한국시뮬레이션학회 '99춘계학술대회 논문집 1999.4.17 동의대학교

다중성 인스턴스 추상화에 기반한 통신망 관리 시스템 설계 모델 형식론

박 수 현*○, 백 두 권**
* 동의대학교 컴퓨터공학과
** 고려대학교 컴퓨터학과 소프트웨어시스템 연구실

The Formalism of Design Model of Network Management System based on Multiplicity Instance Abstraction

Soo-Hyun Park*○, Doo-Kwon Baik**
* Dept. of Computer Engineering, Dongeui University
** Software System Lab, Dept. of Computer Science & Engineering, Korea University

요 약

Farmer 모델은 시스템 개체구조(System Entity Structure) 의 개념을 도입한 지식표현을 위해 사용되는 프레임 구조모델로서 다중성 추상화 개념(Multiplicity Instance Concept)은 하나의 개체를 구성하기 위하여 동일한 형태의 구성요소가 여러 번 발생하는 경우에 이의 대표적인 요소만을 표시하는 추상화 개념이다. 다중성 추상화 개념에서 정의된 대표개체는 자신의 인스턴스들을 가질 수 있다. 이들 인스턴스들은 IM-컴포넌트 타입 개체노드 및 OM-컴포넌트 타입 개체노드이며 다중성 인스턴스 링크를 이용하여 대표개체와 연결된다.

I. 서론

객체지향 모델[1][2][3]을 포함하여 대부분의 모델에서 실세계의 개체는 모델구성자의 관점에 따라 각각 고정된 측면에서 표현되기 때문에 하나의 개체를 여러 측면에서 관찰할 수 없고 따라서 세부적인 계층구조를 나타낼 수 없는 단점을 지닌다. 이러한 단점을 보완하기 위하여 제안된 Farmer 모델[4][5][6]은 실세계의 개체를 각각의 고정된 측면이 아닌 여러 관점의 측면에서 분석한 후 분석완료된 측면요소들을 측면객체로 정의한다. Farmer 모델은 시스템 개체구조(System Entity Structure)의 개념을 도입한 지식표현을 위해 사용되는 프레임 구조모델로서 지식표현 모델인 EA (Entity-Aspect) 모델[6][7]에 기본을 두고 있다.

Farmer 모델의 주요 특징은 디자인 완료된 시스템을 구현시 이미 컴포넌트웨어[8][9][10] 형태로 만들어져 있는 소프트웨어 및 데이터 요소들을 LAN 이나 인터네트를 통하여 외부의 저장소(repository)로부터 아웃소싱(outsourcing)해 가지고 오는 것이다. 본 논문에서는 Farmer 모델 중 다중성 인스턴스 추상화에 기반한 통신망 관리 시스템 설계 모델 형식론에 대하여 설명하고 예를 들었다.

II. 다중성 인스턴스 추상화

1. Farmer 모델의 기본개념

Farmer 모델의 주요 목적은 실제 디자인 하고자 하는 에이전트를 분석하여 에이전트를 구성하고 있는 컴포넌트 요소들을 측면에 따라 분리, 추출해 내는 데 있다. 이러한 결과 추출된 컴포넌트 요소들은 Farmer model tree의 leaf 노드에 위치하게 되며 이러한 컴포넌트 요소들은 network를 통하여 최종적으로 PICR에 저장이 된다. 또한 Farmer 모델은 플랫폼독립형 클래스저장소(PICR)에서 컴포넌트 요소를 에이전트로 동적 또는 정적으로 다운로딩하는 Farming의 개념을 추가한 형식모델이다. Farming 방법론의 핵심 개념은 Farming이며 Farming 방법론의 기본적인 프로그래밍 패러다임으로 개체측면지향 프로그래밍(Entity-Aspect Oriented Programming)을 도입하고 있다. Farmer 모델은 바로 EAOP의 기본모델이 된다.

Farmer 모델은 실세계의 객관적, 추상적 대상체를 표현하는 개체노드 타입 (entity node type), 이러한 개체를 표현하는 관점인 측면노드 타입 (aspect node type), 개체와 측면간의 관계성을 나타내는 링크타입 (link type) 그리고 개체가 가지는 성질을 나타내는 속성타입 (attribute type), 동일한 이름을 가지는 2 노드는 동일한 속성과 동일한 서브트리를 갖는다는 균일성의 원리에 의해 정의되는 균일성 개체노드 타입 (uniformity entity node type)과 균일성 측면노드 타입(uniformity aspect node type) 그리고 Farming 시 ILB 컴포넌트에 해당하는 속성을 지니는 IM-컴포넌트 타입 노드 (IM component type node : ILB Multiplicity Component type node), OLB 컴포넌트의 속성을 지니는 OM-컴포넌트 타입 노드(OM component type node: OLB Multiplicity component type node) 등의 구성요소들과 generalization, aggregation, multiplicity 등의 3가지 추상화 개념들로 구성된다. 이들 중 IM-컴포넌트 타입 노드와 OM-컴포넌트 타입 노드는 다중성 개념과 관련되며 다중성 집합(Multiplicity set)의 요소들이 된다. IM-컴포넌트 타입노드와 OM-컴포넌트 타입노드들은 다중성 인스턴스 링크(Multiplicity Instance Link)에 의해 상위 개체노드와 연결이 된다. 그림 2.1은 이러한 Farmer 모델의 개념을 Farmer 모델 다이어그램을 이용하여 보여주고 있다.

