Ylläpitoikkunaan pääset valitsemalla **Workstation**-vaihtoehdon. Ylläpitoikkuna on samanlainen kuin asennusikkuna, joten voit käyttää sitä asennusvaihtoihin tutustumiseen.

Custom setup -ikkunassa on valmiiksi valittuna oletusasennuksen mukaiset vaihtoehdot, joita voit tarvittaessa muuttaa. Väkänen asennusvaihtoehdon vieressä tarkoittaa, että se on valittuna. Monet valinnoista on järjestetty hierarkkisesti. Jos yksi **Options**-luettelon kohdista on valittuna ja **Change Option** -painike on käytettävissä, pääset sitä napauttamalla tutkimaan kyseisen kohdan valintoja tarkemmin. **OK** ja **Cancel**-painikkeita käyttämällä pääset takaisin hierarkian ylemmille tasoille, joko säilyttäen (ok) tai peruuttaen (cancel) tekemäsi muutokset.

Jos jokin kohdista on valittu, mutta valintaruutu on varjostettu, se tarkoittaa, että vain osa alempien tasojen kohteista on valittu asennettavaksi.

Seuraava harjoitus näyttää, kuinka käytät Custom Setup tai Maintenance -ikkunoita asennettavien osien listan muuttamiseen.

► **Asnnuksen mukauttaminen**

1. Valitse **Microsoft Visual C++ 6.0** -vaihtoehto joko Custom Setup tai Maintenance -ikkunan **Options** -listasta kuten kuvassa 1.8.
2. Napauta **Change Option** -painiketta.

3. Valitse **Options**-listasta **VC++ Runtime Libraries** vaihtoehto. Napauta **Change Option** -painiketta.

   Huomaa, että **CRT Source code** -vaihtoehtoa ei ole valittu. Huomaa myös, että sen asentaminen vaatii 5805 kt levytilaa. C:n ajonaikaisen lähdekoodin asentaminen mahdollistaa myös C:n standardi kirjastojen tutkimisen debuggerilla. Jos CRT-lähdekoodia ei ole saatavilla näet vain assemblykielisiä komentoja tutkiessasi debugerilla kirjastofunktioita. Näiden lähdekoodien tutkiminen voi olla opettavaista, koska niistä näkee havainnollisesti, kuinka C-ohjelmoinnin ammattilaiset kirjoittavat ohjelmakoodia.

4. Jos haluat asentaa CRT-kirjastojen lähdekoodit ja sinulla on käytettävissäsi riittävästi levytilaa asennusta varten, napauta tämän vaihtoehdon vieressä olevaa valintaruutua.

5. Jos olet muuttanut nykyisen ikkunan Options-listassa olevia asetuksia ja halua tallentaa muutokset, napauta **OK**-painiketta päästäksesi takaisin ylemmälle tasolle. Muuten napauta **Cancel**-painiketta jättääksesi valinnat alkuperäisiksi.

   Vihje Jos teet muutoksia yhdellä tasolla ja tallennat ne napauttamalla **OK**-painiketta, ja sitten napautat **Cancel**-painiketta ylemmällä tasolla, alemmalla tasolla tehdyt muutokset jäävät edelleen voimaan. Jos haluat perua alemmalla tasolla tekemäsi muutokset, sinun täytyy palata sille tasolle, jossa muutokset tehtiin ja muuttaa valinnat takaisin alkuperäisiksi ja sen jälkeen napauttaa **OK**-painiketta.

   ---

6. Kun kaikki halutut muutokset asetuksiin on tehty, palaa takaisin Custom Setup tai Maintenance -ikkunaan ja jatka asennusta napauttamalla **Continue**-painiketta. Poistuaksesi asennuksesta napauta **Cancel**-painiketta.

Asennusvaihtoehdot

Tämän luvun loppuosa koostuu Visual C++ -asennusvaihtoehtojen lyhyistä kuvauksista. Etsi jokainen esiteltävä vaihtoehto käyttämällä Custom Setup tai Maintenance -ikkunaa ja mieti, pitäisikö kyseisen osan olla asennettuna tietokoneellesi.

Visual Studio

Visual Studion asennuksessa ovat valittavissa vaihtoehdot kaikki tai ei mitään ja näin ollen sinun tulisi valita oletusvaihtoehto **All** eli kaikki. Oletuksena asennetaan kehitysympäristö, ohjatut toiminnot ja debuggeri.

Visual C++ -käyttöliittymä koostuu joukosta ikkunoita, työkaluja, valikoita, työkalurivejä ja muita elementtejä, joiden avulla voit luoda, testata ja hioa sovelluksesi yhdessä paikassa. Voit käsitellä myös muuntyyppisiä asiakirjoja kuten Microsoft Excel tai Microsoft Word -dokumentteja Visual C++:ssa.

