DOI: 10.3745/KIPSTD.2008.15-D.5.647

자연어를 이용한 요구사항 모델의 번역 기법

오 정 섭[†] · 이 혜 련[†] · 임 강 빈[††] · 최 경 희[†††] · 정 기 현[††††]

요 약

자연어로 작성된 고객의 요구사항은 개발과정에서 모델링 언어로 재작성 된다. 그러나 개발에 참여하는 다양한 계층의 사람들은 모델링 언어로 작성된 요구사항을 이해하지 못하는 경우가 많이 발생한다. 본 논문에서는 REED(REquirement EDitor)로 작성된 요구사항 모델을 자연어로 번역하여 개발에 참여하는 모든 계층의 사람들이 요구사항 모델을 이해할 수 있도록 도와주는 방안을 제시한다. 제시한 방법은 3단계로 구성되어 있다. 1단계 IORT(Input-Output Relation Tree) 생성, 2단계 RTT(Requirement Translation Tree) 생성, 3단계 자연어로 번역의 단계를 거친다.

키워드 : 요구사항, 번역, 모델, 자연어, REED

Translation Technique of Requirement Model using Natural Language

Jungsup Oh[†] · Hyeryun Lee[†] · Kangbin Yim[††] · Kyunghee Choi[†††] · Kihyun Jung[††††]

ABSTRACT

Customers' requirements written in a natural language are rewritten to modeling language in development phases. In many cases, those who participate in development cannot understand requirements written in modeling language. This paper proposes the translation technique from the requirement model which is written by REED(REquirement EDitor) tool into a natural language in order to help for the customer understanding requirement model. This technique consists of three phases: 1st phase is generating the IORT(Input-Output Relation Tree), 2nd phase is generating the RTT(Requirement Translation Tree), 3rd phase is translating into a natural language.

Keywords : Requirement, Translation, Model, Natural Language, REED

1. 서 론

소프트웨어 개발과정의 한 단계에서 작성된 문서는 다른 단계에서 직접적 혹은 간접적으로 다양한 형태로 번역 된다. 예를 들면, 분석가가 영어로 작성한 시스템 요구사항은 소프트웨어 설계가에 의하여 상태 다이어그램 등으로 표현된 소프트웨어 모델로 번역 된다. 역으로, 분석가는 소프트웨어 모델이 원하는 기능을 수행하는 지를 검사하기 위하여 자신이 이해하는 언어로 번역한다. 이외에도, 소프트웨어 모델은 프로그래머에 의하여 혹은 자동 코드 생성기에 의하여 코드로 번역 된다.

정확한 표현이 어려운 자연어로 기술된 요구사항의 경우, 모델로 번역하면서 많은 오류가 발생한다. 정확한 번역이 어렵다는 점은 요구사항의 검증을 어렵게 만들고, 부정확한 요구사항은 프로젝트 일정이나 소프트웨어 품질에 매우 큰 부정적 영향을 준다. James Martin은 소프트웨어 프로젝트에서 발생되는 결함들 중에서 56%는 요구사항 결함이며, 이 결함들 중 약 50%가 엉망으로 쓰여졌거나 모호하거나 명료하지 않거나 부정확한 요구사항 때문이라고 보고하고 있다[1]. 요구사항 결함은 재작업을 발생시킨다. 프로젝트 개발비의 약 30~50%가 재작업 비용이며[2], 이 재작업 비용 중 70~85%가 요구사항의 오류에서 기인한다[3].

요구사항 번역 오류를 줄이기 위하여 UML[4], Simulink[5], SDL[6] 등으로 요구사항을 기술하려는 노력이 시도되고 있다.

UML 2.0은 Class 다이어그램이나 Statemachine 다이어그램 등 13가지 다이어그램을 통해서 시스템을 모델링 하는 언어이다. Class 다이어그램 등을 통해서 시스템을 정적인 관점에서 모델링을 할 수 있고, Statemachine 다이어그램 등을 통해서 시스템을 행위적인 관점에서 모델링을 할 수 있다. 정적인 관점과 행위적인 관점에서 모델링을 수행할 경우, 시스템의 구조뿐 아니라 시스템의 동작에 대한 명확한 모델링이 가능해 진다. 또한 UML은 많은 사용자 층을 확보하고 있고, IBM의 Rational Rose, RequisitePro, Borland의 Together, Telelogic의 Rhapsody 등 많은 도구에서 지원한

† 준 회 원 : 아주대학교 일반대학원 컴퓨터공학과 박사과정
†† 종신회원 : 순천향대학교 정보보호학과 교수(교신저자)
††† 정 회 원 : 아주대학교 정보통신전문대학원 교수
†††† 정 회 원 : 아주대학교 전자공학부 교수
논문접수 : 2008년 4월 28일
수 정 일 : 1차 2008년 6월 20일, 2차 2008년 7월 24일
심사완료 : 2008년 8월 22일

다. 이 도구들 중의 일부는 UML로부터 프로그래머가 작성해야 할 코드를 자동으로 생성하는 기능까지도 제공한다.

Mathworks의 Simulink는 동적인 시스템을 모델링, 시뮬레이션, 분석하기 위한 소프트웨어이다. Simulink는 시스템을 설계하는데 필요한 여러 가지 컴포넌트를 제공한다. 예를 들면, AND/OR 게이트에서부터 신호 생성기, 수학 연산자에 이르기까지 다양한 컴포넌트를 제공한다. 이들 컴포넌트를 이용하여 시스템을 쉽게 모델링 할 수 있다. 시뮬레이션을 수행하여 시스템의 동작을 미리 예측해 볼 수도 있고, 설계된 시스템의 오류를 미리 찾아낼 수도 있다. 또한 다양한 부가 기능을 통하여 요구사항 관리기능, 검증 및 확인 기능도 가능하다.

SDL은 이벤트 기반 시스템, 특히 원격통신 시스템의 명세서를 기술하기 위한 언어이다. SDL은 복잡하지 않으며 상대적으로 적은 하위시스템을 가지고 있는 행위 모델링에 적합하다. Time sequence 다이어그램은 시스템의 행위를 사용자의 관점에서 명세서로 표현하는데 특별히 적합하다. SDL은 실시간 시스템의 행위를 모델링 하기에 편리한 “SDL machine”이라고 불리는 동적인 의미를 지원한다[13].

이들 요구사항 모델링/기술 언어들은 모두 공통적인 특징을 가지고 있다. 가장 주목되는 특징은 이해하기 쉬운 그래픽적인 표기법을 사용한다는 점과 사용된 그래픽적인 구성요소들의 의미가 분명하다는 점이다. 특히, 이들 방법을 사용하여 요구사항을 작성하면, 자동 코드 생성기 등 번역의 자동화가 가능해져 번역 과정에서 발생하는 오류를 줄일 수 있다.

그러나 자연어를 사용하는 분석가가 Statechart나 Stateflow로 기술된 요구사항을 자연어로 정확하게 번역하여 이해하는 데는 많은 어려움이 있다. 이는 Statechart에 관한 지식을 가지고 있을 때에야 비로서 Statechart로 기술된 요구사항을 정확하게 이해할 수 있으며, Stateflow 실행에 대한 의미를 정확하게 이해하고 있을 때 Stateflow로 작성된 요구사항을 정확하게 번역할 수 있기 때문이다. 자연어만을 사용하는 분석가나 고객은 모델링된 요구사항을 이해하기 위해서 Stateflow의 개념을 습득하려고 하지 않기 때문에 자연어가 아닌 요구사항을 이해하기 힘들다.

