NYHETSBLAD FÖR SESAM
Försvarssektorns Adaintressenters Användargrupp för Software Engineering

RENDEZVOUS

Nr 3 okt 2000

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 Claes Wadsten, AerotechTelub, tel 013-231652 .

VU

Andersson Tommy, Ericsson Microwave Systems
tommy.andersson@emw.ericsson.se
Bengtsson Christopher, FMV
chben@tranet.fmv.se
Brandt Roger, FMV
robra@fmv.se
Carlsson Ingemar, adjungerad
ingemar.carlsson@mbox2.swipnet.se
Folkesson Dag, Saab AB
dag.folkesson@saab.se
Johansson Billy, CelsiusTech Electronics AB
bijo@celsiustech.se
Merkell Curt, Bofors
curt.merkell@celsius.se
Wadsten Claes, AerotechTelub
claes.wadsten@aerotechtelub.se

Arbetet utförs i två arbetsgrupper:

Ag Metodik
Håkan Edler, CTH/Datorteknik
edler@ce.chalmers.se
Ag Teknik
Vakant

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
AerotechTelub AB
c/o Kåsjös Kontor
Ytterspåret 14
187 54 TÄBY

Ordföranden har ordet

Vi närmar oss nu redovisning av årets verksamhet inom SESAM. Som jag tidigare skrivit så kommer andelen programvara i olika projekt att öka. Vi har sett detta i de mikroprojekt som föreslogs för detta verksamhetsår. Saknar vi en pådrivande kraft inom projekten så har företagen svårt att få deltagarna att arbeta hårt inom projekten då denna typ av personal är bristvara.

Trots att vissa delar av mikroprojekten inte nått de uppsatta mål vi hade vid årets början har många engagerat sig i SESAM:s verksamhet under året. Det har också pågått stor omstrukturering inom den Svenska försvarsindustrin samtidigt som hela försvaret startat sin stora ominriktning. Allt detta sammantaget har bidragit att kompetens inom programvara och datorsystem idag är en bristvara inom försvarsindustrin. Det kommer troligen också att bestå under en lång tid framöver.

Av mina sonderingar ute bland försvarsindustrin så ser vi alla ett stort behov av samordning och bildande av kontaktnät. SESAM som stimulator och kontaktnät har därför en viktig plats i framtiden. SESAM:s kontakter med högskolorna kan hjälpa till med att vara en länk mellan industrin och utbildningsväsendet så att framtida ingenjörer har rätt utbildning inom dessa frågor. Det blir också en länk mellan mindre och större industrier även då man inte har pågående gemensamma projekt.

Vid årets höstseminarie är huvudtemat ”Programvara i framtidens försvar”. Vi kommer då att få se den ökade betydelsen försvarsmakten lägger på programvara i olika system.

Vi som varit med några år inom ”datorbranschen” har upplevt hur man från ha lagt ner stor vikt på strukturering övergått till utveckling via ”hackers-modell” för att åter försöka skapa ordning och reda vid systemutvecklingen. Skall vi bygga system av system så framträder många frågor i ljuset som SESAM genom åren värnat och belyst. Strukturerade språk, operativsystem, utvecklingsverktyg, realtidssystem, verifieringar, testbarhet, handböcker etc.

I samband höstseminariet kommer vi som medlemmar i SESAM också att få ett antal redovisningar som belyser det jag ovan nämnde.

Jag önskar därför att projekt som vi inte orkat med hittills i år snarast startar så att kontaktnätet hålls levande.

Med hälsningar
Claes Wadsten

Rapport från JavaOne 2000

av Mikael Jeppsson och Annika Ohlsson,
Software Laboratory, Ericsson Microwave Systems AB

Den 6-9 Juni hölls konferensen JavaOne i San Francisco för femte året i rad. JavaOne är en av världens största programvaruorienterade konferenser, med i år ca 25000 deltagare från hela världen. Konferensen riktar sig främst till programvaruutvecklare med inriktning mot Javaprogrammering.
Redan på Söndagen den 4:e juni inleddes konferensen med s.k. universitetsdagar där det var möjligt att skaffa sig fördjupade kunskaper i något av de ämnen som erbjöds. Bland årets ämnen fanns bl.a. Java Technology Security och Jini Technology: Distributed Service Programming.
På tisdagen inleddes sedan själva JavaOne. Konferensdagarna började med en s.k. keynote, där ungefär 10000 personer samlades i en gigantisk föreläsningssal för att lyssna på stora profiler inom Sun Microsystems och Javavärlden som presenterade nya tekniska apparater och hjälpmedel som använder Javateknologi. Efter detta fylldes dagarna upp med presentationer av olika slag, (Technical Sessions och Business Sessions) som på kvällarna följdes av BOFs (Birds-of-a-Feather) vilket är sessioner för de med lite djupare intresse inom specifika områden.
Parallellt med dessa sessioner pågick en mässa där företag visade upp nya produkter med mer eller mindre anknytning till programspråket Java.

Huvudtema för årets konferens var att tjäna pengar på Java, åtminstone kändes det så. Många av föreläsningarna och tillika företag på mässan var inriktade mot B2B, business-to-business, och e-Commerce. Det handlar helt enkelt om att bygga upp stora webportaler med hjälp av Javateknologi och på så vis tjäna pengar, om inte direkt genom att sälja saker så indirekt genom att bygga återanvändbara Javabönor.
Andra områden som behandlades var bl.a. Java 2 Micro Edition, d.v.s. möjligheten att föra in Java även i mindre terminaler såsom mobiltelefoner och PDA:er, XML och JavaCards.

Bland nyheterna för året kan nämnas Jiro, ett dynamiskt administrationssystem för heterogena apparater och tjänster i ett nätverk.

De viktigaste anledningarna för att närvara på konferensen var för Mikaels del att studera hur arbetet med säkerheten i Java fortlöper och för Annikas del att studera dynamik och realtidsegenskaper i Java.

På säkerhetsfronten kan nämnas att Java Authentication and Authorization Service, JAAS, har släppts i version 1.0, detta extra paket utökar den befintliga säkerhetsmodellen med kontroll av vem som exekverar koden. Den ursprungliga modellen baseras på varifrån koden är laddad och vem som har signerat den. Java Cryptography Extension, JCE, har släppts i en ny version, skillnaden mot den gamla är att de leverantörer av krypto som finns nu måste vara signerade av Sun Microsystems, detta för att garantera att leverantörerna har exporttillstånd av krypto. Till hösten förväntas SecureRMI släppas vilket är en utökning av det redan existerande RMI, Remote Method Invocation, men med inbyggd kryptering och autentisering, ungefär på samma sätt som SSL fungerar för sockets.

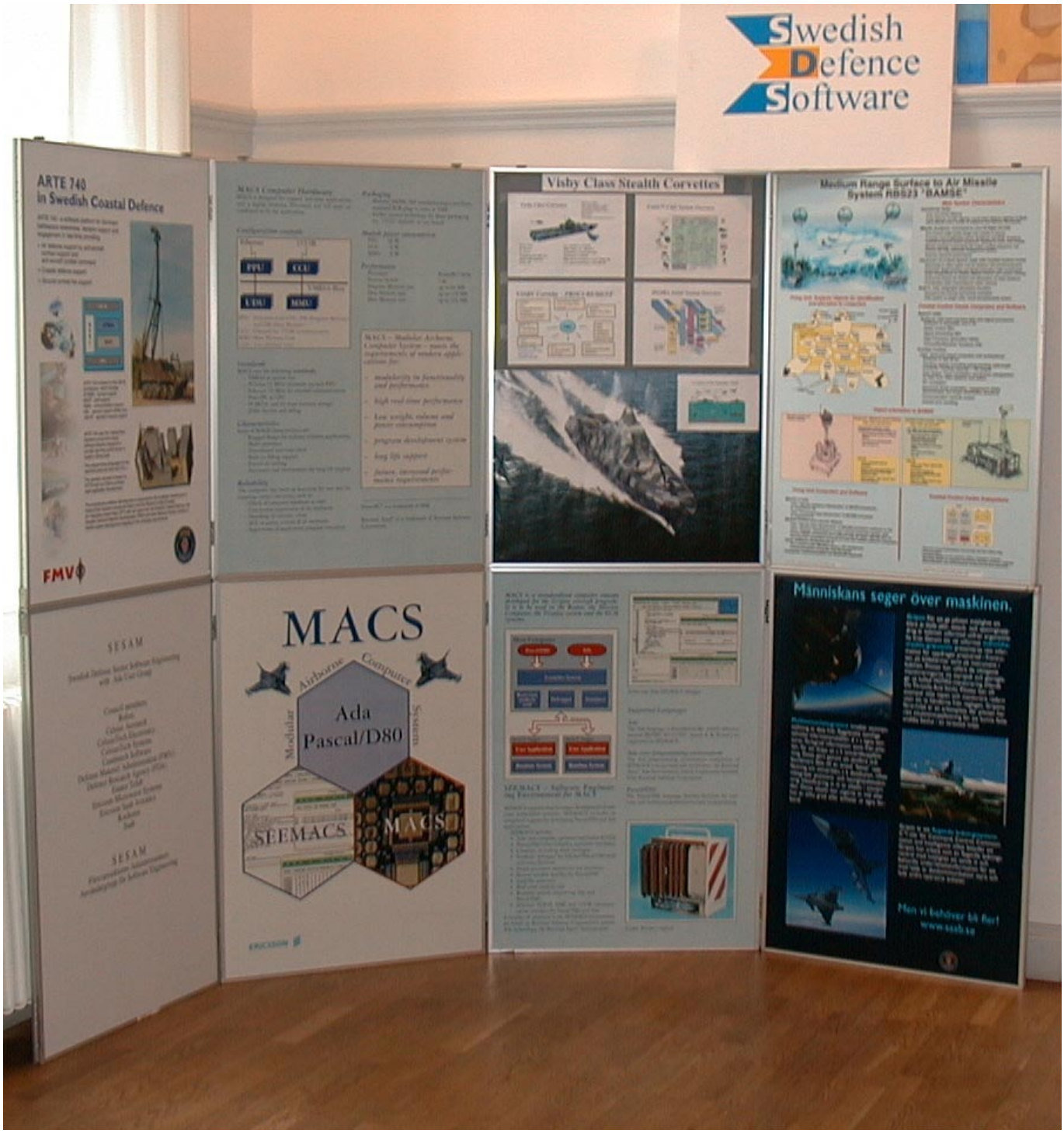
Inom realtidsområdet presenterades en första utgåva av ”The Real-Time Specification for Java”. Specifikationen bygger på bakåtkompatibilitet och förfinar Java specifikationen på en del punkter samt adderar nya klasser utan att utöka syntaxen.
Åsikterna om specifikationen gick dock isär. Enligt vissa är detta den specifikationen som gäller medan andra ansåg att boken ska klassas som ”0 Ed”. Troligtvis kommer det att finnas en specifikation samt implementering man kan tro på lagom till JavaOne nästa år. Redan nu fanns det dock en implementering som klarar av att styra två robotar som spelar piano.