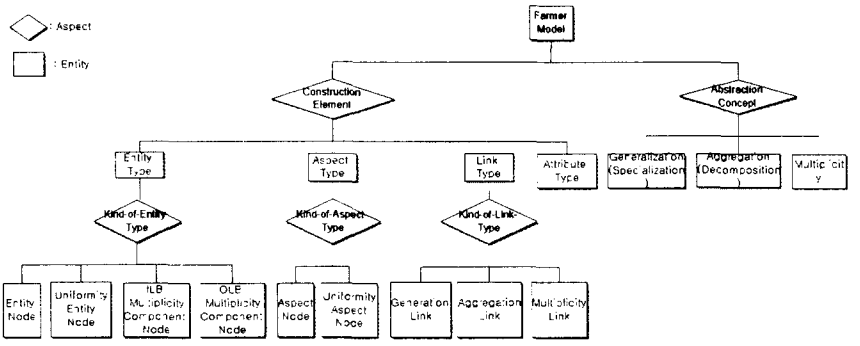

그림 2.1 Farmer 모델의 기본개념 분류

IS-A 관계를 나타내는 generalization 은 공통의 성질을 갖는 서브클래스들을 하나의 클래스로 통합하는 추상화개념이며 specialization 은 이 개념의 반대 개념으로 하나의 클래스를 상호배제적인 서브 클래스들로 분류하는 추상화 개념이다. Specialization 개념은 개체 성질의 상속성(inheritance)을 포함한다. Aggregation은 A-PART-OF 관계로서 여러 개의 개체나 개념을 서로 통합하여 새로운 하나의 통합된 개체나 개념을 만들어 내는 추상화 개념이며 이 개념의 역관계를 갖는 개념은 decomposition이 된다.

Multiplicity 는 하나의 개체를 구성하기 위하여 동일한 형태의 구성요소가 여러 번 발생하는 경우에 이의 대표적인 요소만을 표시하는 추상화 개념이다. Farmer 모델의 구성요소와 기호는 표 2.1과 같다.

Farmer 모델에서의 개체노드는 독립적으로 식별되는 구성원소 또는 실세계의 대상체(real world object)를 분할시킬 때 그 대상체를 구성하는 구성원소가 된다. 측면노드는 한 개체를 여러 관점으로 관찰할 때 관점개개가 된다. 따라서 한 측면노드에서의 개체노드들은 그 측면에서의 구조적 관계를 나타내는 분할된 구성원소가 된다.

Farmer 모델의 주요 특징은 디자인 완료된 시스템을 구현시 이미 컴포넌트웨어 형태로 만들어져 있는 소프트웨어 및 데이터 요소들을 LAN 이나 인터네트를 통하여 외부의 저장소 (repository)로부터 아웃소싱(outsourcing)해 가지고 오는 것이다. JAVA머신과 같은 네트워크 컴퓨터 (NC : Network Computer)의 경우에는 Farmer 모델에 의하여 디자인된 시스템을 구현시 모든 소프트웨어 블록들을 아웃소싱해야 하며, 아웃소싱이 필요없는 경우에는 OM-컴포넌트 타입 노드와 IM-컴포넌트 타입노드가 필요없게 된다.

2. Farmer 모델 형식론

2.1 기본공리

Farmer 모델의 개체 및 측면 노드, 추상화 개념들은 다음과 같은 기본공리들을 만족한다.

【 공리 2.1 】 동일한 이름을 가지는 두 노드는 동일한 속성과 동형의 서브트리를 갖는다.

【 공리 2.2 】 트리의 통로 상에 같은 노드가 두 번 이상 나타날 수 없다.

【 공리 2.3 】 OM 컴포넌트 타입과 IM 컴포넌트타입은 다중성 인스턴스 링크에만 연결될 수 있다.

【 공리 2.4 】 각 노드는 개체타입 노드이거나 측면타입 노드이어야 한다. 한 노드의 모드와 그 노드의 successor 등의 모드는 항상 반대모드이며 루트노드는 항상 개체노드이어야 한다. 단, 다중성 링크 및 다중성 인스턴스 링크에 의해 연결이 이루어지는 경우는 예외로 한다.

【 공리 2.5 】 같은 부모를 갖는 두 노드는 같은 이름을 가질 수 없다.

【 공리 2.6 】 동일한 개체타입에 속하는 노드는 같은 이름을 가질 수 없다.

【 공리 2.7 】 모든 측면객체(Aspect-Object)들은 모델 전체에서 유일하게 식별되는 식별자 (AO identifier)를 갖는다.

2.2 형식론

Farmer 모델의 형식구조는 다음에 내려지는 정의 및 정리들에 의하여 구성된다. Farmer 모델에서 실세계의 객관적, 추상적 대상체는 개체노드로 표현하며 이는 다음과 같이 정의된다. 개체노드를 구조로서 정의하는 이유는 실세계의 대상체를 표현하기 위하여는 최소한 대상체의 이름, 대상체가 가지는 속성 그리고 대상체를 바라보는 관점 등의 요소가 반드시 명기되어야 하며 나아가 Farming 의 개념을 반영하기 위하여 로딩 타입에 대한 속성 또한 필요로 하기 때문이다. 여기서의 structure는 predicate logic 이나 first order logic에서 언급하는 structure의 개념은 아니다.