분석가와 설계가 등 개발과정에 참여하는 다양한 계층의 사람들에게 동일한 언어의 사용을 강제하는 것보다는 사람들 사이의 간격을 메워주는 번역 기법이나 도구를 제공하는 것이 효과적이다. 예를 들면, 자연어로 기술된 요구사항으로부터 핵심적인 요구사항을 추출하는 기법[7, 8, 10] 등은 정확한 요구사항 작성에 많은 도움을 준다. 자동 코드 생성기는 모델을 C/C++/Java 코드로 번역하여 줌으로써 설계가와 프로그래머 사이의 간격을 메워 준다[9]. 이처럼 하위 방향으로의 번역뿐 아니라 상위 방향으로의 번역들 사이의 간격을 위한 도구도 많이 연구되고 있다. 예를 들면, C/C++ 코드로부터 UML 다이어그램들을 생성하기도 한다[12]. 프로그램 코드로부터 UML 객체 다이어그램 등을 자동으로 생성해 주는 기능 등은 이미 MS visual studio를 비롯한 많은 도구에 적용되었다. 그러나 아직 모델링 언어로 기술된 요구사항을 자연어로 번역 하는 상위 방향으로의 번역에 대한 연구는 아직 이루어지지 않고 있다.

본 연구에서는 그래픽적인 표기법으로 표현된 요구사항을 자연어 요구사항으로 번역하는 기법에 대하여 기술한다. 기술된 기법은 요구사항 관리 시스템인 REED(REquirement EDitor)에 적용되었다. REED는 본 연구진이 개발한 요구사항을 관리하도록 지원하는 요구사항 관리 도구이다. REED는 그래픽적인 방법으로 요구사항을 기술한다. 이를 위하여 REED는 유일한 의미를 가진 그래픽적인 요구사항 기술을 위한 그래픽적인 객체들을 정의하고 있다[16].

분석가 혹은 설계가는 REED를 사용하여 모호하지 않고 분명하며, 유일한 의미를 가진 그래픽적인 객체의 연결구조로 요구사항을 기술할 수 있다. 하나의 요구사항은 그래픽적인 객체들의 관계를 나타내는 그래프로 표현된다. 유일한 의미로 인하여 요구사항의 정확성이나 행위를 분석하는 것이 가능해 진다. REED가 비교적 형상화한 언어로 요구사항을 기술하도록 지원한다고 하더라도, REED의 객체들의 의미나 이들의 관계에 대한 정확한 의미를 모르는 분석가는 요구사항을 번역하여 정확히 이해하는 데 어려움을 겪게 된다. 그리고, 이는 결국 모델링된 요구사항이 분석가의 의도를 정확하게 표현하였는지를 이해하는 데 장애를 준다. 따라서 그래픽적인 요구사항으로부터 자연어로의 상위 방향으로 번역된 요구사항은 설계가와 분석가 사이의 간격 제거에 많은 기여를 할 것으로 판단된다.

본 논문의 구성은 아래와 같다. 2절에서는 관련연구를 소개하고 3절에서는 REED의 기능 및 요구사항을 그래픽하게 표현하는 것에 대하여 간략히 소개한다. 4절에서는 그래픽적인 표기법으로 기술된 요구사항을 자연어로 번역하는 알고리즘에 대하여 기술한다. 5절에서는 간단한 예제를 통하여 알고리즘을 평가하고자 한다. 끝으로 6절에서 결론을 기술한다.

2. 관련 연구

소프트웨어 개발과정에서 이루어지는 번역은 편의상 하위 방향으로의 번역과 상위 방향으로의 번역으로 분류할 수 있다. 하위 방향의 번역이란 자연어와 같은 상위 레벨의 언어로 작성된 것을 C언어와 같은 하위 레벨의 언어로 번역하는 것을 의미한다. 반대로 상위 방향으로의 번역은 하위 레벨의 언어를 상위 레벨의 언어로 번역하는 것을 의미한다.

2.1 하위 방향으로의 번역

Liu는 RUP(Rational Unified Process)에 기반하여 자연어로 작성된 요구사항을 분석하여 Class 모델로 자동 생성하는 방법을 제시하였다. RUP에 기반을 두고 있기 때문에 우선적으로 요구사항을 분석하여 Actor와 Use-case를 식별하고 미리 정의한 몇 가지 규칙을 적용하여 Class 모델로 자동 생성하였다[7].

Ilieva는 자연어로 작성된 요구사항을 객체지향 분석 모델로 바꾸는 자연어 처리 방법을 제시하였다[8]. 이 방법은 자

연어를 Class 모델과 같은 객체지향 모델로 바꾼다는 점에서, 앞에서 설명한 Liu의 방법과 매우 유사하다.

Lee는 자연어로 작성된 요구사항 문서를 정형화된 명세 언어로 자동으로 바꾸는 방법을 제시하였다. CNLP(Contextual Natural Language Processing)과 자연어와 명세언어의 차이를 극복하기 위한 TLG(Two-Level Grammar)를 사용하여 VDM (Vienna Development Method)으로 작성된 명세를 생성한다[10].

Niaz는 UML Class 다이어그램과 Statechart 다이어그램을 이용하여 자동으로 코드를 생성한다. Niaz는 JCode 시스템을 개발하였고, 이는 자바 코드를 생성한다[9]. 이는 Rhapsody나 OCode가 자동으로 생성하는 코드와 유사한 결과이다.

2.2 상위 방향으로의 번역

상위 방향으로의 번역은 일반적으로 역공학(Reverse Engineering)이라고 부른다.

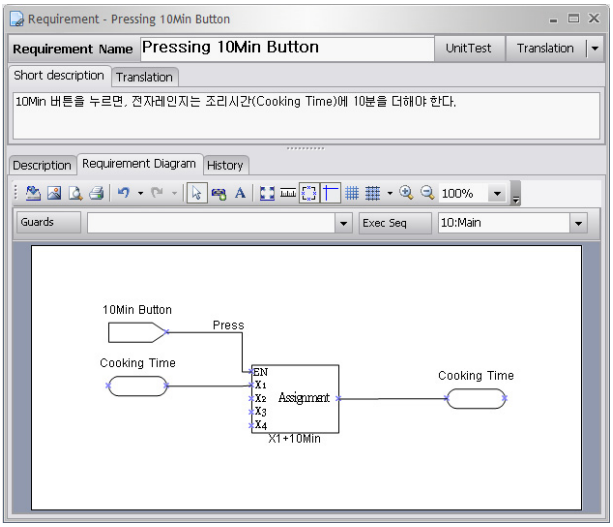
Yu는 일반적인 소스 코드로부터 statechart 다이어그램을 생성하는 방법을 제시하였다. 이를 위하여 몇 단계를 거치었는데, 1) 주석에 기반하여 소스 코드를 재구성하고, 2) 재구성된 코드를 추상화된 구조적 프로그램으로 변환하고, 3) 구조적 프로그램의 추상화된 문법 트리(abstract syntax tree)로부터 목표 모델을 추출하고, 4) 코드와 목표 모델의 추적성에 기반하여 비 기능적인 요구사항을 식별한다[11].

Korshunova는 C++ 소스 코드로부터 XMI(XML Metadata Interchange) 형식의 표현으로 변환하는 방법을 제시하였다[12]. XMI를 이용하여 UML class, sequence, activity diagram을 표현하는 방법이므로, 결과적으로는 C++ 소스로부터 UML model을 생성해내는 방법이다.