Jini har slagit igenom ordentligt och används nu inom ett antal vitt skilda områden. Bl.a. höll US Army och Motorola varsin presentation om användningen av Jini i militära system.
Ett viktigt påpekande som återkom under flera Jini-presentationer var att Jini inte kräver att hela systemet ska vara skrivet i Java. Det räcker att den kod som klienten får av servern är Javakod. Hur servern ser ut och vilken mekanism som klientkoden använder för att prata med servern har ingen betydelse.

SESAM på KTH-seminariet

KTH var värd för den 12e ”Euromicro Conference on Real-Time Systems” den 19-21 juni. SESAM var en av sponsorerna, vilket berättigade oss till att sätta upp ett planscharrangemang i anslutning till konferenslokalerna. Ungefär hur detta tog sig ut framgår av bilden. Det som fattas där är en plansch om CK37, vilken togs med för de historiskt intresserade och för att ge perspektiv till de tre planscherna om MACS (D96), D80s ersättare i JAS 39.

De som undrar om höjdpunkterna i konferensen, rekommenderas kontakt med Dag Folkesson, som i pauserna också försvarade SESAMs färger på utställningen.



First summer school in Engineering of Complex Technical Systems i Skövde

I augusti arrangerade Högskolan i Skövde en sommarkurs, med professor Sten Andler som primus motor (i mitten på bilden till höger). Arrangemanget sponsrades av NUTEKs program för ”Komplexa Tekniska System” och ARTES (A Network for Real-Time research and graduate Education in Sweden).

Själv kunde jag delta i egenskap av deltagare i NUTEK-programmet.

Kursen var upplagd som en doktorandkurs och flertalet deltagare var också doktorander från svenska universitet. Två erfarenheter kändes mest viktiga efter kursen:

- Många tekniker från forskningens framkant, som är relevanta för oss som arbetar med komplexa tekniska system, presenterades.
- Industrin (med undantag för Volvo) lyste med sin frånvaro.

Eftersom jag tror att många Sesammedlemmar skulle haft glädje av att delta kommer jag att referera några av punkterna i Rendezvous och inledningsvis ge en översikt. Förhoppningsvis kan den intresserade få lite ledtrådar till var man hitta relevant information. Dessutom kanske detta kan bidra till en bättre närvaro från industrin om Högskolan lyckas upprepa kursen nästa år.

Inledningsvis en översikt av kursens syfte (kursspråket var genomgående engelska):

Syftet med sommarskolan är att utbilda forskare och studenter om innebörden av komplexitet och utmaningarna i att bemästra denna genom att öka kunskapen om och förståelsen av komplexa tekniska system. Speciellt koncentrerar vi oss på komplexa pålitliga system, som kräver arkitekturellt stöd för distribution, feltolerans, långsiktig lagring av komplexa data med behållande av realtidsegenskaper. Problembilden bearbetas från ett realtidsperspektiv.

Själv tycker jag detta låter som en beskrivning av ett av de viktigaste områden för försvarsindustrin och att denna kunskap inte skall reserveras för forskare och studenter.

Förste talare var professor Jan Bosch från universitetet i Karlkrona/Ronneby, numera på väg till universitetet i Groningen. Han talade om arkitekturell design av programvara med speciell inriktning på återanvändning med ”produktlinjer” och avvägning av arkitekturer med hänsyn till kvalitetsfaktorer. I anslutning till föreläsningarna fick vi elever öva praktiskt på att utforma och anpassa en programvaruarkitektur mot två övergripande kvalitetsfaktorer.

Andra dagen kom Dr. Matthew Chalmers från universitet i Glasgow och talade om visualisering av information, ett ämne av avgörande betydelse för oss alla, som har att göra med människa-maskingränssnitt. Bl. a. förklarade han varför 2D-presentation ofta är att föredra framför 3D-presentation och hur man kan utnyttja den mänskliga förmågan att observera avvikelser.

Nästa föreläsare var vår värd, professor Andler, som berättade om sin forskargrupps resultat inom området ”distribuerade realtidsdatabaser” (DeeDS Real-Time Database Architecture). Han lyckades belysa problemen på detta svåra område och dessutom göra troligt att forskningen har lett fram mot en fungerande lösning, som kan vara högst industriellt intressant.

Professor Per Stenström från Chalmers talade om hur man kan förstå ”flaskhalsar för prestanda” i komplexa datorsystem. Detta handlar om resultat från en omfattande forskning vid Chalmers med sådant som varför prestanda kan gå ner när man ökar antalet processorer, hur man utnyttjar simuleringar och hanterar stora databaser. Viktiga slutsatser var att prestanda är en komplex funktion av samverkan mellan flera la-

ger av programvara och maskinvara samt att simuleringar är en bra hjälp att klarlägga denna samverkan för konstruktören.

Andra dagens föreläsningar avrundades av professor Benkt Wangler från KTH som talade om en arkitektur för ”processmäklare” i systemintegration. Han hanterade ett problem som är väl känt i FM tillämpningar, men från en civil utgångspunkt: Hur skall man åstadkomma integrerade system från en situation där man har ett stort antal fristående system, som vart och ett hanterar en delfunktion inom en komplex verksamhet? Lösningen som presenterades började med att man inför en meddelandeförmedling för att sedan utveckla den mot en situation med integrerade verksamhetsprocesser.

Tredje dagens föreläsningar inleddes av professor Mario Barbacci från SEI i USA, som har ett mångårigt samarbete med Högskolan i Skövde. Hans huvudämne är avvägningsanalyser av programvaruarkitekturer, speciellt i nära samverkan med intressenter (stakeholders) och med utnyttjning av scenarios för bedömning.

Det här handlar om en intressant och välbeprövad teknik, som vi elever fick tillfälle att prova på med en övning, baserad på ett exempel från forskningen i Skövde.

Efter lunch kom professor Mike Hinchey från Australien och talade om formella metoder och inledde med några väl valda och tänkvärda citat, t ex. ”Hoare’s law”:
I varje stort program finns det ett litet program som försöker komma ut

Sedan fortsatte han att berätta formalism och matematik och lite om hur barnsliga, de som inte gillar formalism, är. Presentationen gav en god inblick av ett sätt att tillämpa formella metoder, men också (oavsiktligt) en del förståelse för varför de formella metodernas segertåg låter vänta på sig.

Den tredje dagen hade arrangörerna insett att eleverna riskerade att börja krokna, så dagen avslutades med panel med föredragshållarna som talade om principer för att utveckla komplexa tekniska system. Därpå följe en utmärkt middag, en social övning i form av bowling och avslutningsvis ett pubbesök där resultatet från bowlingtävlingen klarades ut.

Torsdagen började med att Mike Hinchey visade ett praktiskt formellt exempel på hur man kan specificera ett flygbokningssystem formellt, med notation från språket Z.

Sedan kom professor Jeff Ofut från George Mason universitetet och en första föreläsning om integrationstestning. En första insikt var att han använder ordet ”test” för det somliga skulle kalla ”verifiering”, dvs även revyer (granskningar och genomgångar) anses ingå i testbegreppet. Den fortsatta föreläsningen visade klar hur det lönar sig att integrera provningsarbetet genom hela utvecklingsprocessen.

Speciellt intressant var när han gick in på testproblematiken i anslutning till objektorientering och speciellt polymorfism. Det intressanta här är hur till synes enkla objektorienterade konstruktioner tycks kunna ge i det närmaste hopplösa testsituationer.

Torsdagseftermiddagen började med att professor Claes Wohlin från Lund (På väg till Karlskrona-Ronneby) presenterade forskningsresultat. Den enkla grundtanken var att misslyckade IT-projekt får mycket publicitet och orsakar mycket forskning, medan man inte studerar lyckade projekt så mycket. Han hade tagit ett antal lyckade projekt och studerat vilka faktorer man kunde hitta, som kunde vara gemensam för de lyckade projekten. Det viktigaste resultatet syntes vara en fungerande teknik för att jämföra projekt och hitta gemensamma faktorer och en försiktig slutsats är att kvaliteten på planeringen tycks vara av avgörande betydelse för om ett projekt skall lyckas.

