객체 및 시제논리에 기반한 실시간 시스템 모형화 방법

김 정 술† · 강 병 욱††

요 약

우리는 이 논문에서 실시간 시스템을 위한 모형화 방법을 제공한다. 이 방법은 DARTS(Design Approach for Real Time Systems)에 기초하며 훨씬 확장되었다. 기존의 DARTS 방법은 좋은 설계 가이드라인은 제공해 주지만, 구조적 분석방법을 이용하며, 명세언어를 제공하지 않는다. 그래서 본 논문에서는 객체에 기반하고 분석까지 가능한 확장된 DARTS 모형화 방법을 제공하고자 한다. 분석을 위해 수치 페트리넷(NPN)을 이용하여 시스템의 내부행위를 보여주며, 천이동기 순서제어를 위해 시제논리에 기초한 명세언어를 제공한다. 적용 예제를 통하여 제안된 방법이 잘 적용되었고 내부의 메시지 전달과정에 대한 도달성 그래프를 통하여 데드락의 존재여부를 미리 확인할 수 있어 분석이 용이하며 자연스럽게 설계와 연결된다.

The Method for Real-Time Systems Modeling Based On the Object and Temporal Logic

Jung Sool Kim† · Byung Wook Kang††

ABSTRACT

In this paper, we present a modeling method for the real-time systems. This method is based on the DARTS(Design Approach for Real-Time Systems) and widely extended to analysis phase. The DARTS method provides a good guideline for the real-time software design, but it uses structured analysis and does not provide a specification language. So, this paper provides extended DARTS modeling techniques to the analysis area based on the objects. Internal behavior of system showed by means of a NPN(Numerical Petri Net) for analysis, and the specification language is provided based on the temporal logic for transition synchronization sequence control. By the example, we identified the proposed method was applied well. And through the reachability graph, we verified whether the deadlocks may occur or not in the analysis phase before the design phase. Thus, it gives easy way to analysis, so that it will lead to the design phase naturally.

1. 서 론

실시간 시스템은 일반적인 경영정보 시스템(MIS)과는 다르다. 이 분야는 MIS에서 고려하지 않는 프로세스(process)들의 병행성및 시간들을 고려하기 때문이다. 실시간 시스템은 크게 두 분야로 대별되는데 시간이 별로 엄격하지 않은 소프트 데드라인(soft deadline) 시스템과 시간이 엄격하게 제한되는 하드 데드라인(hard deadline) 시스템으로 분류된다. 지금까지 많이 사용되어온 실시간 시스템을 위한 분석, 설계 방법론들은 Ward/Mellor의 RTSA/RTSD[33], Gomaa의

† 정 회 원 : 영남대학교 컴퓨터공학과
†† 종신회원 : 영남대학교 컴퓨터공학과 교수
논문접수 : 1997년 8월 13일, 심사완료 : 1998년 4월 29일

DARTS[5,6], Parnas의 NRL/SCR[25], Jackson의 JSD[10], Harel의 Statemate[7,32], Booch의 OOD [3]등이 있다. 이러한 방법론들은 저마다 분석, 설계시의 장단점을 가지는데 그 중에서도 실시간 소프트웨어 설계에 많이 쓰이고 있는 방법이 바로 DARTS방법이다. 그런데 이 방법은 소프트웨어 설계방법이며 또한 명세언어를 제공하지 않는다[37]. 그래서 본 논문에서는 이 DARTS를 위한 명세언어를 제공하며 설계에 국한된 영역을 분석까지 확장하고자 한다. 소프트웨어 시스템 설계시 가장 중요한 작업중의 하나가 시스템의 신뢰성(reliability)을 보장하는 것인데 특히, 안전에 민감한 시스템에는 더욱 그렇다. 신뢰성의 보증(assurance)은 시스템의 요구사항이 정확하게 사용자의 요구사항을 따르고 있음을 의미하며, 소프트웨어의 구현이 정확하게 설계를 실현하고 있음을 의미한다. 이를위한 도구로 formal method가 사용되는데, 엄격한 정형화의 구현으로 설계오류나 부정확한 명세서의 기술을 피할수 있어 소프트웨어 개발비용의 부수적인 증가를 막을수 있다[12]. 아울러 시스템의 복잡성을 위해서 객체 개념을 추가하여 시스템을 객체 형태로 모델링하고 분석단계의 도달성 그래프[8,18]를 통하여 시스템 요구 분석단계에서의 데드락의 발생유무를 미리 체크하고자 한다. 그래서 자연스럽게 객체에 기반한 DARTS설계의 개념으로 이끌겠다는 것이다. 본 논문은 다음과 같이 구성된다. 2장에서는 관련 연구를 서술한다. 이어 3장에서는 제안된 방법의 전체적인 내부, 외부의 구조와 행위 및 명세언어를 설명하고 4장에서는 간단한 전형적인 실시간 시스템의 사례를 들어 본 명세방법을 적용한다. 끝으로, 5장에서 결론을 내린다.

2. 관련 연구

안전에 민감한 실시간 시스템의 분석, 설계방법은 아래와 같이 다양하다.

2.1 RTSAD(Real Time Structured Analysis Design) [20,33]

이 방법은 구조적 분석/설계 방법을 실시간에 적용하도록 확장된 방법으로 Ward/Mellor와 Hatley의 방법론이 가장 많이 사용된다. DFD(Data Flow Diagram)와 CFD(Control Flow Diagram)가 주요 기호로 사용되는 전통적인 개발방법이다. 지금까지 많이 사용되어온 방법으로 시간요구사항이나 정형적인 시멘틱스(semantics)가 없다.

2.2 DARTS (Design Approach for Real Time Systems) [5,6]

이 방법은 구조적 분석을 기본으로 실시간을 위한 설계방법이다. 시스템을 다수의 병렬 타스크로 구성하고 이들 사이의 통신 인터페이스를 정의하는데 중점을 둔 방법이다. 강력한(powerful) 타스크 구조화의 가이드라인을 제공한다.

2.3 STATEMATE[32]

이 방법은 Reactive 시스템 개발에 적용하기 위해 Harel에 의해 제안된 방법으로 실행 가능한 명세서와 비쥬얼 포말리즘(Visual Formalism)[7]의 지원을 강조한다. 기능, 구조, 동작 모델의 표현을 위해서 Activity Chart, Module Chart, State Charts를 지원하는 계층적 상태표현이 좋은 방법이다. 이 방법은 아직 정형적인 검증을 지원하지 않는다.

2.4 OOA/OOD(Object Oriented Analysis/Design)

객체 지향 개발을 이끄는 다수의 방법들이 존재한다. 추상화와 정보은닉의 개념에 의한 방법론으로 객체 지향적 프로그래밍 언어로 구현될 경우 계승 및 폴리모피즘같은 강력한 데이터 모델링을 지원한다. Rumbaugh의 OMT[29]와 Booch[3]의 방법이 많이 사용된다. 최근에는 ROOM기법[30]과 RTO.k 객체 모델[14,15]이 존재한다.

2.5 시제논리(Temporal Logic)[16,21,22]

시제논리는 시간에 있어서의 상황(Situation)을 다루는 특별한 논리(Logic)들 중의 하나다. 보통의 논리는 정적인 상황을 설명하는데 반해서 시제논리는 시간의 열(Passage)에 기인하여 어떻게 상황이 변경되어 가는지를 보여준다. 시제논리의 접근방법의 장점은 병행(Concurrent)한 프로그램의 행위(Behavior)분석과 정형화(Formalism)를 제공한다는 점이다. 실시간 시제논리는 과거의 선형 시간 시제논리를 확장한 것이다. 기본 연산자는 ○(next의미)와 μ(until의미)인데 이들의 조합으로 다양한 연산자 □ (henceforth의미), ◇ (eventually의미), U(unless의미), P(precede의미)등의 연산자들이 상태공식과 함께 사용된다. 최근의

시제논리의 경향도 그래피칼한 환경과의 접목을 시도[19]하고 있으며 본 논문에서는 실시간 시제논리(RTTL)[21]를 이용한다.

