S/W 재사용을 위한 새로운 접근법으로서의 Domain Theory의 개요

함동한, 김진삼, 하수정, 조진희
한국전자통신연구원 컴퓨터소프트웨어연구소

Outline of Domain Theory As a New Approach to Software Reuse

Dong-Han Ham, Jin Sam Kim, Su Jung Ha, Jin Hee Cho
Electronics and Telecommunications Research Institute
E-mail : {dhham, jinsam, hsj, chojh}@etri.re.kr

Abstract

Over the years, several different approaches have been proposed to enhance the practice of software reuse. These approaches vary in reuse abstraction levels or application domains they focus on. However, it seems that they do not adequately meet the needs of software designers in planning and managing reuse. One reason may be the lack of conceptually well-established framework supporting work domain analysis and modeling. As a new viable solution to this problem, Sutcliffe and his colleagues developed Domain Theory , which is introduced in this paper. Utilizing multidisciplinary perspective, such as cognitive psychology, management science, human-computer interaction, and software engineering, it provides a schematic framework that defines a comprehensive library of generic and reusable models of domain knowledge in terms of generic tasks and meta-domains. It also provides useful methods and guidelines for software reuse, emphasizing the concept of abstraction process in a designer s mind. This paper firstly gives a brief overview of fundamentals of software reuse. Next, it explains the foundation of domain theory and discusses its applicability to software reuse. In particular, the taxonomy of meta-domains and the types of generic tasks are described in more detail. Finally, the future research framework, which primarily addresses the problem of how to apply the domain theory to various work domains, is proposed.

1. 서론 및 관련 연구

소프트웨어의 재사용은 소프트웨어 개발의 생산성 및 품질 향상을 동시에 충족시켜줄 수 있는 유용한 방법으로 여겨지며 그 동안 많은 연구의 대상이 되어 왔다 [1,4]. 소프트웨어 재사용이란 기존에 개발된 소프트웨어 관련 산출물을 동일하

거나 서로 다른 작업 영역 혹은 다른 조직간에 그대로 다시 사용하거나 일부 수정 및 보완 후에 사용하는 것을 의미한다 [4]. 넓은 의미의 소프트웨어 재사용은 프로그램 및 일부 모듈뿐만 아니라 분석 및 설계 지식, 문서, 기법, 절차 등의 모든 소프트웨어 자산 (assets) 을 모두 대상으로 한다.

점점 복잡해지고 대형화되는 소프트웨어의 효율적 설계 및 증가하는 유지보수 비용 등을 고려할 때 기존의 것을 활용하고자 하는 재사용의 개념이 관심의 대상이 되는 것은 자연스러운 현상일 것이다. Carma는 소프트웨어를 비교 분석했을 때 일반적으로 60 ~ 70%의 시스템 기능이 공통적이라는 의미 있는 결과를 보고하고 있다 [8]. 이는 재사용을 통해 소프트웨어 개발의 생산성이 상당부분 향상될 수 있음을 암시한다고 볼 수 있다.

일반적으로 소프트웨어 재사용을 위한 프로세스는 두 개의 문제로 나눌 수 있다 [5-7]. 하나는 재사용 가능한 소프트웨어 자산을 만드는 일이고 (design for reuse) 다른 하나는 사용 가능한 소프트웨어 자산을 이용해 새로운 시스템을 개발하는 일이다 (design by reuse) [6]. 전자는 소프트웨어가 사용될 응용 영역 (application domain)의 지식 내지는 요구사항으로부터 의미 있는 단위의 재사용 모듈 내지는 단위 지식 (이 논문에서는 이들을 컴포넌트라 함)을 파악해서 이 들을 구조화하고 분류한 후에 컴포넌트 라이브러리 혹은 저장소와 같은 곳에 기록, 수정 및 보완하는 과정을 거치게 된다. 반면에 후자는 응용 영역에 대한 요구사항을 파악한 후에 기존에 개발되어 저장소에 있는 컴포넌트에서 사용할 수 있는 것을 추출한 후에 그 컴포넌트를 이해하고 적용하는 과정을 거쳐 소프트웨어 개발을 하게 된다. Design for reuse를 위해 현재 주로 사용되는 기술적 개념이 객체지향 설계, 설계 패턴 및 소프트웨어 프레임워크 등이다. 그런데 두 문제의 경우 모두 응용 영역의 요구사항 및 지식을 아키텍쳐, 기능 및 구조에 관점에서 공통성과 가변성을 염두에 두고 파악 및 분석 하는 것이 핵심 요소임을 알 수 있다.

