RENDEZVOUS

NYHETSBLAD FÖR SESAM
Försvarssektorns Adaintressenters Användargrupp för Software Engineering

Nr 3 dec 1998

FRÅN RÅDETS MÖTE DEN 22 OKTOBER

Vid rådsmötet den 22 oktober valdes Björn Källberg, CelsiusTech Systems till ny ordförande i SESAM-rådet och Christopher Bengtsson, FMV till vice ordförande. Till VU omvaldes förutom dessa två Roger Brandt, Sune Ekfeldt och Billy Johansson.

Rådet beslutade också bl a att den nya arbetsgruppindelningen och det nya arbetssättet i princip enl grupp Wadsten förslag skall gälla från årsskiftet. Till ordförande i de nya grupperna utsågs Torbjörn Andreassson Ericsson Microwave Systems för Ag Teknik och Håkan Edler CTH för Ag Metodik.

Innehåll

Försvarssektorns Adaintressenters Användargrupp för Software Engineering

SESAM

Vad är SESAM?

SESAM har tillkommit för att organisera och stimulera samarbete och samverkan inom programvaruområdet mellan försvarsindustrin, FMV och FOA.

Det avtalsfästa syftet med SESAM är ”att genom organiserat samarbete mellan användargruppens medlemmar främja tillförlitlighet och effektivitet i utveckling och vidmakthållande av programvarusystem i Ada inom försvarssektorn”. Inom ramen härför skall SESAM även anpassa, profilera och förnya sin verksamhet med hänsyn till ändrade tekniska och andra omständigheter av betydelse för intresseområdet.

Följande kommer att ske under den närmaste 2-3-årsperioden.

1. SESAM skall allmänt verka för att sprida information om faktorer som påverkar möjligheterna till tillförlitlig och effektiv utveckling och vidmakthållande av programvarusystem. Särskilt skall härvid Adas betydelse i sammanhanget klargöras.

2. SESAM skall i sin verksamhet fortlöpande bevaka möjligheterna att samla, skapa och sprida information om objektiva mät- och andra resultat och erfarenheter vunna vid användning av ”software engineering”-principer och Ada.

3. SESAM behandlar tillvägagångsättet vid utveckling och vidmakthållande av programsystem. Implicit i detta ligger givetvis att använda processer skall tillförsäkra de resulterande produkterna efterfrågade egenskaper. Produktegenskaper som påverkas av processerna är därför av primärt intresse att bevaka i SESAMs verksamhet.

4. SESAM skall i sin verksamhet fästa stor vikt vid att underlätta samexistens mellan Ada-program och programvara skriven i andra språk. Speciellt skall aspekter vid användning av COTS beaktas.

5. SESAM skall där så är möjligt sätta konkretiserade och mätbara mål för sin verksamhet under avgränsade tidsperioder.

SESAM styrs av ett Råd med representanter för gruppens medlemmar. Rådet har till sin hjälp ett Verkställande Utskott (VU) och ett sekretariat.

Rådets ordförande är Björn Källberg, CelsiusTech Systems, tel 08-580 84813.

VU

Bengtsson Christopher, FMV
chben@tranet.fmv.se
Brandt Roger, FMV
robra@fmv.se
Carlsson Ingemar, FMV
ic@tranet.fmv.se
Ekfeldt Sune, Enator Telub AB
sune.ekfeldt@enator.se
Johansson Billy, CelsiusTech Electronics AB
bijo@celsiustech.se
Källberg Björn, CelsiusTech Systems AB
bjkae@celsiustech.se

Arbetet utförs i ett antal arbetsgrupper och följande är f n etablerade:

Process/Metrik
Göran Anger, Industrilogik
anger@L4i.se
Programmering
Magnus Ericson, Ericsson Saab Avionics
Magnus.Ericson@esavionics.se
Realtidssystem
Gilbert Kennedy, Saab Dynamics
gilke@weald.air.saab.se
Systemgrupp
Håkan Edler, CTH/Datorteknik
edler@ce.chalmers.se
Återanvändning
Vakant
Informationsutskottet

Vilka kan vara med i SESAM?

Medlemmarna i SESAM är svenska företag, organisationer och myndigheter (förvaltningar, utbildningsinstitutioner etc) med anknytning till försvarssektorn. Medlemmarna indelas i följande kategorier
- ordinarie medlemmar
- arbetsgruppsmedlemmar
- informationsmedlemmar.

Enskild person kan endast komma ifråga som informationsmedlem.

Inträde i SESAM

För samtliga medlemskategorier gäller att inträde beslutas av Rådet.

För inträde som ordinarie- och arbetsgruppsmedlem krävs status som leverantör till FMV. Dessutom krävs en skriftlig förbindelse att uppfylla åtagande som ordinarie- och arbetsgruppsmedlem.

För inträde som informationsmedlem (erhåller endast informationsbladet) krävs status som leverantör till FMV eller status som myndighet inom totalförsvaret. Rådet kan emellertid anta annan part som informationsmedlem.

För ansökan om medlemskap i SESAM vänd er till sekretariatet.

SESAM-Sekretariatet
Anna Kåsjö, FMV:INFOSYST, 115 88 STOCKHOLM
Tel: 08-782 6745, Fax: 08-66 77 392
Epost: alkas@tranet.fmv.se

Mer än 40 år i "försvarsmaterielbranschen"

Avgående Ordföranden har ordet

En turbulent tid ligger framför dem som verkar inom försvarsområdet. Hela försvaret omriktas och bantas. Det kommer förstås också att påverka både industri och förvaltning. Det innebär inte bara problem, även om de är dem man först ser, det kommer att öppna nya möjligheter också. Men det är nog tyvärrr oundvikligt med en svacka på många håll, innan man kan börja satsningen fullt ut mot vad som skisseras i Försvarsmaktsidé 2020 och målbilden för 2010. För att realisera dessa framtidsinriktningar kommer att behövas mycket programvara, väldigt mycket. SESAM kan spela en viktig roll i att medverka till att bygga upp de kunskaper och det samarbete som behövs för att våra medlemmar skall kunna vara med och ta fram framtidens försvarssystem.

Man kan säga att vi i SESAM har tagit ett första litet steg på den vägen, genom de projektförslag för forsknings- och teknikutvecklingsprogrammet för anpassning, som medlemmarna hjälpt till att ta fram. Uppdrag att genomföra sådana projekt har just börjat läggas ut från FMV. Engagerat genomförda, kan dessa projekt bli upptakten till ett mer utvecklat sätt att samarbeta mellan myndigheter och industri - mer av partnerskap - även i mera "skarpa" sammanhang.

Jag har haft tillfälle att i ett par omgångar den senaste tiden följa med besök från utlandet hos oss, såväl vid förvaltning, industri som förband. Av reaktionerna från besökarna förstår man att de blivit djupt imponerade, för att inte ibland säga chokade, av vad de sett och hört. Det finns viktiga områden där vi med våra systemlösningar ligger tio år före sådana utländska försvarsorganisationer, som eljest brukar anses teknologiskt helt världsdominerande. Det är inget dåligt utgångsläge inför framtidens utmaningar och de internationella samverkansprojekt som efterlyses på olika håll. Vi behöver inte stå med mössan i handen på den internationella arenan.

