PtolemyII의 CCS 도메인
(Calculus of Communicating Systems Domain in PtolemyII)

황 혜 정 † 김 윤 정 †† 남 기 혁 ††† 김 일 곤 †††† 최 진 영 †††††
(Hye-Jung Hwang) (Ki-Hyuk Nam) (Yoon-Jeong Kim) (Il-Gon Kim) (Jin-Young Choi)

요 약 톨레미II는 내장형 시스템과 같이 이질적 성질을 가진 병렬 시스템을 모델링하고 디자인 할 수 있는 환경을 지원해준다. 톨레미II는 여러 개의 도메인을 가지고 있다. 도메인은 각 시스템의 구성 요소들 간의 통신 방법을 결정하는 물리적인 규칙이다. 톨레미II에는 PetriNet, Timed Multitasking, SR 등 11개의 도메인이 존재한다. 시스템의 구성요소들은 그 특성에 맞는 도메인을 사용하여 명세되어질 수 있다. 톨레미II는 특히 정형적 도메인으로 프로세스 알제브라 언어의 일종인 CSP를 가지고 있다. 그러나 CCS는 도메인으로 구현되지 않았다. CCS는 프로세스 알제브라의 일종으로서 정형적으로 병렬시스템을 명세하고 검증할 수 있는 언어이다. 따라서 본 논문에서는 CCS도메인을 구현함으로써 톨레미II를 사용하는 개발자가 톨레미II에서 사용하고 있는 동일한 모델링 패턴을 사용하면서 CCS의 정형적 의미론을 바탕으로 하여 시스템 명세를 할 수 있게 하였다. 이것은 톨레미II의 도메인의 다양성을 가지고 옴으로써 톨레미II의 표현력과 가용력을 높였다. 본 논문에서는 톨레미II에 구현된 CCS 도메인의 구조와 구현 방법을 설명하겠다.
키워드 : 톨레미II , CCS도메인, 프로세스 알제브라

Abstract PtolemyII is an environment that supports heterogeneous modeling and design of concurrent systems such as embedded system. PtolemyII has several Domains which are physical rules to determine the way of communicating between components. PtolemyII has 11 domains such as PetriNet, Timed Multitasking, SR etc. Components of System can be specified using appropriate domains for their properties. Communicating Sequential Processes(CSP) is implemented as formally designed CSP domain, in PtolemyII. But CCS didn't be implemented as a domain. It is a kind of Process Algebra language which can be used for specifying and verifying concurrent systems formally. Thus, in this paper we implemented CCS domain. And that permitted developers using PtolemyII to use the same modeling pattern used in PtolemyII and to make system specifications in the base of the formal semantics of CCS. This caused the diversity of PtolemyII domains and the power of expression was improved. This paper will explain the structure of CCS domain implemented in PtolemyII and the way of implementing it.
Key words : PtolemyII, CCS domain, Process Algebra

1. 서 론

내장형 시스템은 전형적으로 아날로그 환경과 서로 상호작용 하는 디지털 프로그램의 모음으로 구성되어 있다. 이것은 범용적인 목적을 가진 워크스테이션이나 테스크탑 컴퓨터 등과는 달리 특정 기능을 수행하기 위해, 컴퓨터 하드웨어와 소프트웨어가 조합된 전자제어 시스템을 일컫는다. 의료제품에서부터 가전 제품, 항공기 제어 시스템, 원자력 발전소에 이르기까지 폭넓은 범위에서 이미 사용되고 있다.

내장형 시스템은 DSP(Digital Signal Processor)와 같은 프로그램이 가능한 계산 구성단위(computing unit) 뿐만이 아니라 ASIC과 FPGA로 구성되어 있다. 특히 내장형 시스템의 핵심 부분에는 프로그램이 가능한 마이크로 컨트롤러(micro-controller)가 내장되어 있다. 이렇게 구성된 시스템을 디자인 할 때는 어떠한 부분이 소프트웨어로 구현될 것이며 다른 어떠한 부분을 하드

† 비 회 원 : 삼성전자 기술총괄 기술전략실 모바일 솔루션 연구실
hjhwang@formal.korea.ac.kr
†† 종신회원 : 서울여자대학교 정보통신공학부 교수
yjkim@swu.ac.kr
††† 정 회 원 : 한국전자통신연구원 디지털홈 연구단
khnam@formal.korea.ac.kr
†††† 비 회 원 : 고려대학교 컴퓨터학과
igkim@formal.korea.ac.kr
††††† 종신회원 : 고려대학교 컴퓨터학과 교수
choi@formal.korea.ac.kr
논문접수 : 2003년 9월 18일
심사완료 : 2004년 3월 30일

웨어로 설계할 것인지를 결정하는 작업이 중요한 부분이다[1]. 이렇게 구성된 내장형 시스템은 원천적으로 이질적인 특성을 가지고 있다. 그 때문에 하나의 디자인 방법론으로는 시스템 전체를 완전히 디자인 할 수 없다[2]. 따라서 서로 다른 디자인 방법론을 적용하고 이렇게 적용된 디자인 방법론들을 상호 연결하는 것이 필요하다.

내장형 시스템 중 실시간 내장형 시스템(realtime embedded system)의 경우는 시스템의 주변환경이 요구하는 속도에 맞추어서 지속적으로 환경과 상호작용을 해야만 한다. 이것은 환경이 요구하는 속도와 관련 없이 시스템 자체의 속도로 반응하는 대화형 시스템(interactive system)과는 대조적이며 성능보다는 정확성과 안전성이 더욱 요구되어 지는 생명과 관련된 중요한 환경에서 자주 쓰인다. 예를 들면 원자력 발전소나 항공기 제어 시스템과 같은 것이 이에 해당한다. 이러한 시스템의 경우는 안전성이나 정확성이 만족되어지지 않을 때는 심각한 재해를 가지고 오게 된다. 이러한 시스템에 대한 정형 검증(formal verification)과 자동화 도구를 사용한 구현은 일반적인 테스트보다는 안전성을 보장하는 좋은 방법이다. 따라서 시스템을 하드웨어와 소프트웨어로 나누기전, 추상화된 높은 레벨에서 시스템의 행위를 묘사할 때는, 반드시 하나 이상의 정형 모델(formal model)을 기반으로 해야 한다[1]. 즉 정형 명세(formal specification)가 되어야하는 것이다.

기존의 오토마타는 병렬 시스템(concurrent system)을 정확히 표현하는데 한계가 있었다. 이러한 한계 때문에 병렬적으로 통신하는 시스템을 묘사하고 설계하기 위해 CSP(Communicating System of Process)[3], CCS(Caculus of Communicating Systems)[4], ACP (Algebra of Communicating Processes)[5]와 같은 시간개념이 없는 여러 프로세스 알제브라 언어가 개발이 되었다. 또한 시간개념과 자원의 개념이 추가된 ACSR (Algebra of Communicating Shared Resources)[6] 이 개발되었다. 이러한 프로세스 알제브라 언어는 프로토콜이나 시스템을 보다 정확하게 정형적으로 명세할 수 있도록 의미론과 문법을 제공한다. 그러나 이러한 언어는 각 언어의 전문가적 지식을 가지고 있지 않는 개발자는 다시 이 언어들의 문법과 의미론을 익힌 뒤에야 정형적 명세를 할 수 있다.

본 논문에서는 프로세스 알제브라 언어인 A Calculus of Communicating System을 톨레미II의 정형적 MoC (Model of computation)로 선택하여 Java로 CCS 의미론을 구현하여 톨레미II에 CCS 도메인을 추가하였다. 이것은 기존의 개발자들이 actor-oriented 디자인[7]을 하는 톨레미II의 디자인 방법을 따라 기존의 프로세스 알제브라 언어들의 문법을 익히지 않고도 모델링하고 시뮬레이션 할 수 있다는 것을 의미한다. 또한 톨레미II에 CCS도메인을 추가함으로써 정형적 명세의 다양성을 높였다. 이것은 내장형 시스템 디자인, 모델링 및 시뮬레이션 도구인 톨레미II의 정형적 명세의 가용력을 높일 수 있다.

