ALFReD: 재귀적 설계를 위한 구조기술언어

최윤석, 강성원
한국정보통신대학교 공학부
대전광역시 유성구 문지로 119
{yschoi, kangsw}@icu.ac.kr

요약: 본 연구에서는 소프트웨어구조기술언어인 ALFReD를 소개한다. ALFReD는 논리관점의 구조기술언어로서, 시스템 혹은 하위시스템의 형식적 기능명세를 메시지순차도로 작성하며, 재귀적 구조설계를 지원한다는 특징이 있다. 이러한 특징으로 인하여 ALFReD를 사용하여 기술한 논리적 구조는 다른관점의 구조 도출의 기반이 될 수 있으며, 논리관점의 구조도출과정에 모든 하위시스템들이 형식언어로 기술된 자신의 명세를 갖게 되며, 재귀적 설계로 인하여 구조결정을 단계적으로 수행하는 효과를 얻을 수 있다.

핵심어: Architecture View, ADL, Recursive Design, MSC, Formal Specification

1. 서론

소프트웨어개발에 있어서 소프트웨어구조의 중요성이 인식되면서 소프트웨어구조기술언어(ADL: Architecture Description Language)에 대한 많은 연구가 진행되어 왔다. 일반적으로 소프트웨어의 구조는 하나의 관점(view)으로서 포착할 수 있는 것이 아니라, 시스템의 구조를 적절히 표현하기 위하여 논리관점(Logical view), 실행관점(Execution view), 개발관점(Development view), 배치관점(Deployment view)등 여러가지 관점에서 시스템의 구조를 보아야 한다고 인정된다.[1] 다양한 구조관점 중에서도 소프트웨어의 가장 기본적인 구조정보를 담는 것이 논리관점이다.

논리관점의 구조에서는 시스템을 하위시스템들로 구성된 것으로 보고 전체시스템과 하위시스템간의 관계와 하위시스템 간의 관계에 관심을 갖는다. 그 밖의 다른 구조정보는 필요에 따라 다른 관점을 통하여 얻을 수 있다. 본 연구에서는 논리관점을 구조의 가장 본질적인 관점으로 보고, 따라서 무엇보다도 논리관점의 구조를 적절히 표현해 줄 수 있는 ADL이 우선적으로 필요하다고 본다.[2]

이러한 이유로 ADL은 다음의 바람직한 성질을 가져야 한다. (1) 먼저 논리관점을 표현할 수 있어야 한다. (2) 특히 복잡한 시스템을 표현하기 위하여 여러 개의 단계(Multi-layer)로 시스템의 논리적 구조를 표현할 수 있어야 한다. (3) 나아가 다단계의 구조기술이 동일한 방식으로 이루어질 수 있어야 한다. 마지막 성질은 한 단계가 다른 단계로 상세화(refine)되는 과정에 재귀적 설계(recursive design)를 ADL이 지원할 수 있도록 하기 위하여 필요하다. 우리는 논리적구조의 정립과정이 여러단계에 걸쳐 이루어지는

[1] Kruchten은 논문 [1]에서 시나리오관점(Scenario View)을 포함하여 '4+1 view'를 소프트웨어구조를 위한 핵심적인 관점으로 채택하였다. Kruchten의 논리관점(Logical View)은 본 연구의 논리관점과 대체로 같은 개념으로 볼 수 있고 Kruchten의 프로세스 관점(Process View)은 본 연구의 실행관점에 해당된다.

[2] 하나의 ADL이 하나의 관점의 구조를 표현하여야 하느냐, 아니면 하나의 ADL로 여러가지 관점을 기술할 수 있어야 하느냐 하는 질문이 제기될 수 있다. 지금까지 제안된 ADL은 대체로 하나의 관점을 표현하도록 설계되었으나 그 언어가 지원하는 관점에 대하여 대체로 불분명하였다. 본 연구에서 제안하는 ADL인 ALFReD는 하나의 관점을 표현하는 언어이고, ALFReD가 지원하는 관점은 논리관점임을 밝혀둔다.

절차(process)이고 적절한 ADL은 이러한 과정의 중간적인 구조설계결과를 보여줄 수 있는 언어, 즉 재귀적설계를 지원하는 형식언어가 되어야 한다고 본다.

본 논문에서는 이러한 특징들을 갖는 ADL인 ALFReD (Architecture description Language for Recursive Design)를 제시한다. ALFReD는 메시지순차도로 기술된 시스템 명세에서 출발하여, 최상위 레벨의 구조가 결정되면 시스템명세가 각 하위시스템의 명세로 상세화되어 각 하위시스템으로 넘겨져서 더 상세한 구조적 결정들이 다음 단계에서 이루어 질 수 있도록 재귀적설계(recursive design)를 지원한다. 각 하위시스템은 이를 명세로 갖는 하나의 시스템이 됨으로써 이러한 과정의 반복에 의하여 더 이상 하위시스템으로 분할할 필요성이 없는 최하위시스템(unit system)에 궁극적으로 도달하게 되고 이로써 구조의 상세화과정은 끝나게 된다.

본 논문의 나머지는 다음과 같이 구성된다. 제 2절에서는 본 연구에서 기능요구사항을 기술하기 위하여 사용하는 형식언어인 메시지순차도를 소개한다. 제 3절에서는 본 연구가 제안하는 논리관점의 구조기술언어인 ALFReD를 소개한다. 제 4절에서는 ALFReD를 실제 소프트웨어 시스템의 구조설계에 적용하여 ALFReD가 구조기술언어로서 적절한 언어임을 보인다. 제 5절에서는 관련연구를 검토하고, 마지막으로 결론에서는 본 논문의 공헌과 향후연구방향을 기술한다.

