Mjukvaruarkitektens verktyg och arbetssätt

Sten Sundblad, Sundblad & Sundblad ADB-Arkitektur, Uppsala

Nu är det mars 2005 och dags för den tredje och sista artikeln för Microsofts arkitektwebb på temat arkitektur och arkitektens roll i mjukvarusammanhang.

I den här artikeln tittar jag framåt snarare än bakåt. Det innebär framför allt att jag sätter Microsofts ännu inte befintliga men i Visual Studio 2005 kommande designverktyg i starkt fokus. Jag kommer också framför allt att hålla mig till de faser i arkitekturprocessen som är de mest centrala för lösningsarkitekten.

Som tidigare har jag strävat efter att så mycket som möjligt använda det svenska språket men inte gjort det in absurdum. Ibland finns det inte någon riktigt bra översättning av ett engelskt uttryck, och då kan användning av svenska missleda snarare än vägleda. För att undvika det har jag i förekommande fall inom parentes visat vilket engelskt uttryck ett använt svenskt uttryck är avsett att motsvara. Ibland har jag också valt att använda engelska ”originaluttryck”.

2xSundblads arkitektoniska ramverk

I föregående artikel berättade vi något om om vårt arkitektoniska ramverk, som är ett derivat av John Zachmans ramverk. Figur 1 visar hur vårt ramverk ser ut, och jag kommer att använda det som en slags ryggrad för den här artikeln.

	Mission Viewpoints	**Information Viewpoints**	**Process Viewpoints**	**Infrastructure Viewpoints**	**Security Viewpoints**
Business Viewpoints	e.g. Purpose of System Bsns Opportunities Bsns Goals & Metrics User Profiles Bsns Risks Bsns Vision	e.g. Information Areas Existing Databases	e.g. Capability Context Process Involved Existing Process Models Fnctl Features (Ramblings) Short Term Scope	e.g. Existing Infrastructure Operational Features (Ramblings) Policies, Standards & Guidelines Short Term Scope	e.g. Security Policies & Guidelines
Conceptual Viewpoints	e.g. User Profiles (refined) Solution Vision	e.g. Conceptual Information Model Bsns Rules Model Entity Services Service Interfaces Service Interaction Message Schemas	e.g. Bsns Process Models Use Cases Functional Features Data Use Cases Presentation Services Process/Activity Services Service Interfaces Service Interaction Message Schemas	e.g. Operational Features Service Deployment	e.g. External Security Architecture
Logical Viewpoints	e.g. Use Case Selection for Service Service Vision	e.g. Logical Entity Service and Data Model (Canonical)	e.g. Ref Architecture Logical Service Model Class Model	e.g. Component Deployment	e.g. Security Model
Physical Viewpoints		e.g. Physical Entity Service and Database Design	e.g. Physical Service Design	e.g. Physical Deployment	e.g. Physical Security Mechanisms

Figur 1- Sundblad & Sundblads arkitektoniska ramverk

Business Process Viewpoint

Den översta raden i 2xSundblads arkitektoniska ramverk – se Figur 2 – representerar ”the Business Views och Viewpoints” och är främst till för att dokumentera företagsarkitektens och verksamhetsansvariges krav på det tänkta systemet. Verksamhetsansvarige har förmodligen övergripande verksamhetskrav medan företagsarkitekten är angelägen om att det nya systemet skall passa väl in i den övergripande företags-arkitekturen.

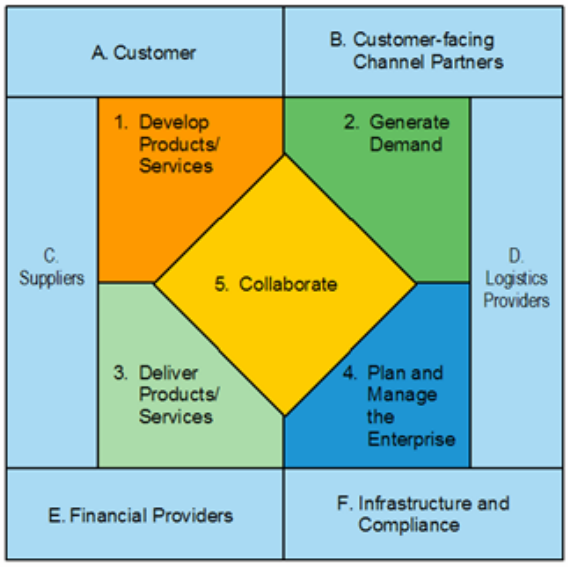
	Mission Viewpoints	**Information Viewpoints**	**Process Viewpoints**	**Infrastructure Viewpoints**	**Security Viewpoints**
Business Viewpoints	e.g. Purpose of System Bsns Opportunities Bsns Goals & Metrics User Profiles Bsns Risks Bsns Vision	e.g. Information Areas Existing Databases	e.g. Capability Context Process Involved Existing Process Models Fnctl Features (Ramblings) Short Term Scope	e.g. Existing Infrastructure Operational Features (Ramblings) Policies, Standards & Guidelines Short Term Scope	e.g. Security Policies & Guidelines

Figur 2 - Business Viewpoints

Lösningsarkitektens roll i framtagandet av uppgifter för Business Viewpoints är ofta tämligen passiv. Det är främst en uppgift för företagsarkitekten och ägare av den del av verksamheten projektet berör. I många företag tas de här uppgifterna dessutom fram i en förstudie snarare än i det verkliga projektet.

Färdigheter

Det är emellertid viktigt också för lösningsarkitekten att dessa uppgifter kommer fram och att han eller hon till fullo förstår dem. De är viktiga utgångspunkter för lösningens arkitektur. och vid behov måste arkitekten ligga på uppåt både för att få fram dem och för att de skall få det innehåll arkitekten behöver.



Figur 3 - Microsofts generiska Capability Map i nivå 1

Det mesta av innehållet i Business Views är övergripande information som oftast presenteras i listform. Ett undantag är den av oss föreslagna ”Capability Context” som beskriver projektets verksamhetsmässiga sammanhang.

Allt talar för att Microsoft, någon gång efter det att Visual Studio 2005 finns tillgängligt, kommer att publicera ett verktyg för hantering av ”business capabilities”. Vi kan inte visa något av detta just nu utan nöjer oss med att ta med samma bild av capabilities som i artikelseriens föregående artikel. Vi återkommer till capabilities och capabilitymodellering längre fram, men vill redan nu ge dig åtminstone övergripande information om vad som komma skall.

”Capability” är för resten ett bra exempel på ord som, åtminstone i detta sammanhang, är svåröversatta och det engelskspråkiga originaluttrycket kan fungera bäst.

Jag slog upp ordet i Norstedts Stora Engelsk-Svenska Ordbok och hittade förmåga, duglighet, skicklighet och möjlighet som enda förslag. Pluralformen översattes med [utvecklings]möjligheter eller anlag. Inget av dessa ord motsvarar egentligen de strategiska verksamhetstermerna capability eller capabilities i det sammanhang vi nu använder dem. Vi har ofta fått förslaget *förmåga*, vilket går bra att använda i singularis men knappast i pluralis. Det går utmärkt att säga att jag har eller besitter en förmåga att till exempel utveckla produkter och tjänster, men när vi använder pluralformen förmågor brukar vi mena människor med anlag snarare än de abstrakta capabilities vi egentligen avser.
Den bästa svenska motsvarigheten vi har kunnat tänka ut är *färdighet*, ett ord som väl täcker avsikten och som fungerar bra även i sin pluralform. Detta ord använder vi ibland, men faktum är att vi oftast ändå väljer originalformerna capability och capabilities.

Grundläggande färdigheter

Figur 3 visar den översta nivån i en hierarki av färdigheter. De inre delarna, numrerade 1 till 5, representerar de grundläggande färdigheter som varje kommersiellt företag måste besitta. Microsoft kallar färdigheter på denna nivå för *core capabilities*. Dessa är:

1. Varje kommersiellt företag måste kunna utveckla produkter och tjänster för att kunna erbjuda dem till försäljning.
2. Varje kommersiellt företag måste kunna generera efterfrågan på dessa produkter och tjänster.
3. Varje kommersiellt företag måste kunna leverera dessa produkter och tjänster.
4. Varje kommersiellt företag måste kunna planera och styra verksamheten i företaget.
5. Varje kommersiellt företag måste kunna samverka med andra företag och organisationer.

Figuren visar också att så gott som varje kommersiellt företag har kontakt med yttre enheter av följande slag:

A. Kunder (och potentiella kunder).
B. Sådana partners som i distributionskanalen finns mellan företaget och dess slutkunder.
C. Leverantörer av produkter och tjänster, vilket självklart inkluderar insatsprodukter.
D. Leverantörer av logistik- och transporttjänster.
E. Leverantörer av finansiella tjänster, som banker och försäkringsbolag.