현재까지 재사용 향상을 위해 여러 각도에서 다양한 연구가 이루어져 왔다. Frakes와 Terry는 기존의 재사용 연구들을 6가지의 관점 (development scope, modification, approach, domain scope, management, and reused entity) 에서 분류하였다 [10]. 각각의 관점은 그 관점에 따라 재사용 연구를 분류할 수 있는 기준을 갖고 있다. 예로 modification의 경우 재사용 가능한 자산이 재사용을 위해 어느 정도 변경되어야 하는가를 의미하는데 세 가지로 분류될 수 있다: white-box, black-box, and adaptive. 첫 번째 방법은 재사용 가능한 컴포넌트를 라이브러리에서 파악하고 그 컴포넌트의 기능 및 내부 구조를 이해한 후에 추가적으로 코딩을 하는 형태로 변경을 해서 소프트웨어를 개발해 가는 방식이다. 두 번째의 경우는 기존의 컴포넌트를 별다른 수정 없이 플러그인 형태로 그대로 사용하는 방식이다. 마지막 방법은 상황의 변화에 맞게 사용할 수 있도록 design for reuse 과정에서 유연한 구조의 컴포넌트를 개발한 후에 design by reuse 과정에서 파라미터 등을 변경해 재사용해가는 방식이다.

한편 재사용을 설계 측면에서의 중요한 두 가지 기준이면서 개발자의 인지적 문제해결 관점에서 흥미 있는 속성이 추상화 (abstraction)와 상세화 (granularity)이다. 추상화는 문제의 목표에 직접적으로 관련 있으면서 핵심이 되는 내용만을 표현하는 것이며 상세화는 문제나 상황에 특별하고 세부적인 내용을 구체적으로 표현하는 것이다. 추상화가 높을수록 재사용성은 높아지지만 개발자 입장에서의 사용성은 떨어지고 상세화의 경우는 반대로 생각할 수 있다. 따라서 재사용성과 사용성의 균형은 추상화와 상세화의 적절한 균형으로 이루어진다. 또한 이 두 가지의 개념은 개발자가 소프트웨어를 개발할 때의 문제해결 과정에서 인지심리학적으로 많은 관심의 대상이 되어 왔다 [2,9].

기존의 연구들이 소프트웨어 재사용 향상을 위한 방법론, 기법, 도구 등의 측면에서 많은 기여를 하였으나 재사용이 보다 체계적이고 일반화된 형태로 이루어지기 위해서는 몇 가지 측면에서 더 깊이 있는 연구가 필요하다. 먼저 재사용을 위해 개발된 컴포넌트를 효율적으로 저장하고 검색할 수 있도록 해줄 수 있는 분류 체계 및 방법에 대한 연구가 더 필요할 것으로 보인다. 또한 재사용 높은 컴포넌트를 도출하기 위해 응용 영역지식의 기능적 추상화 및 분할을 어떻게 할 것인가 (추상화와 상세화의 균형)에 대한 문제가 중요해지는데 이 문제를 위한 개념적 틀도 필요할 것으로 보인다. 그러나 무엇보다도 재사용 향상을 위해서는 응용 영역의 지식 (domain knowledge)을 재사용 관점에서 효과적으로 분석하기 위한 일종의 프레임워크 내지는 모델이 필요하다. 이 모델이 유용하게 활용되기 위해서는 인간으로서의 개발자가 시스템을 바라보는 관점에서 접근해가는 것이 바람직할 것으로 판단된다. 이는 인간의 사물 인식, 문제해결 및 추론 등을 다루는 인지심리학 및 인간과 artifact간의 상호작용을 다루는 인간-컴퓨터 상호작용의 내용이 유용한 배경지식이 될 수 있음을 의미한다. 이러한 문제들을 근본적이면서 원리적인 입장에서 접근할 수 있도록 다양한 학문의 개념을 빌려서 나온 개념이 이 논문에서 소개하고자 하는 domain theory이다 [2-3].

2. Domain Theory의 개요 및 적용

Domain theory는 폭 넓게 재사용될 수 있는 응용 클래스의 모델을 위한 지식 표현에 대한 본체론 (knowledge representation ontology)에 관한 이론이며 다양한 소프트웨어 기반의 시스템 개발에 적용될 수 있는 모델의 라이브러리를 제공한다. 이 이론은 요구사항의 명세 수준에서 재사용 가능한 지식을 파악하고 형식화하는 것을 목표로 한다. 한 응용 영역의 전문 소프트웨어 개발자의 장기기억장소에 내재되어 있는 유용한 지식을 명시적으로 표현해 재사용하고자 하는 것이 domain theory 연구가 시작되었던 동기며 목표였다. 두 가지 핵심 관점은 추상화 이론과 반복적으로 발생하는 전문지식의 이론이다. 이 연구는 요건공학 및 인간-컴퓨터 상호작용을 주로 연구해오던 Sutcliffe를 비롯한 그의 연구 동료들에 의해 이루어져 왔다 [3].