Sedan anknöt Jonas Mellin från Skövde till testningen och berättade om provning av händelsestyrda distribuerade realtidssystem.

Torsdagen avslutades med fortsatt övning i scenariobaserad avvägning av arkitekturer enligt professor Barbaccis metod för arkitekturell avvägning.

Själv hade jag inte möjlighet att delta i den övningen eftersom jag hade tillfälle att demonstrera hur man kan använda verktyget Tofs för att modellera arkitekturer med sin tillhörande information.

Vid fredagens avslutning fick doktoranderna sina diplom från kursen och Sten Andler visade bilder från kursen, som även ses på hemsidan med adress:

www.his.se/ida/conf/ects2000/

Min viktigaste slutsats efter veckan i Skövde är att det här är ett bra och viktigt initiativ som kan visa sig ha avgörande betydelse för vår förmåga att hantera komplexa system i Sverige. Det är bara att hoppas att det blir möjligt att utveckla detta till en tradition. Det viktigaste är naturligtvis att programgruppen orkar upprepa sin jätteinsats och det vore bra om de kan få uppmuntran genom ett bättre industriintresse under kommande år. Själv tänkte jag bidra genom några lite djupare beskrivningar av de områden jag tyckte mig begripa i kommande nummer av Rendezvous.

Ingmar Ögren
iog@toolforsystems.com

Sommarskolan i Skövde: Professor Jan Bosch om programvaruarkitekturer och produktlinjer.

Den förste huvudtalaren vid konferensen inom NUTEKs KTS-program var professor Jan Bosch (JB) från Karlskrona-Ronneby, numera Groningen. Jan Bosch tog upp flera ämnen av avgörande intresse för komplexa programvarusystem inom försvarsvärlden, som sammanhanget mellan programvaruarkitektur, återanvändning med produktlinjer och optimering/avvägning av arkitekturer. Tiden begränsade djupet och detta gäller även för detta lilla referat, varför webplatsen www.ipd.hk-rse/bosch rekommenderas. Den innehåller bl. a. information om Jan Bosch nya bok i ämnet

JB inledde med att referera en imponerande bakgrund med både omfattande forskning och gediget samarbete med flera ledande industriföretag. Det handlar om ett övergripande synsätt på programvaruarkitekturer och -produktlinjer.

Traditionellt programvaruutveckling görs med en tillämpning i taget och fokuserar på leveranstider, med åtföljande problem vad gäller kvalitet och underhållskostnader. Med återanvändning kan man både drastiskt reducera kostnaderna och förbättra kvaliteten.

Återanvändning kan vara opportunistisk eller planerad och den kan göras ”uppifrån” där man utgår från en övergripande struktur eller ”nedifrån” där man utgår från en samling komponenter. Praktisk erfarenhet visar klart att det bara är planerad återanvändning som görs utifrån en övergripande arkitekturell struktur som är praktiskt möjlig.

För återanvändning av programvarukomponenter finns det tre nivåer med olika mogenhetsgrad:

- Återanvändning av komponenter i senare versioner av en produkt. *Detta kan vi.*
- Återanvändning av komponenter i olika produktversioner och olika produkter. *Detta börjar vi lära oss.*
- Återanvändning i olika produktversioner, olika produkter och olika organisationer. *Här har vi mycket kvar att lära, med undantag för begränsade områden som t. ex. Visual Basic.*

En industriell trend är att återanvändbara tillgångar inom industrin tenderar att vara stora och med komplext innehåll (ibland mer än 80 KLOC).

En definition av begreppet programvaruarkitektur: *Arkitekturen för ett program eller ett datorsystem är systemets struktur eller strukturer, som innehåller programvarukomponenter, de externt synliga egenskaperna hos dessa komponenter och relationerna mellan dem* [Bass et al. 98]

Programvaruarkitekturer kan användas för att stödja konstruktion, implementering och underhåll. Det finns en skillnad mellan forskningsvärlden och industrin i det att forskarna tenderar att arbeta med mera specifika arkitekturer, medan man i industrin ofta nöjer sig med en konceptuell förståelse.

En produktlinje för programvara innehåller:

- en produktlinjearkitektur
- en uppsättning återanvändbara komponenter
- en uppsättning produkter, som ansluter till produktlinjen.

(Detta ligger mycket nära vad som kallas ”Domain Engineering” vid Software Engineering Insitute)

JB fortsatte sedan med hur man utvecklar en programvaruarkitektur (SAD = Software Architecture Design) med fyra iterativa processer med utveckling av produktversioner med iterativ arkitekturell konstruktion och produktutveckling. Dessutom med iterativ bedömning och transformation av arkitekturen mot kvalitetsfaktorer.

Metoden innehåller tre huvudfaser:

- Konstruktion, baserad på krävd funktionalitet
- Bedömning av kvalitetsattribut
- Transformation av designen, med hänsyn till kvalitetsattributen.

Kvalitetsattribut är ett viktigt begrepp och man kan utveckla ”scenarioprofiler” för att få ett underlag att väga kvalitetsattributen mot. Som exempel kan man bedöma prestanda mot en användningsprofil och säkerhet mot en riskprofil.

Transformering av en arkitektur innebär att man ändrar strukturen på en arkitektur utan att ändra funktionaliteten. Transformeringen påverkar således arkitekturens förmåga att uppfylla de krav, som är relaterade till kvalitetsattributen. Det går så till man för varje kvalitetsattribut:

1) hittar ställen där kraven som är relaterade till attributet inte är uppfyllda
2) väljer en lämplig transformation av arkitekturen, t. ex. genom att flytta funktionalitet
3) genomför transformationen.

JB gick sedan in på att visa hur man gjort i några verkliga fall som handlade om brandlarm, mätsystem och dialyssystem. Arkitekturerna för respektive system visades som blockschemor i SA-tradition och inklusive icke-programvarudelar av systemen.

Sedan kom en diskussion om ”arketyper”, dvs centrala systemgenerella begrepp. För att identifiera arketyperna börjar man med att söka kandidater för att sedan välja en liten stabil uppsättning och därefter identifiera och välja relationer mellan arketyperna. Exempel på arketyper är:

- För brandlarmsystemet: punkter och avvikelser
- För dialyssystemet: don och styrenheter.

Arketyperna ritades i UML klassdiagram.

Sedan visade JB hur man från arketyperna kan gå vidare mot att få fram delar till sin arkitektur som gränssnitt, domän(er), abstraktionslager, entiteter per domän och instansieringar av arketyper. Därifrån kan man sedan gå vidare mot att beskriva instansieringar av sitt system. Här gäller det att arbeta även med systemets omvärld.

JB återgick sedan till arbetet med bedöming av kvalitetsattribut med dialyssystemet som exempel. Det finns flera tekniker för bedömning:

- Scenariobaserad bedömning, där man arbetar med scenarios för att belysa krav, som är relaterade till de olika kvalitetsattributen.
- Simuleringar där man arbetar med en prototypimplementering av sin arkitektur tillsammans med en simulerad omgivning.
- Matematisk modellering, som kan ses som en ”skrivbordssimulering”
- Erfarenhetsbaserat resonemang.

JB återgick sedan till transformering av arkitekturer med tekniker för att:

- genomföra en arkitekturell stil
- genomföra arkitekturella mönster
- tillämpa konstruktionsmönster
- omvandla kvalitetsrelaterade krav till funktionalitet.

Som exempel på stil visades i brandlarmsystemet hur man kunde välja alternativa arkitekturer med mer eller mindre direkt koppling mellan sensorer och utorgan.

När vi kommit så här långt hade vi elever fått mycket kunskap, det viktigaste kanske insikten att man kan ”transformera” arkitekturer, med behållen funktionalitet för att anpassa till krav relaterade till kvalitetsfaktorer. Nu var det dags att tillämpa kunskaperna på en övning och jag kom i en grupp som valde exemplet ”stoppa bedrägliga samtal” i samband med drift av telefonnät. Med ”bedrägliga samtal” menas olovliga samtal av typen samtal från stulen mobiltelefon eller samtal från nummer med obetalda samtalsräkningar. I förutsättningarna ingick att algoritmer för att indikera bedrägliga samtal fanns tillgängliga och en operatör skulle finnas för att ställa in förutsättningarna för detektering och avstängning av de bedrägliga samtalen. Vår grupp började med att definiera programvarans omgivning genom att utgå från systemets övergripande uppgifter som vi fann vara:

- Identifiera möjliga bedrägliga samtal (Identify potential fraud calls)
- Avsluta bedrägliga samtal (Terminate fraud calls)

Stop_fraud_calls
Identify_potential_fraud_calls
| Operator_to_set_criteria
| | Graphical_user_interface
| | | Criteria_control_software
| | | | Database
| | | | Network.Network_command
| | | Netcom_software
| | | | Database
| | | | Network.Fraud_indicator
| | | Surveillance_software
| | | | Database
Terminate_fraud_calls
| Netcom_software -”-

Den här strukturen kunde vi sedan använda för att transformera arkitekturen med hänsyn till kvalitetsfaktorer där vi skulle behandla två kvalitetsfaktorer: prestanda och underhållsbarhet. Eftersom ingen tid fanns för något annat, så valde vi att arbeta med ”logiskt resonemang” där vi utgick från två antaganden:

- Underhållskostnaden blir minimal om vi kan samla funktionaliteten i ett fåtal centrala programmoduler (GUI, Criteria control software och Terminate fraud calls)
- Maximala prestanda med minimal belastning på kommunikationsnätet får vi om vi flyttar ut så mycket funktionalitet som möjligt i nätet (växlar och basstationer).