Och vi kan förstås inte heller stå stilla och klappa oss för bröstet. Vi vet själva bäst att vi också har tillkortakommanden som behöver rättas till och dessutom kan utvecklingen snabbt springa ifrån den som inte är observant. SESAM måste följa med utvecklingen; helst ligga litet före. Både innehåll och arbetsformer måste förnyas eftersom tiden går. Vi har haft problem med anslutningen i våra aktiviteter de senaste åren och vi behöver finna ämnen och arbetsformer, som gör det mer intressant och givande för medlemmarna att ställa upp, så att både de själva och deras företag prioriterar SESAM-deltagandet. Vi har haft en liten arbetsgrupp som funderat på vad som kan göras. Den har kommit med ett förslag, som skall diskuteras på ett stormöte i samband med höstseminariet. Om medlemmarna ställer upp på något i den riktningen som föreslås, kan det bli den "vitamininjektion" som behövs. Det är ingen brist på utmaningar inom det ständigt expanderande programvaruteknikområdet!

Innehållet i detta nummer av Rendezvous är inte lika brett som i det förra "Ada-Europe"-numret som också var avsett att vara litet PR. Men det Ni kan läsa om här tillsammans med föredragen vid höstseminariet, projektförslagen till FoTA-programmet och en hel del annat som SESAM-folk är inblandade i, visar att SESAM-kretsen ingalunda står och stampar.

Det här är mitt sista inlägg i denna min andra omgång som ordförande för rådet. Jag är lika övertygad nu, som jag var när jag för tio år sedan hade förmånen att få bli den förste rådsordföranden i SESAM, om att ett öppet och förtroendefullt kunskapsuppbyggande samarbete mellan programvaruintresserade i förvaltning och industri, som vi har i SESAM, är ett vitalt intresse för det svenska försvaret. Inget annat teknikområde har så stor betydelse för systemegenskaper och prestanda, flexibilitet, tidplaner och kostnader för försvarsmaterielen som programvaran. Inom inget annat teknikområde är både den individuella och kollektiva skickligheten så avgörande för resultatet.

Personligen närmar jag mig med raska steg den tidpunkt när det anses höra till att gå i pension. Därför har jag dubbel anledning att nu tacka för den tid jag haft förmånen att verka tillsammans med Er och för de många för mig enormt givande vänskapsrelationer, som jag blivit delaktig i. SESAM-tiden är på många sätt höjdpunkten i mina över 40 år i "försvarsmaterielbranschen".

Jag önskar min efterträdare på ordförandestolen och Er alla det allra bästa.

Ingemar Carlsson

Ordföranden har ordet

SESAM har inte bägge fötterna på jorden. Lyckligtvis. För den som alltid har bägge fötterna på jorden går inte framåt. Och SESAM går framåt nu.

Ett stort steg framåt är den omstrukture ring av arbetsgrupperna, som en grupp under ledning av Claes Wadsten föreslog, och som enhälligt antogs på rådsmötet den 22 oktober. Syftet med förslaget är att åstadkomma en effektivare verksamhet. Praktiskt sett innebär förslaget

- Endast två arbetsgrupper, Metodik respektive Teknik.
- Varje arbetsgruppsmedlem genomför ett mikroprojekt.
- Färre möten och mer eget arbete.
- Kortare beslutsvägar. VU kan direkt fördela medel

Hur det i praktiken går till framgår bl.a. av bifogade brev, som skickats ut till alla medlemmar i SESAM. Vi har alla en hemläxa, att se till att detta nya arbetsätt kommer igång direkt efter nyår: Rådsmedlemmar genom att säkerställa att det egna företaget ställer upp med resurser åt arbetsgruppsmedlemmarna, och arbetsgruppsmedlemmarna genom att ansluta sig till en arbetsgrupp, med minst ett förslag till ett sådant mikroprojekt. Förslaget skall vara konkret, och så intressant att man själv är villig att arbeta i mikroprojektet.

Ett annat viktigt steg är det nya avtal, som inom kort kommer att vara undertecknat av alla medlemmar. Det är mycket glädjande att i praktiken har alla företag som tidigare undertecknade avtalet återigen har valt att bli medlemmar. I flera fall har dock företagens namn ändrats, genom den omstrukturering som har skett inom försvarsindustrien. Jag hoppas också att de mindre företag, som är arbetsgruppsmedlemmar också kommer att fortsätta att verka inom SESAMS ram.

Dagens sista steg är dock inte ett steg framåt. Ingemar Carlsson har lämnat ordförandeposten i SESAM:s VU och formellt sett gått i pension. Jag vet att alla inom SESAM instämmer när jag säger.

TACK INGEMAR, för allt engagemang, arbete och intresse som du lagt ner på utvecklingen av programvaruområdet i allmänhet, och på högnivåspråk, Ada och SESAM i synnerhet. Utan dig skulle vi inte ha kommit så långt som vi gjort. Det hål som du lämnar blir omöjligt att fylla. Samtidigt hoppas jag att vi inte skall behöva fylla det helt, utan att du även i fortsättningen kommer att ha viss tid att hjälpa SESAM. Tack.

Björn Källberg

Rådet i SESAM fattade vid mötet den 22 oktober beslut om att effektivisera verksamheten i arbetsgrupperna. Förslaget innebar att endast två arbetsgrupper skulle finnas. Verksamheten i grupperna ska bli effektivare genom en större koncentration på eget arbete och färre möten. Arbetet i grupperna ska bättre anpassas till deltagarnas aktuella arbetsuppgifter. Därigenom blir arbetet i grupperna mer meningsfullt och det blir enklare att ge tillräcklig prioritet till arbetet i arbetsgrupperna. En enklare administration ska möjliggöra att arbetet i grupperna bättre anpassades till företagens nuvarande behov.

Enligt avtalet skall varje ordinarie medlem (företag) bidraga med 30 persondagar/år i arbetsgrupper, typiskt fördelat på 3 personer. Arbetsgruppsmedlem medverkar med 10 persondagar / år.

För varje person skulle detta kunna innebära två endagarsmöten och ett informationsspridningsmöte a två dagar, samt 6 dagar för eget arbete. Det egna arbetet skall ske inom ett mikroprojekt. Ett sådant projekt skall bedrivas av minst två personer från olika företag. Projektet kan t.ex. omfatta informationsinsamling inom ett område, egna mindre undersökningar, verktygsutvärderingar etc. Det är bara en fördel om projektet kan ses som en direkt fortsättning av aktuellt arbete inom företaget.

Varje arbetsgruppsmedlem förväntas komma med minst ett förslag till ett sådant mikroprojekt. Förslaget skall vara konkret, och så intressant att medlemmen är villig att själv arbeta i mikroprojektet. Under arbetsgruppens första möte skall avgöras, vilka projekt som skall bedrivas inom arbetsgruppen. Medlemmen kan då antingen övertyga minst en annan person att medverka i sitt eget mikroprojekt, eller hoppa på ett annat. Redovisning av mikroprojektet sker vid det informationsmöte, som i år kommer att äga rum den 21-22 oktober 1999.

