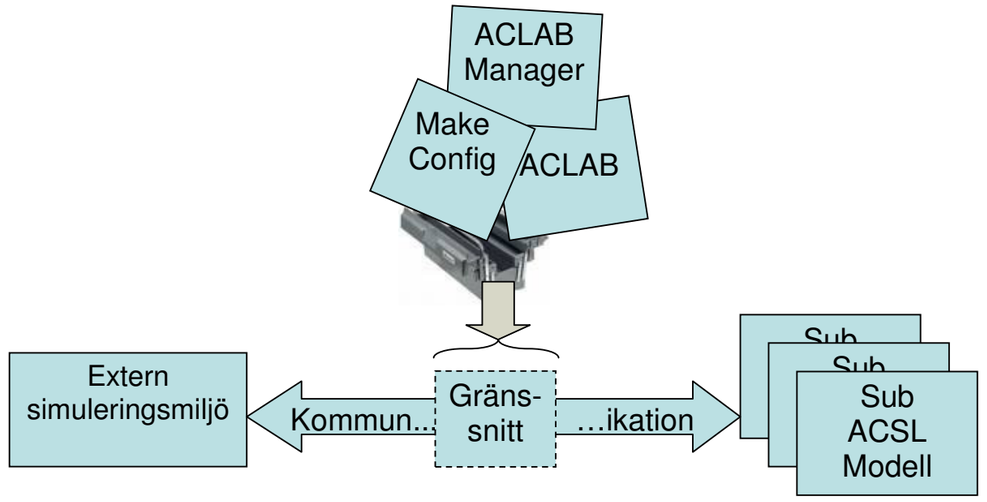
ACLAB Framework 3.0
Ett programmerbart gränssnitt för kommunikation med ACSL-modeller

TORBJÖRN ANDERSSON, MATTIAS VERONA



FOI är en huvudsakligen uppdragsfinansierad myndighet under Försvarsdepartementet. Kärnverksamheten är forskning, metod- och teknikutveckling till nytta för försvar och säkerhet. Organisationen har cirka 1250 anställda varav ungefär 900 är forskare. Detta gör organisationen till Sveriges största forskningsinstitut. FOI ger kunderna tillgång till ledande expertis inom ett stort antal tillämpningsområden såsom säkerhetspolitiska studier och analyser inom försvar och säkerhet, bedömning av olika typer av hot, system för ledning och hantering av kriser, skydd mot och hantering av farliga ämnen, IT-säkerhet och nya sensorers möjligheter.

FOI
Totalförsvarets forskningsinstitut
Ledningssystem
Box 1165
581 11 Linköping

Tel: 013-37 80 00
Fax: 013-37 81 00

www.foi.se

FOI-R--2289--SE Metodrapport **Ledningssystem**
ISSN 1650-1942 Juni 2007

Torbjörn Andersson, Mattias Verona

ACLAB Framework 3.0

Ett programmerbart gränssnitt för kommunikation med ACSL-modeller

FOI-R--2289 --SE
ISSN 1650-1942

Metodrapport
Juni 2007

Ledningssystem

Utgivare FOI - Totalförsvarets forskningsinstitut Ledningssystem Box 1165 581 11 Linköping	**Rapportnummer, ISRN** FOI-R--2289--SE	**Klassificering** Metodrapport
	Forskningsområde 6. Telekrig och vilseledning	
	Månad, år Juni 2007	**Projektnummer** E7121
	Delområde 61 Telekrigföring med EM-vapen och skydd	
	Delområde 2	
Författare/redaktör Torbjörn Andersson Mattias Verona	**Projektledare** Mikael Hansson	
	Godkänd av Mikael Sjöman	
	Uppdragsgivare/kundbeteckning FM	
	Tekniskt och/eller vetenskapligt ansvarig Mikael Hansson	

Rapportens titel

ACLAB Framework 3.0
Ett programmerbart gränssnitt för kommunikation med ACSL-modeller

Sammanfattning

I FOT-projektet *Teknisk hotsystemanalys, THSA*, används simuleringsmiljön ACSL/GM för att utveckla simuleringsmodeller av hotrobotsystem och mål, där det senare vanligtvis representerar någon av Försvarsmaktens plattformar. Modellerna är ofta på detaljnivå. Det är därför önskvärt att en simuleringsmodell består av maximalt en robot och ett mål eftersom stora modeller leder till en rad oönskade effekter. Bland dessa kan nämnas lång kompileringstid, lång exekveringstid samt risk för att uppnå storleksbegränsningar i ACSL.

ACLAB FrameWork är framtaget för att kunna koppla upp en eller flera ACSL-modeller mot en extern simuleringsmiljö. Kravet på denna miljö är att den ska kunna länka dll-filer under Windows. Testade miljöer är MATLAB/Simulink, ett C++ program samt ACSL/GM självt. ACLAB FrameWork öppnar dels upp möjligheten att samköra ACSL-modeller inom THSA, dels att de inom projektet utvecklade modellerna kan komma till nytta i andra projekt inom FOI eller i samarbete med extern kund.

Nyckelord

ACLAB, ACSL, Modellering och simulering

Övriga bibliografiska uppgifter	**Språk** Svenska
ISSN 1650-1942	**Antal sidor:** 37 s.
Distribution enligt missiv	**Pris: Enligt prislista**

Issuing organization
FOI – Swedish Defence Research Agency
Command and Control Systems
P.O. Box 1165
SE-581 11 Linköping

Report number, ISRN
FOI-R--2289--SE

Report type
Methodology report

Programme Areas
6. Electronic Warfare

Month year
June 2007

Project no.
E7121

Subcategories
61 Electronic Warfare including Electromagnetic Weapons and Protection

Subcategories 2

Author/s (editor/s)
Torbjörn Andersson
Mattias Verona

Project manager
Mikael Hansson

Approved by
Mikael Sjöman

Sponsoring agency
Swedish Armed Forces

Scientifically and technically responsible
Mikael Hansson

Report title (In translation)
ACLAB Framework 3.0
A programmable interface for communication with ACSL models

Abstract

Within the project Technical Threat System Analysis, THSA, at FOI the simulation environment ACSL/GM is used for developing simulation models of Threat Systems and Targets. Since these models most often are highly detailed there are wishes not to comprise more than one threat system and one target in a single model. Larger models imply longer compilation time, longer execution time and might reach limitations in the ACSL language due to large models.

ACLAB Framework is developed in purpose to connect ACSL-models to external simulation environments. Those could be any programming environment able to link dll's on Windows. Tested environments are MATLAB/Simulink, Visual Studio (C++) and ACSL/GM itself.
ACLAB Framework makes it possible to link ACSL models to different simulation models. This implies not only extended possibilities to run very large ACSL-simulations withinTHSA, but also possibilities to use the ACSL models in external projects within FOI or by external customer or collaborator.

Keywords
ACLAB, ACSL, Modelling and Simulations

Further bibliographic information

Language Swedish

ISSN 1650-1942

Pages 37 p.

Price acc. to pricelist

Innehållsförteckning

1 Inledning

I FOT-projektet *Teknisk hotsystemanalys, THSA*, pågår kontinuerligt utveckling av simuleringsmodeller av hotrobotsystem och mål, där det senare vanligtvis representerar någon av Försvarsmaktens plattformar. Sedan mitten av 1990-talet används modellerings och simuleringsmiljön ACSL/GM [ACSL] för detta ändamål. Modellerna är ofta på detaljnivå. Det är därför önskvärt att en simuleringsmodell består av maximalt en robot och ett mål eftersom stora modeller leder till en rad oönskade effekter. Bland dessa kan nämnas lång kompileringstid, lång exekveringstid samt risk för att uppnå storleksbegränsningar i ACSL (ACSL tillåter bara ett visst antal diskreta sektioner i en och samma modell).

ACLAB FrameWork är framtaget för att kunna koppla upp en eller flera ACSL-modeller mot en extern simuleringsmiljö. Den externa miljön kan vara ett annat simuleringsspråk, t.ex. MATLAB/Simulink eller C++, men kan även vara en annan ACSL-modell. ACLAB FrameWork öppnar upp möjligheten att samköra olika simuleringsmodeller. Detta innebär inte bara utökade möjligheter att samköra ACSL-modeller inom THSA, utan även att de inom projektet utvecklade modellerna kan komma till nytta i andra projekt inom FOI eller i samarbete med extern kund.

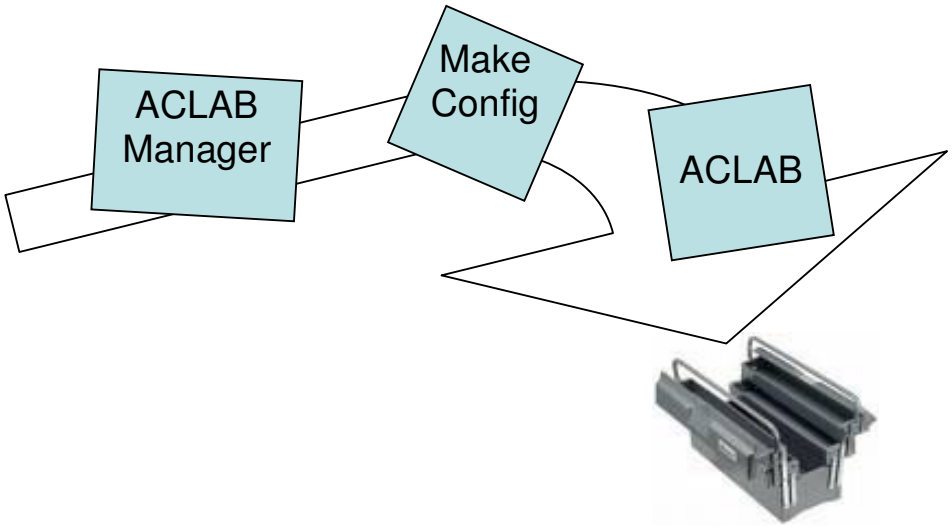
Rapporten är främst skriven för internt bruk men kan vara till användning även för andra utvecklare som använder ACSL och behöver knyta ihop en eller flera ACSL-modeller med någon extern simuleringsmiljö.

I rapporten används begreppet huvudmodell för att benämna en extern simuleringsmiljö som önskar utbyta data med en ACSL-modell. En ACSL-modell som kopplas upp mot huvudmodellen benämns submodell. Kursiv stil används för filer som används i ramverket, t.ex. *ACLAB.dll*.

I Kapitel 2 beskrivs vart och ett av de tre programmen som ingår i ACLAB Framework. Det beskrivs varför programmet behövs och hur det används. I kapitel 3 ges instruktioner om vilka moment användaren behöver gå igenom för att knyta submodeller till en huvudmodell med hjälp av ACLAB Framework. Specifikt behandlar kapitlet uppkoppling av submodeller till MATLAB/Simulink, C++-miljö och ACSL/GM. Kapitel 4 exemplifierar de tre metoder som behandlas i kapitel 3. I Kapitel 5 nämns ett antal begränsningar i ACLAB FrameWork 3.0 samt förslag på fortsatt arbete. Bilaga 1, 2 och 3 ger djupare teknisk information om ACLAB FrameWork med bland annat programkodsexempel.

2 Ramverket

Ramverket ACLAB FrameWork, se Figur 1, består av tre program som tillsammans kan användas för att möjliggöra kommunikation mellan en huvudmodell och submodeller. Dessa är ACLAB som fungerar som en länk mellan en extern miljö och en ACSL-modell, ACLAB Manager som förenklar handhavandet, speciellt vid kommunikation med multipla submodeller, och MakeConfig som skapar konfigurationsfiler och programmeringskod för inkopiering i huvudmodellen. Detta kapitel beskriver de tre programmen var för sig. En beskrivning av programmen på djupare teknisk nivå återfinns i Bilaga 1,2 och 3.