Den här analysen visade att det finns en konflikt mellan de två kvalitetsfaktorerna, så vi behövde en strategi för vidare transformering av arkitekturen:
Börja med en centraliserad struktur, med praktiskt taget all funktionalitet i de centrala programmodulerna. Gå sedan vidare med kravanalys, simuleringar, arbete med scenarios etc. med koncentration på prestandakraven och flytta ut funktionalitet i nätverket till växlar och basstationer.

På så sätt bör det vara möjligt att få en optimal avvägning mellan de bägge konfliktande kvalitetsfaktorerna.

Ingmar Ögren
iog@toolforsystems.com

Twenty Years of Safe Train Control in Sweden

Harold W. Lawson
Lawson Konsult AB
bud@lawson.se

Sivert Wallin
Teknogram AB
swn@teknogram.se

Berit Bryntse
Teknogram AB
bbe@teknogram.se

Bertil Friman
Friman Data Konsult AB
bertil.friman@fdk.se

Abstract

The properties of the Automatic Train Control System that has provided a reliable and safe function in Sweden since 1980 are described. Via an engineering view of the problem domain, an architecture evolved in the mid-1970s that has been a key factor in the success of ATC. ATC version 1 functioned properly from 1980 to 1993 without a single change in the software. Since 1993, ATC version 2 has continued this outstanding record and has been adapted for new markets and new requirements. In Sweden, there are approximately 1000 ATC locomotive installations of the on-board system. The operating system core has been re-utilized several times for new product versions as well as the "black box" recorder and more than 20 ATC simulators. ATC is examined from the architectural, development and maintenance as well as the verification points of view. Finally, lessons learned from ATC as well as further usage of the concepts in Sweden are reviewed.

INTRODUCTION

The safety of millions of train passengers is dependent upon reliable safety related equipment and functions in the entire railway system. One of the important functions is the monitoring of the behavior of train drivers; that is, assuring that they abide by speed limits, signal status and other conditions. There have been several train accidents in Europé and elsewhere during the past twenty years where the proper operation of this function would have hindered these incidents. This function, now often referred to as Automatic Train Protection (ATP), has been implemented since the late 1970s in Sweden as the Automatic Train Control (ATC) system.

In this paper, key properties of the Swedish ATC system are presented. In particular, the on-board system conceived and developed by Standard Radio and Telefon AB in the 1970s, now owned by ATSS (Ansaldo Transporti Signal System), and further developed and maintained by Teknogram AB will be examined. The major reasons for the success of this system in providing safe train control for over 20 years are cited.

The paper is co-authored by four people who have participated in different roles in respect to the Standard Radio ATC system; namely, as architect, developers and maintainers, and verifier of the most recent versions of the software.

ATC FUNCTION AND ENVIRONMENT

The ATC on-board equipment is used in conjunction with railway trackside equipment, such as speedboards and signaling system, to maintain and increase the safety of the trains, and also to increase the capacity of the railway system.

To meet the demands of increased efficiency of railway transportation on both existing and new tracks, the train speed must be increased and the trains must operate with shorter intervals. This requirement increases the demands on both the safety system and the train drivers thus leaving little room for human errors. The high degree of accuracy of the ATC system minimizes the risks for driver error. The ATC should be considered as being a complement to the existing optical signal systems. The primary intention is not to control the driver but to lighten his workload, and provide supplementary information that is not available in the optical signal system.

Initially (in 1980 when the first ATC systems where installed), the plan for the Swedish state railways (SJ) was that the train should be driven entirely according to the external optical signals, and that the ATC system should be considered only as a safety back-up. With the advent of the X2000 high-speed trains (200 km/h), it turned out that the optical system was insufficient for presentation of all information needed, e.g. earlier warning for restrictions ahead, and different speeds for various train types. Also, after operational experience with the ATC system

had been accumulated, it turned out that the ATC system could be trusted for presentation of information not otherwise available along the track.

The resulting system is a very efficient, robust, and safe combination, well matching more expensive and complicated systems being used elsewhere.

If the driver should lose concentration for a moment, the ATC will then take over the control of the train by applying the brakes. This brake application continues until the driver manually acknowledges to the system that he is once more capable of controlling the train. If the driver should fail to regain control, the ATC will continue to brake the train to a standstill.

On-board equipment

The vehicle on-board equipment is portrayed in Figure 1 and consists of the following major components:

- An antenna mounted underneath the vehicle that activates the track equipment (transponders) by transmitting a continuous signal and receiving transponder messages to be evaluated and used to supervise the safe travel of the train
- A set of computer equipment that evaluates the transponder messages, presenting the information to the driver and braking the train to a safe speed level if the driver should fail to take the correct actions, i.e. not braking the train or exceeding speed limits. The driver has to manually cancel each ATC brake application by pushing a brake release button.
- Cab equipment consisting of a driver's ATC panel used by the driver to enter into the ATC system the data that is relevant to that specific train, and all other communication with the ATC equipment. The panel also keeps the driver informed of current speed limits and target speed limits at speedboards and signals ahead.
- Vehicle interfacing devices, such as speedometer connection, main brake pipe pressure sensor and one or more brake valves.

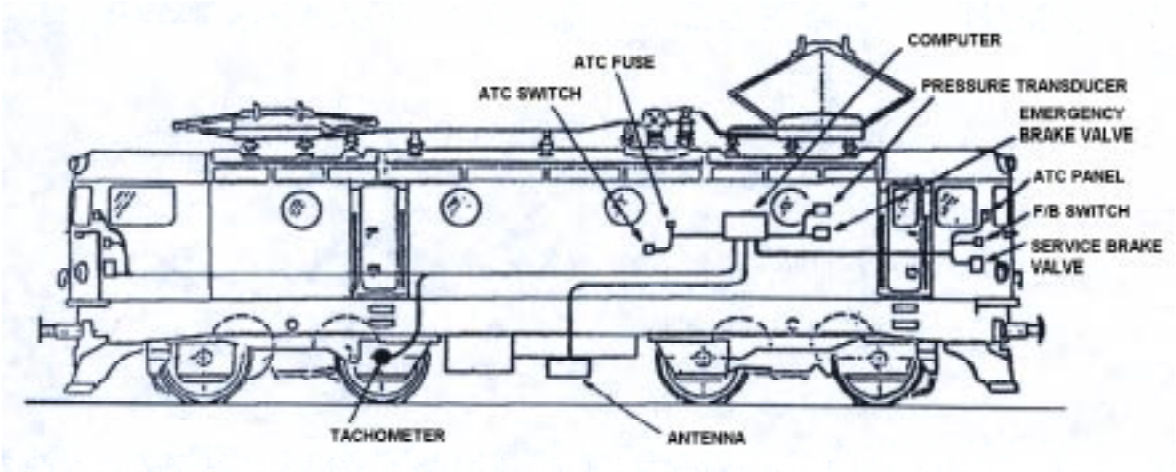


Figure 1. On-board ATC Equipment

Transmission Equipment

The wayside equipment consists of track mounted transponders transmitting messages (telegrams) to the vehicle when activated by the antenna of the vehicle (see Figure 2). Each type of information generates a unique message (telegram). The transponders are combined into groups of minimum two and maximum five transponders. A transponder group can be valid for the current or the opposite direction of travel, or for both travel directions.

The transponders in a group can have either a fixed code or be coded by an encoder connected between the signaling system and the transponder, in such a way that the transponder group can give information corresponding to the current signal aspect to the on-board equipment.

When a vehicle with an active ATC travels over a transponder group, each transponder will be activated by the energy received from

the antenna of the vehicle. The coded message is continuously transmitted to the vehicle equipment as long as the transponder is active.

A valid combination of transponders will transmit all the information necessary for the vehicle equipment to evaluate the message and take the required action. The on-board equipment will detect either a faulty message or an invalid combination of transponders and notify the driver accordingly.

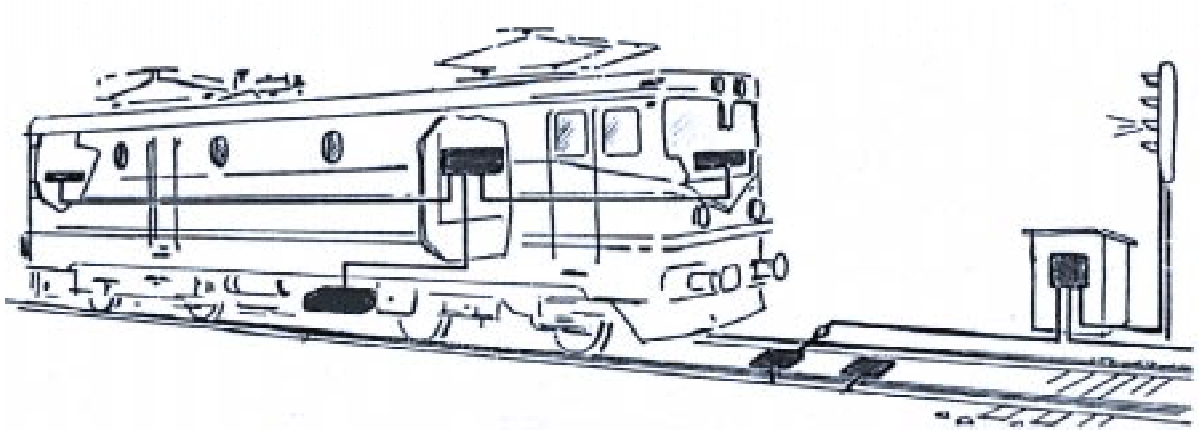


Figure 2. ATC Transmission System Overview

THREE PROCESSOR REDUNDANCY

In order to provide for fault-tolerance, a three-processor solution with majority logic comparison of outputs is utilized for the on-board system. The same program is executed on all three processors thus the redundancy protects primarily against processor hardware failures.