Run-Time Libraries

VC++ Runtime Libraries -valinta, josta luit edellisessä jaksossa, antaa neljä vaihtoehtoa:

- **Static CRT Libraries** Moniajoon kykenevät kirjastot. Nämä kirjastot linkitetään sovellukseesi sovellusta linkitettäessä ja kasvattavat ajettavan tiedoston kokoa. Etu näiden kirjastojen käyttämisestä on se, että ajettava tiedosto sisältää kaiken tarvittavan. Haitta puolestaan on se, että ohjelman tai kirjaston muuttaminen edellyttää koko sovelluksen uudelleen rakentamista.
- **Shared CRT Libraries** Linkitetään sovellukseen suorituksen aikana. Sovelluksen .exe-tiedostosta tulee pienempi kuin jos sovelluksen kaikki osat linkitettäisiin kiinteästi mukaan. Dynaamisen linkityksen etuna on myös se, että sovelluksen rakentaminen on nopeampaa ja se, että uudet versiot ohjelmasta voidaan levittää pienempinä tiedostoina. Huono puoli taas on se, että täytyy varmistua siitä, että kaikilla käyttäjillä on kaikki tarvittavat kirjastotiedostot saatavilla.
- **CRT Source Code** Asentaa C++ run-time kirjastojen lähdekoodit. Lähdekoodien saatavillaolo nopeuttaa ohjelmointia ja virheen etsintää. Voit tarkastaa koodista, kuinka toiset ohjelmoijat ovat ratkaisseet tietyn ohjelmointitehtävän.
- **Single Threaded CRT Libraries** Käytetään, kun kirjastot linkitetään staattisesti ja tehdään ohjelma, joka ei tue moniajoa.

Kaikista kirjastoista on olemassa optimoidut julkaisuversiot, joissa ei ole virheenetsinnässä käytettäviä symboleja ja versiot, joita ei ole optimoitu, mutta joissa debuggaussymbolit ovat mukana. Jos asennat lähdekoodit, voit tutkia myös kirjastofunktioita tehdessäsi debuggausta.

Win32 (eli Windows 95, Windows 98 ja Windows NT) -ympäristöön sovelluksia tekevät haluavat yleensä käyttää näiden ympäristöjen tarjoamaa moniajomahdollisuutta. Jos teet ohjelmia, joiden pitää toimia myös vanhemmissa 16-bittisissä ympäristöissä tarvitset Single Threaded CRT -kirjastoja.

MFC Library

MFC Library on kokoelma C++-luokkia, jotka muodostavat rungon Windows-ympäristöön tehtäville sovelluksille. Windows ohjelmoinnin ammattilaisten tekemien kirjastojen avulla voit helposti ohjelmoida omia sovelluksiasi, kirjastojasi ja ohjelmakomponenttejasi. MFC mahdollistaa nopean kehitystyön kaikilla Windows ohjelmoinnin osa-alueilla, mukaan lukien käyttöliittymän tekeminen, Internetohjelmointi, komponenttitekniikka ja tietokantojen käsittely - rajoittamatta kuitenkaan ohjelmoijan toimintavapautta tai joustavuutta. MFC tarjoaa käyttöösi *AppWizardin*, vaihe vaiheelta etenevän toiminnon, joka nopeuttaa ja helpottaa perusohjelman rungon luomista ja generoi ohjelmien perustoimintojen tarvitseman koodin.

VC++ MFC and Template Libraries -vaihtoehto sisältää seuraavat MFC-kirjastot:

- **Static Libraries** Asentaa header-tiedostot, kirjastot ja debugmerkinnät MFC:n staattista linkitystä varten.
- **Shared Libraries** Asentaa header tiedostot, kirjastot ja debugmerkinnät jaettujen MFC DLLien linkittämistä varten.
- **Static Libraries for Unicode** Sisältää Static Libraries kirjastojen *Unicodea* varten sovitetut versiot. Unicode tarkoittaa Windows NT:n tukemaa 16-bittistä merkistöä, joka on laajudeltaan yli 65 000 merkkiä. Tarvitset Unicode-merkistöä, jos kehität ohjelmistoista kansainvälisiä versioita kielille, joissa merkistö on hyvin laaja, kuten esimerkiksi kiinan kielelle. Huomaa myös, että Windows NT:n sisäiset merkkijonot, kuten käyttäjätilien ja laitteiden nimet, ovat Unicode-muodossa, joten jos ohjelmasi käsittelevät näitä tietoja, sinun täytyy asentaa Unicode-kirjastot.
- **Shared Libraries for Unicode** Dynaamisesti linkitettävät jaetut MFC-kirjastojen Unicode-versiot.
- **Browser Database** Asentaa MFC-kirjastojen debug versioiden selaustietokannan. Selain tarjoaa graafisen esityksen luokkien periytymisestä ja niiden välisistä suhteista. Helpottaa ja nopeuttaa siirtymistä oman koodin sisältä MFC-lähdekoodiin. Selaintietojen liittäminen projektiin kasvattaa kuitenkin käännöksessä kuluvaa aikaa ja ajettavan tiedoston kokoa.
- **Source Code** Yksi parhaita tapoja oppia ohjelmoimaan MFC:n avulla on tutkia sen koodia ja käydä se läpi debuggerin kanssa.