2. 메시지 순차도

본 연구에서는 개발공정이 사용자요구사항에서부터 출발하여 시스템명세가 도출되고 이로부터 소프트웨어구조를 설계한 후 상세설계를 거쳐 구현이 이루어진다고 본다. 실제 개발공정은 반복적인 방법을 사용할 수 있다. 그러나 각 반복과정에서도 이러한 논리적인 순서는 지켜진다. 또한 구조를 개발자동화의 한 단계로 사용할 경우에도 역시 이러한 순서는 지켜지게 된다. 실제공정에 대응되는 개념으로써 이와 같은 공정을 본 논문에서는 논리적 공정이라고 부른다.

또한 본 연구에서는 시스템명세가 데이터모델과 메시지순차도와 같은 형식언어로 기술되어 있다고 가정한다. 데이터모델은 시스템에서 사용되는 개념적 데이터유형을 모두 포함한다. 메시지순차도(message sequence chart)는 개체(entity)들 간의 상호작용(interaction)을 표현하는 ITU-T 표준의 형식언어이다 [3]. UML에서는 순차도(Sequence Chart)라는 명칭을 사용한다. 메시지순차도의 한가지 장점은 개체간의 상호작용을 시간적인 순서에 맞게 표현하는데 용이하다. 행위를 기술하는 형식언어인 상태도(statechart)의 경우 각 개체의 행위를 기술하는데는 좋으나 하나의 개체의 행위를 기술하기 때문에 다양한 개체간의 상호작용을 표현하기에는 적절하지 않다. 메시지순차도의 또 다른 장점은 사용자 요구사항이 UML에서 사용하는 use case 시나리오와 같은 비형식언어가 사용될 경우 쉽게 메시지순차도로 번역될 수 있다는 점이다.

UML의 순차도에서는 상호작용의 주체가 객체(object)이다. 본 연구에서는 메시지순차도의 상호작용의 주체가 객체 이외에도 개발대상인 시스템, 하위시스템, 유닛시스템들이 될 수 있도록 하여 메시지순차도를 포괄적인 상호작용 기술의 언어로 사용한다.

나아가 본 연구에서는 외부개체와의 상호작용과 이 상호작용을 통하여 외부에 전달된 결과가 시스템의 사용자가 기능의 관점에서 시스템에 대하여 관심을 갖는 전부라고 본다. 하위시스템의 경우에도 마찬가지로 하위시스템의 사용자인 시스템의 관심은 상호작용을 통하여 전달된 하위시스템의 처리결과이다.

메시지순차도는 그림 2.1과 같이 시간의 흐름에 따는 하위시스템간[3]의 기본적인 상호작용과 내부적인 계산 및 처리를 표현할 수 있을 뿐 아니라, 그림 2.2와 2.3에서 보는 바와 같이 선택적 행위와 반복적 행위를 표현할 수 있다. 그림 2.1에서 하위시스템 C는 특히 계산(computation)을 포함하고 있다. 본

[3] 하위시스템은 전체시스템과 유닛시스템을 포함한다.

연구에서 사용하는 메시지순차도는 상호작용의 방법에서는 기본적으로 UML의 순차도를 채택하나, 중요한 차이점을 상호작용의 주체를 UML에서는 객체로 한정하는데 비하여 본 연구에서는 시스템, 하위시스템, 단위시스템 혹은 컴포넌트와 같이 특정한 프로그래밍스타일로부터 독립적인 논리적 구조의 단위를 포함한다는데 있다. 뒤에서 설명하는 바와 같이, 상호작용의 주체는 상세화를 통하여 설계의 진행과 함께 구체화 된다.[4]