【 정의 2.1 】 Entity Node Structure

E를 ***개체노드구조의 집합***이라고 하면, E 는 다음과 같이 정의된다.

$$\forall e \in E,\ e = < Eid, A, Sid, LT>$$

where, Eid : 개체노드 e의 이름

A : Attribute Set (**정의 2.2** 참조)

Sid : e가 가지는 view로서 측면노드 이름(id)의 집합을 의미 (**정의 2.3** 참조)

LT : 개체의 loading type로서 다음과 같은 집합으로 정의된다.

Construction Elements & Abstraction Concepts	Symbols
Entity	Entity Name
Uniformity Entity	Entity Name
Aspect	Aspect Name
Uniformity Aspect	Aspect Name
OM-Component	O Component Name
IM-Component	I Component Name
Attribute	~ Attribute Name
Decomposition Link (Aggregation Link)	
Specialization Link (Generalization Link)	
Multiplicity Link	
Multiplicity Instance Link	
Connector	

표 2.1 Farmer 모델 구성기호

LT = { Dynamic, Static, Dynamic-Static, none }

■

정의 2.1의 개체노드구조에 사용되는 속성의 개념은 다음과 같이 정의된다. 개체노드구조의 속성집합은 Farmer 모델의 specialization 추상화 개념정의에 중심적 역할을 담당한다. specialization 링크에 의해 의하여 추상화된 개체노드들은 상위 노드(클래스)의 속성집합을 그대로 *inherit*하기 때문이다.

Farmer model에서는 Entity Node Structure를 구성하는 요소 중 속성(A)을 될 수 있는 대로 간략하게 정의하고 있다. 그 이유로는, 본 개체를 독립된 클래스 저장소로부터 동적/정적으로 아웃소싱하는지에 대한 Farming 개념을 반영하는 로딩 타입(LT)의 속성을 반영하는 요소를 따로 독립시켰기 때문이다.

【 정의 2.2 】 Entity Node Structure 의 Attribute Set

E의 *attribute set* A는 다음과 같은 구조로 정의된다.

$\forall a \in A,\ a = <$ Aid, AT $>$

where, Aid : 속성명

AT : 속성타입 집합으로 다음과 같이 정의된다.

AT = { char, string, integer, real, boolean }

■

이상 **정의 2.1**과 2.2에 의하여 정의되는 개체노드와 속성집합의 예를 들어보면 다음과 같다.

【 예제 】 개체(Entity)의 예

그림 2.2의 TMN 에이전트를 설계시 루트노드에 해당하는 TMN_Agent_ System 노드는 다음과 같이 정의된다.

$E = \bigcup_{i=1}^{n} \{ e_i \}$ 에 대하여 , 다음과 같이 정의되는 임의의 e_i 가 존재한다.

$e_i = <$ TMN_Agent_System, A, Sid, none $>$

where, A = { a_1, a_2, a_3 }

$a_1 = <$ network_id, string $>$

$a_2 = <$ number_of_agents, integer $>$

$a_3 = <$ connectiveness, boolean $>$

Sid = { Function, Communication_Protocol, Configuration }

위의 예는 다음과 같은 의미를 가진다. e_i 의 이름은 TMN_Agent_System 로 정의되며, A 라는 속성집합을 갖는다. A 는 다음과 같은 구성원소로 구성된다.

- network_id : 네트워크 이름을 의미하며 string 속성타입을 갖는다.
- number_of_agents : 본 TMN 시스템에 존재하는 에이전트의 수를 의미한다. integer 속성타입을 갖는다.
- connectiveness : 다른 망과의 연동과 관련된 속성으로 연동여부에 대한 true/false의 값을 갖는다. boolean 속성타입을 갖는다.

또한 TMN_Agent_System 개체노드는 기능 측면(Function Aspect), 통신 프로토콜 측면(Communication_Protocol Aspect), 형상 측면(Configuration Aspect) 의 3 관점을 Sid의 원소로 갖는다. TMN_Agent_System 노드는 루트노드 이기 때문에 LT (loading type) 은 none이 된다.

■

측면은 **정의 2.1**의 개체노드구조에 의하여 모델링된 실세계의 대상체를 분석할 때의 관점을 나타낸다. 측면노드는 개체의 분할을 나타내며 **공리 2.4**에 의하여 하나의 측면노드에서 0 이상의 분할이 가능하며, 각 분할에서는 0개 이상의 개체노드 타입의 offspring이 가능하다.

【 정의 2.3 】 Aspect Node Structure

S를 ***측면구조의 집합***이라고 할 때 이는 다음과 같이 정의된다.

$\forall s \in S,\ s = <$ ASPid, OWNER $>$

where, ASPid : 측면 s의 이름

OWNER : s를 측면으로 갖는 개체노드이름 집합 ■

【 예제 】 Aspect Node의 예

그림 2.2에서 Function 측면노드는 다음과 같이 정의된다.

∀s ∈ S 에 대하여 , 다음과 같이 정의되는 임의의 s 가 존재한다.