ActiveX Template Library

Tämä vaihtoehto on sama kuin **MS ActiveX Template Library** (ATL) -vaihtoehto. Valitse **MS Foundation Class Libraries** asentaaksesi ATL:ään kuuluvat header-tiedostot, kirjastot ja lähdekooditiedostot.

ATL on malleihin perustuva C++-luokkien joukko, jonka avulla voit helposti tehdä Component Object Model (COM) -objekteja. ATL yksinkertaistaa usein monimutkaista COM-objektien kehitystyötä aiheuttaen vähemmän lisäkuormaa kuin MFC. ATL on erityisen käyttökelpoinen Internet-sovelluksia tehtäessä, koska sillä tehdyt komponentit ovat pieniä ja nopeita. ATL:llä on erikoistuki COM-perusominaisuuksia varten, mukaan lukien tärkeimmät käyttöliittymä- ja apuluokat, jotka yksinkertaistavat työskentelyä COM objektien ja tietotyyppien kanssa. COM ja ATL käsitellään tarkemmin myöhemmin tässä kirjassa.

Build Tools

Build Tools -vaihtoehto sisältää kääntäjän, linkittäjän, NMAKEn ja muiden kehitystyökalujen asennukset.

Data Access

Data Access -valinta on C++-vaihtoehtojen tavoin valintaikkunana päätasolla. Tähän ryhmään kuuluvat seuraavat toiminnot:

- **ADO, RDS, and OLE DB Providers** OLE DB on uusi matalan tason käyttöliittymä, jossa otetaan käyttöön yleisten tietolähteiden ajatus. Tämä tarkoittaa sitä, että OLE DB pystyy käsittelemään mitä tahansa tietoa riippumatta sen muodosta tai tallennustavasta, eikä ole näin sidoksissa Indexed Sequential Access Method (ISAM) -menetelmään, Microsoftin Jet -tietokantoihin tai edes relaatiotietokantamalliin. Käytännössä tämä tarkoittaa sitä, että voit käsitellä tietoa, joka on Excelin taulukossa, tekstitiedostossa tai jopa sähköpostipalvelimella kuten Exchangessa.

  OLE DB on joukko käyttöliittymiä, jotka COM-tekniikkaa hyödyntämällä mahdollistavat eri lähteissä olevan tiedon käsittelyn. OLE DB tarjoaa sovellukselle samanlaisen tietokantaliittymän riippumatta siitä, minkälaisesta lähteestä tiedot ovat peräisin. Nämä käyttöliittymät tukevat suurta osaa DBMS-tietokantatoiminnoista mahdollistaen tietojen jakamisen.

  ADO mahdollistaa tietokantapalvelun tietoja käsittelevän asiakasohjelman kirjoittamisen. ADO on suunniteltu helppokäyttöiseksi käyttäjätason käyttöliittymäksi OLE DB -palveluihin. ADOn tärkeimmät edut ovat helppokäyttöisyys, suuri nopeus, pieni muistin kuormitus ja vähäinen levytilan tarve.

  RDS:n palvelin- ja asiakaskomponentit toimivat yhteistyössä tuoden tietoa Internetin tai intranetin kautta Web-sivuille. RDS-komponentit mahdollistavat myös tietojen syöttämisen tietokantaan Internetin kautta.

- **Microsoft ODBC Drivers** Open Database Connectivity (ODBC) tarjoaa APIn, jonka tietokantatoimittajat toteuttavat käyttäen kyseiselle DBMS:lle tehtyä ODBC-ajuria. Ohjelmasi käyttää APIa kutsuakseen ODBC Manageria, joka välittää kutsut edelleen sopivalle ajurille. Ajuri puolestaan toimii vuorovaikutuksessa DBMSn kanssa käyttäen SQL-kieltä (Structured Query Language).
- **Jet Installable ISAM Drivers** Microsoft Jet Database -tietokantamoottori on tietokantojenkäsittely järjestelmä, joka käsittelee käyttäjä- ja järjestelmätieto-kannoissa olevaa tietoa. Microsoft Jetiä voi ajatella tietokantakomponenttina, jonka päälle muut tietojenkäsittely järjestelmät, kuten Microsoft Access ja Microsoft Visual Basic on rakennettu. Installable ISAM Drivers -ajurit mahdollistava useiden ISAM-tietolähteiden, kuten Dbase III -tiedostoformaatin, käsittelemisen.
- **Remote Data Objects and Controls** Remote Data Objects (RDO) -komponentit muodostavat ohuen käyttöliittymäkerroksen ODBC API:lle. Se on erikoisesti suunniteltu etäkäytössä olevien ODBC-tietolähteiden käsittelyyn.
- **Data Environment** Data environment -valikoima lisää käyttöön grid-kontrollin ja tuen kehityksenaikaiselle tietojen käsittelylle.