F. Leverantörer av tjänster som rör infrastruktur och uppfyllandet av lagar, bestämmelser, standards och olika slag av överenskommelser. Här ingår till exempel sådant som advokater och revisorer.

En hierarki av färdigheter

Figur 4 visar ett Excel-ark i vilket de grundläggande färdigheterna har brutits ner till nästa nivå. Microsoft kallar färdigheter på den nivån för *capability groups*, vilket i vår översättning blir *grupper av färdigheter*. Vi skulle också kunna kalla dem för *färdighetsgrupper,* men vi trivs inte riktigt med det ordet. Och – i ärlighetens namn – vi brukar faktiskt tala om dem som just capability groups. Det är ju ändå den termen som folk även här i Sverige mest kommer att möta, och då blir det glasklart för dem vad vi menar. Det tycker vi är viktigare än att ha en överdriven strävan att hitta svenska begrepp på allting. Men det är ju mest en personlig åsikt som vi vet att inte alla delar.

Generic Process Work Flow Index		
1. Develop Product/Service		
	1.1. Develop Product/Service	
2. Generate Demand (CRM)		
	2.1. Partnership Relationship Management	
	2.2. Marketing	
	2.3. Sales	
3. Fulfill Demand (SCM)		
	3.1. Provide Service	
	3.2. Advanced Planning	
	3.3. Procurement	
	3.4. Produce Product	
	3.5. Logistics	
	3.6. Business Intelligence	
4. Plan and Manage Enterprise (ERP)		
	4.1. Financial Management	
	4.2. Project Management	
	4.3. Human Resources	
	4.4. Property and Advisory	
5. Collaboration Management		
	5.1. Strategic Collaboration	
	5.2 Planning Collaboration	
	5.3 Operational Collaboration	

Figur 4 - Nivå 2 - grupper av färdigheter

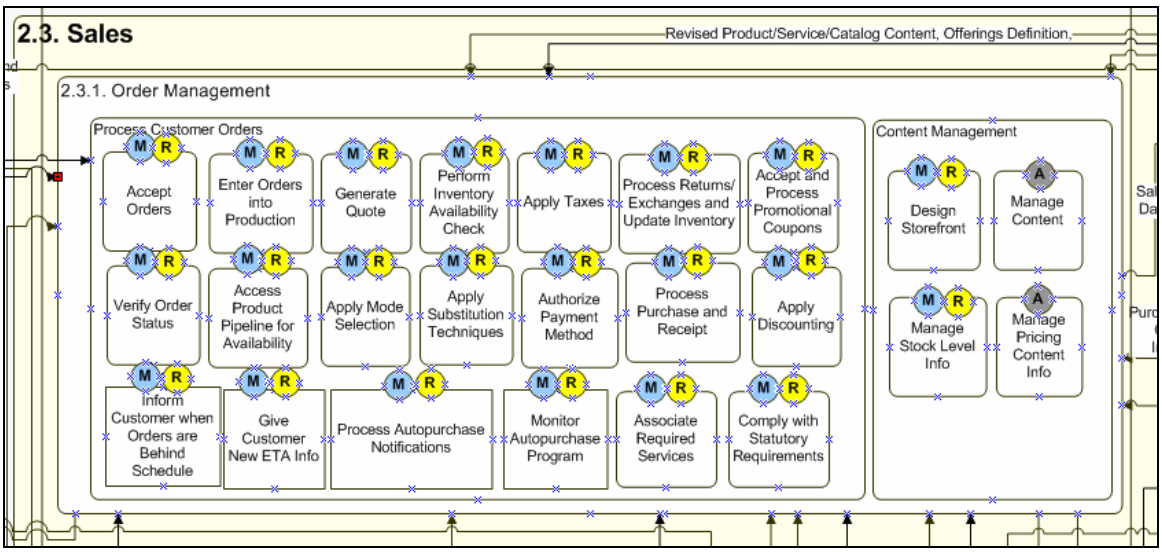
Hur som helst, här kan vi till exempel se att färdighet nummer 2 är nedbruten på tre färdigheter på lägre nivå. För att kunna skapa efterfrågan måste ett företag kunna hantera relationer med sina säljande och säljstödjande partners, det måste kunna marknadsföra sina produkter och tjänster, och det måste kunna sälja dem.

Lägg märke till att färdighet #3 här kallas för *Fulfill Demand*, medan den i Figur 3 kallas för *Deliver Products/ Services.* Jag tror att namnskillnaden beror på att Microsoft, som ju ännu inte har publicerat sitt arbete runt färdigheter, inte slutligt har bestämt namn på alla färdigheter – vi kommer säkert att få se en och annan namnändring innan det hela är tillgängligt på marknaden. Jag tycker också att det är ett bra exempel på hur en sådan här karta kommer att användas i verkligheten. Varje företag måste ju skapa sin egen karta, och även i de fall den generiska kartans *struktur* stämmer överens med företagets syn på sin verksamhet är det inte säkert att alla färdighetsnamn gör det.

Var och en av dessa färdigheter bryts så vidare ned i lägre och lägre nivåer tills man kommer till den nivå man uppfattar som den *primitiva nivån*. Det är på den nivån man övergår från abstraktioner till verkligt handlande. Det är där man accepterar order och lägger produktionsorder, det är där man auktoriserar betalningsmetoder, det är där man beräknar rabatter och det är där man till exempel vid leveransförseningar informerar kunden om ny beräknad leverans- och ankomsttid.

Alla dessa aktiviteter, som var och en kräver sin egen färdighet, har jag hämtat från den övergripande färdighet att kunna sälja som företaget måste besitta. Figur 5 är ett utklipp ur en grafisk presentation av ett

generiskt företags uppsättning av färdigheter. Den färdighet jag har zoomat in är färdighet #2.3 Sales, som i diagrammet är nedbrutet till den femte nivån. För färdigheten Sales är det denna femte nivå som är den primitiva nivå jag nyss nämnde



Figur 5 - Färdighet 2.3 Sales nedbruten till femte abstraktionsnivån

Verktyg att vänta

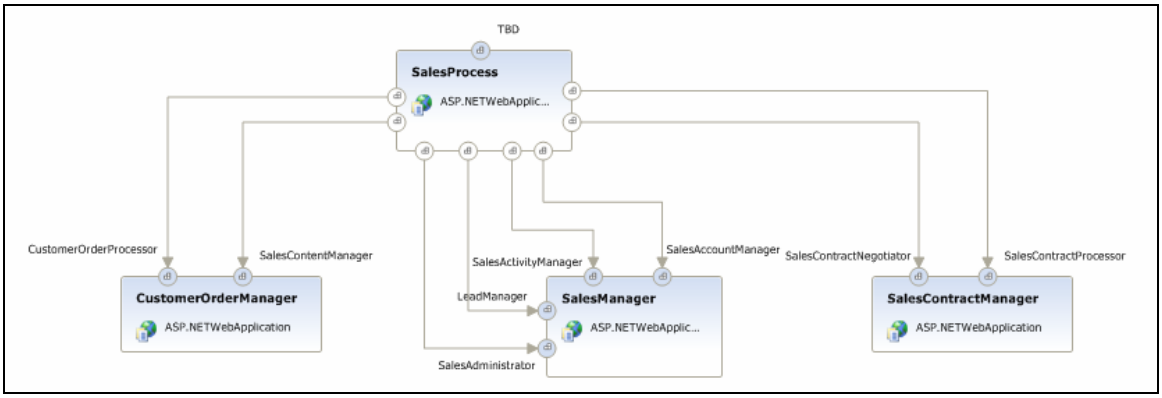
Räkna med att Microsoft kommer att leverera ett verktyg, en metodologi och en generisk *capability map* (eller *module map* som den också kallas). Räkna också med att detta verktyg blir möjligt att plugga in i Team Systems infrastruktur för designverktyg.

Tanken är att du i samverkan med företagsarkitekten skall kunna utgå från den generiska kartan, som i den version vi har tillgång till omfattar exakt 800 capabilities. Utifrån den skall du för ditt företag eller din kund kunna upprätta en karta över just det företagets nödvändiga uppsättning av capabilities.

Är du konsult kan du också etablera en branschspecifik karta över färdigheter. Har du till exempel i samarbete med en bank åstadkommit en capability map för denna bank bör du kunna generalisera den så att den blir en bra utgångspunkt för andra banker, vilket åtminstone på bankmarknaden bör ge dig en konkurrensfördel över andra konsulter

Utgångspunkt för etablerandet av tjänster

Kartan är en enastående utgångspunkt för etablerandet av tjänster enligt SOA-modellen. Ta en ny titt på färdigheterna i Figur 5 och ställ för var och en frågan om den skulle kunna vara utgångspunkt för definitionen av en tjänst eller en web service som kan fungera som programmeringsgränssnitt för en tjänst.