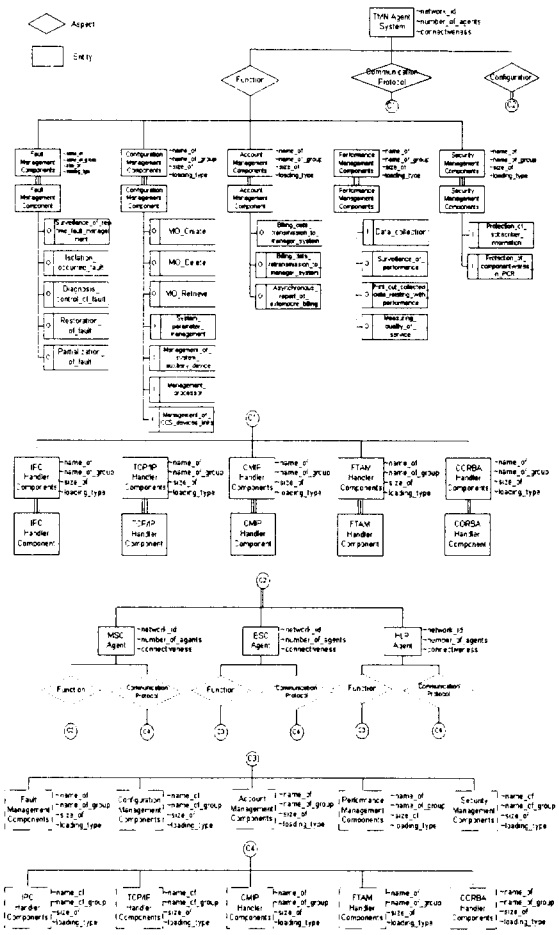

그림 2.2 Farmer 모델에 의한 구성

where, s = < Function, OWNER >
OWNER = { TMN_Agent_System }

위의 예에서 측면객체 s 의 이름은 Function 으로 정의되며, TMN_Agent_ System 개체노드가 s를 측면으로 갖는다. s 의 OWNER는 TMN_Agent_ System 개체노드가 된다. ■

Farmer 모델에서 사용하는 개체의 생성과 삭제에 관련된 알고리즘은 다음과 같으며 개체관리의 일관성을 유지하기 위하여 다음과 같이 개체의 속성을 개체의 생성시에 부여한다.

[알고리즘 2.1] create_entity(Entity) // Creation of new entity

1 개체(Entity)구조들의 집합 $E = \bigcup_{i=1}^{n} \{ e_i \}$ 에 대하여

FOR i = 1 **TO** n
1.1 **IF** Entity.Eid = e_i.Eid **THEN**
1.1.1 error_print(" Already exist... ");
1.1.2 exit;
1.2 **END IF**
END FOR

// 속성의 부여
2 e_{n+1}.Eid ← Entity.Eid
e_{n+1}.A ← Entity.A
e_{n+1}.S ← Entity.S
e_{n+1}.LT ← Entity.LT

3 E ← E ∪ { e_{n+1} }

[알고리즘 2.2] delete_entity(Entity)
// Delete of current entity
// 개체의 삭제시 개체가 가진 속성과 개체의 관계도 함께 삭제된다.

1 개체(Entity)구조들의 집합 $E = \bigcup_{i=1}^{n} \{ e_i \}$ 에 대하여
1.1 **IF** there_exist(Entity) = FALSE **THEN**
1.1.1 error_print(" There does not exist..... ");
1.1.2 exit;
1.2 **END IF**

2 E ← E - { Entity }

3 E의 subtree를 제거한다.

Generalization은 specialization되어 있는 여러 개체들을 하나의 개체로 일반화시키는 개념으로 하위 클래스의 여러 개체들을 상위 클래스의 개체로 표현할 수 있도록 하는 abstraction concept을 의미하고 Aggregation abstraction concept은 하위 클래스의 개체들이 상위 클래스의 개체의 원소로 구성된다는 추상화 개념을 나타낸다. 이에 대한 정의는 생략하기로 한다.
여기서 Aggregation 및 generalization 추상화 개념과 관련된 다음과 같은 정리가 성립된다.

【 정리 2.1 】 Aggregation 및 generalization 추상화 개념으로 연결되는 상위클래스 개체노드 Esup 와 하위클래스 Esub 사이에는 다음과 같은 관계가 성립한다.
| level(Esup) - level(Esub) | = 2 ■

[증명] Farmer 모델에서 상위클래스 개체노드 Esup의 level을 n 이라고 하면

level(Esup) = n

Esup에 aggregation 및 generalization 등의 추상화 개념을 적용하여 하위 클래스 노드 Esub를 생성하기 위하여는 Farmer 모델의 **공리 2.4**에 의하여 임의의 측면개체 S를 거쳐야 한다. 따라서 S에 대한 level 및 Esub에 대한 level은 다음과 같이 정의된다.

level(S) = n+1
level(Esub) = n+2

이에 따라

| level(Esup) - level(Esub) | = | n - (n+2) | = 2

가 성립한다.
■

다음은 실제로 Farming 개념을 포함하는 개체노드인 OM-컴포넌트 타입 개체구조(OLB Multiplicy Component Type Entity Structure) 및 IM-컴포넌트 타입개체구조(ILB Multiplicy Component Type Entity Structure)에 대한 정의를 내려보기로 한다. 이와 같은 IM/OM-컴포넌트 타입개체들은 Farmer 모델에서 실제로 PICR로 부터 네트워크를 통하여 TMN 에이전트로 Farming되는 대상체인 소프트웨어/데이터 컴포넌트웨어를 의미한다. OM-컴포넌트 타입 개체의 경우는 Farming 방법론에서 에이전트의 요구에 의하여 동적으로 로딩이 이루어지는 컴포넌트웨어를 모델링한 개체노드이다. IM-컴포넌트 타입 개체는 에이전트의 설립시 초기에 에이전트 플랫폼으로 다운로딩되어 실행이 이루어지는 컴포넌트웨어를 모델링한 개체노드이다.