그 외에도 수많은 방법론과 기법들이 존재한다. 본 논문도 이러한 기법들 중의 하나이다. 그러나 본 논문은 기존의 시스템 개발방법과 형식명세의 상호 문제점[17]을 고려하여 서로의 장점인 개발방법론의 실용성과 형식명세의 정형화에 관점을 둔다. 즉, 방법론 차원의 그래피칼한 환경과 정형성과의 결합으로 사용자에게는 쉬운 도구로, 시스템에 있어서는 정확성을 보장한다. 특히 DARTS의 타스크 구조화 가이드라인과 통신 인터페이스를 기반으로 객체와 접목하며 객체내부의 동적인 행위는 NPN(Numerical Petri Net)[2]을 이용(상태의 조건이나 동작표기가 가능하며, 데이터 변수 튜플로 표현 가능하므로)하여 고유 객체들이 가지는 상태천이로서 행위를 표현한다. (이는 실 사용자와의 좋은 대화 수단이 된다) 또한 실시간 시제논리(RTTL)는 통신 모듈천이(외부천이)와 객체나 통신모듈의 액션천이(내부천이)를 위한 동기제한자(Synchronization Constraints)로 사용하며, 필요시 시스템의 내부 상태를 표현하는 상태공식(State Formula)으로 사용한다. 그래서 개발방법론이 제공하는 가이드라인과 실제 대규모 시스템 개발 시에 필요한 실 사용자와의 통신수단과 그리고 정확한 시스템의 표현수단을 제공한다. 이러한 정형화된 방법과의 접목은 시스템 내부의 모순을 제거하고, 분석가로 하여금 시스템의 목표(what)설정에 전념하도록 만든다[13,26].

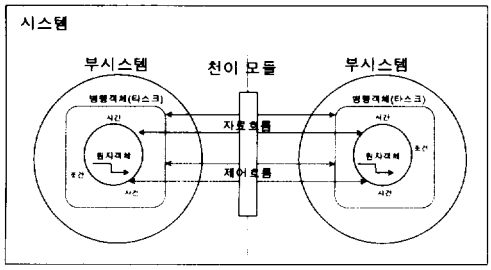
3. 제안된 방법

3.1 시스템 모델링 과정(System Modeling Process)

소프트웨어 개발은 다른 상호작용의 스타일에 따라, 많은 프로세스 에이젼트(agent)들이 협력(cooperate)하는 민감한(clitical) 행위(activity)이다. 개발 과정에서의 협력은 비동기적이거나 동기적 일 수 있다[31]. 하나의 완전한 시스템을 구성하기 위한 구성절차(탑-다운방식)는 다음과 같다. 먼저 하나의 시스템을 실시간 시스템의 특성을 위한 관심분리의 사상에서 부(Sub)시스템으로 분류한다. 분류된 부시스템은 다시 병행(Concurrent)객체로 다시 분류되고 이 병행 객체는 혼합된(Composite)객체로서 다시 내부의 원자(Atomic) 객체를 가질 수 있다. 원자객체의 내부는 순서적(Sequential)으로 수행되며 다른 객체와 통신을 수행한다. 객체간의 통신으로 기능(자료)의 흐름이나 제어(Control)흐름을 나타낼 수 있다. 원자 객체는 내, 외부의 상황에 따라 시간관련 사건이나 조건 및 행동을 수행한다. 이 객체간의 통신은 메시지의 전달로 수행되는데 약결합 통신모듈(Loosely Coupled Communication), 강결합 통신모듈(Closely Coupled Communication), 정보은닉 모듈(Information Hiding Module)등의 통신 인터페이스를 제공한다[5,6]. 일반적으로 객체는 활성적(Active)이거나 수동적(Passive)이다[5]. 객체들이 종종 수동적 이여서 결코 액션(Action)들을 초기화할 수 없는데 반해 객체기반 접근에서는 Active 객체(즉 비동기 객체로서 자치적이고, 자신의 Action을 초기화가능)를 지원한다. Active한 객체는 병행Task이다[34]. Task는 자신의 제어(Thread Of Control)를 가지고 다른 Task와 독립적으로 실행 가능하다. 그래서 본 논문에서는 실시간 시스템의 기본 개념인 Task를 병행 객체로 변환시킨다. 수동 객체는 정보은닉 개념이 적용된 모듈이고 자신의 제어(Thread Of Control)를 가지지 않는다. 이 논문에서의 수동객체의 대상은 동기, 비동기 통신모듈, 정보은닉 모듈 등이 된다. 전통적인 설계기술로서 대규모의 실시간 컴퓨팅 시스템의 신뢰성을 향상시키기에는 어렵기 때문에 설계의 효율성과 시스템의 신뢰성을 위해 객체기반 기술을 적용한다[15]. 최근의 많은 시스템분석, 설계방법이 객체에 기반한다[28, 36]. 아울러, 본 논문의 범위에서는 벗어나지만, 실제 객체를 인식(Identify)하는 방법은 객체 구조화 가이드 라인[5]을 이용하기로 한다. 전체 절차는 다음과 같다.

1. 대상 시스템의 문제 영역(Problem Domain)에서 시스템의 문맥도(Context Diagram)를 작성한다.
2. 복잡한 시스템인 경우는 부시스템으로 나누어 각 부시스템의 문맥도를 작성한다.
3. 각 시스템에 대한 자료흐름도(DFD)를 작성하는데 수준(Level)1 정도로 추상화되게 작성한다.
4. 각 시스템의 자료흐름도에서 DARTS방법의 실시간 가이드라인을 적용하여 병행성있는 타스크(Task)를 생성한다.

5. 각 타스크의 통신 인터페이스 모듈을 설정한다.
6. 타스크를 병행 객체모듈로, 통신 인터페이스를 수동 객체모듈로 인식한다.
7. 병행객체를 분해하는데, 관점을 복합객체와 원자객체로 하여, 내부에 다시 병행객체가 존재할시 위의 과정들을 반복한다.
8. 각 원자 객체들을 점진적으로 개발한다. 즉, 각 객체에 대한 상태 천이도를 작성한다. 이 객체의 내부행위는 고 수준 페트리 넷을 통하여 모델링 한다.
9. 각 객체에 대해 본 논문에서 개발한 명세언어로 명세서를 기술한다.
10. 도달성 그래프[8,18]를 통하여 메시지의 흐름을 분석한다.



(그림 1) 프로세스 모형
(Fig. 1) Process Modeling

위에서 제시된 각각의 소프트웨어 구조(architecture) 명세(specification)의 개념은 대규모 소프트웨어 시스템을 효과적으로 구성하기위한 새로운 방법이다[11].

(그림1)은 전체 시스템에 대한 프로세스 모형이다. 이 모형은 전체시스템이 병행하게 실행되며, 하나의 부시스템에서 서로서로 천이모듈을 통하여 메시지를 전달하며, 원자 객체 내부의 화살표는 순서적임을 의미한다. 예를 들어 각 부시스템들이 서로 토큰을 가지면 천이모듈(통신모듈)의 점화에 의해 메시지나 제어가 전달된다는 의미이다.

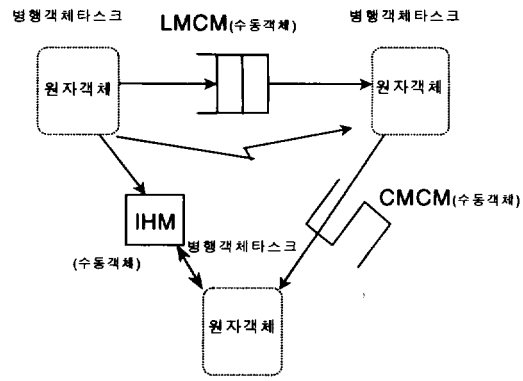
3.2 시스템(부시스템)

OARTS(이하 Object based Approach for Real-Time Systems 라고 부른다)에서는 시스템(또는 부시스템)을 병행 객체 타스크들과 그들 사이의 인터페이스로 구성된다. (그림2)를 참고하기 바란다. 즉 시스템(또는 부시스템)은 다음의 3개의 튜플(Tuple)로 정의된다.

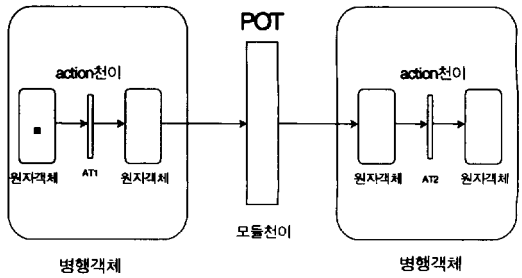
S(SYSTEM) = (CO, POT, I), 여기서, CO : 병행 객체 타스크의 집합, PO : 수동 천이 객체의 집합, I : 인터페이스인 메시지의 흐름을 나타낸다.

본 논문에서 사용되는 기호들의 정의는 다음과 같다.

1. 병행객체 : 이 객체의 기호는 일반론적인 객체 기호로 모서리가 둥근 사각형으로 정의한다. 이는 페트리 넷[27]의 플레이스와 동일한 의미를 갖는다.
2. 수동객체 : 이 객체의 기호는 직사각형으로 정의한다. 이는 페트리 넷의 트랜지션(바(Bar))과 동일한 의미를 가지는데 천이 모듈이 된다.
3. 메시지흐름 : 이 흐름의 기호는 연속적인 화살표로 나타낸다. 이는 페트리 넷의 아크(Arc)의 개념과 동일하다.



