컴퓨터 바둑 프로그램의 設計 明細 硏究

김 영 상*, 이 종 철**

On Computer GO Program Design Description

Young-Sang Kim and Chong-Cheol Lee

Abstract Like computer GO, in complex problem solving system, design description of program specify an ability which a player decides a next move accurately because planning is intermixed with execution which changes the environment at each move. In this paper, we discuss fundamental design concepts for software product and suggest a document format of software design description for Computer Go program. Suggested software design description could be used for the GO program revision conventionally.

I. 서 론

컴퓨터 공학의 눈부신 발전은 과거에 수작업으로 처리하던 기계, 전기 등의 다양한 산업 자동화에 기여해왔다. 또한, 전산 처리 시스템에서 소프트웨어의 의존도가 높아지면서 유지 및 보수에 대한 체계적인 접근 방식의 필요성이 크게 대두되면서 소프트웨어 제품의 품질을 향상시키고, 소프트웨어 엔지니어의 생산성 증진을 그 목적으로 소프트웨어의 개발 요구에 대한 분석 및 정의, 소프트웨어 개발 과정의 제어와 가시성에 대한 연구가 꾸준히 이루어지고 있다. 이를테면, 소프트웨어 개발 관리자들이 당면한 요구 분석의 타당성(satisfaction), 생산성(productivity), 가시성(visibility), 비용(cost) 등의 문제를 해결하기 위하여 구조화 프로그래밍(structured programming), 하향식 기법(top-down), 치프 프로그래밍 (chief programming)등 많은 기술이 제안되어 사용되고 있으며, 인간의 활동 주기처럼 소프트웨어 시스템도, 계획에서 시작하여 필요성이 없게 될때까지 관리하는 기법이 대두되고 있다. 따라서, 대부분의 문제 풀이 시스템에서는 요구 사항의 정의, 설계 명세가 중요한 요소로 부각되는데, 특히 바둑은 계획이 문제 풀이 과정중에 혼합되어 목표를 계속 변화시키므로 그때 그때의 국면에 대해서 다음 수를 정확하게 결정할 수 있도록 설계되어야 한다. 과거에는 설계보다는 프로그래밍에만 편중함으로서 출력물 중심과 개인의 능력에 치중한 나머지 편협한 설계 명세로 인한 모듈 재사용이 힘들었으나, 소프트웨어 공학의 중요성과 관련있는 도구(tools)들의 개발이 본격적으로 이루어지면서 체계적인 문서화에도 주력하고 있다. 본 연구에서는 소프트웨어 개발시 효율적인 개발 및 유지 보수의 용이, 전반적인 프로젝트의 관리에 유용한 소프트웨어 개발 주기에 따른 시스템 개발에서 정의되는 요구 사항을 소프트웨어로의 변환 단계인 설계(design)에 관한 제반 정보의 문서화 및 명세 속성에 대한 모델을 제시하고, 이를 컴퓨터 바둑 프로그램 개발 프로젝트에 적용하였다.

본 논문의 구성을 보면, 제 2 장에서는 소프트웨어 시스템 개발시 이용되는 주요 설계 방법

1993년 1월 14일 접수
* 경북대학교 대학원 컴퓨터공학과 석사과정
** 경북대학교 공과대학 컴퓨터공학과 부교수

론중에서 하향식 방법을 이용한 바둑 프로그램의 구조를 제시하였고, 제 3 장에서는 소프트웨어 설계 명세 속성 및 명세서에 대해 기술하고, 컴퓨터 바둑 프로젝트의 문서화를 그 실예로 제시한다. 제 4 장에서는 바둑 프로그램 명세의 필요성과 향후 설계 방법론의 추세 등에 대해서 논의한다.

