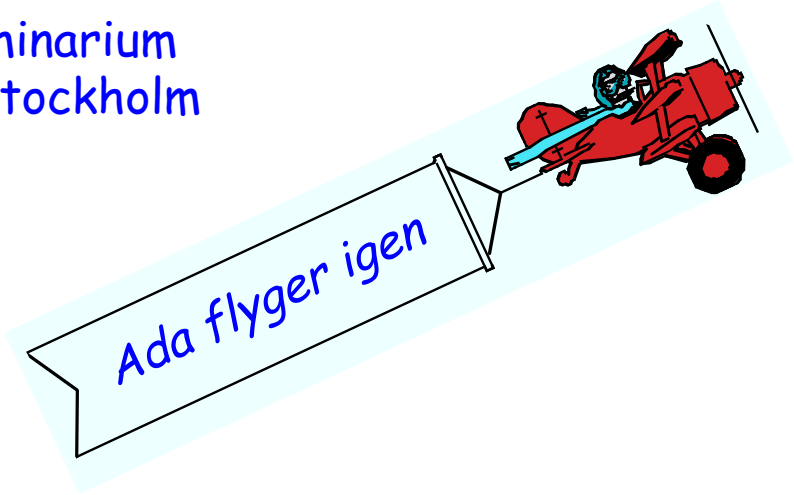
NYHETSBLAD FÖR SESAM
Försvarssektorns Adaintressenters Användargrupp för Software Engineering

RENDEZVOUS

Nr 2 juni 2002

EXTRA
ARKITEKTUR-seminarium
den 28 augusti i Stockholm



Innehåll

Försvarssektorns Adaintressenters Användargrupp för Software Engineering

SESAM

Vad är SESAM?

SESAM är ett samverkansnätverk för projektövergripande och företagsneutral kunskapsuppbyggnad och kunskapsspridning inom området programvaruintensiva försvarssystem.

SESAM skall genom organiserat samarbete mellan dess medlemmar (företag, organisationer, myndigheter och utbildningsinstitutioner) främja tillförlitlighet och effektivitet i utveckling och vidmakthållande av programvaruintensiva försvarssystem.

SESAMs verksamhet att samla, skapa och sprida information och kunskap sker huvudsakligen i arbetsgrupper som verkar inom avgränsade tekniska områden och som efter behov inrättas och avvecklas.

SESAM anpassar, profilerar och förnyar sin verksamhet med hänsyn till ändrade tekniska och andra omständigheter av betydelse för intresseområdet.

När SESAM startade sin verksamhet 1988 var s k inbyggda realtidssystem (i farkoster, sensorer, stridsledningssystem m fl) dominerande inom försvaret och svensk försvarsindustri. Det fanns ett allmänt behov att kunna driva utvecklingen av programvara för dessa alltmer komplexa system på ett mer organiserat sätt, d v s att mer konsekvent praktisera vad som börjat benämnas Software Engineering. SESAMs verksamhet inriktades därför från början på användningen av det speciellt för inbyggda och realtidssystem under början av 80-talet i brett internationellt samarbete framtagna och av ISO 1987 standardiserade programspråket Ada, vilket hade utformats särskilt för att stödja tillämpning av Software Engineering principer. Den fortsatta användningen av Ada, inkl i nya (Ada 95) och framtida versioner, är fortfarande av stort intresse och betydelse för många av medlemmarna i SESAM.

Efterhand har även andra typer av programvarusystem blivit mer förekommande inom försvarssektorn, t ex på informationssystemområdet. Utvecklingen inom kommunikationstekniken, t ex via Internet snabba expansion, har också tillfört nya möjligheter och typer av problemställningar för programvaruutvecklingen. Utbudet av kommersiellt tillgänglig programvara (COTS) som man önskar kunna använda i försvarssystem har också ökat. Inriktningen på uppbyggnad av ett nätverksbaserat försvar, behovet av interoperabilitet vid internationella insatser m m har aktualiserat nya problemställningar i systemutformningen. Denna utveckling gör att nya tekniker, processer och metoder, språk och andra hjälpmedel etc för framtagning och vidmakthållande av programvaruintensiva system av intresse att behandla inom SESAMs verksamhet ständigt tillkommer.

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 AB , tel 013-231652 .

VU

Andersson Tommy, Ericsson Microwave Systems AB,
tommy.andersson@emw.ericsson.se

Bengtsson Christopher, FMV
christopher.bengtsson@fmv.se

Carlsson Ingemar, adjungerad
ingemar.carlsson@mbox2.swipnet.se

Ekman Mats, Saab Aerospace AB
mats.ekman@saab.se

Gustafsson Bengt, SaabTech Systems AB
bbg@systems.saab.se

Hallén Johan, FMV
johan.hallen@fmv.se

Merkell Curt, Saab Bofors Dynamics AB
curt.merkell@dynamics.saab.se

Wadsten Claes, AerotechTelub AB
claes.wadsten@aerotechtelub.se

Arbetet utförs i arbetsgrupper och följande är f n organiserade:

Ag Metodik
Håkan Edler, CTH/Datorteknik
edler@hisafe.se

Ag Teknik
Lars Asplund, Mälardalens högskola
lars.asplund@mdh.se

Vilka kan vara med i SESAM?

Medlemmarna i SESAM är företag, organisationer och myndigheter (förvaltningar, utbildningsinstitutioner etc) i Sverige 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- eller arbetsgruppsmedlem krävs status som leverantör till FMV eller FM. Dessutom krävs en skriftlig förbindelse att uppfylla åtagande som ordinarie- resp arbetsgruppsmedlem. För inträde som informationsmedlem (erhåller endast informationsbladet) krävs status som leverantör till FMV eller FM 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

Detta nummer återspeglar många artiklar kring ett språk som till stora delar var drivmotorn inom SESAM nämligen Ada. Jag själv trodde när jag fick min första kontakt med SESAM att detta var ett språk som inte skulle överleva nya kommande ”moderna” språk och operativsystem. Det dröjde inte lång tid innan jag insåg att detta var ett brist i min värld som jag inte borde förmedla till andra.

Ada är idag flera år senare fortfarande ett språk som används och bör användas i säkerhetskritsiska system. Detta gäller såväl militära som civila system. Vi inom SESAM arbetar ju i första han med militära system men har vid seminarier också visat användare med civil tillämpning som också använder detta språk.

Nu när vi i SESAM utvidgat vår verksamhet till flera språk och frågor som berör programutveckling så får vi inte glömma de problem som Ada löste.

I framtiden kommer vi på alla sätt att hjälpa människan med system uppbyggda av datorer som dessutom arbetar i komplicerade nätverk. För att människan skall acceptera tekniska lösningar så måste man kunna lita på systemet.

Vi har i framtiden betydligt svårare frågor att lösa än de som vi hade när Ada infördes. Vi bör därför titta på hur tidigare problem löstes och överföra dessa till de nya systemen och programspråken så att dessa får de fördelar vi redan har hos Ada.

Claes Wadsten
Ordförande i SESAM

Tfn. dir : 013-23 16 52
Mobil: 070-6000271
E-post: claes.wadsten@aerotechtelub.se

Ada flyger igen

Bo Frisberg
Saab AB
SE-581 88 Linköping
Tel: 013-18 32 18
Bo.Frisberg@saab.se

Under våren lyfte Ada i ännu ett av Gripens delsystem. Då var det nämligen dags för den första provflygningen med GECU (General systems Electronic Control Unit). Ada har tidigare använts i bl a styrsystemet i Gripen. Erfarenheterna är fortsatt positiva, och visar att preemptive scheduling implementerad med tasking kan kombineras med höga krav på tillförlitlighet och ett deterministiskt beteende.

1. INLEDNING

GECU innehåller funktioner från tre olika delsystem som tidigare utvecklats av underleverantörer utomlands. Överföring av systemansvar och kunskaper till Saab, för att ge bättre utvecklingsmöjligheter för framtiden, har varit en drivande faktor vid framtagningen av GECU. Att tre olika "burkar" slås samman till en, innebär också att plats i flygplanet kan utnyttjas för andra ändamål.

GECU har programmerats i Ada 83, precis som tidigare styrsystemet (se artikel i Rendezvous nr 2 1999). Även i GECU har tasking och exception-hantering använts på ett begränsat sätt.

Utvecklingen startade under 1997 och den första provflygningen ägde alltså rum i april i år. Leveranser i serieflygplan planeras till senare i år.

2. SYSTEMBESKRIVNING

Följande tre grundflygplansystem har implementerats i GECU:

ECS (Environment Control System)

Kallas på svenska oftast för luftsystem, eftersom det svarar för luftkylningen av flygplanets elektronik och för att piloten(erna) har en behaglig arbetsmiljö i cockpit (air conditioning). Detta åstadkoms bl a genom reglering av ventiler för styrning av luftflöden.

HS (Hydraulic System)

Svarar i första hand för övervakning av flygplanets hydraulsystem. Larm genereras vid t ex lågt hydraultryck och ev läckage isoleras genom avstängning av en delkrets.

FS (Fuel System)

Förser flygplanets motor med bränsle genom att transferera bränsle mellan olika deltankar (inklusive extratankar som kan hängas i vapenbalkar under flygplanet). FS mäter även hur mycket tillgängligt bränsle som finns kvar, vilket presenteras för piloten. Vidare styr FS tankningen av bränsle på marken före flygning. I framtiden kommer det även att bli möjligt att tanka Gripen i luften från ett tankerflygplan, vilket är väsentligt vid export av flygplanet.