Till VU:s förfogande finns också projektmedel. Dessa kan antingen utnyttjas inom mikroprojekten, främst för inköp av material. Vid större projekt kan medel också utgå till utökad arbetstid inom projektet. FMV räknar med att kunna ställa upp till 500 kSEK till förfogande för sådana projektmedel. Fram till den 1 mars finns 100 kSEK tillgängligt. Detta är ytterligare en anledning att se till att arbetet snabbt kommer igång i början av 1999.

Det är viktigt att de olika medlemsföretagen i SESAM snabbt utser representanter i arbetsgrupperna. Som medlem i SESAM.s råd har du ett speciellt ansvar för att ditt företag anmäler sina representanter till de två arbetsgrupperna. Jag ber dig därför att se till att ditt företag före jul utser sina representanter och att dessa kontaktar sin arbetsgruppsordförande.

Det är min övertygelse, att om vi alla gemensamt lyckas få detta nya arbetssätt att snabbt fungera, så kommer värdet av SESAM-arbetet att kraftigt öka.

Med vänlig hälsning

Björn Källberg
Ordförande i VU.

Ett projekt som använder Ada

UTA JAS 39 MARKSYSTEM.

DATORISERAT SYSTEM FÖR ÅTERSPELNING OCH UTVÄRDERING AV FLYGUPPDRAG

Moderna flygsystem av typen JAS 39 är utrustade med ett avancerat datainsamlings- och dataregistreringssystem för lagring av data från genomförda flyguppdrag. Registrerdata omfattar bl a multiplexad video, tillståndsdata, sensordata, audioinformation m m.

Återspelning och analys av registrerdata sker på förbandet, omedelbart efter landning, i en speciell markutrustning, UTA JAS 39 MARKSYSTEM.

UTA marksystem omfattar bl a funktioner för utvärdering och simulering av vapeninsatser med ostyrda och styrda vapen såsom t ex olika typer av jaktrobotar. Vidare finns funktioner för presentation och utvärdering flygplanets kabinsituation och flygförarens systemhandhavande. Med hjälp av datorgenererad realtidsgrafik kan ett 3D perspektivtransformerat scenario skapas där samövande flygplan deras manövrer och vapeninsatser presenteras. Audioinformation kommunikation, varningssignaler m m återges i högtalare och hörtelefoner.

UTA-systemet är uppbyggt som ett distribuerat datorsystem omfattande resurser för uppackning och beräkning av data, presentation av realtidsgrafik, bearbetning och presentation av audioinformation samt för administration och lagring av data i en SQL-databas. Systemets mekaniska utformning och uppbyggnad sådan att det är transportabelt och kan utnyttjas på krigsbas.

Analysresultaten utnyttjas för utvärdering av föraren insats, fpl-systemets förmåga, bedömning av taktikval, utvärdering av hotbilden samt för generering av rapporter och underrättelseinformation. Utvärdering av registrerdatat utgör bland annat underlag för ett antal olika rapporter. Rapporter och utvärderingsresultat kan lagras i databasen samt distribueras som elektronisk post inom det egna förbandet och till lednings- och kommandostaber.

Återspelning och analys av registrerdata utgör ett effektivt hjälpmedel för att bygga upp förarnas erfarenhet och skicklighet. Ett på detta sätt återskapad animerat flyguppdrag bidrar aktivt till att skapa mentala bilder och föreställningar om intressanta och kritiska moment, befästa korrekta beteenden samt att skapa underlag för uppbyggnad av erfarenheter hos den flygande personalen. UTA-systemet utgör också en viktig och oundgänglig länk i den ”taktiska loopen”.

Claes Wadsten
Celsius Aerotech

Mana-projektet – En säker miljö för Ada-95

Kristina Lundqvist, Tobias Johnsson och Lars Asplund

Mana[1]-projektet, som stöds av NUTEK, SKI och FMV, syftar till att utveckla en **säker** run-time kärna för säkerhetskritiska system. Angreppssättet med formell verifiering, som vi har valt är annorlunda än det traditionella sättet. Formell verifiering används industriellt idag, inom framförallt signalsystem inom järnvägen.

Ada är genom sin uppbyggnad ett mycket lämpligt språk för säkerhetskritiska tillämpningar. Det är av dessa skäl som exempelvis Boeing har valt Ada för den säkerhetskritiska delen i Boeing 777. Ett subset av Ada-83 som har utvecklats för säkerhetskritiska tillämpningar är SPARK. Detta subset innehåller ingen tasking. SPARK har även utökats med en notation för analys av programvaran.

Vår nya standard, Ada-95, är en utökning av Ada-83 i flera avseenden. För att bygga säkerhetskritiska system kan det tyckas omöjligt att utgå från Ada-95. Dessbättre är det tvärtom, ett mycket bra tillägg till språket är Protected Objects. Detta framgår klart när man studerar Ravenscar-modellen. Ravenscar är ett subset av Ada-95, där man har eliminerat mycket men ändock behållit tasking.

Ett system byggt på Ravenscar består av tasks som enbart är på biblioteksnivå, vilket betyder att man inte behöver hantera termineringsproblematiken. Kommunikation mellan tasks sker endast genom Protected Objects. Det förekommer med andra ord ingen rendezvous mellan tasks. Detta har naturligtvis stora fördelar när man vill verifiera att ett system är korrekt. Försöken att formellt verifiera Ada-kod med rendezvous med bland annat select-satser har mer eller mindre misslyckats.

Man har även gjort begränsningar i delay-semantiken och tillåter endast delay until. Eftersom säkerhetskritiska system oftast är inbyggda system har man behållit semantiken för avbrottshantering – den som är specificerad för Protected Objects.

Vårt arbete består av att dels beskriva Run-time-systemet på sådant sätt att det går att formellt verifiera, verifiera detta med något lämpligt verktyg och slutligen implementera run-time-systemet så att man kan köra tester och kanske även certifiera det för exempelvis FAA.

Grundfunktioner i ett RTS för Ravenscar

Genom alla begränsningarna som ingår i Ravenscar, kommer antalet ingångar i Run-time-systemet att reduceras till följande fem:

- Delay until - suspendera en process
- Enter PO Procedure - kommunikation
- Enter PO Entry - kommunikation
- Exit PO - avsluta kommunikation
- Avbrott - Externa (asynkrona/synkrona händelser)

Två av dessa, Delay och Avbrott är naturliga att förstå, de kommer av att modellen inkluderar dessa egenskaper. Det är kanske mer förvånande att man klarar övrig kommunikation och schemaläggning med enbart tre primitiver för hantering av Protected Objects. Kommunikationen sker genom att en task kan göra anrop till en PO's enda entry. Det är inte tillåtet att mer än enda task gör anrop till en specifik PO. En eller fler tasks kan göra anrop till de procedurer som finns i en PO. Semantiken för PO garanterar att endast en task i taget kan 'vara inne' i en PO. Barriären för en PO, och som avgör om en entry är öppen eller ej, måste vara en enkel Boolean.