【 정의 2.4 】 OM-컴포넌트 타입 개체(Entity) 구조

개체(Entity)구조들의 집합 $E = \bigcup_{i=1}^{n} \{ e_i \}$ 의 similarity type E_0는

$e = < Eid, A, S, LT >$

where, ∃e ∈ **E** e.LT = Dynamic
A : Entity Node Structure의 **Attribute Set** (**정의 2.2 참조**)
LT : 개체의 loading type 집합 (**정의 2.1 참조**)
S : 측면노드 구조집합 (**정의 2.3 참조**)

를 만족하는 sequence로 정의된다. 이러한 similarity type E_0를 ***OM-컴포넌트 타입 개체(Entity) 구조***라 한다.
■

[예제] OM-컴포넌트 타입 개체(Entity)의 예
그림 2.2의 Fault_Management_Component 노드에 다중성 인스턴스 링크로 연결되어 있는 OLB 타입의 Surveillance_of_realtime_fault_management 노드는 다음과 같이 정의된다.

∀e ∈ E_0 에 대하여 , 다음과 같이 정의되는 임의의 e 가 존재한다.

$e = <$ Surveillance_of_realtime_fault_management, A, none, Dynamic $>$

where, $A = \{ a_1, a_2, a_3, a_4 \}$
$a_1 = <$ name_of, string $>$ // 컴포넌트 요소의 이름
$a_2 = <$ name_of_group , string $>$
// 다중성에 의하여 본 노드의 leaf 노드에 연결되는 컴포넌트요소들이 PICR에 저장시 결정되어 있는 그룹 이름
$a_3 = <$ size_of, integer $>$ // 컴포넌트 요소의 file size
$a_4 = <$ loading_type, integer $>$
// 0 : dynamic loading
// 1 : static loading
// 2 : dynamic-static loading
// 3 : none
■

【 정의 2.5 】 IM-컴포넌트 타입 개체(Entity) 구조

개체(Entity)구조들의 집합 $E = \bigcup_{i=1}^{n} \{ e_i \}$ 의 similarity type E_1은

$e = <$ Eid, A, S, LT$>$

where, ∃e ∈ **E** e.LT = Static
A : Entity Node Structure의 **Attribute Set** (**정의 2.2 참조**)
LT : 개체의 loading type 집합 (**정의 2.1 참조**)
S : 측면노드 구조집합 (**정의 2.3 참조**)

를 만족하는 sequence로 정의된다. 이러한 similarity type E_1를 ***IM-컴포넌트 타입 개체(Entity) 구조***라 한다. ■

[예제] IM-컴포넌트 타입 개체(Entity)의 예
그림 2.2의 Configuration_Management_Component 노드에 다중성 인스턴스 링크로 연결되어 있는 ILB 타입의 System_Parameter_Management 노드는 다음과 같이 정의된다.

∀e ∈ E_1 에 대하여 , 다음과 같이 정의되는 임의의 e 가 존재한다.

$e = <$ System_Parameter_Management, A, none, Static $>$

A에 대한 정의는 **OM-컴포넌트 타입 개체(Entity)의 예제**와 같다. ■

여기서 개체구조와 IM/OM-컴포넌트 타입 개체사이에 다음과 같은 정리들이 성립이 된다.

【 정리 2.2 】 OM-컴포넌트 타입 개체(Entity)집합을 E_0라 개체구조집합을 E라 하면

$E_0 \subset E$

가 반드시 성립한다.
■

[증명] **정의 2.1**에 의하면 개체구조들의 집합 $E = \bigcup_{i=1}^{n} \{ e_i \}$는

∀e ∈ E, where e.LT = Static, Dynamic, Static-Dynamic, none

를 모두 만족한다. 한편 E_0 는

$\forall e_0 \in E_0$, where $e_0.LT$ = Dynamic

의 조건만을 만족하는 E의 similarity type sequence를 의미한다. 이에 따라 E_0 와 **E** 사이에는 다음과 같은 차집합 관계가 성립된다.

E_0 = **E** - { for $\exists e \in$ **E** ¦ e.LT = Dynamic }

즉, $E_0 \subset$ **E** 의 관계가 성립하게 된다. ■

【 정리 2.3 】 IM-컴포넌트 타입 개체(Entity)집합을 E_1라 하고 개체구조집합을 **E**라 하면

$E_1 \subset$ **E**

의 관계가 반드시 성립한다. ■

[증명] **정의 2.1**에 의하면 개체구조들의 집합 $E = \bigcup_{i=1}^{n} \{ e_i \}$는

$\forall e \in E$, where e.LT = Static, Dynamic, Static-Dynamic, none

를 모두 만족한다. 한편 E_1 는

$\forall e_1 \in E_1$, where $e_1.LT$ = Static

의 조건만을 만족하는 E의 similarity type sequence를 의미한다. 이에 따라 E_1 와 **E** 사이에는 다음과 같은 차집합 관계가 성립된다.