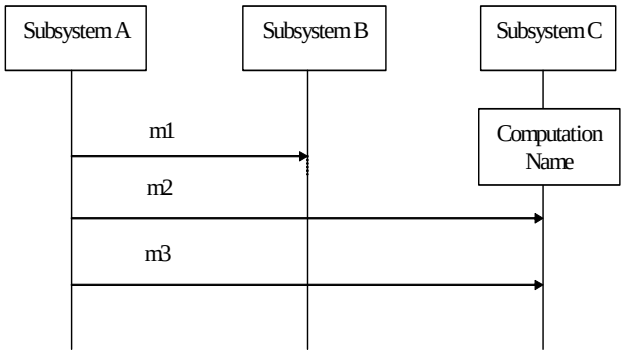

그림 2.1 기본적인 메시지순차도의 예

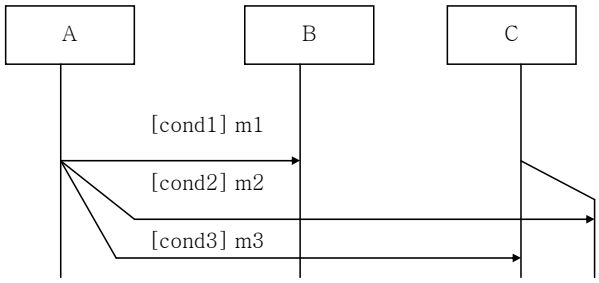

그림 2.2 조건적 행위의 표현

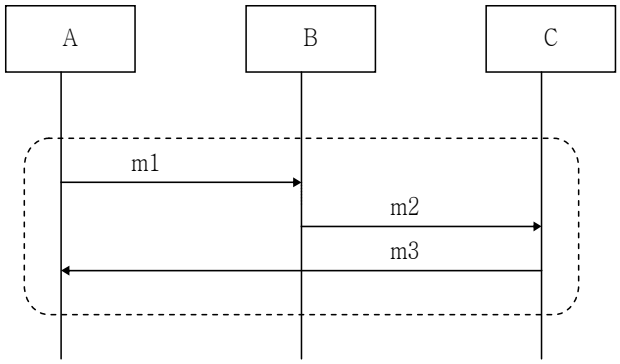

그림 2.3 반복행위의 표현

그림 2.1, 2.2와 2.3에서는 ALFReD의 메시지순차도가 상호작용의 일반적인 시나리오를 표현할 수 있을 만큼 일반적임을 보여준다. 우리는 여기서 메시지순차도의 구체적인 메시지의 형태는 제시하지 않는다. 메시지순차도에 사용가능한 메시지의 형태는 데이터모델링단계에서 정의된 데이터로서 상호작용의 대상이 되는 모든 데이터가 된다.[5]

3. ALFReD: 논리관점의 구조기술 언어

3.1 논리관점

논리관점(Logical View)에 대하여 보편적으로 합의된 정의는 없다. 그러나 Kruchten의 '4+1 view'의 정의에서는 논리관점을 "문제 영역을 바탕으로 시스템이 사용자들에게 어떤 서비스들을 제공해야 하는지를 주요한 추상화된 형태로 나눈 구조이다" 으로 정의하고 있다 [1]. [6] Hofmeister 은 저서 [2]에서 소프트

[4] 메시지순차도로 객체혹은 프로세스와 같은 컴퓨팅개체(computing entity)의 생성, 동기화, 소멸도 표현할 수 있다. 본 연구에서는 이러한 컴퓨팅 개체(entity)에 관계되는 사항은 실행관점의 구조에서 고려되어야 하는 사항으로 논리관점의 고려대상의 범위를 벗어나는 것으로 본다.

[5] 데이터모델링과 논리관점의 구조정립이 서로 얽힌(interwined) 최적화된 구조정립 공정이 실용적으로 사용될 수 있는 구조정립공정으로 볼 수 있다. 그러나 본 연구에서는 논리적 공정을 정립한 뒤에 사용자가 이를 적절히 응용하여 실용적 공정을 만들어 갈 수 있다고 본다. 따라서 데이터모델링이 논리적으로 선행되고, 구조정립은 이에 종속된는 후속단계로 본다.

[6] "The logical architecture primarily supports the functional requirements – what the system should provide in terms of services to its users. The system is decomposed into a set of key abstractions, taken

웨어구조의 관점으로 'Siemens four view'를 제시하였으며 네 개의 관점 중에서 개념관점(conceptual view)을 "응용영역(Application Domain)과 가장 근접한 관점으로 시스템을 상호연결된 개념적 컴포넌트(component)와 커넥터(connector)들의 집합으로 모형화한 것"으로 정의하고 있다.[7] 이 두 개의 정의는 논리관점과 개념관점이라는 서로 다른 용어를 통하여 우리가 소프트웨어시스템에 있어서의 '구조'라고 할 수 있는 가장 본질적인 모습을 잡아내려 했다는 공통점을 지니고 있다.

본 연구에서는 논리관점을 기능적인 관점에서 시스템의 구조를 보여주는 관점으로서 다른 관점보다 우선적으로 정의되어야 하는 가장 기본적이고 근본적인 소프트웨어구조관점으로 본다.[8]

따라서 논리관점에서는 시스템의 기능적인 분할에 관심을 가지며, 기능들이 물리적인 시스템에 어떻게 배치되며, 컴퓨팅자원의 할당은 어떻게 할 것이며, 개발을 목적으로 어떻게 기능요소들을 분해(factoring)하고 재사용하는가와 같은 최적화 문제, 개발임무를 개발조직에 어떻게 할당하는가 등의 문제는 고려하지 않는다. 본 연구에서는 실행관점은 논리관점에 비하여 시스템구현에 더 근접한 개념으로, 논리적구조에 대하여 어떻게 컴퓨팅자원을 할당할 것인가를 결정하는 관점으로 본다. 따라서 실행관점은 최종적인 논리구조를 상세화 한 것으로 논리관점의 정보를 모두 담고 있다고 볼 수 있다. 배치관점은 나아가 컴퓨팅자원 할당계획을 물리적 시스템에 배치하는 계획을 제시한다.

3.2 구조기술언어

구조기술을 위하여 시각적인 언어가 많이 사용된다. 그 이유는 구조의 큰 역할의 하나가 시스템전체의 개략적인 모습을 잘 보여주는 것인 한편, 시각적인 언어는 텍스트기반의 언어에 비하여 전체적인 정보를 직관적으로 한꺼번에 전달할 수 있기 때문이다. 따라서 본 연구에서도 시각적인 언어를 사용한다.

그림 3.1 상자(Box)와 선(Line)

구조를 표현하는 가장 기본적인 기호로 상자와 선이 사용되어왔다. (그림 3.1 참조) 이들은 보통 각각 컴포넌트와 커넥터로 일컬어지고, 컴퓨팅개체(computing entity)와 이들간의 연관방식(association 혹은 relationship)을 의미한다.[9] 컴퓨팅개체는 능동적인 프로세스와 같은 개체일 수도 있고 데이터베이스와 같은 수동적인 개체일 수도 있다. 컴퓨팅개체의 연관방식은 그 구현을 위하여 별도의 (컴파일대상이되는) 소프트웨어유닛을 필요로 할 수도 있고 그렇지 않은 연관방식일 수도 있다. 전자의 예로서 통신프로토콜과 같은 연관방식이 있고, 후자의 예로서 함수를 호출과 같은 연관방식이 있다.