다음과 같은 순서로 톨레미II의 A Calculus of Communicating System(CCS) 도메인 구현을 위하여 CCS [4]와 CSP[3]의 의미론을 고찰하고 톨레미II의 CCS 도메인의 구현 방법을 설명하도록 하겠다. 톨레미II에 대한 소개는 2 절에서 그리고 프로세스 알제브라 언어인 CSP와 CCS에 대해서 3절에서 의미론을 간단히 보고 그 차이점을 보도록 하겠다. 그리고 4절과 5절에서는 기존의 톨레미II의 CSP도메인을 고찰해 보고 전체적인 CCS 도메인 구현방법 및 소프트웨어 구조와 예제에 대한 소개하도록 하겠다. 마지막으로 6절에서 결론을, 6절에서 향후 연구방향을 제시하고자 한다.

2. 톨레미II 소개

U.C Berkeley에서 내장형 시스템을 정형적인 이론을 기반으로하여 설계하고, 이질적인 의미(semantic)들을 이용하여 설계된 내장형 시스템을 통합하여, 디자인 할 수 있는 도구인 톨레미II를 개발하였다[8]. 톨레미II는 여러 가지 다른 의미론을 가진 이른바 도메인이라고도 불려지는 여러 개의 Model of Computation(MoC)을 가지고 있으며 이중에서도 프로세스 알제브라 언어의 하나인 Communicating Sequential Processes(CSP)[3]를 가장 정형적인 MoC중 하나로 채택하여 구현하였다.

모델링(Modeling)이라는 것은 시스템이나 하위 시스템을 정형적으로 표현하는 행위를 일컫는다. 반면 디자인(Design)은 시스템이나 하위 시스템을 정의하는 행위를 일컫는 것이다[8]. 어떠한 시스템을 설계하는 과정에서 디자인과 모델링 하는 과정은 필수적이며 또한 개발단계시에 중요한 작업이다. 내장형 시스템과 같은 이질적인 특성을 가진 시스템에서의 모델링을 디자인 할 경우, 하나의 의미론만을 제공하는 도구로는 구현하고자 하는 시스템을 완벽하게 구현하는 것이 불가능하다. 예를 들면 Statechart의 경우는 유한 상태 기계 (Finite State Machine)의 의미론을 기반으로 하여서 모델을 명세할 수 있으나, 실세계의 지속되는 시간을 모델에 명세할 수 없다.

보통 시스템을 개발할 때 높은 레벨(high-level)에서 시스템의 행위를 디자인 할수록 개발자나 사용자의 이해를 보다 쉽게 할 수 있도록 시스템의 행위가 많이 추상화 되어 진다. 그러나 모델을 개발하는 데 드는 비용은 추상화 레벨이 낮아질수록 차츰 더 많이 들게 되어

진다. 이는 개발 할 때 모델의 요구사항들이 여러 요인들로 인해 변경되어 질 수 있기 때문이다[9].

이러한 개발 단계에 생길 수 있는 문제점들을 보완하고 보다 정확하고 효율적으로 시스템을 디자인하고 모델링하기 위해 만들어진 것이 톨레미II이다. 톨레미II 의 큰 장점은 여러 가지 프레임워크를 하나의 도구에서 구현할 수 있고, 각각 다른 프레임워크로 디자인된 모델을 계층적으로 연결하여 시스템을 하나의 도구를 사용하여서 디자인하고 모델링하며 시뮬레이션까지 설계과정에서 할 수 있다는 것이다. 현재 톨레미II에서는 총 11개[8]의 프레임워크를 제공하고 있으며 현재 매뉴얼에 추가되지 않고 도구에만 추가된 도메인 2개를 합하여 총 13개의 다른 의미론(semantic)을 가진 프레임워크들이 존재한다. 프레임워크는 다른 말로 '도메인'이라고도 일컬어지는데, 이러한 여러 도메인들은 'Model of Computation'이며 이것은 각 구성 요소들 사이에 계산(computation)하는데 있어서의 물리적인 규칙을 말한다. 이 예로 Discrete Event, PetriNet, Countinuous Time 등을 들 수 있다. 이것은 시스템을 디자인할 때 편리성을 가져올 뿐만이 아니라 한 시스템 안의 이질적인 특성을 가진 여러 부분들에 최적의 Model of Computation을 적용함으로써 시스템을 디자인 할 때 질적인 부분의 향상을 가질 수 있다. 톨레미II는 시스템을 명세할 때 기본적으로 component-based design을 가능하게 한다. 이것은 시스템의 행위별로 나누어서 행위의 추상화를 계층적[그림 6]으로 재정의 할 수 있기 때문에 개발시에 생길 수 있는 오류를 쉽게 발견하고 수정할 수 있다. 따라서 시스템 개발시의 개발비용을 단축시켜 줄 수 있다.

톨레미II는 기본적으로 2가지 방법으로 모델을 디자인하고 모델링 할 수 있게 한다. 첫째는 톨레미II에서 제공하고 있는 커널을 사용하여서 개발자가 직접 자바로 코딩하여 모델을 디자인 하는 방법이고, 두 번째는 그래픽 에디터 Vergil 사용하여서 모델을 디자인하고 모델링하고 시뮬레이션 하는 방법이다. Vergil[8]은 개발자가 구현된 모델을 actor 단위로 나누어 그래픽적으로 디자인 할 수 있게 함으로써 시스템 개발자에게 편리하고 친숙한 환경을 제공한다.

3. Process Algebra Languages

CCS[3]와 CSP[4]는 프로세스 알제브라 언어로써 서로 비슷한 시기에 Milner와 Hoare에 의해서 개발되었다. 이 두언어는 표현력은 같지만 그 문법이나 법칙 등의 차이를 보인다. 이 절에서는 두 언어의 차이점을 밝혀보고자 한다.

3.1 CCS(A Calculus of Communicating Systems)의 개요

프로세스 알제브라는 어떤 동기화 메커니즘을 통하여 프로세스 간의 상호작용을 하는 부분에 대해 건설적인 방법을 제시함으로써 병렬시스템(concurrent system)을 잘 명세하고 검증할 수 있는 방법을 제공한다. CCS[4]는 프로세스 알제브라의 언어로써 Milner에 의해 1989년에 개발되었다.

CCS라는 용어는 프로세스들을 의미한다. 프로세스는 자신 고유의 성질을 가지며, 프로세스 자신은 자신이 상호 작용을 할 수 있는 능력에 의해서 통신이 제한되어 진다. 프로세스끼리 혹은 환경과 프로세스 간에 상호작용은 일어난다. 각 프로세스들은 통신 활동을 하는데 이것을 action이라고 일컫는다.

셀 수 있고 더 이상 나눠 질 수 없는 action들의 집합 Act = {α, β...}은 action들의 집합 A와 co-action들의 집합 $\overline{A}$ 그리고 외부에서 관찰할 수 없는 action tau(τ)를 포함한다. 이 집합은 프로세스간 통신을 나타낸다. A와 $\overline{A}$는 모두 셀 수 있는 무한한 집합으로 가정되어 진다. agent는 다음과 같은 문법에 의해서 행위가 정의될 수 있다.

$$E ::= A \mid \alpha.E \mid E+E \mid E|E \mid E\backslash L \mid E[f]$$

여기서 f는 relabelling function[9]을 의미한다. 결합자(combinator)의 의미는 다음과 같다. action은 Prefix를 사용하여서 표현되어진다. Summation(Σ)은 agent E_i,(i∈I)들의 특성을 분리시켜 표현한다. 하나의 agent가 어떤 action을 수행하면 다른 agent는 자신의 action을 수행하지 못하게 된다. 즉 어떤 action을 수행한 하나의 agent만이 선택이 된다. Summation은 가능하면 무한적으로 그리고 임의적으로 프로세스를 취한다. Summation은 다음과 같은 형태중 하나를 취한다. i) 어떤 action도 취하지 않는 Deadlock agent 0(0은 $\sum_{i\in\emptyset} E_i$) ii) 두개의 summand만을 취하는 binary Summation, E1 + E2는 $\sum_{i\in\{1,2\}} Ei$ iii) set에 대한 indexed Summation $\sum_{i\in\{e\}\cup I} E_i$=E(e)+$\sum_{i\in I} E_i$ 그리고 자연수에 대한 무한 summation 이 있다. Parallel Composition (|)은 동시성을 표현하는데 사용되어 진다. E | F는 E와 F는 각각 병렬적으로 진행되어 진다. 그러나 서로 상호작용을 한다. Relabelling combinator ([])는 port label의 이름을 다시 붙이는데 사용되어 진다. 예를 들면 E[f]는 f로 나타내어진 action만 이름이 다시 붙여지는 것을 제외하고는 E처럼 행위를 한다. Restriction(\)은 내부적인 port 에 대해 사용되어 진다. 예를 들면 E\L의 경우는 어떤 action $l \in L \cup \overline{L}$를 수행하는 것을 제외하고는 E처럼 행위를 한다.