TIME LINE FOR THE PRODUCT

The Standard Radio and Telefon ATC product became the property of ATSS (Ansaldo Transporti Signal System) in 1990. Since 1984, a significant part of the further development of the product and maintenance has been contracted to Teknogram AB. In the following time line, major product events are presented.

1973 Standard Radio decides to enter the train control market place
Swedish State Railways (SJ) requests proposals on a transmission system

1974 Standard Radio, Philips, Ericsson Signal develop transmission solutions

1975 SJ selects the Ericsson Signal approach for the transmission system
Standard Radio starts work on an on-board system concept
SJ favors the Standard Radio on-board mechanical structure
Work on the software architecture concept begins

1976 A problem related architecture evolves
Guidance for development, production, testing, and maintenance

1977-79 Standard Radio selected for the on-board system for SJ trains
Development, testing and verification
Contract to Ericsson Signal for on-board system for SL trains only *
Integration of transmission and on-board systems followed by validation

**SL – Stockholm's Local Traffic. Utilizes a different on-board solution based upon N-version programming. Three different program solutions deployed on each of the three processors. Inherited by Elektrisk Byrå AB, ABB Signal AB and finally Adtranz AB.*

1980 Installation of ATC1 on SJ locomotives.

1980-93 ATC1 operates successfully without any changes in software

1988-92 ATC2 plan: SJ, NSB*, EB-Signal, Standard Radio-ATSS, Teknogram
Further development based upon ATC1, testing, verification, validation

*NSB - Norweigen State Railways

1993 Installation of ATC2

After this, two further developments of ATC2 have led to ATC2.1 developed especially for the Västervik line where a radio based control instead of the transponder system was employed. Further ATC2.2 that was developed to be integrated in the locomotives that will be traveling over the Öresund bridge. In this case, Teknogram has also developed an interface PC-board and software based upon the same operating system as ATC2 for communication with the Siemens solution utilized on the Danish railways. This system will begin operation during the summer of 2000 when the bridge officially opens.

In addition to the main ATC products, a separate PC-board and software running under the same operating system was developed to function as the "black box" recorder for ATC. The recorder collects information for up to three days of train operation and includes telegram information and all transitions of speed greater than 2 km per hour. The most recent version of the recorder utilizes flash memories. Earlier versions utilized solid state memories that required constant power.

It was hoped by Standard Radio that ATC1 would be an export product. Unfortunately, this market did not materialize. Several potential customers, including British Railways examined the product, but decided not to buy it. Very unfortunate since it has now been proved that it has worked reliably for train traffic for over 20 years. This is a truly impressive record. The cost of one single serious accident would most likely pay for the installation of the system.

Since 1990 the solutions utilized in ATC1 and ATC2 have been further exploited by AT Signal System. ATSS along with Teknogram have been involved in several installations of ATC. The installations have included an ATP (Automatic Train Protection) system for Keretapi Tanah Melayu Berhard of Malaysia (installation 1996), ATP for Hammersley Iron Ore Railways in Australia (installation 1998), the ATC system for Roslagsbanan in suburban Stockholm (installation during 2000), and ASES (Advanced Speed Enforcement System) for New Jersey Transit in USA. All of these systems have been based upon the same architecture and operating system core.

Further, Teknogram AB has successfully utilized the same architecture and operating system to develop and market more than 20 train simulators. Consequently, the ATC architecture has been the basis for the Teknogram business concept.

In summary, a truly exceptional example of reuse of concept and operating system solution for a period of over twenty years.

ATC SOFTWARE STATISTICS

As indicated in the time line, there have been two major versions developed and two minor variations on the second version that have been developed for utilization by the Swedish Railways (SJ). The size in terms of number of procedures, lines of assembly code and number of memory bytes are as follows:

Version	Number of Procedures	Number of Instructions	Number of Bytes
ATC1	157	4116	10365*
ATC2	308	10281	26284**
ATC2.1	313	10523	27029**
ATC2.2	339	11178	29522**

* Motorola 6800 microprocessors
** Motorola 68HC11 microprocessors

The small size, clear structure, and simplicity of the software solution have led to many advantages in respect to verification as well as further development and maintenance as described below.

ARCHITECTURE PERSPECTIVE

In 1975, the consultant services of Harold Lawson were contracted by Standard Radio to assist Roger Andersson, project leader,

and Sivert Wallin, chief designer, in the conceptualization of the architecture. Following a review of the work done to date on the software, Harold Lawson and Sivert Wallin re-examined the fundamental requirements of the ATC function and developed the problem oriented architecture that has successfully provided product stability as well as a sound basis for further development under the entire twenty year life cycle of the ATC product.

The major conceptual aspect of the design is the treatment of the system as being continuous in time as opposed to being discrete and event driven. Given the fact that a 250 millisecond resolution (*dT*) of the state of the train in respect to its environment was determined to be sufficient to maintain stability, it became clear that the simplest approach was to simply execute all relevant processes (procedures) during this period of time. So a cyclic time driven approach became the basis for solution.

This simplification led to the fact that the processors only needed to be interrupted by two events. One interrupt to keep track of time (1 millisecond) and one interrupt when information from a transponder is available. The time in the 250 ms *dT* is more than adequate to perform all processing. Adding more structure to the problem, for example, via the use of an event driven operating system approach would have had negative consequences in terms of complexity, cost as well as reliability and risk thus affecting safety. The fundamentals of the approach were documented in Lawson, 1975. The operating system organization is illustrated in Figure 3.

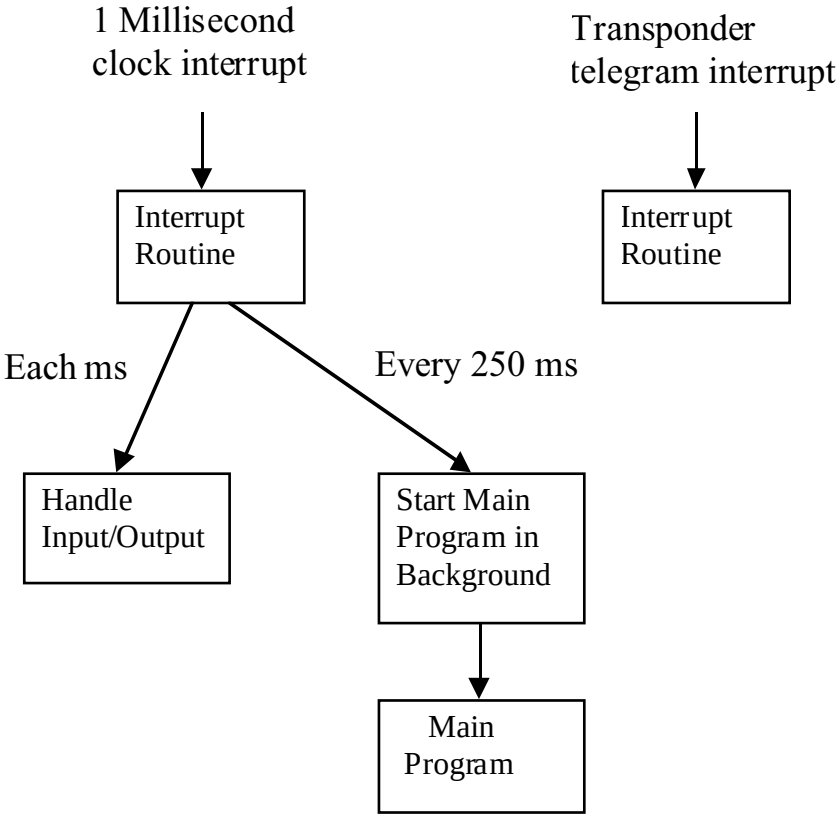


Figure 3. Operating System Structure

As development proceeded, it became clear that treating the application software in a "circuit like" manner made sense and led to highly simplified coding of processes (procedures). While it would have been useful to deploy a higher level language in the solution, it was deemed unnecessary due to the low volume of code that was expected. Experience has indicated that this was a reasonable decision at that time. On the other hand, it was decided to comment the code in a higher level language. In earlier versions of the product, the Motorola MPL (a PL/I derivative) was employed. In later versions, a more Pascal like annotation has been consistently employed. In system tests, MPL, respectively Pascal versions have been executed in parallel with the execution of the assembly language version in order to achieve system verification.

As the product concept evolved, the key factors and goals became evident as documented in a comprehensive software plan by Lawson, 1976.

"A comprehensive plan for the specification, development, testing, verification, production and maintenance of the software components of the ATC project is presented. The goal is to produce reliable software parts to complement the three processor Motorola 6800 system so that a trustworthy total system is provided. A further goal is to assure that the software constituent remains reliable under the life time of the product. That is, that future modifications to the software will not affect the reliability due to oversights concerning design features and software component interrelationships."

