'96 대한전기학회 하계학술대회 논문집 (1996. 7. 22~24)

페트리 네트를 이용한 시퀀스 흐름 제어기 설계 및 구현

°김문철*, 김응석*, 허우정*, 신경봉*, 조영조**, 류해영***
* 삼성종합기술원 산업전자연구실
** 한국과학기술원 정보전자연구부
*** 삼성중공업 기전연구소

Sequence Flow Controller Design and Implementation Using a Petri net

°Moon-Cheol Kim*, Eung-Seok Kim*, Woo-Jung Huh*, Kyeong-Bong Shin*,
Young-Jo Cho**, Hae-Young Ryu***
* System Control Laboratory, Samsung Advanced Institute of Technology.
** Division of Electronics and Information Technology, Korea Institute of Science and Technology.
*** Research Institute of Machinery and Electrotechnology, Samsung Heavy Industries Co. Ltd.

Abstract - Design methods for sequence flow controllers play an important role in industrial automation. To design a flexible, reusable, and maintainable control software has become a key issue nowadays. Petri net has been emerging as an important tool to provide an integrated solution for modeling, analysis, simulation, and control of industrial automated systems recently. This paper describes an international standard of programming languages for programmable logic controllers, IEC 1131-3 SFC(Sequential Function Chart), and the sequence flow control method for an SFC using a Petri net-like model.

1. 서 론

기존의 LLD(Ladder Logic Diagram)는 시스템의 작업 순서를 구성하는데 이용되어왔으나 이러한 LLD는 시스템의 크기가 커짐에 따라 그 복잡성이 훨씬 심해져서 어떤 문제를 찾는 일이 상당히 어려워졌고, 시스템의 사양이 변하게 되면 대부분의 내용을 바꾸어야하는 경우가 많았다.[1]

이와 같은 LLD의 한계를 극복하기 위해 PLC 업체들은 여러 가지의 programming language들을 개발해 왔는데, 여러 가지 language들 사이의 호환성과 규격통일을 위한 IEC(International Electrotechnical Commission) 1131 규격이 만들어지게 되었다. 시스템 전체의 흐름를 기술하기 위해 IEC 1131-3 규격에서 정의된 SFC(Sequential Function Chart)를 이용하고 SFC내의 step들은 실제 application을 표현하기 위한 programming language들을 이용하여 만들어진다.[2]

본 논문에서는 이 SFC로 표기된 순차 프로그램을 수행함에 있어, SFC를 Petri net과 유사하게 modeling하고 SFC의 각 step을 별도의 task로 만들어 이들 사이에서 프로그램의 수행을 위한 정보를 주고 받으면서 전체의 흐름을 자체적으로 제어하는 방법을 보인다.

2. 시퀀스 흐름 제어기

2.1 IEC 1131-3 규격의 SFC

다음의 그림 1에 SFC를 이용한 전체 시스템 제어 프로그램의 한 예를 보이고 있다.

그림 1의 SFC에서 네모상자로 표시된 것을 "step"이라고 부르고 가로로 짧게 그어진 막대로 표시되어 step과 step을 연결하는 것을 "transition"이라 부른다. 각 step은 IEC 1131-3 규격에서 제시된 다른 programming language들에 의해 시스템의 어떤 동작내지는 작업을 기술하게 되고, transition은 주어진 "transition condition"을 검사하면서 다음 step으로 흐름을 연결시키는 기능을 한다. 이때, condition이 참인 경우에만 다음으로 흐름이 진행될 수 있고, 거짓인 경우는 참이 될 때까지 그곳에 머물러 있게 된다. transition condition은 하나의 이진 산술식으로 표현된다.

SFC는 "initial step"이라 불리는 특별한 step에서부터 흐름이 시작된다. initial step은 한 SFC내에 하나만 존재하고 (그림 1의 S0) 특별한 네모상자로 표현된다.

SFC에서의 흐름의 종류에는 크게 3가지가 있는데, 그 첫째로는 그림 1의 'S1→T1→S2'와 같이 한 step의 완료후에 하나의 transition condition을 검사하고 다음에 하나의 step으로 진행되는 것이다. 이를 "single sequence"라고 한다.