(그림 2) 객체타스크 인터페이스 모형
(Fig. 2) Object Task Interface Modeling



(그림 3) 복합객체 구조
(Fig. 3) Composite Object

3.3 (Object)모델링

객체 기술은 시스템의 생산성, 신뢰성, 유지보수성, 소프트웨어 재사용성을 향상시켜주고 전략적인 목표달성과 사용자의 요구를 만족시키는데 도움을 준다[1]. 본 논문에서 적용하는 객체기술은 다음과 같다.

3.3.1. 활성객체(Active Object)의 구조

앞에서도 설명한 바와 같이 활성객체는 자신의 제어구조(Thread Of Control)를 가진다. 실시간 시스템의 병행 타스크(TASK)의 개념과 동일하게 사용된다. 활성객체는 복합객체와 원자객체로 구성된다. 만약 내부의 구조가 또 다른 객체를 가진다면 원자객체가 될 때까지 외부구조는 메시지 통신모듈인 수동(Passive) 객체와 관련된다. (그림2)를 참고하기 바란다.

3.3.1.1 복합객체(Composite Object)

복합객체[35]는 아직 분해(Decompose)되지 않은 (즉 내부에 여러 객체가 존재하는)요소를 가진 객체이다. 이 의미는 객체가 계층적으로 구성될 수 있음을 의미한다. 즉 그들 자신의 데이터와 동작(Operation)을 가진 객체들로 아직 구성되어 있는 객체이다. 병행성은 복합 객체에만 허용이 되고 원자 객체에는 허용이 되지 않는다. (그림3)을 참고하기 바란다. 복합 객체의 내부 구조는 다음처럼 표현된다.

CMO_i : 복합객체i(i : 임의의 특정 복합객체), CO_i : 병행객체i(i : 임의의 특정 병행객체), AO_i : 원자객체i (i : 임의의 특정 원자객체), 이때, O(Object) = (CMO ∪ AO)이고, 여기서, $CMO = \cup_{i(i:1에서\ n까지의\ 정수)} CO_i$, $AO = \cup_{i(i:1에서\ n까지의\ 정수)} AO_i$,복합객체 CMO_i의 내부구조 $ICMO_{i(i:임의의\ 특정내부구조)} = (X, Y, Z, I)$ 여기서,

X : X = { X | X ∈P(CMO), CMO_i ∉X},
X : CMO의 멱집합(power set)의 요소.

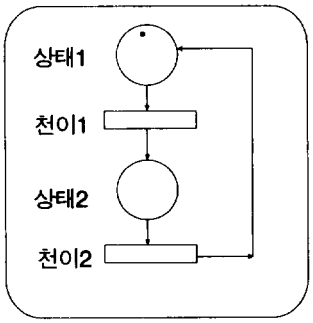
Y : Y = { Y | Y ∈P(CO) }, Y : CO의 멱집합의 요소.

Z : Z = { Z | Z ∈ P(AO) }, Z : AO의 멱집합의 요소.

I : Interconnection Object Set, 여기서
I ⊆ O x O x O, 만약 $(O_i, O_j, O_z) \in I$ 이면,
$OM_z \cap IM_j \neq \emptyset$, 여기서,
IM_j = 객체O_j의 입력통신 모듈
∈ {LMCM,CMCM,OCM,IHM},
OM_z = 객체O_z의 출력통신 모듈
∈ {LMCM,CMCM,OCM,IHM},
I_{jz} = { (O_j, M_k, O_z) | $M_k \in OM_z \cap IM_j$}

3.3.1.2 원자객체(Atomic Object)

이 객체는 모든 분해가 끝난 객체이다. 오직 하나의 최하위 레벨의 객체만을 의미한다. 이 객체는 내부에서 순서적인(Sequential) 제어구조를 가지며 병행성이 존재하지 않는다. 고수준 넷(High Level Petri-Net)[2]으로 표현되어 사용자가 쉽게 분석자와 토론할 수 있는 근거를 제공한다. 원자 객체의 내부구조 IAO는 다음처럼 구성된다. (그림4)를 참고하기 바란다. (그림4)는 초기 상태에서 토큰(메시지)을 가지면 다음천이에 의해 이동되었다가 원래의 자신의 상태로 되돌아오는 액션을 간단하게 묘사한 것이다. 이 객체는 외부의 통신 모듈천이나 다른 원자객체와 천이 동기를 가지게 될 것이다.



(그림 4) 원자객체 구조
(Fig. 4) Atomic Object

IAO = { EXT , INT }
EXT = {ISA, LMCM, CMCM, IHM, OSM, FLOW}
INT = {GROUPED_BY, TIME, ATTR, STATE_VAL, STATE, TRANS, FUN_FLOW, CON_FLOW, INSTAN, MARK, PRI, EXCEPTION}
GROUPED_BY : AO객체들의 집합.
TIME : 시간 구성 요소들의 집합.
ATTR : AO의 매개변수(attribute)의 집합.
STATE_VAL : AO의 상태 변수들의 집합.

PRI : 우선 순위.
STATE : AO의 상태들의 집합. TRANS : 천이의 집합.
MARK : AO의 초기 토큰위치. FUN_FLOW : 자료흐름 관계의 집합.
CON_FLOW : 제어흐름 관계의 집합.
INSTAN : Instance의 집합. EXCEPTION : 인터럽트 사항들의 집합.

3.3.2 수동 객체의 구조

이 객체는 천이 모듈로 메시지 통신 메커니즘의 모든 것을 담고 있다. 외부로부터 들어오는 입력 메시지를 전달받아 외부로 출력 메시지를 전달하는 모듈이다. 이 객체의 구성은 다음과 같다.

객체 $POT_{i(i:임의의\ 수동객체)} \in$ PO(Passive Object)는 4개의 tuple로 구성되어 있다.

$POT_i = (G_i, IM_i, OM_i, F_i)$,

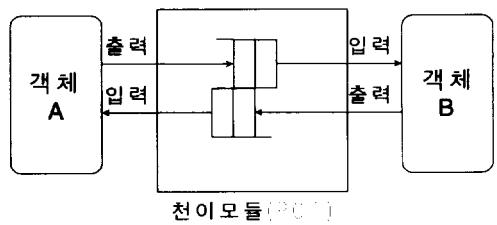
여기서,(i:각 요소들의 임의의 특정 성분)

G_i : 객체 POT_i의 천이 집합.
IM_i : 객체 POT_i의 입력 메시지 큐의 집합.
OM_i : 객체 POT_i의 출력 메시지 큐의 집합.
F_i : 객체 POT_i의 흐름 관계의 집합.

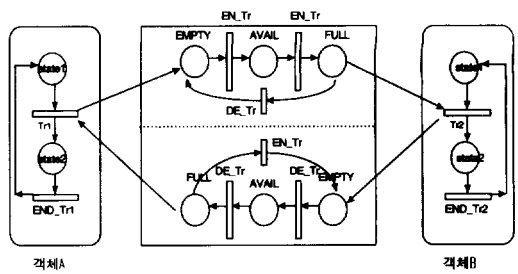
또한 이 객체의 타입은 3가지가 있다. 첫째는 비동기 통신모듈을 위한 것이고, 둘째는 동기 통신모듈이며, 셋째는 정보 은닉모듈이다. 동기 사건 모듈[6]은 병행 객체 내부에 포함되므로 여기서는 언급하지 않는다. 각 모듈에 대한 설명은 다음과 같다. 먼저 이 천이 모듈(POT)의 역할을 보면, 천이는 페트리 넷의 천이(Transition)의 부분 집합이다. 이는 입력 메시지 큐와 출력 메시지 큐를 연결해준다. 그래서 이 천이의 점화는 출력 메시지 큐에 내재된 메시지를 제거하고 입력 메시지 큐에 그 메시지를 전달한다. 이때 천이는 천이 조건을 가지고 원하는 액션(Action)을 수행한다. 메시지가 허용할 수 있는 타입(Type)인지를 조사하고, 출력될 메시지 큐의 타입으로 변환한다. (그림3)을 참고하기 바란다.

