*Forts nästa sida*

# SDL och UML för realtidsmjukvara

**Att utveckla realtidsmjukvara för embeddedsystem med SDL eller UML är till synes enkelt, men det finns problem och invändningar. Emmanuel Gaudin, grundare av PragmaDev, analyserar problemen från sin egen utgångspunkt och pekar på en mera pragmatisk väg med SDL-RT.**

SDL, Specification and Description Language, definierades av Internationella Teleunionen (ITU-T) för att beskriva telekomprotokoll. Erfarenheten har visat att de grundläggande principerna kan utvidgas till en rad andra områden, framför allt på grund av språkets grafiskt funktionella sätt att beskriva arkitektur och beteende i ett system.

Men när det handlar om att implementera och integrera kod för ett målsystem visar sig datatyperna och några av högnivåkoncepten i SDL ställa till problem för konstruktörerna. De tappar kontrollen över den slutliga målmaskinskoden.

Dessutom saknas en del realtidsaspekter, som semaforer och pekare i språket. Det har lett till att de flesta SDL-konstruktörer blandar C-kod inuti de grafiska SDL-representationerna. Och eftersom det inte funnits några verktyg för att stödja kombinationen av SDL och C tvingades de skriva sin egen kodgenerator för C utifrån den här blandningen av SDL och C.

Allt detta har hållit tillbaka användningen av SDL på en bredare marknad. SDL-RT är ett försök att lösa problemen, samtidigt som det också inkluderar en del UML-diagram.

## SDL FÖR REALTID

Realtids embeddedmjukvara baseras oftast på realtidsoperativsystem, där det grundläggande strukturella elementet är ett ”task”. Flera tasks kan exekvera samtidigt för att utföra en grundfunktion. Grundfunktioner läggs samman för att realisera mera komplexa funktioner, som i sin tur läggs samman upp till högsta applikationsnivån. SDL-arkitekturen, med tasks i funktionella block, som i sin tur kan samlas i block på högre nivå, har visat sig vara ett bra sätt att grafiskt beskriva ett realtidssystem.

Arkitekturen kompletteras sedan med en beskrivning av gränssnitten mellan de olika funktionerna i systemet. Ett gränssnitt definieras av:

- Ett kommunikationsformat (exchange format) baserat på strukturerade data.
- Ett scenario som beskriver sekvensen av kommunikationer.

SDL-signaler har parametrar baserade på abstrakta datatyper (ADT), som tillåter en komplett beskrivning av det statiska gränssnittet. Den dynamiska aspekten av informationen som utväxlas hanteras grafiskt av MSC, Message Sequence Charts. MSC används både för att beskriva utbyte av information inom systemet och mellan systemet och yttre moduler, till exempel drivrutiner.

Ett realtidssystem baseras på oberoende tasks som körs parallellt. Därför är det viktigt att inte slösa bort CPU-tid när en task inte har något att göra. Det har lett till att de flesta realtidsapplikationer baseras på finita tillståndsmaskiner. Den grundläggande principen är där att vänta på att ett RTOS-systemobjekt avslutar sitt arbete med till exempel en meddelandekö. SDL passar bra för att beskriva beteendet i dessa finita tillståndsmaskiner grafiskt.

SDL är ett objektorienterat grafiskt språk (sedan 1992 års version). Det gör det lätt att bygga upp bibliotek av mjukvarukomponenter som sedan kan specialiseras. SDL språket innehåller dock några nackdelar vid konstruktion av realtidssystem.

## ABSTRAKTA DATATYPER PASSAR INTE

När det är dags att konstruera visar sig de abstrakta datatyperna i SDL (ADT) ställa till problem av flera skäl:

- Syntaxen är till för att specificera protokoll, inte konstruera dem. Mjukvarukonstruktörer får problem eftersom de inte har tillgång till den precision som de är vana vid när de använder C.
- Det finns inga SDL-korskompilatorer utan SDL-koden måste översättas till C- eller C++-kod för att kunna implementeras på ett målsystem.
- De abstrakta datatyperna i SDL är baserade på koncept som inte direkt kan översättas till C eller C++. Detta innebär att specifika operatorer måste genereras och det gör den genererade C/C++-koden oläsbar.
- Integration av befintlig kod är svårt eftersom datatyperna måste översättas mellan SDL och C/C++

## SDL OCH RTOS

När kod som genererats från ett SDL-system skall integreras med ett RTOS måste man lösa en del semantiska aspekter:

- SDLs sätt att hantera prioritet baseras på meddelanden, medan ett RTOS hanterar prioritet utifrån tasks.
- SDL-semantiken förutsätter att en del transaktioner (automaton transaction) inte kan avbrytas. Ett RTOS kan avbryta exekveringen när som helst, t ex om ett systemanrop med högre prioritet kommer in.