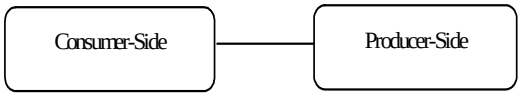

그림 3.2 시스템 구조의 예

그림 3.2는 상자와 선을 사용하여 상호작용하

(mostly) from the problem domain, in the form of objects or object classes." [1]

[7] "The conceptual architecture view is closer to the application domain because it is the least constrained by the software and the hardware platforms. In the conceptual view, you model your product as a collection of decomposable, interconnected conceptual components and connectors." [2]

[8] Rational Rose에서는 논리관점이라는 용어를 (1) 클래스, (2) 클래스 다이어그램, (3) 상호작용 다이어그램, (4) 상태차트 다이어그램, (5) 패키지를 포함하는 뜻에서 사용하고 있어서 본 연구에서 사용하는 개념과는 다름 개념으로 사용하고 있어서 주의가 요구된다.

[9] 컴퓨팅개체라고 말함으로써 이미 우리는 응용 도메인이 아니라, 소프트웨어 솔루션의 도메인에서 말하고 있다. 실제로 소프트웨어 구조는 도메인의 구조와 소프트웨어솔루션의 구조가 만나는 부분이고, 따라서 소프트웨어 구조를 얼마나 잘 정립하느냐 하는 것은 우리가 얼마나 좋은 솔루션을 궁극적으로 구축할 수 있는가를 결정한다.

는 생산자와 소비자가 어떤 방식으로 연관을 갖고 있음을 나타낸다. 그림 3.2가 보여주는 것은 시스템의 구성요소가 두 개가 있고 이들의 이름이 Consumer-Side, Producer-Side이고 서로 연관되어 있다는 극히 간단한 것만을 기술하고 있다. Consumer과 Producer의 기능이 무엇이고 그 연관의 성격이 어떤 것인지 명시되지 않은 채로 남아있다. 그림 3.3이 시스템 이름이 Consumer-Producer라는 것 이외에 아무런 정보도 주고 있지 않는데 비하여 그림 3.2는 시스템을 차하위시스템(the immediate subsystems)으로 구성된 구조체로서 보여주고 있다.

그림 3.3 하나의 상자로 표시된 시스템

그림 3.2는 모든 정보를 보여주지 않고 구조수준의 추상화된 정보만을 제공함으로써 어떤 목적을 달성하고 있다고 볼 수 있다. 만일 구조정보를 담고 있는 산출물(artifact)을 개발의 목적으로도 활용하고자 한다면, 시스템의 명세정보가 전달되어 각 하위시스템 단위의 설계와 개발이 독립적으로 이루어질 수 있도록 하여야 한다. 본 연구에서 제안하는 구조기술언어인 ALFRed는 한 단계의 구조설계의 산출물이 구조를 형성하는 하위시스템(혹은 컴포넌트)들과 그들에 대한 형식명세(formal specification)로서, 각 하위시스템과 그 명세가 다음 단계의 구조설계의 입력물이 되어 동일한 활동을 반복적으로 수행하는 재귀적 설계를 지원하는 언어이다.

그림 3.4의 시스템의 시나리오에서부터 먼저 그림 3.2의 구조를 결정하였다고 가정하자. 그 경우 그림 3.2의 구조를 반영한 하위시스템의 상호작용을 그림 3.4와 같이 나타내었다고 하자. 예를 간단히 하기 위하여 이러한 동작이 시스템에 대하여 요구되는 동작의 전부라고 하자. Producer-Side에서 M은 여러 차례 반복되어 생산될 수 있고 소비자는 소비하지만 소비하는 속도보다 더 빨리 소비자에게 전달되지 않도록 생산자와 소비자 간에 흐름제어(flow control)의 기능을 두는 것이 시스템의 하나의 제약사항이라고 하자. 이를 위하여 Producer-Side와 Consumer-Side를 각각 상세화하여 그림 3.5의 결과를 얻을 수 있다.

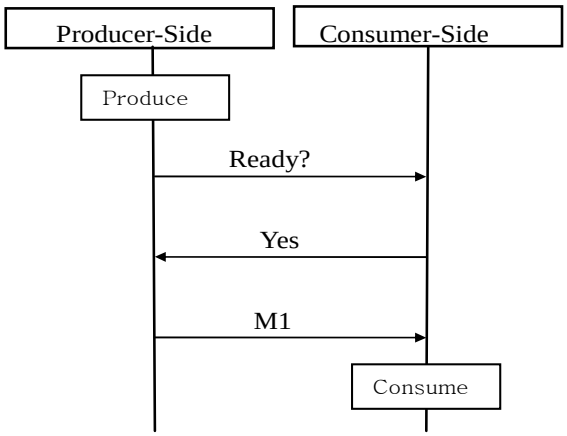

그림 3.4 두 개의 하위시스템간의 메시지순차도

어떻게 시스템을 하위 하위시스템들로 나누고 이들간에 어떠한 상호작용을 허용할지에 대한 결정에 따라 비기능적인 요구사항을 더 많이 시스템이 충족시킬 수도 있고 그렇지 않을 수도 있다. 이러한 소프트웨어구조의 결정과정은 소프트웨어 아키텍트(software architect)들의 창의적인 통찰력을 필요로 하는 과정으로 이 과정에 architecture styles, architectural patterns, metaphor, quality tactics 등의 지식이 활용될 수 있다. 뿐만 아니라 다양한 관점의 소프트웨어구조도 관점들간에 상호 영향을 주면서 다른 관점의 구조를 고려하여 정립되어야 실질적으로 최적화된 구조가 정립될 수 있다. 그러나 비기능적인 요구사항에 대한 고려는 본 연구의 범위를 벗어나는 것으로서, 본 연구에서는 순수히 기능적인 측면만을 다룬다.