CCS의 semantics를 살펴보자. 프로세스는 labelled transition System을 통하여서 의미를 부여받게 된다. labelled transition System (ε, *Act*, $\{\xrightarrow{a} : a \in Act\}$)에서 $\xrightarrow{a}$는 다음의 inference transition rule에 의해서 주어지는 가장 작은 transition relation이다. inference rule은 다음과 같다.

$$\text{Act}\ \frac{-}{a.E \xrightarrow{a} E} \qquad \text{Com}_1\ \frac{E \xrightarrow{a} E'}{(E \mid F) \xrightarrow{a} (E' \mid F)}$$

$$\text{Com}_2\ \frac{F \xrightarrow{(a)} F'}{(E \mid F) \xrightarrow{(a)} (E \mid F')} \qquad \text{Com}_3\ \frac{E \xrightarrow{l} E',\ F \xrightarrow{\bar{l}} F'}{(E \mid F) \xrightarrow{\tau} (E' \mid F')}$$

$$\text{Con}\ \frac{E \xrightarrow{a} E'}{A : E \xrightarrow{a} E'}\ A \stackrel{def}{=} E \qquad \text{Sum}_j\ \frac{E \xrightarrow{\alpha} E'}{\sum_{i \in I} E_i \xrightarrow{\alpha} E'}\ j \in I$$

$$\text{Rel}\ \frac{P \xrightarrow{a} P'}{P[f] \xrightarrow{f(a)} P'[f]} \qquad \text{Res}\ \frac{E \xrightarrow{\alpha} E'}{E \backslash L \xrightarrow{\alpha} E' \backslash L}\ (\alpha, \bar{\alpha} \notin F)$$

3.2 CSP(Communicating Sequential Processes)의 개요

CSP[3]는 Hoare에 의해서 1980년에 개발된 프로세스 알제브라 언어로써 통신 프로토콜의 정형적 설계 및 검증을 위해서 개발되었다. CSP의 기본적인 연산으로는 외부 *choice* (□), 내부 *choice* (⊓), *concurrency* (||), *interleaving* (|||), *hiding* (\)과 *composition* (;)이 있다. CSP에서 프로세스간의 통신과 프로세스의 행위의 기본 단위는 이벤트이며 이벤트는 더 이상 나눠지지 않는 기본 단위로써 atomic이거나 행위와 관련된 데이타를 포함할 수 있다.

프로세스는 행위를 가지는 객체의 행동이며 이벤트와 연관된다[10]. 또, 프로세스 'P'에 연관되는 모든 이벤트들의 집합을 'α(P)'라고 표시하며, 알파벳 혹은 인터페이스로 일컬어진다. *hiding*(\) 은 이러한 이벤트를 외부에서 보이지 않게 가리는 것이다.

CSP의 중요한 기본적 연산의 하나인 *choice*를 살펴보자. 외부(external) *choice* (□)는 프로세스의 외부 환경에 의해서 이벤트가 결정되어지는 것을 의미한다. 반면에 내부(internal) *choice* (⊓)는 외부 환경과는 독립적으로 프로세스 내부에서 발생하는 이벤트가 임의로 결정되어 지고, 이렇게 결정된 이벤트는 프로세스 외부에서 보이지 않는다. 내부(internal) *choice* (⊓)의 경우는 비결정성(non-determinism)이 존재하게 된다. *concurrency* (||) 연산은 'P || Q'로 표현되어지며, 프로세스 'P'가 프로세스 'Q'와 병렬적으로 행동하는 것을 뜻한다. 그리고 두 프로세스가 통신을 할 경우는 공통되는 이벤트가 두 프로세스에 존재할 때 두 프로세스는 단일 채널을 기반으로 하여서 메시지 통신을 하는 것을 의미한다. *interleaving* (|||) 연산은 *concurrency* (||) 연산과 표현과 비슷하나 두개의 프로세스가 서로 동시에 상호작용하지 않으면서 행위를 하는 것을 표현한다. 이외에 프로세스간의 동치성을 표현하는 *trace, failure, refusal, divergence*가 있다.

3.3 CSP와 CCS의 차이점

CCS와 CSP는 표현력은 같지만 그 문법이나 법칙에서 차이를 보인다. 본 논문에서는 CCS와 CSP에서 제공하는 두 프로세스의 동치관계를 증명할 수 있는 의미론의 차이는 생략하도록 하겠다. CCS에서는 CSP의 프로세스를 에이전트(agent), 이벤트를 행위(action)라고 일컫는다. CCS는 CSP와 다르게 그림 2에서 보듯이 두 개의 에이전트가 하나의 채널로 연결되어 있을 경우 각 채널의 양끝에서 발생하는 action (a)과 complement action($\bar{a}$)으로 구분하여 나타낸다. CSP는 pure parallelism (||)과 interleaving(|||)의 개념을 모두 채택하고 있는 반면 CCS의 경우 기본적으로 pure parallelism을 제공하지 않는다. 모든 에이전트들은 인터리빙하게 통신하며 composition(|)을 사용하여서 동시성(concurrency)의 개념을 나타낸다. 그리고 CSP에서는 두 가지의 choice를 제공하는 반면 CCS는 선택적(alternative) choice(+) 하나만이 제공된다. 그림 2의 아랫부분에서 보듯이 어떤 에이전트에 특정한 action을 외부에서 보이지 않도록 제한을 가하는 Restriction이 있다. CSP에서 두개의 프로세스에 공통적으로 연관된 이벤트가 있을 경우 그 이벤트가 발생할 때 동기화 되어서 프로세스간 통신이 이루어지게 된다. 그러나 CCS는 action과 그것의 complement action이 발생이 되더라도 tau(τ)가 발생되지 않으면 agent간의 통신은 이루어 지지 않는다[11]. tau(τ)란 외부에서는 보이지 않는 내부 action으로서 CCS에서는 Restriction이 가해질 경우 어떤 action과 complement action이 발생하지 않고, tau(τ)의 발생으로 인해 두 agent간에 통신이 일어나게 된다. 즉 프로세스간의 동치성을 증명하는 것을 제외한 상태에서 CCS와 CSP의 차이점 중 가장 큰 것은 그림 2에서 나타내고 있는 restriction과 composition에서 볼 수 있는 tau(τ)의 발생에서 찾아볼 수 있다.

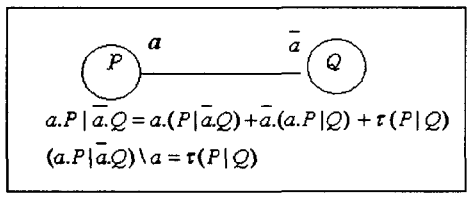


그림 1 CCS에서의 tau(τ), restriction

4. 톨레미II에서의 CCS를 기반으로 한 CCS 도메인 구현

톨레미II에서는 하나의 도메인은 Model of Computation을 정의한다. 모든 도메인은 하나의 director와 receiver를 가지고 있으며 각각은 자바 클래스로 구현되어 있다. director는 Model of Computation의 의미론에 따라서 프로세스들의 통신방법이나 실행순서 등을 제어하는 역할을 한다. 톨레미II에서는 actor를 사용하여서 component based design[12]을 할 수 있다. 이때 각각 나누어진 구성요소(component)들은 자바의 객체처럼 데이타와 데이타에 대한 연산을 포함하고 있으며 각각의 actor는 계층적으로 세부화 될 수 있다. 이러한 actor를 Composite actor[13]라고 한다. 또 각각 세부화된 계층은 최상위 계층과 같은 MoC를 적용할 수도 있고 다른 MoC를 적용할 수도 있다(그림 5). Receiver는 데이타의 token을 받아 저장할 수 있는 일종의 저장소이다[14]. 톨레미II에서 데이타인 token을 주고받을 시에는 모든 도메인에서 공통적으로 put()과 get() 메소드를 사용한다. 이것은 각 도메인의 통신 법칙에 맞게 각 도메인의 Receiver에서 overriding되어 진다. 이 두 메소드는 여러 다른 도메인에서 다르게 정의 되지만 이름을 같이 사용함으로서 여러 도메인의 계층적 연결을 가능하게 한다. 이 절에서는 CCS도메인 구현에 있어서 톨레미II에 있는 정형 도메인 중의 하나인 CSP 도메인의 구조를 살펴보고 CCS와의 의미론적인 차이점을 기반으로 하여 CCS 도메인을 어떻게 구현하였으며 구현된 소프트웨어 구조에 대해서 살펴보도록 하겠다. 이때 CCS 도메인을 기반으로 하는 랑데뷰를 수행할 때 어떠한 순간에 하나의 프로세스와 통신하고자 하는 프로세스는 반드시 하나여야 한다.