E_1 = **E** - { for $\exists e \in$ **E** ¦ e.LT = Static }

즉, $E_1 \subset$ **E** 의 관계가 성립하게 된다. ■

ILB/OLB 다중성 컴포넌트 타입 개체집합의 요소사이에는 다음과 같은 정의가 성립된다.

【 정의 2.6 】 Farmer 모델 형식구조

Farmer 시멘틱 structure 는 ordered sequence로 정의되며 **공리 2.1 - 2.7**을 만족하며 **정의 2.1 - 정의 2.5**에 기본을 둔다.

Farmer 모델 형식구조 **F**는 다음과 같이 정의된다.

F = < C, AO, E, S, λ_1, λ_2, λ_3, CNST >

where, C : 컴포넌트 집합
AO : 측면객체 집합
E : 개체 집합
S : 측면구조들의 집합
측면객체 AO의 측면을 $s_1, s_2, \ldots, s_n$ 이라 하면, AO의 측면집합 S는 다음과 같이 표기된다.

$$S = \bigcup_{i=1}^{n} \{ s_i \}$$

λ_1 : C → AO, λ_1(c) = **Aspect_Object (c)**,
λ_2 : E ∪ S → AO,
for $\forall$h ∈ **E** ∪ **S**,
λ_2(h) = **Aspect_Object (h)**
λ_3 : C → AO
for $\exists$c ∈ **C** in PICR
λ_3(c) = **moveto(AO** in an Agent_platform)
CNST : Farmer 모델구조에서 사용하는 상수(constant)들의 집합으로 **정의 2.6**과 같이 정의된다 ■

【 정의 2.6 】 Farmer 모델 형식구조에서 사용하는 ***상수들의 집합*** CNST는 다음과 같은 특성을 갖는 the smallest set이다.

1. $\overline{c_i} \in C$ for $i \in I \Rightarrow \lambda_1(\overline{c_i}) \in AO$
2. $\overline{k_i} \in E \cup S$ for $i \in I \Rightarrow \lambda_2(\overline{k_i}) \in AO$

■

Farmer 모델 **공리 2.1**에 의해 정의되는 균일성 개체와 균일성 측면은 표 4.1에서 보는 바와 같이 점선으로 표시되며 다음과 같다.

【 정의 2.7 】 균일성 개체노드 (uniformity entity node)

U^E를 ***균일성 개체(Entity)노드의 집합***이라고 할 때, ***균일성 개체(Entity)***원소 e_u은 다음과 같은 조건을 만족한다.

1. $U^E \subset E$
2. $\exists e_u \in U^E$, $\exists e \in E$ 에 대하여
   1) level (e_u) > level (e) (e_u 가 e 의 상위 level)
   2) e_u.Eid = e.Eid
   3) e_u.A ≡ e.A
   4) e_u.S ≡ e.S (본 객체노드에 측면객체가 존재할 경우)
   5) e_u.LT ≡ e.LT

E, A, S, LT : **정의 2.1** 참조 ■

균일성 개체란 Farmer 모델의 구성에서 동일한 이름을 가지는 두 개의 개체가 존재시, 균일성의 원리에 의하여 자신보다 상위에 있는 동일 이름의 개체로부터 그 개체의 속성과 서브트리를 복사해 가져오는 하위의 동일한 이름을 가지는 개체를 의미한다. 균일성 개체의 예는 다음과 같다.

[예제] 균일성 개체 (uniformity entity)의 예

그림 2.2에서 level 2에 있는 개체노드인 Fault_Management_Components 개체노드는 level 3에 나타나는 측면노드인 Function 측면노드의 하위레벨 노드로서 다시 나타난다. (연결자 C2 로 연결되어 있으며 level은 4가 됨) 균일성 개체 정의에 의하여 두 노드의 균일성을 고려해 보도록 하자.

level 2에서 처음으로 나오는 개체노드인 Fault_Management_Components를 e라 하고, level 4에 나타나

는 Fault_Management_Components 개체노드를 e_u 라 하자. e와 e_u 사이에는 다음과 같은 관계가 정의된다.

1. $e_u \in U^E$, $e \in E$
2. $U^E \subset E$
3. 1) level (e_u) = 4, level (e) = 2
   따라서 균일성 개체의 정의 2항 1)을 만족한다.
   2) e_u.Eid와 e.Eid 모두 Fault_Management_Components의 이름을 갖는다. 따라서 균일성 개체의 정의 2항 2)를 만족한다.
   3) e_u와 e는 동일한 속성집합을 갖는다. (**공리 4.1**에 의하여)
   4) Fault_Management_Components 개체노드는 측면을 가지고 있지 않음
   5) e_u와 e의 loading type은 모두 none 이다.

이상의 사실에 따라 e_u는 e의 균일성 개체가 된다. ■

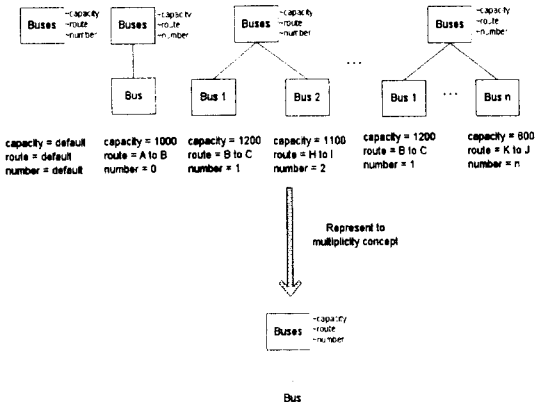

그림 2.3 다중성 개념의 예 1

【 정의 2.8 】 균일성 측면 (uniformity aspect)
U^S를 ***균일성 측면(Aspect)노드들의 집합***이라고 할 때, ***균일성 측면*** 원소 s_u는 다음과 같은 조건을 만족한다.