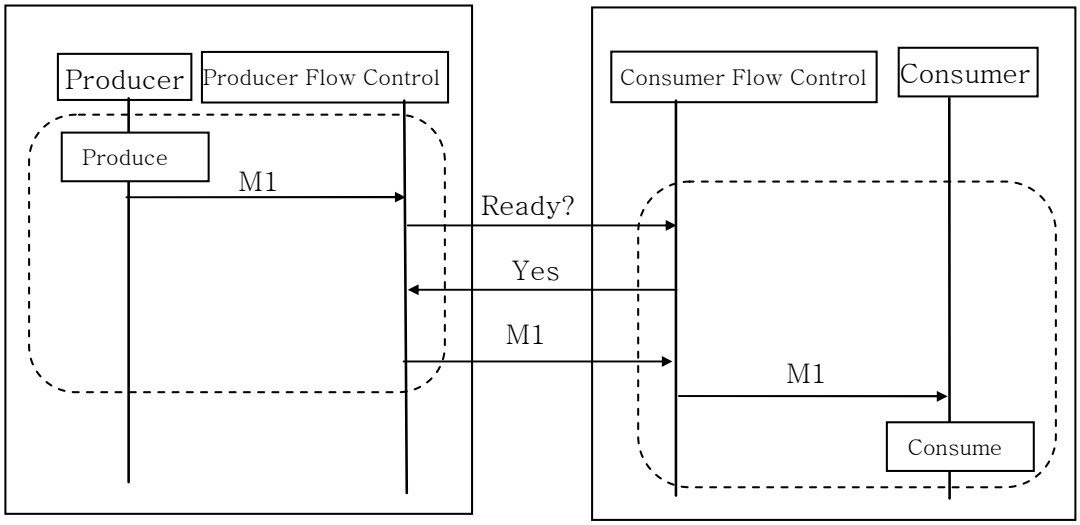

그림 3.5 그림 3.4의 각 하위시스템의 상세화

4. ALFRed 의 적용 예

이 절에서는 ALFRed의 적용 예를 보인다. 예제 시스템은 판매 물품들을 기록하고 고객으로부터의 지불을 관리하는 Point Of Sale (POS) 시스템이다. 예제를 위한 요구사항과 시나리오는 [4]의 예를 축소하였다. 다음은 축소된 POS 시스템의 요구사항이다.

(1) 고객은 원하는 물품을 직원에게 건네주면 직원은 POS시스템을 통해서 판매에 대한 처리를 할 수 있다.
(2) 고객은 물품에 이상이 있을 경우에는 반품을 할 수 있다.
(3) 직원은 POS시스템을 통해서 물품관리를 할 수 있다.

그림 4.1은 POS 시스템의 부분적인 Use Case Diagram이다. 그림 4.2는 Use Case들 중에서 Process Sale Use Case의 시나리오를 기술하고 있다. Use Case는 시스템이 무엇을 하는가에 중점을 두며, 어떻게 그것을 할 것인가는 관심을 두지 않는 사용자 관점의 시스템의 쓰임새에 대한 서술이다. Use Case 시나리오는 Use Case의 기능을 실행하기 위해 어떤 일이 발생하는가를 이벤트 흐름의 단계별로 기술한 것으로 기본흐름(Basic Flow)과 대체흐름(Alternative Flow)으로 나누어 진다.

그림 4.3은 Process Sale Use Case 시나리오를 메시지순차도로 바꾼 그림이다. 이는 시스템과 직원간의 상호작용만을 고려하기 때문에 시스템의 세일을 언제 어떻게 기록을 하는지 또 고객의 지불금을 어떻게 처리하는지에 대한 정보를 보여주지 않는다. 이러한 정보들의 처리는 POS 시스템이 어떤 하위시스템들로 나누어져 있는가에 따라 달라진다.

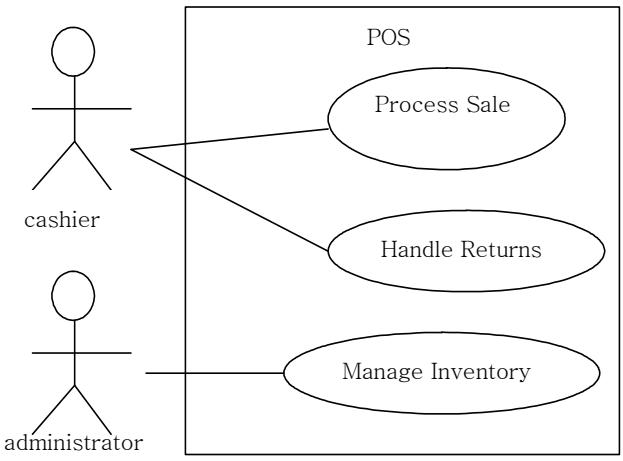

그림 4.1 POS 시스템의 Use Case Diagram

Process Sale Use Case Scenario

Precondition: Cashier is identified and authenticated.
Postcondition: Sale is successfully saved

Main Success Scenario
1. Customer arrives at POS checkout with goods.
2. Cashier starts a new sale.
3. System records sale line item and presents item description, price, and running total.
4. Cashier tells Customer the total, and asks for payment. Cashier repeats step 3-4 until indicates done.
5. Customer pays and System handles payment.
6. System logs completed sale.