Figur 1 Ramverket ACLAB FrameWork kan ses som en verktygslåda med tre verktyg för underlättandet av kommunikation med ACSL-modeller

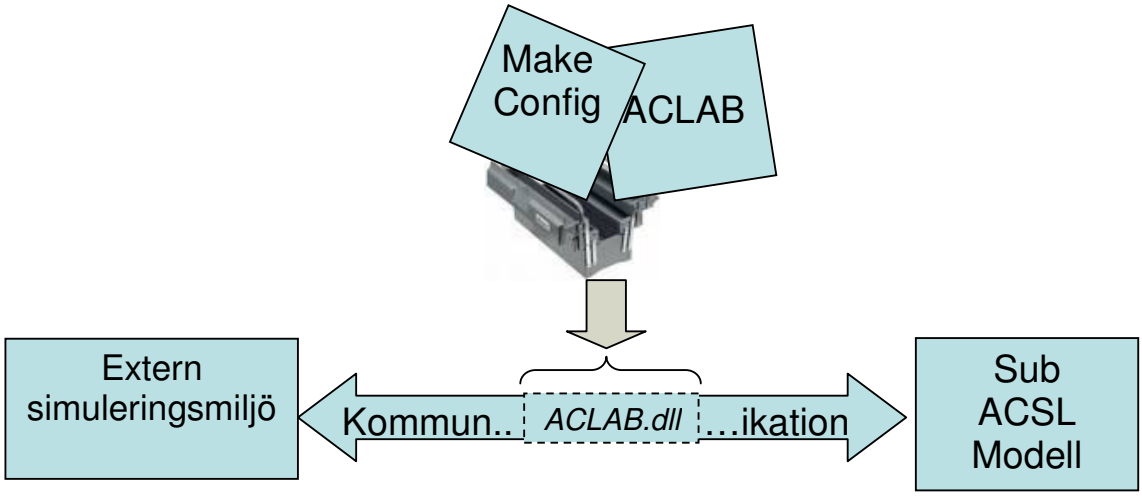
2.1 ACLAB

ACLAB är ett program som utvecklats i syfte att kunna kommunicera med en ACSL-modell (submodell) ifrån en extern simuleringsmiljö (huvudmodell), se Figur 2.

ACLAB är kompilerad till en *dll* som länkas in i huvudmodellen. Denna publicerar tre funktioner utåt som på så sätt blir åtkomliga från huvudmodellen. Det är genom dessa tre funktioner som all kommunikation med submodellen sker.

ACLAB.dll är konstruerad för att kommunicera med endast en submodell. För kommunikation med flera submodeller krävs multipla kopior av *ACLAB.dll*, en för varje submodell. Kommunikationen kräver att en kopia av *ACLAB.dll* följer med den aktuella submodellen och att den ligger i en separat mapp som döps till *ACLAB*, som skall ligga på samma nivå som modellens körbara fil, den s.k. *prx-filen*. Då kommunikation med endast *en* submodell önskas, kan ACLAB länkas in statiskt. För att en huvudmodell ska kunna kommunicera med flera submodeller har programet ACLAB Manager utvecklats. Detta utnyttjar dynamisk länkning och beskrivs i kapitel 2.2.

I mappen ACLAB i submodellen ska förutom en kopia av *ACLAB.dll* även en konfigurationsfil till submodellen ligga. Namnet på denna måste i ACLAB 3.0 FrameWork vara *AclabConfig.txt* eftersom det är detta filnamnet ACLAB söker efter. Denna fil innehåller information om de parametrar i submodellen med vilka kommunikation ska ske. För att skapa filen kan programmet MakeConfig användas. Detta program beskrivs ingående i kapitel 2.3.



Figur 2 ACLAB används för att skapa en kommunikationsväg mellan en huvudmodell och en submodell.

Anrop till ACLAB måste ske i kronologisk ordning eftersom ACLAB inte kan förmå submodellen att backa i tiden. Om ACLAB anropas med en tidigare tidsstämpel än vad som gjorts i tidigare anrop så kommer ACLAB att lämna samma utdata som lämnades vid anropet med senaste tidsstämpeln.

De tre funktioner i ACLAB genom vilka huvudmodellen kan kommunicera med submodellen är:

- `fnACLAB_EXPORTInit`
- `fnACLAB_EXPORT`
- `fnACLAB_EXPORTTerm`

Nedan följer en beskrivning av inargumenten till respektive funktion. Den exakta syntaxen återfinns i Bilaga 1 Aclab. Gemensamt för de tre funktionerna är att all numerisk data som sänds är av typen double. Detta innebär att parametrar av andra typer, t.ex. integer eller boolean, bör typkonverteras till double innan anropen görs[1].

`fnACLAB_EXPORTInit` anropas då en submodell ska initieras. Inargument till funktionen är:

- Sökvägen till den submodell med vilken kommunikation ska ske.
- Huvudmodellens aktuella tid. Denna tid ansätts som tiden noll lokalt i submodellen.
- Kortnamn på de parametrar i submodellen med vilka kommunikation ska ske. Detta görs i fem vektorer: INITIAL_INPUT, INITIAL_OUTPUT, INPUT, OUTPUT och TERMINAL.
- Antal parametrar i var och en i de fem vektorerna nämnda i föregående punkt.
- Initiala indatavärden. Dessa anges i en vektor och ordningen måste överensstämma med ordningen i vilken kortnamnen på parametrar i vektorerna INITIAL_INPUT är angivna.
- Initiala utdatavärden. Dessa anges i en vektor och ordningen måste överensstämma med ordningen i vilken kortnamnen på parametrar i vektorerna INITIAL_OUTPUT är angivna.

`fnACLAB_EXPORT` anropas upprepande under simuleringens gång. Inargument till funktionen är:

- Huvudmodellens aktuella tid. Denna tid omvandlas till lokal tid i submodellen.
- En minsta steglängd. Submodellen avancerar bara i tiden ifall det önskade tidssteget är större än detta värde.

[1] Då ACSL används som huvudmodell, och GSL-filen automatgenereras, kommer typkonvereringen att ske automatiskt i den genererade koden.

- Indata från huvudmodellen till submodellen. Ordningen måste överensstämma med ordningen i vilka kortnamnen angavs i vektorn INPUT i `fnACLAB_EXPORTInit`-anropet.
- Utdata från submodellen till huvudmodellen. Ordningen måste överensstämma med ordningen i vilka kortnamnen angavs i vektorn OUTPUT i `fnACLAB_EXPORTInit`-anropet.

`fnACLAB_EXPORTTerm` aropas då huvudmodellen önskar avbryta kontakten med submodellen. Inargument till funktionen är:

- Flagga som ska ha värdet 1.0. Denna visar att Terminal-funktionen i ACLAB har anropats externt och inte internt från ACLAB självt.
- Terminala utdata från submodellen. Ordningen måste överensstämma med ordningen i vilka kortnamnen angavs i vektorn TERMINAL i `fnACLAB_EXPORTInit`-anropet.

2.2 ACLAB Manager

Att kommunicera med ACLAB via ACLAB Manager har två fördelar gentemot att kommunicera direkt mot ACLAB:

1. Förenklad kommunikation med flera submodeller i samma simulering
2. Parameternamn behöver ej anges som inargument till initialfunktionen utan dessa läses automatiskt från *AclabConfig.txt*

2.2.1 Bakgrund

Att kommunicera direkt med ACLAB från en extern simuleringsmiljö fungerar bra så länge som endast en submodell är inblandad. Om kommunikation med flera submodeller ska göras uppstår dock problem. Till var och en av submodellerna följer filen *ACLAB.lib* som innehåller namnen på de exporterade funktionerna i den kompilerade filen *ACLAB.dll*. Vid statisk länkning kommer flera funktioner med samma namn länkas in eftersom samma namn återkommer i varje kopia av *ACLAB.dll*. Vid ett senare anrop till en sådan funktion är det inte tydligt vilken av submodellerna som anropas.

Problemet med att kommunicera med flera submodeller kan lösas genom att istället för att länka in ACLAB.lib för varje submodell, länka in motsvarande *ACLAB.dll* dynamiskt. Detta kan göras med hjälp av sökvägen till respektive submodell. Den dynamiska länkningen innebär att unika handtag erhålls till respektive submodell genom vilka kommunikation kan göras. Kontakt med submodeller kan väljas att upprättas först när/om huvudmodellen så önskar i den aktuella simuleringen, och inte förutsättningslöst i huvudmodellens initialdel som är fallet vid statisk länkning. Ett exempel som gick ut på att implementera dynamisk länkning i en huvudmodell beskivs i kapitel 4.2.

2.2.2 ACLAB Manager

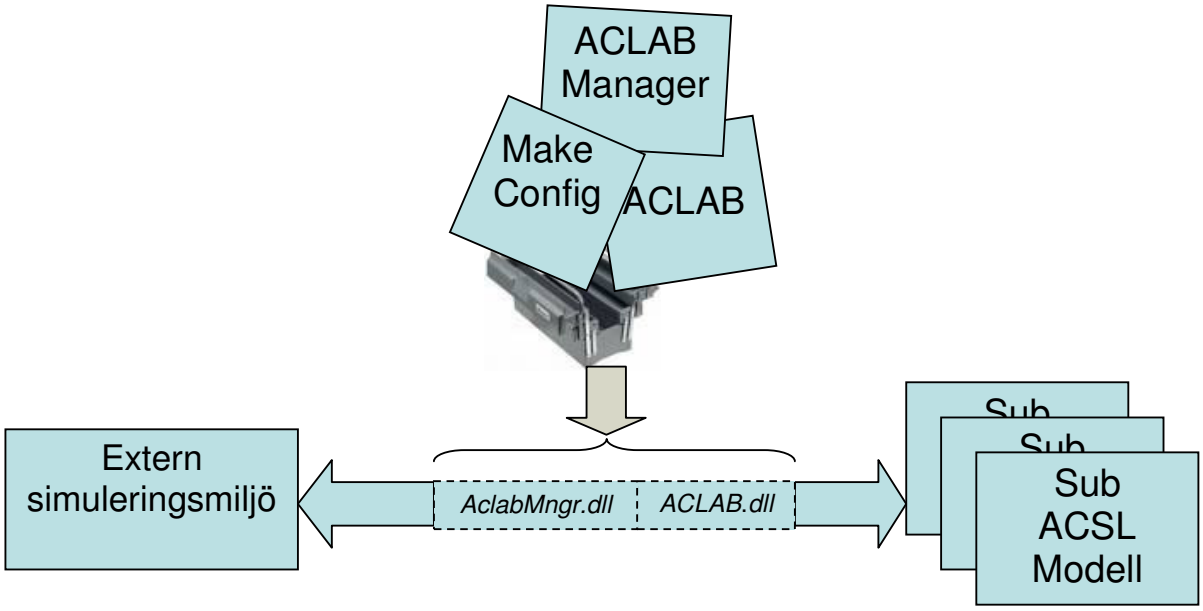
Då dynamisk länkning är en aning komplicerat har ACLAB Manager utvecklats som ett mellanlager för att underlätta för användaren. ACLAB Manager gör den dynamiska länkningen automatiskt och är ytterligare ett hjälpmedel för att förenkla och generalisera kommunikationen mellan en huvudmodell och en submodell, se Figur 3. ACLAB Manager länkas in statiskt i huvudmodellen och innehåller tre externa funktioner motsvarande dem i ACLAB. De tre funktionsnamnen är: AclabMngrEntryInit, AclabMngrEntryRun och AclabMngrEntryTerm.

Vid kommunikation direkt med ACLAB krävs, som nämndes i kapitel 2.1, att samtliga parametrar med vilka värden ska utbytas måste anges i funktionshuvudet till initialfunktionen. Då ACLAB

Manager används ska detta inte göras. Istället läser ACLAB Manager in parameterinformationen från *AclabConfig.txt* och ACLAB Manager antar att utbyte ska göras med samtliga dessa parametrar och ingen därutöver. Detta avser förenkla handhavandet för utvecklaren av huvudmodellen. En konsekvens av detta är att samtliga parametrar som finns i *AclabConfig.txt* behöver tas om hand i huvudmodellen. Det vill säga huvudmodellen måste se till att den innehåller motsvarigheter till dessa parametrar.

För att ACLAB Manager ska veta vilken submodell som huvudmodellen vill kommunicera med vid ett anrop, så måste sökvägen till den aktuella submodellen skickas med som ett inargument. Detta gäller vid anrop med samtliga tre funktionerna.

Det är upp till programmeraren av huvudmodellen att se till att anropen görs i önskade tidpunkter. Terminalanropet har bara effekt då det görs första gången. Om anrop görs flera gånger kommer någon ACLAB-instans av den aktuella submodellen ej längre finnas, och därmed finns heller ej några terminaldata sparade. Det är därför viktigt att programmeraren av huvudmodellen ser till att terminalvärden sparas efter första terminalanropet och att dessa ej skrivs över vid eventuella ytterligare anrop.



Figur 3 Ramverket erbjuder tre verktyg för att möjliggöra kummunikation med ACSL-modeller

Nedan följer en beskrivning av inargumenten till respektive funktion. Den exakta syntaxen återfinns i Bilaga 2 ACLAB Manager. Gemensamt för de tre funktionerna är, liksom för motsvarande funktioner i ACLAB, att all numerisk data som sänds är av typen double. Detta innebär att parametrar av andra typer, t.ex. integer eller boolean, bör typkonverteras till double innan anropen görs[2].

`fnACLAB_EXPORTInit` anropas då en submodell ska initieras. Inargument till funktionen är:

- Sökvägen till den submodell med vilken kommunikation ska ske.
- Huvudmodellens aktuella tid. Denna tid ansätts som tiden noll lokalt i submodellen.

[2] Då ACSL används som huvudmodell, och GSL-filen automatgenereras, kommer typkonvereringen att ske automatiskt i den genererade koden.