1. $U^S \subset S$
2. $\exists s_u \in U^S$, $\exists s \in S$ 에 대하여
   1) level (s_u) > level (s) (s_u 가 s 의 상위 level)
   2) s_u.ASPid = s.ASPid
   3) OWNER(s_u) ≠ OWNER(s) ■

다시 말해 균일성 측면이란 Farmer 모델의 구성에서 동일한 이름을 가지는 두 개의 측면이 존재시, 균일성의 원리에 의하여 자신보다 상위에 있는 동일 이름의 측면으로부터 그 측면의 속성과 서브트리를 복사해 가져오는 하위의 동일한 이름을 가지는 측면을 균일성 측면이라 한다.

하나의 개체를 구성하기 위하여 동일한 형태의 구성요소가 여러 번 발생하는 경우에 이의 대표적인 요소만을 표시하는 추상화 개념인 다중성에 대한 정의는 다음과 같다.

【 정의 2.9 】 다중성 (Multiplicity)
개체구조집합 E의 요소를 $e_1, e_2, \ldots, e_n$ 이라 할 때 E는 다음과 같이 정의되며 $E = \bigcup_{i=1}^{n} \{ e_i \}$

이때, $\forall e \in E$ 인, e 가 동일한 속성을 가질 경우 E내의 임의의 e_i ($0 \le i \le n$)는 E의 모든 요소를 대표할 수 있다. 이를 ***다중성***이라 한다.

E : 개체구조노드 집합 (**정의 2.1** 참조) ■

[예] 다중성 (Multiplicity)의 예 1
도시교통관리 시스템에서의 Buses라는 개체는 용량(capacity : 버스가 태울 수 있는 사람 수), 노선(route), 번호(number) 등과 같은 속성을 지닌다. 그림 2.3에서 보는 바와 같이 Bus 1과 Bus 2는 각각의 속성들에 대한 실제 값만 다를 뿐 동일한 속성을 Buses와 같은 속성을 갖는다. 이렇듯 하나의 개체(ex : Buses)를 구성하기 위해 동일한 형태의 구성요소가 여러 번 발생하는 경우(ex : Bus 1, Bus 2,..., Bus n) 이들 중 대표적인 개체(ex : Buses)만을 표시하면 되는 것이 다중성의 개념이 된

그림 2.4 다중성 개념의 예 2

다. Farmer 모델에서는 다중성 개념에 적용되는 개체는 OLB 다중성 컴포넌트 타입노드(OM-component type node)와 ILB 다중성 컴포넌트 타입노드(IM-component type node)가 이에 해당한다. ■

【 정의 2.10 】 개체구조집합 E의 요소 중 $\exists e \in E$ 에 대하여 level (e) = 0 인 경우에도 다중성 추상화개념을 직접 적용할 수 있다.

E : 개체구조노드 집합 (**정의 2.1** 참조) ■

다시 말해 개체노드가 root node의 경우에도 다중성 추상화개념을 직접 적용할 수 있다.

[예] 다중성 (Multiplicity)의 예 2
군대(army)의 체계를 예로 들어보자. 군대(army)는 사단(division)들의 모임으로 구성되며 각 사단은 연대(regiment)들의 모임으로 구성된다. 또한 각 연대는 대대(battalion)들의 모임으로 구성된다. 이외 같이 하여 군대(army)는 개개의 병사(soldier)로 까지 분해(decompose) 되어진다. 여기서 각각의 연대를 구성하는 사단(division)들은 모두 동일한 속성을 지니며

연대, 대대, 부대(unit), 병사 들 또한 각각 동일한 속성을 지니는 집합의 요소가 되며 이들은 다중성 개념에 의해 각 계층군(예 : 연대, 대대 등)을 대표하는 대표요소로 표현이 된다. 그림 2.4는 이와 같은 개념을 보여주고 있다. ■

다중성 추상화 개념에서 정의된 대표개체는 자신의 인스턴스들을 가질 수 있다. 이들 인스턴스들은 IM-컴포넌트 타입 개체노드 및 OM-컴포넌트 타입 개체노드이며 다중성 인스턴스 링크를 이용하여 대표개체와 연결된다. 다중성 인스턴스 링크의 정의는 다음과 같다.

【 정의 2.11 】 다중성 인스턴스 링크 (Multiplicity Instance Link)

다중성 인스턴스 링크 L_M는 다음과 같은 structure에 의해 정의된다.

$$L_M = < E_M, E_c, \varphi, CNST >$$

where, E_M : 다중성 링크에 의해 추상화된 개체노드 집합

E_c : 개체노드 집합 $E = \bigcup_{i=1}^{n} \{ e_i \}$ 의 원소 중 다음과 같은 특성을 가지는 원소들만을 원소로 갖는 개체노드 집합을 의미한다.