Figur 6 - Möjliga Process & Activity Services, baserade på Capability Map

Figur 6 visar ett diagram ritat med den Application Designer som blir en del av Microsoft Visual Studio 2005 Team System. Diagrammet ger en del av det svar på frågan *vi* kom fram till. Du kanske kommer fram till ett annat svar, men det är inte viktigt för resonemanget. Det jag i artikeln vill göra är att visa en möjlig serviceorienterad arkitektur som är baserad på den capability map vi nyss har tittat på.

Vi valde att etablera en processtjänst som driver hela säljprocessen. Den använder sig av (i exemplet) tre aktivitetstjänster – i verkligheten skulle det vara ytterligare några. Om du jämför tjänsten SalesProcess med diagrammet i Figur 5 finner du att tjänsten *SalesProcess* är tänkt att implementera capability 2.3 Sales och att tjänsten *CustomerOrderManagement* är tänkt att implementera capability 2.3.1 Order Management. Det framgår inte av det urklipp som finns i Figur 5, men tjänsten *SalesManager* motsvarar capability 2.3.2 Sales Management medan tjänsten *SalesContractManager* motsvarar capability 2.3.5 Contract Management.

En ny titt på Figur 5 visar att capability 2.3.1 Order Management är nedbruten i två capabilities på lägre nivå: Process Customer Order och Content Management. Dessa båda capabilities motsvaras av var sin *endpoint* som kan ta emot meddelanden avsedda för tjänsten CustomerOrderManagement. Det ser du i den nedre vänstra delen av Figur 6. Dessa båda endpoints kommer att implementeras som var sin web service.

Hur blir det då med de primitiva färdigheterna som syns i Figur 5? Jo, de blir webbmetoder som ingår i den web service som implementerar deras övergripande färdighet. Färdigheten *Accept Orders* blir webbmetoden *AcceptOrders,* och den kommer att exponeras av webbtjänsten CustomerOrderProcessor.

Tanken med det här avsnittet var egentligen inte att gå så här långt, men jag kunde inte låta bli. Jag stannar emellertid här nu, men jag lovar att återkomma till allt detta i en senare artikel som kommer ut på vår webb och på Microsofts arkitektwebb under kvartalet april-juni 2005.

Conceptual Information Viewpoint

Den konceptualla nivån – se Figur 7 – är den nivå som mest berör lösningsarkitekten. Det är här arkitekten – eventuellt i samverkan med verksamhetsanalytiker – utreder kraven på det tänkta systemet. Det är också här lösningsarkitekten etablerar och dokumenterar den övergripande arkitektur av tjänster, servicegränssnitt (”endpoints”), servicemetoder och scheman som behövs för en effektiv lösning av problemet.

	Mission Viewpoints	Information Viewpoints	Process Viewpoints	Infrastructure Viewpoints	Security Viewpoints
Conceptual Viewpoints	e.g. User Profiles (refined) Solution Vision	e.g. Conceptual Information Model Bsns Rules Model Entity Services Service Interfaces Service Interaction Message Schemas	e.g. Bsns Process Models Use Cases Functional Features Data Use Cases Presentation Services Process/Activity Services Service Interfaces Service Interaction Message Schemas	e.g. Operational Features Service Deployment	e.g. External Security Architecture

Figur 7 - Conceptual Viewpoints med Conceptual Information Viewpoint i fokus

En tidig aktivitet i ”the Conceptual Information View” är att etablera en konceptuell informationsmodell. Den är konceptuell eftersom den utgör en dokumentation av den vokabulär verksamhetens folk använder för att beskriva begrepp och förhållanden i verksamheten, och eftersom den (helst) gör det utan att binda element i denna vokabulär till någon fast struktur som tabeller, klasser eller liknande.

Vi använder gärna Terry Halpins Object Role Modeling (ORM) för detta ändamål. Det är en metod som har verktygsstöd i Microsoft Visio idag och kommer att få det i Visual Studio 2005 Team System framöver. Det är inte Microsoft som tar fram detta verktygsstöd. Det är i stället en grupp människor, ledda av Terry Halpin, som gör det. De gör det för Open Source, och de gör det för Whitehorse som är Microsofts interna kodbenämning på det projekt som tar fram infrastrukturen för Microsofts kommande designverktyg i Team System.

Det finns flera skäl till att föredra ORM framför metoder för klassmodellering, entitetsmodellering eller datamodellering. Jag tänker inte räkna upp dem här, men jag hoppas att de exempel jag väljer för att visa ORM i denna artikel ändå skall i någon grad förmedla dessa skäl.

Attributfritt

ORM arbetar med termer som representerar informationsobjekt och med fakta som säger något om en eller flera termer. ORM erkänner inte begreppet attribut utan ser alla termer och objekt som likvärdiga. Det betyder att du kan skjuta frågan om vilka informationsobjekt som är attribut, och vad de i så fall är attribut till, till ett senare skede, nämligen till designskedet där sådana frågor egentligen hör hemma. Du kan alltså nöja dig med att analysera och dokumentera informationsobjekt och informationssamband utan att behöva binda dem till en användbar struktur som en tabellstruktur, entitetsstruktur eller klasstruktur.

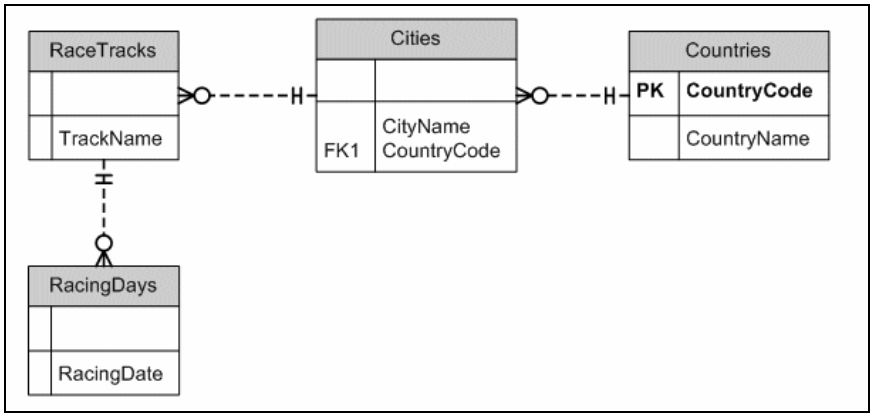
Möjligheten att skjuta upp bindning till en ”användbar struktur” är bra för dig som arkitekt, för du kan skjuta upp designbeslut till designtillfället. Det är också bra för dina uppdragsgivare, för de kan koncentrera sig på informationsstrukturerna som sådana utan att behöva tyngas av implementationsstrukturer.

Låt mig bara ta ett mycket enkelt exempel från det hästkapplöpningssystem vi brukar exemplifiera våra alster med. Kunder som använder det systemet har användning av information om kommande tävlingsdagar till vilka det finns startlistor. Denna information presenteras i tabellform, och kundens vy över denna information är, som synes av exemplet i Tabell 1, superenkel:

Datum	Bana	Stad	Land
2005-05-01	Täby Galopp	Stockholm	Sverige
2005-05-03	Jägersro Trav & Galopp	Malmö	Sverige
2005-05-05	Øvrevoll Galopp	Oslo	Norge
2005-05-07	Køpenhavns Galopbane	Køpenhavn	Danmark

Tabell 1 - Förteckning av kommande tävlingsdagar.

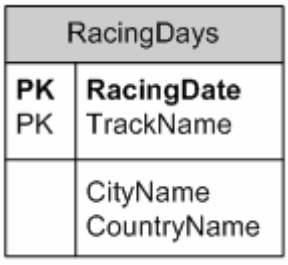
Med en attributbaserad designmetod, som logisk datamodellering, entitetsmodellering eller klassmodellering, räcker det inte med att du analyserar de informationsobjekt och de relationer mellan dem som är inblandade i produktionen av en sådan tabell. Du måste också fastställa vilka av dem som är att betrakta som objekt av något slag (tabeller, klasser, eller entitetstyper) och vilka som är att betrakta som attribut till något av dem.



Figur 8 - datamodell för tabell med tävlingsdagar

Den logiska datamodellen i Figur 8 visar hur det skulle kunna se ut. Jag har försökt att göra den så realistisk som möjligt, så jag har ännu inte bestämt hur galoppbanor, tävlingsdagar eller städer skall identifieras. Med länder är det i galoppsammanhang självklart att det finns en landskod, så den använder vi och tar som primärnyckel – med de andra primärnycklarna väntar vi.