## SEMAFORER

Semaforer finns inte definierade i SDL, samtidigt som det är en av de vanligaste och mest klassiska synkroniseringsmekanismerna i ett realtidssystem. Orsaken till att semaforer inte finns är att SDL definierades för att specificera protokoll för att kommunicera med yttre enheter. Där fungerar inte semaforer.

## ÄNDRAD STANDARD

Av de tidigare nämnda skälen blev det svårt, tidskrävande och dyrt att implementera realtidssystem beskrivna i SDL. Det krävdes mycket utbildning och ofta många konsulttimmar för att få ett fungerande målsystem. I många fall förlorade man fördelarna med att använda ett grafiskt språk under integrationsfasen. SDL-verktygstillverkarna höll sig hårt till standarden, oavsett vilka tekniska problem man fann, vilket inte gjorde situationen bättre. Detta innebar att SDL-användarna i allt större utsträckning började skriva C-kod inuti SDL-diagrammen. De bröt upp SDL-semantiken för att få systemen att passa i den tekniska realiteten. Men eftersom en blandning av SDL och C inte stöddes av verktygstillverkarna ledde detta till att användarna utvecklade egna verktygsflöden. resultatet är att alla stora telekomföretag har byggt sin egen interna C-kodgenerator utifrån en mix av SDL och C. Några exempel på företag är Alcatel, Nokia, Nortel, EADS och Sagem.

SDL-RT är en uppgradering av SDL som formaliserar industrins vana att blanda C-kod med grafiska SDL-representationer. För att göra det möjligt att använda SDL-RT till alla typer av realtidstillämpningar adderar SDL-RT även grafiska symboler för att hantera semaforer.

## UML-VÅGEN

UML, Unified Modelling Language, är som namnet anger en kombination av flera olika typer av modelleringsdiagram. I första hand var tanken med UML att förenkla dokumentering och kommunicera modeller mellan alla involverade i ett mjukvaruprojekt. Sedan första utgåvan 1997 har UML, med hjälp av en kraftig marknadsföring, spritts till många olika områden.

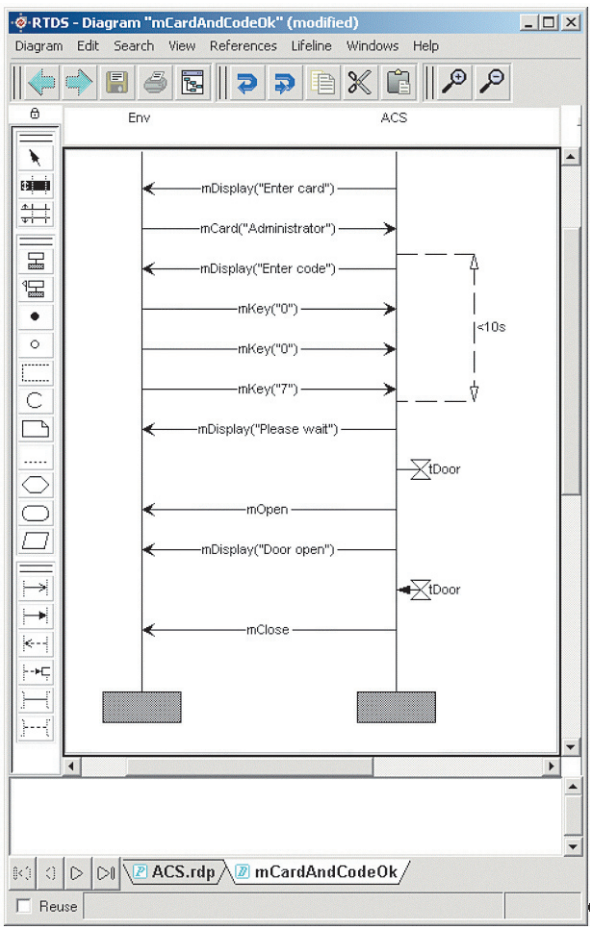
UML-språket karakteriseras av att det är helt objektorienterat. Varje element i vilket system som helst beskrivs i grunden som en klass. Denna mycket generella ansats tillåter en mycket hög abstraktionsnivå och språket kan användas för vad som helst. Å andra sidan är UML-språket inte dedikerat för någon speciell applikationsdomän och är inte precist nog för att beskriva applikationer i detalj. Oftast används språket i tidiga analys- och specifikationsfaser.