3.3.2.1 비동기 통신 모듈(Loosely Message Communication Module : LMCM)[3]

이 모듈의 메시지 큐는 플레이스(Place)의 부분집합(Subset)인 입력 메시지 큐와 출력 메시지 큐로 구성되어 있지만 2개의 메시지 큐가 함께 결합되어 한쪽은 입력 메시지를 받아 저장했다가 천이의 점화 순서에 의한 내부의 천이 점화로 출력 메시지 큐를 통해 원하는 출력을 외부 객체로 전송한다. 그 역도 마찬가지로 동작한다. 그러나 이 구조는 비동기 통신을 위한 구조로 동기통신을 위한 응답이 필요 없기 때문에 메시지를 보낸 객체들은 메시지의 처리 여부에 관계없이 계속 메시지를 전송할 수 있다. 이 비동기 통신을 위한 모듈의 구조는 (그림5)와 같고 (그림6)의 내부에서 일어나는 행위표현은 객체A의 액션천이 Tr1이 점화되었을 때 state1의 토큰은 state2와 Empty 상태로 동시에 전달된다. 이는 비동기의 경우이므로 메시지 큐로 데이터를 전달과 동시에 자신의 상태도 대기하지 않고 천이됨을 의미한다. 비동기통신의 경우 많은 천이제어가 필요한데 이러한 천이점화의 제어는 시제논리로 표현된다.



(그림 5) 비동기 통신을 위한 모듈
(Fig. 5) Asynchronous Comm. Module

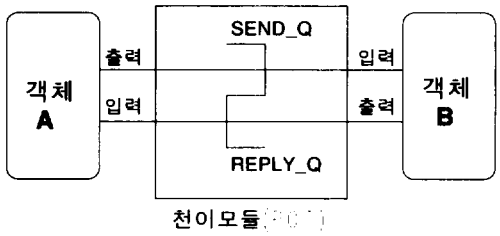


(그림 6) 비동기 통신 행위
(Fig. 6) Asynchronous Comm. Behavior

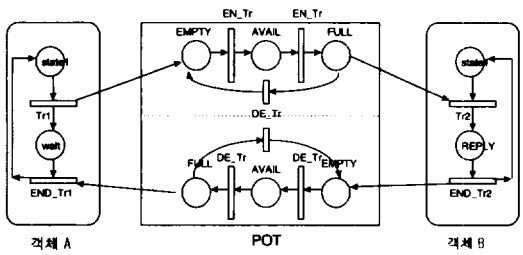
3.3.2.2 동기 통신을 위한 모듈(Closely Message Communication Module : CMCM)[5]

동기 통신을 지원하는 메시지 큐의 구조는 강결합 통신(Closely Coupled Communication)구조이다.

이는 메시지를 보내는 측이 메시지를 받는 측에서 응답이 있을 때까지 기다리는 동기 화된 통신 방법이다. 각 메시지 큐의 크기가 1인 경우의 2개의 메시지가 서로 반대 방향으로 묶여져 있는(비동기 통신과 같은)경우인데 양측 객체간의 송신, 수신측 구분이 없이 먼저 자료를 보내는 측이 송신이 되고 나머지가 수신 측이 된다. 동기 통신을 위한 모듈 구조는 (그림7)과 같다. (그림8)의 동기 통신의 내부 행위는 Tr1이 점화되면 state1의 토큰은 POT의 Empty로 이동하며 객체A의 다음상태는 응답을 받을 때까지 wait상태에 존재한다. 객체B의 천이 Tr2가 점화되면 객체B는 Reply상태가 된다.



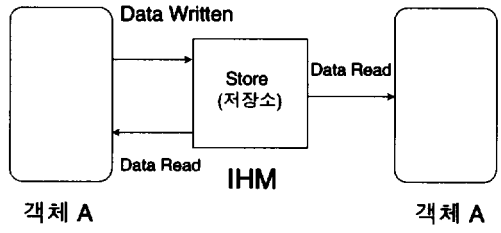
(그림 7) 동기 통신을 위한 모듈
(Fig. 7) Synchronous Comm. Module



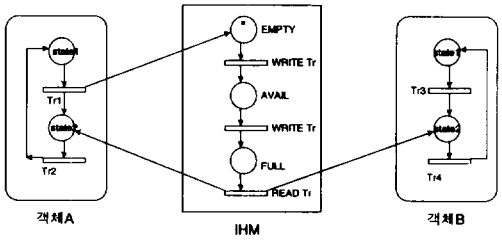
(그림 8) 동기 통신 행위
(Fig. 8) Synchronous Comm. Behavior

3.3.2.3 정보은닉 모듈(Information Hiding Module)[24]

풀(Pool)또는 자료 저장소는 자료를 참조하기 위해 사용된다. 공유된 자료는 2개 이상의 병행 객체들이 읽기(Read)나 읽기/쓰기(Read/Write)를 할 수 있다. 이 모듈은 접근 절차나 자료 저장소를 의미한다. 이의 목적은 주요 설계 결정(Design Decision)들을 숨기는 것이다. 즉 각 중요한 설계 결정은 한 모듈에만 알려져야 하는 것이다[24]. 이의 장점으로는 모듈들이 독립적이어서 수정하기 쉽고 그래서 유지보수가 쉽다는 것이다. 이것은 객체들 사이의 결합을 최소화시킨다[37].



(그림 9) 정보 은닉 모듈
(Fig. 9) Information Hiding Module



(그림 10) 정보은닉 모듈 행위
(Fig. 10) Information Hiding Module Behavior

(그림9)를 참고하면 될 것이다. (그림10)도 Store가 Full일 때까지 읽기/쓰기의 제어가 가능하다.

3.4 객체 명세(Object Specification)

본 절에서 제시하는 객체 명세언어는 병행객체와 원자 객체에 대해 각각 기술 가능하다. 본 명세언어에서 제공하는 각 성분들은 선택적으로 사용 가능하다. 먼저 시스템 분해가 안된(또는 덜 분해된) 복합객체에 대한 언어 구조(Framework)는 아래와 같다. 여기서C_OBJECT

```
C_OBJECT < object_name >
   EXTER : ISA    : < object_name >;
   FLOW  : FM(<object_name>,<msg_name>),
           TO(<object_name>,<msg_name>);
   INTER : GROUPED BY <object_name>;
ENDC OBJECT
```

(그림 11) 복합(병행) 객체 명세 언어
(Fig. 11) Composite(Concurrent) Object Spec. Language

은 복합(Composite) 또는 Concurrent(병행)의 의미이고, 그 아래의 명세 틀(Template)은 원자객체 및 병행객체이다(즉, 병행객체 자신이 원자 객체일 때). 여기서 ||기호는 병행기호이며, |는 천이 한다는 의미이다.

객체의 전체 구조를 보면 한 객체의 외부 환경과의 통신을 위한 인터페이스부분인 EXTERnal과 객체가 내부에 감추어진 INTERnal 성분으로 대별된다. 외부와 전체 구조를 표현하기 위한 성분들은 다음과 같다. 객체가 내부에 구성하고 있는 객체와의 상속관계를 보여주는 ISA성분, 메시지 송수 신을 위한 모듈객체인 MCM성분, 정보은닉 모듈과의 접속을 위한 IHM성분, 사건신호 송, 수신을 위한 OSM성분, 외부 인터페이스의 흐름 순서 관계를 나타내는 FLOW(기능, 제어가 포함된)성분으로 구성된다. 객체의 내부성분을 나타내는 INTERnal부분은 다음과 같다. 먼저, 복합 객체 안에 내포된 병행객체나 원자객체를 구성하는 GROUPED BY, 이 객체가 실행하는 시간성분인 TIME, 메시지들의 자료형(Data Type)을 제공하는 ATTR성분, 우선순위를 제공하는 PRI성분, 전체 상태변수를 나타내는 STATE_VAL성분, 각 상태를 나타내는 STATE성분, 상태천이를 통한 Action천이를 제공하는 TRANS성분, 객체의 INSTANCE성분, 내부의 자료와 제어의 흐름 관계를 명시하는 FUN_FLOW, CON_FLOW성분, 초기토큰의 위치를 제공하는 MARK성분, 그리고 인터럽트를 처리하기 위한 EXCEPTION 성분으로 구성되어 있다. 객체의 각 성분에 대한 상세한 내부 설명은 다음과 같다. 먼저 EXTERnal부분은 현재 그 객체를 중심으로 외부 객체와의 관련성을 나타낸다. 이 객체의 성분들은 선택적으로 사용될 수 있다. 그의 성분 요소들을 각각 살펴보면,

▶ ISA : 이 성분은 객체간의 관련성을 나타낸다. 이는 특별화(Specialization)와 일반화(Generalization)를 의미한다. 즉 특정 객체의 is_a 관계를 의미한다.

▶ MCM : 이는 메시지 통신을 위한 메시지 큐로 구성된 통신 모듈이다. 외부와의 데이터 전달을 위해 사용된다. 이 모듈 구조는 약결합 통신과 강결합 통신을 제공하고 있다.