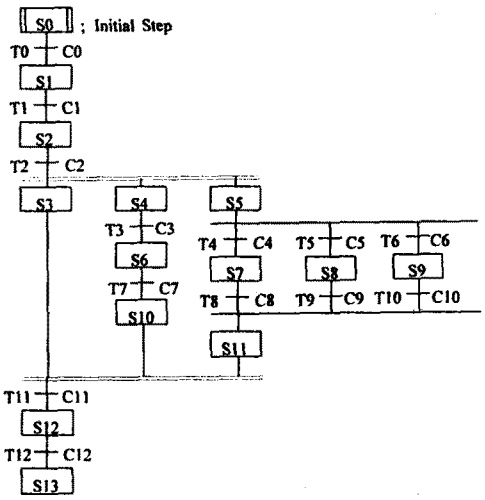

그림 1. SFC (Sequential Function Chart).

둘째로는 그림 1의 'S2→T2→(S3 and S4 and S5)'와 같이 하나의 step이 완료된 후에 다음 transition을 검사하고, 이 conditioin이 참일 경우, 여러개의 step이 동시에 수행되는 경우이다. 이 경우를 "divergence of simultaneous sequences"라고 하고, 그림상으로는 횡으로 겹줄로 표시한다. 이때, 동시에 수행되는 step들간에는 priority와 같은 제약은 없고, step들은 동시에, 독립적으로 수행된다. 분기가 이루어진 다음에 이들이 다시 모이게 되는 과정에서도 위와 같은 식의 진행이 되는데, 그림 1의 '(S3 and S10 and S11)→T11→S12'의 경우처럼 동시분기된 것이 다시 모이는 경우는 "convergence of simultaneous sequences"라고 하고, 이 경우의 흐름은 앞의 모든 step들(S3,S10, and S11)이 작업을 끝내고 해당 transition condition(C11)이 참일 경우에 다음 step(S12)으로 진행된다.

마지막 세번째 경우는 그림 1의 'S5→T4→S7 or →T5→S8 or →T6→S9'과 같이 여러개중에 하나의 작업만을 수행하는 경우이다. 이는 "divergence of sequence selection"이라 하고,

그림상으로는 횡으로 한줄로 표시한다. S5 step 수행후에 연결된 다음 transition들중에 가장 왼쪽의 것부터 차례로 조건을 검사하여 참인 조건에 해당하는 step으로 흐름이 연결된다. 이들이 다시 모이게 되는 경우를 "convergence of sequence selection"이라 하고, 앞서 진행된 한 step의 작업이 완료된 후에 그 step의 해당 transition condition을 검사하여 흐름이 진행된다.

2.2 SFC & Petri net model

앞 절의 설명에서 보듯이 SFC의 동작은 Petri net과 상당히 유사하다. SFC의 step은 Petri net의 place에, SFC의 transition은 Petri net의 transition에 각각 대치시켜 볼수 있으며, SFC에서의 transition condition이 Petri net에서의 transition firing condition에 해당한다고 볼수 있다. 그림 1을 Petri net으로 표현한 경우가 그림 2인데 1:1로 match됨을 알수 있다.

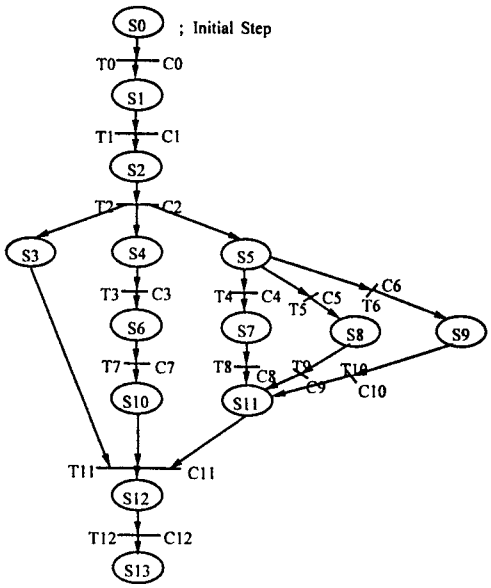

그림 2. Petri Net model.