I/O

Till GECU kommer insignaler från ett relativt stor antal sensorer som bl a mäter temperatur och tryck hos luft och hydraulolja, ventillägen, probar för mätning av bränslemängder, m m. Utsignaler skickas till bl a ventiler för att reglera luftflöden, för avstängning av en hydrauldelkrets samt för att transferera bränsle mellan deltankar.

Kommunikation med andra delsystem i flygplanet sker via en 1553 databuss. Dessutom finns direktlänkar till flygplanets displayer (presentation sker numera även i färg), samt diskreta direktsignaler för att kunna tända varningslampor.

Till GECU finns även Ethernet-anslutning, som f n endast används vid inladdning av ny programvara och för utläsning av felinformation. I framtiden kan det också bli aktuellt att använda Ethernet för registrering av data under flygning.

Pentiumprocessor

I GECU finns en Pentiumprocessor med DRAM-minne för exekvering av programmet som vid start laddas upp från ett flash-minne.

Hårdvaran består vidare av tre separata kort för hantering av I/O för respektive delsystem

samt en gemensam kraftenhet med redundant matning av 115 VAC och 28 VDC.

I hårdvaran finns även inbyggd "safe-state"-logik som gör det möjligt att flyga hem och landa på ett säkert sätt i händelse av t ex ett CPU-fel.

Hårdvaran i GECU har utvecklats och produceras av Saab Avionics i Jönköping.

3. ANVÄNDNING AV ADA

GECU är programmerad i Ada 83. Använd kompilator är Rational VADScross med ordinarie run-time system och med Sun Solaris som värdmiljö. Ada används både för den flygande delen av programmet (normalmod) och för den inbyggda testen (BIT) som utförs på marken före flygning.

Den starka typningen och kontrollerade interface mellan paket är kännemärken i Ada som stödjer utvecklingen av tillförlitlig programvara. Abstrakta Data Typer (ADT) används för implementering av grundläggande byggblock.

Programarkitekturen i GECU har till stora delar återanvänts från styrsystemet.

Vidare kan nämnas att en stor del av funktionerna som implementerats i GECU har modellerats i verktyget System-Build. Adakoden genereras inte automatiskt, men modellen utnyttjas som referens vid enhetstestning.

3.1 Tasking

GECU programvara är implementerad i Ada med fem periodiska tasks som körs i följande harmoniska frekvenser: 60, 30, 15, 7.5 och 1 (0.9375) Hz.

Dessa periodiska tasks är tilldelade fasta prioriteter i enlighet med algoritmen för rate monotonic scheduling, dvs högre prioritet till task med kortare periodtid.

För delsystemen ECS och HS sker beräkningarna huvudsakligen i 30 Hz, medan FS arbetar i 1 Hz (förändringar i bränslesystemet sker relativt långsamt).

Scheduleringen är implementerad med ett särskilt task med den högsta prioriteten, som aktiveras i 60 Hz med ett avbrott från en timer (standard Ada interrupt entry). Detta task aktiverar övriga periodiska tasks i rätt intervall med rendezvous. Vidare kontrolleras att ett periodiskt task inte har missat sin deadline. I denna tillämpning innebär en missad deadline att "safe-state"-logiken i hårdvara aktiveras.

Ett bakgrundstask mäter och lagrar exekverings-belastningen för de periodiska tasksen. I bakgrunden sker även checksummering av programminnet.

Även om tasking tillåts, så gäller följande restriktioner (som överensstämmer väl med den s k Ravenscar profilen som definierats för Ada95):

- Alla tasks är deklarerade på biblioteksnivå och skapas vid programmets start.
- Inget task tillåts att terminera, vilket innebär att varje task innehåller en yttre loop. Abort är givetvis inte heller tillåtet.
- Alla tasks har unika och fasta prioriteter (inga dynamiska prioriteter tillåts).

3.2 Designegenskaper

Eftersom det inte förekommer någon direkt kommunikation eller synkronisering mellan de periodiska tasksen, betyder det också att det inte finns någon risk för prioritetsinversion, deadlock eller annan oönskad blockering.

All hantering av tasking hanteras på en exekutivnivå som är skild från funktionerna i själva applikationen. Det innebär att de sekvensiella delarna av programmet kan analyseras och testas separat. Detta är också en väsentlig fördel ur underhållssynpunkt. När gamla funktioner ändras eller nya läggs till, så påverkas normalt inte ramverket (exekutivnivån med tasking).

Vid design av mjukvaran i GECU har det även varit väsentligt att separera de tre delsystemen (ECS, HS och FS) i största möjliga utsträckning. Förutom gemensamma exekutivfunktioner för schedulering och viss hantering av I/O, så är funktionerna åtskilda i programmet.

3.3 Exception-hantering

I GECU hanteras exceptions, men utan avancerad återställning efter ett exception. Exceptions i programmet bör givetvis undvikas så långt det är möjligt. Men det kan vara svårt att förutse alla situationer där ett exception kan uppstå, t ex kan ett exception utlösas av ett hårdvarufel.

Om ett exception uppstår, så registeras alltid typ av fel och var i programmet det inträffade. Exception-information och andra felindikeringar lagras i ett icke-flyktigt minne

för utläsning efter landning.

Exceptions hanteras i GECU på tre nivåer:

- Den allmänna åtgärden efter ett exception är att aktivera "safe-state" i hårdvara (på samma sätt som efter en missad deadline).
- Om ett exception fångas inom ett av de tre delsystemen finns det även möjlighet att isolera felet, så att enbart detta delsystem påverkas och försätts i "safe-state".
- Undantagsvis kan också fel fångas i isolerade procedurer och hanteras lokalt, så att en korrekt normalfunktion fortfarande kan garanteras.

3.4 Ethernet

Inladdning av ny mjukvara i GECU sker via Ethernet. Detta hanteras av ett separat laddprogram som kan aktiveras av tekniker under vissa villkor.

Eftersom samma Ethernet-krets finns i både GECU och systemdatorn, har driver-programvara som utvecklats av Ericsson Microwave i Mölndal kunnat återanvändas på ett framgångsrikt sätt efter vissa anpassningar.

Sockethantering har också programmerats i Ada genom användning av bibliotek i Rationals VADSnetwork.

4. ADA I GRIPEN

Ada var inte med i Gripen-projektet från början, men har nu kommit in i flera delsystem.

Förutom styrsystemet och GECU, kan nämnas att i de PowerPC-baserade D96-datorerna, som även kallas MACS (Modular Airborne Computer System), är det möjligt att ha program skrivna i både Pascal/D80 och Ada. Ada används där bl a i radardatorn samt i ett nytt navigeringssystem (det sistnämnda är implementerat i en av CPU-erna i systemdatorn och beskrevs närmare i Rendezvous nr 3 1999).

Ytterligare delsystem med Ada är under utveckling.

Så sammanfattningsvis kan man säga att det är uppåt för Ada i Gripen.

David Parnas i Stockholm 19 augusti

Professor David Parnas, en av de mest kända profilerna och debattörerna inom programvarutekniken, är huvudtalare vid ett heldagsseminarium om "Improving Software Quality with Improved Documention" på KTH den 19 augusti.

Det är SNART – Svenska NAtionella Real Tidsföreningen – som arrangerar detta industriseminarium i samarbete med att bl a ARTES, Strategiska Stiftelsen och ENCRESS.

Bland Parnas tidigaste insatser var ett projekt som studerade hur man skulle kunna nå kostnadsreduceringar genom att tillämpa Software Engineering metoder på utveckling av programvara för flygplansystem. Där hade han attackplanet A7 (ungefär samtida med vårt AJ37) som studieobjekt. Senare har han förutom ett omfattande författarskap, undervisning etc, bl a ägnat sig åt säkerhetskritisk programvara i kärnkraftverk. Parnas blev också känd bland den teknikintresserade allmänheten genom sin mycket beska kritik av Star Wars projektet, vilket han ansåg inte var genomförbart för att man p g a programvaruomfattningen aldrig skulle kunna testa det eller visa att det verkligen skulle komma att fungera.

Parnas är en mycket populär och engagerad föreläsare, vilket de som får tillfälle att lyssna på honom kommer att få erfara. Det kan nog vara klokt att anmäla sig i god tid.

Information och anmälningsblankett finns på
www.artes.uu.se/events/summer02/parnas.pdf

Use Ada's syntax and semantics for understandable systems engineering

Ingmar Ögren
Tofs corporation, Fridhem 2, SE 76040 Veddoe, Sweden Tel +46 176 54580
email: iog@toolforsystems.com

Abstract

Systems engineering is where you work with complete systems where operators cooperate with software and hardware modules to complete missions. This is of increased interest to software engineers in order to decrease the risk that the wrong software with the wrong requirements is built.

The present trend is to extend the Unified Modelling Language (UML) to cover not only software but also systems. This may lead to a risk for problems with understanding of system models because of many, partly overlapping, diagrams being used.

A proven alterative, which decreases this risk, is to use a few UML diagrams together with Ada 95 syntax, modified for use in Systems Engineering.

Keywords: Systems engineering, UML, Ada 95 .

1 Introduction

Systems engineering is the art of building systems where people cooperate with software and hardware to complete one or several missions.