- Initiala indatavärden. Dessa anges i en vektor och ordningen måste överensstämma med ordningen i vilken parameternamnen förekommer i *AclabConfig.txt*.
- Initiala utdatavärden. Dessa anges i en vektor och ordningen måste överensstämma med ordningen i vilken parameternamnen förekommer i *AclabConfig.txt*.

`fnACLAB_EXPORT` anropas upprepande under simuleringens gång. Inargument till funktionen är:

- Sökvägen till den submodell med vilken kommunikation ska ske.
- Huvudmodellens aktuella tid. Denna tid omvandlas till lokal tid i submodellen.
- En minsta steglängd. Submodellen avancerar bara i tiden ifall det önskade tidssteget är större än detta värde (sätts till 0.0 ifall ingen sådan begränsning önskas).
- Indata från huvudmodellen till submodellen. Dessa anges i en vektor och ordningen måste överensstämma med ordningen i vilken parameternamnen förekommer i *AclabConfig.txt*.
- Utdata från submodellen till huvudmodellen. Dessa anges i en vektor och ordningen måste överensstämma med ordningen i vilken parameternamnen förekommer i *AclabConfig.txt*.

`fnACLAB_EXPORTTerm` aropas då huvudmodellen önskar avbryta kontakten med submodellen. Inargument till funktionen är:

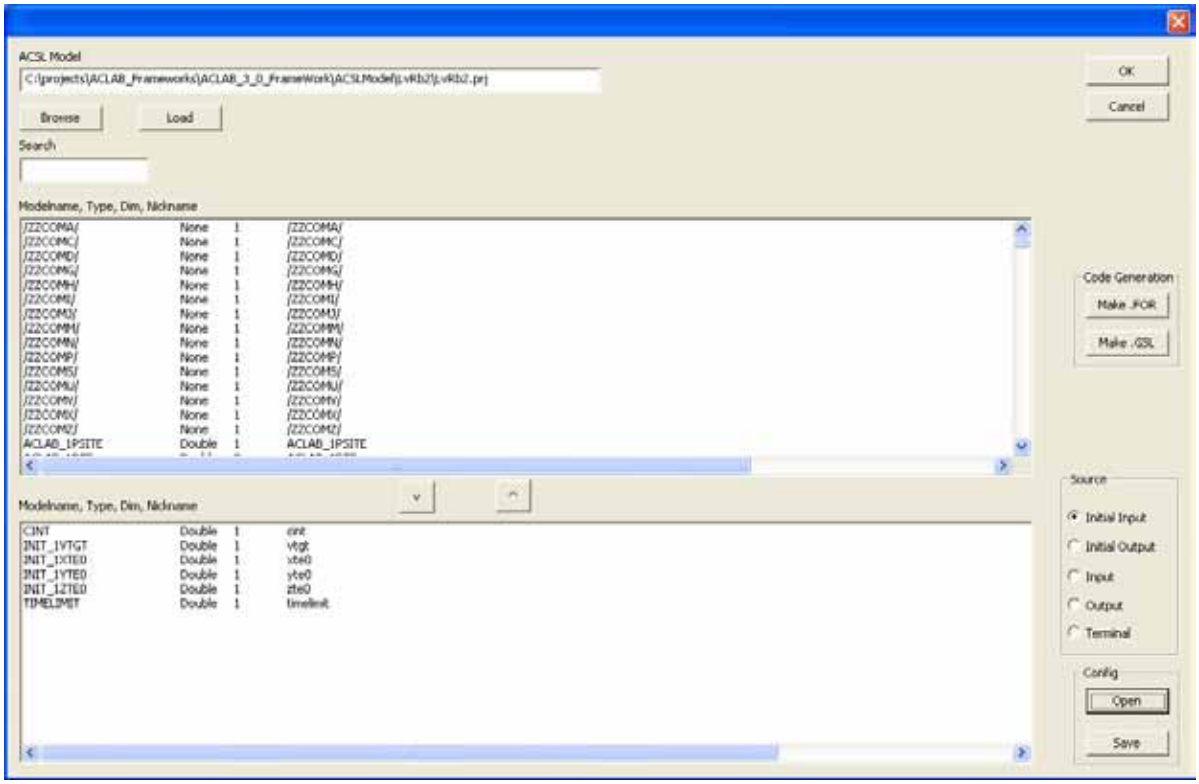
- Sökvägen till den submodell med vilken kommunikation ska ske.
- Terminala utdata från submodellen. Dessa anges i en vektor och ordningen måste överensstämma med ordningen i vilken parameternamnen förekommer i *AclabConfig.txt*.

2.3 Make Config

Ramverket ACLAB Framework använder ett antal filer vid körning. ACLAB och ACLAB Manager använder en konfigurationsfil som listar de parametrar som skall utbytas mellan huvudmodellen och submodellen utvecklad i ACSL. Då huvudmodellen även den utgörs av en ACSL-modell är det dessutom nödvändigt med subrutiner som anropar ACLAB Manager samt ACSL-kod, s.k. CSL-kod, som anropar subrutinerna. För att underlätta framtagandet av alla dessa filer har applikationen Make Config tagits fram. I Make Config laddas submodellen in och användaren kan sedan enkelt välja de parametrar som skall utbytas och sedan automatgenerera nödvändiga filer för den aktuella tillämpningen. Filerna som kan gereneras med Make Config är *AclabConfig.txt*, som används av submodellerna, samt .for- och .gsl-filerna som används av huvudmodellen i det fall huvudmodellen är en ACSL-modell.

MakeConfig startas via den körbara filen *MakeConfig.exe*.

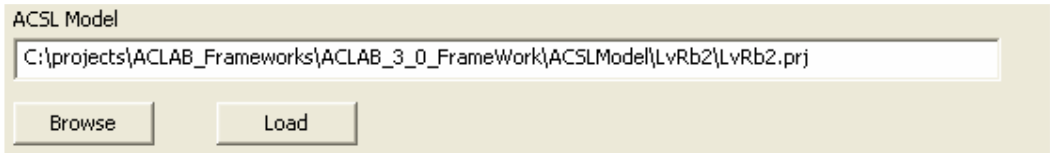
Figur 4 nedan visar användargränssnittet för Make Config.



Figur 4 Användargränssnittet Make Config

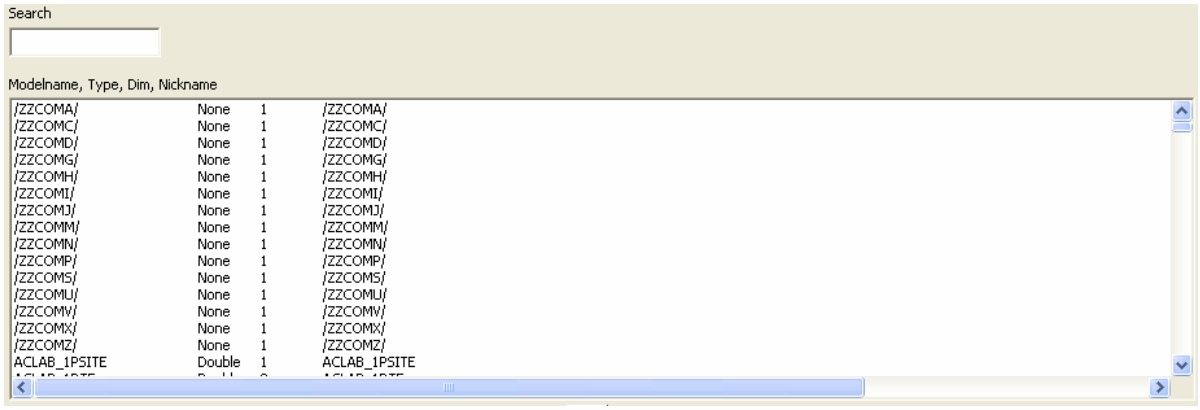
Nedan följer en beskrivning av de olika val som kan göras i användargränssnittet.

1. Ladda simuleringsmodell:



Används för att ladda den submodell som huvudmodellen skall utbyta värden med. Samtliga modellvariabler kommer att laddas in i Make Config i och med att load görs på en vald simuleringsmodell.

2. Modellvariabellista:

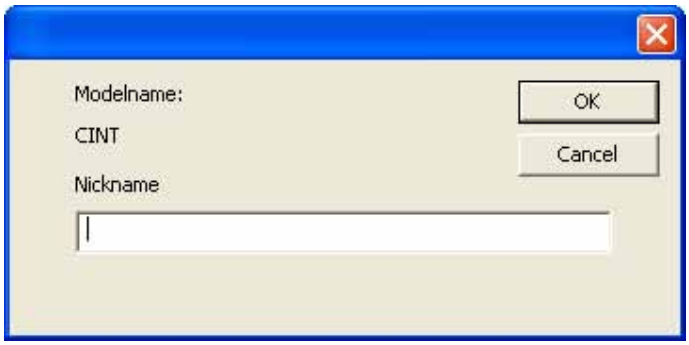


Då modellen laddats listas alla modellvariabler i den övre listboxen. En modellvariabel per rad visas där varje kolumn motsvarar modellnamn, typ av variabel, dimension på variabeln samt eventuellt nickname valt på variabeln (i de fall inget nickname valts är nickname samma som modelname). Listan kan sedan göras kortare genom att söka efter aktuellt modellnamn i edit boxen ”Search”. En inledande asterisk, ’*’, kommer att ge alla variabler som innehåller den valda sökströngen, d.v.s. *miss ger träff på Msl_Final_missdistance o.s.v.

3. val av parametrar:

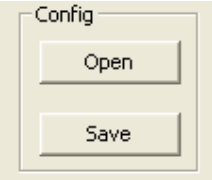
Pilknapparna används för att välja respektive ta bort parametrar som skall utbytas med submodellen. Valda parametrar visas i den nedre listboxen på samma sätt som i den övre listboxen. Innan en parameter flyttas ned till den nedre listboxen skall ett val göras, under Source, vilken typ av parameter det är. Fem val kan göras: Initial Input, Initial Output, Input, Output respektive Terminal. Input-parametrar är parametrar som skall sättas i submodellen från huvudmodellen initialt respektive under körning. På samma sätt är Outputs parametrar som returneras från submodellen. Terminal är parametrar som returneras i ACSL-modellens terminalsektion. Observera dock att inga parametar kan sättas i submodellen i terminalsektionen. När parametrar är valda kommer sedan listboxen att uppdateras dynamiskt då knappvalet ändras under Source för att visa de valda parametrarna av respektive kategori.

4. Val av nicknames:



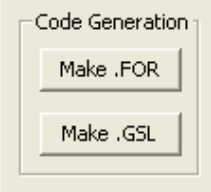
Val av nicknames för parametrarna kan göras via dialogen visad ovan. Dialogen öppnas då användaren dubbelklickar på en parameter i listan. Flera parametrar kan markeras på en gång. En dialog per parameter, uppifrån och ner i parameterlistan, kommer då att öppnas.

5. Öppna/skapa konfigurationsfil:



Knapparna används för att öppna respektive automatgenerera en konfigurationsfil. Konfigurationsfilen innehåller de parametrar som kommuniceras mellan huvudmodellen och submodellen. Bilaga 3 visar utseendet på en konfigurationsfil, skapad av Make Config, och beskriver dess funktionalitet närmare. Default-namn hos konfigurationsfilen är *AclabConfig.txt* vilket är det namn ACLAB och ACLAB Manager söker efter under en simulering.

6. Kodgenerering:



Då huvudmodellen är en ACSL-modell som skall kommunicera med en ACSL-modell kan ovanstående knappar användas för att automatgenerera nödvändig kod. ”Make .FOR” skapar en Fortranfil med subrutiner som skall länkas in statiskt i huvudmodellens ACSL/GM-miljö. ”Make .GSL” skapar på samma sätt en gsl-fil som kan länkas in i ett block i samma ACSL/GM-modell. Blocket anropar, via gsl-filen, subrutinerna i Fortran. Dessa kommunicerar i sin tur med ACLAB Manager som i sin tur kommunicerar med ACLAB som kommunicerar med submodellen. Denna kommunikationskedja beskrivs ytterligare i kapitel 0. Utgående parametrar från submodellen går sedan samma väg tillbaka för att returneras i gsl-filen. Det användaren måste göra, förutom att länka in nödvändiga filer, är att mata gsl-blocket med nödvändiga parametrar samt ta hand om de returnerade värdena. Bilaga 3 beskriver de genererade filerna.

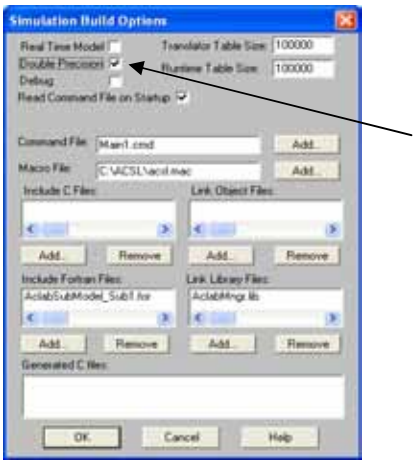
Hur filerna, som kan generas av Make Config, länkas in i en ACSL-modell beskrivs i kapitel 3.1.