Data access -toiminnot kannattaa asentaa, mutta on hyvä miettiä tarvitsetko koskaan tehdä sovelluksia, joissa tarvitset yhteyttä ISAM-lähteeseen Jet enginen avulla ja ovatko **Microsoft ODBC Drivers** -kohdassa valmiiksi valittuina olevat tietokantamoottorit ne, joita todennäköisimmin tarvitset.

Enterprise-työkalut

Visual Studion Enterprise-version mukana toimitetaan seuraavat työkalut:

- **Application Performance Explorer (APE)** Mallintaa sovelluksesi suunnittelua ja testaa odotettavissa olevan suorituskyvyn, sekä jaetun arkkitehtuurin osien välisen vuorovaikutuksen. Voit tallentaa parhaat suunnittelemasi sovellusrakenteet ja käyttää niitä pohjana uusien muunnelmien ja suorituskykytestien pohjana.
- **Repository** Tarjoaa paikan, jossa voit säilyttää tietoja objekteista ja niiden välisistä yhteyksistä. Näin ollen se tarjoaa standardin tavan kuvata ohjelmistotyökalujen käyttämiä objektiorientoituneita tietoja: ohjelmointityökalut käyttävät eri objektiluokkien tarjoamaa informaatiota, objekteilla on omat käyttöliittymänsä, ominaisuudet, metodit ja kokoelmat muodostavat käyttöliittymän, kokoelma sisältää yhteydet muihin käyttöliittymiin, ja siksi kokoelmat liittävät objektin toisiin objekteihin. Ohjelmointityökalua kuvataan repositoryssä tietotyypin avulla. Jokainen tietotyyppi tallennetaan tietokantaan Repositorytyyppikirjastona.
- **Visual Component Manager** Työkalu, jonka avulla voit järjestellä, löytää ja lisätä Visual Studio -projektiin komponentteja. Työkalu huolehtii kolmesta komponenttien varastoimiseen ja järjestelyyn liittyvästä tehtävästä: julkistamisesta, etsimisestä ja uudelleen käyttämisestä. Visual Component Manageria käyttämällä voit etsiä olemassaolevia komponentteja, joita voit hyödyntää uusiokäytössä.
- **Visual C++ Enterprise Tools** Lisää tuen useammille tietokantatyökaluille ja tuen SQL-debuggaukselle.
- **Visual Modeler** Kolmitasoisten jaettujen sovellusten suunnitteluun tarkoitettu työkalu, jossa hyödynnetään luokka- ja komponenttidiagrammeja. Visual Mo-delerin avulla voit suunnitella visuaalisesti luokat ja komponentit, joita so-velluksesi tarvitsee ja tuottaa suunnitelmasi pohjalta Visual C++ -koodia.
- **Visual Studio Analyzer** Työkalu, jonka avulla voit kokeilla, analysoida ja debuggata jaetun sovelluksen rakennetta, suorituskykyä ja vuorovaikutteisuutta komponenttitason sijasta sovellustasolla. Visual Studio Analyzer käyttää tapahtumiin perustuvaa mallia komponenttien vuorovaikutteisuuden analysointiin.

Yhteiset työkalut

Osa työkaluista toimitetaan kaikkien Visual C++ -versioiden mukana. Suosittelemme, että asennat kaikki seuraavat:

- **MFC Trace Utility** Auttaa sinua Windows-ohjelmien debuggauksessa. Voit seurata seurantaikkunan tai konsolin avulla sovelluksesi toimintaa, MFC-kirjaston sisäistä toimintaa sekä varoituksia ja virheilmoituksia, mikäli ongelmia esiintyy. Ohjelma toimii vain sovellusten debug versioiden kanssa. Trace tallentaa seurantalippujen arvot Afx.ini tiedostoon. Nämä liput kertovat, minkä luokan viestejä sovellus lähettää seurantaikkunaan tai konsoliin.
- **Spy++** Win32-pohjainen apuohjelma, jonka avulla saat graafisen näkymän systeemin prosesseihin, säikeisiin, ikkunoihin ja viesteihin.
- **Win32 SDK Tools** Sisältää joukon sovelluskehityksessä hyödyllisiä apuohjelmia. Esimerkiksi DDESpyn avulla voit tarkkailla käyttöjärjestelmän DDE-palvelujen käyttöä. PvieW (process viewer) puolestaan tarjoaa mahdollisuuden tutkia ja muuttaa useiden järjestelmän prosessien ominaisuuksia.
- **OLE/COM Object Viewer** Näyttää kaikki rekisteröidyt kontrollit. Object Viewer lukee objektin tyyppikirjaston ja käyttöliittymän sekä rekisterin ja muuta asiaan kuuluvaa tietoa.
- **ActiveX Control Test Container** Auttaa kontrollien ohjelmoijaa testaamaan ja korjaamaan ActiveX kontrolleja. Test Containerissa voit myös testata kontrolliesi toimivuuden muuttamalla sen ominaisuuksia, kutsumalla metodeja ja aiheuttamalla tapahtumia.
- **VC Error Lookup** Yksinkertainen dialogiohjelma, joka hakee järjestelmä- ja moduulivirheitä ja näyttää virhekoodia vastaavan tekstin.