4.1 톨레미II에서의 CSP 도메인

톨레미II의 CSP 도메인[15]에서는 단방향 채널을 사용하여서 메시지 통신을 하는 프로세스들의 네트웍 시스템을 모델링 할 수 있다. 톨레미II의 CSP 도메인에서는 랑데뷰를 기반으로 하여서 프로세스간 통신을 한다. 랑데뷰 통신은 프로세스가 통신을 하기 원할 때 상대방 프로세스가 통신 준비가 될 때까지 기다린 후 준비가 되어지면 통신을 하게 된다. 톨레미II의 CSP 도메인 에서는 그림 2에서 보는 바와 같이 CSP에서 사용되어지는 조건 문장(guarded statement)인 CIF(Conditional IF)와 CDO(ConditionalDo)(그림 8)를 구현하였다. 또 통신의 경우는 통신을 하기 위한 기본적인 동작으로 CSP의 ?(input) !(output)을 나타내는 put()과 get()으로 구현했다. 그림 1의 조건 문장(guard communication statement)은 각각 자바의 쓰레드로 구현되어진다. guard는 statement의 수행 가능 여부를 boolean으로 표현하고, communication은 수행시 입출력 여부를 표현한다. 결국 guard(G1)가 참일 경우 입력 혹은 출력을 수행하는 프로세스(C1)가 다음의 statement(S1)를 수행한다. 그림 2에서 guard 문장 안에 '[]'로 구분되어지는 문장들은 각각 쓰레드로 표현되어서 CSP의 비결정성을 나타낸다.

이들 조건 구조(conditional construct) 안의 C1에 해당되는 프로세스들은 각각 쓰레드로 생성되어 진다. 따라서 랑데뷰알고리즘을 정확하게 수행하기 위해서는 쓰레드를 제어해야 한다. 이를 위해서 보완된 알고리즘이 첨가 되어있다. 또 livelock과 deadlock의 발생을 제어하기 위해 알고리즘을 수정하여 구현하였다[15]. 이 부분에 대해서는 본 논문에서 언급하지 않도록 하겠다. 톨레미II에서 구현된 CSP는 기존의 CSP에 시간을 추가한 것으로 랑데뷰를 하기 위해 각 프로세스들은 임의의 시간동안 정지되어지는 특징을 가지고 있다.

```
        guard ; communication => statements;
CIF {                          CDO{
    G1;C1 => S1;                   G1;C1 => S1;
[]                             []
    G2;C2 => S2;                   G2;C2 => S2;
[]                             []
      ....                           .....
    }                          }
```

그림 2 guard, CIF, CDO

4.2 CCS 도메인 구현

그림 1에서 보여진 수식은 action 'a'가 발생하거나 complement action, '$\bar{a}$'가 발생하여 각각 에이전트 P와 Q가 한 스텝씩 다음으로 진행하거나 tau(τ)가 발생하여서 통신이 이루어지는 세가지 중의 하나가 발생하는 것을 의미한다. 즉 'a'와 '$\bar{a}$'가 발생해도 임의적으로 tau(τ)가 발생할 때만 통신이 일어나게 된다. 또 restriction이 가해졌을 경우는 tau(τ)의 발생과 함께 통신이 이루어져야 한다.

여기서부터는 에이전트를 톨레미II에 맞춰 설명하기 위해 '프로세스'로 명칭을 바꾸도록 하겠다. 톨레미II의 커널의 구조를 사용하면서 위의 두 가지 조건을 만족시키기 위해 본 논문에서는 tau flag, restriction 매개변수를 기존의 CSP 도메인의 커널에 추가하는 방법을 제안하고 그림 4와 같이 기존의 랑데뷰 알고리즘이 구현되어 있는 CCSReceiver와 비결정적 선택(nondeterministic choice)을 가능하게 하는 조건 구조(conditional structure)에 해당하는 클래스의 알고리즘을 수정하였다.

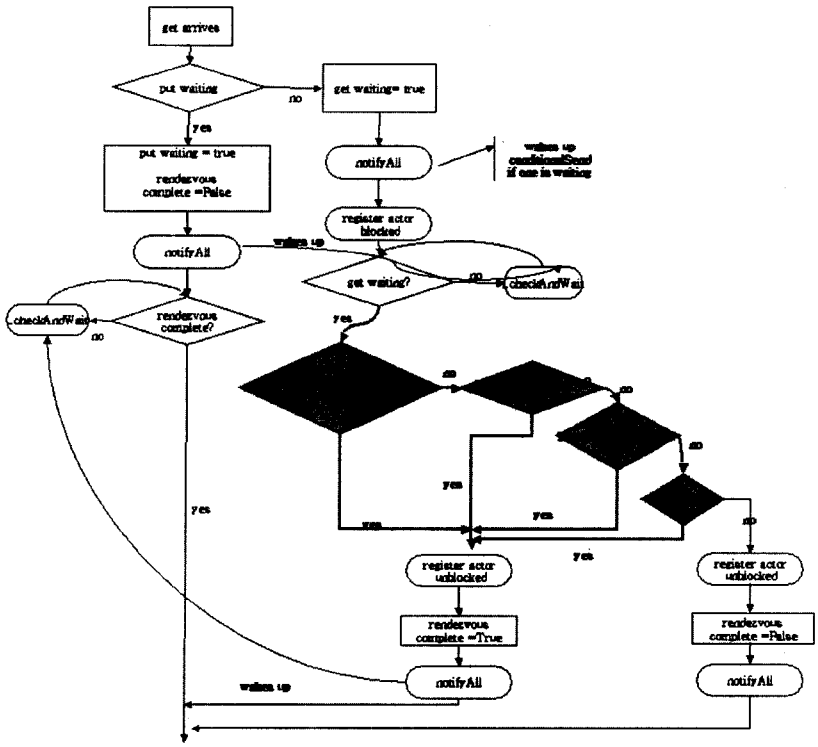


그림 3 tau(τ)와 restriction 확인이 추가된 랑데뷰 Algorithm

4.2.1 CCS Receiver

Receiver는 톨레미II에서 output port로부터 데이타를 받아서 저장하는 일종의 저장소나 저장된 데이타를 보내기도 한다[14]. 랑데뷰를 기반으로 하는 모델은 동시적이며 순차적인 프로세스들(concurrent sequential Processe)로 구성되어 있다. 프로세스들은 동기화 시점(synchronization points)에서 서로 통신한다. 즉, 동기화 시점(Syncronization point)에 도달해야만 한다. 프로세스 알제브라의 언어들은 앞에서 언급한 랑데뷰 통신을 기본으로한 모델을 사용하는데 이러한 대표적인 프로세스 알제브라 언어로는 CCS와 CSP가 여기에 속한다.

CSP도메인의 경우 데이타인 token을 put()과 get() 메소드를 사용하여서 주고받을 경우 랑데뷰 통신을 기반으로 한다. 이러한 랑데뷰는 receiver에 구현이 되어 있다 그러나 CCS의 경우는 랑데뷰를 기반으로 하여 프로세스간 통신을 하지만은 tau와 restriction의 발생의 여부에 따라서 각각 프로세스들은 동기화 시점(syncronization point)에 이를 수도 있고 이르지 못할 수도 있다[4].

그림 3은 기존의 랑데뷰알고리즘이 수정되어 CCS-Receiver에 구현된 것을 나타낸다. 이것은 get() 메소드가 불리워졌을 경우 get() 메소드안에 구현된 랑데뷰 알고리즘을 나타낸다. 기존의 일반적인 랑데뷰알고리즘은 tau와 restriction의 확인 없이, 통신하고자 하는 두 프로세스가 통신할 준비가 되었다면은 랑데뷰에 성공하게 된다. 이 때 통신하는 프로세스의 타입은 일반적인 조건 객체(conditional object)가 아닌 것(i.e. CCSActor)일수도 있고 조건 객체(conditional object)(i.e. conditional-Receiver, conditionalSend)일 수도 있다. 그림 4는 두 가지의 경우를 하나의 그림으로 표현했기 때문에 대칭적으로 해석이 되어진다.

