Comparing Asynchronous and Synchronous Approaches to Knowledge Processing

Anders Skoglund (andsk668)

May 16, 2011

1 Introduktion
- Syfte
- Verktyg och språk

2 Bakgrund
- Synkron och asynkron programmering
- DyKnow
- FlexDx
- SIGNAL

3 Implementation

4 Resultat

5 Slutsattser

Kortfattat

Syfte

Undersöka skillnaderna mellan synkron och asynkron programmering inom området kunskapsbearbetning.

Metod

Skapa en synkron implementation av feldiagnostiseringssystemet FlexDx i det synkrona programmeringspråket SIGNAL.

Detaljer

Två frågor att besvara:

1. Om SIGNAL kan användas för att implementera ett system för kunskapsbearbetning som är så pass komplext som FlexDx.
2. Om det finns begränsningar i SIGNAL eller den synkrona beräkningsmodellen som gör dem olämpliga att använda i praktiken.

Verktyg

- **`FlexDx:`** Ett feldiagnostiseringssystem
- **`DyKnow:`** Ett asynkront middleware för kunskapsbearbetning som tidigare använts för att implementera FlexDx
- **`SIGNAL:`** Ett synkront programmeringsspråk

Bakgrund

Mycket kortfattat.
Läs rapporten för detaljer.

Beräkningsmodeller

Två olika beräkningsmodeller:

- Asynkron
- Synkron

Asynkron

- Vad man använder i “normal” programmering
- Exekveringstiden är oberäknelig och påverkas av schemaläggning, CPU-användning, osv
- Dessa faktorer kan variera mellan körningar
- Vet ej i förväg hur lång tid något kommer att ta, vilket inte är bra för realtidssystem

Synkron

- Tid delas upp i diskreta steg
- Alla operationer görs ögonblickligen och samtidigt
- Programmeraren behöver inte tänka på tid
- Tidsmässigt deterministiskt och förutsägbart
- Plattformsoberoende

DyKnow

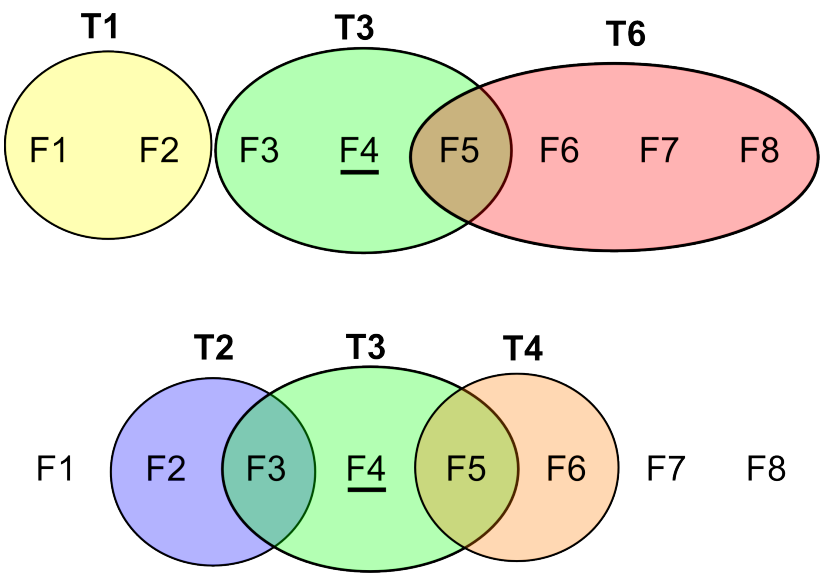
- Middleware för kunskapsbearbetning
- Lågnivå-data -> bearbetning -> abstrakt data
- Nätverk av asynkrona strömmar som kopplar samman beräkningsenheter
- Kraftfullt och flexibelt
- Kan läsa tidigare värden i strömmar från godtyckligt långt bak i tiden
- Kan strukturera om strömmar och noder under körning

FlexDx

- Ett strömbaserat feldiagnostiseringssystem
- Övervakar ett system för att upptäcka möjliga fel från sensordata
- Arbetar asynkront
- Tidigare implementerat med DyKnow
- Endast en del av alla tester körs på en gång
- Testval anpassas under körning