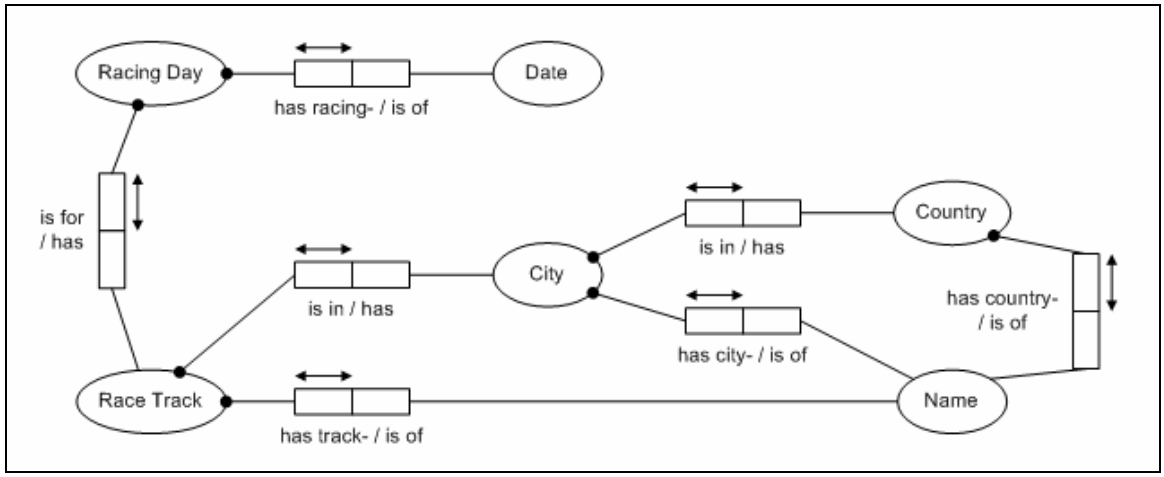
Diagrammet i Figur 8 är emellertid bara en tänkbar lösning på problemet att redovisa en informationsstruktur som den i Tabell 1. Diagrammet i Figur 9 innehåller bara en enda tabell, men utgör ändå en korrekt struktur för den information som kan generera förteckningen i Tabell 1. Skillnaden är att vi här bara har utgått från det vi verkligen vet, det vi kan läsa oss till i förteckningen av kommande tävlingsdagar, medan vi i Figur 8 gjort antaganden vi av användarvyn inte kan sluta oss till.



Figur 9- Enklare modell för implementering av förteckningen i Tabell 1

Poängen med det här enkla exemplet är att en attributorienterad modelleringsansats ständigt tvingar dig att ta ställning till problem du ännu inte har underlag nog att ta ställning till. De tidiga strukturer du bygger upp kommer att påverkas av tillkommande vyer. Den tilläggsinformation du får då kan mycket väl stå i konflikt med de strukturer du redan byggt upp, och då bör du i allmänhet ändra dessa strukturer. Så småningom tar det emot, och du tenderar att ”pressa in” den nya informationen i befintlig struktur.

Det är bättre att vänta med bindning till användbar struktur tills du hittat, analyserat och dokumenterat alla informationsobjekt och alla relationer dem emellan. Du kan då koncentera dig på att hitta alla informationselement och att hitta alla relationer *utan* att behöva fundera på hur de skall implementeras.

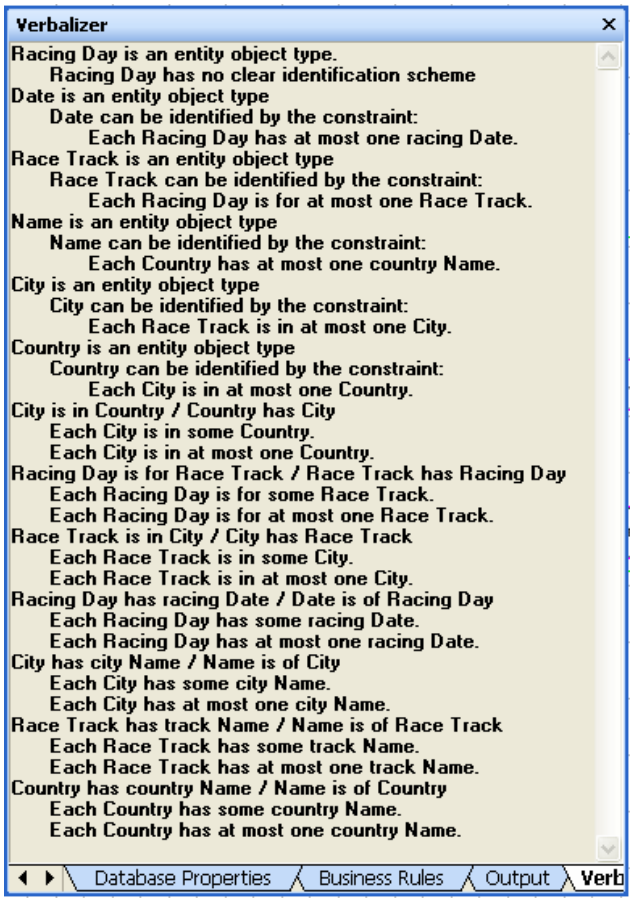


Figur 10 - ORM-diagram för samma informationsobjekt

ORM-diagrammet i Figur 10 visar att du kan modellera samma uppsättning informationsobjekt och deras inbördes relationer utan att behöva fundera över vilka objekt som skall bli modeller eller klasser och vilka som skall bli attribut till dem. ORM är en helt attributfri metod för informationsanalys och informationsmodellering. Med ORM kan du vänta tills informationsanalysen är klar med att etablera en ”användbar struktur”, det vill säga en datamodell eller en klassmodell.

Det är inte bara ORM som är en attributfri modelleringsmetod. ORM ingår i en grupp metoder som karaktäriseras av att vara *faktabaserade* snarare än attributbaserade. De karaktäriseras också av att kunna *verbalisera* en informationsstruktur, så att domänexperter kan ta ställning till verbala påståenden och inte bara till grafiska diagram.

Figur 11 visar hur det ORM-verktyg som finns med i Microsoft Visual Studio 2003 Enterprise Architect Edition – Visio-delen – har verbaliserat det diagram som visas i Figur 10.



Figur 11- Verbalisering av diagrammet i Figur 10.

Lägg märke till att verbaliseringens övre del omfattar de informationsobjekt som ingår i urvalet. Längre ner – från *City is in Country / Country has City* – kommer diagrammets fakta.

Ett faktum säger något om minst ett av modellens informationsobjekt; oftast knyter ett faktum ihop två eller flera informationsobjekt och säger något om relationen mellan dem.

ORM låter dig också notera restriktioner i sambandet mellan två informationsobjekt. Så till exempel kan du i verbaliseringen läsa att:

- *Each City is in some Country*
- *Each City is in at most one Country*

Av dessa uttalanden kan du sluta dig till att varje stad finns i exakt ett land. (Skulle det finnas någon stad som finns i mer än ett land har du med modellen sagt att du inte tar hänsyn till det.) Du behöver därmed ingen struktur som tillåter en stad att kopplas till två länder. Det du däremot behöver är en restriktion som förhindrar att en stad läggs till utan angivande av det land staden tillhör.

ORM för Team System

Jag vet att jag redan har sagt det, men jag upprepar det gärna. Under ledning av Terry Halpin och i samarbete med Microsoft håller Team System på att utveckla ett nytt ORM-verktyg (ORM 2005). De gör det för Open Source, och det skall kunna pluggas in i Microsoft Visual Studio 2005 Team System.

Det finns också en god chans att ORM-verktyget och verktyget för capability mapping skall kunna samverka med varandra, åtminstone på sikt. Det vore värdefullt, för capabilities har informationsbehov och därmed informationsvyer. En av uppgifterna för en informationsmodell är att kunna tillgodose alla kända informationsvyer. Därför vore ett giftermål mellan dessa båda verktyg av stort värde. Vi vet att inflytelserika drivkrafter i båda projekten delar denna uppfattning, och vi vet att de har bestämt sig för att prata med varandra om det. Vi ser med spänning fram mot resultatet av dessa samtal.

Entitetstjänster

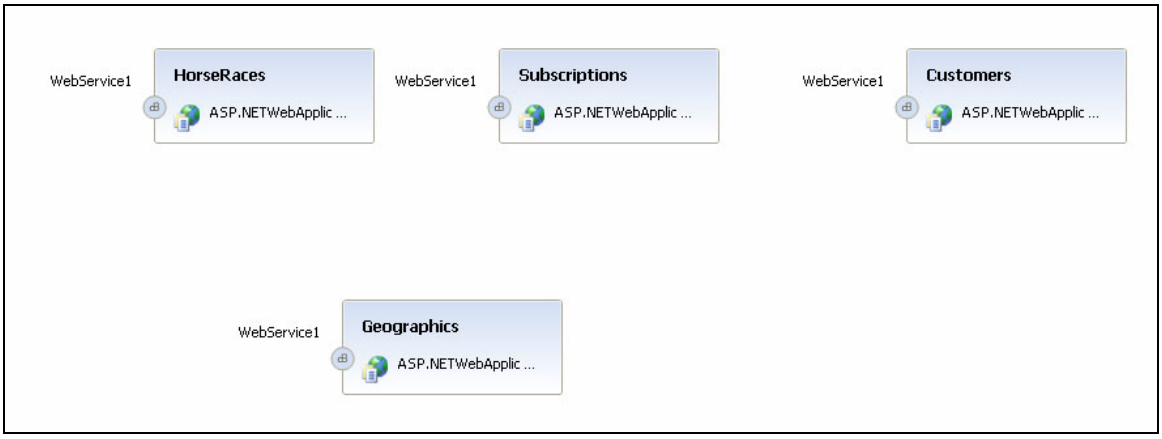
En av lösningsarkitektens viktigaste uppgifter i en SOA-miljö är att tillsammans med företagsarkitekten bestämma hur företagets informationsobjekt skall delas in i entitetstjänster.