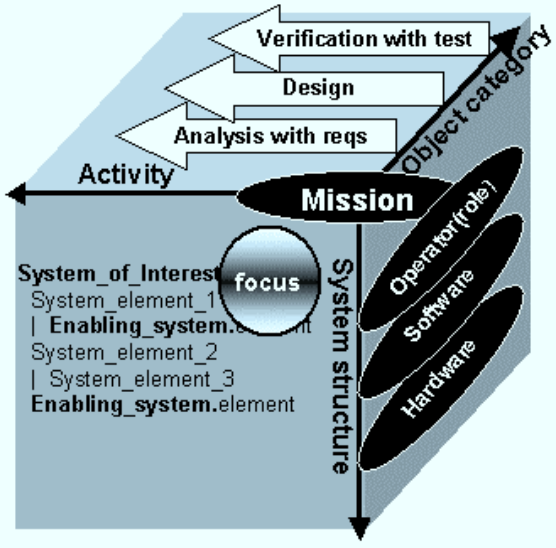


Figure 0, The Systems Engineering work space

Engineering of small systems is normally not a problem. Can often be done "on the side" as part of a software engineering effort. When systems are getting larger and the information grows beyond what is humanly manageable the situation gets different and the development group feels like moving in a "dark three-dimensional space", where you can only focus on a small part of the system at one time.

The three dimensions are:

- Activity, which defines what you do
- Object category, which defines what you are working with
- System structure, which defines where in the system structure you are working.

You may wonder what this has to do with software engineering and Ada. In fact a lot and the two disciplines depend on each other. Software engineering depends on correctly completed systems engineering, prior to programming, in order to ensure that you do not only produce correct software, but that you also produce the right software. This is extremely important since so much software is produced to great cost and then never used.

On the other hand software engineering has been around for a long time and systems engineering should benefit from using much of the hard-won experience from software engineering.

2 Requirements on a modelling language for systems engineering

A key activity in systems engineering is modelling, where you build a model of an existing or planned system. The more or less completed model can then be used for guidance of further work, including software development. Modelling is done in one or several modelling languages. Some of the key requirements on a modelling language are:

Understandability

The is the most important requirement since systems engineering often involves several stakeholders with each of these being an expert in her own field and with little time to learn a modelling language. If the stakeholders do not understand and review the system model, someone will inevitably tell you after you believed the system was completed: "this is no good".

Structure

Just like software, systems can be seen as a collection of co-operating modules with dependencies between the different modules. It is important that the modelling language can define the dependencies and show how the different modules contribute to completion of the system's mission(s).

Behaviour

Each module in a system, be it an operator role, a software or a hardware module, has a behaviour space, which defines its possible behaviour. Consequently system understanding requires the modelling language to be able to describe behaviour.

Communication

Communication and interfaces is a key issue, not only for software, but also for systems work. Consequently a systems modelling language must model both "invocation style" communication and "message passing style" communication (between concurrent processes).

3 Today's solutions

Today various diagrammatic and textual languages are used for system modelling purposes such as IDEF0, Structured Analysis with Data Flow Diagrams, UML (Unified Modelling Language[2, 3, 4, 8]), Entity-Relationship diagrams and natural language.

These solutions all have their advantages, but also problems with the main problem being understandability. What is not always understood is the difference between teaching some students a modelling language for use on a small system and modelling a large and complex system and have the stakeholders concerned understand it although they have no time to learn the language used.

Today's trend is towards extended use of the UML, not only for software modelling, but also for modelling of systems. Several problems, with this extended use of the UML, have been observed, one of the most important being the language's difficulties to visualize "deep dependency structures". The solution to the problems, which seems to be most popular, is to introduce extensions to the UML. If you consider the understandability problems already present with the UML this may not be a possible road towards a useful system modelling language.

4 Ada 95 developed into a modelling language

When Ada was introduced in Swedish defence it was also used as a vehicle for introduction of software engineering. An Ada-based pseudo language the "Adel" (Ada Design Language) was then developed. When later the need for organized systems engineering was identified, Adel was further developed into "Odel" (Object Design Language[ref 9]). Odel is still Ada-based, but compared to Adel it is more formal and allows for objects, other than software.

The Odel syntax is alphanumeric just like Ada 95 but it is closely connected to two UML diagrams, the Component diagram[ref 1, 2] and the Message Sequence diagram [ref 2].

The result is a modelling language which meets the requirements listed above as follows:

Understandability

Understandability is ensured through a combination of simplified Ada95 syntax (Formalized English) and simplified UML Component diagrams (Object graphs), drawn from the Odel descriptions.

Structure

The structure in the Odel descriptions is based on Ada's "withing principles" developed into dependency structures, including object (types) of categories Mission, Operator, Software and Hardware. After an idea from Håkan Lindegren of Örebro University the "deep dependency structure" is visualized in "Tree graphs".

Behaviour

Behaviour is modelled in a simplified and extended Ada 95 syntax, with the main modifications being:

- Procedures, Functions and Tasks are combined into "Actions" after an idea from professor Vitalis S Kaufman of

Moscow State and Tammerfors Universities.

- Concurrency between concurrently active actions is described through a "concur" statement after an idea from Ingalill Bratteby-Ribbing of the Swedish Defense Material Administration.

Communication

Communication in system modelling is done basically in two ways: through invocation, with parameter passing, and through message passing between concurrent processes. Invocation is no problem with use of Ada's principles for parameter passing. Message passing is different, but a solution was found, with introduction of "messages", based on Ada's principles for management of global variables and with introduction of the reserved words **send** and **receive**.

The Odel language was defined in the style of the Ada 95 Language Reference Manual and primarily through references to the Ada 95 LRM. The Odel Language Definition is freely available to anyone interested. To request it, send a mail to

info@toolforsystems.com

5 An example

As an example consider a system called "Car window control" with the mission to control the windows in an automobile and, besides the mission object, containing objects of categories operator (Driver), Software (Central window control and Local window control) and Hardware (buttons and switches). Note further in the Tree graph:

- Each object has a "little clock", which indicates its development status
- A system is defined as a "Configuration Item" (CI), in accordance with ISO/IEC 12207 standard[ref 6]. The different system-related concepts in the ISO/IEC 15288 standard are all managed as CIs. In figure 2, besides the current CI "Car window control", you can see an attached CI "Door", representing an "enabling system in the wording of the ISO/IEC 15288[ref 7].

If you focus on the mission object "Car window control", it can be expanded into a component diagram. For simplicity reasons arrows are omitted and only two levels are shown.

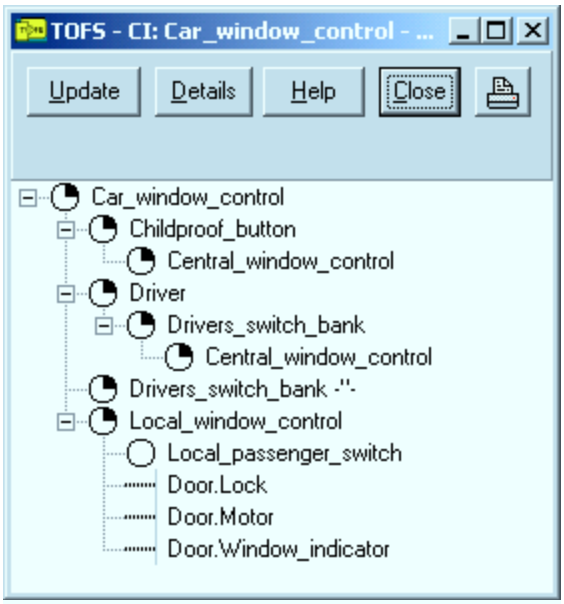


Figure 2 Tree graph for "Car window control"

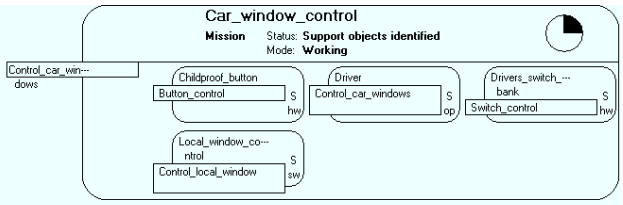


Figure 3 Top component diagram (object graph) for "Car window control"

Note that this diagram also shows the offered and required interface for the Car window control object:

- In the left hand "action box" is shown that the object offers the action "Control car windows"
- Each support object shows action(s) in their "action boxes". Together these constitute the required interface for "Car window control".

If you go into the action description for the action "Control Car windows" you find a "concur" statement, which defines that the actions in the support objects shall be active concurrently:

```
concur
  # Driver.Control_windows
  # Drivers_switch_bank.Switch_control
  # Childproof_button.Button_control
  # Local_window_control.Control_local_window
end concur
```

If you look at the component diagram for the "Local window control" object, you can see how objects from attached CIs are drawn "outside" in the tradition from HOOD (Hierarchical Object Oriented Design) syntax[ref 5].

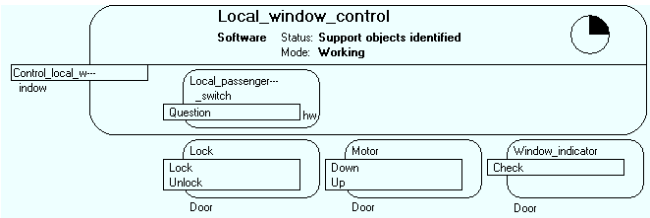


Figure 4 Component diagram with objects from an attached CI

In the example the two "window control" software objects are active concurrently and communicate by way of messages. The Odel fragment below shows how an action for sending such messages can be described. In the other software object a corresponding action will be found with reception of the "up" and "down" messages.