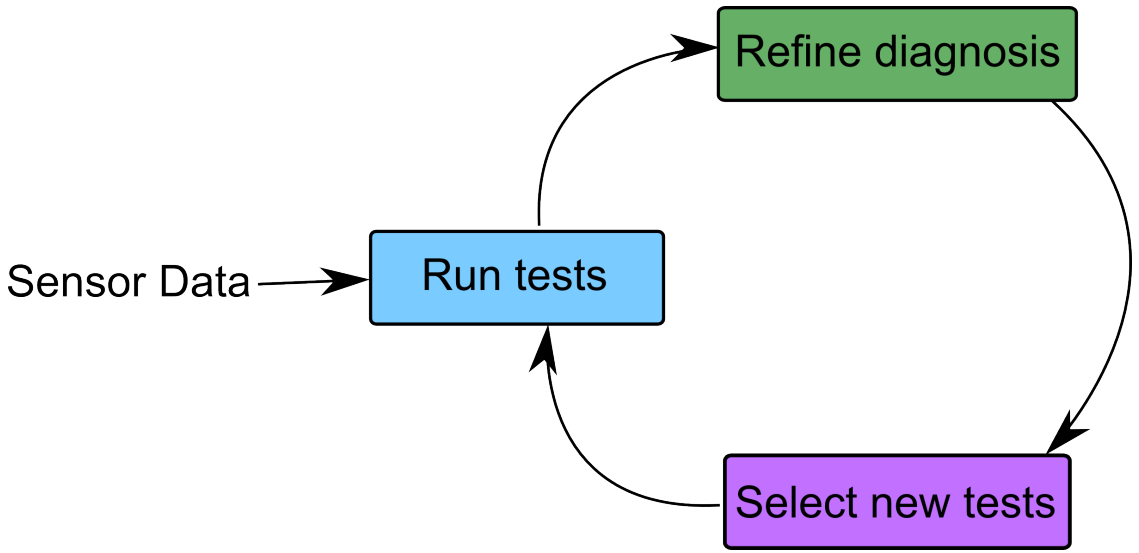
Tester

- Varje test kan triggas vid ett antal fel
- Börjar med en uppsättning tester som tillsammans kan upptäcka alla fel
- Förfinar diagnosen iterativt genom att välja nya tester