Visual SourceSafe 6.0

Asennuksen pääikkunasta löydät Visual SourceSafe 6.0 (VSS) -asennuksen. VSS helpottaa projektin hallintaa tallentamalla tiedostosi tietokantaan riippumatta niiden tyypistä (tekstitiedostot, kuvat, binaaritiedostot, ääni- ja videotiedostot). Jos tiedostoja täytyy jakaa useamman sovelluksen kesken, se käy nopeasti ja tehokkaasti. Kun lisäät tiedoston VSS:ään, se pakataan tietokantaan ja saatetaan muiden saataville. Tiedostoon tehtävät muutokset tallennetaan, joten vanha versio voidaan palauttaa koska vain. Ryhmäsi jäsenet voivat nähdä minkä tahansa tiedoston uusimman version, tehdä niihin muutoksia ja tallentaa uudet versiot tietokantaan.

Osan 2 luvussa 3 kerrotaan, kuinka VSS:ää käytetään lähdekoodin hallintaan, joten varmista, että se on asennettuna koneellesi. Asennuksen aikana sinulta saatetaan kysyä, haluatko käyttää vanhaa vai uutta VSS-tietokantamuotoa. Yleensä kannattaa valita uusi muoto, koska myös vanhat tietokannat voi muuttaa uuteen muotoon käyttäen DDCONV.EXE-apuohjelmaa.

Asennuksen viimeistely

Kun asennuksen valinnat on tehty, aloita tiedostojen asentaminen painamalla **Continue**-painiketta. Sinua kehotetaan käynnistämään koneesi uudelleen asennuksen viimeistelemiseksi. Uudelleenkäynnistyksen jälkeen kysytään, haluatko asentaa Microsoft Developer's Network (MSDN) -kirjaston, joka toimitetaan Visual C++:n mukana. Se kannattaa asentaa, sillä se on tapa, jolla saat Visual C++:n online-ohjeet käyttöösi.

Oppitunnin yhteenveto

Visual C++ tarjoaa sovelluskehittäjille monipuolisen ohjelmointiympäristön. Ympäristö sisältää laajan valikoiman työkaluja, joiden avulla voit tuottaa tehokasta ja virheetöntä koodia. Ympäristön tarjoamat kirjastot ja mallit auttavat yleisimpien sovellustyyppien koodaamisessa ja rakenteen määrittelyssä, niin että voit keskittyä sovelluksesi ainutkertaisten ominaisuuksien kehittämiseen.

Tässä luvussa tehty yleiskatsaus asennusvaihtoehtojen pääkohtiin toimii johdatuksena monien Visual C++ -kehitysympäristön ominaisuuksien tutkimiseen myö-hemmin tässä kirjassa. Näihin kuuluvat:

- Visual Studio ja debuggaustyökalut
- Microsoft Foundation Classes (MFC)
- ATL
- Tietokantatekniikat
- Visual SourceSafe (VSS)

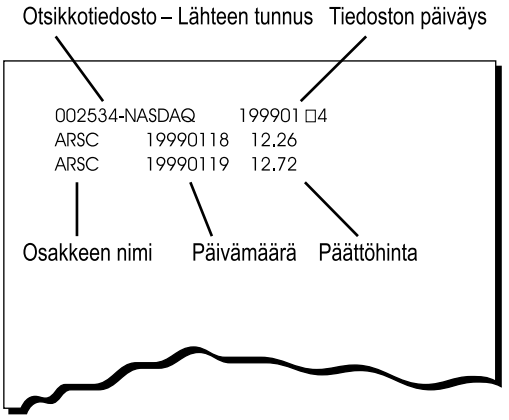
Laboratorio 1: STUpload-sovelluksen esittely

Tämän laboratorion tarkoituksena on tutustuttaa sinut sovellukseen, jota tulet kehittämään tämän kurssin aikana. Sinulle esitetään asiakkaan vaatimukset ja sen jälkeen sinun tulisi pohtia ratkaisua käyttäen MSF-mallia.

Asiakkaan vaatimukset

Asiakkaasi, Stock Watch Data Services Inc. SDS, on yritys, joka on erikoistunut tarjoamaan pörssitietoja meklareille ja rahoitusjohtajille. SDS vastaanottaa viikoittain suuren määrän tietoa, joka haetaan automaattisesti ASCII-muodossa eri puolilla maailmaa olevilta keskustietokoneilta. He haluavat sovelluksen, jonka avulla tietojenkäsittelykeskuksen työntekijät voivat hakea näitä tekstitiedostoja, tarkistaa visuaalisesti, että tiedot ovat oikeita ja tallentaa tarkistetut tiedot keskustietokantaan. Sovelluksen tulee täyttää seuraavat vaatimukset:

- Sovellus tullaan asentamaan Windows NT 4.0 -verkkoon, jolla on 50 käyttäjää. Kaikki käyttäjät työskentelevät tietojenkäsittelykeskuksen työasemilla.
- Keskustietokoneilta tuleva tieto saapuu työntekijöille sähköpostin liitetiedostoina. Liitetiedostojen tarkennin on *.dat*.
- Yhdessä tiedostossa on tietoja vain yhdestä pörssistä, esimerkiksi NASDAQ:ista tai Lontoon pörssistä, mutta tiedostossa saattaa olla tietoja useista osakkeista.
- Tekstitiedostot ovat standardin mukaisia, 8-bittisiä ASCII-tiedostoja. Tiedostoissa on otsikkorivi, josta selviää, mistä pörssistä tiedot ovat ja päivämäärä, sekä ennalta tuntematon määrä datarivejä, jotka koostuvat tasalevyisistä kentistä, kuten kuvasta 1.9 selviää.