En entitetstjänst ansvarar för en sammanhängande mängd entitetstyper och entiteter. Entitetstjänsten är den enda auktoriteten på data och information som rör de entitetstyper tjänsten ansvarar för. En sådan tjänst innehåller hela det regelverk tjänstens entiteter måste rätta sig efter. Alla förändringar i form av nyinläggning, modifiering eller borttag av dataobjekt måste beställas hos den entitetstjänst som svarar för informationen i fråga, och tjänsten ansvarar för att inget data som inte uppfyller regelverket kommer in i datakällorna.

Verksamhetens folk har oftast en god uppfattning om hur den information företaget arbetar med kan delas in i informationsområden. Frågar du till exempel en marknadsansvarig vilka informationsområden han eller hon befattar sig med får du antagligen svar som ”kunder, produkter, order och leveranser”. Sådana informationsområden kan tjäna som utgångspunkt för identifikation av entitetstjänster. Mer detaljerad informationsanalys, till exempel med ORM, kan ge ytterligare vägledning. Både övergripande synpunkter på vilka informationsområden som finns och mer detaljerade överväganden om hur informationen bör struktureras kan alltså påverka indelningen i entitetstjänster.

Visual Studio 2005Team Systems Application Designer är ett utmärkt verktyg för att utforma entitetstjänster. Vi har tidigare visat hur du kan använda det för att modellera process- och aktivitetstjänster; nu skall vi visa hur du kan använda det för att modellera entitetstjänster. Lägg märke till att det problem vi *nu* försöker lösa är ett annat än problemet att implementera säljprocessen.

Figur 12 visar hur lösningsarkitekten har identifierat fyra entitetstjänster, behövliga för lösandet av ett specifikt problem.



Figur 12 - Entity Services som tillhandahåller information och data

Varje ruta i diagrammet representerar en entitetstjänst. Lägg märke till att var och en av diagrammets tjänster har ”dekorerats” med en så kallad *endpoint.* Dessa endpoints – en till varje entitetstjänst – har det gemensamma namnet WebService1.

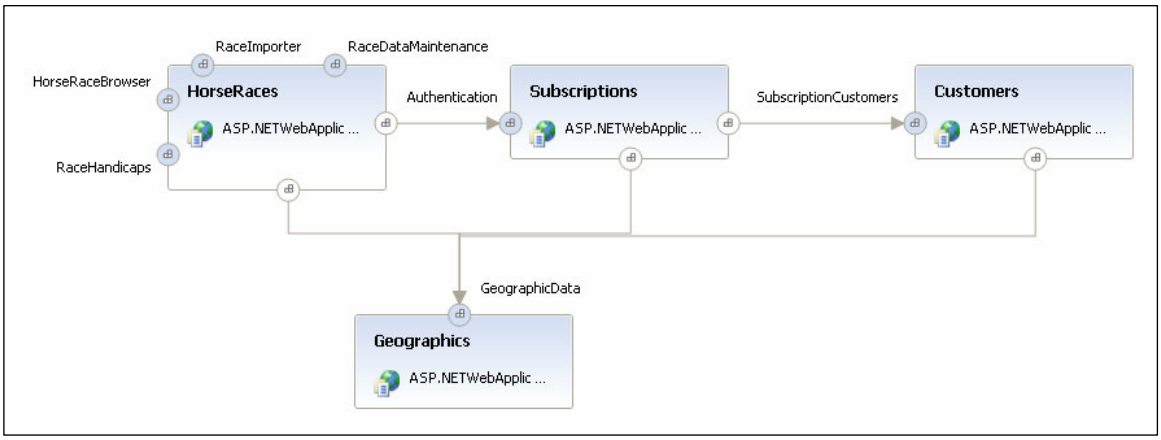
Det är inte lösningsarkitekten som har bestämt att det skall vara så; När du lägger ut en Web Service Application i ett diagram får den automatiskt en web service endpoint. (En endpoint är en adress till vilken man kan sända meddelanden. Det som finns bakom en endpoint är vederbörandes ensak, medan en endpoint

måste vara väl definierad för den konsument som skall använda den.) Som diagramförfattare kan du sedan döpa om och konfigurera varje sådan web service så att den gör exakt det du vill att den skall göra.

Steve Swartz är en av arkitekterna bakom Indigo, som ju är Microsofts nästa plattform för utveckling och exekvering av lösningar i allmänhet och servicebaserade lösningar i synnerhet. Steve säger att en typisk service inte exponerar ett servicegränssnitt utan många. Hans exakta formulering var att ”each service typically has a bunch of endpoints”.

Maarten Mullender är en av de mest framstående arkitekter på Microsoft som har till uppgift att utarbeta arkitektoniska idéer som alla vi Microsoftkunder kan dra fördel av. Han vill helst ha ett eget servicegränssnitt till varje konversation. Vi på Sundblad & Sundblad har länge talat om separata servicegränssnitt per användningsfall, men Maarten går alltså ännu längre. ”På det sättet isolerar jag varje konversation från varje annan konversation”, säger Maarten och fortsätter: ”Detta oberoende mellan konversationer gör att tjänsten i fråga blir lättare att underhålla och vidareutveckla, något som får större betydelse i en serviceorienterad miljö än det haft i traditionell objektorientering.”

Figur 13 visar hur jag som lösningsarkitekt etablerat ett antal servicegränssnitt, där varje gränssnitt har sin specifika uppgift i förhållande till respektive tjänsts konsumenter. Jag har också ritat ut de kommunikationsvägar som finns mellan diagrammets tjänster.



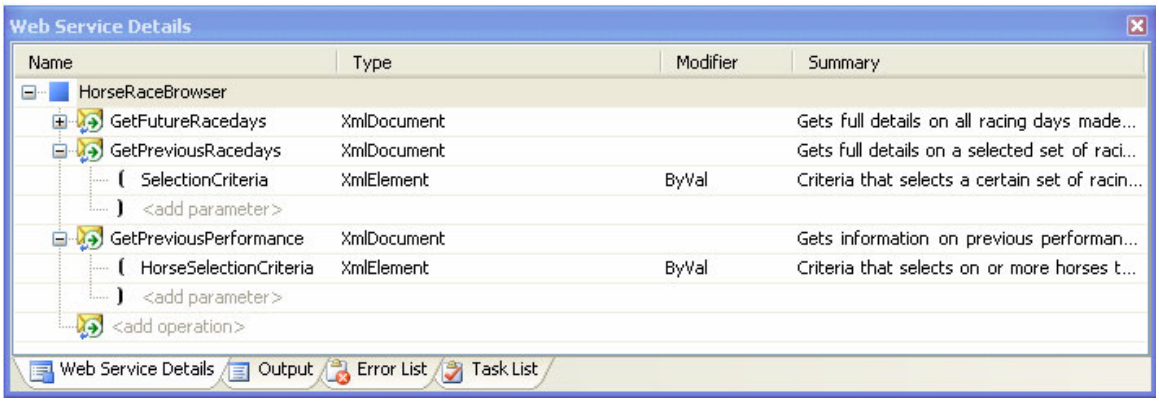
Figur 13 – Web Services Endpoints och kommunikation med tjänster nu tillagda

Ta en titt på den entitetstjänst som ligger längst till vänster i diagrammet. Vi har kallat den för HorseRaces, och den ansvarar för all data som finns om själva kapplöpningsverksamheten i vårt exempelfall. Här finns uppgifter om galopphästar, tränare och ryttare. Här finns också uppgifter om kommande lopp och deras startlistor samt om historiska lopp och deras resultatlistor.

Förutom att vara exponent av servicegränssnitt är HorseRaces också konsument av andra servicegränssnitt. HorseRaces känner till exempel inte till vilka potentiella användare som har giltiga prenumerationer till de tjänster HorseRaces exponerar. Det är bara tjänsten Subscriptions som gör det. Därför exponerar Subscriptions servicegränssnittet *Authentication*. När en användare startar en session med HorseRaces (via ett webbformulär som ännu inte syns i diagrammet) sänder HorseRaces ett meddelande om detta till Authentication och frågar därmed om vederbörande har en giltig prenumeration.