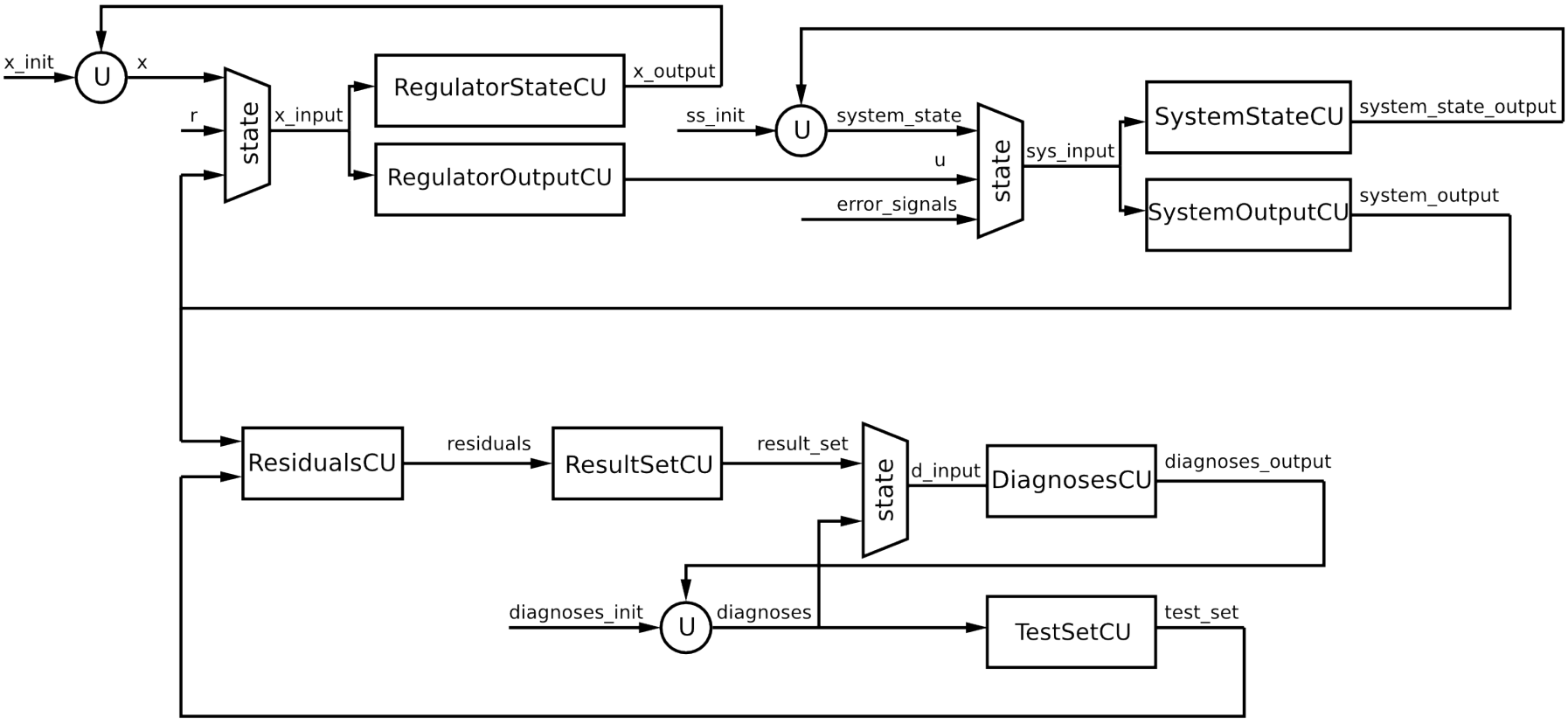


Algoritmen

1. Kör tester
2. Om ett test triggas ta fram en uppsättning möjliga fel
3. Välj nya tester som kan skilja mellan dessa fel
4. Fortsätt med punkt 1



Översikt


x_init
U
x
r
state
x_input
RegulatorStateCU
x_output
RegulatorOutputCU
ss_init
U
system_state
u
error_signals
state
sys_input
SystemStateCU
system_state_output
SystemOutputCU
system_output
ResidualsCU
residuals
ResultSetCU
result_set
state
d_input
DiagnosesCU
diagnoses_output
diagnoses_init
U
diagnoses
TestSetCU
test_set

Vad är SIGNAL

- Ett strömbaserat och synkront programmeringsspråk för realtidssystem
- Baserad på formell matematik och en mycket strikt kodningsstil
- Tidsmässigt deterministisk
- Mycket begränsat och strikt
- Tid abstraheras bort

Polychrony

- Kompilator för SIGNAL
- Kompilerar SIGNAL-kod till C,C++ eller Java
- Analyserar koden och alla klockor
- MYCKET strikt
- Om något kompilerar så är koden nästan undantagsvis korrekt

Begränsningar

SIGNAL har många begränsningar:

- Inga dynamiska arrayer
- Antalet iterationer i loopar måste vara konstant
- Nästan inget kan ändras under körning
- Inga sidoeffekter

Signaler

- Programmerar med signaler
- En signal är en typ av ström som vid en tidpunkt antingen håller ett värde eller är “frånvarande”
- Signaler fungerar som variabler
- Varje signal har en klocka som anger om den håller ett värde eller ej vid en specifik tidpunkt

Operatorer

Operatorer = relationer mellan signaler.
Vad man använder för att bygga ett program. `x := y + z`