Kuva 1.9 Ladattavan ASCII-tiedoston rakenne

- Tiedostoissa on usein samoja rivejä useita kertoja, tyhjiä arvoja sisältäviä rivejä ja rivejä, joilla olevat tiedot ovat sekoittuneet. Sovelluksen pitää varmistaa, että tällaiset rivit suodatetaan pois ja vain kelvollisia tietoja sisältävät rivit ladataan sovellukseen.
- Sovelluksen pitäisi näyttää lista kaikista osakkeista, joiden tietoja käyttäjän avaamassa tiedostossa on.
- Käyttäjän valitessa jonkin osakkeen nimen kyseisen osakkeen hintatietojen pitäisi tulla näkyviin viivakaaviona visuaalista tarkistusta varten.
- Kun tarkastaja on todennut tiedoston sisältämien tietojen oikeellisuuden, hänen pitää voida tallentaa tiedot keskustietokantapalvelimelle.
- Tietokantapalvelin on SQL Server 7.0 -palvelin, joka kuuluu tietojenkäsittely keskuksen domainiin.
- Tämä palvelin käsittelee kaikkien sovelluksemme asennusten päivitykset, sekä toimittaa kaikkien muiden sovellusten pyytämät tiedot.
- Jokaiseen tietokantaan lisättävään tietueeseen tulee liittää kaksi seurantakenttää, joista selviää milloin tieto on lisätty, sekä kuka sen on lisännyt.
- Taulukossa ei saa esiintyä samaa riviä toistamiseen.
- Kun tieto on haettu tekstitiedostosta, se voidaan tallentaa sovelluksen omassa tiedostomuodossa.
- Sovelluksen tiedostot säilytetään arkistona.
- Tietojen käsittelijä voi avata kerralla useampia .dat-tiedostoja, jolloin useiden tekstitiedostojen sisältö tallennetaan tietokantaan samalla kertaa ja yhteen arkistotiedostoon.
- Asiakas on ilmoittanut, että tämä on ensimmäinen useista sovelluksista, jotka käyttävät samaa tietokantaa.
- Asiakas toivoo, että sovelluksen tietokantaa käsittelevät osat suunniteltaisiin mahdollisimman suurelta osin uudelleenkäytettäviksi Windows NT -ympäristössä.
- Sovelluksen avulla voidaan tehdä kyselyjä tietokannasta osakkeen nimen sekä alku- ja loppupäivämäärän perusteella.
- Käyttäjä voi selata hakutuloksia, jotka ovat vain lukumuodossa, esimerkiksi varmistaakseen, että tiedot, jotka hän aikoo kantaan lisätä, eivät jo ole siellä.
- Keskustietokantaan olemassaoleva yhteys täytyy voida havaita käyttöliittymästä.
- Jos yhteyttä keskustietokantaan ei ole, täytyy käyttäjän silti voida avata tekstitiedostoja ja tallentaa tietoja sovelluksen omassa tiedostomuodossa, niin että kun yhteys jälleen muodostetaan, tiedostot voidaan avata ja niiden sisältö päivittää tietokantaan.

Ratkaisun avaimet

Pohdi, kuinka luvussa 1 esitetyt asiat voisivat auttaa sinua ratkaisun löytämisessä sekä loogisen että fyysisen suunnittelun kannalta.

Looginen suunnittelu

Mieti, ketkä sovellusta tulevat käyttämään ja missä. Asiakas on erityisesti määritellyt, että noin 50 työntekijää, jotka työskentelevät kaikki samalla Windows NT -toimialueella, tulee tallentamaan tietoja keskuspalvelimelle. Tämä näkökohta tuntuisi johtavan ajatukseen melko kevyestä asiakas/palvelin-sovelluksesta. Sinun kannattaisi kuitenkin tiedustella, onko asiakasyrityksellä laajenemissuunnitelmia. Onko todennäköistä, että käyttäjäkunta tullaan sijoittamaan useampiin rakennuksiin tai useampiin toimipaikkoihin? Kasvaako sovelluksen käyttö mahdollisesti niin paljon, että riittämättömät resurssit (kuten tietokantayhteyksien määrä) hidastavat sovelluksen käyttöä.