II. 바둑 프로그램의 설계 방법론

소프트웨어의 기본 설계는 소프트웨어 제품의 외부적인 특징을 생각하고, 계획하며, 이를 기술하는 과정이며, 이때 취해야할 작업으로는 화면 설계, 보고서 형식 결정, 외부의 자료 공급처, 기능적인 특징, 성능 요구 사항, 제품의 프로세스 구조 등을 만든다. 설계 단계는 주로 기본 설계(preliminary design)와 상세 설계(detailed design)로 구분하는데, 기본 설계는 요구 사항 정의 및 분석 단계에서부터 비롯하여 시스템의 구조적 측면을 높은 차원에서 설정하는 것이고, 상세 설계라 함은 소프트웨어 제품의 내부 구조 및 처리 내역을 고려하여 설계 상의 결정 사항을 기록하며, 설계 대안들을 분석하여 원하는 설계의 선택 배경을 설명하거나 시스템의 개념적 측면을 명세화하고, 내부적인 처리 기능을 밝히고, 고차적인 기능을 서브 함수로 분해하며, 내부적 통신 자료 및 자료 저장소를 정의하고, 기능 자료의 흐름 및 자료 저장소간의 관계와 상호 작용을 정의하며, 일괄적인 테스트 계획을 수립하고, 구현, 테스팅, 유지 보수에 대한 청사진을 제공하는 역할을 한다.

소프트웨어 설계의 기본 개념으로는 분리와 합병(divide and conquer), 추상화(abstraction), 정보은폐(information hiding), 구조화(structuring), 모듈화(modularity), 병행성(concurrency), 검증 등을 들 수 있으며, 본 장에서는 이들 기본 개념을 바탕으로 하여 바둑 프로그램의 하향식 개발을 중심으로 설명한다. E. Yourdon[11]이 고안한 하향식 설계 기법은, 전

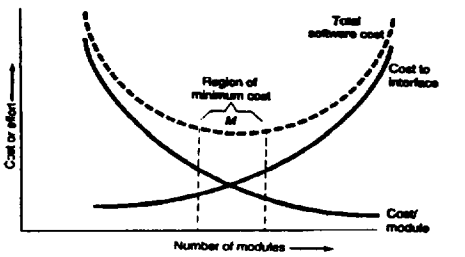

그림 1. 모듈 수와 소프트웨어 비용간의 관계
Fig. 1. Modularity and software cost.

체 시스템을 기능상의 차이에 따라 여러 모듈로 분해하여 순차적으로 문제를 해결해 나가는 기법으로, 설계의 최상위 단계에서 추상적인 알고리즘을 정의한 후, 최하위 단계의 모듈에서 상위 단계의 역할을 구현할 기본적인 연산을 제공한다. 계속되는 정련(refinement)과정은 기본적인 시스템 모델을 단계적인 분해 과정을 통하여 처리 가능한 모듈이 산출될때까지 계속 나뉘어진다. 이때, 모듈은 분해시에 생기는 작은 조각으로서, 프로그램의 작업 할당 또는 유일한 기능을 수행할 수 있어야 한다. 경우에 따라서 한 모듈은 다른 서브 모듈을 포함할 수도 있다. 모듈간의 인터페이스는 해당 모듈을 이용하는 프로그래머가 이외 기능에 행할 가정을 말하는데, 설계의 질을 향상시키는 요소로[3]에서는 일치성(consistency), 필연성(essential), 일반성(general), 최소성(minimal), 은닉성(opaque)를 기준으로 제시하고 있다. 그림 2.1에서 보는 바와 같이 과소 모듈화(undermodularity) 및 과대 모듈화(overmodularity)의 현상은 피하고[8], 적절한 모듈 크기를 고려한 시스템 개발이 요구된다. 이는 시스템에서 모듈로의 반복적인 분해 과정 즉, 정련과정을 통해 요구하는 시스템을 설계하기 때문이다. 이를 근간으로 그림 2.2에서 전체적인 바둑 프로그램의 구조중 2단계까지만을 보였다.