Schemaläggning

Systemet arbetar med ICPI (Immediate Ceiling

---

[1] ma.na \'ma:n-\ n [of Melanesian - Polynesian origin; akin to Hawaiian - Maori mana] 1: the power of the elemental forces of nature embodied in an object or person 2: moral authority : PRESTIGE*

Priority Inheritance), vilket är ett väl definierat protokoll och det ger minsta möjlig fördröjning av högprioritets tasks (dvs blocking är minimal). Eventuell blocking sker alltid i början av en period. Något som är viktigt om man strävar efter högsta möjliga predikterbarhet

Formell verifiering

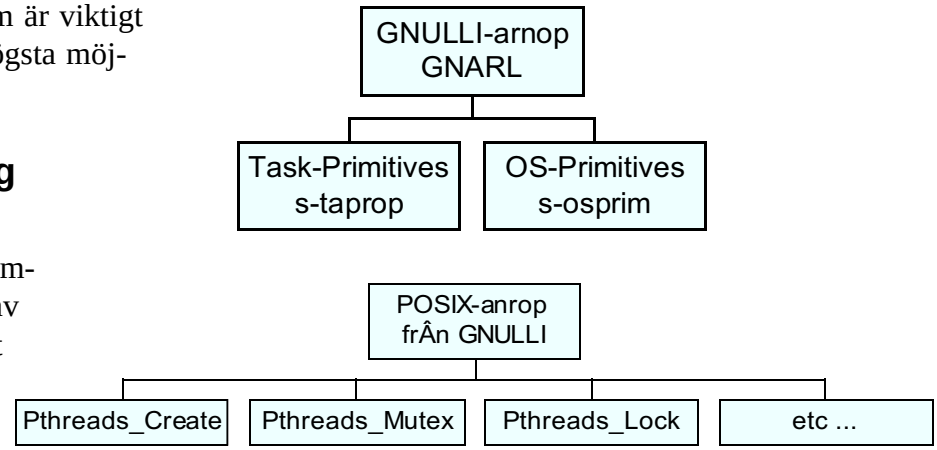
Själva verifieringen kommer att göras med hjälp av UPPAAL, vilket är ett verktyg för design och analys av real-tids-system. Verktyget är baserat på en constraint-lösare och on-the-fly teknik. Den är lämpad för modellering av en samling icke-deterministiska processer med finit kontrollstruktur och realtidsklockor där kommunikation sker genom kanaler eller delade variabler. Verktyget har hittills visat sig vara användbart vid analys av säkerhetsegenskaperna i komplicerade tillämpningar. Huvudtillämpningarna är de vars tidsegenskaper är kritiska, såsom realtidskontrollers och realtidsprotokoll. För vidare studie av UPPAAL kan vi hänvisa till K.G. Larsen, P. Pettersson, W. Yi, ”UPPAAL in a Nutshell”, International Journal on Software Tools for Technology Transfer, 1997.

UPPAAL är utvecklat i samarbete mellan Datorteknik, Uppsala universitet och BRICS (BRICS (Basic Research in Computer Science, Centre of the Danish National Research Foundation) vid Ålborgs universitet. Ytterligare information kan man få på http://www.docs.uu.se/uppaal.

Implementering

Vi avser att implementera vårt RTS som en del av gnat-kompilatorn. När man använder gnat’en utnyttjar man gcc för att genera koden. Man kan själv bygga gcc från grunden, vilket normalt sker på ett UNIX-system. Ett körande Ada program gör RTS-anrop enl. följande modell:

GNARL och GNULLI är den del av runtimesystemet som berör tasking. Internt ser GNARL/GNULLI i grova drag ut så här:



Det som kommer att vara vårt RTS kommer inte alls att utnyttja POSIX-trådpaket och vi kommer att göra en implementering där GNARL och GNULLI smälter samman. Eftersom endast en liten del av POSIX trådpaket kommer att behövas. POSIX gränssnitt är mycket snarlikt GNULLI, varför de av effektivitets- och utrymmesskäl, mycket väl kan smälta samman i denna implementering. Delmängden av POSIX blir ju ändå specialsydd för detta projekt, så återanvändbarhets-argumentet försvinner. Läsbarheten ökar också när ett nästan tomt abstraktionslager försvinner .

Även mycket av GNARLI (och därmed GNARL) kommer inte heller att behövas för Ravenscar. Troligtvis behövs bara funktionalitet för Protected objects, Delays och Interrupthantering vid runtime. Därutöver behövs diverse initieringskod.

Beroende på hur saker utvecklar sig kanske det kan vara lämpligt att lägga ihop både GNARL, GNULL och vårt POSIX-substitut till ett enda lager.

Även om Ravenscar koncentrerar sig på taskingdelen av RTS’et, kan det vara av intresse att studera övriga RTS’et, som t.ex. någon form av I/O.

Editors: Jerzy W. Rozenblit, University of Arizona, ECE 320E, Tucson, AZ 85721; jr@ece.arizona.edu; and Sanjaya Kumar, Honeywell Technology Center, MS MN65-2200, 3660 Technology Dr., Minneapolis, MN 55418; skumar@htc.honeywell.com

From Busyware to Stableware

Harold W. Lawson, Lawson Konsult AB

The integrated engineering of complex systems depends, among other things, on the supply of reliable, cost-effective, stable components. However, as "Salvation from System Complexity," (*Computer*, Feb. 1998, pp. 118-120) points out, unnecessary complexity abounds in today's computer and communication structures, including the platforms that host many complex, often critical, systems. Thus, even if an organization accurately designs, develops, and supports reliable, cost-effective, nonplatform components, deploying them in an unstable platform will counteract even the most stringent engineering and management efforts. The system will be no stronger than its weakest components.

Robert Colwell summed it up nicely in his contribution to "Challenges and Trends in Processor Design" (*Computer*, Jan. 1998, pp. 45-47):

> Antilock brakes that "mostly work" or "hardly every crash" wouldn't be acceptable, but that describes general-purpose computing today. This situation arises because we do not design hardware in conjunction with software, application developers don't design software with the OS, and companies place less emphasis on the overall hardware-software system reliability than in getting to market quickly. Ultimately the lack of system dependability could well become industry's concern because it will become society's burden.

I agree with Colwell in all but his use of the future tense in the last sentence. Societal burdens are already here. A recent example is the Windows NT error that caused the Yorktown, an Aegis guided missile cruiser, to be disabled. The Yorktown could not perform its mission and had to be towed into port. An otherwise well engineered system failed because of a platform bug. There are likely many platform-related failures in a variety of critical application domains that are never made public.

A concerted (hopefully industry-driven) effort must be made to move toward the stable integrated platform solutions that Colwell indicates. This requires a transformation from what I call busyware to stableware. Figure 1 illustrates:

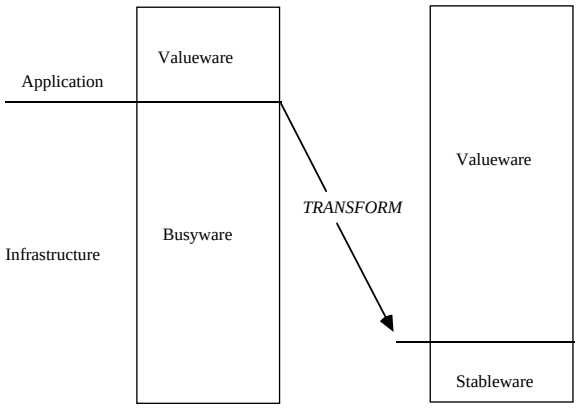


Figure 1. The essential transformation from busyware to stableware. The current scenario (left) portrays the proportion of time and energy required to develop and support applications (valueware) in relationship to the time and energy spent dealing with the idiosyncrasies of

platform hardware and software (busyware). The stableware scenario (right) radically reduces the time and energy required to dealing with infrastructure complexities and directs major efforts toward developing and supporting high-quality applications.