3 Användarhandledning

Detta kapitel beskriver tillvägagångssätt för att koppla upp en eller flera submodeller mot MATLAB/Simulink, en submodell mot C++-miljö, samt en eller flera submodeller mot en ACSL-modell.

För samtliga fall av uppkoppling gäller att:

- i ACLAB FrameWork 3.0 ligger *ACLAB.dll*, *ACLAB.lib*, *ACLAB_EXPORT.h*, *AclabMngr.dll*, *AclabMngr.h*, *AclabMngr.lib* samt *MakeConfig.exe* i katalogen *bin*.
- det finns minst en kompilerad submodell tillgänglig. Med kompilerad menas att en fil med ändelsen *.prx* och en fil med ändelsen *.prj* har skapats. Den katalog i vilken dessa filer befinner sig kallas submodellens projektmapp.
- den aktuella submodellen ska ha en katalog i sin projektmapp på hårddisken som heter ACLAB. Denna ska innehålla en kopia av *ACLAB.dll* samt en konfigurationsfil med namnet *AclabConfig.txt*.
- MakeConfig.exe kan användas för att generera den till respektive submodell nödvändiga konfigurationsfilen *AclabConfig.txt*, se kapitel 2.3.
- en submodell inte kan kopplas upp mot en huvudmodell om den samtidigt används av ett annat program t.ex *MakeConfig.exe*, se avsnitt 2.3, ACSL/GM [ACSL], AFE [AFE] eller Batchman [BATCHMAN].
- dubbel precision skall användas för de data som ska utbytas, vilket innebär att flyttal blir av typen double som beskrivs med 8 byte. Detta gäller för version 3.0 av ACLAB Framwork då datautbyte med enkel precision ej är tillräckligt utprovat. För ACSL-modeller (submodell samt ev. huvudmodell) görs detta genom att kompilera med dubbel precision, se Figur 5 (dialogen öppnas från GM från menyn ”Simulate/Build Options…”). Från C/C++ görs detta t.ex. genom manuell typkonvertering (eng. cast) av de parametrar som ska utbytas.



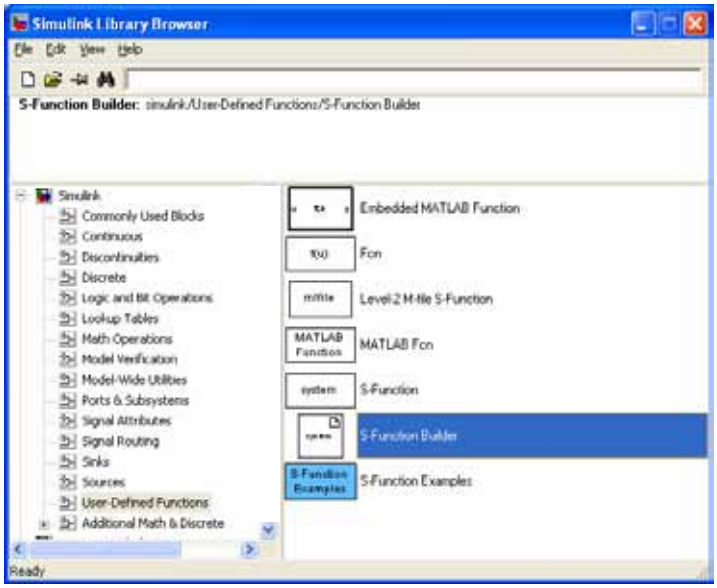
Figur 5 Dialog i ACSL/GM för att ändra precision hos de i modellen ingående flyttalen

- huvudmodellen ska innehålla en kopia av *AclabMngr.lib* (länkas in i huvudmodellen), *AclabMngr.dll* samt *AclabMngr.h*. *AclabMngr.h* behövs dock ej om huvudmodellen är en ACSL-modell.
- kommunikation sker genom att från huvudmodellen anropa tre funktioner i AclabMngr. Inargumenten till dessa funktioner innehåller flyttalsarrayer med parameterdata, se kapitel 2.2.2 samt Bilaga 2. Vilka parametrar datat tillhör och vilken ordningsföljd dessa ska ha i arrayerna definieras av den anropade submodellens konfigurationsfil, *AclabConfig.txt*. Om en parameter är en flerdimensionell vektor läggs vektorns ingående element efter varandra

i arrayen. Då huvudmodellen är en ACSL-modell kan filer med dessa funktionsanrop genereras automatiskt med MakeConfig. Mer om detta i kapitel 2.3 och 0.

3.1 Anrop från Matlab/Simulink

Genom att använda blocket S-Function Builder, se Figur 6, kan C-kod skrivas i en Simulinkapplikation. Vid dubbelklick på blocket visas en dialog från vilken blocket specificeras. Här anges vilka portar som ska knytas till blocket, vilka filer som ska inkluderas och länkas in och här skrivs C-kod. Lägg en kopia av *AclabMngr.lib*, *AclabMngr.dll* och *AclabMngr.h* i huvudmodellens projektmapp. Genom att i dialogen länka in *AclabManager.lib* (behövs för länkaren) och inkludera *AclabMngr.h* (behövs för kompilatorn) och sedan anropa de tre exporterade funktionerna från C-koden så sker det önskade datautbytet med ACSL-modellen. Kapitel 4.1 visar ett enkelt exempel där kommunikation med en ACSL-modell sker från Simulink.



Figur 6 S-Function Builder block

Flera instanser av samma submodell fås genom att kopiera och döpa om katalogen, på hårddisken eller motsvarande lagringsmedium, som submodellen finns i. Dessutom måste ett nytt S-Function Builder block läggas in i huvudmodellen och indata till detta måste tillföras. Koden från det andra blocket kan kopieras in i detta block. Därutöver måste sökvägen till den nya submodellen anges som det första argumentet i de tre funktionsanropen.

Som nämndes i avsnitt 2.2 så är det upp till programmeraren av huvudmodellen att se till att anropen görs i önskade tidpunkter. Terminalanropet har bara effekt då det görs första gången. Om anrop görs flera gånger kommer någon ACLAB-instans av den aktuella submodellen ej längre finnas, och därmed finns heller ej några terminaldata sparade. Det är därför viktigt att programmeraren av huvudmodellen ser till att terminalvärden sparas efter första terminalanropet och att dessa ej skrivs över senare.

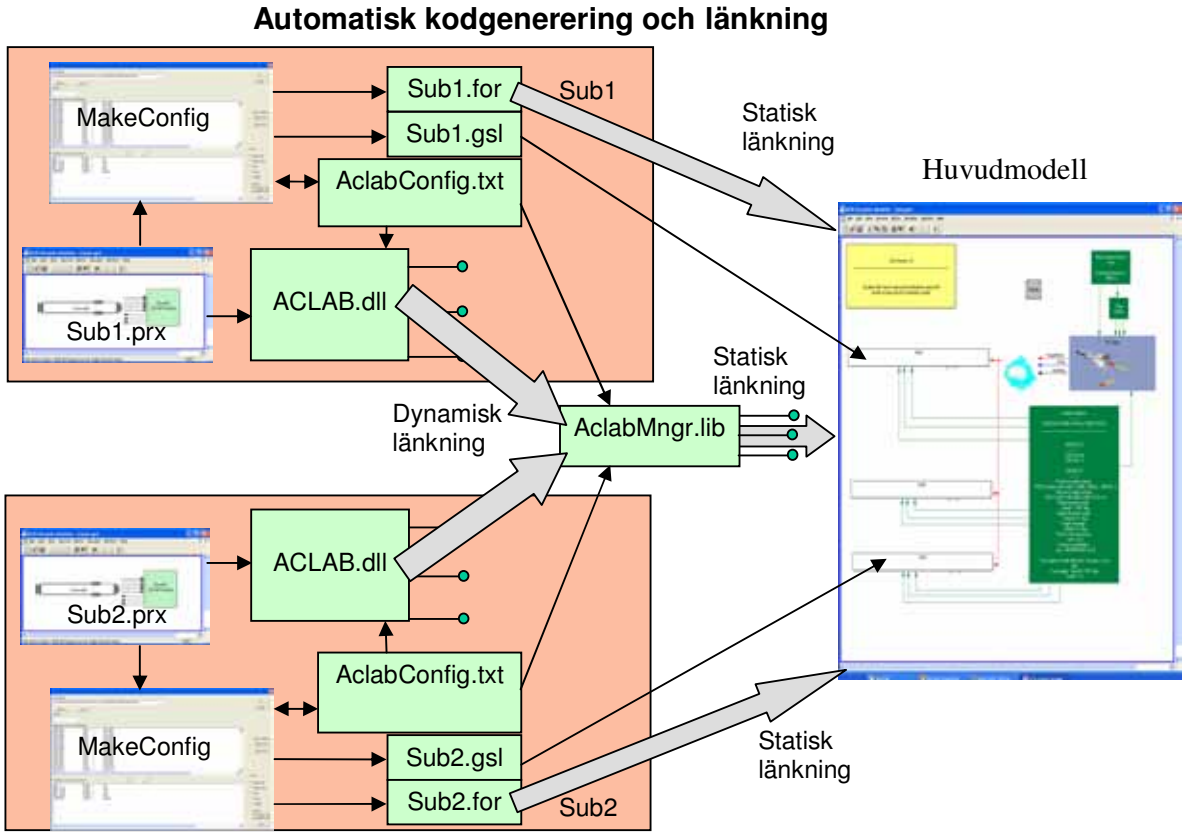
3.2 Anrop från C++program

På samma sätt som anrop till ACLAB Manager görs från Matlab Simulink, via ett C-anrop, kan anrop göras från en godtycklig C++-applikation. Genom att länka in AclabManager.lib och inkludera *AclabMngr.h* i ett C++-projekt kan anrop göras till ACLAB Managers tre exporterade funktioner genom vilka kommunkationen med ACSL-modellen sköts.

Kapitel 4.2 visar ett exempel där ACLAB använts i en C++-applikation för att knyta ihop ett multispektralt nätverk med en detaljerad robotmodell utvecklad i ACSL.

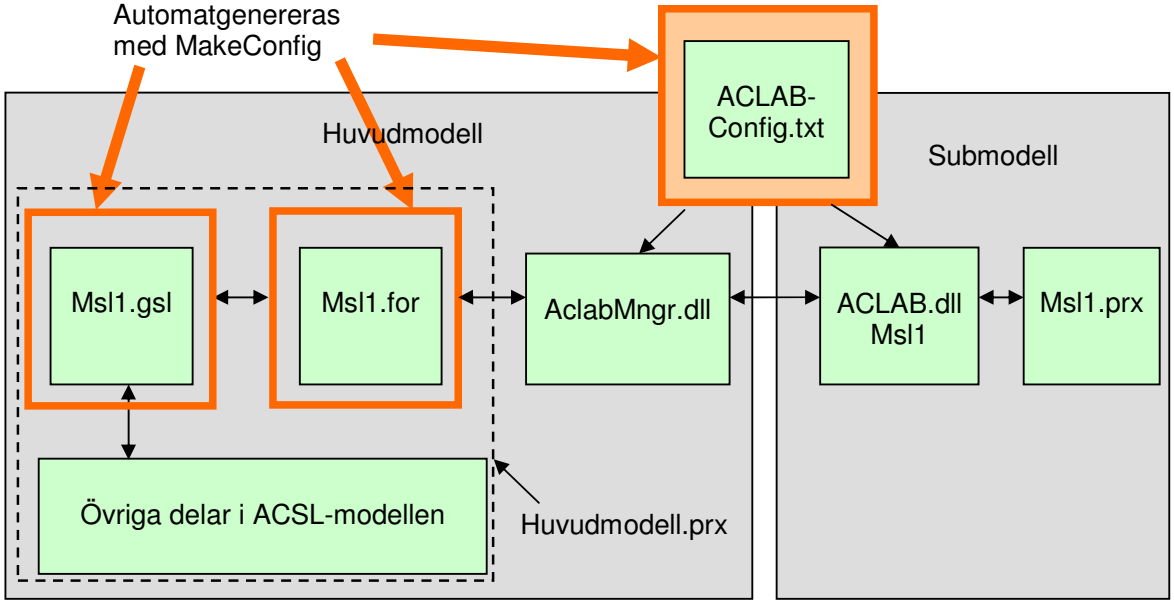
3.3 Anrop från ACSL/GM

ACLAB Manager kan användas för att knyta ihop flera ACSL-modeller. Huvudmodellen kommer i detta fall vara en ACSL-modell som anropar en eller flera submodeller utvecklade i ACSL. I kapitel 2.3 beskrevs hur Make Config kan användas för att automatgenerera de filer som behövs för att länka ihop en ACSL-modell med andra ACSL-modeller. Figur 7 visar hur filerna samverkar. Make Config läser in submodellernas robotmodellparametrar via modellernas respektive prx-filer, *Msl.prx*, och använder sedan dessa för att generera filerna *AclabConfig.txt*, *Msl.gsl* och *Msl.for*. *AclabConfig.txt* behövs för submodellerna och *Msl.gsl* respektive *Msl.for* behövs för huvudmodellen. ACLAB Manager skall länkas in statiskt, via dess lib-fil, i huvudmodellens ACSL-miljö. Detta gäller även genererade Fortranfiler. ACLAB Manager kommer, vid körning, att länka in respektive modells *ACLAB.dll* dynamiskt.