LLD(Ladder Logic Diagram)와 Petri net을 design complexity와 response time의 관점에서 비교해 볼때 graphical complexity, adaptability 면이나 scan time, execution time 면에서 Petri net의 우수성이 증명되었고, Petri net은 digital input/output, computer system을 가진 discrete event control system에 잘 적용될 수 있다는 사실이 증명되었다.[1] 그러므로 SFC 역시 이러한 Petri net의 장점을 충분히 가질 수 있는 구조라 볼수 있다.

그러나 SFC와 Petri net이 완벽하게 동일한 구조를 가지는 것은 아니고, 몇가지 차이점이 있다. 그 첫째는 Petri net과 달리 SFC는 각 step이 하나의 token만을 가진다는 것이고, 다음으로는 SFC는 initial marking이 initial step이라는 특수한 곳에만 존재하는 형태라는 점이다. 앞서 설명된대로 SFC의 흐름은 initial step에서 시작되고 이러한 initial step은 SFC내에 단 하나만 존재하므로 initial marking이 항상 일정한 셈이다. 이런 차이점들로 인해 다음에 설명하게 되는 flow control algorithm이 일반적인 Petri net model의 경우와는 다른 모습을 가지게 된다.

2.3 Flow control algorithm

기존의 distributed control system이나 flexible manufacturing system등에서 자원을 공유하는 경우에 발생될수 있는 deadlock / starvation 문제의 해결방법이나 flow control등을 위해 Petri net model이나 이와 유사한 형태의 model들을 이용하고 있다.[3][4] 그러나 어떤 시스템이 Petri net model로 simulation될때, transition을 통한 token의 전달이 data의 흐름 및 그 출력을 의미하게 되는데, token의 이동은 data 이동만을 표시할 뿐이고 실제 어떤 data가 전달되는지에 대한 정보는 없다.[5]

본 논문에서 제안하고자 하는 방법은 Petri net model을 이용하지 않고, SFC 자체의 특성을 감안한 token passing algorithm이다. 이 방법의 가장 중요한 특징은 model 자체가 token passing algorithm을 가진 채로 만들어진다는 점이다. 즉, 별도의 scheduler가 필요없는 구조라서 임의 형태의 SFC 구조에 대해서도 적용될수 있는 방법이며 scheduler에 의한 overhead가 전혀 없고 확장성이나 유지관리면에서도 상당히 유연한 구조라는 장점을 가지고 있다.

이 방법은 SFC의 각 step을 별도의 task("executer"라고 부름)로 만들고 이들 executer 내부에서 자신의 입력 / 출력 transition condition을 검사하면서 다음에 token을 전달할 executer를 찾아가는 것이다. 이를 위해서 그림 1에 있는 SFC를 그림 3과 같은 SFC code로 만든다.

```
SFC code {
      TRANSITIONS = 13  STEPS = 14
      T0 : (S0) > (S1) @ C0
      T1 : (S1) > (S2) @ C1
      T2 : (S2) > (S3,S4,S5) @ C2
      T3 : (S4) > (S6) @ C3
      T4 : (S5) > (S7) @ C4
      T5 : (S5) > (S8) @ C5
      T6 : (S5) > (S9) @ C6
      T7 : (S6) > (S10) @ C7
      T8 : (S7) > (S11) @ C8
      T9 : (S8) > (S11) @ C9
      T10 : (S9) > (S11) @ C10
      T11 : (S3,S10,S11) > (S12) @ C11
      T12 : (S12) > (S13) @ C12
      C0 = TRUE
      C1 = INPUT1 > INPUT2
      ...
      C12 = FALSE
}
```

그림 3. SFC code.

이 SFC code를 해석하여 executer가 사용할 표 1과 같은 data structure를 만든게 되는데, transition condition의 경우는 해석 단계에서 그 값이 필요한 것이 아니라 실제 수행되는 시점에서 값을 검사하게 되므로 별도의 data block에 해당 상수 또는 변수의 address만을 담아두었다가, 실제 그 값의 검사는 executer 내부에서 이루어지게 된다.