BUSYWARE

As indicated in the figure, inordinate numbers of staff-hours must be spent understanding (or attempting to understand) the complex idiosyncrasies of mainstream hardware and system software. Inexplicable system crashes, poor error treatment, nonterminating programs, incompatible versions, incompatible file formats, security violations, viruses, and so on are some of the sources of frustration leading to increased attention to busyware at the expense of valueware efforts. On the up side, there is an enormous market for services and training; on the down side, the costs and added risks for complex system projects are significant- and can even be devastating.

As employment advertisements make clear, a characteristic of busyware is that the mastery of even portions of a complex platform provides unprecedented levels of economic remuneration, even for such mundane tasks as the ability to alter system parameters.

Busyware complexities stem from an increasing push for more functionality as well as the "hacking" of modifications and extensions to the mainstream hardware and system software platform base. As a result, the design of mainstream hardware almost exclusively emphasizes performance and memory capacity. When faster processors and more memory are available, software vendors provide more functionality, which then requires even more processor speed and memory capacity. The industry spirals more deeply into a black hole of complexity.

Moreover, because mainstream processor architectures are not hospitable hosts for true higher level languages (such as Smalltalk, Modula, Eiffel, or Ada), platform developers continue to deploy lower level languages, including C and C++, for most all systems programming. Once again, this leads to unnecessary complexity—with its attendant costs and risks—especially in maintenance.

Suffice it to say that if we continue along busyware lines for mainstream products, we will never see stable, reliable, safe, and secure platform components. A transformation must take place.

THE TRANSFORMATION

With stableware platforms, system life cycle efforts shift from dealing with mundane busyware complexities to the creation and maintenance of high-quality applications (valueware), as Figure 1 shows.

However, the following key things must occur before our industry can achieve this transformation:

- *We must break the current spiral of performance-increased functionality*. Key design decisions for combined hardware and systems software must emphasize understandability, maintainability, security, and safety above performance.

- *We must elevate the level of program abstractions used to create system software*. In this way, we will improve the understandability and maintainability of all software.

- *We must rigorously define key protocols, operating system functions, and programming language semantics in the form of machines (with concrete instruction definitions)*. These instruction sets become the target languages for platform implementation.

These stableware prerequisites have historical anchoring. Thus, to see how we can move forward toward stableware, we must look back to before the microprocessor revolution. Several, but certainly not all, of the technical ideas we need to produce stableware have already been realized. For example, my first exposure to using higher level languages for system programming dates back to 1959, to the construction of the Univac II Cobol compiler. My boss at that time,

the legendary Grace Hopper, demanded that the compiler be written in Univac II Flowmatic. In the second half of the 1960s, the father of secure operating systems thinking—Multics—was developed at MIT using PL/I.

Another example of looking to the past for solutions are some general-purpose microprogrammable machines, developed in the late 1960s and early 1970s, which can provide a basis for what I have termed FIMs (function integration microprocessors). FIMs would let us emulate existing and new target languages—an essential ingredient of the transformation.

These are only a few historical examples; I could cite many more.

MOTIVATION

Getting the computer industry to make the transformation or even address the underlying issues is, mildly stated, nontrivial. Patents, trade secrets, and other enormous economic interests are involved. But I have seen some rays of hope. One is the willingness of Sun Microsystems to cede control over Java to international standards bodies. Another is the success of the Internet Engineering Task Force in producing consensus standards for TCP, IP, and HTTP as described in "The IETF: Laying the Net's Asphalt" (*Computer*, Aug. 1998, pp. 116-117). Quite simply, we need to lay asphalt for stableware platforms—ideally through industry consensus, as the IETF did.

Regardless of how we do it, it is vital to coordinate all related government efforts (acquisition, funding, legislation, justice, and so on) to find a constructive way to empower our industry to generate stableware.

Stimulating competition

The establishment of standards (even de facto) that accurately define the syntax and semantics (as executable machines) of higher level abstractions for programming languages as well as for key operating system and protocol functions, would provide a real competitive stimulus for manufacturers and developers of processors and system software.

While this would seem to work against today's short-term profit goals, the long term benefits for industry and society would be enormous. There would be a major shift to intense competition in producing high-quality applications.

Product liability limits

The moment stableware standards can be verified and platform products certified, product liability will have entered a new era. In fact, that moment should kick off the biggest insurance boom in history; suppliers can be insured to limit their liability in accordance with claims levied against the infrastructure components they supply.

Mainstream computing has moved society in new exciting directions. At the same time, it has proved unnecessarily costly and risk filled. For critical systems, it has also proved dangerous.

Establishing stableware is undoubtedly one of the most important challenges facing the computing industry and society in general. Systems engineers must be able to demand the same quality of platform components as they do on other system components. Although a concerted, integrated push toward stableware is technically feasible, overcoming political and commercial aspects will be a formidable challenge.

There are several reasons for meeting that challenge. Stableware will provide the ultimate in open systems; existing and new industry players will be able to implement products. Imagine current complex systems based on stableware instead of busyware. The cost reduction related to the Y2K problem alone would finance stableware development and support many times over. Further, our new lifelines of society would be based on solutions that are as reliable, safe, and secure as possible.

Harold W. (Bud) Lawson is an independent consultant in Stockholm. An IEEE and ACM fellow, he has contributed to several pioneering hardware, software, and application related endeavors. Contact him at bud@lawson.se.

• • • • • • • •

Synpunkter på modellering

Ingmar Ögren, iog@toolforsystems.com

Under åren med utveckling av metoden O4S (Objects For Systems) och verktygen ASP (Ada Specification Producer) och Tofs (Tool For Systems) har jag förstått tre saker:

- **de flesta som vill rationalisera sitt systemarbete, börjar med överväga anskaffning av ett verktyg**
- **för att bygga ett verktyg måste man utgå från en metod och det underlättar att känna till metoden för att förstå verktyget**
- **metoder innehåller alltid minst en modelleringsprincip. För att kunna använda metoder och verktyg hjälper det att ha förstått den underliggande modelleringsprincipen.**

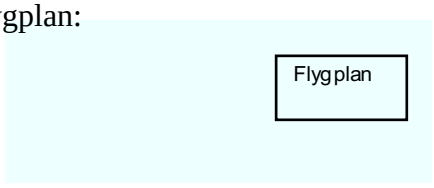
Här har jag sammanställt lite synpunkter på modellering för systemarbete, som är resultat från arbetet med O4S och Tofs. Sammanställningen gjordes först på begäran för INCOSEs tidning INSIGHT.

Hur vet man vad man modellerar?