그밖에 권고할만한 설계 방법으로는, 자료 구조 지향 설계법(Data Structure Oriented

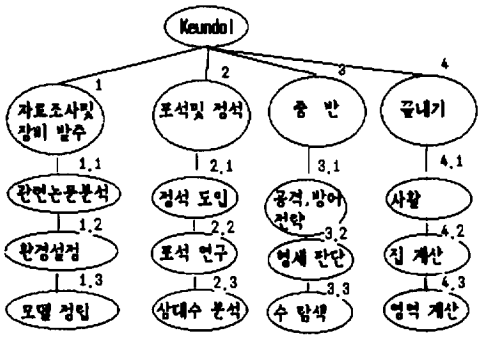

그림 2. 바둑 프로그램의 하향식 설계 구조
Fig. 2. Top down design of computer GO.

Design)과 객체 지향 설계법(Object Oriented Design)을 들 수 있는데, D. Jackson & P.Wanier에 의해 제안된 자료 구조 지향 설계 방식은 COBOL언어를 이용한 응용 소프트웨어 시스템 개발에 유용하고, 특정 소프트웨어 시스템에서 요구되는 입력 데이타, 내부 저장된 정보(database), 출력 데이타 등은 각각 유일한 자료 구조를 가지며, 이 기법에서는 이들 자료 구조를 소프트웨어의 기초로 활용한다. 즉, 시스템 설계시 자료 구조를 먼저 정의한 후, 정의된 자료 구조를 중심으로 한 프로그램 단위로의 분해 과정을 통해 시스템을 설계한다. 하향식 방법과의 차이점은, 자료 흐름도(Data Flow Diagram)를 명확히 이용하지 않고, 모듈간의 독립성에 무관하며, 정보 구조를 표현하기 위해서 계층적 Wanier 다이어그램을 이용하고, 예비 설계와 상세 설계의 정확한 구별이 어렵다는 점 등을 들 수 있다. 또한, 객체 지향 설계 기법은 Dijkstra에 의해 정의된 추상화(abstraction)이론과 Panas에 의해 제시된 정보 은폐(information hiding)이론을 근간으로 하여, 매사츄세츠 연구소의 B.Liskov & J.Guttag의 추상적인 데이타형(abstract data type)의 연구 이론에 영향을 받아 R. Abbot에 의해 본격적으로 소프트웨어 설계의 신기술로 소개되었다. 이는 ADA로 구현할 소프트웨어 시스템 설계를 위하여 제안된 것인데, 시스템을 객체(object) 및 변형 함수(transformation function) 집합으로 간주한다[7].

객체 지향 설계법의 기본 과정은 구조적 분석 및 설계 기법(structured analysis and design techniques)과 자료 흐름도로써, 주어진 문제에 대한 설계 및 구현에 수반하는 사항을 정의하는 문제 정의 단계, 정의된 문제에 대해 수행될 타스크를 언어로 표기하는 비정형화 전략 개발 단계, 객체의 속성 식별과 객체상에서 수행될 모든 연산의 식별, 객체의 인터페이스 형성, 프로그램의 원형 작성및 연산의 구현들을 총괄하는 전략의 형식화 단계로 나누어진다. 이 기법은 복잡한 자료 구조의 사용시에 유용하며, 다른 기법과 비교할때 좋은 모듈화를 유도하고, 판독성, 유지 보수의 용이성등의 잇점이 있다.
일반적으로 소프트웨어 시스템의 개발은 제어와 자료 구조의 두 측면이 항상 고려되므로 이들 설계 방법의 몇몇 장점들을 혼합한 프로젝트의 진행이 바람직하다고 하겠다.

III. 바둑 프로그램의 설계 명세(Software Design Description)

이 장에서는 설계 명세의 구성 요소와 소프트웨어 설계 명세(SDD)에 대한 정보를 제시하고, IEEE의 권고안을 참고로, 기업체 또는 연구소에서 프로젝트 개발시 주 관심 분야의 하나인 설계 단계의 문서화를 살펴보고, 본 연구실에서 진행중인 컴퓨터 바둑 제품(프로젝트명: Keundol)을 실예로 들어 설계 명세서를 제안한다.