3. 요구사항의 그래픽적인 표현

요구사항을 자연어로 표현할 때 발생하는 문제점을 해결하기 위한 다양한 방법이 있다. 특히 그래픽적인 표현은 기술하기 쉽다는 점과 요구사항 모델의 의미가 분명하다는 장점 때문에 널리 사용되고 있다[4, 5, 6]. 또한, 시뮬레이션 등의 방법을 통하여 요구사항 모델로부터 시스템의 행위를 예측할 수 있다는 장점 때문에 널리 적용되고 있다[14, 15].

우리가 개발한 REED는 앞에서 설명한 도구들처럼 그래픽적인 표현을 사용하여 요구사항을 기술하도록 지원하는 도구이다. UML 등에서 사용되는 Statechart는 시스템의 행위를



(그림 1) REED에서의 요구사항에 대한 그래픽적인 표현

표현하기 위해서 시스템의 상태에 기반을 둔 Statemachine 다이어그램을 사용하지만, REED는 시스템의 행위를 표현하기 위해서 입/출력에 관심을 가지고 요구사항을 기술한다. 임베디드 시스템은 사용자의 입력에 시스템이 반응하여 원하는 출력을 생산한다는 점에 기반을 두고 있기 때문에 요구사항을 입/출력에 관심을 가지고 작성할 수 있도록 지원한다.

(그림 1)은 전자레인지의 요구사항 중 하나를 REED로 작성하는 화면의 모습이다. 화면 위의 *'Requirement Name'* 에 기술된 'Pressing 10Min Button'은 이 요구사항의 이름이다. *'10Min 버튼을 누르면, 전자레인지는 조리시간(Cooking Time)에 10분을 더해야 한다.'*는 자연어로 표현된 요구사항이다. 화면 아래에 보이는 다이어그램은 요구사항을 REED의 객체를 이용하여 표현한 요구사항이다. 이러한 다이어그램을 요구사항 다이어그램이라 부른다.

REED는 요구사항의 작성을 용이하게 하기 위하여 다양한 객체를 정의하고 있다. (그림 1)의 요구사항 다이어그램에 사용된 객체들의 의미는 다음 <표 1>과 같다.

REED에서 객체는 엔티티 객체와 연산 객체로 구별된다. 입력장치, 출력장치, 메모리 등의 객체를 엔티티 객체로 분류하고, Assignment와 같이 어떠한 기능을 수행하는 객체는 연산 객체로 분류한다. 각 객체는 서로의 연결을 위하여 포트를 가지고 있다. 엔티티 객체와 연산 객체는 연산 객체의 포트를 통하여 연결된다. 연산 객체의 출력은 다른 연산 객

〈표 1〉 REED의 그래픽적인 객체의 예

기호	이름	의미
<이름>	입력장치	시스템의 입력을 표현한다. 예를 들면, 버튼, 센서 값 등이 이에 속한다. 입력장치의 이름은 <이름>이다.
<이름>	메모리	요구사항의 기술을 용이하게 하기 위해서 사용되는 변수이다. 변수의 이름은 <이름>이다.
EN, X1, X2, X3, X4, Assignment	Assignment	EN 포트로 어떤 값이 입력될 때, 입력 포트 X_1~X_4으로 입력된 값을 사용하여 함수를 실행한 후, 그 값을 출력 포트로 출력한다.

체의 입력 포트와 연결이 가능하다. 요구사항 다이어그램에서 연산 객체들은 엔티티 객체로부터 입력을 받아 다른 엔티티 객체 혹은 연산 객체로 값을 출력하는 그래프의 형태를 가진다. (그림 1)에서 보는 바와 같이, 요구사항 다이어그램은 자연어로 기술된 요구사항이 쉽게 연상될 정도로 매우 직관적으로 표현되었다.

4. 요구사항 번역 알고리즘

4.1 번역 알고리즘 개요

요구사항 다이어그램은 자연어로 작성된 요구사항을 기반으로 생성한다. 그러나 요구사항 다이어그램과 자연어 요구사항이 100% 일치하지는 않는다. 자연어는 그 의미가 분명하지 않기 때문에 의미를 분명하게 하기 위한 추가 작업이 필요하기 때문이다. 이에 요구사항 다이어그램을 생성하는 사람은 추가적으로 파악한 내용을 다이어그램에 표현하고 이에 의해서 원래의 요구사항과 다른 다이어그램이 생성될 수 있다. 따라서 생성된 요구사항 다이어그램과 자연어로 작성된 요구사항이 일치하는지를 판단하는 일은 중요하다.

다이어그램이 요구사항의 의미를 직관적으로 표현하고 있더라도 그 의미를 정확하게 이해하기 위하여는 객체의 의미들을 이해하여야 한다. 따라서 자연어로 번역된 요구사항은 다이어그램에서 사용하는 객체의 의미가 익숙하지 않은 분석가로 하여금 요구사항 다이어그램을 이해하는 데 많은 기여를 하게 된다.

다이어그램은 3 단계를 거쳐서 자연어로 번역 된다. (그림 2)는 번역 알고리즘을 C# 형태의 가상코드로 표현한 것이다.

1단계에서는 요구사항에서 사용된 최종 출력들, 각각의 출력을 위한 입출력관계트리(IORT : Input-Output Relation Tree)로 분리한다. (그림 1)에서 보는 바와 같이, 요구사항 다이어그램은 왼쪽에 입력들이 위치하고, 오른쪽에 출력이 위치한 수평적인 모양의 트리로 표현된다. IORT에서 자식 노드들과 부모 노드는 입력-출력 관계가 성립한다. 여러 개의 출력을 생산하는 다이어그램은 각 출력을 루트로 갖는 여러 개의 IORT로 분할된다. IORT로 나누는 목적은 하나의 출력을 내보내기 위한 일련의 과정들이 결국에는 요구사항의 최소 단위이기 때문이다. 또한 요구사항을 여러 개의 요구사항으로 나누어 번역 하면 더욱 명확하게 그 뜻을 파악 할 수 있을 뿐 아니라, 테스트 할 때에도 요구사항에서 체크할 출력 값들이 중복되지 않아 오류를 명확하게 판단하는 데 기여한다. IORT로 분할하는 알고리즘은 4.2절에서 자세히 기술한다.

2 단계에서는 IORT를 자연어로 번역 하기에 적합한 트리로 변환된다. 2 단계에서 생성되는 트리를 요구사항번역트리(RTT:Requirement Translation Tree)라 부른다. 2 단계에서는 IORT를 자연어의 특성을 고려하여 번역에 적합한 RTT로 변환한다. IORT는 자연어로 번역되었을 때 사람이 이해하기 쉬운 순서대로 정렬된 구조가 아니기 때문에 RTT로 변환한다. IORT를 RTT로 변환하는 알고리즘은 4.3절에서 자세히 기술한다.

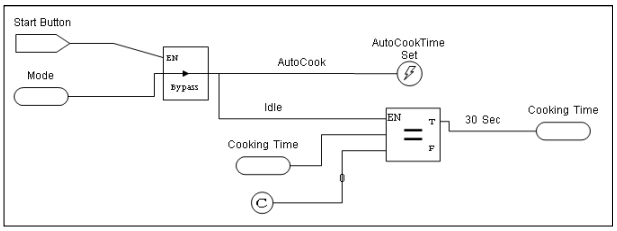
3 단계에서는 RTT의 루트 노드부터 RTT의 노드들을 차례대로 전위순회로 방문하면서 자연어로 변환한다. 이미 RTT는 번역하기에 좋은 구조로 되어 있으므로 이를 순서대로 따라가면서 각각의 객체를 자연어로 번역한다. RTT를 자연어로 표현하는 알고리즘은 4.4절에서 자세히 기술한다.