▶ IHM : 이는 이 객체와 통신하는 STORE 객체를 의미한다.

▶ OSM : 이는 객체가 외부와의 동기목적으로 사용

```
C_A_OBJECT ⟨object_name⟩
   EXTER : ISA  :⟨object_name⟩;
           MCM : IN(⟨obj_name⟩.⟨Q_name⟩) ||
                 IN(⟨obj_name⟩.⟨Q_name⟩),
                 OUT(⟨obj_name⟩.⟨Q_name⟩) ||
                 OUT(⟨obj_name⟩.⟨Q_name⟩);
           IHM : IN(⟨obj_name⟩.⟨Q_name⟩) ||
                 IN(⟨obj_name⟩.⟨Q_name⟩),
                 OUT(⟨obj_name⟩.⟨Q_name⟩) ||
                 OUT(⟨obj_name⟩.⟨Q_name⟩);
           OSM : IN(⟨sig_name⟩,⟨source_object⟩ ||
                 ⟨sig_name⟩,⟨source_object⟩),
                 OUT(⟨sig_name⟩,⟨destination_object⟩);
           FLOW : FM(⟨obj_name⟩,⟨msg_name⟩),
                  TO(⟨obj_name⟩,⟨msg_name⟩);
   INTER : GROUPED_BY ⟨object_name⟩;
           TIME : ⟨statrt_time, ending_time, period_time,
                  delay_time⟩;
           ATTR : ⟨type_name : type_value⟩
                  CON⟨constraints⟩;
           PRI : ⟨priority_level⟩;
           STATE_VAL : ⟨state_variable_name⟩ OF
                       ⟨state_value⟩ CON⟨constraints⟩;
           STATE : ⟨state_name⟩  PRE_FOR  ⟨predicate
                   formula(state formula or temporal
                   formula)⟩;
           TRANS : ⟨transition_name⟩ IN_COND
                   ⟨condition_predicate_name⟩ :
                   TR_COND  ⟨transition  condition⟩;
                   ACTION ⟨action_name⟩ TIME_part;
           INSTAN : ⟨instance_name⟩ | ⟨type_name⟩;
           MARK : ⟨start_state_name,
                  terminate_state_name⟩;
           FUN_FLOW :
                  ⟨module_name.Q_name.state_name⟩ |
                  ⟨object_name.transi_tion_name⟩,
                  ⟨object_name.transition_name |
                  module_na_me.Q_name.state_name⟩;
           CON_FLOW : ⟨obj_name.transition_name |
                      obj_name.state_name⟩,
                      ⟨obj_name.state_name |
                      obj_name.transition_name⟩;
           EXCEPTION : WHEN ⟨interrupt_name or
                       time_out⟩
                       HANDLER ⟨handler_name⟩;
ENDC A OBJECT
```

(그림 12) 병행(원자)객체 명세언어
(Fig. 12) Concurrent(Atomic) Object Spec. Language

된다. 신호(Signal)를 보내기도 하고 수신(Wait)하기도 한다.

▶ FLOW : 이는 페트리 넷에서 플레이스나 천이를 연결하는 아크의 부분집합으로 전체 객체들과의 메시지 흐름방향이나 제어의 흐름을 나타낸다. 이 흐름 관계는 먼저 처리되는 Process들의 순서가 명시될 수도 있다. 다음은 INTERnal부분으로 이의 요소들은 외부 객체들에는 감추어진 구조로 그 객체가 수행하는 내부구조를 의미한다.

▶ TIME : 이는 객체가 수행하는 시간 단위에 관한 성분으로, 세부시간으로 나누어진다. 즉, 이 객체가 실행되는 초기 시작시간(Start time)과 이 객체가 끝나야하는 한계시간인 종단시간(End time), 그리고 이 객체의 성질이 주기적인 실행을 행할 시의 주기시간(Period time)과 전, 후 프로세스와 관련하여 대기해야하는 하한시간인 대기시간(Wait time) 등을 기술할 수 있다[9].

▶ ATTR : 이는 매개변수(Attribute)성분으로 이 객체가 처리하는 메시지의 타입이나 사건 및 상태들을 나타낼 수 있다. 이는 원자 객체의 정적인 특성이며 상태 변수와 더불어 객체의 현재 상태를 나타낸다. 구성변수의 이름과 타입을 기술할 수 있다. 세부사항의 CON은 제한자(Constraints)로 변수사용의 제약이 따를 시 사용 가능하다.

▶ PRI : 이는 이 객체의 우선 순위 Level을 나타낸다.

▶ STATE_VAL : 이는 각각의 상태에 대한 전체의 상태변수로 천이 조건에 이용될 수 있다.

▶ STATE : 상태는 객체 내부의 전체 상태를 보여주는 페트리 넷의 플레이스로 표현되는 NonEmpty의 부분집합이다. 각 상태는 상태 변수들을 특성화하는 상태공식(State Formula)이나 시간이 추가된 시제공식(Temporal Formula)과 연관된다. 상태와 시제공식은 어떤 상태가 객체의 상태 값들로 사상(Mapping)되는 함수로 정의[21]된다.

▶ TRANS : 이는 각 객체가 보유하고 있는 Action 천이 테이블을 참조하여 Action천이의 조건이 참이될 때 미리 정의된 Action을 수행하고 동기의 역할을 하는 페트리 넷의 천이의 부분집합이다. 이 천이에 관련된 세부사항의 IN_COND는 NPN의 구조와 같이 천이에 들어가기 위한 입력 조건으로 *Ti(천이Ti의 입력 플레이스)의 허용할 수 있는 마킹(토큰의 수)을 의미한다. TR_COND는 위 조건이 True일 때 천이 가능한 조건으로서 상태공식이나 시제공식으로 정형화하게 기술할 수 있다. ACTION은 객체가 가지고 있는 기능들을 실제로 수행하는데 이는 외부에 대해서 사건만을(내부 처리 없이) 수행할 수도 있고, 사건의 구동 없이 내부의 처리를 수행할 수도 있다. 다양한 종류의 Action Type[23]을 지원한다.

▶ INSTAN : 이는 객체나 객체타입(클래스)의 인스턴스를 표현하기 위해 인스턴스 타입과 메시지 타입으로 구성되는데, 실시간의 특성상 여러 개의 인스턴스들을 한꺼번에 표현해야할 때에 사용할 수 있는 인스턴스(실 개체의 이름)타입과 또는 현재 전달하는 메시지 타입의 토큰의 마킹으로 표현할 수 있다.

▶ MARK : 이는 토큰의 초기위치와 종단위치를 나타내는데 이용된다. 초기 상태는 현재 시작하기 전의 상태이고, 종단 상태는 초기상태에 진입하기 전의 상태이다.

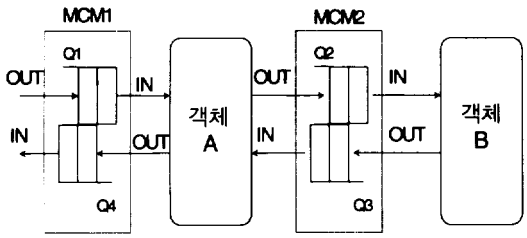
▶ FUN_FLOW, CON_FLOW : 이는 객체의 내부 통신절차(Procedure)를 보여주는 부분이다. 이 성분은 어떻게 객체가 메시지나 사건의 입력에 대해 내부의 상태 및 기능과 제어의 흐름이 이루어지는가를 보여준다. 이 성분은 메시지 전달을 위한 외부 통신 모듈과의 동기를 수행하면서 현 객체의 상태변화를 보여준다.

▶ EXCEPTION : 이는 객체의 수행 시에 발생할 수 있는 외부 인터럽트들이나 액션들의 시간초과(Time out)시 사용되는 성분이다. 이 성분은 하드 실시간 시스템에서 종종 구현된다[4].

객체 명세언어의 구조 중에서 EXTERnal의 MCM 성분을 예로 설명하자면, 다음 (그림13)과 같이 시스템이 구성될 시의 설명과 명세는 다음과 같다. MCM의 IN성분은 해당 병행객체를 대상으로 메시지가 들어오는 측(생산자)의 수동 객체 모듈을 의미하며, 구체적인 대상 큐 이름을 기술한다. OUT성분은 해당 병행 객체가 메시지를 보내는 측(소비자)의 수동모듈을 기술하며 대상이 되는 큐 이름을 기술한다. 모듈 내에 존재하는 양 방향의 Q1,Q2,Q3,Q4는 선택적으로 사용 가능하다. 또한 분산 시스템이 아닌 경우, 동시에 전송되는 메시지가 없을 경우는 병행기호 ||를 생략할 수 있다. 세부적인 설명은 적용예제 (그림18)을 참고하기 바란다. (그림13)의 경우로 시스템이 구성될 시 복합객체 A의 명세는 다음과 같다.