…

"The key to a successful software product lies in the ability to decompose the system to be implemented into well defined units such as processes, procedures, blocks, etc. Further, the operation, inputs, and outputs of these units must be well specified and the specification must serve as a control over the implementation, testing, production, and maintenance."

…

"In the ATC project, the process is the unit to which the system structure has been decomposed. A process should be viewed as a testable component, precisely as a hardware component (integrated circuit). It must have a clear specification and have a well defined component test procedures."

…

"A system can never be more reliable than its components and their interconnections. Assuming that each software component has been tested, the interconnections of subsystems of components and finally the total system must be developed, tested, and verified systematically."

Thus, it is clear that even at this early point in the product history conceptualization, the importance of architecture as a controlling factor for the life-cycle of the product was clearly identified. Even though the owners of the product and development and maintenance has changed management, the fundamental concepts established in the mid-1970s are still in place and have led to a successful solution for train safety.

DEVELOPMENT and MAINTENANCE PERSPECTIVE

The early development work was based upon using a PDP-15 computer both for simulation as well as for assembly language translation. The target system based upon Motorola 6800 processors was connected to the PDP-15 so that both procedure and system testing could be well controlled.

Due to the simplicity of the architecture, many advantages were discovered, for example:

- The structure of procedures provided clear points of built-in controls that aided in testing and fault isolation.
- The stack pointer must be returned to the same point in each execution cycle providing a general control of proper cycle execution.
- No wild loops can occur.
- No backward jumps are permitted other than in well controlled loops in procedures.
- Quick reliable changes can be made and verified thus reducing costs.
- The operating system core can easily be reused by removing procedures and incorporating new procedures for new functionality (recorder, simulator).

The target system changed to the usage of Motorola 68HC11 processors for ATC2. The development system was moved from the PDP-15 to a PC based solution that as with the older version also provides for system test via simulation.

VERIFICATION PERSPECTIVE

Verification is carried out via module testing, code inspection, and system test. Early verifications of ATC where carried out by SINTEF at the Technical University in Trondheim, Norway. Bertil Friman has been involved in verification of ATC2, the latest of which is reported in Friman, 1999. The report describes the verification of the ATC2.2 version that will be used for trains crossing the Öresunds bridge between Denmark and Sweden.

Module testing

The software circuit like procedures of the ATC system have, since the beginning of the ATC project, been tested by running them in parallel with equivalent software circuits written in a high level language, and comparing the results. Back in 1975-76 when the original ATC was developed, this was done by connecting the target system (6800-based) directly to the bus of a minicomputer (PDP-15). The high level version was then run on the minicomputer which also was used to control the execution of the target system and to compare the results. The same principle, although more refined, is also used today. The high level version is now written in Pascal and run on a PC computer. The PC computer has direct read/write access to the 64k byte memory space of the target system which is now based on 68HC11. (A similar system based on the 68331 processor is under development.) This configuration makes it possible to test approximately 1000 value combinations per second. Two million combinations can be tested in roughly half an hour. If a software circuit has a small number of input variables, then it can be tested exhaustively. If the number of input variables is large, then the value ranges are limited to values around min, max and close to the decision points in the code.

Code inspection

Back in 1988, when the major revision of ATC that resulted in ATC-2 was started, it was decided that because of the increased complexity of the program, it would be subject to a thorough and detailed inspection. This inspection was contracted to Bertil Friman at Friman Datakonsult AB. The inspection was mainly done by the use of informal proof techniques. A goal was defined, and then an informal proof was built up to see if it was satisfied.

It was soon noticed that most goals were associated with variables and their contents. A (simplified) goal could for instance be that the variable HS (main signal speed) should always be zero after the passage of a stop signal transponder. Since most goals were associated with variables, the goal-proof-technique was successively replaced by a systematic analysis of individual variables. This analysis was done by tracing all places where a variable could be assigned a new value, and for each such place, finding out the real world conditions that were associated with the variable change. These real world conditions could often be directly checked against sentences in the requirement specification.

Associating real world conditions to places in the code where a variable changes value requires an incremental analysis of variables. First variables that only depend on hardware inputs must be analyzed. Then variables that depend on these variables can be analyzed and so on. Sometimes two or more variables can be dependent on each other in a circular fashion. Analyzing such a loop requires more effort because all involved variables have to be analyzed together.

The variable based inspection method has been very successful both for ironing out special case errors and for enhancing the confidence in the ATC system.

Johan Fredrik Lindeberg and Øystein Skogstad at SINTEF encouraged at an early stage the development of CASE tools to support the code inspection. Several such tools have been developed. The most important is VTR (=Variable TRacer) which is directly associated with the variable based inspection method.

System testing

The bulk of the system testing of ATC is done with the use of a simulator. The ATC system is tested by simulating the train start-up and travel on the rails that are equipped with transponders. The simulator has handles, buttons and indicators that correspond to handles, buttons and indicators in the locomotive cabin. The transponders are simulated with a file that contains their positions (from the starting point) and telegrams. A new scenario (use case) is tested by editing a track file and executing the new version on the simulator. After a track file has been changed, it can be run on the simulator instantly. On some occasions, an interesting

scenario has been discussed on the phone and at the same time been tested on the simulator. A superb trouble shooting mechanism. Many parties have contributed track files including Teknogram, ATSS, Banverket and Adtranz. Each track file is accompanied by a specification of how the ATC system shall react at each place on the route. ATSS has an archive containing hundreds of track files that can be used for the validation of new versions of the ATC system.

Quick cycle-time simulation has been a key ingredient in the ATC project since its beginning. The first simulator was a program that ran on the same PDP-15 mini computer that was used to assemble the code. It was directly, over the PDP-15-bus, connected to the development version of the ATC system. Today, the simulator uses a 68HC11 CPU with essentially the same operating system and program structure as the ATC program itself. A PC is used for storing the track files and for controlling the parameters of the simulation through the screen and keyboard.

LESSONS LEARNED

There are several lessons that can be learned from the ATC product experience. These lessons could well be applied in other products, particularly safety critical computer-based systems. Some of the most significant lessons are as follows:

Architecture is a key aspect

The definition and consequent deployment of a problem relevant architecture is a key factor for success. While it is important to have well defined work processes for all life cycle stages of a product, a good architecture reduces the need for heavy processes with multiple activities and tasks. Decision-making is simplified when decisions are bounded by the architectural concepts.

Engineering view is superior to software view

Instead of creating significant quantities of software, an engineering view of the functions to be performed was taken. The analogy between hardware circuits and the logic of the software, later called software circuits (see Hansson, et. al. 1996, 1997) provides a strong, simplifying solution. We can conclude that software, especially in large quantities, is dangerous but can be controlled with the proper engineering viewpoint.

Do not add more structure than necessary

Adding more structure to a solution than necessary for achieving desired behaviors leads to unnecessary complexity thus costs and risks. This pitfall is very common, even for safety critical systems. Operating systems and programming languages that provide elaborate structures for interrupt handling, multi-tasking, etc. complicate verification, further development, and especially maintenance. In addition, complex methods and tools are often deployed. All of these supporting methods and tools implicitly become a part of the product. Together they often are an overkill solution leading to increased cost and risk.

Verification is a vital aspect of safety critical systems

All safety critical systems must be verified in respect to their specifications and safe behavior in various situations. The combination of module testing, code inspection, and system test via simulation has proved to be an adequate approach for ATC. Simplicity in the architecture and code structure simplifies verification and contributes significantly to safety verification.

FURTHER DEVELOPMENT OF THE CONCEPTS

The architectural concepts developed for ATC have been used in other projects in Sweden. During the early 1990s, Harold Lawson the ATC architect participated in the Nutek sponsored Prometheus project for the automotive industry. The engineering view of software was once again proposed as a means of developing the logic for safety critical functions in vehicles in the BASEMENT system (see Hansson, et. al, 1996, 1997). A methodology based upon the use of "software circuits" evolved during this project.

The work on BASEMENT also led to the development, by Arcticus AB of an operating system concept called Rubus (see Lundbäck, Ericsson and Lawson, 1995). Rubus identifies two types of tasks to be performed; namely time driven (called Red) and event driven (called Blue). In relationship to the ATC solution, execution is carried out in time intervals (dT) where the Red tasks are always executed first and time remaining in dT is available for Blue task execution. Rubus has been successfully applied in developing several embedded system products including the Limited Slip Coupling device developed

by Haldex Traction AB and now incorporated in all new Volkswagen automobiles as well as for medical equipment at Siemens-Elema. Arcticus has also produced supporting development tools and utilized them providing embedded systems solutions for Volvo Construction Machines.

Lawson, 1990 and Lawson, 1992b, reported on the importance of architectural philosophy as a key to the engineering of computer-based systems. ATC was cited as one of the case studies in these articles. Ideas related to how to evolve the concepts into a complete resource adequate model called CY-CLONE have been reported (see Lawson, 1992a). Further development of the CY-CLONE model for distributed and parallel execution has also been reported (see Lawson and Svensson, 1993).

CONCLUSIONS

The Automatic Train Control system produced by Standard Radio in the late 1970s has proven to be a successful product. It is based upon an engineering view of the problem domain that led to a straight-forward architecture. The architectural concept has been a key factor in relation to further development, maintenance and verification of this successful product.

The concepts used in the ATC product have been further developed, however, given the success of the approach, it is surprising that more safety critical systems are not constructed in a similar manner.

ACKNOWLEDGEMENTS

Several people have had important roles related to ATC. In this regard, the authors wish to thank Bengt Sterner, Bengt Wenning, Roger Anderson, Johan Fredrik Lindeberg, Øystein Skogstad, Lennart Backing, Folke Nordlander, and Bertil Sjöbergh.