4.2 IORT(Input Output Relation Tree)의 생성

하나의 요구사항은 여러 개의 엔티티 객체와 연산 객체가 서로 연결된 다이어그램의 모양을 가지고 있다. 전자레인지의 'Start Button'이 눌려 졌을 때의 요구사항을 하나의 다이어그램으로 기술한 다음의 (그림 3)을 예로 들어 보자.

이 요구사항 다이어그램은 "사용자가 'Start Button'을 눌렀을 때, 전자레인지의 'Mode'가 'AutoCook'으로 설정되어 있으면 조리 시간을 자동 조리에 적합하도록 설정하고, 'Mode'가 'Idle'로 설정되어 있을 경우, 만약 설정된 'Cooking Time'이 0이라면 'Cooking Time'을 30sec로 설정한다."라는 요구사항을 기술한 것이다.

(그림 3)의 요구사항 다이어그램은 'AutoCookTimeSet'과 'Cooking Time' 라는 두 개의 출력을 내보낸다. 'AutoCook



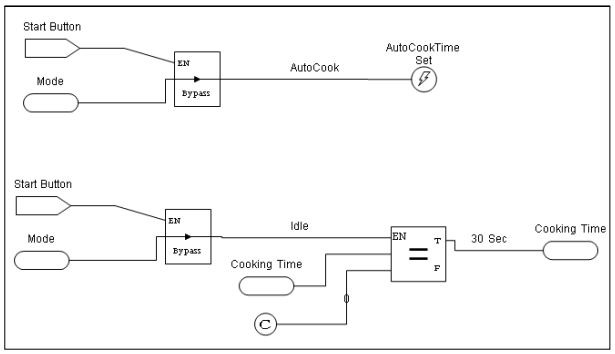
(그림 3) 전형적인 요구사항 다이어그램 (전자레인지 예제)

```
// 번역 알고리즘

Let ReqDiag be the requirement diagram to be translated

GS = GenIORT (ReqDiag);                                    // 1 단계
foreach (IORT in GS) {
    RTT = Transform_IORT_in_ReqTransTree( IORT );          // 2 단계
    Translate_Into_NaturalLanguage(RTT, Description);      // 3 단계
}
```

(그림 2) 요구사항 번역 알고리즘 가상코드

〈표 2〉 (그림 3)에 사용된 REED 객체의 의미

기호	이름	의미
	Bypass	EN 포트가 활성화 되었을 때, 입력으로 들어온 값을 출력으로 내보낸다.
	Equal	EN 포트가 활성화 되었을 때, 입력으로 들어온 2개의 값이 같으면 출력 T를 활성화 하고, 입력 값이 서로 다르면 출력 F를 활성화 한다.
	Internal Event	<name>이라는 이벤트를 발생 시킨다.
	Constant	연결된 링크에 적힌 상수 값을 의미한다.



(그림 4) (그림 3)의 요구사항 다이어그램으로부터 생성된 2개의 IORT

TimeSet'을 출력하기 위하여 'Cooking Time' 앞에 있는 'Equal' 객체는 필요가 없다. 그러나 'Bypass'라는 객체는 'AutoCook TimeSet'과 'CookingTime'을 출력하기 위하여 모두에게 필요하다. 하나의 다이어그램을 출력 별로 그 출력에 영향을 미치는 객체들로만 연결된 다이어그램으로 분리할 수 있다. (그림 3)에서 출력 별로 관계되는 객체들만을 추출하면 (그림 4)와 같은 2개의 IORT를 만들 수 있다.

(그림 4)의 요구사항 IORT는 위에서 예시된 하나의 요구사항 문장을 다음과 같은 두 개의 문장으로 표현한 것으로 간주할 수도 있다.

1) 사용자가 'Start Button'을 눌렀을 때, 'mode'가 'AutoCook'으로 설정되어 있으면 자동 cooking으로 작동하도록 cooking 시간을 설정한다
2) 사용자가 'Start Button'을 눌렀을 때, 'Mode'가 'idle'로 설정되어 있을 경우, 만약 설정된 'Cooking Time'이 0이라면 'Cooking Time'을 30sec로 설정한다.

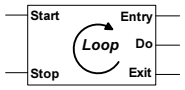
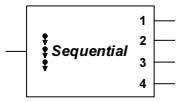
요구사항 다이어그램은 <표 3>에 나타난 FC(Flow of Control) 객체를 사용할 수도 있다. 요구사항 다이어그램이 FC 객체를 사용하는 경우, IORT를 추출하는 작업은 다소 복잡하다. 예를 들면, 두 개의 출력 A와 B가 'Sequential' 객체의 출력 포트에 연결되어 있을 경우, 출력 A를 생산하는 요구사항과 출력 B를 생산하는 요구사항을 독립적으로 번역되도록 별개의 IORT를 만드는 것 보다는 두 개의 출력 A와 B가 순차적으로 생산된다고 번역될 수 있도록 하나의 IORT에 속하도록 추출하는 것이 타당하다.

또한 IORT의 수를 줄이기 위하여 하나의 연산 객체가 여러 개의 최종적인 출력을 생산하는 경우에도 독립적인 여러 개의 IORT를 만드는 것보다는 하나의 IORT로 묶는다.

하나의 요구사항 다이어그램을 IORT들의 리스트로 만드는 알고리즘의 개략적 흐름은 다음의 (그림 5)와 같다.

우선, 다이어그램의 최종 출력 리스트를 만든 후, 이 리스트내의 각각 객체 "L" 에서 출발하여 거꾸로 탐색 하면서 "L"에 사용되는 값을 출력하는 객체들을 찾는다. 거꾸로 탐색하는 도중, 'Loop'나 'Sequential' 등의 FC 객체를 만나면, 이들을 사용하여 이미 어떤 IORT가 만들어 졌는지를 확인하기 위하여 "SetG"라는 IORT 집합을 검색한다. 이미 만들어진 IORT 중에서 동일한 FC 객체를 가진 IORT를 찾으면, 현재 작업 중이던 IORT에 병합시켜 하나의 IORT로 구축한 후, 작업을 계속한다. 이러한 작업을 다이어그램 내의 모든 객체들이 적어도 하나의 IORT에서 포함될 때까지 계속된다.

〈표 3〉 FC(Flow of control)를 표현하기 위한 객체

기호	이름	의미
	Loop	'Loop'는 동일하게 반복되는 작업을 표현할 때 사용하는 객체이다. 'Start' 포트로 입력되면, Entry에 연결된 작업을 수행한 후, 'Do'에 연결된 작업을 'Stop'으로 입력이 발생할 때까지 반복한다는 것을 의미한다. 'Stop'에 입력이 발생하면, 'Exit'에 연결된 작업을 수행하고, 'Loop'는 종료된다.
	Sequential	'Sequential'은 일련의 행동을 차례대로 기술할 때 사용된다. 각 출력 포트에 연결된 작업을 번호순으로 차례대로 수행한다.

```
// IORT 생성
GraphSet GenIORT (Graph ReqDiag)
{
    Let SetG be an empty set
    ListofLastOP = List of last operation objects of ReqDiag

    foreach (L in ListofLastOP) {
        1)mark all objects ∈ ReqDiag as unvisited
        2)Create an IORT G, and initially L∈ G, mark L as visited
        3)while(there is an unvisited object Obj that is researchable by backward traverse from an
           object already visited) {
           (1)mark Obj as visited
           (2)if(Obj is 'Sequential' or 'Loop' object and Obj is already used by other IORT F in
                  SetG) {
                  Set all objects ∈ F as visited
                  Merge F into G
                  remove F from SetG
              }
              else {
                 Expand G using Obj
              }
        4)Insert G into SetG
     }
    return SetG
}
```