Asiakasta voisi myös pyytää arvioimaan, tullaanko sovellusta mahdollisesti myöhemmin laajentamaan niin, että siihen tarvitaan Web-pohjainen käyttöliittymä. Jos asiakasyrityksellä on jo käytössään intranet, voitaisiin miettiä, olisiko mahdollista käyttää sovellusta selaimen kautta, jolloin työntekijät voisivat käyttää sovellusta miltä tahansa tietokoneelta. Tämä vaihtoehto saattaisi myös yksinkertaistaa asennusta ja ylläpitoa.

Sinun tulisi miettiä myös niitä liiketoimintaan liittyviä sääntöjä, jotka sovelluksessa tulisi huomioida. Pohdi, tulisiko näiden sääntöjen hoitamisen tapahtua omalla tasollaan, vai voisiko ne integroida esitys- tai tietotasolle. Tämän sovelluksen tarvitsee huolehtia vain muutamista liiketoimintasäännöistä. Kaksoisarvojen estäminen voidaan hoitaa tietokannassa pääavainten avulla. Pääavain ei salli kaksoisarvojen syöttämistä tietokantaan. Vahingoittuneitten tietojen ja kaksoisarvojen poistaminen tekstitiedostoista voidaan helposti tehdä asiakasosassa.

SDS päättää, että heidän budjettinsa ja liiketoimintasuunnitelmansa huomioiden asiakas/palvelin-malliin perustuva sovellus sopii heidän tämän hetkisiin tarpeisiinsa parhaiten. Kaksikerroksinen arkkitehtuuri on myös heidän mielestään soveltuva malli. He toistavat kuitenkin vaatimuksensa, jonka mukaan tietokannan päivitysmodulit tulisi suunnitella mahdollisimman hyvin uusiokäytettäviksi tulevia laajennuksia ajatellen.

Täyttääksesi tämän vaatimuksen päätät suunnitella tietokannan päivitysosan niin, että se koostuu yhteistyössä toimivista komponenteista, joilla on selkeästi määritellyt rajapinnat, jotka puolestaan määrittelevät ne palvelut, joita muut sovellukset voivat käyttää. Sinun ei tarvitse toteuttaa näitä komponentteja erillisenä tasona - ne voidaan sijoittaa osaksi sovellusta ja levittää DLL:inä. Tulevaisuudessa ne voidaan joka tapauksessa uudelleen käyttää jaettujen sovellusten keskitasolla.

Fyysinen suunnittelu

Tämän kirjan laboratorioharjoitukset sisältävät sovelluksen fyysisen suunnittelun. Asiakkaasi saattaa yllättyä siitä, että käytät Visual C++:aa sovelluksen toteuttamiseen. Heidän mielipiteensä voi olla, että voisit tehdä vastaavan sovelluksen nopeammin Visual Basicilla.

Visual C++:ssa on monia ominaisuuksia, jotka kokeneen C++-ohjelmoijan käsissä mahdollistava nopean sovelluskehityksen. Esimerkiksi MFC AppWizardia käyttäen luodaan asiakas/palvelin-sovelluksen runko, jonka päälle voit toteuttaa sovelluskohtaiset toiminnot. Visual C++:n visuaaliset suunnittelu-toiminnot mahdollistavat käyttöliittymän nopean laatimisen ja koodiin liittämisen. MFC tarjoaa käyttöösi tehokkaat ja helppokäyttöiset tietorakenteet kuten vaihtelevan pituiset merkkijonot ja objektikokoelmat. ADO-tietokontrollin avulla voidaan tietokantakyselyn näyttö suunnitella muutamassa minuutissa. ATL:ää käyttäen voidaan toteuttaa asiakkaan vaatima uudelleen käytettäviin komponentteihin perustuva tietokankannan päivitysmoduuli.

Kaiken kaikkiaan voidaan todeta, että Visual C++ mahdollistaa sovellusten nopean kehittämisen ja muita Windows sovelluskehittimiä pienempien ja nopeampien moduulien tekemisen. ATLää käyttäen tehdyt komponentit ovat varsin tehokkaita — ATL suunniteltiin täyttämään Internet-ympäristön vaatimukset erittäin kevyistä komponenteista. Tämä tarkoittaa sitä, että asiakas voi olla varma, että ATL:ää käyttäen tekemäsi tietokantakomponentit soveltuvat myös sovelluksen Web-pohjaiseen jakeluun. Myöskin COM:in käyttäminen takaa sen, että komponenttien tarjoamat palvelut ovat käytettävissä monissa asiakassovelluksissa useissa eri ympäristöissä.