Den svaga länken mellan UML-diagram och konstruktionen leder till två återkommande problem:

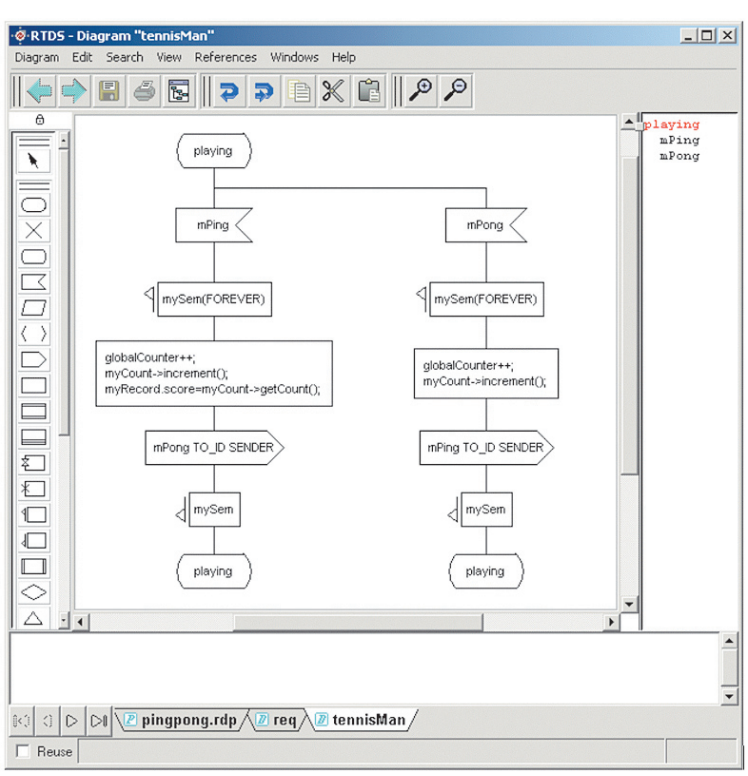
* Dålig synkronisering mellan UML-dokumentationen och den slutliga koden.

* Avsaknad av ett strukturellt ramverk för att driva utvecklingsprocessen.

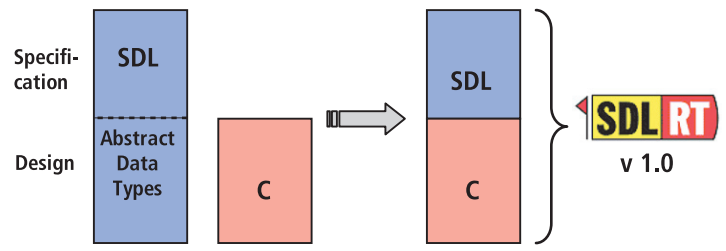
De här problemen kan även förklara framgångarna för UML.




*Exempel på en ”Message sequence Chart, MSC.*




*I SDL-RT finns grafiska sätt att representera sändning och mottagning av meddelanden och hantering av semaforer.*



## SDL och UML...

*forts från föregående sida*

Utvecklingsgrupperna kan fortfarande arbeta på det sätt de är vana och dokumentera oberoende av språket. Tyvärr är fördelarna med ett sådant arbetssätt ytterst begränsade och en del mycket stora företag har t ex insett nackdelarna och slutat att kräva UML-användning av sina konstruktionsgrupper.