(그림 5) IORT를 생성하기 위한 알고리즘

```
// IORT로부터 Intermediate Tree 생성
IntermediateTree Convert_IORT2IntT (IORT G)
{
    Let IMT as a Empty Intermediate Tree
    FC = Find 'Loop' or 'Sequential' object in G
    if FC is not found {
         Initialize IMT with G
    }
    else {
        1)Set FC as a root node of IMT
        2)Make G_in, connected to each input port of FC, as a child of FC
        3)Make G_out, connected to each output port of FC, as a child of FC
        4)Call Convert_IORT2IntT(G_out)
    }
    return IMT
  }
```

(그림 6) IMT 생성 알고리즘

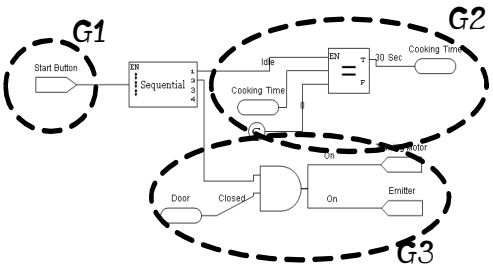
4.3 RTT(Requirement Translation Tree)의 생성

요구사항은 출력을 중심으로 설명할 수 있기 때문에 IORT를 출력 객체부터 거꾸로 탐색하면서 직접 번역한다. 그러나, 'Loop'나 'Sequential' 등 FC 객체를 가진 경우에는 자연어로 번역할 때, 번역 순서가 달라야 한다. 번역 측면에서 볼 때, 다른 객체 보다 우선해서 번역 되어야 하는 특징을 가진 FC 객체가 IORT 하위에 나타난다 하더라도 우선하여 번역 되도록 트리 구조상 상위에 나타나도록 하는 것이 적합하다. IORT를 번역에 편리하도록 재구성한 자료 구조를 RTT(Requirement Translation Tree)라 한다.

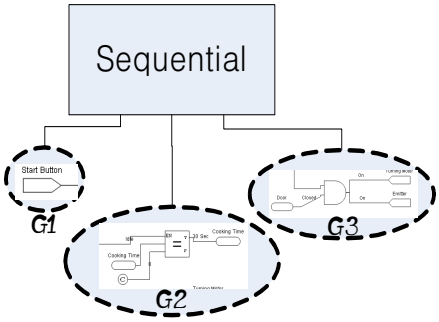
IORT는 두 단계를 거쳐 RTT로 변환된다. IORT를 RTT로 변환 시 2단계를 거치는 이유는 구현의 편의성 때문이다. 첫 단계는 IORT에서 사용된 FC 객체를 중심으로 바라본 그래프로 변형하는 작업이다. 이 그래프를 RTT로 만들기 위한 중간 그래프라는 뜻으로 IMT(Intermediate Tree)라고 부른다. IMT는 IORT에서 사용된 FC 객체를 루트 노드로 이동한 자료구조이다. 그리고 FC 객체 이외의 객체들은 모두 그룹화 하여 서브그래프라는 노드로 표현한다.

IMT를 만드는 상세한 알고리즘은 (그림 6)과 같다. 우선 IORT 내에서 사용된 하나의 FC 객체를 찾은 후, 이들 객체의 링크를 따라 가며 탐색하면서 찾은 객체들을 그룹으로 묶은 후, 이들을 FC 객체의 자식으로 만든다. 만약 이렇게 생성된 자식들 중에 또 다른 'Sequential'이나 'Loop' 객체가 있다면, 동일한 방법에 의하여 그룹화를 실시한다. 이러한 작업을 통하여 FC 객체를 가리키는 노드와 FC 객체가 포함되지 않은 서브그래프를 가리키는 노드 들로 이루어진 그래프를 만들 수 있다. 이 그래프를 IMT라 한다.

(그림 7)은 IORT(a)로부터 만들어진 IMT(b)의 모습이다.



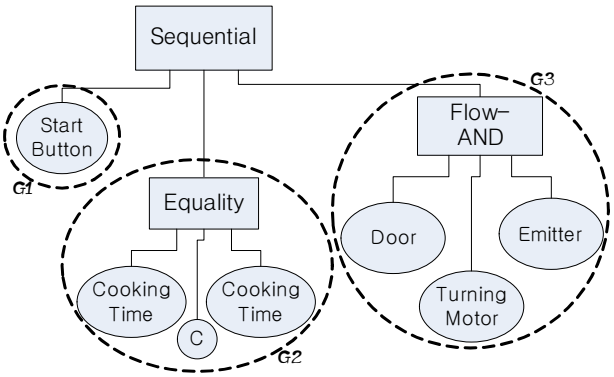
(a) IORT



(b) IMT

(그림 7) IORT(a) 및 IMT(b)

세 개의 서브그래프를 자식으로 가지며 'Sequential' 객체가 루트인 IMT 이다.



(그림 8) RTT(Requirement Translation Tree)

두 번째 단계에서는 IMT의 각 서브그래프들은 번역하기에 적합한 트리로 변형한다. IMT에서 구축된 서브그래프는 관련된 객체들의 IORT이다. 번역에 적합한 순서를 얻기 위해서는 IORT를 연산 객체를 중심으로 출력에서 입력 방향으로 거꾸로 정렬한다. (그림 7)의 각 서브그래프를 단순히 거꾸로 탐색하면서 연산 객체를 중심으로 트리를 구성하면 (그림 8)과 같은 RTT를 생성할 수 있다.

RTT를 생성하는 상세한 알고리즘은 (그림 9)와 같다. (그림 9)의 ReqTransTree 알고리즘은 IMT의 노드 종류, 즉 노드가 FC 객체인 경우와 서브그래프인 경우로 구별하여 처리한다. 루트 노드가 서브그래프인 경우는 IMT에 하나의

```
RTT ReqTransTree(IntermediateTree IntTREE)
{
    Let N be a root node of IntTree;
    1) if (N is a subgraph node) {
         R = convertSubgraph2Tree(N);
         return R;
       }
    2) Let R be a RTT whoose root node is N
       foreach (child node C of N) {
          (1) if (C is a subgraph node){
                 T = convertSubgraph2Tree (C);
                 Make subtree T into a child node of N;
              }
          (2) else {        // C is a FC object node
                 Let ST be a subtree in IntTree whoose root node is C
                 T = ReqTransTree(ST);
                 Make subtree T into a child node of N;
              }
       }
    3) return R;
}
RTT convertSubgraph2Tree(subgraph N)
{
    Let L be a last operation node of N;
    Let R be a RTT whoose root node is L;
    1)Mark all objects ∈ N as unvisited
    2)Create an RTT R, initially set L as root node of R, and mark L as visited
    3)while(there is an unvisited object Obj that is reachable by backward traverse from an object
       already visited) {
       (1)mark Obj as visited
       (2)make input objects of Obj children node of Obj;
    }
}
```

(그림 9) IMT로부터 RTT로 변환하는 알고리즘

노드만 존재하게 된다. 따라서 서브그래프만 트리로 변환하면 곧바로 RTT가 얻어진다. FC 객체가 루트인 경우는 자식들을 대상으로 다시 ReqTransTree을 재귀적으로 호출하여 IMT를 구축한다.

서브그래프를 트리 구조로 변형하는 알고리즘은 (그림 9)의 convertSubgraph2Tree 이다. 출력 객체에서 출발하여 거꾸로 탐색하면서 트리 구조로 만든다. 연산 객체는 루트에 놓이며, 연산 객체의 입력과 출력은 자식 위치에 놓인다. 어떤 자식이 입력인지, 혹은 출력인지는 연산 객체의 특성에 따라 결정된다.