När man granskar ett diagram för programvaru- eller systemarbete hittar man ofta enkla entiteter (företeelser) som t. ex. “lufttemperatur” eller “flygplanets position”. Då kan man fråga diagrammets författare vad som menas: “Är det verkligen flygplanets position eller datorns uppfattning om flygplanets position?” Frågan kan orsaka förvirring och ofta blir svaret något i stil med: “Det är den här posten i datalexikonet, som representeras av det här flyttalet” Om man sedan ställer nästa fråga “Hur vet man att det verkligen är flygplanets position”, så kan man få ett klart och begripligt svar. Det händer också att man får en förvirrande diskussion med data, kommunikationsvägar och fördröjningar genom systemet, som ger dålig förståelse av hur väl dataposten representerar sin motsvarighet i verkligheten..

I sådana fall kan det hjälpa att rita ett omvärldsdiagram. Det är ett enkelt sätt att förbättra kunskapen om hur entiteter i ett system representerar och kopplas till entiteter i systemets omvärld.

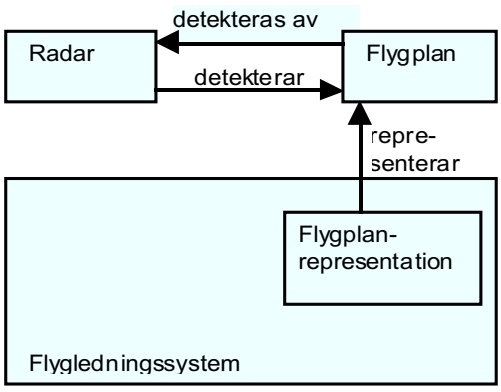
För att rita ett omvärldsdiagram börjar man med en entitet i systemets omvärld, som t. ex. ett flygplan:



Sedan kan man föra in en radar för att detektera flygplanet och vidare ett par relationer mellan flygplanet och radarn:

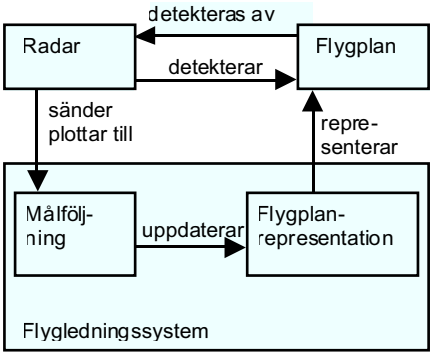
Relationerna ritas här “bägge vägarna” för att visa att det här inte är något dataflödesdiagram utan ett ER-diagram, som bara visar entiteter och relationer. För att läsa och förstå diagrammet läser man helt enkelt: <boxtext> <piltext> <boxtext>. När man skall granska ett omvärldsdiagram är det en bra idé att läsa de här enkla texterna och kontrollera att de både är läsbara och säger något meningsfullt om systemet..

Om man vill bygga ett flygledningssystem, så hamnar flygplanet och radarn utanför systemet, men man måste representera flygplanet i systemet:



Vad diagrammet säger så här långt är helt enkelt att man använder radar för att detektera flygplan och att man måste representera flygplan i flygledningssystem. Detta kan verka, och är också, alldeles trivialt men att etablera grundläggande fakta kan visa sig vara av avgörande betydelse i andra och mera komplexa sammanhang.

Diagrammet säger dock ännu ingenting om hur man skall åstadkomma "Flygplan-representation". För att komplettera diagrammet kan man införa en ytterligare entitet "Målföljning", med sina relationer:



Nu har man ett enkelt omvärldsdiagram med sina relationer. Diagrammet visar en företeelse i omvärlden eller "verkligheten" (Flygplan) och dess representation i ett system (Flygplan-representation). Diagrammet innehåller "dubbel koppling", som innebär att diagrammet visar:

- Hur en företeelse i omvärlden representeras i systemet
- Hur företeelsen i omvärlden påverkar sin representation i systemet.

Diagrammet uttrycker också några enkla fakta om systemet i sin omgivning om man läser <box text> <piltext> <box text>. Om de här meningarna inte verkar vettiga eller är grammatiskt felaktiga, behöver man troligen arbeta lite till med diagrammet.

Vad som är gjort med diagrammet är modellering på två nivåer:

- diagrammet är en modell av ett system i sin omgivning
- diagrammet belyser en aspekt av hur systemets omgivning modelleras i systemet.

Observera att "omgivningen" inte nödvändigtvis är den "fysiska omgivningen". Exempelvis kan ett inbyggt datorsystem mycket väl ha andra datorsystem som sin omgivning.

Man kan rita omvärldsdiagram med papper och penna och det är ofta ett mycket bra alternativ, speciellt tidigt i systemanalys när man vill bygga upp en förståelse av ett befintligt eller framtida system. Att rita omvärldsdiagram tillsammans med en erfaren slutanvändare på en svart tavla eller ett blädderblock är ett mycket bra sätt att skapa förståelse för och dokumentera grundläggande fakta.

Försök komma ihåg:

- Vad man ritar är entiteter (företeelser) och deras relationer, inte ett dataflödesdiagram
- Gör det inte för invecklat. Flera små enkla diagram är bättre än ett stort komplext diagram eftersom det kan vara svårt att hitta felen i en komplex grafisk struktur.

Verktyg är ett problem när det gäller omvärldsdiagram. En svart tavla är ett mycket bra verktyg, som dock har sina begränsningar vad gäller lagring. Datorlagring är bättre och man kan använda enkla ritverktyg som PowerPoint eller Visio för att rita och lagra omvärldsdiagram. Man kan också använda andra verktyg med ritfunktion, som t. ex. CAD- eller CASE-verktyg.

Innan man väljer ett verktyg är det dock bra att kontrollera att verktyget inte har några besvärliga syntaktiska begränsningar och att det kan göra "nyttiga konster", som "gummibandning" och "snappning".

Hur skall man modellera?

Det finns många sätt att modellera ett system och man kan undra vilket man skall välja. Svaret är enkelt: "det beror på". Det beror på vilken aspekt på systemet man skall modellera och på vem som skall läsa modellen. Ett annat svar är att man måste bemästra ett urval av modelleringstekniker för att täcka behoven under ett systemarbete. Man måste också betänka vad som krävs för att modellera ett system. Bland andra finns det tre krav:

1. Determinism med formalitet

Detta betyder att allt som uttrycks i modellen skall ha en entydig, definierad och uppenbar innebörd.

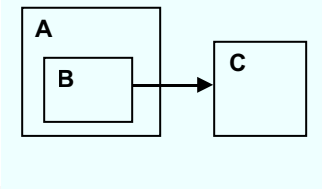
2. Förståelighet

Eftersom systemarbete bör göras i nära samverkan med slutanvändare, måste modellerna vara lätta att begripa även för den som inte har omfattande utbildning i och erfarenhet av programvaruarbete eller matematik.

3. Innefattande av uppdrag

Detta innebär att en modell bör belysa systemets uppdrag, och kunna uttrycka hur olika delar av systemet bidrar till att genomföra uppdragen.