Note that the Odel description is formal enough to be analysable for correct syntax and to be useful to guide a programmer. It may not be directly understood by all stakeholders, but should be understood, after a short introduction, because of its closeness to natural language.

```
action Window_command
(direction_up : in Boolean,
window_concerned : in seat_position) is
visibility: Offered
purpose: { Receive orders to control windows and forward
these by way of the CAN bus }
messages:
up_message,
down_message

variables:

begin
  case direction_up is
    when true  => send up_message(window_concerned)
        {send message to move window upwards to the local
        control software for the concerned window in the car,
        defined by seat position
        (driver, other_front, rear_left or rear_right)}
    when false  =>
    send down_message(window_concerned)
    when others => null
  end case
end
```

6 Conclusions

The principles described above have been applied in several industrial and defence project with the experience that the combination of a few diagram types and Ada 95 based textual descriptions gives a system modelling language, which is both understandable to stakeholders concerned and useful as a formal basis for design of software and hardware modules.

The main advantage, when compared with a completely diagrammatic alternative, such as "extended UML" is the greater simplicity and the resulting better understandability. This is important, since better understanding by stakeholders should decrease the risk of building the wrong software (which is never used).

Another advantage is that some formality is introduced, through the coupling to Ada 95, without introduction of "mathematical syntax", which may introduce new problems to understand the descriptions.

References

1. Booch, Grady, Software Engineering with Ada, Benjamin Cummings 1983
2. Fowler, Martin, UML Distilled, Addison Wesley 1999
3. Ogren Ingmar, On Principles for Model-based Systems Engineering, *Systems Engineering,* Vol 3, No 1, pp. 38-49, 2000
4. Ogren, Ingmar, Possible Tailoring of the UML for Systems Engineering Purposes. *Systems Engineering*, Vol 3, No4, pp. 212-224, 2000
5. Rosen, Jean-Pierre, HOOD An industrial approach for Software Design, HOOD technical group 1997
6. www.12207.com, the website that gives information on the standard for Information Technology –Software Life Cycle processes
7. www.15288.com, the website that gives information on the standard for Systems Engineering –System Life Cycle processes
8. www.omg.org, the web site where you find UML-related document, including definition of the "Component diagram"
9. www.toolforsystems.com, the website which contains a downloadable version of the present release of the Tofs system modeling software, including an analyzer for the Odel language

Ada i Sverige vårmöte och seminarium 2002 i Göteborg

AiS-mötet den 25 april hölls denna gång i Göteborg (andra gången där i AiS historia) med ett 30-tal deltagare.

Seminariet hade två föredragshållare med tre föredrag, Dan Johansson från IFS i Göteborg och Tucker Taft från SofCheck i USA.

Refererat och intryck från seminariet redovisas nedan. Vi tackar föredragshållarna och deras företag för att de mycket tillmötesgående ställt sitt presentationsmaterial till förfogande för Rendezvous för denna redovisning.

Från det efterföljande årsmötet i Ada i Sverige noteras bl a att till styrelseledamöter omvaldes Örjan Leringe och Kristina Lundqvist (f n vid MIT) samt nyvaldes Torbjörn Johnsson.

Ada SOM ETT BÄTTRE C?

Föredragshållaren var Dan Johansson; Senior Technology Architect, IFS Research & Development, Göteborg.

Företaget IFS AB

IFS (Industrial and Financial Systems) AB utvecklar och säljer affärssystem och konsulttjänster i anslutning till dessa, till medelstora och stora företag. Man är ca 3200 anställda i 45 länder, varav ca 600 pers i IFS R&D och ca 2000 i IFS Consulting. Av IFS R&D 600 anställda finns bl a ett hundratal vardera i Linköping och Göteborg och ca 200 i Sri Lanka.

IFS omsättning 2001 var 304 miljoner dollar.

IFS är ett av faktiskt en handfull svenska företag som de senaste åren framgångsrikt slagit sig fram på världsmarkanden för affärssystem. De utgör ett mycket väsentligt tillskott till svensk programvaruverksamhet.

(IFS och konkurrenten Intentia kommer enligt pressmeddelande nyligen att börja samarbeta för att undersöka vad som händer när värdekedjor samverkar. Kanske första steget till ett bredare samarbete.)

Produktutbudet

En uppfattning om bredden av tillämpningar som kan stödjas av IFS moduliserade produktutbud fås av bilden [IFS Applications 2002]. IFS säljer dels ett antal standardsystem eller "paketerar" sina egna och tredje-partsleverantörers moduler och anpassar till resp kunds behov, se bild [IFS Implementation]. Större användare, vilket t ex General Electric och British Aerospace är, kan själva sköta paketering och integration i sina systemkomplex.

Model Driven Design & Implementation

IFS tillämpar modelldriven utveckling och implementering. Som bilden [Model Driven Design & Implementation] visar genererar man programvarukod från process- och UML-modeller till den 3-lagersarkitektur man utnyttjar. IFS är en tidig (1997) användare, om än ej fullt ut, av UML.

IFS modellerar hela den affärsverksamhet som skall stödjas i form av Business Components, vilka består av ett antal programvarukomponenter. Ett 150-tal affärskomponenter kan ingå i ett system. Dessa komponenter samverkar för att åstadkomma olika affärsprocesser och aktiviteter, bild [Business Components].

Prov med användning av Ada

IFS började få behov i sina system av en annan slags databehandling än bara för databasaccesser. En del av den programvaran gjordes i C/C++, med åtföljande bl a portabilitetsproblem. 1999 började man testa Java som ett alternativ, men fann att prestanda var tveksamma. Då kom frågan upp om man kunde använda Ada 95 i stället och få lika portabel kod som med Java, men med prestanda i närheten av de man fick med C. Under 2000 började man prova att jobba med Ada i vissa utvalda Business Logic och Data Storage moduler, se bild [Multi-Tier Architecture].

Förutsättningarna var att man skulle pröva att bygga ramverkskomponenter och en prototyp av en Business Component. Vidare skulle kodmängden vara liten och för endast en plattform. De mest erfarna programmerarna på IFS skulle göra jobbet. Den upplevelse man fick av arbetet med detta var att det verkade rimligt att hantera, att man hade

fått tillgång till världens bästa (?) kompilator alla kategorier (GNAT), men att det var mycket "låg-nivå"-jobb, d v s sådant man normalt (med tidigare använda språk/hjälpmedel) inte hade behövt göra själva.

Användningen av Ada verkade i alla fall så pass intressant att man beslöt att gå vidare under 2001 med en breddad användning där man även kopplade in ett 15-tal för IFS "normala" utvecklare (de flesta ej utbildade programmerare utan med bakgrund från affärsvärlden och vana med Visual Basic), kompletterade med 5 experter. Man skulle utveckla en komplett modul (innehållande ett 30-tal aktiviteter) och testa den på både Unix och Windows.

Nu blev upplevelserna av att använda Ada helt annorlunda.

* Man fann inget bra IDE/debug-verktyg
* Kompetensen hos utvecklarna var för låg; det tog för lång tid för dem att lära sig Ada
* Kompileringstiderna kunde bli mycket långa (uppåt 4 timmar) och erforderligt minne mycket stort (kunde gå upp till 2 Gb)
* Dålig tillgång till klassbibliotek XML/XSL, Http, etc
* Produktiviteten blev låg; en faktor 5-50 ggr mindre affärslogik per dag och utvecklare
* Dyra utvecklingsverktyg (ca 10000 kr/ användare)
* Sist men inte minst, man riskerade mycken "bad-will" på marknaden. D v s det skulle inte gå att kräva att alla konsulter, partners och användarföretag skulle investera i att använda Ada, som de eljest aldrig utnyttjade.

Slutsatser betr Adas användbarhet i IFS verksanhet

Slutsatsen under förra året blev därför att man inte kan begära att alla partners och IFS Consulting skall hålla kompetens och betala licenser för Ada.

Därför beslöts att Ada endast får användas i server-komponenter som aldrig skall anpassas av partner eller konsult, vilket är samma regler som gäller för användning av C.

Under början på detta år har man produktifierat och releasat några av ramverkskomponenterna på Windows och Unix. Upplevelsen var att koden kompilerade direkt på alla OS, men att det ej fanns support för senaste versioner på alla OS.

Svaret på rubrikens fråga, var alltså att ur IFS synpunkt: Ada är ett bättre C, men inte mer.

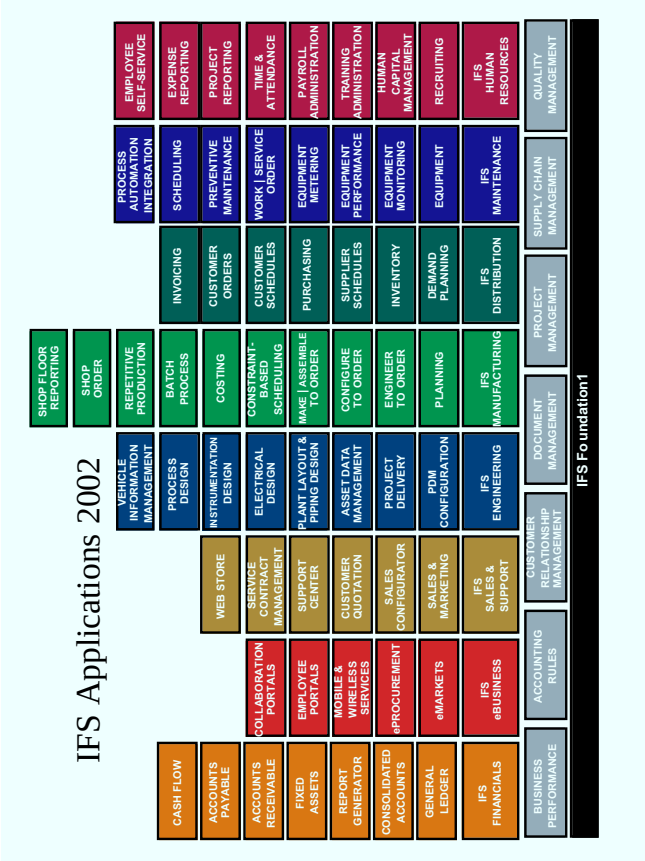
Man får visserligen C-prestanda för rak procedurell kod, men objektorientering och inkapsling kostar även i Ada.