표 1. data structure

한 SFC내의 transition 개수	한 SFC내의 step 개수

step과 transition과의 관계를 표시하는 테이블

trans \ step	#0	#1	#2	...	#(MaxStep-1)
#0	.	OUT	0	...	0
#1	0	IN	OUT	...	0
#2	0	IN	0	...	0
...	0	0	0	...	0
#(MaxTr-1)	0	0	0	...	OUT

※ 윗 테이블에서의 IN/OUT 표시는 transition 기준임.
즉, 왼쪽 윗 두번째 블럭의 경우, transition#0의 output step이 step#1이라는 의미이다.

※ MaxStep : 한 SFC내에 허용되는 step 갯수의 최대값
MaxTr : 한 SFC내에 허용되는 transition 갯수의 최대값

각 transition의 condition block array

0	앞 parameter	관계 연산자	뒷 parameter
1	앞 parameter	관계 연산자	뒷 parameter
...	...	...	...
MaxTr	앞 parameter	관계 연산자	뒷 parameter

다음에 executer의 내부 구조가 설명되어 있다.

Executer의 구조 :
Start of executer {
① token을 기다린다. → 어느 step task로부터 token을 받으면 통과한다. token을 받을때, token을 준 step task의 ID number를 같이 받는다.
② input transition 갯수를 검사한다.
(1) 갯수가 1개인 경우
input transition의 condition을 검사하여 참(TRUE)이면 통과하고 거짓(FALSE)이면 참(TRUE)이 될때까지 기다린다.
(2) 갯수가 2개 이상인 경우
token을 준 task id를 이용하여 그 task의 output transition만 검사한다. 참(TRUE)이면 통과하고 거짓(FALSE)이면 참(TRUE)이 될때까지 기다린다.
③ 해당 step 내부의 동작을 수행한다.
④ 이 task의 output transition의 갯수를 검사한다.
(1) 갯수가 1개인 경우
output transition의 condition을 검사하여 참(TRUE)이면 통과하고 거짓(FALSE)이면 참(TRUE)이 될때까지 기다린다. 참(TRUE)이 되면 이 task의 모든 output transition에 연결된 task들에게 token을 전달한다. 이때, token속에는 이 token을 보내는 task의 id number가 들어 있다.
(2) 갯수가 2개 이상인 경우
맨 왼쪽 output transition의 condition부터 차례로 검사하면서 처음으로 참(TRUE)이 되는 transition을 찾아 그 transition의 output step task에 token을 전달한다. 이때, token속에는 이 token을 보내는 task의 id number가 들어 있다.
} End of executer

executer가 만들어지면 선두에서 token을 기다리게 된다. 이 token이란, executer들 사이에 동기를 맞추고, 정보를 전달하기 위한 도구로서 일반적인 multi-tasking operating system에서 제공되는 것들이다.[6] 본 executer에서는 이 token이라는 것이 특별한 대상을 지칭하는 것은 아니고, 이러한 목적으로 사용되는 것들에 대한 통칭으로 이용하고 있다. 이 token을 보내는 executer에서 token의 내부에 임의의 정보를 담아 보낼수도 있다. (예: VRTXsa의 mailbox, event flag group, message queue등)

token을 받은 executer는 표 1의 data structure를 이용하여 자신의 input transition의 갯수를 검사하게 되는데 그림 1의 T1→S2 또는 T2→S3등의 경우와 같이 갯수가 하나인 경우에는 해당 transition의 condition을 검사하여 그 조건이 참(TRUE)이면 다음으로 진행한다. 이때, 그림 1에서의 (S3 and S10 and S11)→T11→S12의 경우와 같이 input transition은 하나지만 그 transition에 연결된 step 갯수가 복수개인 경우는 그 갯수에 해당하는 만큼의 token을 받게 된다. 즉, 그 갯수만큼의 token을 받고나서 해당 transition condition(C11)을 검사한다는 뜻이다. 이는 앞서 설명한 convergence of simultaneous sequences의 경우인데, S3, S10, S11의 세 executer들이 모두 완료된 후에 transition condition을 검사하고 다음 executer로 진행한다는 SFC의 사양을 만족시키고 있다.