### REALTID OCH UML

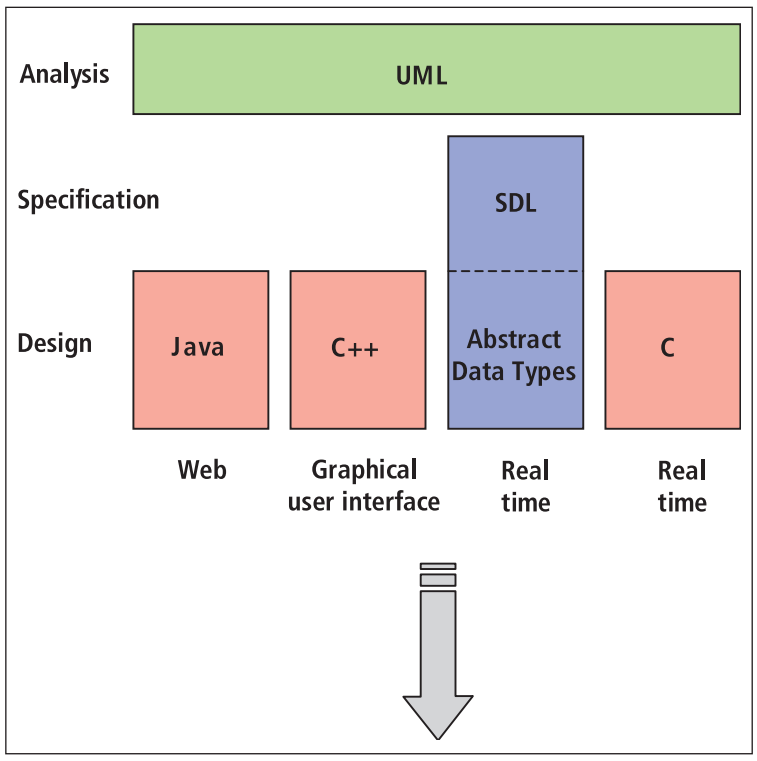
I ett realtidssystem kan egentligen bara klassdiagrammen, sekvensdiagrammen och tillståndsdiagrammen användas. Eftersom UML-verktygen i grunden är ritverktyg med mycket begränsade möjligheter till kodgenerering är de mycket billigare än SDL-verktyg. Också detta har hjälpt spridningen av UML.

Flera erfarenheter med UML-språket i stora utvecklingsgrupper ledde till katastrofala resultat. Verktygen genererade alldeles för mycket kod med för låga prestanda och inga möjligheter till debug på målsystemen. Erfarenheterna ledde till att de ansvariga för verktyg och metoder ifrågasatte om en fullt objektorienterad ansats verkligen är gångbar för ett realtidssystem där den traditionella metoden är funktionell. Till saken hör att framgångsrika objektorienterade projekt ofta har börjat med en funktionell ansats och sedan brutit ner den till komponenter organiserade i bibliotek för att förenkla återanvändning.

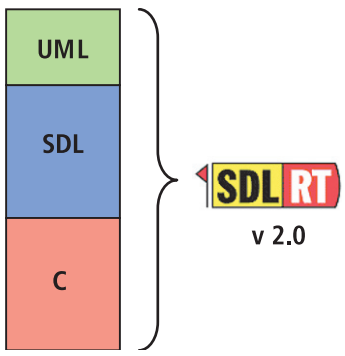
### UML 2.0

I UML 2.0 finns stöd för MDA, Model Driven Architecture, vilket gör det möjligt att definiera specifika profiler för varje applikationsdomän.

Version 2.0 av UML har integrerat existerande funktioner från SDL, som blockdiagram i de strukturella diagrammen och MSC i sekvensdiagrammen. Det är också möjligt att definiera domänspecifika profiler, men tyvärr finns ännu ingen slutlig realtidsprofil färdig. En del stora företag tvivlar på att en sådan profil någonsin kommer till stånd, bland annat på grund av intressekonflikter inom OMG.