REFERENCES

B. Friman, 1999. Software Validation Inspection Report for Combined Danish-Swedish ATC System Version 2.2, Validation report, June 4, 1999. (ATSS Company Confidential)

H. Hansson, H. W. Lawson, M. Strömberg, and S. Larsson, 1996, BASEMENT: A Distributed Real-Time Architecture for Vehicle Application, Real Time Systems, The International Journal of Time-Critical Computing Systems, Vol. 11, No. 3, November, 1996.

H. Hansson, H. W. Lawson, O. Bridal, C. Eriksson, S. Larsson, H. Lön, and M. Strömberg, 1997. BASEMENT: An Architecture and Methodology for Distributed Automotive Real-Time Systems, IEEE Transactions on Computers, Vol. 46, No. 9, September, 1997.

H.W. Lawson, 1975. Recommendations for Software Organization and Execution Control for the MPU, Consultants Report to Standard Radio and Telefon AB, October 23, 1975

H.W. Lawson, 1976. Preliminary Proposal for a Comprehensive Software Plan for ATC, Consultants Report to Standard Radio and Telefon AB, November 9, 1976

H.W. Lawson, 1990. Philosophies for Engineering Computer-Based Systems, IEEE Computer, Vol. 23, No. 12, pp. 52-63, December, 1990.

H.W. Lawson, 1992a. CY-CLONE - An Approach to the Engineering of Resource Adequate Cyclic Real-Time Systems, Real Time Systems, The International Journal of Time-Critical Computing Systems, Vol. 4, No. 1, February, 1992.

H. W. Lawson, 1992b. Engineering Predictable Real-Time Systems, appearing in Real Time Computing, Springer Verlag, 1994, Lectures from a NATO Advanced Study Institute, October, 1992.

H. W. Lawson and B. Svensson 1993. An Architecture for Time-Critical Distributed/ Parallel Processing, Proceedings of the EUROMICRO Workshop on Parallel and Distributed Processing, IEEE Computer Society Press, January 1993.

K-L Lundbäck, C. Eriksson, and H.W. Lawson, 1995. A Real-Time Kernel Integrated with an Off-Line Scheduler, Proceedings of the 3rd IFAC/IFIP Workshop on Algorithms and Architectures for Real-Time Control, Ostend-Belgium, 1995.

Possible Tailoring of the UML for Systems Engineering Purposes

del 1

(Accepted for publication in the Journal of Systems Engineering)

Abstract

För systemarbete behöver vi ett gemensamt "Systems Engineering Modeling Language" (SEML). Med systemarbete menas då inte bara produktion av programvara utan arbete med kompletta system som innefattar arbete med verksamheter och förmågor med operatörsroller, programvara och maskinvara som samarbetar för att genomföra verksamheten.

Inom programvaruvärlden börjar Unified Modeling Language (UML), som skapats av Rational Inc. få status av standard och ett sätt att skapa SEML är att utgå från UML. Några av kraven på ett SEML analyseras tillsammans med UML och resultatet är att en delmängd av UML, framför allt komponentdiagrammet kan kombineras med delmängd av Ada 95 (pseudokod) för att tillgodose de beskrivna kraven. Speciellt kraven som avser verksamhetens uppdrag (Missions) och förmågor, kombinerat med krav på formalitet och förståelse orsakar problem med omodifierad UML, varför man både behöver begränsa och utvidga UML. Komponentdiagrammet stöder arkitekturella beskrivningar med systemkomponenterna ordnade i beroendeordning, med systemets uppdrag som toppkomponenter.

Resultatet blir ett SEML, som är kompatibelt med UML, som är begripligt för analysatörer och slutanvändare och som är tillräckligt formellt för att genomföra automatiska konsistenskontroller.

För att anpassa originalartikeln till Rendezvous format har den delats upp i tre delar, varav detta är den första, som innefattar det inledande resonemanget, som leder fram till en anpassning av UML.

Introduction

Every other major engineering area (building, electricity, mechanics, electronics, hydraulics, etc.) has an international common language as a basis for mutual understanding among practitioners. For the time being Systems Engineering is different with various languages used to describe systems. This situation is not very satisfactory since it hampers progress and understanding among practitioners and system users.

Until recently the Software Engineering community had a similar situation. Here the problem has now, to some extent, been solved by the advent of the Unified Modeling Language (UML) [5, 6, 7].

For the Systems Engineering area, there are several alternative ways to establish a common modeling language, for example:

1. Use an existing notation, such as for example the IDEF0 notation [1] and tailor it to the needs of the Systems Engineering community
2. Accept the existing UML, not only for software, but also for systems engineering
3. Tailor the UML to meet the requirements for Systems Engineering
4. Invent a new modeling notation for Systems Engineering.

To select an alternative, you need an understanding of what the requirements on an efficient Systems Engineering Modeling Language (SEML) might be. A set of possible requirements is:

1. The SEML shall include modeling of system components of categories Operator, Software and Hardware
2. The SEML shall include modeling of a system's missions and abilities
3. The SEML shall support modeling of a system's structure
4. The SEML shall support modeling of a system's behavior
5. The SEML shall be simple enough for practitioners to learn in a short time and its notation should not be too difficult to explain to end-users.
6. The SEML shall include sufficient formalism to allow automatic analysis.

The alternatives and the requirements are listed and commented in table 1. The table indicates that the two existing alternatives both have problems to meet the requirements listed. To define and market a completely new modeling notation would be a chancy enterprise, which is why the continued discussion concentrates on the alternative "Modified UML"

	IDEF0	UML unmodified	UML modified	New notation
1. Model components	Yes	To some extent	Possibly	Possibly
2. Model missions and abilities	Yes	No	Possibly	Possibly
3. Model system structure	Yes	Yes	Possibly	Possibly
4. Model system behavior	Requires some additional notation to define activity content	Yes, multiple ways	Possibly	Possibly
5. Simplicity	Difficult to distinguish controls from inputs	No, too many diagrams	Possibly	Possibly
6. Formality	Yes for structure but requires a formal notation to define activity content	Requires some extensions as defined by UML tools	Possibly	Possibly

Table 1 Alternatives and requirements for an SEML

The Unified Modeling Language (UML)

The UML was created by the "Three amigos" at Rational Software. Each of the three has an extensive background of Software Engineering and to some extent also Systems Engineering:

- Grady Booch [2] with a background from US Air Force and with long experience as "method guru" at Rational.
- Ivar Jacobsson [8], inventor of the "Use cases" has taught software engineers to start their work by considering how the software is intended to be used. He has an Ericsson background.
- James Rumbaugh [4], who took part in development of the popular Object Modeling Technique (OMT), while being employed by General Electric.

The UML seems to have been defined by the "three amigos" through taking their favorite notations and including them in the new modeling language (with some exceptions). The result is a rich language, which has something for most software engineers. This is why the UML has rapidly become popular in the software world. The richness is also a problem since some concepts can be described in alternative ways within the language. Consequently most practitioners prefer to use a subset of the UML, rather than the full language.

To understand the different diagrams of the UML, you can categorize the diagrams after what they support. Below the diagrams are listed as supporting requirements, structural, communication or behavioral work. For details of the diagrams, see references [5, 6, 7]

The Use Case diagram

The Use case diagram shows the interaction between a user (human or other) and a software system. This is basically a dependency relation where the user depends on the system.

The Use Case diagram primarily supports requirements work.

The Class diagram

The Class diagram is basically an enhanced entity-relationship diagram where the entities are classes with name, attributes and actions. A "software class" can then be seen as a "template" from which instances are created to implement the software. The diagram includes several kinds of relations such as generalization, association, aggregation. The diagram can be used for conceptual modeling, for specification and for documenting a software solution.

The Class diagram primarily supports structural work.

The Sequence diagram

Message sequence charts have a long tradition for visualizing the interaction between concurrent processes communicating by way of messages. These charts are included in

the UML as Sequence diagrams.

The Sequence diagram supports communication.

The Collaboration diagram

The Collaboration diagram resembles Structured Analysis diagrams with its "boxes and arrows". It basically contains the same information as the Sequence diagram why it also supports communication.

The Component diagram

The Component diagram shows dependencies between components and can also be drawn to show aggregation (components included in other components). It supports modeling of system structures, showing how the different components in a system depend on each other.

The Package diagram

The Package diagram shows dependencies between major system components. Consequently it is an alternative for modeling of system structures.

The State diagram

The State diagram is well proven to visualize behavior. It is used to model behavior within a system component.

The Activity diagram

The Activity diagram is an enhancement of the well-proven flow chart. Like the State diagram it can be used to visualize behavior.

The Deployment diagram

The Deployment diagram shows how different software objects are distributed on hardware nodes.

Consequently it is used to supplement structural descriptions.

The listing above shows some of the richness of the UML. It is obvious that definition of a simple SEML from the UML requires that you limit the UML to a subset of the language, possibly with some extensions, wherever the UML does not completely meet the requirements for a SEML. To find the right subset you must first return to the basics of Systems Engineering.

Back to basics

The first concern in system modeling is to find a notation to model the system's architectural structure. You must then decide on how to relate the different structural elements to each other. Some alternatives are:

- Aggregation with the main relation being "contained in". Used in "Structured Analysis" [11]
- Inheritance with the main relation being "is derived from". Supported by the UML Class diagram.
- Communication, with the main relations being "sends to", "receives from" and "invokes". Used in "Structured Analysis" and SADT with the IDEF0 notation [1]
- Dependency with the main relation being "depends on". Supported by the UML Component and Class diagrams with the Component diagram being the simplest since it contains only system components (objects) and their dependencies.