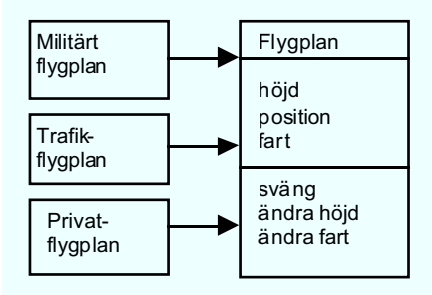
Nedan diskuteras några användbara tekniker för modellering:



Blockschemat är kanske det äldsta sättet att modellera system. Det är mycket enkelt att förstå. Exemplet ovan visar att A innehåller B och att B överför någonting till C. Blockschemor används ofta för att visa maskinvarustrukturer, för organisationsschemor och för att strukturera programvara (som dataflödes- och kontextdigram). Blockschemor är bra på att visa ett systems struktur på ett begripligt sätt. De säger dock väldigt lite om ett systems uppdrag.

Man kan undra om det inte finns någon användbar standard för modelleringsdiagram. De "three amigos" (Grady Booch, Ivar Jacobson och James Rumbaugh) vid Rational Software har skapat "Unified Modeling Language" (UML), som håller på att standardiseras.

UML innehåller åtskilliga modelleringsdiagram, bland andra klassdiagrammet och komponentdiagrammet.



Diagrammet ovan visar ett flygplanexempel, ritat i en liten delmängd av UMLs syntax för klassdiagram. Det visar att klassen "Flygplan" har attribut (höjd, position och fart) och aktioner (sväng, ändra höjd och ändra fart). Det visar vidare att tre klasser (Militärt flygplan, Trafikflygplan och Privatflygplan) alla härleds från (ärver) klassen "Flygplan".

Detta är bara en liten delmängd av vad man kan modellera med klassdiagrammen i UML. Man kan även modellera beroende, associering, aggregering och kardinalitet. För detaljer om detta, studera Rationals websida eller litteraturen om UML.

Den rika syntaxen och uttrycksfullheten är uppenbara fördelar för UMLs klassdiagram.

Den rika syntaxen kan också vara en nackdel eftersom det är enkelt att rita komplexa och förvirrande diagram om man använder hela syntaxen. Speciellt om man arbetar tillsammans med slutanvändare kan det vara en bra idé att nöja sig med en delmängd av klassdiagrammets syntax.

Ett annat problem är att, även om klassdiagrammet kan användas för att visa ett systems struktur mycket detaljerat, så säger det väldigt lite om systemets uppdrag. Här erbjuder UML de separata "användningsfallsdiagrammen". De hjälper till att skapa en förståelse för uppdragen, men kan orsaka problem när man behöver göra en pålitlighetsanalys av ett system med operatörsroller integrerade i systemet.

Ett annat UML-diagram är komponentdiagrammet. Det publicerades i Grady Boochs bok om Software Engineering från 1983 och det har vidareutvecklats i programvarumetoden HOOD.

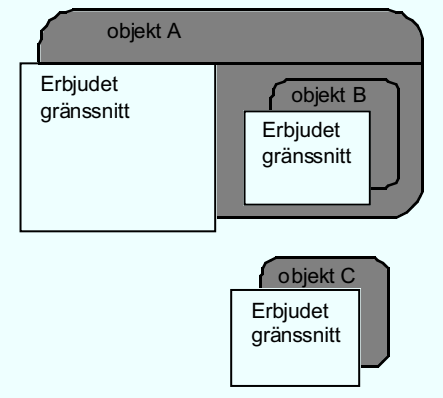
Det här diagrammet är mycket användbart och det kan användas för att modellera kompositiva objektstrukturer. När man arbetar med kompositiva objektstrukturer, koncentrerar man sig på objektens gränssnitt och på förbindningarna mellan objekten, snarare än på hur objekten ärver varandra.

Diagrammet nedan visar att:

- Objektet A har ett erbjudet gränssnitt (sammansatt av ett antal aktioner, som kan initieras från objektets omvärld)
- Objektet A har ett krävt gränssnitt, sam-

mansatt av delar av de erbjudna gränssnitten för stödobjekten B och C

- Objektet B ingår i samma system som objektet A medan objektet C finns utanför detta system.



Komponentdiagrammet tillåter att man modellerar, inte bara programvaru- och maskinvarukomponenter, utan också operatörsroller och uppdrag. Det här gör det möjligt att modellera komplexa system, som en uppsättning diagram på olika nivåer, med klara kopplingar mellan objekten och en klar förståelse av hur de olika objekten bidrar till att genomföra systemets uppdrag.

En nackdel med komponentdiagrammet är att det kräver en del förklaring, när man samarbetar med slutanvändare som inte är bekanta med principerna för "kompositiv objektinriktning".

Projektgemensam modell

När man arbetar med komplexa system, som ofta i försvarssammanhang, är det normalt att många problem dyker upp under systemanalys och konstruktion. Många av dessa problem kräver simulering, vilket resulterar i en situation där man har:

- Ett "verkligt system" i en eller flera versioner, som är mer eller mindre färdiga
- Ett antal simulatorer, som var och en modellerar en aspekt av det "verkliga systemet".

I den här situationen kan en hemsk misstanke dyka upp i utvecklingsorganisationen: "stämmer alla simulatorerna med varandra och med det verkliga systemet?"

Ett annat problem när man arbetar med komplexa system är att slutanvändaren ofta vill lägga in produkten som en komponent i ett system på högre nivå:

- Om man producerar ett jaktflygplan kommer kunderna att föra in det som en komponent i sina luftförsvarssystem tillsammans med stridsledningS- och luftbevakningskomponenter.
- Om man producerar kraftverk, kommer kunderna att lägga in produkten som en komponent i sina kraftdistributionssystem.

Eftersom slutanvändarna behöver analysera och modellera sina system, kommer de att behöva modeller av alla delsystem som ingår som komponenter i deras system. Då är det en uppenbar fördel om en leverantör och en slutanvändare kan använda samma modell av ett system. Dessutom vore det ju bra om modellen kunde baseras på en gemensam förståelse. Dessa tankar har lett

till arbetsprincipen med "projektgemensam modell" (PGM).

En PGM är gemensam på två sätt:

- Den är gemensam för olika implementeringar av ett system, såväl olika simuleringar som olika versioner av det "riktiga systemet".
- Den är gemensam för, och ägs gemensamt av en eller flera slutanvändare och leverantörer som är berörda av det modellerade systemet.

Hur skall man då bygga en PGM? Svaret är återigen att "det beror på". Det finns åtskilliga användbara modelleringsverktyg och de flesta av dem kan användas för att bygga en PGM. Några krav på verktyg för att bygga en PGM är:

- Verktyget måste producera datorlagrade modeller. Även om det kan vara möjligt att bygga och underhålla en pappersbaserad modell, så är det nog tveksamt om tekniken med pärmar och bläckändringar duger för att dokumentera och underhålla en komplex projektgemensam modell, när det modellerade systemet förändras med tiden och flera versioner kan behöva hanteras parallellt.
- Verktyget måste stödja en kompositiv (hierarkisk) princip som gör det möjligt att klarlägga vilka objekt som är ansvariga för vilka delar av ett system.
- Verktyget måste kunna hantera, inte bara modeller av objekten som ingår i systemet, utan också viktiga attribut som krav, problem och testfall.
- Verktyget måste på ett effektivt sätt anknyta till olika sorters modeller och dokument, som används av leverantörer och slutanvändare, som berörs av det aktuella systemet (Det är viktigt att en PGM anknyter till och inte försöker ersätta befintlig dokumentation).