C_A_OBJECT 〈객체 A〉

```
EXTER : MCM : IN (⟨MCM1⟩.⟨Q1⟩) ||
              IN (⟨MCM2⟩.⟨Q3⟩),
              OUT (⟨MCM2⟩.⟨Q2⟩) ||
              OUT (⟨MCM1⟩.⟨Q4⟩);
ENDC_A_OBJECT
```



(그림 13) MCM성분 모사
(Fig. 13) Description of MCM

4. 적용 예제

지금까지 작성한 객체모델링과 명세의 사용 예를 적용해 보고자 한다. 사용 예로 선정한 모델은 로봇 제어기(Robot Controller)[5,6]이다. 이 예는 아직 행위분석 부분이 고려되지 않았기 때문에 시간에 민감하게 반응하는 시스템은 아니다. 그러나 지금까지 설명된 프로세스의 개발절차를 충분하게 보여줄 수 있기 때문에 예로 선정하였다. 아래에서 각 절차들을 하나씩 적용한다.

4.1 문제 정의(Problem Description)

로봇 제어기는 디지털 센서와 액추에이터와 상호 작용하며 6개의 모션(Motion)축(Axe)까지를 제어한다. 로봇 제어기는 제어 패널로부터 초기화된 프로그램을 실행함으로써 축(axe)과 디지털 I/O를 제어한다. 제어 패널은 많은 푸시(Push)버튼들과 프로그램 선택을 위한 선택자 스위치(Selector Switch)로 구성되어 진다.

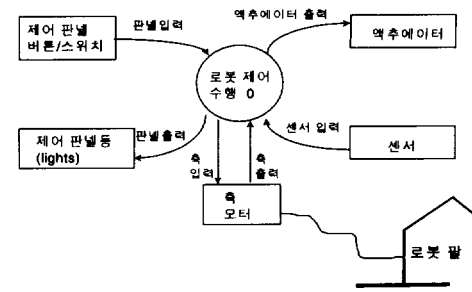
4.2. 시스템 모형단계

1. 시스템 문맥도(Context Diagram)의 작성

로봇 제어를 수행하는 소프트웨어 객체를 중심으로 한 시스템의 문맥도는 (그림14)와 같다.

2. 부 시스템의 작성

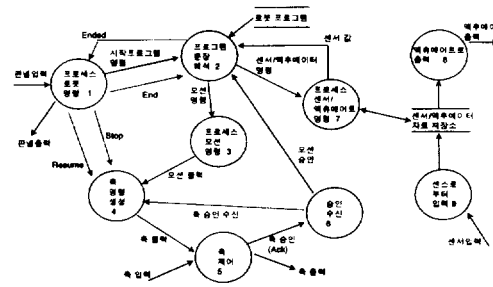
이 시스템은 부 시스템이 존재하지 않는 작은 규모의 시스템이다.



(그림 14) 로봇 제어기 시스템의 문맥도
(Fig. 14) Robot Controller Context Diagram

3. 데이터 흐름도의 작성

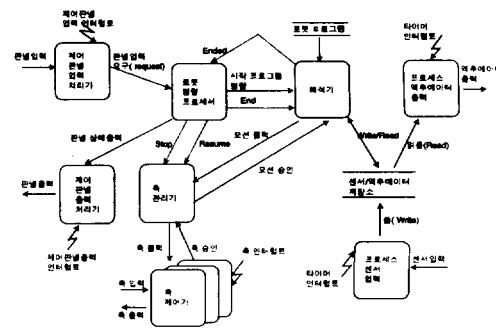
전반적인(Overall) 로봇 제어기 시스템의 DFD는 (그림15)와 같다.



(그림 15) 로봇 제어기의 DFD
(Fig. 15) Robot Controller DFD

4. 실시간 가이드 라인의 적용

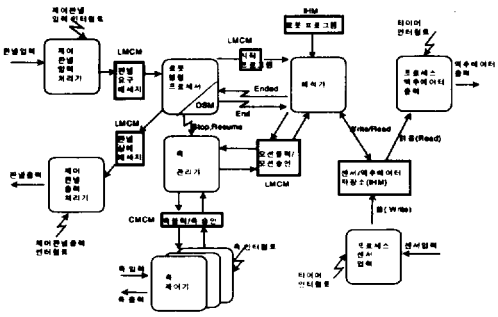
DARTS에서 제공하는 실시간 가이드 라인[6]을 적용하여 병행성있는 타스크를 구성한다. (그림16)은 이 기법을 적용하여 구성한 것이다.



(그림 16) 자료흐름과 제어흐름을 가진 병행 다스크
(Fig. 16) Concurrent Task with data and control flow

5. 통신 인터페이스 모듈 설정

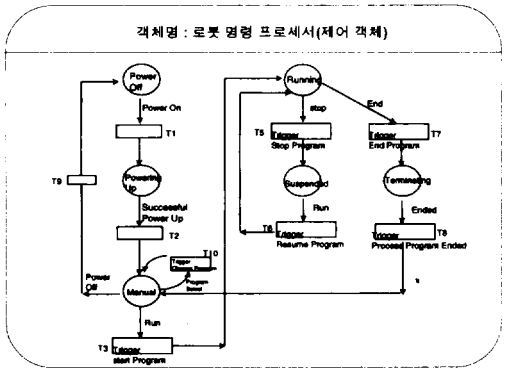
통신 인터페이스의 모듈은 먼저 비동기 통신을 위한 모듈과 동기통신을 위한 모듈 그리고 정보은닉 모듈과 사건을 발생시키는 동기모듈로 구성된다. 정의된 각 모듈들은 (그림17)이다.



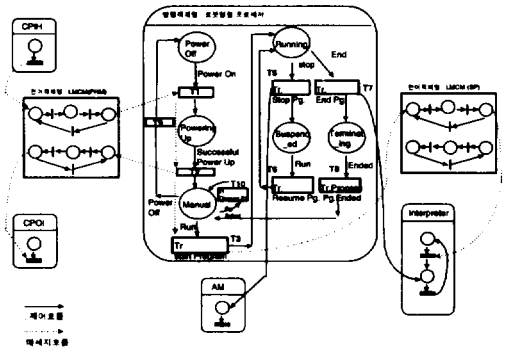
(그림 17) 통신 인터페이스 모듈
(Fig. 17) Comm. Interface Module

6. 선택된 타스크와 통신 인터페이스 모듈의 전체그림도 (그림17)과 같다. 아울러 타스크는 병행 객체로 인터페이스 모듈은 수동객체로 인식한다.

7. 인식된 각 병행객체와 각 수동객체에 대해 고 수준 페트리 넷으로 모델링한다. 즉, 상태 천이도를 작성한다. 여기에서는 초기 DFD에서 결정된 제어(타스크) 객체인 로봇명령 프로세서(RCP)의 예를 든다. 이 객체를 모델링한 결과는 다음 (그림18,19)를 참고하기 바란다.



(그림 18) 로봇 명령 프로세서의 상태 천이도
(Fig. 18) Robot Command Processor State Tran. Diagram



(그림 19) RCP관련 객체와의 통신행위
(Fig. 19) Comm. Behavior for RCP Object

8. 객체 명세

복합 객체인 로봇 제어기 시스템의 명세는 아래와 같이 기술 가능하다.

```
OBJECT 〈perform robot control〉
  EXTER : ISA : 〈비 존재〉
    FLOW : FM(〈control panel buttons/switch〉,
           〈panel input〉),
           TO(〈control panel lights〉, 〈panel output〉),
           FM(〈sensors〉, 〈sensor input〉),
           TO(〈axis motors〉, 〈axis output〉),
           FM(〈axis motors〉, 〈axis input〉),
           TO(〈actuators〉, 〈actuator output〉);
  INTER : GROUPD BY 〈 비존재〉;
ENDOBJECT
```

병행객체의 예는 다음과 같다.

```
C_OBJECT 〈robot command processor〉
  EXTER : ISA : 〈비존재〉;
    FLOW : FM(〈ctl panel input handler〉,〈panel input
           request〉),
           TO(〈ctl panel outpt handler〉,〈panel status
           output〉),
           TO(〈interpreter〉,〈start program
           command,end〉),
           TO(〈axismanager〉,〈start,resume〉);
    MCM : IN(〈panel request messages〉,〈Q1〉) ||
          (〈비존재〉);
          OUT(〈pannel status messages〉,〈Q2〉) ||
          (〈비존재〉),
          OUT(〈start program〉,〈Q1〉) || (〈비존재〉);
    IHM : IN 〈비존재〉, OUT 〈비존재〉;
    OSM : IN(〈ended〉,〈interpreter〉 || 〈비존재〉);
          OUT(〈stop or resume〉,〈axis manager〉),
          OUT(〈end〉,〈interpreter〉);
  INTER : GROUPD BY 〈atomic objects〉;
          TIME : 〈0t0m10ms : 100ms : : : 〉;
```