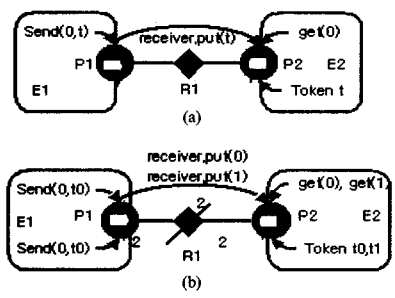


그림 4(a) 톨레미II에서 하나의 channel을 가지고 있는 relation (b) 두개의 channel을 가지고 있는 relation

일단 통신하고자 하는 프로세스의 restriction은 CCSIOPort에서 시스템을 디자인할 때 개발자가 restriction의 값을 설정해 주며 설정이 안되었을 경우는 기본적으로 거짓 값을 가진다. restriction에 대해서는 CCSIOPort에서 설명하도록 하겠다. get이 일단 호출되면은 통신을 하려고 기다리고 있는 상대방 프로세스가 있는지를 확인한다. 이것은 putwaiting이라는 flag를 확인함으로 알 수 있는데 put 메소드를 호출한 sender는 자신이 도착했음을 나타내기 위해 flag를 참으로 값을 바꾼다. 그리고 일단 대기 상태로 들어간다. putwaiting flag가 참일 경우는 서로 랑데뷰 통신을 하고자 준비를 한다. 준비하는 과정은 아직 랑데뷰가 이루지지 않았음을 나타내는 flag인 RendezvousComplete를 거짓 값으로 바꾼 다음 대기하고 있는 모든 프로세스들을 깨운다. 이때 대기하고 있던 쓰레드는 통신을 위하기 위해 깨어난다. 그리고 대기상태가 해제된 get() 메소드를 호출한 상대방 프로세스는 자신의 flag의 값이 변경이 되었는지를 확인한 다음에 자신과 통신하고자하는 프로세스의 restriction 정보를 확인한다. 자신과 통신하는 프로세스가 조건 객체(conditional object)인 conditionalReceive나 conditionalSend일 수도 있고 조건 객체(conditional object)가 아닌 일반 Actor일 수도 있다. 따라서 자신이 통신하려는 프로세스가 조건 객체(conditional object)일 경우는 조건 객체(conditional object)의 각 클래스안(예를 들면 conditionalReceive, conditionalSend 클래스)에서 통신하려는 상대방 프로세스의 restriction을 먼저 확인하고 자신과 상대방의 restriction이 참인 경우와 tau flag의 값을 모두 먼저 확인하기 때문에 CCSReceiver에서는 통신하고자 하는 프로세스가 조건 객체(conditional object)인지의 여부만 확인하다. 이미 CCSReceiver안으로 들어온 조건 객체들은 restriction이 true이거나 tau flag가 참인 경우이다. 그리고 조건 객체(conditional object)가 아닌 것과 통신할 경우는 자신의 restriction과 상대 프로세스의 restriction 값을 확인하고 그 값에 따라 랑데뷰를 수행한다. 또 restriction이 설정이 되어지지 않았으나 tau의 값이 참이 될 경우도 랑데부를 수행한다. 그러나 tau의 값이 거짓일 경우는 두 메소드(get()과 put())는 null을 반환하게 되고 RendezvousComplete는 거짓 값을 갖게 된다.

4.2.2 CCS Director

일반적으로 Director는 전체 actor들을 실행하는 순서를 스케줄링하고, 전체 actor들 간의 통신 방법을 제어한다. CCSDirector에서는 기존의 CSPDirector의 시간의 개념을 그대로 유지하였다. 여기서 시간의 개념이란 timed CSP와는 조금 다르다. 전체적으로 actor의 시간의 개념이 아니라 랑데뷰 수행시 상대방 프로세스가 통신 준비가 되어 있지 않을 경우 일정 시간 동안 잠시 제지되어 질 수 있는 것이다. 이것은 시간을 설정하지 않으면 기본적인 CSP의 의미론을 기반으로 하여 통신할 수 있게 된다. 이러한 CSP의 시간개념을 CCS에서도 그대로 유지하였다. 그러나 restriction이나 tau의 값을 확인할 때 단지 actor만 block list에 저장되었다. 전의 CSP와는 달리 조건 객체(conditional object)인 쓰레드도 block을 시킬 때 block된 쓰레드의 list를 생성함으로 인해서 각각의 조건 객체(conditional object)가 아닌 것뿐만 아니라 조건 객체(conditional object)까지도 block list 생성이 됨으로써 조건 객체(conditional object)가 가지고 있는 restriction과 그 외의 정보를 사용할 수 있도록 하였다. 이것은 프로세스를 제어하는 간접적인 방법으로는 get()과 put()함수를 들어가기 전 자기의 정보를 list에 넣고 restriction값까지 넣음으로 해서 get()과 put()메소드 안에 일단 들어가면은 그때 자신의 정보를 찾아서 restriction값을 가지고 파라메터화 할 수 있게 되어진 get()과 put() 메소드에 들어가게 된다.

4.2.3 CCSIOPort

톨레미II에서의 IOPort는 actor와 actor간의 데이타를 전송하고 받는 입구이다. 모든 Port는 입력 채널과 출력 채널들의 집합이다. 하나의 채널은 통신하고자 하는 두 개의 프로세스의 연결을 의미한다. 즉 소비자 actor와 생산자 actor가 가지고 있는 각각 한 개씩의 port의 연결을 말한다. 입력 port와 출력 port를 연결하는 channel은 CCS의 의미론으로 설명이 되는데, 예를 들면 입력 port는 a라는 action을 말하고 출력 port는 $\overline{a}$ 라는 action을 의미한다. 이 channel을 통해서 token이 주고 받아진다면 이것은 tau나 restriction값에 의해서 랑데부 통신이 발생한 것을 의미한다.

Port는 반드시 자신을 포함하고 있는 컨테이너(container)를 가지는데 이것은 actor를 의미한다. port는 두개의 타입을 가지는데 하나의 채널만을 가질 수 있는 단일 port와 여러 개의 channel을 가질 수 있는 다중 port가 있다. channel은 두개의 port가 서로 통신이 가능하도록 연결된 것을 말한다.

또 Port들은 relation에 의해서 묶여질 수 있다(그림 5). CCSIOPort는 기존의 IOPort와 그 하위 클래스인 TypedIOPort를 상속하였으며, restriction 파라메터를 가지고 있다.

이 restriction 파라메터는 시스템 디자이너가 CCS 도메인을 사용하여서 디자인 했을 경우 이 값을 정해줌으로써 CCS 기반의 통신에서 restriction을 표현할 수 있게 하였다. 또한 이렇게 정해진 값인 restriction은 여러 메소드를 통해 값이 CCSIOPort에 저장되고 사용되어 진다. 이것은 랑데뷰가 이루어지는 해당 Receiver에

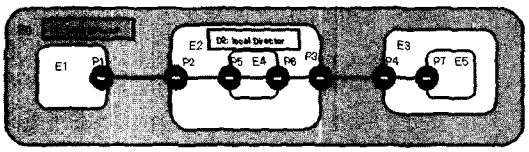


그림 5 툴래미II 모델이 계층적으로 연결된 형태를 추상화한 예

서 랑데뷰통신 전에 확인된다.

4.2.4 조건 통신 구조(Conditional Communicating Construct)의 제어

조건 통신 구조(Conditional Communicating Construct)는 Neil Smyth에 의해서 툴래미II CSP[15] 도메인 구현시 제시되었다. 이것은 CSP의 internal choice의 비결정성을 나타내며 비결정적으로 랑데뷰를 수행한다. nondeterministic 랑데뷰는 guard communicating statement를 기반으로 하고 있다. 툴래미II에서 하나의 프로세스 혹은 프로세스 안의 하위 프로세스로 여겨질 수 있는 Actor와는 다르게 쓰레드로 생성되어 진다. 이러한 조건 구조(conditional construct)는 guard가 참이거나 값이 설정되지 않은 것을 골라서 각각의 branch 마다 쓰레드로 생성된다. 따라서 쓰레드를 제어하기 위해 랑데뷰알고리즘과는 별도의 제어가 필요하다. 따라서 CSP 도메인에서는 conditionalSend와 conditionalReceiver 클래스 안에 제어하는 알고리즘을 넣었다[15]. 랑데뷰 알고리즘과는 별도로 쓰레드의 정확한 통신을 위해 만들어진 알고리즘이다.