I den tekniska plattformen föredrar man Ada framför C/C++, men plattformssupporten kan ge problem.

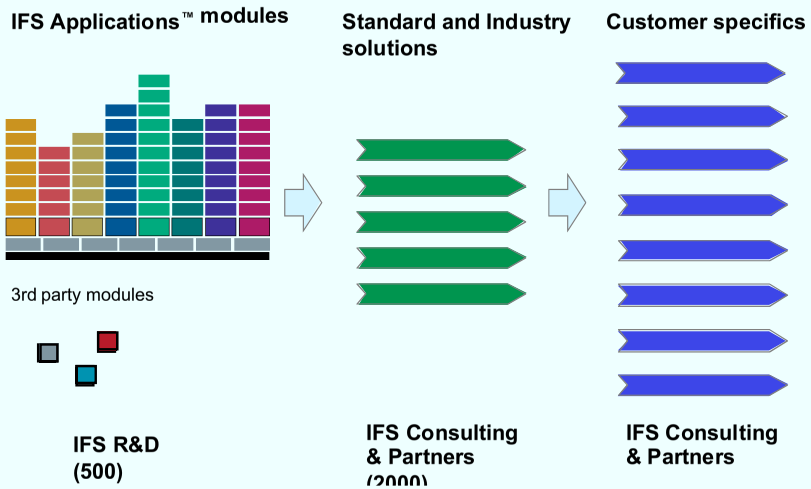
Det som har hänt på markanden under tiden är att Java med HotSpot-tekniken (prestandaoptimerande kompilering m m) börjar att få prestanda som är acceptabla, vilket ytterligare kan minska anledningen att använda Ada i den aktuella klassen av tillämpningar.

Användning av Ada i de specialiserade tillämpningsmodulerna är ej möjlig av flera skäl: För kraftfull och tungrodd utvecklingsmiljö; tillgång till kompetens, standardverktyg och snabb "round-trip" prioriteras högre.

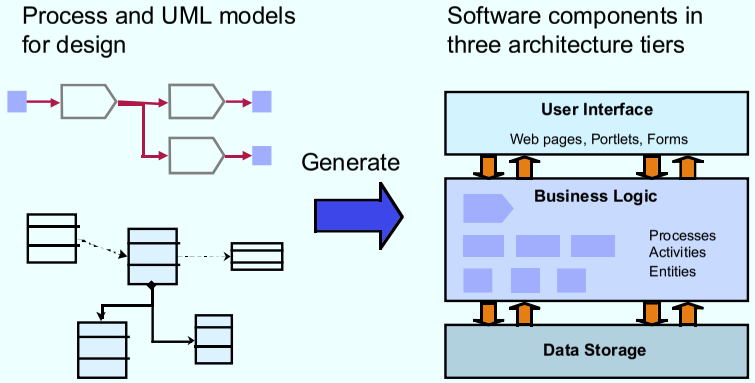
Referentens reflexion: Om Ada 200Y, enl Tafts följande föredrag, funnits redan nu, hade kanske slutsatserna blivit något annorlunda. Men å andra sidan kommer inte övriga världen att stå still fram till dess den versionen kommer fram, så man får nog räkna med problem även då.



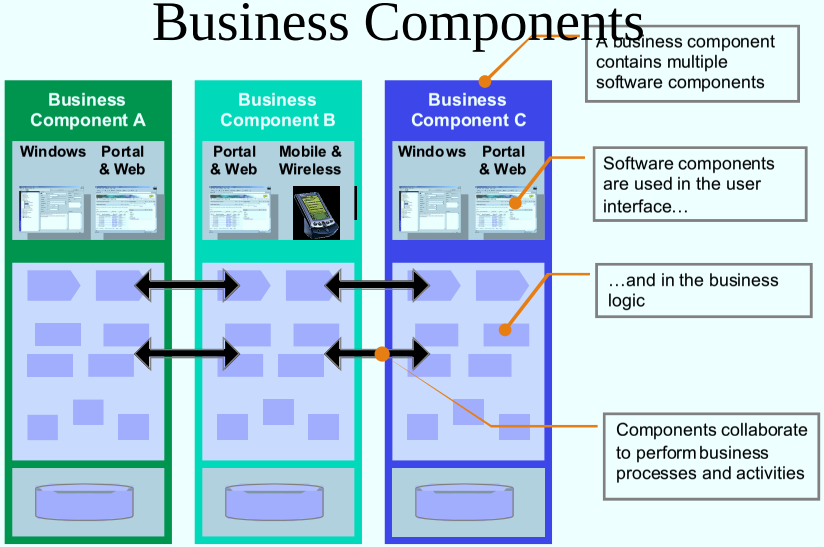
IFS Implementation



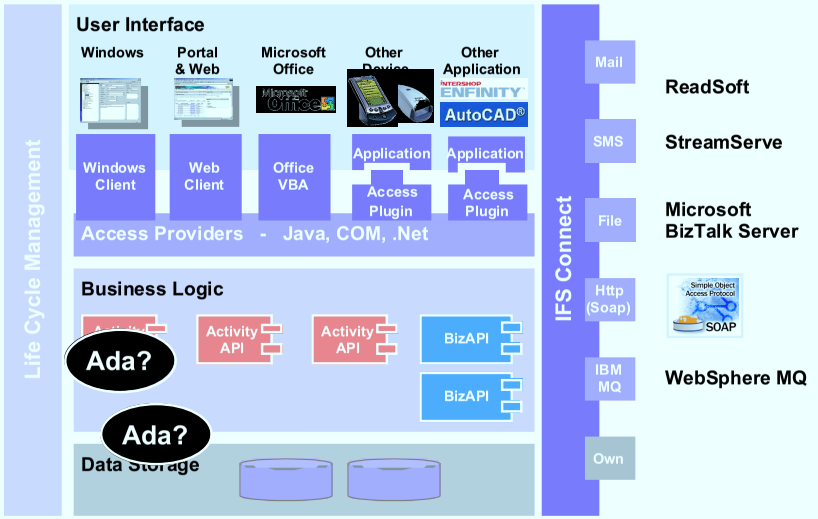
Model Driven Design and Implementation



Business Components



Multi-Tier Architecture



Ada 200Y – VAD OCH VARFÖR

Föredragshållaren Tucker Taft är en av de mest kända personligheterna i Ada-världen, främst för sin ledande roll i utvecklingen av Ada 95. Taft har just startat sitt eget företag SofCheck, Inc., men arbetade tidigare i många år vid AverSoft och dess föregångare SofTech. SofCheck skall syssla med analys och verifiering av programvara.

Nedanstående är huvudsakligen en översättning (ibland säkert av tveksam kvalitet) av de bilder Taft visade.

Taft fick f ö mycket god respons från det till stora delar mycket kunniga auditoriet, när han ville ha synpunkter på värdet av de ändringsförslag han presenterade och ville veta om det fanns andra frågor som borde tas upp.

Ada LEVER OCH UTVECKLAS

Enligt ISO-reglerna skall standarder omprövas och kunna revideras eller avvecklas från tid till annan. För Adas del kommer vi främst ihåg Ada 95, som var en rätt väsentlig "uppgradering" av Ada 87, vilken var den första ISO-godkända versionen av Ada; dessförinnan fanns Ada 83, som var en ANSI-standard.

Ada 83s "mantra" kan sägas ha varit *"Inga subset, Inga superset"*, en rätt så stelbent inställning som säkert kostade Ada en del "marknadsandelar".

För *Ada 95* kan mantrat sägas ha varit *"Portabel kraft till programeraren".*

Adas ursprungliga tillkomst och dess tidiga förbättringar styrdes hårt av DoD. Efter det att DODs "Ada-mandat" upphörde för ca fem år sedan, sker utvecklingen mer på frivillig väg och av användare och leverantörer. De senare verkar främst genom branschorganisationen Ada Resource Alliance, som kan sägas ha övertagit AJPOs (DOD Ada Joint Program Office) roll som "pådrivare" för Ada.

Genom Internet har det främst via Comp.Lang.Ada och Team-Ada (ett löst nätverk av Ada-entusiaster) utvecklats en bred aktiv samverkan mellan användare, leverantörer och "språkadvokater" i en öppern diskussion kring nya idéer och möjliga språkförbättringar.

En annan viktig faktor är tillgången till en "open source" (≈ gratis) kompilator i GNAT, vilken genererat ett intresse på gräsrotsnivå för Ada och också lett till andra "open source"-bidrag till kompilator och bibliotek. Den har också lett till experiment med ny syntax och ny semantik.