그림 1의 (T8 or T9 or T10)→S11와 같이 S11 executer의 input transition의 갯수가 복수개인 경우에는 선두에서 받은 token을 검사하여 그 token을 어느 executer가 보낸 것인지를 찾는다. 그리하여 그 executer의 output transition의 condition만을 검사하고 그 조건이 참(TRUE)이면 다음으로 진행한다. 이는 앞에서 설명한 convergence of selection sequence의 경우에 해당하는데, 작업의 흐름이 여러 분기중에 하나로만 흘러온 경우이므로 token을 보낸 분기의 transition condition만 검사하면 된다.

input transition에 대한 검사가 완료되고 나면 executer 내부의 작업을 수행하고, 이 작업이 완료되면 다음에 어느 executer로 흐름을 진행시킬 것인지를 결정하게 된다. 즉, 이번에는 executer의 output transition의 갯수를 검사한다. 그림 1의 S3→T11 또는 S10→T11의 경우처럼 이 갯수가 하나인 경우는 바로 그 transition의 condition을 검사하고 그 조건이 참(TRUE)이면 이 transition에 연결된 executer로 token을 전달한다. 이때, 이 token에는 보내는 executer의 id number가 들어가 있게 된다. 반면에 그림 1의 T2→(S3 and S4 and S5)의 경우처럼 output transition에 연결된 executer의 갯수가 두개 이상일 경우에는 복수개의 executer들 모두에게 동일한 token을 전달하게 된다. 이는 앞서 설명된 divergence of simultaneous sequences에 해당하는 것이다.

output transition의 갯수가 복수인 경우는 그림 1의 S5→(T4 or T5 or T6)와 같은 경우인데, transition중에 맨 왼쪽의 transition condition부터 차례로 검사하여 처음으로 참(TRUE)이 되는 transition에 연결된 executer로 token을 보내면 된다. 이는 앞서 설명한 divergence of selection sequence에 해당하는 부분인데, 그 사양대로 흐름이 진행되게 되어있는 것을 볼수 있다. 이 token에는 보내는 executer의 ID number가 들어 있어 받는 executer가 이를 확인할 수 있다.

3. 결 론

본 논문에서는 IEC1131-3 SFC내에서의 작업의 진행을 별도의 scheduler없이 model 스스로 진행시킬수 있는 방법을 제시함으로써, SFC의 구조가 복잡해지거나 서로 다른 성격의 분기가 중첩되더라도 전체 흐름을 이어가는데 아무런 문제가 없는 구조를 제시하였다. 이 구조는 SFC 변경에 따른 별도의 작업이 필요치 않으므로 상당히 유연하며, 확장성이 있고, 한번 확인된 프로그램은 하나의 단위로 구성되어 다른 프로그램에서도 이용할 수 있어 재사용이 용이하고, 모듈화될수 있는 장점을 가지고 있다. token의 흐름을 추적하면 전체의 흐름을 파악하기가 쉽고, 응용프로그램의 개발단계에서 각 부분부분을 따로 시험해보기도 용이하며, 그림으로 표시된 내용을 직접 알대일로 변환함으로서 전체 구조파악이 용이한 구조이다.

4. 참고문헌

[1] K.Venkatesh, M.C.Zhou, and R.J.Caudill, "Comparing Ladder Logic Diagrams And Petri Nets For Sequence Controller Design Through A Discrete Manufacturing System", IEEE Transactions on Industrial Electronics, vol.41, no.6, pp.611-19, December 1994

[2] Internation Standard IEC 1131-3, IEC 1993

[3] K.Y.Xing, B.S.Hu, and H.X.Chen, "Deadlock Avoidance Policy for Petri-Net Modeling of flexible Manufacturing Systems with Shared Resources", IEEE Transactions on Automatic Control, vol.41, no2, pp.289-95, February 1996

[4] T.Murata, N.Komoda, K.Matsumoto, and K.Haruna, "A Petri Net-Based Controller for Flexible and Maintainable Sequence Control and its Applications in Factory Automation", IEEE Transactions on Industrial Electronics, vol.IE-33, no.1, pp.1-8, February 1986

[5] T.Nakano, T.Hayashi, K.Nogi, and K.Mori, "System Function Simulation Method and Apparatus Therefor Using Petri Net Symbols", US Patent #4,866,605, 1989

[6] Spectra VRTX/OS VRTXsa User's Guide, Microtec Research, July 1994