본 논문에서는 이 알고리즘에서 각각 조건 객체(conditional object)인 ConditionalReceiver와 CondtionalSend가 포함하는 CCSIOPort의 restriction값을 조건 객체(conditional object)안에서 확인하고 이 프로세스와 통신하고자 하는 상대방 프로세스의 restriction도 확인하도록 수정하였다(그림 6). 통신하려는 프로세스 양쪽 다 restriction의 값이 참이 아닐 경우는 tau의 값까지 확인을 하여서 CCSReceiver안의 put()과 get() 메소드를 부르게 된다. 즉 CCSReceiver안의 get()과 put()메소드에 들어가게 되는 조건 객체는 이미 tau의 값이 참이거나 restriction이 설정이 되어 있는 경우이다.

조건 통신 구조에 해당하는 클래스 안에서 tau와 restriction을 각각 확인하고 CCSReceiver안에 들어감으로써 기존의 랑데뷰 알고리즘에서 조건 객체가 아닌 프로세스만의 tau와 restriction의 값을 확인함으로써 알고리즘이 복잡해지는 것을 막았다. 조건객체를 제어하는 알고리즘은 세부분으로 크게 나누어 설명할 수 있다. 그

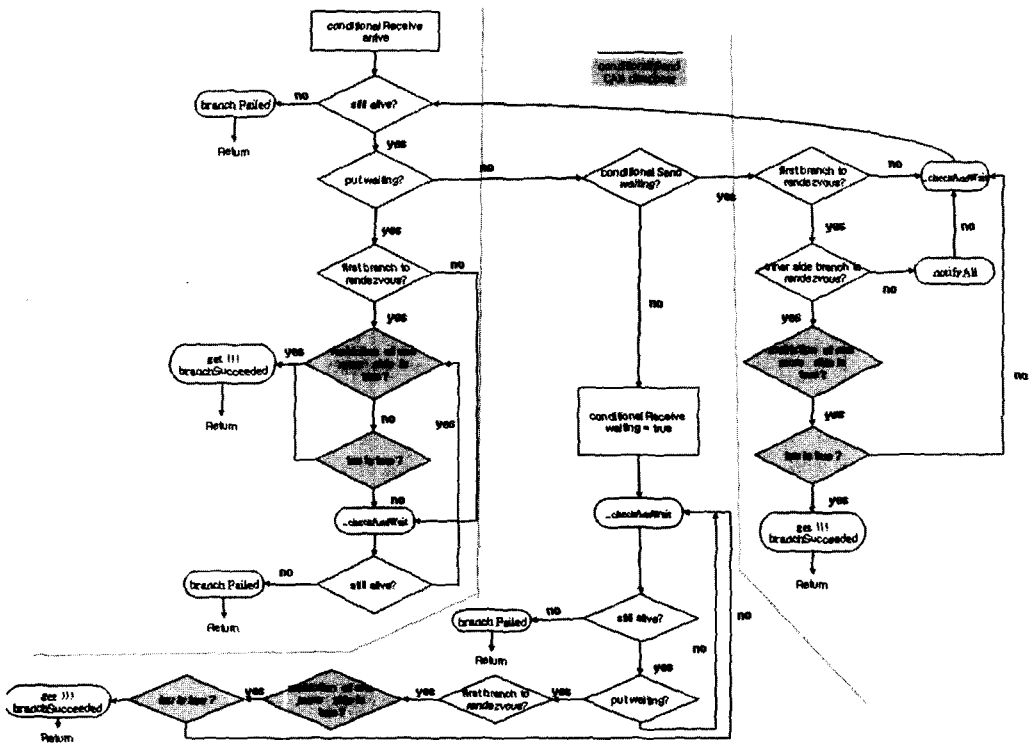


그림 6 conditional Rendezvous branch의 랑데뷰가능 여부를 결정하는 알고리즘

림 6에서 왼쪽에서 오른쪽 방향으로 차례차례 설명하면 (1) conditionalReceive가 먼저 도착하고 Put이 기다리고 있을 경우와 (2) conditionalReceive가 먼저 도착했을 경우, (3) 마지막으로 conditionalReceive가 도착했을 때 conditionalSend가 기다리고 있는 경우이다[9]. 첫 번째의 경우는 조건객체가 랑데뷰를 하기 위해 도착했을 때 자신이 통신을 할 수 있는 첫 번째 객체인지 확인한다. 첫 번째 객체인 경우, 통신을 위해 연결된 두 객체의 port의 restriction 정보를 확인한다. 이 정보중 하나가 참일 경우는 통신을 하게 되고 아닐 경우는 tau의 값이 참인지를 확인하다. 그 값이 참이 아닐 경우는 잠시 동안 대기 상태가 되어 지면 일정 기간이 지나고 깨어 났을 EO 자신이 여전히 통신을 할 수 있도록 살아 있는 지 확인하고 restriction의 값과 tau의 값을 확인하여 계속적으로 통신을 하기 위한 시도를 한다. 두번째의 경우는 Put이 기다리고 있지 않고 conditional Receiver가 Put 보다 먼저 도착했을 경우는 있으면 대기 플래그를 참으로 설정하고 conditionalReceive는 Put을 계속적으로 기다리는 동안 대기 상태가 되어지는데 일정기간이 지나면 깨워져서 자신이 여전히 살아있는지를 확인하고, Put이 와서 기다리고 있는 것을 나타내는 Putwaiting 플래그를 확인한 후 그 값이 참일 경우는 자신이 첫 번째 통신할 수 있는 조건객체인지 확인한 후 통신할 수 있는 조건 객체라면은 (1)번에서 했던 것과 같은 동일한 절차인 restriction값을 확인하고 그 값이 참이 아닐 경우는 다시 tau의 값을 확인한 후 그 값에 따라서 통신의 여부를 결정하여 통신을 하게 된다. 세 번째의 경우는 conditionalReceive가 도착하고 상대방 통신 객체가 통신을 하기 위해 미리 기다리고 있을 경우이다. 이 경우 도착한 conditionalReceive가 통신할 수 있는 첫 번째 객체라면은 상대방 객체도 통신할 수 있는 첫 번째 객체인지를 확인한 후 둘다 통신이 가능한 첫 번째 객체일 경우는 각각 restriction과 그 값에 따라 tau를 확인한 후 통신을 하게 된다. 첫번째 객체가 아닐 경우는 계속적으로 기다리고 다시 통신을 하기 위해 시도를 하게 된다.

이 알고리즘에서 역시 통신하고자 하는 두 프로세스의 restriction을 먼저 확인하고 restriction의 값이 거짓인 경우는 tau를 확인하게 된다. 이렇게 통신하고자 하는 프로세스가 조건 객체(conditional object)일 경우는 따로 restriction과 tau를 확인하고 랑데뷰를 수행함으로 인해 CCSReceiver안의 기본 랑데뷰 알고리즘을 더 간단히 구현할 수 있었다.

5. 예 제

톨레미II에는 시스템을 명세할 때 2가지의 명세 방법이 있다고 앞에서 언급하였다. 하나는 톨레미II의 각각의 도메인 커널등을 마치 JAVA의 API처럼 취급하여 사용하여 결과는 애플릿으로 보여주는 방법이 있고, 다른 하나는 그래픽 도구인 vergil을 사용하여 시스템을 명세하는 것이다.

본 논문의 예제에서는 두가지 benchmark 예제를 보기로 들었고, 각각은 위의 두가지의 다른 명세방법으로 명세되어 질 수 있다.

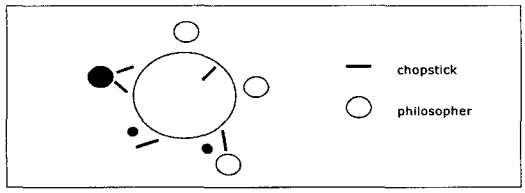


그림 7 생각하는 철학자 문제

5.1 DiningPhilosopher

DiningPhilosopher 문제는 1965년 Dijkstra에 의해서 처음 제시되었다. 이것은 concurrent programming의 두개의 기본적인 Property인 Liveness와 Fairness에 대해 설명하는 고전적인 문제이다. 이 문제에서 Liveness를 묻는 것은 Deadlock을 피할 수 있게 디자인 할 수 있는 지의 여부를 묻는 것을 의미한다. 이와 같이 Fairness를 묻는 것은 starvation을 피할 수 있게 디자인 할 수 있는 지의 여부는 묻는 것이다.