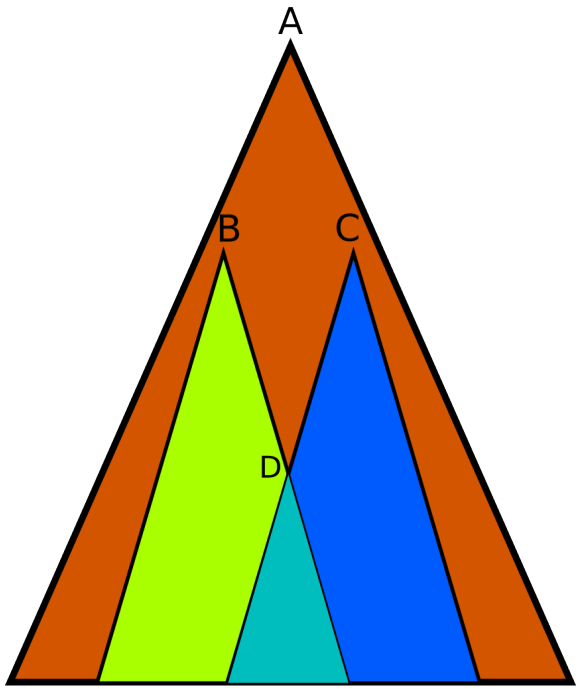
x:	5	11	17	23	29	...
y:	1	3	5	7	9	...
z:	4	8	12	16	20	...

Klockor

Kan ses som en mängd av tidpunkter då en signal har ett värde. Alla operatorer skapar relationer mellan klockor. `x := y + z`

x:	5	11	17	23	29	...
y:	1	3	5	7	9	...
z:	4	8	12	16	20	...

Klockor



Klockor

Kan ses som en mängd av tidpunkter då en signal har ett värde. Alla operatorer skapar relationer mellan klockor. `x := y + z`

x:	5	11	17	23	29	...
y:	1	3	5	7	9	...
z:	4	8	12	16	20	...

Mer operatorer

Finns två typer av operatorer.

- Monoclock operatorer
- Multiclock operatorer

```
x := var y
```

x:	·	1	·	·	3	3	·	·	·	5	6	6	...
y:	1	·	2	·	3	·	4	·	5	·	6	·	...

Klockoperatorer

Skapar mängdrelationer mellan klockor. Finns fyra olika:

- `^=` Ekvivalens
- `^+` Union
- `^*` Snitt
- `^-` Komplement

Oversampling

Ger en klocka som är “snabbare” än en annan klocka.

Exempel:

- En klocka Y ska ha 10 tidsinstanser mellan varje instans av klockan X

Kan användas för att skapa något likt loopar från imperativa språk.

Oversampling

```
1  process count_from_n =
2  ( ? integer n;
3    ! integer i;
4  )
5  (| cnt := n default (zcnt - 1)
6   | zcnt := cnt $ 1 init 0
7   | n ^= when (zcnt = 0)
8   | i := zcnt when zcnt > 0
9   |)
10 where
11     integer cnt, zcnt;
12 end;
```

n:	2	·	·	3	·	·	·	2	·	·	...
cnt:	2	1	0	3	2	1	0	2	1	0	...
zcnt:	0	2	1	0	3	2	1	0	2	1	...
i:	·	2	1	·	3	2	1	·	2	1	...

Externa processer

- Används precis som vanliga processer
- Skapar ett gränssnitt mellan SIGNAL och ett imperativt språk
- Upp till programmeraren att skriva de imperativa funktionerna
- Kan användas för att få in funktionalitet som SIGNAL inte har

Externa typer

- Fungerar precis som externa processer, men för datatyper
- Kompilatorn skapar kod som använder typerna
- Det är programmerarens ansvar att skriva dem
- Kan användas för att få in datatyper som SIGNAL inte stödjer

