Protokoll versus gränssnitt

Nästan allt sedan programmering av datorer började, har avvägningar skett mellan att använda *programmeringsgränssnitt* eller att använda *protokoll* för att låta program kommunicera. I synnerhet när det gäller kommunicerandet med varandra. Speciellt tydligt blir detta ständiga dilemma i och med att konstruktörerna fick välja mellan olika implementationer av protokoll eller gränssnitt (i synnerhet API) sedan datorn blev hopkopplad mer via Internet. Det kunde hända att det som tidigare löstes med gränssnitt för programmering löses i stället med protokoll, och tvärtom. Det finns inte så givna regler för vad man bör välja i alla situationer, och det kan vara också därför soppan mellan lösningar varierar stort.

Fördelen med protokoll är att strukturen definieras klart och lätt, men sker ofta seriellt. Detta gör att t.ex. samtidig behandling där delbara data mellan sessioner existerar, måste hanteras på ett protokollsäkert sätt. Utökningar och förändringar av programmens struktur som kommunicerar kan naturligtvis ställa till en del problem, och protokollen måste ibland revideras, kompletteras eller bytas ut mot nya. Den stora fördelen är dock att protokollen medger att programmen kan kommunicera oavsett vilka system eller programspråk de är skrivna i. En av nackdelarna är att konstruktionen inte medger allt för snabba kommunikationer.

Fördelarna med programmeringsgränssnitt (API eller Application Programming Interface) är att de kan vara så mycket smidigare att arbeta med, speciellt om de inte behöver specialbehandlas utan finns tillgängliga direkt i programspråket. Delbara data eller resurser (shared resources) kan mer omedelbart överblickas och behandlas, om programspråket eller systemet mot vilket kommunikationen sker, tillåter det. Det är inte ovanligt att dagens operativsystem bygger till mycket stora delar på gränssnitt av API-typ. Skilda programbibliotek för t.ex. fönsterhantering förutsätter anrop till biblioteken sker smidigt och inte minst snabbt. En protokollösning i detta sammanhang skulle helt enkelt inte passa. Men om fönsterhanteringen gjordes som klient och denna kommunicerade med en server, vore en delvis baserad protokollösning förmodligen att föredra.

Internet å sin sida består mycket av protokoll av skilda slag, på högre eller lägre nivåer i TCP/IP (den grundläggande familjen av protokoll). Faktum är att Internet kan nästan sägas vara definierat av sina protokoll. Fördelarna här är naturligtvis att låta mycket skilda datorer kommunicera över avstånd. Protokollen existerar på olika nivåer, och helt protokollöst är inte ens realistiskt alternativ. Även HTTP och HTML är en protokollösning, snarare än API. Men sådant som Java via WWW kan delar överlåtas åt gränssnitt, där de på ett underliggande plan sköts om av protokoll.

API och protokoll står heller nästan aldrig i varandras motsats utan förekommer i viss mån blandade. Däremot kan det vara väsentligt att reflektera över *hur* och *när* det ena eller andra skall och kan brukas.

Protokoll & gränsnitt

En idé kunde vara att alltid försöka implementera såväl *protokoll* som *API*. Hur vore det om t.ex. Microsoft Windows inte bara implementerade sina API (som de redan har för operativsystemet) men protokoll som t.ex. TCP/IP sviten? Tänk att nå operativsystemet med anrop ifrån långväga avstånd? Det är precis det, såväl Microsoft som tredjeparts tillverkare försöker i nuläget göra (vidare betoning av sådant som COM/DCOM). Men inte utan svåra och hjärteknipande manövrar. Microsofts operativsystem, liksom många andra, är inte konstruerade med utgångspunkt från denna typ av förhållanden. Säkerheten sätts snabbt ut ur spel, och försöken att lappa ihop säkerheten som ett tillbehör kan bara misslyckas. Säkerheten måste byggas in i ursprunget, med utgångspunkt ifrån att vara ett nätverkskopplat operativsystem. Byggs framtida operativsystem med denna utgångspunkt är chansen att få väl fungerande system betydligt bättre. Men en API skulle också vara bra att behålla då vissa implementationer säkert föredra denna variant.

En generellare attityd är CORBA (Common Object Request Broker Architecture). CORBA är en arkitektur från OMG (Objekt Management Group) som beskriver hur (objekt-orienterade) tillämpningar kan kommunicera med varandra. CORBA 1.1 (från 1991) definierade gränssnitt för hur tillämpningar kan kommunicera via en implementation av en ORB (Object Request Broker). CORBA 2.0 (från 1994) beskriver hur *olika* implementationer av en ORB från *olika* tillverkare kan samverka. En ORB är en förmedlande programvara (middleware) som existerar mellan en server och en klient. Klienten och servern kan befinna sig inom samma nätverk eller t.o.m. på samma maskin. ORB:en tar emot en förfrågan från klienten, letar upp objektet, skickar parametrar, anropar metoder och tar emot resultatet. Klienten behöver heller inte veta var objektet finns, i vilket

språk det är konstruerat eller vilket operativsystem det körs under. Det enda synliga utåt är objektets gränssnitt, dess inre natur är väl avgränsat. Idén med en ORB är ett *protokoll* (oberoende av språk som tillämpningarna eller de anropade objekten är konstruerade genom) som beskriver hur ett *gränssnitt* mot ett objekt ser ut via gränssnittsspråket IDL (Interface Definition Language). ORB:en ger ett oberoende av operativsystem, språk eller operativsystem.

[*To be continued* …]

/Set Lonnert