ISO WG9 och Ada Rapporteur Group

ISO förvaltar Adas standardisering och evolution genom sin WG9 och Ada Rapporteur Group, vilka innefattar användare, leverantörer och "språkadvokater".

ISO släppte sitt första "officiella" Corrigendum" till Ada 95 standarden i sept 2000. (Se föreg nummer av Rendezvous.) Man fokuserar nu på förbättringar och tillägg till språket.

Så Tucker Tafts fråga var; vart går vi? Han redovisade sin syn på saken enligt följande.

ALLMÄNNA MÅL FÖR SPRÅKETS EVOLUTION

Dessa är dels att vidareutveckla Adas ställning som
- säkert
- högprestanda
- flexibelt
- portabelt
- tillgängligt
- distribuerat, parallellt, realtids-, objektorienterat programspråk

och dels att slutföra arbetet att integrera objektorienterade koncept i Ada.

Säkerheten är det viktigaste med Ada

I dag kan man konstatera att Ada är det främsta språket för säkerhetskritisk programvara. De säkerhetskritiska egenskaperna är samtidigt grundläggande för att göra Ada till ett hög-produktivt språk inom alla dess tillämpningsområden.

Det är viktigt att ändringar eller tillägg i nya versioner av Ada inte öppnar några nya säkerhetshål, utan de bör tiilhandahålla ännu högre säkerhet, ge fler tillfällen att fånga misstag vid kompileringen.

IDÉER FÖR NY VERSION AV Ada

Taft fortsatte med att ge exempel på ändringar och kompletteringar som f n diskuteras inför en nästa revidering av Ada-standarden till Ada 200Y. (Man siktar på Y=5.) Han varnade för att allt som behandlas nedan inte skulle återfinnas i det slutliga förslaget.

Det är nödvändigt att användare ser till att bli inblandade i detta arbete för att ange prioriteringar och hjälpa till att förfina förslagen.

ISO WG9/ARG, som ansvarar för att få fram revideringsförslaget, kommer att under året publicera mer om hur arbetet skall gå till.

Möjliga säkerhetsändringar genom nya pragma

Pragma (direktiv till kompilatorn) är ett naturligt sätt att lägga till säkerhetskontroller. Den enda effekten av en ytterligare säkerhetskontroll är att avvisa ett för övrigt korrekt program. Det har ingen inverkan på semantiken hos program som klarar kontrollen.

Pragma kan assocoeras med en enstaka deklaration, en punkt i exekveringsföljden, en deklarativ region, en källfil eller ett helt bibliotek (configuration pragma).

Följande nya pragma skulle kunna vara aktuella:

- Pragma för att förhindra oavsiktlig *override* (ersättning, överskuggning) eller inte *overriding* av primitiva operationer.

  Detta skulle fånga stavfel, missanpassningar i parameterprofiler, osäkerhet vid underhållsåtgärder.

- Standardiserade Assert ("säkerställ") pragma

  och även andra Precondition/Postcondition/Invariant pragma som hör ihop med subprogram, paket eller typer.

- Pragma/Attribut för att specificera fysikaliska enheter som hör ihop med särskilda subtyper.

  Detta skulle fånga inkonsistenser i komplexa uträkningar.

- Pragma Configuration skall kräva initiering av lokala variabler i alla grenar (paths) innan användning.

  För att matcha krav på bytekodverifierare för JVM; att fånga en vanlig orsak till fel.

Att hantera dagens verklighet

Dagens verklighet på programvaruområdet karaktäriseras bl a av

Cykliskt beroende mellan typer är normen i komplexa objektorienterade system

Begreppet "Interface" kan ha flera implementationer (CORBA; Java, C#, COM).

Ökningen i betydelsen av Java Virtual Machine och .Net gemensamma run-time

Allt komplexare APIer; *API-krigen*

Komponentbaserade system

Flerspråkiga system

Dynamiskt bundna system

Förändringar och tillägg i Ada kan hjälpa till att hantera denna verklighet

Att förbättra interoperabiliteten med dagens verklighet

- Stöd cykliskt beroende mellan typer i olika paket.

  Olika alternativ övervägs här. Ansatsen f n är "type T is [tagged] separate in P". Detta är också relaterat till ett förslag om anonyma accesstyper.

- Stöd begreppt "Interface" som det används i Java, CORBA, C# etc.

  Stöds redan på något sätt av Ada till JVM-kompilatorer

  T ex Pragma Convention(Java_Interface, T). Därtill av några magiska kompilatortillhandahållna kroppar för primitiver som anropar likalydande op av "encloser". Det finns också ett förslag till "abstract interface" typer.

Portabilitetsförbättringar

Ada ger utmärkt stöd för att bygga portabel kod. Ada-biblioteket är dock fortfarande relativt begränsat; tillägg för att definiera ytterligare standardbibliotek kan förbättra portabiliteten.

Fokus bör i första hand vara på att säkerställa portabilitet för Ada på server-sidan, t ex:

Filer och biblotek
Sockets
HTTP/CGI Servlet interface
Tidszoner
Omgivningsvariabler
ODBC/JDBC ekvivalenter

Dessa bör vara baserade på POSIX eller Win32, men förenklade och gjorda OS-oberoende

Förbättra tillgängligheten till Ada, d v s åtgärder som underlättar övergång till Ada

Inget mandat ovanifrån längre
Det finns inte längre något mandat ovanifrån (i USA DoD) om att man måste använda Ada. Ada måste därför kunna infiltrera från botten eller sidan i en organisation.

Man behöver titta på paradigmer och ramverk som ökar i popularitet, som

JVM, J2EE, EJB
Microsoft COM och .Net
XML/XSL
ODBC/JDBC
HTTP/Servlet

UML-baserad modellering
Likaså ökar UML-baserad modellering i popularitet. Det måste därför göras lätt att gå mellan UML och Ada.

Det måste bli en fullständig integration av objekt-orienterade koncept i Ada.

Vilken väg vill vi gå

Vi bör lära av utvecklingen av nya språk och andra programmeringsparadigmer

Det fanns ingen bra modell för multipelt arv under Ada9X-processen, men numera har multipelt arv av interface kommit fram som en bra kompromiss

UML etablerar OO under design såväl som kodning

Anvädbara parallella och distribuerade OO-modeller börjar komma fram

Bortse ej från marknads- och övergångsfrågor

Så t ex kan ObjectOperation(....) syntax hjälpa till att bevara OO-vyn

Vi bör komma ihåg våra "core values

Säkerhet, Höga prestanda, Portabilitet

SLUTORD

Taft påpekade att ARG är ute efter att få in förslag betr bl a

* Paket värda att standardisera
* Pragma/Attribut värda att standardisera
* Att eliminera onödiga implementeringsberoenden och olämpliga "erroneous" eller "bounded error" situationer, etc

Förslag/kommentarer bör skickas till Ada-Comment@ada-auth.org

Ordna gärna en AiS/Ada Europe brainstorming.

Tala ut nu!

* * *

FIXING SOFTWARE BEFORE IT BREAKS

I sitt andra föredrag behandlade Taft problemställningar som verkade ligga honom nära i verksamheten i hans nya företag, nämligen betydelsen av statisk analys i säkerhetskritiska tillämpningar.

Varför statisk analys

Taft hävdade att statisk analys bl a behövs därför att

komplett täckningstest är mycket dyrbart

om man använder run-time exceptions blir täckningstester så mycket svårare

tasking skapar ohanterligt många olika fall att testa

generik kan endast testas vid run-time för vissa speciella instantieringar - allmän testning vid exekvering är inte definierad

Jämförelse av statisk felupptäckt i Ada och C-baserade språk

Taft visade en lista med jämförelse mellan nuvarande Ada-version och C-baserade språk enligt nedan.

Test av arrayer
Ada: Indextypen måste vara korrekt
C/C++ och Java: vilken integertyp som helst får indexera en array. Java kollar indexgränser vid exekvering.

Uppräkningstyper
Ada: Ingen implicit omvandling till/från en uppräkningstyp
C: Fritt omvandlingsbar med vilken integertyp som helst
C++: Fritt omvandlingsbara till integer, men inte tillbaka
Java: Har inga uppräkningstyper utan måste använda integer med namngivna konstanter

Generiska mallar
A*da: Testar vid kompilering att generiska kroppar använder parametrar korrekt*
C++: Ingen test av generiska mallar innan de instantierats

Synkroniserade/Skyddade Operationer
Ada: Alla operationer med tillgång till skyddade datakomponenter är låsta under anrop
Java: Där kan man alltid lägga till operationer som inte låser, antingen i samma klass eller i en underklass

Läsning av icke-initierade variabler
Ada: Bounded error till referens
Java: Förbjudna vid kompilering

Hantering av exceptionella händelser
Ada: Icke-hanterade exceptionella händelser fortplantas
Java: Icke-hanterade exceptionella händelser förbjuds vid kompilering, om de inte inte finns i en "throws clause" (eller i en "unchecked" exception)

Hur långt kan man komma med statisk analys

Påstående
Alla situationer som kan orsaka fel i språkdefinierade tester, resultera i "bounded error" eller felaktig exekvering, kan elimineras vid kompilering.

Farhåga
Hur konservativa, belastande eller restriktiva skulle kompileringstesterna bli.
SPARK-Ada95 är ett exempel på ett rätt så restriktivt subset, som ändå utvecklats till ett högst användbart system.