구현에서 각각의 철학자는 임의적으로 첫 번째 젓가락을 집도록 되어 있다. 모든 철학자가 같은 비율로 생각하고 먹는다. 각각의 철학자와 젓가락은 하나의 프로세스들로 설계된다. 각각의 젓가락은 CDO를 사용하여서 조건 객체(conditional object)로 설계된다. 일단 철학자가 젓가락을 잡으면 임의의 시간동안 먹는다. 각각의 프로세스들은 다른 프로세스들과 CCSIOPort를 사용하여서 연결이 되는데 CCSIOPort는 restriction을 의미하는 파라메터를 가지고 있다.

DiningPhilosopher 문제에서 하나의 젓가락은 각각의 양옆의 철학자와 연결된 port를 2개씩 가진다. 즉 양쪽의 철학자와 통신을 위한 입출력 port를 가진다. 예를 들면 Leftout은 젓가락 프로세스가 왼쪽 철학자와 통신하기 위한 port이다. 이렇게 젓가락은 각각 두개씩의 입출력 port를 가지고 있다. 세부적으로 설명을 하자면, 이 때 젓가락은 두개의 입력 port를 가지고 있는데 양옆의 철학자가 하나의 젓가락을 동시에 사용하고자 할 때 젓가락 안에서는 조건 객체(conditional object)인 conditionalDO(그림 8)을 사용하여서 그 안에서 conditionalSend를 생성하게 된다.

```
boolean continueCDO=true;
while(continueCDO){
    //step 1:
    conditionalBranch[] branches = new ConditionalBranch[#branchesRequired];}
    //Create a CondtionalReceive of a ConditionalSend for each branch
    //e.g. branches[0] = new ConditionalReceive( (guard), input, 0, 0 );

    //step2
    int result = chooseBranch(branches);

    //step3
     if(result == 0){
              // execute statements associated with first branch
    }else if(Result == 1){
             // execute statements associated with second branch
    }else if....// continue for each branch ID

    }else if(result ==-1){
           // all gaurds were false so exit CDO.
           continueCDO = false;
    }else{
```

그림 8 Condtional structure인 CDO의 Pseudo code

이 때 chooseBranch함수를 호출하여 두개의 쓰레드 중에서 통신을 할 쓰레드가 정해지고 이 때 어떠한 철학자와 통신할지 결정되어 지게 된다. 다음 그림은 철학자의 코드이다. 여기서 tau의 값이 false일 경우는 계속해서 프로세스가 진행해야 한다. 따라서 keepgoing이라는 flag로 다음 스텝을 진행하는 여부를 결정하게 하였다.

```
        //obtain the forks.
    if(leftIn.get(0) != null){
            // tau occurs
            // set  gotleft flag true
            // set  waitingRight  falg true

     }else {
           // tau doesn't occur then just keep going
           // set gotleft flag false
         }

    if( rightIn.get(0)!= null){
            // tau occurs
            // set  gotRight flag true
            // set  waitingLeft falg true

     }else{
          // tau doesn't occur then just keep going
          // set  gotRight flag false
     }

// Release the forks.

      leftOut.send(0, t);
      if(leftOut.getSucessRendezvous()){

      //set gotLeft false

       }else{

      //set letf out flag which represents whether process can
        keep going or not false

      }

      rightOut.send(0, t);

     if(rightOut.getSucessRendezvous()){
      //set gotRight false

     }else{

        //set Right out flag which represents whether process can
          keep going or not false

     }

    if( !keepgoingLeftOut & !keepgoingRightOut ){
     // notify this status not for chaing the graphic view

    }
```

그림 9 Philosopher의 fire() 매소드의 부분별 코드

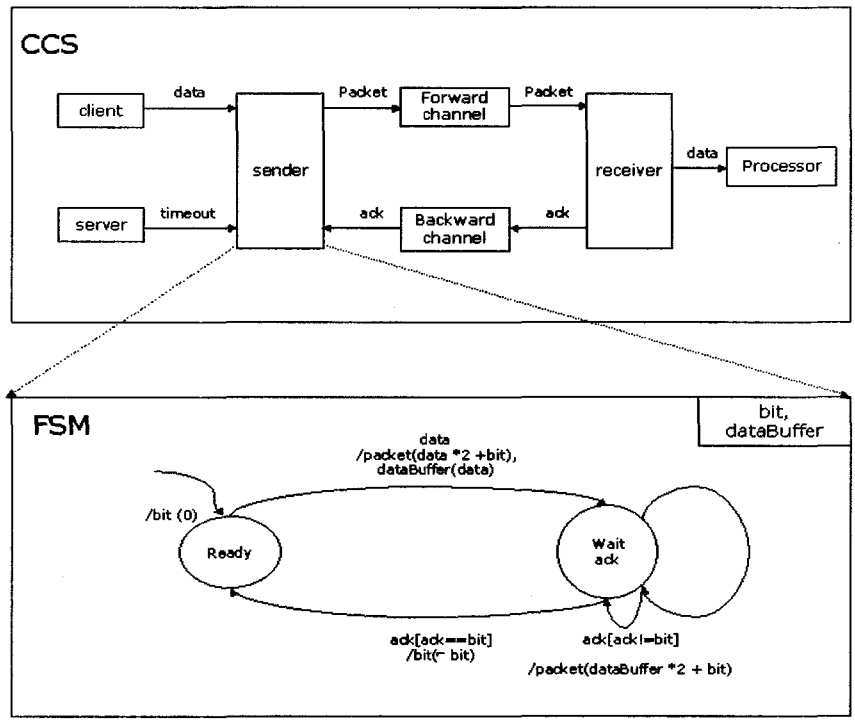


그림 10 Alternating Bit Protocol 중 sender 명세 간략화 그림

5.2 Alterating Bit Protocol

Alternating Bit Protocol은 단순한 재전송 프로토콜이다. 이것은 메시지를 잃어 버렸을 때 보내는 쪽에서 다시 재전송을 하고, 중복 메시지가 재전송에 의해 도착했을 경우는 수렴하는 쪽에서 확인하여 무시하도록 하는 특징을 가진다. 이 프로토콜은 다음과 같이 동작한다.

보내는 쪽인 sender는 프로토콜을 포함한 데이타 패킷을 전송한다. 받는 쪽인 receiver측의 동작을 먼저 살펴 보자. receiver가 데이타 패킷을 수렴했을 때 받은 패킷의 비트(bit)와 같은 비트를 포함한 ack를 보낸 쪽에 보낸다 sender는 데이타를 받을 때 이 데이타가 중복된 데이타 패킷인지 아닌지를 확인해야할 필요가 있다. 따라서 다음과 같은 동작을 한다. 만약 들어온 데이타 패킷이 처음으로 들어온 것이라면은 receiver는 이 패킷을 첫번째 것으로 확인하고 데이타를 사용한다. 그 이후 receiver에 들어온 패킷은 새롭게 들어온 프로토콜 비트와 마지막으로 들어온 패킷의 프로토콜 비트와 비교한다. 만약 같다면 중복 비트로 취급하여 그 데이타를 무시하고 아니라면 새로운 데이타로 생각하여 취급한다. 반면 보내는 쪽인 sender는 receiver로부터 ack를 기다린다. 만약 일정시간이 지나서도 자신이 보낸 데이타 패킷의 프로토콜 비트와 같은 비트를 포함한 ack가 오지 않으면 ack가 올 때까지 동일한 데이타를 재전송한다.

위와 같은 동작을 하는 Alterating Bit Protocol은 추상화된 그림 9와 같이 톨레미II의 그래픽 도구인 vergil로 구현되어 질 수 있다. 각각의 사각형은 동작하는 객체를 표현하고 있다. sender와 receiver의 경우는 톨레미II에서 다른 도메인이 FSM을 사용하여서 다시 명세되어 질 수 있다. 이렇게 다른 도메인으로 정의된 sender와 receiver는 CCS 도메인 안에서 하나의 객체처럼 취급되어 동작되어 진다. 그림 9의 경우는 가장 추상화 되어서 나타낸 것이다. 그림의 sender의 경우는 sender의 행위를 FSM으로 표현한 것이며 bit와 dataBuffer로 표현된 것은 일종의 광역 변수 이다. sender는 dataBuffer안의 data와 bit를 포함한 패킷을 전송한다. sender는 같은 프로토콜 비트를 가진 ack가 도착할 때만 프로토콜 비트를 0에서 1로 바꾸고 ready 상태로 이동한다. receiver의 경우도 FSM을 사용하여 위에 언급된 receiver의 동작대로 명세 한다.