Figur 7 Samband mellan samverkande filer då ACLAB Manager används för att koppla upp två submodeller mot en huvudmodell utvecklad i ACSL/GM. Grå pilar anger länkning som behövs för simulering. Svarta pilar anger övrig kommunikation mellan filer och program.

Figur 8 visar kommunikationskedjan då ACSL anropar en annan ACSL-modell via ACLAB Framework. Ett gsl-block i huvudmodellen, *Huvudmodell.prx*, anropar en subrutin i filen *Msl1.for*. Fortranfilen anropar *AclabMngr.dll* som i sin tur kommunicerar med submodellen via *ACLAB.dll*. ACLAB och ACLAB Manager läser parameterspecifikationer från *AclabConfig.txt*.



Figur 8 Funktionsanropskedja vid anrop ACSL till ACSL.

Då ACLAB Framework skall användas för att koppla ihop två eller flera ACSL/GM-modeller, måste ett par handgrepp genomföras manuellt av användaren för att länka in nödvändiga filer i huvudmodellen. Dessa beskrivs nedan.

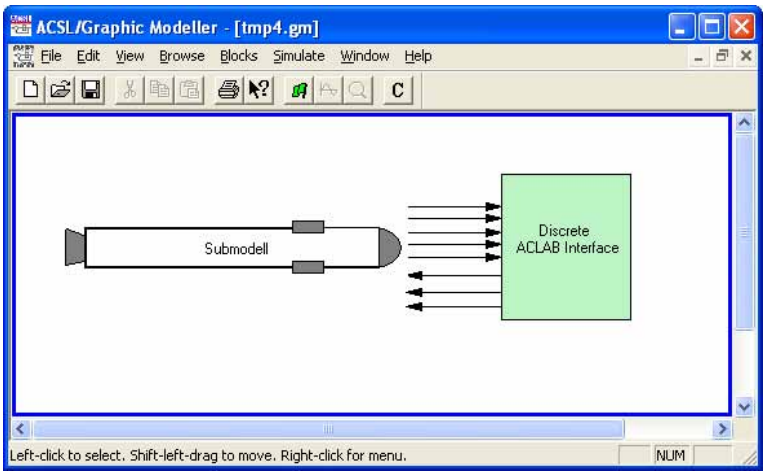
1. **Statisk inlänkning av subrutinsfil.**
   Subrutiner som anropar ACLABManager måste länkas in i huvudmodellen. Dessa kan vara skrivna i C eller Fortran. Används MakeConfig kan en Fortran-fil som gör detta automatgenereras. Filen eller filerna länkas in i huvudmodellen via ACSL/GM menyn ”Simulate->Build Options…” under antingen ”Include Fortran Files” eller ”Include C Files”.
2. **Anrop till subrutinerna.**
   Subrutinerna måste i sin tur anropas från ett ACSL-block. Om Make Config används för att generera Fortran-filen enligt ovan, kan även en matchande gsl-fil genereras på samma sätt. Denna kan sedan knytas till ett ACSL/GM-block. Filen innehåller anrop till de subrutiner som länkats in ovan. Variabler som skickas till subrutinerna måste matas med värden från huvudmodellen, vilket kräver manuella åtgärder av användaren även om filen automatgenererats från Make Config. ACSL/GM-blocket måste sedan göras diskret och anropas via ”Schedule at” eller ”trigged by interval”[3].
3. **Diskret gränssnitt i submodellen**
   De parametrar i submodellen vilkas värden ska utbytas bör vara samlade i ett och samma diskreta block i submodellen, se Figur 9. Förutom att detta ger en bra överblick över vilka parametrar som utbyts i submodellen så garanterar detta att data utbyts i önskade tidpunkter[3]. I ACLAB FrameWork 3.0 finns ingen hjälp för att generera detta block. För att indata till subblocket ska nyttjas från och med samma tid som det skickas från

[3]ACSL använder integrationsalgoritmer av antingen fast eller variabel steglängd. Vid variabel steglängd kommer ACSL att hoppa framåt och bakåt i tiden för att hitta en steglängd som ger ett fel som understiger det minsta fel som tillåts. Kronologiska uppdateringar av tiden i ACSL garanteras dock inte ens om en integrationsalgoritm med fast steglängd valts. T.ex. kan ett Schedule statement, som triggas av en nollgenomgång, medföra att steglängden ändras och att algoritmen hoppar bakåt i tiden för att hitta den exakta tiden för nollgenomgången. Endast diskreta sektioner, som triggas av Schedule At eller som är trigged by interval, säkerhetsställer att tidsuppdateringarna sker kronologiskt. Eftersom ACLAB kommer att koppla ihop två olika ACSL-modeller med separata integrationsalgoritmer bör därför sammankopplingen ske via diskreta block.

huvudmodellen måste det diskreta blocket triggas (via ”Schedule at” eller ”trigged by interval”) med samma tidsintervall som motsvarande diskreta block i huvudmodellen.

4. **Statisk inlänkning av ACLAB Manager.**
   ACLAB Manager´s lib-fil länkas in i ACSL/GM. Även detta görs via ”Simulate->Build Options”.

För att länka in flera submodeller upprepas steg 1,2 och 3 ovan. Varje submodell måste ha egna subrutinsanrop. Observera att varje submodell också måste ha ett unikt namn. Om t.ex. samma robot skall länkas in flera gångar, för exempelvis test av dubbelskott, måste en separat modell med ett nytt namn skapas för varje instans av roboten. Detta beror på en begränsning i ACSL. Kapitel 4.3 visar ett exempel på hur en huvudmodell i ACSL/GM länkar in två robotmodeller.



Figur 9 Submodell utökad med ett diskret ACSL-block för att samla utbytesparametrar samt för att garantera korrekt tidshantering.

En begränsning i ACLAB FrameWork 3.0 är att det saknas stöd för matrishantering. Då en matris skickas från en modell till en annan genom ACLAB kommer den mottagande modellen erhålla matrisen i form av en vektor där kolumnerna i den ursprungliga matrisen placerats under varandra med den första kolumnen överst. Då exempelvis matrisen

$$matr = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

med dimensionen 2,3 (`DIMENSION(2,3)`) skickas, kommer den motagande modellen erhålla en vektor *vect* med dimensionen 6 (`DIMENSION(6)`):

$$vect = \begin{bmatrix} 1 \\ 4 \\ 2 \\ 5 \\ 3 \\ 6 \end{bmatrix}$$

Hanteringen av detta måste i ACLAB FrameWork 3.0 ske manuellt av användaren.

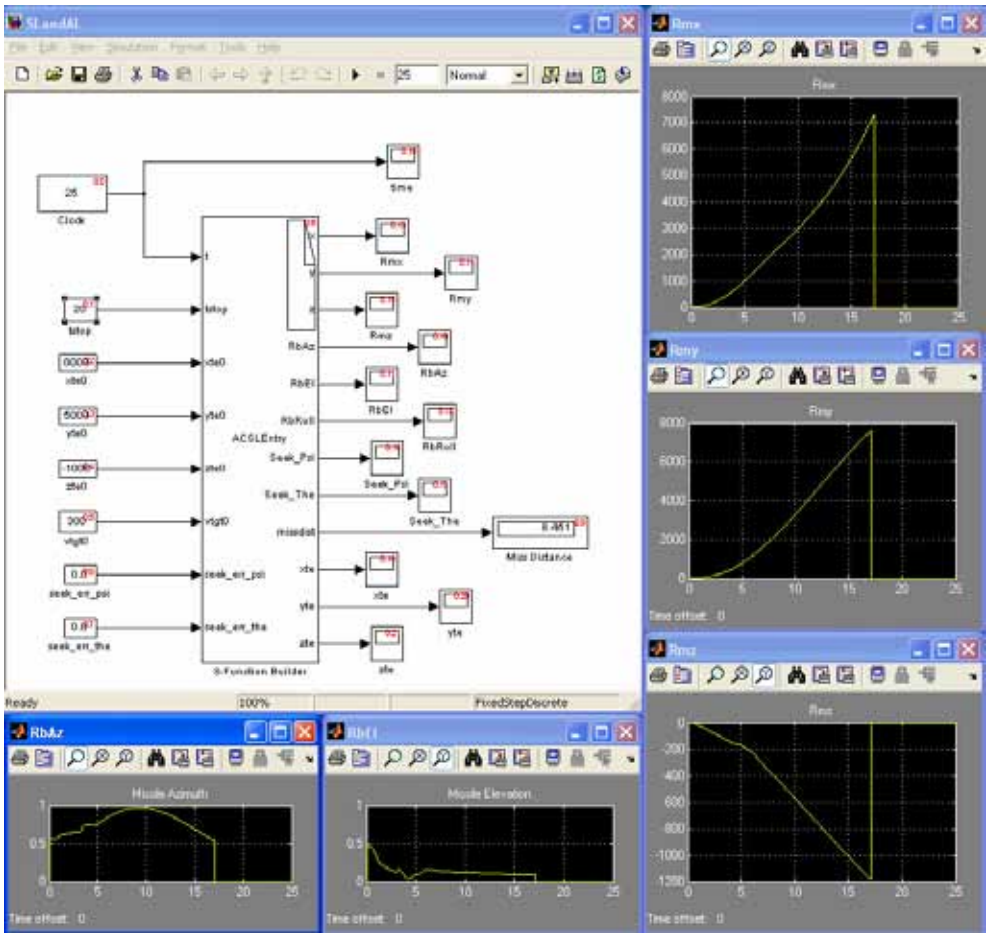
4 Exempel på applikationer

I detta kapitel exemplifieras de metoder som beskrivits i kapitel 3 genom att ett antal genomförda tillämpningar, där ACLAB har använts, presenteras.

4.1 Från Matlab/Simulink

I detta exempel är huvudmodellen en enkel applikation i Matlab/Simulink, se Figur 10. Huvudmodellen innehåller ingen väsentlig funktionalitet förutom att leverera indata till, samt plotta utdata från submodellen som i detta fall är en lv-robot och ett mål. Initiala indata består av initial position och hastighet för submodellens målobjekt, samt en maximal tid vid vilken huvudmodellen tvingar submodellen att terminera. Löpande input är huvudmodellens aktuella simuleringstid, som ger submodellen nya tider att tillåtas avancera till. Utdata är robotens position, eulervinklar samt bomavståndet.

Blocket ACSL Entry är ett S-function Builder block som anropar submodellen via ett C-anrop till ACLAB Manager.



Figur 10 Enkel Simulink-applikation för kommunikation med en ACSL-modell.