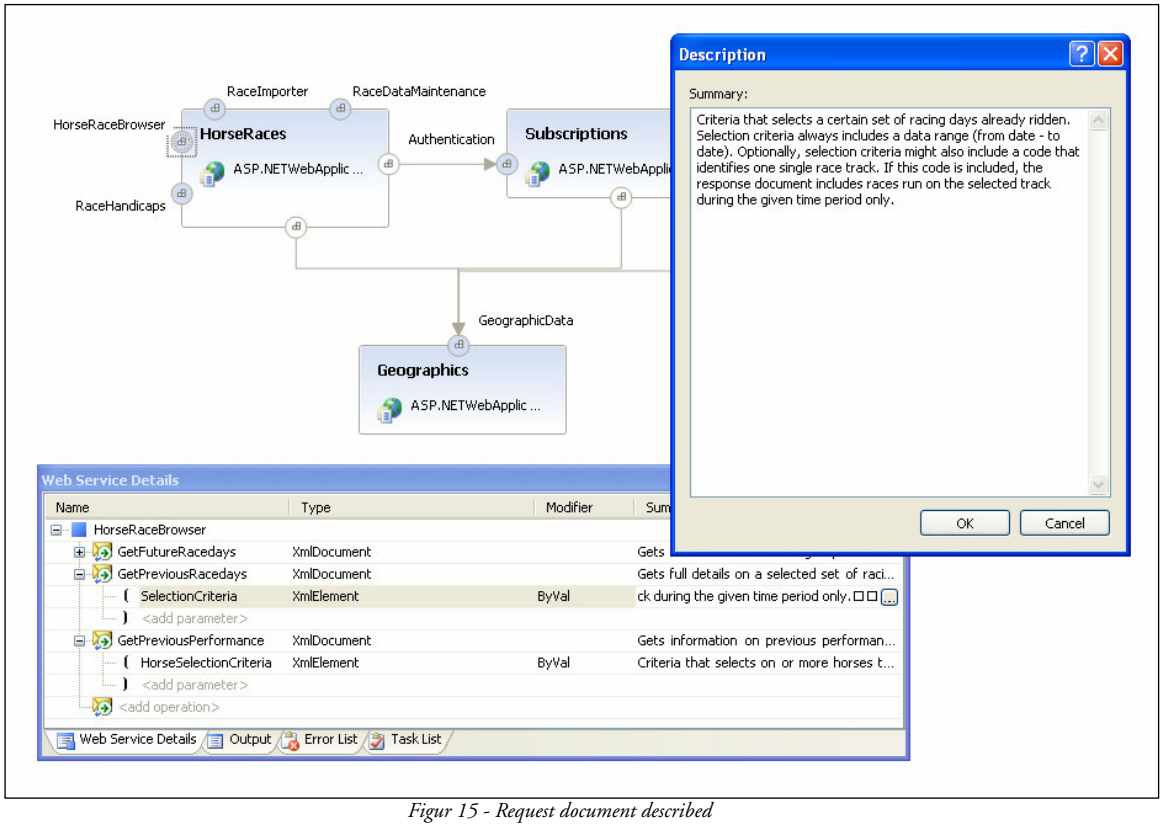
När du som lösningsarkitekt är färdig med ett diagram som det i Figur 13 genererar Application Designer det kodskal som behövs för att implementera diagrammets arkitektur. Application Designer håller sedan via synkroniseringsmekanismer kod och diagram i ständig enighet. Relevanta ändringar i koden påverkar diagrammet, och alla ändringar i diagrammet påverkar koden. Den i diagrammet utritade förbindelsen mellan HorseRaces och Authentication (i Subscriptions) leder automatiskt till att HorseRaces får en webbreferens till Authentication.

Konsumenter av tjänsten HorseRaces ser emellertid inte hela tjänsten utan endast det eller de servicegränssnitt konsumenten använder sig av. Ett av dessa gränssnitt heter *HorseRaceBrowser.* Figur 14 visar hur lösningsarkitekten har försett HorseRaceBrowser med ett antal operationer (som leder till webbmetoder i koden). Varje operation har en returtyp och – om relevant för operationen – också en anropsparameter. I vårt exempel är både returtyp och anropsparameter av dokumenttyp.



Figur 14 - Web Service Endpoints definierade och beskrivna.

Application Designer låter dig också textuellt beskriva såväl servicegränssnitt som operationer och parametrar. Figur 15 visar ett exempel på det. Dessa beskrivningar hamnar automatiskt i koden, antingen via initial generering eller via den synkronisering som efter initial generering ständigt håller kod och diagram i synk med varandra.



Figur 15 - Request document described

Conceptual Process Viewpoint

Hittills har vårt diagram endast omfattat entitetstjänster. Vi har i artikelns exempel valt att börja i den ändan, eftersom entitetstjänster ger en stabil grund för hela arkitekturen som sådan, men vi skulle lika gärna ha kunnat börja med presentationstjänster eller processtjänster.

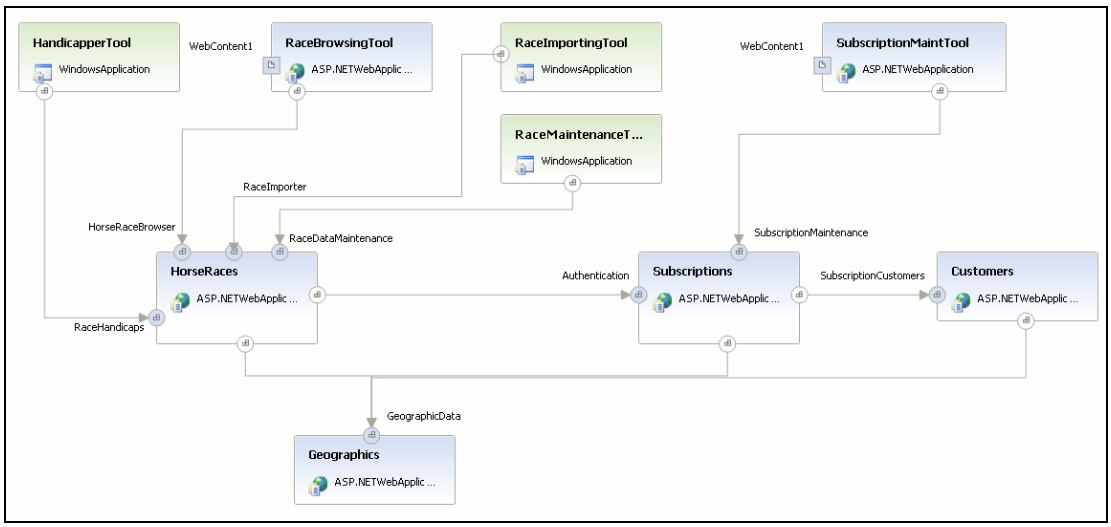
Vårt exempel omfattar inga process- eller aktivitetstjänster. Den aktuella verksamheten är inte så process-orienterad utan mer orienterad åt informationssökning. Därför valde vi ett arkitekturmönster som endast omfattar presentationstjänster och entitetstjänster. Den processorientering som ändå finns sköts antingen manuellt vid sidan om IT-systemet eller är inlagd i presentationstjänsterna. Vi ser alltså presentations-tjänsterna som en del av vårt ramverks *process viewpoints,* något som också framgår av Figur 16.

	Mission Viewpoints	Information Viewpoints	Process Viewpoints	Infrastructure Viewpoints	Security Viewpoints
Conceptual Viewpoints	e.g. User Profiles (refined) Solution Vision	e.g. Conceptual Information Model Bsns Rules Model Entity Services Service Interfaces Service Interaction Message Schemas	e.g. Bsns Process Models Use Cases Functional Features Data Use Cases Presentation Services Process/Activity Services Service Interfaces Service Interaction Message Schemas	e.g. Operational Features Service Deployment	e.g. External Security Architecture

Figur 16 - Conceptual Process Viewpoint

Klientapplikationer

Vi talar då och då om presentationstjänster som om det rörde sig om tjänster i samma betydelse som till exempel entitetstjänster, processtjänster och aktivitetstjänster, det vill säga SOA-baserade tjänster. Så är det emellertid inte. Enligt SOA är en tjänst en programenhet som endast kan nås via ett meddelandeorienterat servicegränssnitt medan det vi kallar presentationstjänster ju primärt nås via vanliga användargränssnitt. Benämningen presentationstjänst är alltså inte helt korrekt, men den är så bekväm att använda att vi ofta ändå gör det. En mera korrekt benämning vore förmodligen klientapplikationer.