Tro
Noggrann design av (annotations) språket kan ge ett mycket flexibelt och användbart språk, vilket icke har några språkdefinierade feluppträdanden vid exekvering
Många högre nivå eller tillämpningsspecifika feltillstånd eller inkonsistenser kan också elimineras vid kompilering genom användning av ett "programmerbart" annotationsspråk. Test av fysikaliska enheter är ett exempel.

Behov av ytterligare studier

Taft såg behov av studier i vilka de typiska orsakerna är till att människor gör fel.

Han gav några exempel på typiska misstag:

Utelämnandesynden
Ointierade variabler
Utelämnat alternativ i switch/case uttryck
Utelämnat tecken (t ex "break;". "=", "&")
Utelämnat return uttryck
Utelämnat "inkrement" steg i en loop

Förvirringssynden
Byta parametrarna till"strcpy"
Förväxla 1=succé med 0=succé
Att glömma implicita effekter eller företrädesregler

Närsynthetssynden (också känd som lathet)
Fast bufferstorlek (det kända "gets" hålet i "sendmail")
Fast tabellstorlek (förtretliga "för många revideringar" i ändringsstyrningssystemet)

Ett (från människans synpunkt) väl utformat språk kan
minimera förekomsten av dessa misstag, eller ...
fånga dessa misstag vid kompilering

Att skylla på arbetaren för C´s dåliga utformning

Som stöd för sina teser hänvisade Taft till "A software Fault Prevention Approach in Coding and Root Cause Analysis" (Weider D. Yu Bell Labs Technical Journal, April-June 1998).

Enligt denna var de tre största underliggande orsakerna till fel vid programvaruutveckling:

exekvering/förbiseende (38 %)

> otillräcklig uppmärksamhet på detaljer (75 %) och otillräckligt hänsynstagande till alla relevanta aspekter (11%)

resurser/planering (19%)

> otillräcklig ingenjörstid (76%) och otillräckligt internt stöd (4%)

utbildning och träning (15%)

> inom det tekniska ansvarsområdet (68%) och betr användning av programmeringsspråk (15%)

Lucents Anvisningar för C-programmerare på 5ESS framgår av bilden.

Lucent 5ESS C Programmer Advisories

- Initialize all variables before use
- Control flow of break and continue statements
- Check C operator associativity and precedence for correct usage
- Ensure loop boundaries are correct
- Do not over-index arrays
- Ensure value of variables is not truncated
- Reference pointer variables correctly
- Check pointer increments/decrements
- Ensure logical OR and AND tests are correct

Konsten att fånga fel tidigt handlar enligt Taft om

> Redundans
>
> Avgränsningar (Compartmentalization)
>
> Att kräva fullständighet
>
> Att kräva tydlighet (Explicitness)

Möjliga nya (annotations) språkegenskaper som kan förbättra statisk felupptäckt

Taft nämnde förslag till åtgärder inom ett antal områden där statisk analys för Ada kan förbättras.

Vissa av dessa kan komma med i Ada 200Y, i vissa fall som exekveringstester.

Andra kommer att åstadkommas av separata verktyg eller förbättrade kompilatorer.

Han utvecklade exempel inom följande områden:

Ointierade variabler
Noll-tester
Felaktig Concurrent access
Access-före elaborering
Test av fysikaliska enheter
Region-baserad minneshantering
Programmerbar statisk analys

Tafts slutsatser/påståenden

- Att bevisa korrekthet är mycket svårt
- Att bevisa att något är rimligt, konsistent etc är möjligt vid kompilering
- Att flytta alla Adas nuvarande exception, bounded-error och feltillstånd till kompileringstester är möjligt utan att skapa överdrivna begränsningar
- Att lägga till programmerbar statisk analys kan tillåta att man vid kompilering testar nya former av "rimlighet" (t ex test av fysikaliska enheter, hantering av säkra regioner, tillämpningsspecifika tester etc)

Ingemar Carlsson,
som hade bevistat AiS-seminariet i Göteborg

"Allt" om en allt starkare kraft i programvaruvärlden

Open Sources : Voices from the Open Source Revolution

Denna intressanta exposé över open_source-rörelsen, som utgavs i bokform av O'REILLY 1999, finns nu att läsa på nätet hos http://www.oreilly.com/catalog/opensources/book. Det går också att ladda ner den i HTML kapitelvis.

Där finns en mängd information om olika företeelser inom denna rörelse, författade av många kända namn. Det har så klart hänt en hel del kring Open Source de tre åren sedan boken skrevs, men det mesta i den är tidlöst.

Som framgår av innehållsförteckningen, finns det mycket intressant att hämta här. I förbigående kan man dessutom läsa om hur historien om DNA och Nobelpriset har en koppling till Open Source.

Innehåll:

Introduction
Chris DiBona, Sam Ockman, and Mark Stone

A Brief History of Hackerdom
Eric S. Raymond

Twenty Years of Berkeley Unix: From AT&T-Owned to Freely Redistributable
Marshall Kirk McKusick

The Internet Engineering Task Force
Scott Bradner

The GNU Operating System and the Free Software Movement
Richard Stallman

Future of Cygnus Solutions: An Entrepreneur's Account
Michael Tiemann

Software Engineering
Paul Vixie

The Linux Edge
Linus Torvalds

Giving It Away: How Red Hat Software Stumbled Across a New Economic Model and Helped Improve an Industry
Robert Young

Diligence, Patience, and Humility
Larry Wall

Open Source as a Business Strategy
Brian Behlendorf

The Open Source Definition
Bruce Perens

Hardware, Software, and Infoware
Tim O'Reilly

Freeing the Source: The Story of Mozilla
Jim Hamerly and Tom Paquin with Susan Walton

The Revenge of the Hackers
Eric S. Raymond

Appendix A: The Tanenbaum-Torvalds Debate

Appendix B: The Open Source Definition, Version 1.0

Jada-effekten

Det pågår en mycket intensiv utveckling och diskussion kring utvidgningar av Java™ för realtidsändamål. En hel del av de ändringar som görs i Java hämtas från Ada-världen. För SIG-Ada 2001 i oktober, hade Ben Brosgol från Ada Core Technologies och Brian Dobbing från Praxis (B&D i forts.), skrivit ett intressant föredrag (som dock ej kunde framföras p g a 9/11) med titeln ”Real Time Convergence of Ada and Java™”, som behandlade detta. Vi kan också se av referatet från Tucker Tafts föredrag om Ada 200Y vid AiS-seminariet nyligen (se separat artikel), att en hel del av de ändringar som diskuteras i det sammanhanget inspirerats av lösningar i Java eller av en önskan att uppnå bättre samfunktion mellan Ada och Java. Det sker alltså en korsbefruktning mellan de två språken. B&D kallar detta fenomen för *”Jada-effekten”*.

Vi ser också av Bo Frisbergs artikel i detta nr att t o m gamla Ada 83 fortfarande håller och har sin givna plats för många försvarssystemkritiska tillämpningar.

Samverkan ej konkurrens är Adas framtid

B&D formulerar i sitt föredrag varför de anser att Adas framtid ligger i en sömlös samverkan med Java-miljön, snarare än i att försöka konkurrera med Java.

De konstaterar att Javas exekveringsmiljö varit mest framgångsrik i programvarudelen av den revolution på kommunikationsområdet (e-post, WWW, mobiltelefoner, e-banker etc) som ägt rum under 90-talet och då speciellt via Internet. Den underliggande potential som ligger i Javas bytekoder och Java Virtual Machine vilka kan ge den eftersträvade ”write-once-run-anywhere”-förmågan, tillsammans med mängden av praktiska och portabla API-er har gett Java en stark ställning på dessa nya marknader. De Java-tillämpningar som hittills funnits har dock inte haft så svåra storleks- och prestandakrav.

B&D ser framför sig en triljon kommunikationsapparater år 2025, vilka kommer att påverka alla delar av människors liv då. Bland alla dessa ”devices” kommer det att finnas en hel del med inte bara prestanda- och storlekskrav på programvaran, utan även med krav på hög tillgänglighet, hög integritet och på att klara hård realtid. En ökande andel kan t o m vara säkerhetskritiska.

Det pågår ett mycket omfattande arbete i Java-kretsar att förbereda Java-teknologin för de realtidskrav och de begränsningar som inbäddade tillämpningar ställer inför den nya revolutionen. Det gäller själva språket, JVM:erna och API:erna.

Ada-förespråkare kan hävda att Ada95 redan klarar alla de hårda kraven hos inbäddade system bättre än något annat språk och att Adas egenskaper i högintegritets- och säkerhetskritiska sammanhang inte står något annat språk efter, Ett faktum är dock att Ada inte figurerade i 90-talets kommunikationsrevolution och att det inte startade det nya årtusendet med en växande anhängarskara.

”Slagfältet” närmar sig Adas styrkepositioner

B&D undrar om Ada trots sina många goda egenskaper kommer att kunna hitta en roll i den nya revolutionen när ”slagfältet” flyttas till Adas traditionella styrkepositioner. De är själva rätt så säkra på att Adas framtid ligger i en ”sömlös” samverkan med Java-miljön, i stället för att konkurrera med den.

Man kan se hur de båda språkmiljöerna börjar att till viss del konvergera. Som exempel anger han att många av de nya idéerna för ”realtids-Java”, som de som behövs för predikterbarhet och determinism, lånats från Ada. Redan ”grund-Java” hade tagit Adas modell för exceptionhantering och nu kan man se att realtidsutvidgningarna har motsvarigheter för protected objects (incl entries), priority ceiling emulation, väldefinierade scheduleringsregler, absolute delay, high precision timers, suspension objects, dynamisk prioritetsändring, interrupthanterare, asynchonous transfer of control, tillgång till det fysiska minnet, abort-deferred regions etc.