4.2 Från C++ program

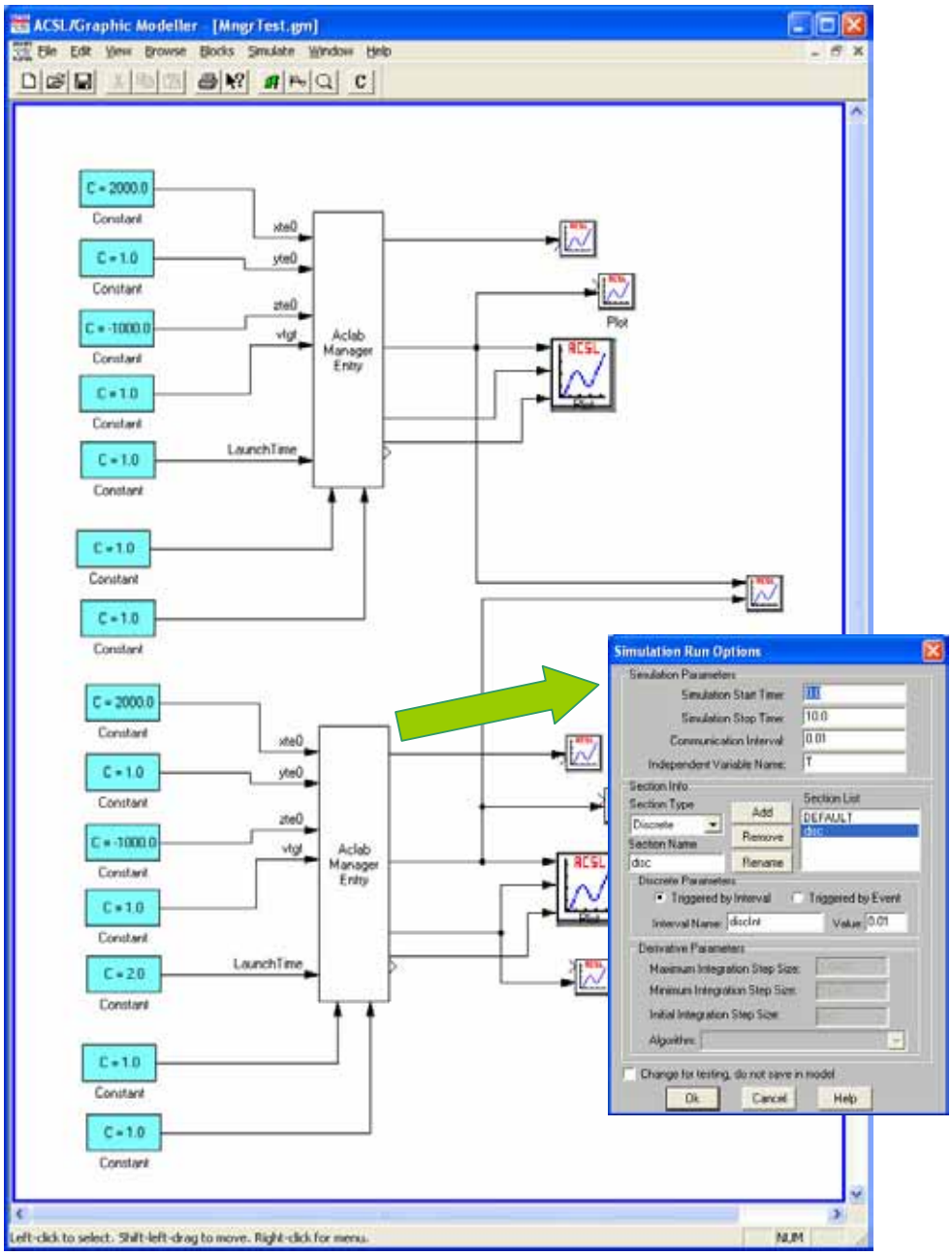
I studien Telekrigaspekter vid Asymmetrisk Krigföring vid Internationella insatser [TAAKII] har ACLAB använts för att knyta ihop ACSL med C++-applikationen EWSim. EWSim är ett multispektralt nätverk som simulerar elektrooptik, radar och kommunikation. Ramverket utvecklas vid FOI i miljön Visual Studio .NET [VISUAL]. I TAAKII-studien användes ACLAB för att knyta ihop EWSim med en ACSL-modell av ett robotsystem. Figuren nedan visar användargränssnittet i EWSim. En skytt riktar robotsiktet mot målet och avfyrar roboten. EWSim används för att räkna ut målsökardata som sedan skickas via ACLAB till ACSL-modellen.

I denna applikation har det dynamiska inlänkningsförfarandet programmerats explicit i huvudmodellen. Syftet med att länka dynamiskt är att möjliggöra kommunikation med flera ACSL-modeller i en och samma simulering. För att förenkla programmeringsarbetet för utvecklaren av huvudmodellen så har verktyget ACLAB Manager, se avsnitt 2.2, senare utvecklats för att sköta den dynamiska länkningen med submodeller.

Figur 11 EWSim under robotens inflygning, figurens övre vänstra del, samt i träffögonblicket, figurens nedre högra del

4.3 Från ACSL-modell

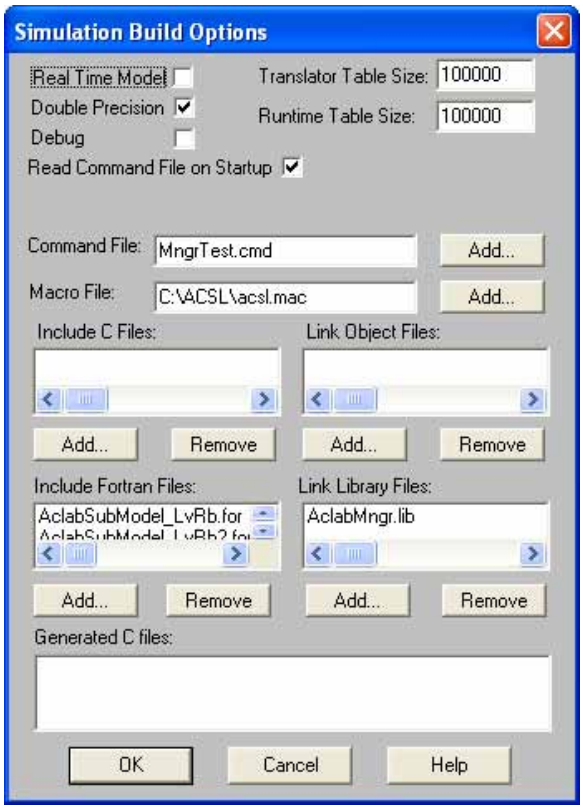
Huvudmodellen består i detta exempel av en ACSL/GM-modell som länkar in en submodell, även den gjord i ACSL/GM. Huvudmodellen heter i exemplet MngrTest.gm. MngrTest anropar de två submodellerna LvRb och LvRb2. LvRb-modellerna, som är modeller av en luftvärnsrobot, matas med startvärden för roboten. Returnerade robotparametrar tas emot och plottas för att verifiera resultatet. Figur 12 visar huvudmodellen MngrTest.gm.



Figur 12 Huvudmodellen MngrTest.gm

Robotmodellen, LvRb2 i exemplet, är en kopia av LvRb. Skillnaden är att robotarna kommer att avfyras vid tiderna t=1 respektive t=2 sekunder, vilket styrs av parametern LaunchTime. Det övre blocket, vid namn Aclab Manager Entry, sköter kommunikationen med LvRb. Det undre blocket, med samma namn, sköter kommunikationen med LvRb2. De två blocken är knutna till en diskret sektion som triggas med intervallet 0.01 sekunder. Deras respektive gsl-filer innehåller kod som anropar subrutiner i inlänkad Fortran-kod. Figur 13 visar åtgärder gjorda i Build Options-dialogen

för att länka in Fortran-filerna, *AclabSubModel_LvRb.for* och *AclabSubModel_LvRb2.for* samt ACLAB Managers lib-fil, *AclabMangr.lib*. Dessa filer är nödvändiga för kommunikationen mellan ACSL och ACLAB Manager. Fortran-filerna, som anropar de exporterade funktioner i lib-filen, är automatgenererad i Make Config och visas i Bilaga 3 Make Config. gsl-filerna, som tillhör Aclab Manager Entry, visas också i Bilaga 3 Make Config.



Figur 13 Build Options-dialogen. Fortran-filer och Lib-fil är inkluderade för anrop till ACLAB Manager.

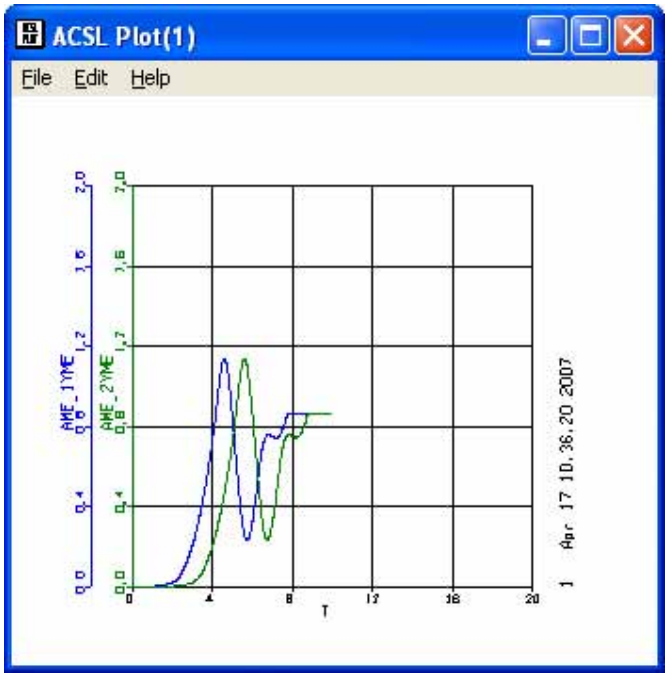
Hur de nödvändiga filerna länkas in i ACSL/GM beskrivs även i kapitel 0.

Robotarna initieras via subrutinanropen AclabMngrInitF_LvRb resp. AclabMngrInitF_LvRb2, som skickar över initialparametrarna path, t, timelimit, cint, xte0, yte0, zte0 och vtgt. Path och t skickas alltid med och motsvarar sökväg till modellen samt aktuell tid i huvudmodellen. Sökvägen sätts automatiskt, om filen automatgeneras i Make Config, men den är hårdkodad och måste uppdateras om sökvägen ändras. Övriga parametrar är simuleringsparametrar och startvärden för roboten. Startvärdena för robotarna måste sättas manuellt av användaren och detta görs via konstanter som visat i Figur 12. Samma subrutinanrop kan även returnera initialdata från submodellen. I exemplet returneras variabeln phi0. Andra gången gsl-filen exekveras kommer sedan submodellens runtime-funktioner, AclabMngrRunF_LvRb och AclabMngrRunF_LvRb2, att anropas. Dessa skickar över parametern t1 till respektive robot och även dessa sätts i exemplet via konstanter i huvudmodellen. Parametrar som alltid skickas i runtime är path, t och minstep, där minstep är minsta tid som respektive submodell tillåts exekvera innan den returnerar (denna sätts till ett defaultvärde på 0.0001 sekund om gsl-filen automatgenereras av Make Config). Parametrar som returneras från respektive robot är bl.a. robotens position och eulervinklar.

Anropet till runtime-funktionerna upprepas fram till huvudmodellens terminalsektion varvid submodellerna anropas en sista gång via funktionerna AclabMngrTermF_LvRb och AclabMngrTermF_LvRb2. Om submodellerna inte exekverat klart i detta skede kommer de att

avslutas av ACLAB. Terminal-funktionen tar inga inparametrar. I exemplet returneras bomavståndet från robotmodellerna.

Figur 14 visar robotarnas y-positioner plottat mot tiden:

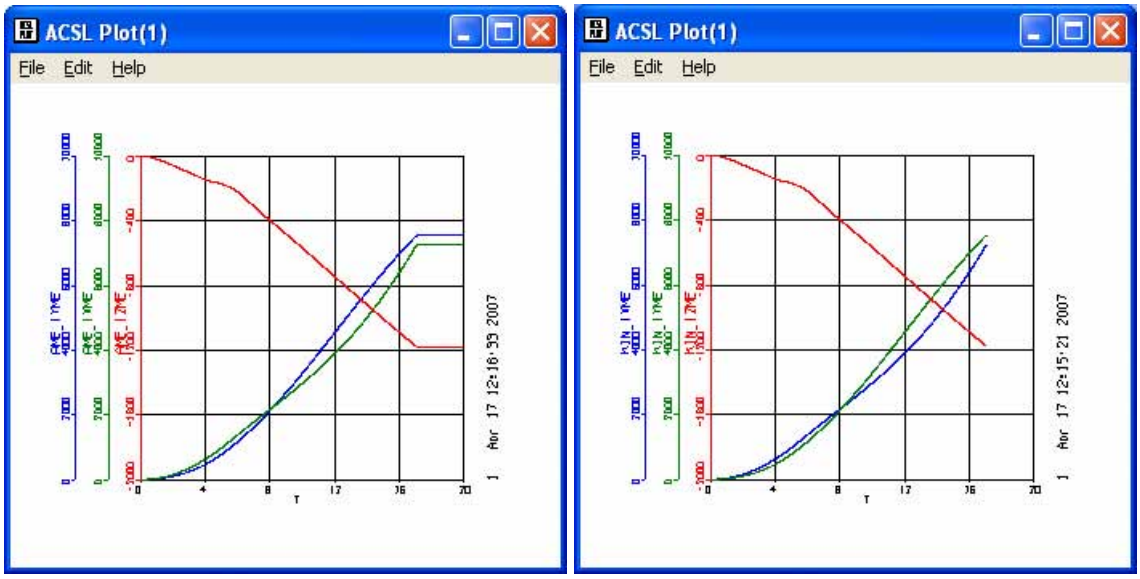


Figur 14 Robotarnas y-position

Som figuren visar startar första roboten vid tiden t=1 sekund och den andra en sekund senare. Robotarna flyger sedan fram till positionen där målet, som är i princip stillastående, befinner sig. Robotarna hinner i exemplet fram till målet innan huvudmodellen terminerat. Slutvärdena för roboten bibehålls därför till simuleringens slut. Returnerat bomavstånd kan i huvudmodellen läsas av till ca 7.2 meter för båda robotarna.