GALS

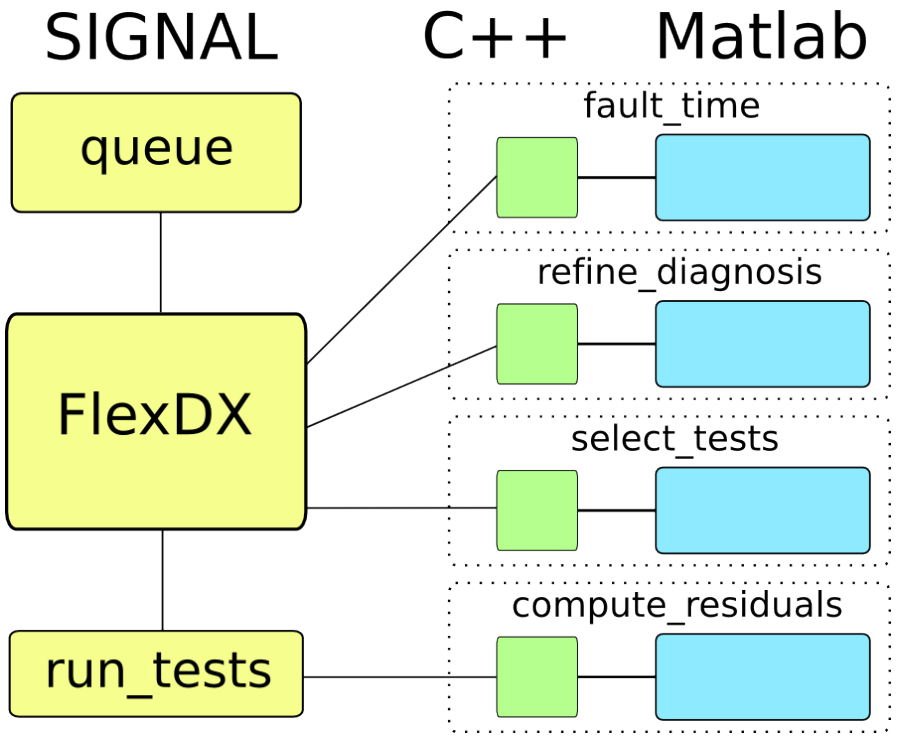
- Globaly Asynchronous, Localy Synchronous
- SIGNAL kan “omsynkronisera” asynkrona signaler
- Kan användas när man har synkrona komponenter som kommunicerar över ett asynkront nätverk

Mer om detta senare.

Implementation

Tre delar:

- SIGNAL
- C++
- MATLAB



Kö

Action	Argument	Result	Stored signals				
IN	2	-	-	-	-	[2]	
OUT	-	2	-	-	-	2	[]
IN	7	-	-	-	2	[7]	
OUT	-	7	-	-	2	7	[]
IN	1	-	-	2	7	[1]	
OUT	-	1	-	2	7	1	[]
DEC	3	-	-	[2]	7	1	
OUT	-	2	-	2	[7]	1	
OUT	-	7	-	2	7	[1]	
OUT	-	1	-	2	7	1	[]
IN	8	-	2	7	1	[8]	

Resultat

	DyKnow implementation				SIGNAL implementation			
	t_f	t_a	Diagnoses	Active tests	t_f	t_a	Diagnoses	Active tests
1	0	101.9	NF	1,2,**5**	0	101.7	NF	1 2 **5**
2	98.9	101.9	1,3,5,6	1,3,10,**13**	99.2	104.5	1,3,5,6	1,3,10,**13**
3	98.6	102.2	1,3,25,26,45,46	1,2,6,7,**8**,11,**12**	99.6	106.1	1,3,25,26,45,46	1,2,6,7,**8**, 11, **12**
4	98.4	101.4	1,25,26,35,36,45	1,2,**6**,7,**9**,10,11	99.7	108.3	1,25,26,35,36,45	1,2,**6**,7,**9**,10,11
5	98.5	102.1	1,26,36,45	1,2,7,10,**11**	99.8	112.9	1,26,36,45	1,2,7,10,**11**
6	98.8	-	1,26,36	1,2,7,10	101.2	-	1,26,36	1,2,7,10

SIGNAL

SIGNAL är:

- elegant
- rätt kraftfullt
- strikt och begränsat
- mycket svårt att arbete med

Problem

- Inga dynamiska arrayer
- Inga ordentliga loopar
- Massor av klockor
- Kompilatorn

Framtida arbete

Kombinera DyKnow och SIGNAL i ett GALS-system.

Sammanfattning

- SIGNAL är ett trevligt och elegant språk, men mycket svårt att arbeta med
- FlexDx är inte anpassat för detta
- Det går att implementera FlexDx i SIGNAL, men man måste göra kompromisser
- Att implementera FlexDx i endast SIGNAL kanske går, men det skulle vara svårt

Frågor?