Alternative Flow for 3~5
(3-5) Customer asks Cashier to remove an item from the purchase.
i. Cashier enters item identifier for removal from sale.
ii. System displays updated running total

그림 4.2 Process Sale의 Use Case 시나리오

Process Sale의 시나리오는 다음과 같다: 고객이 쇼핑 후에 직원에게 선택한 물품에 대한 계산을 요구한다. 직원은 사용자가 구입한 물품의 바코드를 스캔하여 시스템의 고유한 번호를 입력한다. 고객이 같은 물품을 여러 개 구입한 경우에는 스캔을 한 후에 구입 개수를 입력한다. 시스템은 고객에게 현재 물품의 가격과 전체 가격을 보여준다. 직원이 이와 같은 과정을 통해서 고객의 모든 물품을 시스템에 입력을 한 후에 총 금액을 고객에게 지불하기를 요청한다. 시스템은 고객의 지불을 처리한 후에 세일에 관한 모든 정보(물품들, 처리한 직원, 날짜 및 시간, 지불금액)을 기록함으로써 세일을 마친다.

POS시스템의 구축은 이와 같은 시나리오를 수행하고 요구되는 계산, 처리 및 자료의 관리를 위하여 어떻게 컴퓨팅개체에 역할을 분담할 것이가에 대한 결정에서 출발하여야 한다. 이를 위하여 먼저 시스템에 Sale Manager와 Inventory Manager를 두고 이 두개의 하위시스템에 역할을 분담하기로 한다. 이러한 결정이 반영된 시스템구조를 그림 4.4가 보여준다.[10]

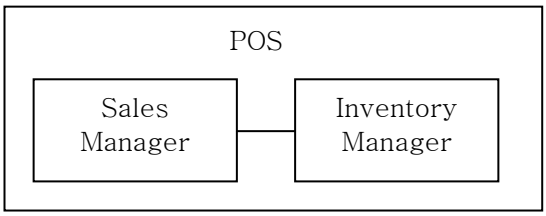

그림 4.4 POS 의 서브 시스템들

Sale Manager는 세일에 관한 정보를 받아들여 필요에 따라 Inventory Manager에게 도움을 받아 물품의 가격, 위치 등을 얻을 수 있고 세일이 끝나면 Sale Log에 저장한다. Inventory Manager는 물품에 관한 정보를 가지고 있으며 수량, 개수, 위치, 가격 등의 정보를 저장하고 관리를 하고 지불에 대한 처리를 한다. 따라서 Sale Manager는 물품가격 정보와 지불금액을 Inventory Manager 에게 요청하여 받아서, 세일정보를 기록하고 고객에게 돌려줄 거스름돈을 계산한다.

그림 4.5는 이러한 책임을 부여받은 하위시스템간의 상호작용으로 상세화된 Use Case 시나리오를 보여준다. 여기서 다시 Sale Manager와 Inventory Manager의 명세가 메시지순차도의 형태로 추출되어 다음 단계의 상세화로 이어질 수 있다.

본 절의 예에서는 Process Sale Use Case만을 고려했으나 시스템을 하위시스템으로의 분해(decompose)하여 하위시스템간의 상호작용을 도출하는 과정은 보통 복수개의 관련된 Use Case를 고려하여 이루어져야 한다. 앞에서 최적화된 구조의 도출을 위하여 비기능적인 요구사항과 다양한 구조관점을 고려한 반복적인 구조결정 과정이 필요하다는 점을 지적하였다. 마찬가지로 복수개의 관련된 Use Case를 반영한 정합적이고(coherent) 최적화된 구조도출을 위하여도 실제 사용하는 구조결정과정은 반복적인 결정과정이 되어야 한다.

[10] POS시스템개발의 고객은 시스템이 여러 직원들이 동시에 여러 고객들로부터 세일을 처리할 수 있기를 원한다. 이 경우 하나의 Inventory Manager가 다수의 Sale Manager와 상호작용함으로써 시스템이 의도된 기능을 수행하여야 한다. 그러나 논리관점에서 볼 때에 다수의 시스템 운영될 때 Sale Manager들은 동일한 행위를 보이는 개체들기 때문에, 동일한 컴포넌트의 instance로 구현할 수 있다. 따라서 Sale Manager의 복수성(multiplicity)은 논리구조에서는 표현하지 않고 실행구조에서 표현되어야 할 사항이다. 이러한 방법으로 다양한 구조관점은 한 번에 한가지 측면에 초점을 맞추어 개발을 수행할 수 있도록 한다.