Figur 15 visar ett annat test där det ena målet, LvRb2, avfyras vid tiden t=0 från positionen (x,y,z) = (8000,5000,-1000) med hastigheten 300m/s. Till höger i figuren visas samma scenario kört i den separata modellen, LvRb2.gm, utan inblandning av ACLAB Framework. Figuren visar robotposition i x, y och z. Som figuren visar överenstämmer resultaten från simuleringarna bra. Även bomavståndet blir identiskt mellan simuleringarna.

Samma simulering gjordes även från Matlab/Simulink i exemplet i kapitel 4.1.



Figur 15 **En jämförelse mellan robotposition för LvRb körd från MngrTest.gm samt från LvRb.gm**

Trots att flera instanser av samma robot skall skapas måste anrop göras till olika modeller med olika namn på de körbara prx- och prj-filerna, vilket beror på en begränsning i ACSL. De måste även ligga i olika kataloger. I exemplet har två snarlika ACSL-block skapats för att sköta kommunikationen med de två olika modellerna och modellerna är kopierade i två olika kataloger på hårdisken (det som skiljer mellan ACSL-blocken är sökvägen till modellerna, som skickas via variabeln path, samt namnen på subrutinanropen). På samma sätt har två olika Fortranfiler skapats, en för varje modell. Detta är det enklaste och det rekommenderade sättet att skapa flera instanser av samma simuleringsmodell då huvudmodellen är en ACSL/GM-modell.

I exemplen ovan bibehålls slutvärdet från submodellen till huvudmodellen terminerats. Om submodellens terminering även skall innebära att huvudmodellen termineras måste detta lösas av användaren. Lämpligt är att skicka en flagga från submodellen som returvärde. Huvudmodellen kan sedan testa på denna flagga varje gången submodellen returnerar värden för att avgöra om simuleringen skall termineras.

5 Begränsningar / Fortsatt arbete

Begränsningar:

- Då submodellen kopplas upp mot en ACSL-modell krävs att båda modellerna är kompilerade med dubbel precision. Detta innebär även att parametrarna i *AclabConfig.txt* har tillhörande typen Double (ej Real) och att parametrar i Fortranfilen är har typen double precision (ej Real).

- En matris som skickas genom ACLAB kommer inte att behålla sina dimensioner utan levereras alltid som en vektor av längden $\text{rader}_{\text{urspr matris}}$*$\text{kolumner}_{\text{urspr matris}}$, se kapitel 0.

- I ACLAB Framework 3.0 initieras submodellerna först när dessa anropas från huvudmodellen, vilket kan vara en stund efter att huvudmodellen initieras. Det kan tänkas att huvudmodellens initialberäkningar i en applikation beror av submodellens initialberäkningar. Den nödvändiga initiala informationen från submodellerna fås ej.

- Sökvägen till submodellen är i huvudmodellens gsl-fil hårdkodad om MakeConfig har använts för att automatgenerera filen. Detta beror på att MakeConfig känner till sökvägen till submodellen men inte till huvudmodellen. Om huvudmodell och submodell flyttas till en annan plats måste därför sökvägen uppdateras manuellt i gsl-filen.

Förslag på fortsatt arbete:

- Hantera både dubbel och enkel precision.

- Utökat stöd för matrishantering. Matriser bör ha samma dimensioner i submodell och huvudmodell. Det bör således skickas med information om hur vektorn ska omformas till matris.

- Införa en option huruvida submodellerna ska initieras i samma skede som huvudmodellen initieras.

- Prestanda på ACLAB Framework bör testas. Hur förändras kompileringstiden och simuleringstiden samt finns det avvikelser i resultat jämfört med att köra en submodell separat?

- Införa möjligheten att peka ut huvudmodellen i MakeConfig för att möjliggöra relativa sökvägar vid skapandet av gsl-filen.

Referenser

[ACSL] ACSL (Advanced Continuous Simulation Language) for Windows, Version 11, 2000m, AEgis Technologies Group Inc, USA

[TAAKII] Verona M m.fl., – ”Telekrigaspekter vid Asymmetrisk Krigföring vid Internationella Insatser”, FOI-RH--0467--SE, Nov. 2005

[VISUAL] Visual Studio .NET 2003, Microsoft Corporation, USA, 2003

[AFE] Eckerland Johnny. FOI-R--00-1254--SE, maj 2004, “AFE 3.3 Reference Manual”

[BATCHMAN] Verona Mattias, FOI-R--1991--SE, maj 2006, “BatchMan 1.7 Reference Manual”

[MFC] Prosise Jeff, “Programming Windows with MFC - Second Edition”, 1999

Bilaga 1 ACLAB

Exporterade funktioner i ACLAB

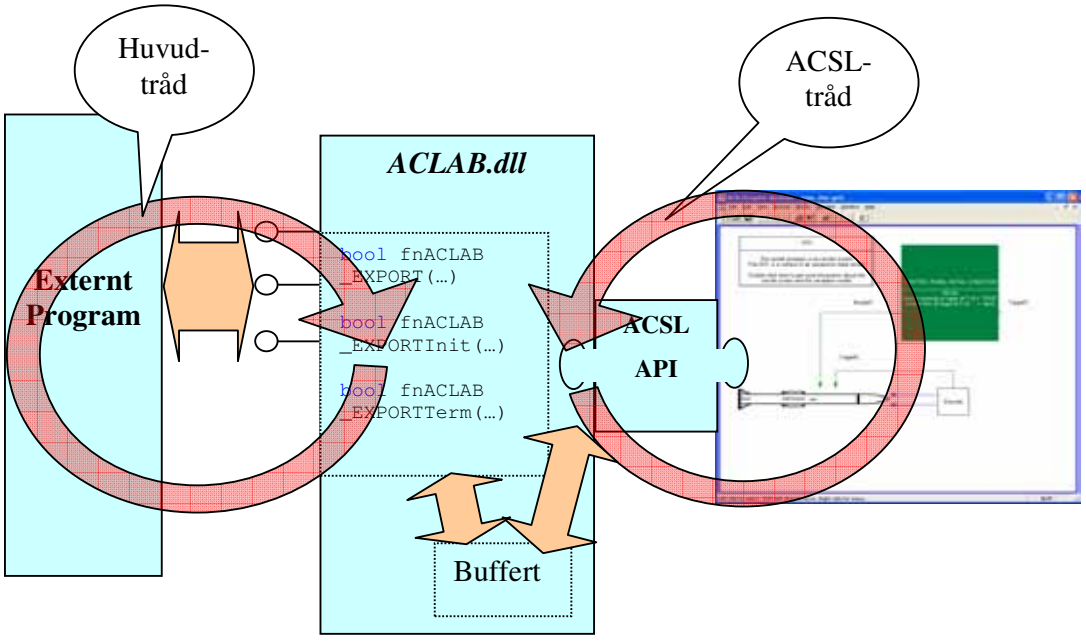
Nedan angiven programkod anger syntax för de tre exporterade funktionerna i ACLAB. Funktionerna handhar i tur och ordning initiering, löpande datautbyte samt terminering av submodell.

```
__declspec(dllexport)
        bool fnACLAB_EXPORTInit(
        char            sACSLLibAndName[_MAX_PATH],//Path to the ACSL-model
        double          dExternTime,   //External time corresponding to t=0 in the ACSL-model
        char*           pcInitialInputDoubleNicknames,
        int             nNumInitialInputDoublePar,    // Number of initial doubles
        char*           pcInitialOutputDoubleNicknames,
        int             nNumInitialOutputDoublePar,   // Number of initial output doubles
        char*           pcInputDoubleNicknames,
        int             nNumInputDoublePar,           // Number of input doubles
        char*           pcOutputDoubleNicknames,
        int             nNumOutputDoublePar,          // Number of output doubles
        char*           pcTerminalDoubleNicknames,
        int             nNumTerminalDoublePar,        // Number of terminal doubles
        double*         pdInitialInputDouble,         // Initial Input values
        double*         pdInitialOutputDouble         // Initial Output values
        );

__declspec(dllexport)
        bool fnACLAB_EXPORT(
        double          dExternTime,          //Time the ACSL-model are allowed to proceed to
        double          dMinASCLStep,         //Step limit to proceed the ACSL-model
        double*         pdInputDouble,        //Order according to pcInputDoubleNicknames
        double*         pdOutputDouble        //Order according to pcOutputDoubleNicknames
        );

__declspec(dllexport)
        bool fnACLAB_EXPORTTerm(
        int nTerm,                            //=1 -> Force ACSL Model to stop
        double*         pdTerminalDouble      //Order according to pcTerminalDoubleNicknames
```

Programstruktur

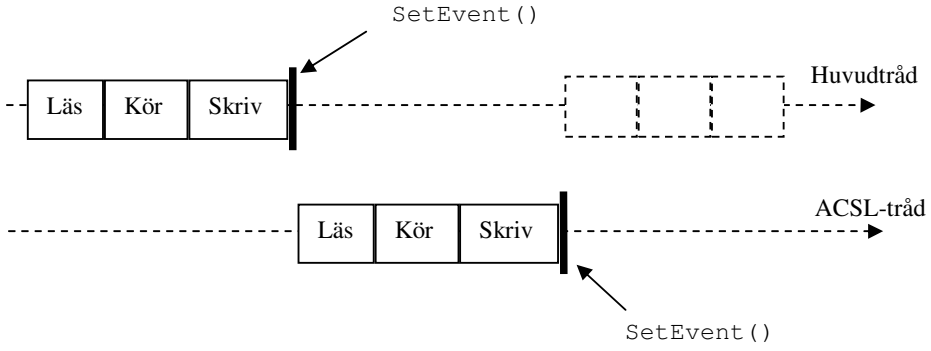
Kommunikationen mellan ett externt program och en ACSL-modell kräver att processen delas upp i två trådar, se Figur 16. Orsaken till detta är att det till ACSL medföljande API'et saknas en stegfunktion med vilken man kan stega ett steg i taget. Under exekvering sker inget arbete parallellt i dessa trådar utan en synkroniseringsstruktur har implementerats i ACLAB så att endast en av trådarna kan arbeta åt gången.

All kommunikation mellan de båda trådarna sker via en databuffert. Detta är en klass vars medlemsvariabler mellanlagrar all data som ska utbytas mellan de två trådarna. Förutom dessa så har bufferten tre medlemsvariabler av typen händelser (MFC klass CEvent). Syftet med dessa är att synkronisera kommunikationen mellan de båda trådarna.



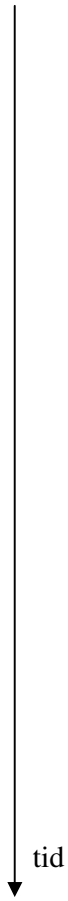
Figur 16 Bruna pilar anger datautbyte. Röda pilar markerar processtrådar.

Synkroniseringen görs genom att den tråd som ska lägga sig i viloläge talar om detta genom att meddela en händelse. Därefter lägger sig denna tråd i vila och förbrukar nästan ingen processorkraft förrän den händelse inträffar som tråden väntar på. Medan den ena tråden vilar flyttas processorkraften över till den andra tråden som på motsvarande sätt arbetar sig framåt i tiden varpå den uppdaterar bufferten, meddelar en händelse för att därpå lägga sig i viloläge. Synkroniseringsproceduren visas förenklat i Figur 17 och Tabell 1.



Figur 17 Huvudtråden och ACSL-tråden jobbar växelvis. Huvudtråden läser data från buffertklassen, kör ett tidssteg, uppdaterar buffertklassen och släpper sedan den väntande acsl-tråden via kommandot SetEvent().

Huvudtråd	**Händelse 1:** Huvudtråd har uppdaterat data till bufferten	**Händelse 2:** ACSL-tråd har uppdaterat data till bufferten	ACSL-tråd
	0	0	
Huvudtråd arbetar och uppdaterar bufferten.			
	1	0	
Huvudtråd väntar på att Händelse 2 ska inträffa.			Händelse 1 har inträffat. Gå vidare! Börja med att nollställa händelse 1!
	0	0	
			Kör ACSL-modellen så långt som tillåts. Uppdatera bufferten och meddela att detta gjorts.
	0	1	
Händelse 2 har inträffat. Gå vidare. Börja med att nollställa händelse 2.			Vänta på att händelse 1 inträffar.
Gå till början av tabellen!			



Tabell 1 Händelseschema för trådsynkronisering. En etta anger att händelsen har inträffat. Detta har meddelats med kommandot SetEvent(). En nolla anger att händelsen har nollställts. Detta görs med kommandot ResetEvent().

Implementationen av synkroniseringen görs för huvudtråden i funktionskroppen i de tre exporterade funktionerna, och för ACSL-tråden i `ModelCommunicationInterval` vilken är den funktion som anropas av ACSL-API:ts callback-funktion.