1. 설계 엔티티(Design Entity)

설계 단계는 프로그래밍의 바로 전단계로, 요구 분석 및 기능 명세 단계에서 명시된 각 기능들을 구현할 소프트웨어의 내부 구조를 표현하게 되며, 소프트웨어의 구현 단계에 사용될 주요 알고리즘과 자료 구조의 선정 또는 다른 대안의 평가로 행하여지므로, ① 소프트웨어 제품의 기능, ② 시스템의 종합 구조, ③ 시스템의 각 부분, 혹은 타 시스템간의 인터페이스, ④ 여

러 설계 대안중에서 최적의 설계안 선택, ⑤ 주어진 제약 조건하에서 신뢰도, 정확도, 가격등의 상치된 요구 사항을 고려한 설계와 같은 사항을 포함해야 한다. 이 단계의 결과는 프로그래밍의 청사진으로 각 모듈의 구조를 표현하는 구조 도표(structure chart)와 각 모듈의 내용을 문서화한 SDD로서 설명될 수 있다. 이러한 구조 도표와 SDD를 작성할때는 설계 엔티티를 적절히 기술하여 체계적으로 문서화할 수 있다. 설계 엔티티는 구조 및 기능적으로 구별되면서 유일한 명칭과 참조 방법을 갖는 설계 요소로서, 소프트웨어 설계 명세 단계에서 요구될 최소한도의 세트로서, 명칭, 타입, 목적, 기능, 부속 관계, 종속 관계, 인터페이스, 자원, 처리, 내부 데이타 등의 속성에 관련된 정보를 정의하게 된다. 바둑 프로그램에서의 다음 수 선정에 대한 설계 엔티티는 그림 3.1과 같다.

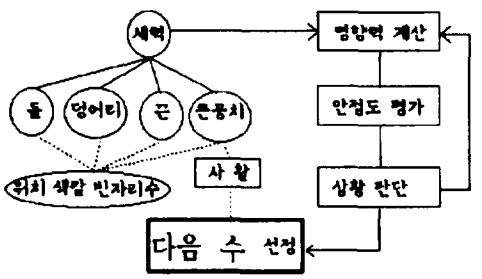

그림 3. 다음수 선정의 설계 엔티티
Fig. 3. Design Entity of next move.

2. 설계 명세(Design Description)

설계 관점(design views)이란 소프트웨어 프로젝트의 수행 요구상 적합한 설계 엔티티 속성의 부분 집합을 말하며, 설계 관점의 차이에 따라 엔티티 속성의 다른 구성을 갖는다. 본 연구에서는[1,20]에서 소개된 소프트웨어 설계 명세를 컴퓨터 바둑 프로그램 개발 프로젝트에 적합하도록 수정한 문서화 모델을 그림 3.2에서 제시하고 있으며, Keundol 프로젝트의 여러 모듈중에서 영역 계산[13] 모듈의 명세는 다음과 같다.

1. 서 론
  1.1 목 적
  1.2 범 위
  1.3 참고 문헌
2. 자료 구조 설명
  2.1 자료 구조 1 설명
  2.2 자료 구조 2 설명
3. 분해 명세
  3.1 모듈 구성
  3.2 모듈의 분해 명세
4. 상세 설계
  4.1 모듈 1 명세
    4.1.1 함수 1 상세 설계
      4.1.1.1 구성
      4.1.1.2 인터페이스
      4.1.1.3 알고리즘
      4.1.1.4 구현상 설계 제약 조건
    4.1.2 함수 2 상세 설계
  4.2 모듈 2 명세

그림 4. Keundol의 소프트웨어 설계 명세서
Fig. 4. SDD for computer GO, Keundol.