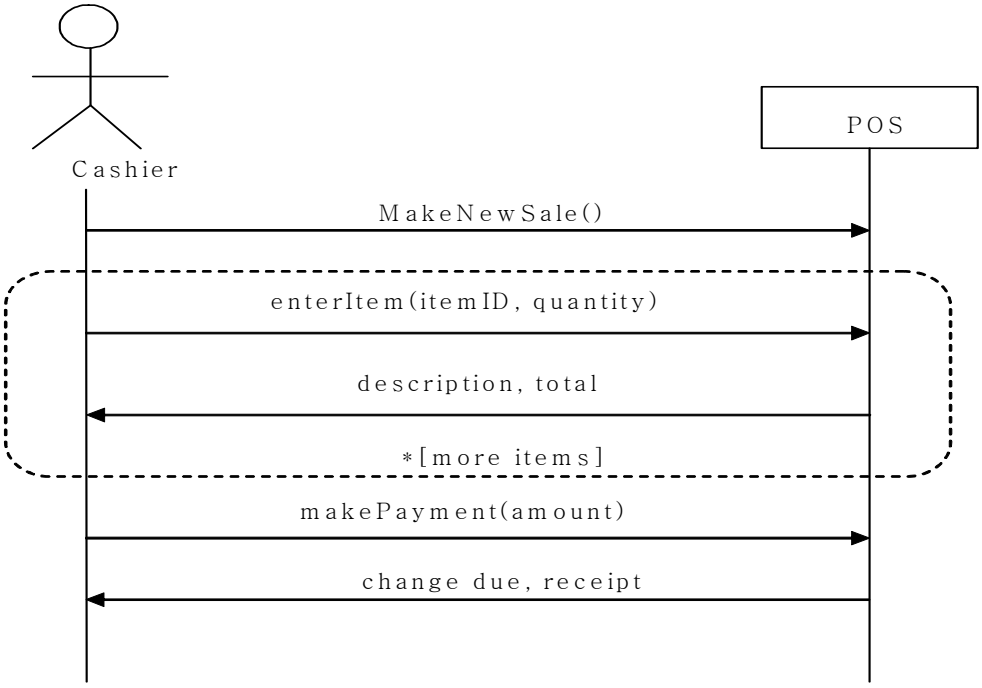

그림 4.3 시스템과 직원 간의 메시지순차도

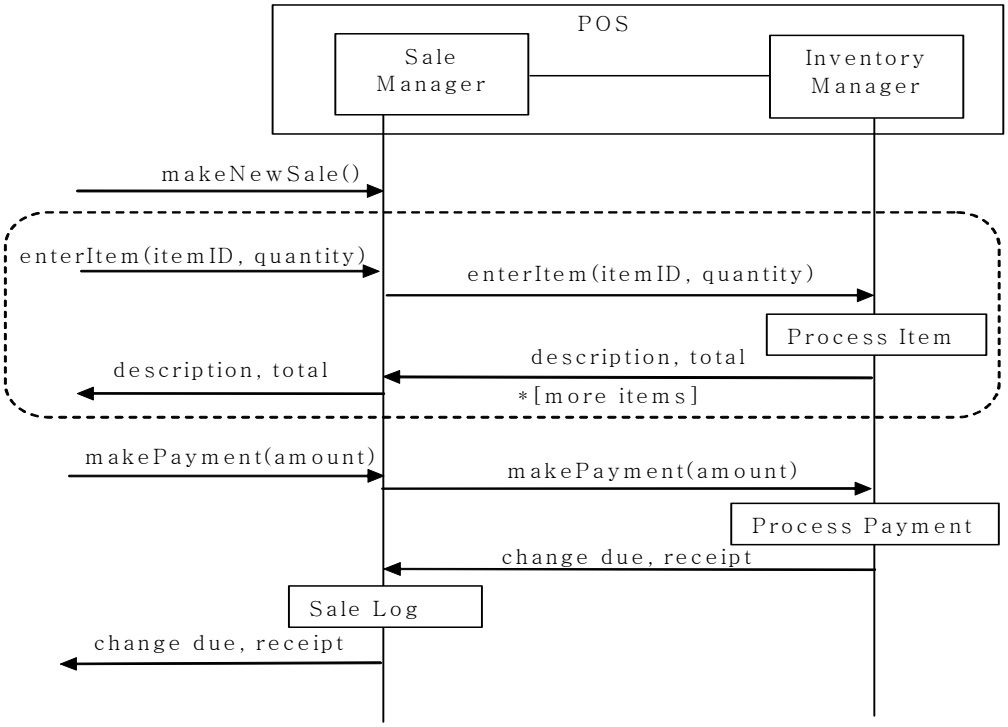

그림 4.5 두 하위시스템 간의 메시지순차도

5. 관련연구

소프트웨어 아키텍쳐를 정형화된 형식으로 표현 함으로써 설계와 분석 그리고 진화에 도움을 주고자 과거 많은 ADL 들이 만들어 졌다. [5] 대표적인 ADL 로 Wright [6,7], UniCon [8], Rapide [9], C2 [10]등이 있다. Wright 는 병렬시스템(concurrent system)의 행위에 대한 모델링과 분석에 초점을 둔 언어이고, Unicon 는 공통 인터액션 프로토콜(Common Interaction Protocol)들을 써서 이미 존재하고 있는 컴포넌트 간에 동작이 원할하도록 만드는 접착 코드를 생성하는데 초점을 두고 있다. C2 는 분산 지향적 실시간 시스템의 구조를 표현하는데 초점을 두고 있다. ACME 는 이런 ADL 들이 가지고 있는 공통점에 기반하여 서로 다른 ADL 간에 정보교환을 가능하게 하는 구조기술교환언어(Architecture Description Interchange Language)이다. [11, 12]

그러나 기존의 ADL들은 다양한 구조관점들 간의 상관관계를 고려하지 않고 만들어졌고 대체로 실행관점을 지원하기 때문에 구조설계의 가장 기본이 되는 논리관점에 대하여 적절한 지원을 사용자에게 하지 못하고 있다.

예를 들어 기존의 ADL들은 포트(port)와 역할(role)을 분명히 구분하고 있고 이를 위하여 시스템의 실행시의 상호작용의 구현정보을 포함함으로써 논리구조의 설계단계에 지나치게 많은 사항들을 한꺼번에 고려하도록 하고 있다. 대부분의 ADL들에서 컴포넌트와 커넥터의 타입들을 명시하고 인스턴스를 생성하도록 하고 물리적 구현을 위한 제약사항등을 명시하게 함으로써 논리적 뷰를 벗어나고 있다.