4.4 자연어로의 번역

자연어로의 번역은 RTT를 전위순회로 탐색 하면서 이루어진다. 즉, RTT의 루트 노드를 번역 한 후, 루트의 각 서브 트리를 차례대로 번역한다. 각 서브 트리 역시 동일한 방법으로 전위순회로 탐색하면서 번역한다.

하나의 다이어그램에는 여러 개의 RTT가 생성될 수 있기 때문에 다이어그램의 화면상의 위치와 번역된 문장의 순서를 일치시키기 위해서 다이어그램의 화면 위치를 이용하여 RTT를 정렬하였고, 화면상 제일 위에 위치한 RTT 에서부터 해석 한다. 화면에서 보이는 순서대로 문장이 해석되어야 사용자가 이해하기 쉽기 때문이다.

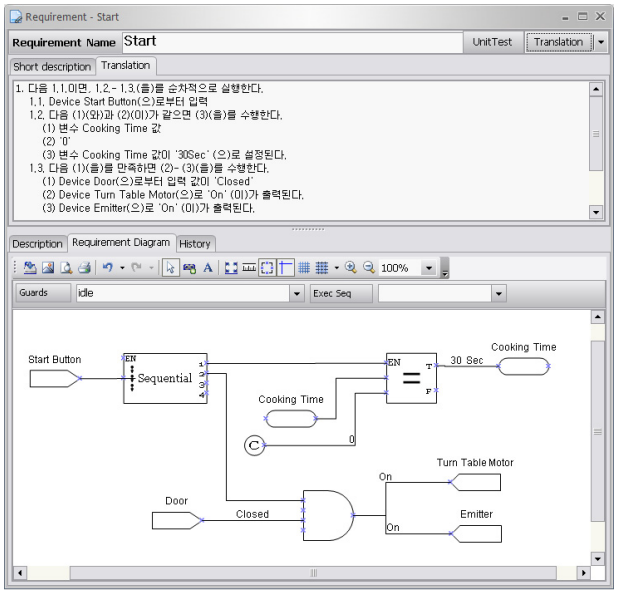
RTT의 각각의 객체는 전위순회로 탐색하면서 번역된다. 그러나 하나의 객체가 항상 동일한 문장으로 번역되지는 않는다. 객체가 다른 객체와 어떠한 연결 상태를 가지느냐에 따라 다르게 번역될 수 있다. 즉, 객체의 출력 링크에 값이 나올 경우에는 그 값을 해석에 포함시켜야 하고, 객체의 특정 포트에 링크가 연결되거나 연결되지 않을 경우 해석이 달라질 수 있다. 따라서 각 객체 별로 사용 패턴을 분류하고, 각 패턴에 대한 자연어 템플릿을 만들었다.

RTT의 노드 객체의 사용 패턴을 판단한 후, 해당 번역 템플릿의 문자열 일부를 RTT 내의 객체의 속성으로 대체한다. 사용 패턴은 객체의 타입, 입력을 제공하는 객체의 개수 및 유형, 출력을 넘겨줄 객체의 개수 및 유형 등에 의하여 결정된다. 객체들을 서로 연결하는 링크에 조건이 주어졌는지의 여부도 사용 패턴을 결정하는 데 매우 중요하게 사용된다. 자연어 자체는 사용 패턴을 결정하는데 영향을 미치지 않는다. 즉, 영어로 번역할 때와 한글로 번역할 때 사용하는 패턴은 동일하다. 사용 패턴은 자연어의 특성보다는 각 객체가 다른 객체와의 연결된 상태에 따라 결정된다. REED가 지원하는 총 44개 종류의 객체 중, 많이 사용되는 객체의 사용 패턴에 따른 템플릿의 수는 다음 <표 4>와 같다.

RTT가 많은 자식들을 포함하고 있을 경우, 많은 자식들을 포함한 하나의 다이어그램을 하나의 문장으로 번역하는 것은 번역 된 문장의 이해를 어렵게 한다. 다이어그램을 하나의 문장으로 표현하기 보다는 하나의 노드를 하나의 문장으로 번역하고, 각각의 문장에 문장 번호를 부여하면 이해하기가 쉽다. 이러한 작업은 RTT의 각 노드를 독립적으로 번역함으로써 새로운 객체가 추가되더라도 확장을 용이하게 한다. 문장 번호는 문장의 깊이에 따라 차례대로 1, 1.1, (1), (A), (a) 등으로 부여한다. (그림 10)은 번역된 모습과 문장 번호 부여 체제를 보여준다.

〈표 4〉 객체 별 템플릿 개수

객체의 종류	템플릿 개수
Input	5
Output	5
Memory	8 (input:4, output:4)
Event	4 (input:2, output:2)
Sequential	2
Equal	17
Bypass	8
Assignment	12



(그림 10) 자연어로 해석된 예제

자연어는 단어 이외에도 다양한 기능을 가진 조사를 사용한다. 조사는 단어가 주어인지 목적어인지를 나타내기도 하며, 단어와 단어를 연결하는데 사용되기도 한다. 앞 단어의 문자적 특성에 따라 조사가 달라지기도 한다. 예를 들면, “감”이라는 단어를 주어로 만들기 위하여 “이”라는 조사가 사용되지만 “가”라는 조사는 사용되지 않는다. 본 연구에서는 이러한 언어적 특성으로 발생하는 문제에 대한 해결을 향후 연구 과제로 미루고, “이(가)” 처럼 동일한 의미를 가진 조사를 모두 표기하는 방법을 사용한다. 즉 “감 이(가)”와 같이 번역 한다. 조사를 정교하게 구별하여 사용하는 알고리즘에 대한 연구가 이루어지는 경우, 다이어그램의 가독성이 크게 향상되리라 본다.

5. 예제와 평가

5.1 예제

REED를 이용하여 임베디드 시스템의 요구사항을 다이어그램으로 기술하고, 기술된 요구사항을 바탕으로 테스팅 작

업을 수행하였다. 자동차 내부 온도 자동조절장치(TC), 버스 요금 계산기(BusCard), 굴삭기 제어기 등 3개의 시스템의 요구사항은 워드프로세서 혹은 엑셀로 작성되어 있었다. 본 연구에서는 이들 요구사항들을 REED를 이용하여 요구사항 다이어그램으로 재구성하고, 재 작성된 다이어그램을 바탕으로 시스템들의 임베디드 소프트웨어를 테스트 하였다. 전자레인지(Oven)는 사용자 메뉴얼을 분석하여 요구사항 다이어그램을 작성하였다. 전자레인지는 사용자 메뉴얼로부터 요구사항 다이어그램 작성이 가능한가, 메뉴얼은 충분히 그리고 충돌 없이 전자레인지의 기능을 설명하고 있는지를 알아 보기 위함이었다. 전자레인지에 대하여는 시뮬레이션만을 수행하였으며, 실제 전자레인지에 임베디드된 프로그램을 테스트 하지는 못하였다.

요구사항 다이어그램의 작성 및 검증이 완료된 후, 각 시스템의 다이어그램을 본 논문에서 기술된 방법으로 한글 혹은 영어로 번역하였다. 각 장비들의 간단한 특성, 요구사항 다이어그램 관련 통계, 번역된 글에 관련된 통계는 <표 5>와 같다.