1. Territory Calculation Module (영역 계산 모듈)
이 모듈은 임의의 바둑 국면에 놓여진 각 돌에 대한 힘의 세기와 영향력을 구하고, 형세 판단의 자료로 활용한다.
1.1 명 칭 : Score.c
1.2 관련 함수 : abs()
1.3 포함 함수 : check_influence(), score_clear(), score_buffer(), check_color()
1.3.1 check_influence()
이 함수는 각 돌에 대한 영향력을 계산한다.
1.3.1.1 구 성 : x, y, color, flag
1.3.1.2 인터페이스 : stone의 위치 (x, y)
1.3.1.3 알고리즘
(1) x,y가 국면밖이면 리턴
(2) x,y에서 국면 전체에 걸쳐 힘의 크기를 구한다.
(3) 위에서 계산된 값을 현재 영향력에 합산하거나 최대치를 구한다.
1.3.1.4 구현상 제약 조건
놓여진 돌간의 거리는 백터 값이다.
1.3.2 score_color()

설계 명세는 분해 명세, 종속성 명세, 인터페이스 명세, 상세 명세로 분류할 수 있는데, 분해

명세(decomposition description)는 전체 시스템을 설계 엔티티로 분류한 내용의 기술로 전체적인 시스템 구성, 엔티티의 기능 및 목적, 추후 각 엔티티의 상세 명세의 참조 가능한 명칭등을 제공하며, 각 엔티티에 관하여 명칭, 타입, 목적, 기능, 부속 함수 등의 엔티티 속성에 대해서 기술한다. 이때, 전체적인 시스템의 구성은 일반적으로 계층 분해도(hieraracical decomposition diagram)와 자연어에 의해서 표현된다. 종속성 명세(dependency description)는 설계 엔티티간의 관계를 정의하는 것으로 종속 엔티티, 엔티티의 결합력, 요구되는 자원, 설계 엔티티간의 상호 인터페이스 방법, 실행 순서, 엔티티간에 존재하는 관계의 유형 등을 식별 기술하며, 이 명세는 보통 공식적인 명세 언어(formal specification language) 또는 자료 흐름도(data flow diagram), 구조 도표(structure chart), 트랜잭션 다이아그램 등으로 표현된다. 각 엔티티에 대해서는 명칭 타입, 목적, 종속성 및 자원 등의 속성을 상세히 기술한다. 인터페이스 명세(interface description)서에는 시스템 설계자, 프로그래머, 테스터들이 해당 함수의 이용시 필요한 모든 정보를 제공하게 되며, 소프트웨어 요구사항에서 제공되지 않은 내, 외부 인터페이스의 상세 명세를 비롯한 각 설계 엔티티에 대한 명칭, 기능, 인터페이스 등의 속성도 포함된다. 상세 명세(detailed description)은 시스템 구현전에 프로그래머가 필요한 엔티티의 내부적인 상세 설계 명세를 정의하는 것으로 각 엔티티에 대한 명칭, 처리, 데이타 등의 속성에 따라 상세히 기술하며 CASE와 같은 언어, 메타 코드, 구조화된 영어, 흐름도(flowchart)등의 도구를 사용한다.

Ⅳ. 설계 명세서의 중요성

컴퓨터 소프트웨어 혹은 소프트웨어 제품은 원시 코드는 물론, 이를 구성하는 관련 문서 및 문서화 체계로서 요구 사항 문서, 설계 사양, 테스트 계획, 운영 원칙, 품질 보증 절차, 소프트웨어 문제 보고서, 유지 보수 절차, 사용자 지침서, 설치 명령서, 훈련 교재뿐만 아니라 특정 문제를 해결하기 위해 개발된 응용 프로그램도 포함한다. 원시 코드의 내부 문서화도 필연적이지만, 체계적인 유지 보수를 위해서는 외부 문서화가 절실하게 요구된다. 품질(quality)이 선호되는 추세에 부응하여 가용성(usefulness), 선명성(clarity), 신뢰성(reliability), 효율성(efficiency), 비용 효과성(cost-effectiveness)등의 품질 속성을 고려하여 사용자의 요구를 최대한 만족시키려면, 유지 보수에 적절한 문서화 체계를 정립해야 한다. 특히, 시스템 설계 단계의 명세는 알고리즘을 표현하는 것이므로, 제품 실패를 방지하고, 재사용 또는 비용 절감의 효과를 얻을 수 있다. 보통 소프트웨어 제품의 개발 기간이 1~3년인데 비해서, 유지 보수 기간이 5~15년이라는 사실은, 문서화가 수반되지 않은 노력의 분포는 생산성을 저하시킬 수 있음을 말해준다. 따라서, 이러한 문서화를 자동화해주는 CASE(computer aided software engineering)와 같은 도구의 개발은 시급한 과제라 하겠다. CASE[9]는 제공되는 범위에 따라서 툴킷(toolkit)과 워크벤치(workbench)로 나누는데, 툴킷은 소프트웨어 개발 주기중 일부분만을 제공하고, 워크벤치 또는 통합 CASE는 전(全)단계의 모든 기능을 제공하는 도구이다. 현재 보급되고 있는 CASE로는 Index Technology사의 EXCELLARATOR, Knowledge Ware사의 IEW(informartion engineering workbench), Arthur Andershon사의 FOUNDATION, ANATOOL, POSE(picture oriented software engineering), Oracle등이 있다.