또한 기존의 ADL들은 계층적인 하위시스템의 분할에 대한 지원이 없거나 방법을 제시하지 못하고 있다. 상위 레벨의 컴포넌트와 커넥터를 다시 하위의 컴포넌트와 커넥터의 조합으로 상세화 과정을 지원하는 대표적인 언어로 ACME, Siemens ADL [2]를 들 수 있다. 그러나 이 언어는 컴포넌트들의 인터액션을 표현하는데 있어서 컴포넌트의 인터페이스와 커넥터를 분리하고 물리적인 고려사항도 포함하게 함으로써 재귀적인 설계를 어렵게 만든다.

본 연구에서는 –소프트웨어구조의 도출을 방법(method) 혹은 과정(process)의 적용결과로서 얻어지는 것으로 봄으로써 체계적으로 결과를 얻을 수 있는 개발단계로 만들려 노력한 반면, 과거 연구에서는 구조패턴과 분석에 의존한 것으로 보아 많은 지식과 경험을 요구하는 개발단계로 머물러 있었다.~~전체 구조에서 하위구조로의 상세화를 자연스럽게 연결해주지 못하고 전체 시스템의 상호작용의 문맥에서 하부시스템으로 상세화 되기 보다는 상위의 컴퍼넌트가 인터페이스(port)가 분배되는 서브 컴퍼넌트중에 한 컴퍼넌트의 인터페이스에 1:1 로 맵핑(binding)이 됨으로써 서브 컴퍼넌트를 상위 컴퍼넌트의 인터페이스의 구현의 개념으로 보게 된다. 또한 한 컴퍼넌트가 가질 수 있는 여러 인터페이스들간의 논리적 관계를 전혀 보여주고 있지 않기 때문에 시스템적으로 서브 컴퍼넌트의 분배가 어렵다.대부분 부분적인 컴포넌트들의 인터액션들의 집합으로서 전체 구조를 보여주게 된다.~~

6. 결론

기존의 ~~본 연구에서는 이와 같은 기존의~~ ADL~~은~~ ~~의~~ 뷰에 대한 명확한 개념이 없이 실행뷰에 바탕을 둔 언어를 제공함으로써 요구사항으로부터 적절한 아키텍처를 도출하기 위해 필수적인 논리적인 뷰를 제공하지 못하고 있다. 또한 ADL의 표기법만을 제시함으로써 요구사항과 소프트웨어구조도출을 연결짓는 방법은 제시하지 않고 있다.

본 연구에서는 이와 같은 기존의 ~~이에 기존~~ ADL의 한계를 넘어서 본질적인 구조개념에 해당하는 논리관점의 구조기술을 위한 언어를 제시하였다. 이를 위하여 먼저 논리관점의 정의를 내리고, 논리관점을 기술하는 ADL이 지녀야 하는 바람직한 특징들을 제시한 뒤 논리적 관점을 표현하는 ADL인 ALFReD를 제시하였다. ALFReD는 메시지순차도의 형

식언어로 작성된 기능요구사항으로부터 재귀적설계 공정의 각 단계를 통일성있게(uniform) 표현하도록 하는 특징이 있다. --그리고 ALFReD를 예에 적용하여 효과적으로 사용될 수 있는 ADL임을 보였다.

본 연구에서 하위시스템은 넓은 의미의 컴포넌트와 같은 개념이고 따라서 하위시스템도 엄밀히 정의된 인터페이스를 가져야 한다. 본 연구에서는 기능적 요구사항이 하위시스템에 전달됨으로써 재귀적설계가 이루어지는 과정만을 보였다. 향후연구에서는 하위시스템의 인터페이스를 엄밀히 정의하는 체계를 제시하고, 나아가 본 연구의 체계 위에 비기능적 요구사항을 구조정립에 어떻게 반영할 수 있는가에 대하여 연구를 수행하고자 한다.

참고문헌

[1] Kruchten, P., "Architectural Blueprints - the '4+1' View Model of Software Architecture," *IEEE Software 12(6)*, November, 1995.

[2] Hofmeister, C., Nord, R., and D. Soni, *Applied Software Architecture*, Addison Wesley, 1999.

[3] ITU-T Recommendation *Z.120: Message Sequence Chart (MSC)*, 1999.

[4] Larman, G., *Applying UML and Patter*n, 2nd Ed., Prentice Hall, 2002.

[5] Clements, P. C, "A Survey of Architecture Description Languages," *Eighth International Workshop on Software Specification and Design*, Germany, 1996.

[6] Garlan, D, and Allen, R, "Formalizing Architectural Connection," *Proceedings of the 16th International conference on Software Engineering*, Sorrento, Italy, 1994.

[7] Allen, R., and Garlan, D., "A Formal Basis For Architectural connection," *ACM Transactions on Software Engineering and Methodology*, 6(3), July 1997.

[8] Shaw, M., et al, "Abstractions for Software Architecture and Tools to Support Them," *IEEE Transactions on Software Engineering 21(6*), 1995.

[9] Luckham, D. C., et al, "Specification and Analysis of System Architecture Using Rapide," *SIGSOFT Software Engineering Notes 19(5)*, 1994.

[10] Medvidovic, N., Oreizy, P., Robbins, J. E., and Taylor, R. N., "Using object-oriented typing to support architectural design in the C2 style," In SIGSOFT'96: Proceeding of the Fourth ACM Symposium on the Foundations of Software Engineering. ACM Press, October 1996.

[11] Garlan, D., Monroe, R., and Wile, D., "ACME: An Architectural Description Language," *Proceedings of CASCON 97*, Toronto, Ontario, November 1997.

[12] David Garlan and Zhenyu Wang, "A Case Study in Software Description Interchange Language", Submitted for Publication, 1998