*UML kan användas oberoende av applikationsdomän, men det krävs ytterligare ett språk för att beskriva modellen.*




*Version 2.0 av SDL-RT inkluderar relevanta UML-diagram i realtidsdomänen och garanterar att de olika representationerna är konsistenta.*

UML 2.0 finns tillgänglig på marknaden, men målet att förenkla portabilitet och läsbarhet uppfylldes knappast eftersom det finns så många proprietära profiler. De här profilerna är heller inte öppet tillgängliga och verktygen innehåller sällan en profileditor, utan användaren blir mer eller mindre låst till den UML-verktygstillverkare han väljer.

### TEKNSKA PROFETER

Vare sig programmeringsverktygen är baserade på SDL eller UML har många språkkonstruktörer i många fall haft en alltför högtravande attityd till sitt språk. De har lagt massor av energi på att demonstrera att just deras version representerar den enda sanningen. Problemet är att marknadsföringen fungerat så bra att vi fått en mängd tekniska fanatiker i industrin som bländats av dessa profeter. Tekniska realiteter har fått många användare att inse verkligheten. Användarna får betala ett högt pris i form av verktyg, konsulter och förlorad tid för att komma till insikt om att UML-standarden inte pasar särskilt bra för realtidssystem.

SDL-RT föreslår mera jordnära lösningar, baserade på vad som faktiskt används i industrin, med fokus på att generera exekverbar kod och integrera utvecklingsmiljön med den debugmiljö som RTOS-tillverkarna erbjuder. Standarden kombinerar på ett pragmatiskt sätt de användbara UML-diagrammen, de flesta SDL-diagrammen och klassiska programmeringsspråk som C och C++ i ett konsistent ramverk. SDL-RT-diagrammen sparas i öppna XML-format för att vara återanvändbara utan något beroende av speciella verktyg.

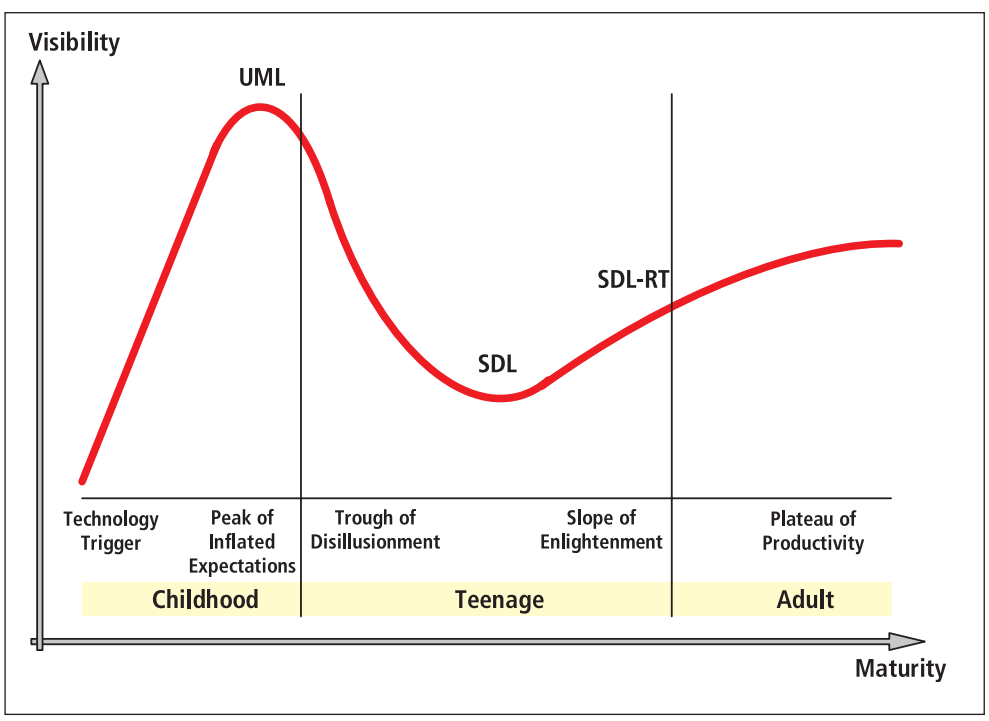
SDL-RT står för SDL för realtidssystem och är en ITU-T standard. ITU-T gör officiella releaser av nya SDL-versioner vart 4e år och kommer att ge SDL-RT en officiell stämpel i och med SDL-2008.