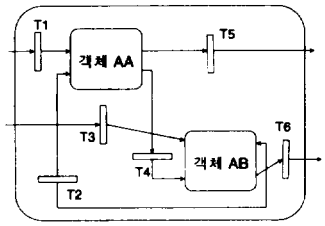
```
ATTR : 〈bool : true, false〉 CON 〈 true := 1, false := 0〉,
       〈event : power_on,successful_power_up, run,end,stop,ended,power_off,program_select〉 CON 〈비존재〉,
       〈action : enpowering,enmanual,change_program,start_progr_am,end_program, process_program_ended,stop_pro_gram,resume_program〉 CON 〈비존재〉;
PRI : 〈 2 〉 ;
STATE_VAL : 〈status〉 OF 〈waitting,processing〉
            CON 〈status := waiting, if have event = {power_on,succe_ssful_power_up,end,stop,ended,power_off,program_select},
            status := processing, if have event = { run } 〉;
STATE : 〈init,powered_off,powering_up,manual,running,suspended,terminating〉
   PRE_FOR 〈 power_off =((status(waitting(powered_off)) ∨ (init) = 1),
             powering_up =((status(waiting(power_on))) =1),
             manual = ((status(waiting(successful_power_up)∧(ended)∧(program_select)) = 1),
             terminating = ((status(waiting(end))) = 1),
             suspended = ((status(waiting(stop))) = 1),
             running = ((status(processing(run))) = 1)〉;
TRANS : 〈T1〉 IN_COND 〈 m(powered_off) ≤ 1〉;
        TR_COND 〈 ((status(waitting(powered_off)) ∨ (init) = 1) 〉;
        ACTION 〈 □((enpowering(power_on)) → ○(powe_ring_up)) TIME 〈 : : : :〉;
    〈T2〉 IN_COND 〈 m(powering_up) ≤ 1〉;
        TR_COND 〈 ((status(waiting(power_on)))=1) 〉;
        ACTION 〈 □((enmanual(succ_power_up)) → ○(man_ual)) TIME 〈 : : : :〉;
    〈T3〉 IN_COND 〈 m(manual) ≤ 3〉;
        TR_COND 〈 ((status(waiting(successful_power_up)∧(ended)∧(program_select)) = 1) 〉;
        ACTION 〈 □((start_program(run)) → ○(running))〉;〈T5〉IN_COND 〈 m(running) ≤ 2 〉;
        TR_COND 〈 ((status(processing(run))) = 1) 〉;
        ACTION 〈 □((stop_program(stop)) → ○(suspended))〉;
    〈T6〉 IN_COND 〈 m(suspended) ≤ 1 〉 ;
        TR_COND 〈 ((status(waiting(stop))) = 1) 〉;
        ACTION 〈 □((resume_program(run) →○(running))〉;
    〈T7〉 INCOND 〈 m(running) ≤ 2 〉;
        TR_COND 〈 ((status(processing(run))) = 1) 〉;
        ACTION 〈 □((end_program(end) → ○(terminating))〉;
    〈T8〉 INCOND 〈 m(terminating) ≤ 1 〉;
        TR_COND 〈 ((status(waiting(end))) = 1) 〉;
        ACTION 〈 □((process_program_ended(ended) → ○(manual)) 〉 TIME 〈 : : : :〉;
    〈T9〉 INCOND 〈 m(manual) ≤ 3 〉;
        TR_COND 〈 ((status(waiting(successful_power_up)∧(ended)∧(program_select)) = 1) 〉;
        ACTION 〈 □((enpower_off(power_off) → ○(powered_off) 〉 TIME 〈 : : : :〉;
    〈T10〉 INCOND 〈 m(manual) ≤ 3 〉;
        TR_COND 〈 ((status(waiting(successful_power_up)∧(ended)∧(program_select)) = 1) 〉;
        ACTION 〈 □((change_program(program_select) → ○(manual)〉 TIME 〈 : : : :〉;
INSTAN : 〈 비존재〉;
MARK : 〈powered off, manual〉;
FUN_FLOW : 〈prsm.Q1.full | rcp.t1〉 | 〈rcp.t2 | prsm.Q2.empty〉, 〈rcp.t3 | sp.Q1.empty〉;
CON_FLOW : 〈rcp.t5 | am.any_state1〉, 〈rcp.t7 | interpreter.any_state2〉,
           〈interpreter.any_state3 | rcp.t8〉;
EXCEPTION : WHEN 〈power_crash〉
            HANDLER 〈break〉;
ENDOBJECT
```

9. 행위분석

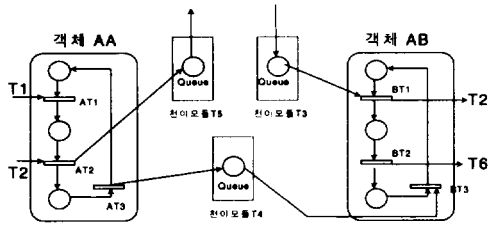
9.1 계층적 분석

복잡한 시스템을 처리하기 위해 본 논문에서 제시된 시스템 모델링 방법은 복합객체에서 원자객체로의 행위 분석이 가능하다. 원하는 추상화의 레벨에서의 복합객체의 객체 구조화 가이드 라인을 따라 병행객체를 분류하고, 병행 객체의 내부행위를 분석하여 원자 객체를 정의할 수있다. 아울러 메시지의 성질과 흐름을 분석하

여 수동객체를 정의할 수 있다. 수동객체와 합성된 객체 구조도를 따르면 전체적인 객체의 내, 외부의 행위들을 분석 가능하다. (그림20,21)을 참조하면 복합객체 A를 시스템이라 하면, A = {AA, AB, $T_{i(i=1..6)}$}로 구성되며 AA = {$State_{i(i=1..3)}$, $AT_{i(i=1..3)}$}로 구성된다. $T_{i(i=1..6)}$ = {$Queue_{i(i=1,2)}$,$t_{i(i=1..6)}$}($t_{i(i=1..6)}$는 천이모듈 내부의 액션천이)로 앞의 (그림6)이나 (그림8)과 같이 구성된다.



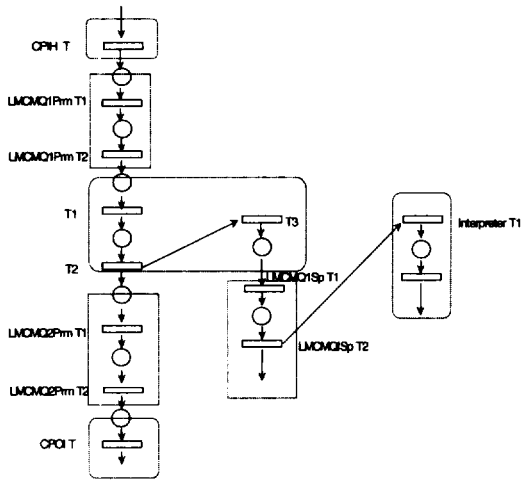
(그림 20) 복합객체 A
(Fig. 20) Composite Object A



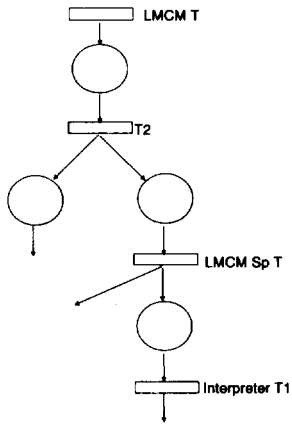
(그림 21) 객체A의 내부구조
(Fig. 21) Object A's Structure

9.2 동기 분석

이는 통신인터페이스인 천이 모듈들과의 동기분석이다. 먼저 원자 객체 내부의 행위가 자연스럽게 도출되며, 외부천이와의 도달성 그래프가 생성된다. 이 도달성 그래프[8,18]를 통하여 데드락(Dead Lock)발생 여부가 나타난다. 실제 외부천이 모듈 자체도 내부의 액션천이를 통하여 병행객체나 원자객체의 액션천이와 동기를이룸으로서액션천이 순서과정이 나타나게 된다. 이 과정에서 같은 상태나 천이를 통과하는 메시지들만이 동기제어가 우선 필요하게 되며 아울러, 이러한 부분들을 제외한 상태나 천이들을 감소시켜 도달성 그래프의 크기를 줄일 수 있다. 병행시스템의 성질들을 검증하기위한 다양한, 정적인 분석기법들이 제시되고 있다[18]. 앞의 (그림19)를 참고하여 생성된 도달성 그래프는 (그림22)와 같다. 이 그래프에서 보이는 것과 같이 T2천이와 LMCMQ1Sp T2에서 동기가 필요함을 알 수 있다. 그리고 메시지의 최초 흐름 방향에서부터 전달되는 경로가 전달되는 곳까지가 폐곡선(Closed Loop)을 형성하지 않으므로 메시지의 전달 시 데드락이 발생하지 않음을 알 수 있다. (그림21)의 경우는 T2에서 T1로 그리고 T5에서 T3로의 천이 관계가 발생한다면 데드락이 발생한다. 아울러 (그림22)의 그래프는 (그림23)과 같이 상태 축소가 가능하다.