<표 5>에서 보는 바와 같이, 연구된 시스템들이 요구사항의 개수는 많으나 다이어그램으로 표기하기 위하여 사용된 객체의 수는 비교적 적다. TC는 다양하며 정교한 온도 조절 기능을 299개의 요구사항으로 표현하고 있다. 예를 들면, TC의 요구사항은 다양한 센서로부터 감지된 온도의 평균을 사용하거나, 센서의 고장에 대한 처리 기능 등 매우 다양하며 세밀한 제어 기능을 명시하고 있다. 이에 따라, 31종의 다양한 객체를 사용하여 요구사항을 재구성하였다. 그러나 굴삭기 제어기는 TC과 비교할 때 요구사항의 수는 많으나 사용된 객체의 수는 적다. 굴삭기 제어기는 TC보다는 많은 수의 입출력 장치들에 대한 제어를 기술하기 위하여 많은 요구사항을 명시하고 있다. 그러나 TC보다는 적은 28종의 객체를 사용하여 요구사항을 재구성할 수 있다. BusCard는 입출력 장치의 수는 적으나 다른 시스템과 TCP/IP 통신을 하는 분산 시스템이다. BusCard의 일부 기능만을 요구사항으로 표현하고 테스트를 수행하였다. TCP/IP 패킷에 기록된 정보는 많으나 그 처리가 비교적 간단하여 다이어그램에 사용된 객체의 종류는 적으며 요구사항당 많은 수의 객체를 사용하고 있다. 전자레인지는 다이어그램의 수는 적으나 다양한 객체를 사용하고 있다.

이러한 실험을 통하여 볼 때, 시스템의 종류나 기능, 복잡도에 따라 차이는 있으나, 대략적으로 20~30종의 객체를 사

〈표 5〉 시스템들의 통계

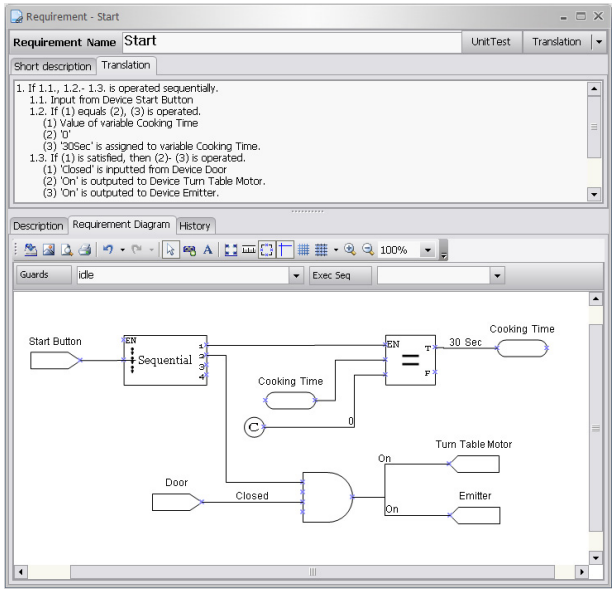
		시스템			
		TC	Bus Card	굴삭기 제어기	Oven
개요		자동차에 내부 온도조절장치	버스 요금 계산을 위한 운전자 단말기	굴삭기 제어기용 엔진과 펌프의 마력 매칭을 위한 장비	일반 전자 레인지
장치	입력장치	센서(7종), 스위치(13종)	키패드, GPS	센서(6종), 스위치(15종)	스위치(10종)
	출력장치	Actuator(8종)	LCD 스피커	Solenoid Valve 등 (7종)	LCD Actuator (Motor, Microwave Emitter)
	입/출력장치	CAN	RS-232, 무선 랜 USB	CAN bus (engine 등) Gauge Pannel	
요구사항 총 개수		299	262	373	26
사용된 객체의 종류		31	23	28	20
객체의 최대 사용 개수 (요구사항별)		58	53	37	17
객체의 평균 사용 개수 (요구사항별)		9.49	7.29	4.54	7.96
문장의 최대 깊이 (요구사항별)		5	5	5	3
문장의 평균 깊이 (요구사항별)		2.08	1.73	1.32	1.87
번역 문장 최대 줄 수 (요구사항별)		44	64	44	18
번역 문장 평균 줄 수 (요구사항별)		10.26	8.11	4.46	8.35
IORT 최대 개수 (요구사항별)		15	12	7	4
평균 IORT 개수 (요구사항별)		1.91	1.52	1.31	1.87

용하여 250~400개 정도의 요구사항 다이어그램이 표현됨을 알 수 있다. 문장의 최대 깊이는 번역된 문장들의 깊이를 가리킨다. TC, BusCard, 굴삭기 제어기의 경우 최대 깊이가 5이다. 깊이가 5인 요구사항의 경우, 사용된 문장 번호 부여 방식은 1, 1.1, (1), (A), (a) 등이다. 문장의 깊이는 RTT의 레벨을 가리키며, 하나의 출력을 내보내기 위해서 사용된 객체의 단계가 최대 5 단계라는 의미이기도 하다.

번역 문장 최대 줄 수는 가장 길게 번역된 문장의 줄 수이다. BusCard의 경우 최대 64줄의 문장으로 구성된다. 하지만 문장 최대 깊이는 5 레벨임을 알 수 있다. 가장 긴 문장은 이더넷으로 수신되거나 혹은 이더넷으로 송신할 패킷에 담긴 정보들을 구별하기 위하여 많은 객체들을 사용하는 문장이다. 이러한 경우, 패킷에 기록된 정보 각각에 대한 처리를 하나의 문장으로 번역하면서 많은 수의 문장이 필요하였으나, 그 경우의 문장의 깊이는 깊지 않다. 즉, 이 요구사항의 경우, 사용된 객체가 많아서 번역이 수행된 결과 문장은 길지만, 하나의 출력을 내보내기 위해서 필요한 객체의 단계는 많지 않다는 의미이다.

요구사항당 사용된 객체의 개수를 보면 굴삭기 제어기의 경우는 4.54개인 반면에 TC의 경우에는 9.49개로 2배 이상의 많은 차이가 나는 것을 볼 수 있다. 물론 시스템의 복잡도에 따라 다르겠지만 이 숫자는 굴삭기 제어기의 요구사항이 훨씬 간결하게 작성되었다는 것을 알 수 있다. 굴삭기 제어기의 요구사항은 TC 보다 약 70개 가량이 더 많다. 하지만 번역 문장 평균 줄 수는 굴삭기 제어기가 4.46으로 훨씬 적다. 따라서 굴삭기 제어기의 요구사항이 훨씬 잘게 쪼개져 있다고 생각할 수도 있다. 이처럼 요구사항을 작은 단위로 나누어서 작성하는 경우, 번역된 문장의 줄의 수가 줄어들고 가독성도 좋아진다.

5.2 평가와 응용



(그림 11) (그림 10)에 대한 영어 번역 결과

출력을 기준으로 기술된 번역은 해당 출력이 생산되는 정확한 절차와 조건을 설명하고 있다. 즉, 번역된 문장은 입력과 출력의 관계를 다이어그램보다는 오히려 더욱 분명하게 설명하고 있다. 번역은 한 곳에서 여러 개의 출력을 생산하는 요구사항에 대하여 알려주었다. 작은 요구사항은 읽기 쉬우며 명확하고, 테스트하기도 쉽다. 또한, 요구사항을 작은 단위로 나누도록 하면 요구사항을 작성하는 사람들로 하여금 요구사항을 더 이상 쪼개지지 않도록 만들려고 많은 노력을 기울이도록 한다. 이처럼 번역이 요구사항을 단순히 번역하여 개발과정에 참여하는 계층들 간의 격차를 없애주는 역할을 할 뿐만이 아니라 요구사항을 보다 단순하게 만드는데 도움을 준다는 사실을 알 수 있다.