Figur 17 - Klientapplikationer inlagda i systemet

Figur 17 visar hur vårt diagram ser ut när vi även lagt ut våra klientapplikationer. De som i diagrammet visas i en grönaktig färgton är Windowsapplikationer; de som visas i en blåaktig färgton är webbapplikationer och använder ASP.NET. Diagrammet ger nu en arkitektonisk översiktsbild av hela galoppsystemet. Genom att klicka på olika delar av diagrammet kan du borra dig ner och få en mer detaljerad bild av det du klickat på. Du kan välja en web service endpoint och se hur den skall vara konfigurerad. Du kan också se vilka operationer den exponerar och vilken signatur som gäller för var och en av dess operationer. Allt är också – om du en gång har genererat kod från diagrammet – manifesterat i en .NET-lösning som ständigt och automatiskt hålls synkroniserad med diagrammet.

Om du är nöjd med den lösningsarkitektur du har skapat, och om du är övertygad om att den uppfyller alla krav på lösningen, skulle du nu kunna överlämna den till en eller flera grupper av servicearkitekter eller servicedesigners, vad du nu väljer att kalla den som är ansvarig för den övergripande strukturen på *insidan* av en tjänst eller en applikation.

Men först kanske du borde undersöka om de egenskaper du definierat för systemets tjänster överensstämmer med de restriktioner som byggts in i den tekniska infrastruktur du tänker installera systemet på. Det betyder oftast att du måste förhandla med en eller flera infrastrukturarkitekter.

Conceptual Infrastructure Viewpoint

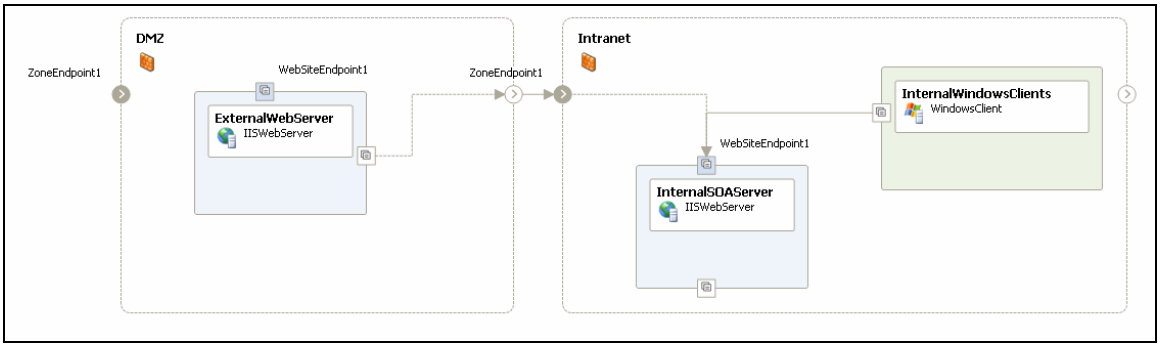
Som framgår av Figur 18 hamnar du därmed i Conceptual Infrastructure Viewpoint.

	Mission Viewpoints	Information Viewpoints	Process Viewpoints	Infrastructure Viewpoints	Security Viewpoints
Conceptual Viewpoints	e.g. User Profiles (refined) Solution Vision	e.g. Conceptual Information Model Bsns Rules Model Entity Services Service Interfaces Service Interaction Message Schemas	e.g. Bsns Process Models Use Cases Functional Features Data Use Cases Presentation Services Process/Activity Services Service Interfaces Service Interaction Message Schemas	e.g. Operational Features Service Deployment	e.g. External Security Architecture

Figur 18 - Conceptual Infrastructure View och Viewpoint

Design av logiskt datacenter

Visual Studio Team System 2005 innehåller en Logical Data Center Designer. Med hjälp av detta verktyg kan du – eller kanske snarare infrastrukturarkitekten – rita upp en bild av den tekniska infrastruktur du har till förfogande för installation av din lösning. Figur 19 är ett enkelt exempel på ett sådant diagram.



Figur 19 - Logical Data Center Designer

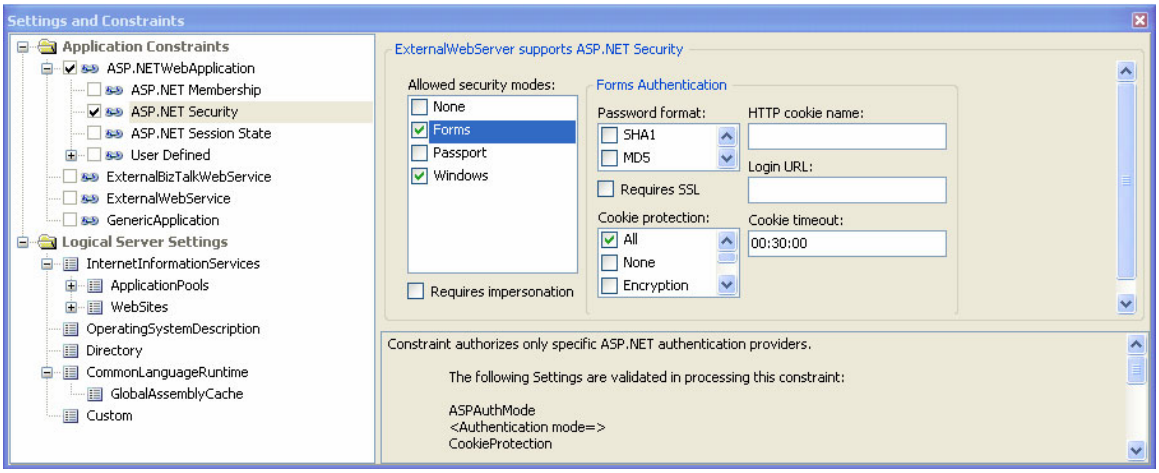
Figuren visar två zoner, som var och en skyddas av sin egen brandvägg. Den längst till vänster belägna zonen är den demilitariserade zonen (DMZ); den längst till höger är den interna och mera skyddade intranätzonen (Intranet).

I den demilitariserade zonen finns endast en logisk maskin. Det är en extern webbserver som har Microsoft Internet Information Server (IIS) som värd. Denna värd kan inhysa såväl web services som gamla hederliga formulärbaserade webbapplikationer.

Intranätzonen innehåller två logiska servrar. Den ena har IIS som värd; den andra har Windows som värd.

Begreppet ”logisk server” innebär att en av diagrammets servrar mycket väl kan komma att representeras av ett stort antal servrar. ExternalWebServer kan till exempel vara en webbfarm som består av 10 servrar eller fler. Det innebär också att de båda logiska servrar som finns i intranätet mycket väl kan vara en enda fysisk server som i så fall har både Windows och IIS som värdar.

Diagrammet visar också vilka kontaktvägar som skall vara tillåtna. ExternalWebServer skall till exempel kunna sända meddelanden till intranätzonen, och dessa meddelanden skall delegeras till den logiska server vi kallat InternalSOAServer, som i sin tur skall kunna ta emot meddelanden från InternalWindowsClient.



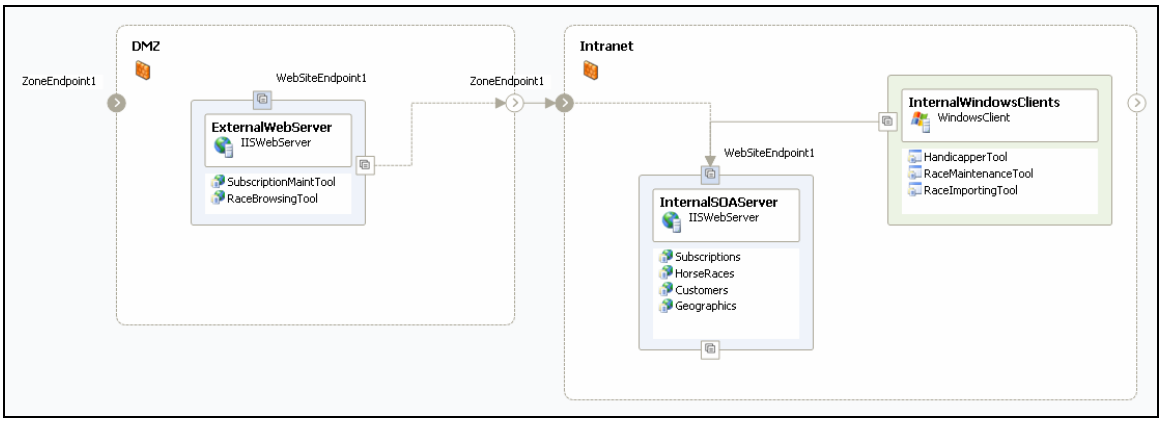
Figur 20 - Restriktioner för ExternalWebServer

Figur 20 visar hur diagrammets servrar också kan konfigureras. Vi har valt att visa säkerhetsaspekten, men som framgår av figuren finns det också andra aspekter som kan konfigureras med hjälp av Logical Data Center Designer.