För att meddela händelsen att huvudtråden uppdaterat bufferten görs anropet:

```
CDataBuffert::theDataBuffert->m_peMainHasUpdatedItsData->SetEvent();
```

För att därpå lägga sig i vila krävs att ett singlelock objekt skapas som knyts till händelsen att ACSL-tråden har uppdaterat bufferten:

```
CSingleLock singleLock(CDataBuffert::theDataBuffert->
m_peACSLHasUpdatedItsData);
```

Därpå anropas:

```
singleLock.Lock();
```

Detta innebär att tråden kommer ligga här tills det att den lyckats med att låsa singleLock-objektet, vilket inträffar när ACSL-tråden gjort anropet:

```
CDataBuffert::theDataBuffert->m_peACSLHasUpdatedItsData->SetEvent();
```

Huvudtråden kan nu gå vidare genom att först göra anropet:

```
CDataBuffert::theDataBuffert->m_peACSLHasUpdatedItsData->ResetEvent();
```

ACSL-trådens händelseförlopp görs i analogi med huvudtrådens.

Säker trådavslutning

Eftersom en tråd inte har en destruktor (eng. destructor), som är fallet då en instans av en klass skall termineras, så har funktionen `WaitForSingleObject` valts att användas för att vänta på att en tråd ska avslutas. Funktionen anropas med trådens handtag (eng. handle) som inargument och returnerar när tråden avslutats. Ett problem som kan uppstå är att en tråd som redan avslutats anropas vilket kan få till följd att funktionen fastnar. Win32-funktionen::`DuplicateHandle` har använts för att undvika detta. Tråden skapas i ”suspended state”. Sedan skapas ett `DuplicateHandle`, vilket är en kopia på trådens handtag. Tråden släpps därefter med `ResumeThread`. `WaitForSingleObject` anropas på kopian av handtaget. Handtaget måste slutligen avslutas med ::`CloseHandle` (då detta bara görs automatiskt på originalhandtaget) [MFC].

Bilaga 2 ACLAB Manager

Exporterade funktioner i ACLAB Manager

Nedan angiven programkod anger syntax för de tre exporterade funktionerna i ACLAB Manager. Funktionerna handhar i tur och ordning initiering, löpande datautbyte samt terminering av submodell.

```
__declspec(dllexport)
void __stdcall  AclabMngrEntryInit(
       char     sACSLLibAndName[_MAX_PATH],  //Path to the ACSL-model
       double  dExternTime,                  //External time corresponding to t=0 in the ACSL-model
       double* pdInitialInputDouble,         //Input initial values
       double* pdInitialOutputDouble         //Output initial values
       );

__declspec(dllexport)
void __stdcall AclabMngrEntryRun(
       char sACSLLibAndName[_MAX_PATH],      //Path to the ACSL-model
       double  dExternTime,                  //Current external time. The submodel won't proceed
                                             //  past corresponding local time
       double  dMinACSLStep,                 //Submodel will wait to proceed if commanded
                                             //  time step (from dExternTime) is lower than this
                                             //  value
       double* pdInputDouble,                //Input values
       double* pdOutputDouble                //Output values
       );

__declspec(dllexport)
void __stdcall AclabMngrEntryTerm(
       char sACSLLibAndName[_MAX_PATH],      //Path to the ACSL-model
       double* pdTerminalDouble              //Terminal Values
       );
```

Bilaga 3 Make Config

Nedan beskrivs de filer som kan automatgenereras av Make Config samt hur de länkas in i modellen.

Konfigurationsfil skapad av Make Config

Ett exempel på en konfigurationsfil automatgenererad av Make Config visas nedan:

```
! LvRb2
! double precision
INITIALINPUT
DOUBLE TIMELIMIT    timelimit    1
DOUBLE CINT   cint   1
DOUBLE INIT_1XTE0   xte0   1
DOUBLE INIT_1YTE0   yte0   1
DOUBLE INIT_1ZTE0   zte0   1
DOUBLE INIT_1VTGT   vtgt   1

INITIALOUTPUT
DOUBLE KIN_1PHI0    phi0   1

INPUT
DOUBLE ACLAB_1T1    t1     1

OUTPUT
DOUBLE KIN_1XME     xme    1
DOUBLE KIN_1YME     yme    1
DOUBLE KIN_1ZME     zme    1
DOUBLE KIN_1PSI     psi    1
DOUBLE KIN_1THE     the    1
INTEGER      TGT_1XTI     RTELOG 3

TERMINAL
Double MISS_1BOMDISTmiss   1

END
```

Överst visas modellnamnet, LvRb2 i exemplet, samt den precision som används i modellen, *real* för enkel precision respektive *double precision* för dubbel precision[4]. Modellnamnet lagras i config-filen för att användas till namngivning av gsl-fil samt Fortanfil (och funktionsanropen i Fortranfilen) om även dessa filer automatgenereras av Make Config. Om en modell är laddad i Make Config är det alltid dess namn som också används vid namngivning av gsl- och Fortranfilerna. Om ingen modell är laddad används det namn som står överst i *AclabConfig.txt* för detta. Det krävs dock att modellen laddas för att sökvägen till denna skrivs in till gsl-filen. Modellen måste också vara laddad för att tillägg av parametrar skall kunna göras i konfigurationsfilen. *Det är därför rekommenderat att alltid ladda modellen då en konfigurationsfil öppnas.*

Alla parametrar listas sedan för de olika kategorierna: *Initial Input, Initial Output, Input, Output* respektive *Terminal*. Konfigurationsfilen skall läggas i en mapp som ska heta ACLAB och som ska ligga i modellkatalogen för submodellen som skall anropas. D.v.s. om modellens körbara prx-fil ligger under ’`C:\mymodel`’ skall konfigurationsfilen läggas under ’`C:\mymodel\ACLAB`’.

Fortranfil skapad av Make Config

Ett exempel på en Fortranfil, med subrutiner, automatgenererad av Make Config visas nedan:

[4] I version 3.0 av ACLAB Framework ska double användas,

```
*$pragma aux AclabMngrEntryInit "_AclabMngrEntryInit@20" parm \
C (value,\
C value,\
C reference,\
C reference)\
C routine []
      subroutine AclabMngrInitF_LvRb2(
     *       path,
     *       t,
     *       pdInitialInput,
     *       pdInitialOutput
     *       )
      CHARACTER*260 path
      double precision t
      double precision pdInitialInput(6)
      double precision pdInitialOutput(1)
      call AclabMngrEntryInit(
     *       path,
     *       t,
     *       pdInitialInput,
     *       pdInitialOutput
     *       )
      end

*$pragma aux AclabMngrEntryRun "_AclabMngrEntryRun@28" parm \
C (value,\
C value,\
C value,\
C reference,\
C reference)\
C routine []
      subroutine AclabMngrRunF_LvRb2(
     *       path,
     *       t,
     *       minstep,
     *       pdInput,
     *       pdOutput
     *       )
      CHARACTER*260 path
      double precision t
      double precision minstep
      double precision pdInput(1)
      double precision pdOutput(8)
      call AclabMngrEntryRun(
     *       path,
     *       t,
     *       MinStep,
     *       pdInput,
     *       pdOutput
     *       )
      end

*$pragma aux AclabMngrEntryTerm "_AclabMngrEntryTerm@8" parm \
C (value,\
C reference)\
C routine []
      subroutine AclabMngrTermF_LvRb2(
     *       path,
     *       pdTerm
     *       )
      CHARACTER*260 path
      double precision pdTerm(1)
      call AclabMngrEntryTerm(
     *       path,
     *       pdTerm
     *       )
      end
```

Filen innehåller tre subrutiner som skall anropas från huvudmodellen: AclabMngrInitF_LvRb2, AclabMngrRunF_LvRb2 respektive AclabMngrTermF_LvRb2, där LvRb2 är modellnamnet. Dessa tre funktioner är en initial-funktion, som sätter initialparametrar i submodellen samt returnerar önskade initialparametrar, en runtime-funktion, som sätter respektive returnerar parametrar, samt en terminalfunktion, som returnerar värden från submodellen under terminalsektionen. Varje funktion gör sedan ett anrop till motsvarande funktion i ACLAB Manager. Parametrar som skickas till ACLAB Manager är: path, t, minstep samt vektorer med in- respektive utparametrar. Path är sökvägen till modellen, t är tid i huvudmodellen och minstep är minsta tillåtna tid som submodellen får framskrida innan den returnerar (minstep sätts per default till 0.0001 då filen automatgenererats i Make Config). Vektorerna innehåller värden av antingen dubbel eller enkel precision beroende på precisionen använd i modellen. Värdena är parametervärden för de parametrar som är listade i konfigurationsfilen.

gsl-fil skapad av Make Config

Ett exempel på en gsl-fil automatgenererad av Make Config visas nedan:

```
!This file was created by MakeConfig: Mon May 21 16:41:54 2007


INITIAL

  CHARACTER*260 path
  path = 'C:\Projects\ACLAB_FrameWorks\ACLAB_3_0_FrameWork\ACSLModel\LvRb2\LvRb2.prj'
  minstep = 0.0001
  LOGICAL FirstTime
  FirstTime = .TRUE.
  DIMENSION pdInitialInput(6)
  DIMENSION pdInitialOutput(1)
  DIMENSION pdInput(1)
  DIMENSION pdOutput(8)
  DIMENSION pdTerm(1)
  !Initialize outdata
  phi0 = 0.0
  xme = 0.0
  yme = 0.0
  zme = 0.0
  psi = 0.0
  the = 0.0
  INTEGER RTELOG
  DIMENSION RTELOG(3)
  DO loop_RTELOG i_RTELOG= 1,3
    RTELOG(i_RTELOG) = 0
  loop_RTELOG..CONTINUE
  miss = 0.0
  !END Initialize outdata

END !Initial

Procedural(pdInitialInput,pdInput,phi0,xme,yme,zme,psi,the,RTELOG=timelimit,cint,xte0,yt
e0,zte0,vtgt,t1,path,pdOutput,pdInitialOutput)

!Cast integers and logicals to doubles (or real)
!End Cast integers and logicals to doubles (or real)

IF(t.GE.LaunchTime) THEN
  IF(FirstTime) THEN
       pdInitialInput (1) = timelimit
       pdInitialInput (2) = cint
       pdInitialInput (3) = xte0
       pdInitialInput (4) = yte0
       pdInitialInput (5) = zte0
       pdInitialInput (6) = vtgt
       CALL AclabMngrInitF_LvRb2(path,t,pdInitialInput,pdInitialOutput)
       phi0 = pdInitialOutput(1)
       FirstTime = .FALSE.
  ELSE
```

```
      pdInput (1) = t1
      CALL AclabMngrRunF_LvRb2(path,t,minstep,pdInput,pdOutput)
      xme = pdOutput(1)
      yme = pdOutput(2)
      zme = pdOutput(3)
      psi = pdOutput(4)
      the = pdOutput(5)
      RTELOG(1) = int(pdOutput(6))
      RTELOG(2) = int(pdOutput(7))
      RTELOG(3) = int(pdOutput(8))
  END IF
END IF

!Cast back output to original types
!END Cast back output to original types

END !Procedural

TERMINAL

Procedural(miss=path,pdTerm)

  CALL AclabMngrTermF_LvRb2(path,pdTerm)
  miss = pdTerm(1)

END !Procedural


END !Terminal
```

Filen innehåller anrop till de subrutiner i Fortran-filen som finns beskriven ovan. En variabel skapas för varje inparameter vars värde skall sättas i submodellen. Dessa variabler måste matas med värden från huvudmodellen, vilket kräver manuella åtgärder av användaren. Värdena skickas sedan vidare via de vektorer som automatgenererats, d.v.s. pdInitialInput respektive pdInput. I exemplet ovan kommer t.ex. timelimit, cint, xte0, yte0, zte0 och vtgt att sättas initialt i submodellen. Dessa variabler måste därför ges värden från huvudmodellen. På samma sätt skapas en variabel för varje parametervärde som kommer att returneras från submodellen via vektorerna pdInitialOutput, pdOutput och pdTerm. Initialt kommer phi0 att returneras från submodellen. För att submodellen skall kunna startas i en tid skild från tiden t=0 i huvudmodellen, görs anropet till submodellens initialsektion vid tiden t större än Launchtime i huvudmodellen. Anropet görs alltså inte i huvudmodellens initialsektion. Detta är en begränsning som varit nödvändig för att möjliggöra exekvering av en submodell vid godtycklig tid från huvudmodellen. Andra gången gsl-filen exekveras kommer submodellens runtime-funktion att anropas. Detta upprepas fram till huvudmodellens terminalsektion varvid submodellen anropas en sista gång. Om submodellen inte exekverat klart i detta skede kommer den att avslutas av ACLAB och eventuella slutvärden returneras.

I den automatgenererade filen kommer alla värden att typkonverteras om till antingen real eller double precision, beroende på vilken precision som är vald i modellen. I exemplet ovan är utvaribeln RTELOG en array av integers och modellen har dubbel precision. För att fylla RTELOG med data från utdatavektorn krävs således att dessa data typkonverteras om till integers.