På motsvarande sätt evolverar Ada mot Java. Redan Ada 95 förde in stöd för en omfattande modell för objekt-orienterad programmering, med hierarkier för enkelt arv, constructors och finalizers, vilka inte skiljer sig så mycket från Javas. I Ada 200Y kan vi mycket väl få se tillägg för interface enligt Java-modell, som skulle ge samma begränsade form av multipelt arv som i Java (från en klass till godtyckligt antal interface). Vidare kanske Ada 200Y kommer att lätta på reglerna som nu hindrar ömsesidigt beroende paketspecifikationer, med en ny with type konstruktion. Detta skulle tillåta ömsesidigt beroende Java-klasser att modelleras som Ada-paket vilka vart och ett definierar en tagged typ med sina primitiva operationer,

utan att behöva tillgripa klumpiga ”work-arounds” för att undvika cirkularitet i with-beroendena. Slutligen säger B&D, är det t o m en del diskussion om att tillåta en Java-lik OOP-syntax för att anropa de primitiva operationerna i en tagged typ.

Detta skulle kunna användas i stället för den traditionella proceduranropsstilen, som fordrar regler för att identifiera vilken parameter som är det objekt som styr den dynamiska bindningen, och i stället ersätta den med en OOP-stil enligt principen *objectOperation(parameters).*

B&D drar den slutsatsen att både Ada och Java är föremål för *Jada-effekten.*

Även exekveringsmiljöerna måste kunna samexistera

De framhåller också att det inte räcker med att språken konvergerar. Det är också mycket viktigt att exekveringsmiljöerna gör det, om de två teknologierna skall samexistera framgångsrikt. Där har redan en del gjorts med att integrera Javas och Adas exekveringsmiljöer.

Han nämner som exempel Aonix *AdaJNI,* som tillåter Ada-program att samverka med Java-klasser och APIer som exekveras av en lokal eller annorstädes belägen JVM via interfacepaket enligt Ada-stil.

Vidare ACTs *JGNAT,* som kompilerar Ada95 till Java bytekod i standard klassfiler, vilket tillåter JVM-baserade program att innehålla en blandning av Ada-kod och Java-kod. Även här finns möjligheten för Ada-koden att ha tillgång till Java-klasser och APIer via interfacepaket enligt Ada-stil.

Det finns också företag som tillhandahåller *Ada ORB*er som ger tillgång till CORBA-objekt från Ada-program, vilket gör det möjligt för logiskt distribuerade system i blandade språk (inkl Ada och Java) att kommunicera via CORBAs client/server modell.

Kompatibilitet med Java ger Ada nytt liv som Lady Jada

B&D slutar med att säga att om Ada skall få fotfäste i den nya generationens kommunikationssystem så måste man bygga på den grund som skisserats. Alla inom Ada-världen måste fokusera på att utveckla och evolvera Ada på sätt som är kompatibla med de nya realtids-Java exekveringsmiljöerna, deras JVMer och APIer. Om vi kan göra det kan Ada få ett helt nytt liv och vi kan t o m vilja döpa om henne till *Lady Jada*.

En observation man kan göra, är att B&D inte nämner något om Microsoft. Åtminstone indirekt kan man kanske tänka sig att Microsofts ”java-ersättare” C# samt .Net, kan få en inverkan på Javas framtid. Men det kanske är en annan historia.

Brosgols och hans medförfattare Dobbings föredrag, som också utförligt behandlar tekniska och andra aspekter kring arbetet med Javas Real-Time Specification som inte refererats här, finns i ACMs Ada Letters, December 2001, samt på den CDROM som ACM gett ut från konferensen.

I Carlsson hade läst Ada Letters och kommenterat.

SESAM 2001 CD

Det finns fortfarande en hel del ex kvar av denna mycket innehållsrika CD (version 2) om Arkitektur och Systembygge, som togs fram i anslutning till höstseminariet. SESAM-intressenter som inte ännu tagit tillfället att sprida denna CD till sina system och programvarumänniskor, liksom enskilda medlemmar kan kontakta sekretariatet för att få leverans av lämpligt antal ex.

SESAM i samarbete med FMV inbjuder till

EXTRA ARKITEKTUR-seminarium den 28 augusti i Stockholm

Architecture Framework Evolution in the United States and Sweden

Direkt från källorna! Fånga tillfället att bli uppdaterad om det senaste betr. arkitekturramverk i amerikanska försvaret och andra federala myndigheter. Och hur ser det svenska försvarets arkitekturansatser ut i detta perspektiv. Notera att interoperabilitet och samfunktion vid svenska internationella insatser prioriteras av statsmakterna.

Den första föredragshållaren är Paula K. Sowell från the Mitre Corp., som är djupt engagerad i det amerikanska arkitekturarbetet. En artikel av Sowell "The C4ISR Architecture Framework: History, Status and Plans for Evolution" fanns med på CDn SESAM 2001.

Ms Sowell kommer att beskriva några olika arkitekturramverk som används inom de amerikanska federala myndigheterna, med tyngdpunkt på den ökande gemenskapen mellan en delmängd av dessa ramverk. Denna gemenskap framväxer genom att ett antal av arkitekturmodellerna i DoD Architecture Framework (tidigare C4ISR Architecture Framework) inlemmas i de andra ramverken. Exempel ges på hur dessa modeller anpassas för användning i sammanhang utanför försvaret. Två av dessa ramverk, DoD Architecture Framework och Federal Enterprise Architecture Framework är f n föremål för revidering och utveckling och planerna för denna evolution beskrivs.

Ms Sowells presentationer följs av en genomgång av Johan Bendz FMV om läget betr arbetet med Försvarsmaktens Arkitekturramverk (FM A), som till del inspirerats av de amerikanska arbetena.

Slutligen planeras en frågestund med aktuella frågor kring utformning och verifiering av arkitekturramverk samt om institutionella mekanismer som måste till för att en långsiktig hållbar uppbyggnad av försvarssystemen i enlighet med ramverket skall bli möjlig.

Idag gäller mycket starkt prioriterade krav på interoperabilitet och samverkansförmåga mellan svenska och andra länders stridskrafter vid insatser inom ramen för våra åtaganden inom EU och NATO Partnership for Peace. I ljuset av dessa blir givetvis jämförelser mellan svenska och, i det här fallet, amerikanska ansatser på arkitekturområdet särskilt relevanta.

Seminariet hålls i sin helhet på engelska

Boka in en intressant dag på Garnisonen!

Kalender

2002-06-12	SESAM VU
2002-08-30	SESAM/FMV Extra-seminarium ”Architecture Framework Evolution in the United States and Sweden”
2002-10-03	SESAM VU
2002-10-22	SESAM Tutorial kurs
2002-10-23	SESAM Höstseminarium ”Metoder och verktyg för modellbygge i produktframtagningen”
2002-10-24	SESAM redovisn + Rådsmöte
2002-11-28	SESAM VU

Ta gärna en titt på vår websajt

http://sesam.tranet.fmv.se

där finns konferenser för hela året uppräknade

Från rådsmötet 25 april

SESAM startar intressegrupp för programvarusäkerhet

Rådsmötet beslöt efter förslag från VU att inrätta en intressegrupp för programvarusäkerhet. Gruppen skulle bl a fortsätta den samverkan som sker inom FoTA P12, som är ett samlingsprojekt som stödjer och bevakar säkerhetsaspekter i programvaruprojekten inom FoTA-programmet. Verksamheten har skapat ett nätverk av säkerhetsintresserade programvarutekniker med bl a en egen hemsida. Intressegruppen skall också bygga vidare på det intresse som visats av flera SESAM-medlemmar, som har följt FMVs framtagning av ”Handbok för programvara i säkerhetskritiska tillämpningar – H ProgSäk” genom att deltaga i granskningar och seminarier. Intressegruppen avses bl a kunna tillvarata och bygga vidare på resultat och erfarenheter från dessa verksamheter, t ex genom informationsspridning inom området programvarusäkerhet via SESAMs hemsida och utbildning med gemensamma insatser från FMV och industri. Rådet beslöt tillfråga Inga-Lill Bratteby-Ribbing om hon skulle vilja vara sammanhållande för gruppen.

Ag ”Arkitektur för programvaruintensiva försvarssystem”

Det inom VU framtagna förslaget att bilda en särskild arbetsgrupp för övergripande arkitekturfrågor avvisades av Rådet med motiveringen att arkitekturfrågor på produktnivå behandlas tills vidare inom SESAM nuvarande grupper och omfattar arkitektur relaterat till SESAM kärnområde. Arkitekturfrågor på högre nivå är ej något för SESAM.

Informationssäkerhet föreslås som potentiell intressegrupp

Vid rådsmötet föreslogs vidare att Informationssäkerhet skulle kunna vara ett intressant ämne för en ny intressegrupp.

Rådet skall till höstens rådsmöte undersöka om SESAM kan bemanna dessa grupper.

Anm. Intressegrupp är ett nytt begrepp i SESAM-sammanhang. Det är ej närmare definierat ännu, men har diskuterats som en lösare form än en arbetsgrupp och med kanske friare och mer nätverksbaserade verksamhetsformer.

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