1) for $\forall e \in E$, e.LT = Dynamic $\vee$ e.LT = Static
(로딩타입이 Dynamic 이나 Static 인 원소들의 집합. 즉, OM-컴포넌트 타입 개체노드와 IM-컴포넌트 개체 타입 노드의 집합을 의미)

2) e_m = super_class_of(E_c)
(for $\forall e_c \in E_c$, e_c 의 상위클래스 개체노드는 e_m)

φ : (for $\exists e_m \in E_M$) $\rightarrow E_c$
φ(for $\exists e \in E$) = E_c

CNST : Farmer 모델구조에서 사용하는 상수(constant)들의 집합으로 여기서는 empty set이 된다. ■

【 예 】 다중성 인스턴스 링크의 예

그림 2.2에 존재하는 모든 다중성 링크에 의해 추상화된 개체노드 집합을 $E_M = \bigcup_{i=1}^{n} \{ e_{mi} \}$라고 하였을 때, 다음과 같이 정의되는 임의의 e_{mi} 가 존재한다.

e_{mi} = < Fault_Management_Component, A, none, none >

where, A = { a_1, a_2, a_3, a_4 }
a_1 = < name_of, string >
a_2 = < name_of_group, string >
a_3 = < size_of, integer >
a_4 = < loading_type, integer >

또한 E_c 는 다음과 같이 정의된다.
E_c = { e_{c1}, e_{c2}, e_{c3}, e_{c4}, e_{c5} }
where,
e_{c1} = < Surveillance_of_realtime_fault_management, A, none, Dynamic >
e_{c2} = < Isolation_occurred_fault, A, none, Dynamic >
e_{c3} = < Diagnosis_control_of_fault, A, none, Dynamic >
e_{c4} = < Restoration_of_fault, A, none, Dynamic >
e_{c5} = < Partialization_of_fault, A, none, Dynamic >

위에서 정의한 E_M 과 E_c에 대하여, 다음이 성립한다.

$$\varphi : (\text{for } \exists e_m \in E_M) \rightarrow E_c$$

즉, OM-컴포넌트 타입 개체노드인 Surveillance_of_realtime_fault_management, Isolation_occurred_fault, Diagnosis_control_of_fault, Restoration_of_fault, Partialization_of_fault 들은 다중성 인스턴스 링크에 의해 상위 클래스 개체노드인 Fault_Management_Component에 연결이 된다. ■

다중성 추상화 링크 및 다중성 인스턴스 링크와 관련하여 다음과 같은 정의가 내려진다.

【 정의 2.12 】 Farmer 모델에서 다중성 링크에 의해 개체 사이의 추상화가 이루어지는 경우 **공리 4.4에** 의해 개체노드사이의 연결이 가능하며 이때 하위 클래스의 개체노드를 ***대표개체***라 한다. ■

III. 결론

본 논문에서는 실세계의 개체를 각각의 고정된 측면이 아닌 여러 관점의 측면에서 분석한 후 분석완료된 측면요소들을 측면객체로 정의하여 시스템을 설계할 수 있는 Farmer 모델의 내용, 특징에 대하여 서술하였다. Farmer 모델의 주요 특징은 디자인 완료된 시스템을 구현시 이미 컴포넌트웨어 형태로 만들어져 있는 소프트웨어 및 데이터 요소들을 LAN 이나 인터네트를 통하여 외부의 저장소(repository)로부터 아웃소싱(outsourcing)해 가지고 오는 것으로서 특히 본 논문에서는 Farmer 모델 중 다중성 인스턴스 추상화에 기반한 통신망 관리 시스템 설계 모델 형식론에 대하여 설명하고 예를 들었다.

참고문헌

[1] Ivar Jacobson, *Object-Oriented Software Engineering, A Use Case Driven Approach*, Addison-Wesley, 1992.

[2] James Rumbaugh and Michael Blaha, *Object-Oriented Modeling and Design*, OMG, 1991

[3] Ann L.Winblad, Samuel D. Edwards, and Davis R. King, *Object-Oriented Software*, Addison-Wesley, 1992.

[4] Soo-Hyun Park, Doo-Kwon Baik, "Platform Independent TMN Agents Based on the Farming Methodology", The IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, The Institute of Electronics, Information and Communication Engineers (IEICE), pp.1152 - 1163, Japan, 1998

[5] Soo-Hyun Park, Doo-Kwon Baik, "The Construction of Network Management System by Component Outsourcing Concept", *In Proceedings of The 4th Joint International*

Conference on Computer Science and Informatics (JCS&I'98), Volume III, pp.23 - 26, Research Triangle Park, NC, U.S.A., 1998

[6] Soo-Hyun Park, Doo-Kwon Baik, "Adaptive Modeling of Distributed Agents by the Farmer Model", *In Proceedings of The 6th IEEE International Workshop on Intelligent Signal Processing and Communication Systems (ISPACS'98), IEEE Computer Society*, Volume 2, pp.928 - 932, Melbourne, Australia, 1998

[7] Zeigler B. P, *Multifaceted Modeling and Discrete Event Simulation*, Academic Press, 1984.

[8] ComponentWare Consortium, "ComponentWare ArchI tecture : A technical product description", *I-Kinetics, Inc*, 1995.

[9] Robert Orfali, Dan Harkey, and Jeri Edwards, *The Essential Distributed Objects, Survival Guide*, John Wiley & Sons, Canada, 1996.

[10] Robert Orfali and Dan Harkey, *Client/Server Programming with JAVA and CORBA*, John Wiley & Sons, Canada, 1996.