Ⅴ. 결 론

일반적으로 프로젝트 개발 주기(Project Development Life Cycle)는 개략적인 요구 사항 분석을 통해 이의 설계, 코딩, 테스트 및 수정

단계를 거치는 동안 요구 사항의 변동시, 이 사이클의 반복적인 수행을 통하여 요구하는 소프트웨어 시스템에 접근한다.

본 연구에서는 하향식 방법으로 설계한 컴퓨터 바둑 프로그램의 구조를 제시하였고, 다음수 선정의 설계 엔티티를 체계적인 문서화에 유용하도록 설정하였다. 이는 다른 소프트웨어 프로젝트를 개발할 때에의 문서화 정보로도 이용될 수 있으며, 다른 속성을 추가 또는 삭제 보완함으로서 프로젝트의 특성에 적절한 기법을 선정하는데 도움이 된다. 앞으로는 코딩 중심에서 벗어나 CASE 등과 같은 문서화 체계의 이용으로 소프트웨어의 비용을 절감하고, 생산성 증대와 품질 개선 등을 고려하는 것이 바람직하다.

참 고 문 헌

1. *"IEEE recommended practice for software design descriptions,"* ANSI/IEEE Std. 1016-1987, The Institute of Electrical and Electronics Engineers, Inc.
2. *"An american national standard IEEE guide to software requirements specifications,"* ANSI/IEEE Std. 830-1984, The Institute of Electrical and Electronics Engineers, Inc..
3. D. Hoffman, "On criteria for module interface", *IEEE Trans. on Software Engineering*, vol. 16, no. 5, May 1990.
4. G. Booch, "Object oriented design in tutorial on software design techniques," *P. Freeman and A. I. Wasserman, Eds.*, : IEEE Computer Society Process, 1983.
5. M. L. Shooman, *Software Engineering: Design, Reliability, and Management*, McGraw-Hill , Inc., pp. 107-113. 1983.
6. M. Olan, *Unconventional Design*, Computer Language, pp. 36-41, May 1988.
7. P. Jalote, "Functional refinement and nested objects for object oriented design," *IEEE Trans. Software Eng.*, vol. SE-15, no. 3, pp. 264-270, Mar.1989.
8. R. S. Pressman, *Software Engineering: A Practitioner's Approach*, McGraw-Hill International Editions, pp. 323-325. 1992.
9. C. McClure, *CASE is Software Automation*, Printice Hall, 1989.
10. L. E. Moser, "Data dependency graphs for ada programs," *IEEE Trans. on Software Engineering*, vol. 16, no. 5, May 1990.
11. R. Fairley, *Software Engineering Concepts*, Addsion Wesley, pp. 137-151. 1985.
12. S. R. Schach, *Software Engineering*, The Aksen Associates, pp. 43-58, 1990.
13. Y. S. Kim., "Modeling of the probability theoretical siatuation judgement appropriate for a dynamic environment," *M. Eng. Thesis*, Dept. of Computer Engineering, KPNU, pp. 23-26, 1992.