Deploymentförslag

Nu finns det i projektet en bild av hur lösningen ser ut och vilka krav den ställer på den tekniska infrastrukturen. Det finns en annan bild av den tekniska infrastrukturen samt information om vilka restriktioner den sätter upp för lösningen. Frågan är om de båda bilderna är kompatibla med varandra.

I allmänhet får man vänta med svaret på den frågan tills lösningen är klar och kan installeras för stor systemtest. I Visual Studio Team System finns det en Deployment Designer som låter dig undersöka det redan innan du skrivit någon kod alls. Figur 21 visar hur var och en av lösningens komponenter – som ju är tjänster och klientapplikationer – funnit sin plats i den tekniska infrastrukturen.



Figur 21 - Ett deploymentförslag som godkänts.

Om bara lösningsarkitekten har beskrivit tjänsternas och klientapplikationernas krav på den tekniska infrastrukturen på ett korrekt sätt, och om infrastrukturarkitekten beskrivit den logiska infrastrukturens tekniska krav på lösningen korrekt, då kommer också deployment att fungera den dagen vi kommer dit. I varje fall om inga sådana förändringar som påverkar situationen har genomförts.

Skulle till exempel kraven på säkerhet för en av de tjänster som skall installeras på InternalSOAServer medföra behov av delegation, medan InternalSOAServer inte accepterar delegation, då får lösningsarkitekten och infrastrukturarkitekten reda på det redan innan lösningen börjat implementeras. Deploymentförsöket kommer då att misslyckas redan i diagrammet. Kostnaden för att genomföra behövliga ändringar av arkitekturen blir då obetydliga jämfört med om problemet identifierats först när hela systemet och alla dess tjänster redan var implementerade.

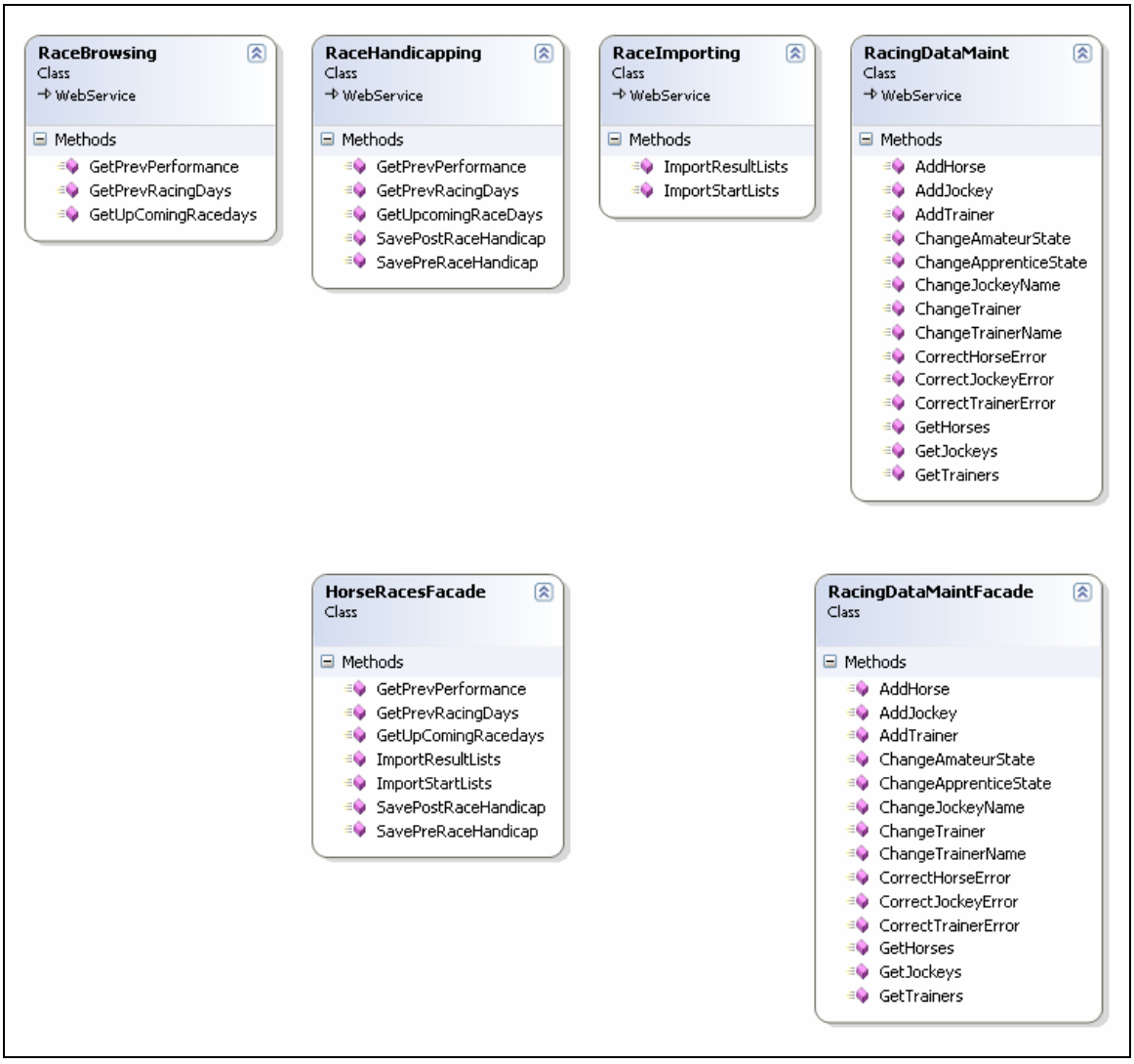
Logiska vyer

Vi skall avsluta den här artikeln med en titt på den Class Designer som en servicearkitekt eller servicedesigner skulle kunna tänkas behöva. Vi hamnar då på nästa nivå i vårt arkitektoniska ramverk, den nivå där vi har våra *Logical Viewpoints.*

	Mission Viewpoints	Information Viewpoints	Process Viewpoints	Infrastructure Viewpoints	Security Viewpoints
Logical Viewpoints	e.g. Use Case Selection for Service Service Vision	e.g. Logical Entity Service and Data Model (Canonical)	e.g. Ref Architecture Logical Service Model Class Model	e.g. Component Deployment	e.g. Security Model

Figur 22 - Logical Viewpoints

Figur 23 visar ett exempel på ett klassdiagram, ritat med Team System Class Designer.



Figur 23 - Exempel på klassdiagram, upprättat med Team System Class Designer

Det finns ett problem med Team Systems Class Designer, men efter att ha börjat använda den ser vi det inte med lika stort allvar som vi gjorde när vi först uppmärksammade det. Problemet är att Class Designer endast stödjer två slag av relationer mellan klasser, arv och association. ”Usage relationship”, där relationen är sådan att metoder i en klass använder metoder i en annan, stöds inte i den första version som kommer ut av verktyget. I vårt diagram är den saknade relationstypen faktiskt den vi i första hand skulle haft bruk för.

Det finns ett bra skäl till denna brist. En absolut förutsättning för Team Systems designverktyg har varit fullständig och ständig samstämmighet mellan kod och diagram. Arv och associationer är relationer mellan klasser på klassnivå medan ”usage” är en relation mellan metoder i en klass och metoder i en annan. Förhållandet på *klassnivå,* vilket är det klassdiagrammen visar, mellan två klasser där objekt av den ena klassen använder objekt av den andra klassen kan vara av flera olika slag, vilket framgår av följande tabell:

Förhållande mellan klasser	**Förklaring**
En-till-en	En metod i klass A använder en metod i klass B.
En-till-flera	En metod i klass A använder flera metoder i klass B.
Flera-till-en	Flera metoder i klass A använder en metod i klass B.
Flera-till-flera	Flera metoder i klass A använder flera metoder i klass B.

Att finna ett sätt att beskriva dessa förhållanden för en usage-relation, att sedan generera kod för den, och att dessutom hålla denna kod och diagrammet synkroniserade med varandra, är minst sagt utmanande. Långa diskussioner har förts inom den grupp som utvecklat klassdesignern, men hittills har de lett till att

relationstypen inte stöds. Vi har varit något inblandade i dessa diskussioner, och vi tror att det kan komma stöd för usage relationships, men vi tror inte att det kommer redan i Visual Studio 2005. Den som lever får se.

Referenser

Jack Greenfield and Keith Short et.al., Software Factories, Wiley Technology Publishing. ISBN 0-471-20284-3

Terry Halpin, Information Modeling and Relational Databases, Morgan Kaufmann Publishers. ISBN 1-55860-672-6.

Terry Halpin et. al., Database Modeling with Microsoft Visio for Enterprise Architects, Morgan Kaufmann Publishers. ISBN 1-55860-919-9.

Rajesh Jain's Weblog on Emerging Technologies. http://www.emergic.org/archives/indi/003800.php