6. 결론 및 향후 연구 방향

본 논문에서는 톨레미II에 CCS 도메인을 구현 및 추가하였다. CCS와 CSP의 차이점을 밝혀 CCS의 의미론을 기반으로 한 tau 구조체와 tau flag, 그리고 CCSIOPort를 설계함으로써 인해 CCS 도메인을 구현하는 방법을 제시했다.

이것은 단지 CCS 도메인을 설계하여 톨레미II에서 디자인을 설계할 수 있는 정형 도메인의 다양성을 가져왔을 뿐만이 아니라, 톨레미II를 사용하는 개발자가 기존의 톨레미II에서 시스템을 명세하고 디자인 하는 패턴을 그대로 사용하면서도 정형적 명세가 되어진 시스템을 개발할 수 있게 한다.

그러나 CCS 도메인은 시간과 자원 그리고 우선순위를 정형적으로 설계 및 검증할 수 있는 톨레미II에 ACSR[8] 도메인을 구현 및 추가할 수 있는 기반을 마련하였다. ACSR은 CCS를 기반으로 하여서 만들어진 Process Algebra Language의 하나이다. CCS의 경우는 시간개념이 없고 자원의 개념이 명확히 구분되어서 정의 되어지지 않았다. 이에 반해 ACSR은 CCS에 시간, 자원 그리고 우선순위의 개념이 첨가 되어 졌다. 따라서 시스템을 보다 애매모호함이 없이 정형적 명세를 할 수 있다. 톨레미II에서는 CCS의 의미론이 Java로 구현이 되었기 때문에 구현된 CCS도메인의 소프트웨어 구조를 사용하여 ACSR 도메인으로의 확장을 가지고 올 수 있다. 따라서 향후 연구방향은 CCS를 기반으로 한 ACSR을 톨레미II에 도메인을 구현 추가하는 것이다. 기존에 시간과 우선순위 개념이 들어간 실시간 내장형 시스템을 설계할 때 기존의 방법이라면 Timed Multitasking도메인[16] 등과 같은 우선순위 개념이 들어간 도메인과 CCS도메인 이렇게 두개이상의 도메인을 사용하여 디자인 하였으나 ACSR이 추가되어 진다면 ACSR은 앞에서도 언급한 바와 같이 자원과 시간의 개념 그리고 우선순위의 개념이 들어가 있기 때문에 ACSR 도메인 하나로 시간의 개념, 자원의 개념, 그리고 우선순위의 개념 까지 한번에 하나의 도메인 안에서 디자인 할 수 있다. 이것은 톨레미II의 디자인 시의 편리성을 더욱 증가 시킨다. 또 Timed Multitasking 도메인의 경우는 프로세스 알제브라 언어처럼 정형적 이론을 기반으로 한 의미론을 구현한 것이 아니다. 따라서 ACSR도메인의 추가는 하나의 도메인으로 실시간 시스템의 정형적 명세를 가능하게 할 것이다.

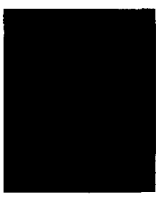
참 고 문 헌

[1] Edward A. Lee 외 4명, Design of Embedded Systems:Formal Models, Validation and Synthesis. *Proceedings of IEEE* VOL.85 No.3 pp.366-390, March, 1997.

[2] Edward A. Lee, What's Ahead for Embedded Software? *IEEE Computer Magazine*, pp. 18-26, September 2000.

[3] C.A.R. Hoare, Communicating Sequential Processes, Prentice Hall International, 1985

[4] Robin Milner, Communication and Concurrency, Prentice Hall international Ltd, 1989.
[5] J.A. Bergstra & J.W. Klop, Algebra of Communicating Processes with Abstraction, Theor Comp Sci. 37(1):77-121 1985.
[6] I. Lee, H. Ben-Abdallah, and J.-Y. Choi, A Process Algebraic Method for Real-Time Systems, Formal Methods for Real-Time ComputingC. Heitmeyer and D. Mandrioli (eds), John Wiley Sons Ltd, 1996.
[7] Edward A. Lee 외 3명, Actor-Oritented Control System Design : A Resonsible Framewo Perspective IEEE Transztions on control systems technology. VOL.X No. Y Month-Z 2003.
[8] Edward A. Lee 외 17명 Heterogeneous Concurrent Modeling and Design in Java, UCB/ERL M01/12, University of California, Berkeley, March 15, 2001.
[9] Edward A. Lee, "Embedded Software," Advances in Computers(M. Zelkowitz, editor), Vol. 56, Academic Press, London, 2002.
[10] Steve Schneider, "Concurrent and Real-time systems(the CSP Approach)," John Wiley & Sona, http:// www.cs.rhbnc.ac.uk/books/ concurrency, 1993.
[11] R.J. van Glabbeek, Notes on the methodology of CCS and CSP, Theorical Computer Science 177 329-349, 1997.
[12] Edward A. Lee and Yuhong Xiong, "Behavioral Types for Component-Based Design," Memorandum UCB/ERL M02/29, University of California, Berkeley, CA, 94720, USA, September 27, 2002.
[13] Jörn W. Janneck, "Actors and their composition," Memorandum UCB/ERL M02/37, University of California at Berkeley, 18 December 2002.
[14] Edward A. Lee and Yuhong Xiong, "System-Level Types for Component-Based Design," Technical Memorandum UCB/ERL M00/8, Electronics Research Lab, University of California, Berkeley, CA 94720, USA, February 29, 2000.
[15] Neil Smyth, Communicating Sequential Processes in Ptolemy II, University of California, berkeley, December 15, 1998.
[16] Jie Liu and Edward A. Lee, "Timed Multitasking for Real-Time Embedded Software," invited paper in IEEE Control Systems Magazine, special issue on "Advances in Software Enabled Control," draft version.

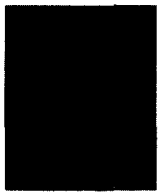


황 혜 정

2000년 덕성여자대학교 전산학과 졸업 2004년 고려대학교 컴퓨터학과 석사 졸업. 2004년~현재 삼성전자 기술총괄 기술전략실 모바일 솔루션 연구실. 관심분야는 정형기법, 소프트웨어 공학, 소프트웨어 라디오

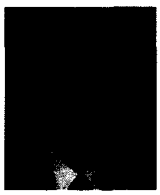
김 윤 정

1991년 서울대학교 컴퓨터공학과 졸업 1993년 서울대학교 컴퓨터공학과 공학석사 졸업. 2000년 서울대학교 전기컴퓨터공학부 공학박사 졸업. 2000년~2001년 (주)엔써커뮤니티 제품개발연구소 차장 2001년~2002년 (주)데이타게이트인터내셔날보안기술연구소차장. 2002년~현재 서울여자대학교 정보통신공학부 조교수. 관심분야는 암호학, 시스템 보안, 암호 응용



남 기 혁

1997년~2002년 고려대학교 컴퓨터학과 졸업. 2002년~2004년 고려대학교 컴퓨터학과 석사 졸업. 2004년~현재 ETRI 디지털홈 연구단 인터넷 서버 그룹 온디맨드서비스연구팀. 관심분야는 정형기법, 소프트웨어 스트리밍



김 일 곤

1993년~2000년 경기대학교 영어영문학과 졸업. 2000년~2002년 고려대학교 컴퓨터학과 석사 졸업. 2002년~2003년 고려대학교 컴퓨터학과 박사 과정 수료. 관심분야는 정형기법, 소프트웨어 공학, 보안 프로토콜, 보안 운영체제

최 진 영

1982년 서울대학교 컴퓨터공학과 졸업 1986년 Drexel University Dept. of Mathematics and Computer Science 석사. 1993년 University of Pennsylvania Dept. of Computer and Information Science 박사. 1993년~1996년 Research Associate, University of Pennsylvania. 1996년~1999년 고려대학교 컴퓨터학과 조교수. 1999년~2003년 고려대학교 컴퓨터학과 부교수. 2003년~현재 고려대학교 컴퓨터학과 교수. 관심분야는 컴퓨터이론, 정형기법(정형 명세, Formal verification), 실시간 시스템, 분산 프로그래밍 언어, 소프트웨어 공학