(그림 22) 도달성 그래프
(Fig. 22) Reachiability Graph



(그림 23) 축소된 도달성 그래프
(Fig. 23) Reduced Reachiability Graph

5. 결 론

본 논문에서는 DARTS방법에 기초하여 실시간 소프트웨어 시스템 모형화를 위한 방법과 명세언어를 제시하였다. 본 논문은 복잡한 실시간 시스템의 모형화를 위해 객체에 기반하며, 객체의 천이액션을 분석하기 위해서 마련된 시제논리는 객체 내, 외부의 행위들에 대해서 정확성을 기할 수 있도록 제공되었다. 아울러 객체에 기반을 둠으로써 효율적인 명세서의 재사용을 통하여 생산성을 높일 수 있다. 또한 작성된 모형화 방법이나 명세서를 통하여 시스템이 도달 가능한 상태 그래프를 통하여 분석을 용이하도록 하였다. 본 논문은 현재 개발중인 OARTS 도구의 첫 번째 부분이다. 본 논문은 향후 정형화된 천이사건 순서도와 결합되어지며 본 논문에서 정형화되지 못한 NPN구조도 더욱 확장되어 검증 가능하도록 수학적 정형화에 기초하여 보강되어 질 것이다.

참 고 문 헌

[1] Babak,S., and Patricia,J., "An OO Project Management Strategy" IEEE computer, pp. 33-38, Sep. 1996.

[2] Billing,J., et al., "PROTEAN : A High-level Petri Net Tool for the Specification and Verification of Communication Protocols" IEEE Trans. SE, Vol.14, No.3, pp.301-16, March, 1988.

[3] Booch,G., "Object-Oriented Design with Applications" Menlo Park pub, CA, 1991.

[4] David,S., and Pradeep,k., "Mechanisms for Detecting and Handling Timing Errors" CACM, Vol.40, No.1, pp.87-93, Jan., 1997.

[5] Gomaa,H., "Software Design Methods for Concurrent and Real-time Systems" A/W pub.1993.

[6] Gomaa,H., "A Software Design Method for Real-time Systems" CACM, Vol.27, pp.938-949, Sep., 1984.

[7] Harel,D., "On Visual Formalisms" CACM Vol.31, No.5, pp.514-530, May, 1988.

[8] Huber,P., et al., "Reachability Trees for High-Level Petri Nets" Theoretical CS, pp. 261-292, 1986.

[9] Ishikawa,Y., et al., "Object-Oriented Real-time Language Design: Constructs for Timing Constructs" TR CMU-CS-90-111, March, 1990.

[10] Jackson,M., "System Development" Prentice-Hall, 1983.

[11] Jeffrey,J.P., et al., "Parallel Evaluation of Software Architecture Specifications" CACM, Vol.40, No.1, pp.83-86, Jan., 1997.

[12] Jonathan, P., et al., "An Invitation to Formal methods" IEEE computer, pp.16-30, April. 1996.

[13] Joseph,A., et al., "Safety-Critical Systems Bulit with COTS" IEEE computer, pp.54-60, Nov., 1996.

[14] K.H.Kim and Kopetz,H., "A Real-Time Object Model RTO.k and an Experimental Investigation of its Potentials," Proc. COMPSAC'94, Taipei, pp.392-402, Nov., 1994.

[15] K.H.Kim and Chittur,S., "Fault-Tolerant Real-Time Objects" CACM, Vol.40, No.1, pp.75-82, Jan., 1997.

[16] Manna,Z and Pneueli, "Verification of Concurrent Programs, part1: the temporal frame work" TR STAN-CS-81-836, Dept of CS, Stanford Univ, June, 1981.

[17] Marlin,D.F, Kuldeep,K., and Vaishnavi,V.K, "Strategies for Incorporating Formal Specifications in Software Development" CACM, Vol.37, No.10, pp.74-86, Oct., 1994.

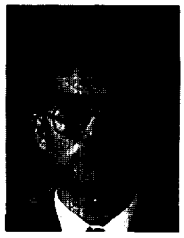
[18] Matthew,B. and Lori,A., "A Compact Petri Net Representation and Its Implications for Analysis" IEEE Trans. on SE, Vol.22, No. 11, pp.794-811, Nov., 1996.

[19] Moser,L. et al., "A Graphical Environment for the Design of Concurrent Real-Time Systems" ACM Trans. on SE and Methodology, Vol.6, No.1, pp.31-79, Jan., 1997.

[20] Nielsn,K., "Designing Large Real-time Systems with ADA" M/H pub.1988.

[21] Ostroff,J.S., "Temporal Logic for Real-time Systems" research studies press, 1989.

[22] Ostroff,J.S., "Statetime - a visual toolset for design and verification of real-time systems" Tech. Rep. CS-ETR-94-07, Department of CS, York Univ., 1994.
[23] Pamela,Z., and Michael,J., "Four Dark Corners of Requirements Engineering" ACM Trans. on SE and Methodology, Vol.6, No.1 pp.1-30, Jan. 1997.
[24] Parnas,D., "On the Criteria for Decomposing a System into Modules" CACM, pp.1053-1058, Dec., 1972.
[25] Parnas,D.L., et al., "The Modular Structure of Complex Systems" Proc. International Conf. on SE, March, 1984.
[26] Peter,G.L., et al., "Applying Formal Specification in Industry" IEEE software, pp.48-56, May 1996.
[27] Peterson,J., "Petri Net Theory and the Modeling of Systems" P/H pub. 1981.
[28] Rebecca, H., "A Time-Sensitive Object Model for Real-Time Systems" ACM Trans. on SE and Methodology, Vol.4, No.3, pp.287-317, July, 1995.
[29] Rumbaugh,J., et al.,"Object-Oriented Modeling and Design" P/H pub. 1991.
[30] Selic,B., Gullekson,G., and Ward,P., "Real-Time Object-Oriented Modeling" J/W & Sons, Inc., 1994.
[31] Sergio,B., et al., "Supporting Cooperation in the SPADE-1 Environment" IEEE Trans. on SE, Vol.22, No.12, pp.841-863, Dec., 1996.
[32] "Statemate 4.5 User Reference Manual", i-Logix Inc., Burlington. MA, Aug., 1992.
[33] Ward,P "Structured Development for Real-time Systems", Vol.4, P/H pub. 1985.
[34] Wegner,P.,"Dimensions of Object-based Language Design" Proc. of ACM OOPSLA, 1987.
[35] Y. K. Lee., "Object-oriented High-level Petri Net for Manufacturing System Modeling" TR, KAIST. Management Science. 1991.
[36] Jiazhong,Z., and Zhijian,W., "NDHORM: An OO Approach to Requirements Modeling" ACM SIGSOFT, SE Notes, Vol.21, No.5, pp.65-69, Sep., 1996.
[37] 김정술 "DARTS 방법의 행위 명세언어" 포항공과대학교, 석사학위논문, 1994.



김 정 술

1985년 2월 한양대학교 전자공학과 졸업(학사)
1994년 2월 포항공과대학교 정보통신학과(공학석사)
1998년 2월 영남대학교 컴퓨터공학과(박사수료)
1988년~1991년 L.G전자 PC/MNT설계실 근무
1994년~1996년 한진정보통신(주) SI사업부 근무
관심분야 : 소프트웨어공학(실시간 시스템, Formal Method, 재사용)

강 병 욱

1970년 영남대학교 전기공학과(공학사)
1977년 영남대학교 전자공학과(공학석사)
1994년 경북대학교 전자공학과(공학박사)
1979년~현재 영남대학교 전산공학과 교수
관심분야 : 소프트웨어 공학, 프로그래밍 언어, 데이터 압축