Målet med SDL-RT är att fylla behovet av en standard för grafiskt programmering av realtidssystem. Men till slut är det ändå verktygens användbarhet som skall övertyga slutanvändarna. Målet för en utvecklingsgrupp är ju inte att ansluta sig till en standard, som en del språkkonstruktörer tycks tro, utan att producera högkvalitativ återanvändbar programvara med höga prestanda på ett effektivt sätt.

EMMANUEL GAUDIN, PRAGMADEV

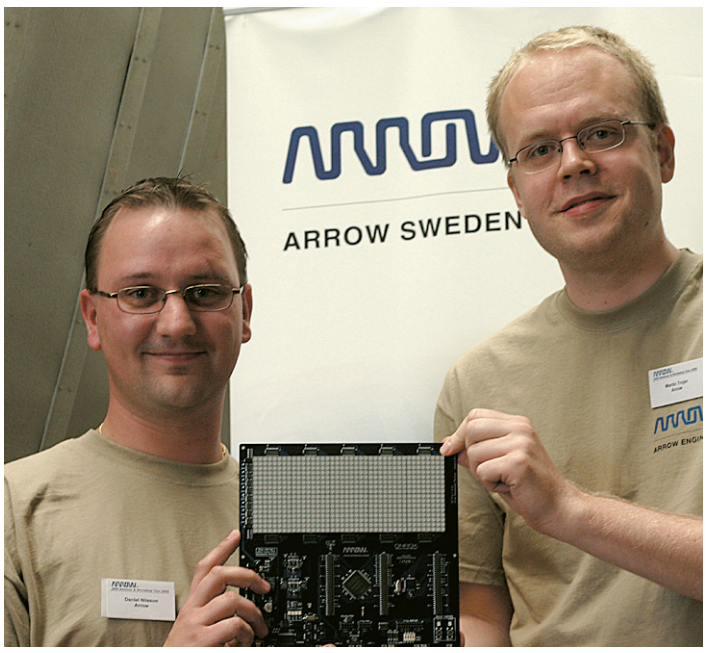
*Det finns mer all läsa om standarden SDL-RT på www.sdl-rt.org.*

*PragmaDev representeras i Norden av Embedded Wireless Engineering AB, www.ewe.nu.*




*Gartners "Hype Cycle" beskriver var ett språk befinner sig i förhållande till mognad och exponering. SDL-RT har fördelen av att ha mognaden från C och SDL, men också visibiliteten från UML.*

# ARM fortsätter att öka




*Daniel Nilsson och Martin Trojer har tagit fram ett prototypkort baserat på ARM. I den här versionen, med 640 flerfärgade lysdioder blev demoeffekten extra stor.*

**Intresset för ARM-processorn fortsätter att växa. Det märktes i somras när Arrow bjöd in till ARM-konferens i Göteborg och Stockholm. Normalt sett är det svårt att få konstruktörer att lämna sina arbetsplatser och ett tjugotal besökare klassas som OK. Men här kom över hundra personer i Göteborg och knappt hundra i Stockholm.**

Det intressanta med ARM är bland annat att samma arkitektur finns hos så många tillverkare och att ARM-processorerna finns paketerade på så många olika sätt. De olika leverantörerna har olika målgrupper och kombinerar ARM-kärnor med periferienheter för att passa så bra som möjligt in i respektive målgrupp.

Den andra stora fördelen är förstås att ARM-arkitekturen är så vanlig att en gigantisk bas av verktyg och programvara växt upp. Det är relativt lätt att hitta drivrutiner, oberoende av vilket operativsystem man väljer. Och det är lätt att hitta gratisverktyg och gratisoperativsystem för den som vill komma igång utan att ha en välfylld plånbok. Årets embeddedpris på studentsidan är ett bra exempel på det, baserad på en 100 MHz ARM-processor.

Allt fler tar idag steget direkt till en ARM-processor i stället för att försöka begränsa konstruktionen till en enkapseldator. Det är lättare att skriva koden om man har tillgång till en fullskalekompilator för C eller C++. Det är också lättare att lägga in ett komplett Linux- eller Windows CE-operativsystem om systemet skall kommunicera via standardgränssnitt eller ha grafiskt gränssnitt.

GÖTE FAGERFJÄLL