Hur bygger man då och använder en PGM?

Figuren ovan visar hur en PGM bör planeras från början av ett systemutvecklingsprojekt även om det naturligtvis är möjligt att etablera modellen på ett senare stadium.

Idealt börjar man modellera redan på konceptstadiet för ett nytt system, för att sedan låta modellen växa tillsammans med systemet. Den färdiga modellen skall omfatta, inte bara det modellerade systemet, utan också systemets omgivning för att man skall kunna få en god uppfattning om hur systemet skall köras regulljärt och för att modellen skall ge tillräckligt underlag för simulering.

Figuren ovan visar också hur en PGM kan användas från tillämpningssynpunkt och från utvecklingssynpunkt. Kanske viktigast är att en PGM tvingar leverantörer och slutanvändare att överväga alla gränsyteproblem innan de byggs fast i systemet. En annan viktig aspekt är att, förutsatt att modellen underhålls, så kommer den att fungera som ett "gemensamt minne" för organisationerna som äger modellen.

Ovan har några aspekter av modellering diskuterats. Vad som inte har sagts är att modellering är roligt**, även om det kan vara nog så ansträngande. Det känns bra när man har tillbringat en heldag med en slutanvändare vid svarta tavlan och sedan har fått in modellen i datorn med känslan att alla berörda förstår och håller med!**

Författaren: **Ingmar Ögren har tidigare arbetat vid FMV och har nu sedan många år arbetat med metoden O4S (Objects For Systems) och verktyget Tofs (TOol For Systems).**

För en biografi och mera information om O4S och Tofs, se http://www.toolforsystems.com

Därifrån kan man också ladda ner en fullt fungerande gratisversion av Tofs och de kompletta utvecklingshandböckerna för metoden O4S.

• • • • • • •

AiS ger första SESAM-sponsrade Ada-priset till Uppsala Universitet

Priset i denna tävling som AiS har utlyst skall gå till den studerande eller forskare som i sitt arbete visat den största kreativiteten kombinerad med praktisk nytta för svensk industri och där Ada ingår som en väsentlig komponent.

Prisnämnden har funnit att två tävlingsbidrag bör belönas och därför beslutat att den tillgängliga prissumman delas i ett första och ett andra pris. De två arbeten som belönas behandlar båda Ada i distribuerade system, ett område som har hög aktualitet. Flera delarbeten i de två belönade avhandlingarna har publicerats gemensamt av de två pristagarna.

Andra pris, 10 000 kronor, har tilldelats Kristina Lundqvist för hennes avhandling ”Distribution of Ada by Means of Software and Hardware”.

Första pris, likaså 10 000 kronor samt en resa till endera av konferenserna SIGAda,98 eller Ada-Europe,99, har tilldelats Göran Wall för hans avhandling ”Performance Analysis and Communication Models for Ada Programs”. Prestandaanalys är ett område som har ett stort allmänt intresse.

498ans liv blev kort

MIL-STD-498, “Software Development & Documentation”, den åldrande 2167s efterträdare fick ett kort liv. 498 blev ett offer för DODs korståg mot militära specifikationer o dyl. och upphävdes formellt den 27 maj i år. Den har fått en ”kommersiell” efterträdare som tagit till sig en hel del av sakinnehållet i 498, nämligen IEEE/EIA 12207, ”Information Technology-Software Life Cycle Processes”, till vilken DOD nu hänvisar när det gäller information om programvaruutveckling och dokumentation. IEEE/EIA 12207 är vad som en tid betecknades US 12207, d v s en amerikansk implementation av ISO/IEC 12207.

SESAM på Ada-Europe

SESAM deltog i utställningen i samband med Ada-Europe konferens i Uppsala den 8-11 juni. Materialet bestod av tre dubbel höjd fyrbredds planscharrangemang som visade dels vad SESAM är och vill, dels olika system som SESAMs intressenter utvecklar. Dessutom fanns två datorer för att demonstrera AWESOME-skivans innehåll.

Temat för 1998 - ”Software Engineering med Ada och Java”

Försöket att ange ett gemensamt tema som verksamheten i arbetsgrupperna, vid sidan av sin ”ordinarie” verksamhet, skulle fokusera på, torde få sägas ha hittills gett blandat resultat. En orsak kan vara att många av arbetsgruppernas medlemmar ännu inte haft någon erfarenhet av Javas användning. Att Java ändå intresserar många visas av att ett 30-tal personer deltog vid det extra SESAM-seminarium som med kort förvarning anordnades på Ericsson Microwave den 18 september. De två av Ag-ordförandena Hans-Olof Danielsson och Håkan Edler presenterade på det ordinarie höstseminariet en, bl a på inläggen vid extraseminariet baserad, preliminär rapport och rekommendation till intressenterna betr hur man skulle kunna ställa sig till Javas användning och vilka åtgärder som skulle kunna vara möjliga. Denna preliminära rapport är tänkt att gå på remiss till arbetsgrupperna inför en slutskrivning och leverans till intressenterna någon gång vid årsskftet.

AWESOME-skivan

Genom en mycket hård spurt av de inblandade, främst Jesper Andersson och hans kompisar på högskolan i Växjö, men också många andra, hölls med ett par dagars marginal (den reviderade) målsättningen att få ut denna CD-ROM i tid till Ada-Europe-konferensen den 8 juni. Skivan blev till slut mycket innehållsrik och en bra exponent också för programvarukunnandet i försvarssektorn. Huvudändamålet var dock att få fram ett utbildningspaket och en kvalificerad gratis utvecklingsmiljö på PC, inkl två Ada95-kompilatorer och en Ada/Java bytecode kompilator. Alla SESAM-medlemmar skall ha fått sitt eget ex av skivan. Ca 2000 ex har spritts till SESAM-intressenterna och en del andra intresserade, 500 ex sprider AiS till vissa utvalda gymnasieskolor.

Restlagret uppgår f n till ca 500 ex som sekretariatet gärna vill bli av med till något nyttigt ändamål. Spekulanter bör kontakta Anna Kåsjö.

Bidrag till Rendezvous och SESAM hemsida efterlyses

Varken Rendezvous eller hemsidan blir bättre än vad medlemmarna gör den till. Håll ögon och öron öppna för intressanta händelser i eller nära SESAM-världen och skicka in en notis till sekretariatet, alkas@tranet.fmv.se. När det är aktuellt att kunna publicera rapporter eller beskrivningar från projekt som Ni är inblandade i, glöm då inte våra egna medier.

http://sesam.tranet.fmv.se

Kalender

1999-03-07--12	International Conference and Workshop on the Engineering of Computer Based Systems (ECBS), Nashville, TN
1999-04-27	SESAM Rådsmöte,FMV, TrV rum K401, Stockholm
1999-05-02--06	Software Technology Conference , ”Software and Systems for the Next Millennium”, Salt Lake City, Utah
1999-05-16--22	ICSE’99 - 1999 International Conference on Software Engineering, Los Angeles, CA
1999-06-07--11	Ada-Europe ’99 Conference, Santander, Spanien