(그림 11)의 결과처럼 한글과 영어를 모두 지원하여 한글로 제시된 초기의 요구사항을 영어로 번역하는 효과도 얻을 수 있었다. 또한 고객의 입장에서는 자신의 요구사항이 모델로 작성되고, 고객이 이해하기 힘든 모델링 표현을 자연어로 번역하여서 확인할 수 있어서 많은 오류를 방지할 수 있었다.

6. 결론 및 향후 과제

자연어로 작성된 요구사항은 모호성이 많아서 직접 개발에 사용하기에 힘들기 때문에 그래픽적인 표기법을 사용하여 요구사항을 표기한다. 본 논문에서는 그래픽적인 표기법으로 표현된 요구사항을 자연어로 번역하는 알고리즘을 제시하고 상용 제품을 상대로 알고리즘을 적용하여 보았다.

그래픽적인 표기법으로 표현된 요구사항을 자연어로 번역하는 기능은 개발과정에 참여하는 모든 계층의 사람들이 요구사항을 이해할 수 있도록 도와준다. 또한 요구사항을 보다 단순하게 만드는데 도움을 준다. 표기법에 익숙하지 않은 사람들도 무슨 의미로 작성한 요구사항인지를 이해하는데 큰 도움이 되었다. 제안한 알고리즘을 적용하여 다양한 분야의 상용 시스템의 요구사항을 번역한 결과는 자연어에 매우 가까워 그 효용성이 입증되었고, 여러 계층의 사람들이 요구사항을 이해하는데 실제적인 도움이 되었다.

현재는 한국어와 영어를 지원하고 있지만, 앞으로 보다 많은 언어를 지원할 수 있도록 확장 중에 있다. 또한 개발된 알고리즘은 본 연구에서 적용한 REED이외에도 Simulink나 UML 등으로 기술한 요구사항을 번역하는데도 응용할 수 있을 것으로 예상한다.

참 고 문 헌

[1] J. Martin, *An information Systems Manifesto*, Prentice Hall, 1984.

[2] B. Boehm, and P. Philip, "Understanding and Controlling Software Costs," *IEEE Transactions on Software Engineering*, Vol.14, No.10, pp. 1462-1476, October, 1988.

[3] D. Leffingwell, "Calculating the Return on Investment

from More Effective Requirements Management," *American Programmer,* Vol.10, No.4, pp.13-16, April, 1997.
[4] Object Management Group, "Unified Modleling Language (UML), Version 2.1.2", http://www.omg.org/spec/UML/2.1.2/, November, 2007.
[5] The MathWorks, Inc., http://www.mathworks.com/products/simulink/
[6] R. Saracco and P. A. J. Tilanus, "CCITT SDL: Overview of language and its application," Computer Networks and ISDN Systems, Vol.13, No.2, pp.65-74, March, 1987.
[7] D. Liu, K. Subramaniam, A. Eberlein and B. H. Far, "Natural language requirements analysis and class model generation using UCDA," *Proceedings of the 17th international conference on Innovations in applied artificial intelligence*, pp.295-304, Ottawa, Canada, May, 2004.
[8] M. Ilieva and O. Ormandjieva, "Automatic Transition of Natural Language Software Requirements Specification into Formal Presentation," *Proceedings of the 10th International Conference on Applications of Natural language to Information system*, pp.392-397, Alicante, Spain, June, 2005.
[9] I. A. Niaz, "Automatic Code Generation From UML Class and Statechart Diagrams," *Ph.D. Dissertation, University of Tsukuba*, November, 2005.
[10] B. Lee and R. Bryant, "Automated conversion from requirements documentation to an object-oriented formal specification language," *Proceedings of the 2002 ACM symposium on Applied computing*, pp.932-936, Madrid, Spain, March, 2002,
[11] Yijun Yu, Yiqiao Wang, J. Mylopoulos, S. Liaskos, A. Lapouchnian, and J.C.S. do Prado Leite, "Reverse engineering goal models from legacy code," Proceedings of 13th IEEE International Conference on Requirements Engineering, pp. 363-372, Aug., 2005.
[12] E. Korshunova, M. Petkovic, M. G. J. Brand and M. R. Mousavi, "CPP2XMI: Reverse Engineering of UML Class, Sequence, and Activity Diagrams from C++ Source Code," *13th Working Conference on Reverse Engineering (WCRE 2006)*, pp. 297-298, Benevento, Italy, October, 2006.
[13] Graphical Development Process Assistant, "Specification and Description Language (SDL)", http://www.informatik.uni-bremen.de/gdpa/methods/m-sdl.htm.
[14] E. Haywood and P. Dart, "Analysis of Software System Requirements Models," *Proceedings of the 1996 Australian Software Engineering Conference (ASWEC '96)*, pp.131-138, Melbourne, Australia, July, 1996.
[15] C. Seybold, S. Meier and M. Glinz, "Evolution of Requirements Models by Simulation," *Proceedings of 7th International Workshop on the Principles of Software Evolution (IWPSE '04)*, pp.43-48, Washington, DC, USA, September, 2004.
[16] 오정섭, 이홍석, 박현상, 김장복, 최경희, 정기현, "그래픽 언어를 이용한 임베디드 시스템의 단일 요구사항 모델링," 정보처리학회논문지D, 제15-D권 제4호, 2008년 8월

오 정 섭
e-mail : jsoh@ajou.ac.kr
1997년 아주대학교 정보및컴퓨터공학부 (학사)
1999년 아주대학교 대학원 컴퓨터공학과 (석사)
1999년~현 재 아주대학교 일반대학원 컴퓨터공학과 박사과정
2001년~2003년 ㈜디오텔 선임연구원
2003년~2006년 ㈜삼성탈레스 책임연구원
관심분야 : 소프트웨어 공학, 요구사항 공학, 임베디드 시스템, 실시간 시스템, 분산 시스템 등

이 혜 련
e-mail : cocom12@ajou.ac.kr
2006년 조선대학교 인터넷소프트웨어공학과 (공학사)
2008년 아주대학교 정보통신전문대학원 (공학석사)
2008년~현 재 아주대학교 일반대학원 컴퓨터공학과 박사과정
관심분야 : 소프트웨어 공학, 명세 기술, 임베디드 시스템 등

임 강 빈
e-mail : yim@sch.ac.kr
1992년 아주대학교 전자공학과(공학사)
1994년 아주대학교 전자공학과(공학석사)
2001년 아주대학교 전자공학과(공학박사)
1999년 3월~2000년 2월 (미)아리조나주립대학교 연구원
2003년 3월~현 재 순천향대학교 정보보호학과 교수
2005년 3월~현 재 한국정보보호학회 이사
관심분야 : 임베디드 시스템 보안, 운영체제 보안, 접근제어 등

최 경 희
e-mail : khchoi@ajou.ac.kr
1976년 서울대학교 수학교육과(학사)
1979년 프랑스 그랑데꼴 Enseeiht대학(석사)
1982년 프랑스 Paul Sabatier대학 정보공학부(박사)
1982년~현 재 아주대학교 정보통신전문대학원 교수
관심분야 : 운영 체제, 분산시스템, 실시간 및 멀티미디어시스템 등

정 기 현

e-mail : khchung@ajou.ac.kr

1984년 서강대학교 전자공학과(학사)

1988년 미국 Ilinois주립대 EECS(석사)

1990년 미국 Purdue대학 전기전자공학부 (박사)

1991～1992년 현대반도체 연구소

1993년～현 재 아주대학교 전자공학부 교수

관심분야 : 컴퓨터구조, VLSI 설계, 멀티미디어 및 실시간 시스템 등