STUpload-sovellus

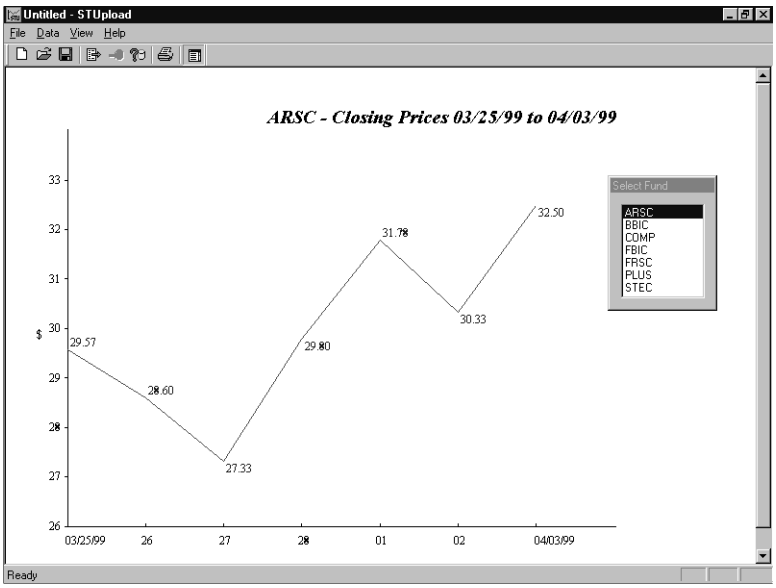
Tässä ensimmäisessä laboratoriossa tutustut STDUpload-sovellukseen. Jokaisen tämän kurssin laboratorion aikana jatkat sovelluksen tekemistä siitä, mihin edellisessä laboratoriossa jäit, joten kun olet käynyt koko kurssin läpi, sinulla on toimiva, asiakkaan vaatimukset täyttävä sovellus valmiina. Chapter1\Lab-kansio sisältää valmiin STUpload.exe-tiedoston ja Chapter1/Data-kansiossa on tiedosto Ch1Test.dat, joka sisältää tietoja, joita voit ladata sovellukseen ja edelleen tietokantapalvelimelle vietäviksi.

Suositeltavaa on, että teet STUpload-ohjelmaa samassa järjestyksessä kuin työ laboratorioissa etenee. Seuraa laboratorioita numerojärjestyksessä. Tallenna tekemäsi työ ja käytä sitä pohjana jatkaessasi työskentelyä seuraavassa laboratoriossa. Jos et saa laboratoriota valmiiksi, voit aloittaa seuraavan laboratorion tekemisen käyttäen \Labs\Labn\Partial-kansiossa olevaa projektia, joka mahdollistaa työn jatkamisen oikeasta kohdasta.

Saatuasi laboratorion valmiiksi sinun tulisi verrata työtäsi \Labs\Lab*n*\Solution-kansiossa olevaan projektiin.

► STUpload-sovelluksen käynnistäminen

1. Kaksoisnapauta **STUpload.exe**-sovelluksen kuvaketta Labs\Lab1\Solutions-kansiossa. STUpload-sovellus käynnistyy.
2. Tutki sovelluksen valikoita. Pysäytä osoitin työkalurivillä olevien tuntemattomien kuvakkeiden päälle saadaksesi selville niiden toiminnan.
3. Valitse **Data**-valikosta **Import**.
4. Etsi **File**-dialogia käyttäen Ch1Test.dat-tiedosto Chapter1/Data kansiosta. Avaa tekstitiedosto STUpload-sovellukseen napauttamalla **Open**-painiketta.
5. Lue viesti-ikkunasta, kuinka tiedoston tuominen onnistui ja päätä tiedoston lataaminen napautamalla **OK**.
6. Selec Fund -ikkuna avautuu. Napauta ARSC-kohdetta, jolloin pääset tutkimaan ARSC-osakkeen hintahistoriaa kuten kuvassa 1.10.



Kuva 1.10 STUpload-sovellus

7. Valitse **File**-valikosta **Save**. Tallenna tiedosto nimellä **Lab1.stu** Chapter1\Data-kansioon.
8. **Data**-valikossa huomaat kaksi lisätoimintoa, **QueryDatabase** ja **Upload**. Nämä valinnat eivät ole käytettävissä koneellasi ennen kuin konfiguroit sovelluksen tietokantayhteyden ja rekisteröit Upload COM -komponentit koneeseesi. Nämä molemmat toimenpiteet tehdään näiden laboratorioiden aikana.
9. Sulje STUpload-sovellus.

Kertaus

Seuraavien kysymysten tarkoituksena on kerrata tämän osan keskeiset asiat. Jos et osaa vastata johonkin kysymykseen, kertaa sopiva luku ja yritä uudelleen. Vastaukset näihin kysymyksiin löydät liitteestä.

1. Mikä on MSF?

2. Mitkä mallit liittyvät kaikkein läheisimmin liiketoimintakeskeisten sovellusten kehittämiseen?

3. Mikä on loogisen suunnittelun tärkein näkökulma?

4. Kuinka asiakas/palvelin-sovelluksessa voidaan käyttää kolmetasoista sovellusmallia?

5. Mitä hyötyä MFC:stä on ohjelmoijille?

6. Teet MFC-sovellusta, jonka avulla verkon pääkäyttäjä voi hallita NT-verkon käyttäjätilejä. Mitä kirjastoja täytyy asentaa?

7. Minkälaisia tietoja voidaan käsitellä OLE DB:n avulla?