Since you need to model not only system components, but also abstractions such as Missions and Abilities and since you want to keep the modeling notation simple, the obvious choice, within the UML, is the Component diagram.

Figure 1 shows a system outlined as a UML component diagram with components (objects) on different levels and with actions indicated in these objects. From the figure can be understood:

- At the top level, a system can accomplish one or more missions through use of the system's abilities.
- External events may trigger the system to take action, using its abilities within its mission space.

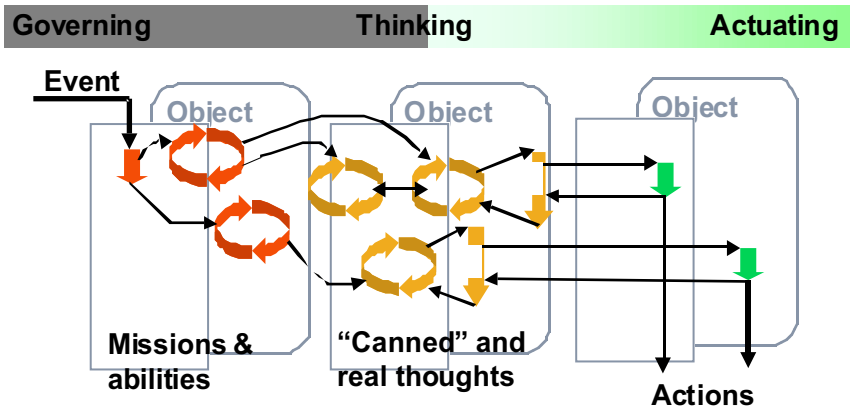


Figure 1 Objects and actions in a system

- An essential part of any non-trivial system's actions
- Actions may be continuos or "single shot".
- Actions may invoke other actions.
- Actions may communicate with other, concurrently active, actions.
- A system will normally deliver actions, which interact with the external world, physically or in some other way.

Note in Figure 1 that the UML Component diagram has been "extended" beyond its stated use to show "how software components depend on each other[5]" in two ways:

- "Upwards" towards the system's operators, and ultimately missions/abilities to include the Operator roles ("real thoughts") and the Missions as objects with Abilities being actions in the Mission objects. This extension makes the separate UML Use Case diagrams unnecessary since the Use cases will be represented as interfaces between Operator objects and objects supporting them.
- "Downwards" towards the system's hardware to include Hardware objects. This extension makes the UML Deployment diagrams unnecessary and also allows for inclusion of non-computer hardware in the system model.

As discussed above it is possible to meet the requirements for structural (architectural) modeling, with inclusion of Missions and Abilities through extension of the UML Component diagram. We then still have to meet the requirements on behavioral modeling and formalism. UML here offers the State and Activity diagrams, both being well proven. However it is doubtful if any of these meet the requirements on formalism and understandability, since they do not include typing to support variables and since non-experts do not immediately understand them. For those, who can read and write the UML behavioral diagrams, they can contribute significantly towards understanding behavior.

An alternative is to start from natural language and find a supplementing standard. Ada 95 [12] is special among programming languages, since it is "talkative" and close to natural language as compared to most other programming languages. Consequently a pseudo code subset of Ada 95 can meet the requirements on formalism and understandability and qualify as "Formalized English". Such a simplified subset of Ada 95 is the Odel design language [9]. Besides simplifications, Odel includes extensions to manage real concurrency as required for Systems Engineering purposes.

Concurrency also needs to be clarified in system models with visualization of the messages between concurrent processes. This is traditionally done with message sequence charts, which are included in the UML as Sequence diagrams.

The conclusion of this discussion is that a combination of UML Components diagrams, UML Sequence diagrams and "Formalized English" (Odel), derived from the Ada 95 programming language, could be used to create a useful notation for systems modeling. To show how this can be done, a couple of system examples are presented, selected from projects where the technique has been applied. One is a simple technical system, which controls the windows of an automobile and one is a more abstract example from the Defense domain, which concerns establishment of "Dominant Battlefield Awareness".

References

1. David Marca and Clement McGowan IDEF0/ SADT™ Business Process and Enterprise Modeling, Eclectic Solutions
2. Grady Booch Software Enginering with Ada. Benjamin Cummings 1983
3. Jean-Pierre Rosen and HOOD User's group. HOOD An Industrial Approach for Software Design 1997
4. James Rumbaugh and others. Object-oriented modeling and design. Prentice Hall 1991
5. The UML Notation guide. Published at http://www.rational.com
6. Martin Fowler & Kendall Scott. UML Distilled. Addison Wesley 1999
7. Craig Larman. Applying UML and Patterns. Prentice Hall 1998
8. Ivar Jacobson. Object-Oriented Software Engineering. Addison-Wesley 1992
9. Odel language description. Published at http://toolforsystems.com
10. Ingmar Ogren. On principles for Model-Based Systems Engineering. Systems Engineering Journal Nr. 1 2000
11. Tom DeMarco. Structured Analysis and System Specification. Yourdon Press 1979
12. Annotated Ada 95 Reference Manual, Version 6.0. Intermetrics Inc. 1995

Fortsättning

I nästa nummer kommer en fortsättning med de två utlovade exemplen och slutsatserna.

Ingmar Ögren
iog@toolforsystems.com

Nytt teknikbevakningsprojekt skall försöka höja läsvärdet i Rendezvous och på hemsidan

SESAM VU kommer vid rådsmötet 19 oktober att föreslå Rådet att ett särskilt projekt i storleksordningen 75-100.000 kr skall drivas för att bevaka nyheter och händelser på programvarufronten, som kan förtjäna att rapporteras i Rendezvous eller på hemsidan. Källor som avses bevakas är viktigare konferenser, artiklar i de större programvaruorienterade tidskrifterna etc.

Avsikten är att ge ekonomiska bidrag till rapportering från konferensdeltagande, till att skriva referat av tidskriftsartiklar och till sammanställning av nyhetsnotiser etc. Konferensbidragen är tänkta att främst täcka extrakostnader för att skriva referat, men i undantagsfall skall även andra kostnader kunna ersättas.

Konferenser som bedöms intressanta att få intäckta är ACM SIG Ada, Ada-Europe, IRTAW, SNART, STC, ICSE, JAVAOne, OOPSLA o d, medan exempel på tidskrifter av intresse kan vara Ada User Journal, ACM Communications, IEEE Computer, IEEE Software, Software Engineering Notes, INCOSE Systems Engineering Journal m fl.

För att hålla samman det hela är meningen att en teknikbevaknings- och redaktionskommitté (TRK) skall bildas. Vidtalade för denna är Torbjörn Andreasson EMW, Ingemar Carlsson och Ingmar Ögren Romet. Under förutsättning att Rådet välsignar förslaget, önskar TRK snarast komma i kontakt med SESAM-medlemmar som är villiga att bidra på sätt som skisserats ovan. En första åtgärd kan vara att försöka få ihop en lista över medlemmarnas planerade deltagande i konferenser närmaste året för att konstatera ”täckningsgrad” och identifiera möjliga rapportörer.

SESAM värvar nya intressenter

I enlighet med tidigare rådsbeslut har inbjudan att gå med i SESAM nyligen gått ut till följande företag med verksamhet i försvarssektorn:

Calisto Data AB
CAP Gemini
Communicator AB
Enea Data AB
Frontec
HiQ International AB
Mandator AB
S&T Datakonsulter AB
WM-data AB

Förhoppningen är förstås att vi därigenom skall få in ytterligare motiverade personer i SESAMs arbetsgrupper och andra sammanhang och kunna bredda verksamheten.

Höstseminariet och anknutna evenemang lovar mycket

SESAM-medlemmarna bör vid det här laget ha fått inbjudan till höstseminariet och därmed sammanhängande tilldragelser. Kortfattat rör det sig egentligen om tre eller fyra skilda evenemang:

18 oktober

Seminariet: Programvaran i Det nya försvaret

Diskussion med middag:" Det framtida försvarsindustriella komplexet i Sverige; kommer det att finnas, hur kan det se ut, under vilka förhållanden har det att verka" med mycket prominenta inledare.

19 oktober

Medlemsmöte i SESAM, med rapportering från interna och FoTA projekt

SESAMs rådsmöte

Programmet i sin helhet kan laddas ner från hemsidan. Alla evenemangen (utom rådsmötet förstås) är öppna för anställda i företag och myndigheter som är medlemmar i SESAM Råd, ej endast för medlemmar i arbetsgrupperna och Rådet. Dessutom har medlemmar i föreningen Ada i Sverige inbjudits delta i seminariedelen.

Du besöker väl vår websajt?
http://sesam.tranet.fmv.se

Kalendern

15-19 okt	OOPSLA 2000: Object Oriented Programming Systems Languages and Applications Conference Minneapolis, MI, USA
18-19 okt	**Höstseminariet**
19 okt	**Rådsmöte**
12-16 nov	ACM SIG Ada 2000 Conference, Johns Hopkins University, Applied Physics Laboratory, Laurel, MD, USA
8 dec	**VU**

SESAM-Sekretariatet: AerotechTelub AB
c/o Kåsjös Kontor
Ytterspåret 14
187 54 TÄBY

Telefon: 08-510 51866
Telefax: 08-510 51932
GSM: 070-716 9702
E-post: alkas@tranet.fmv.se