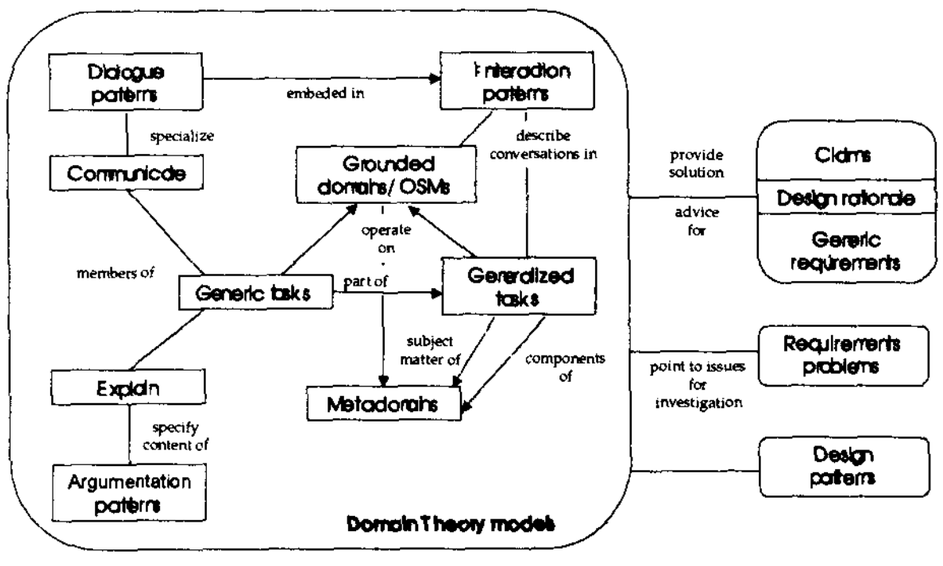

그림 1. Domain theory의 구성 요소들 및 관계

그림 1은 domain theory를 구성하는 요소들과 그들 간의 관련성을 보여주고 있다. 우선 domain theory는 3개의 큰 요소를 갖고 있다. 첫 째는 generic model의 의미를 정의하기 위한 metaschema이며, 두 번째는 광범위한 응용 영역에 사용될 수 있는 generic model의 모음이다. 마지막 세 번째는 이 모델들을 활용한 재사용에서 모델의 검색 및 매칭에 대한 프로세스이다. 이 중에서 핵심은 generic model들의 집합으로 이 논문에서도 이를 중심으로 설명하도록 하겠다.

Generic model은 실세계에 존재하는 문제 혹은 지식들을 비교적 높은 추상화 수준에서 설명하려고 하는 모델이다. 이는 수 년간 여러 응용 영역의 전문 개발자들과의 인터뷰를 위주로 해서 개발되어졌다. Generic model은 여러 모델들로 다시 구분되어진다. 첫 번째 모델인 grounded domains은 하나 혹은 여러 직무 (task)와 연관되어 있으면서 하나의 목표 (purpose)를 달성하기

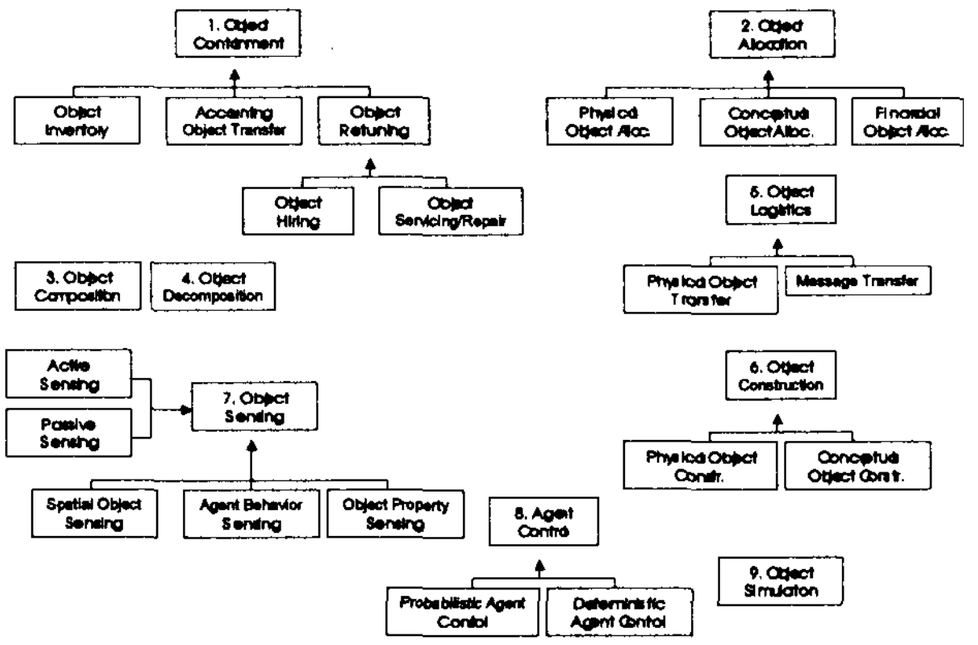

그림 2. Family of OSM (Object Systems Model)

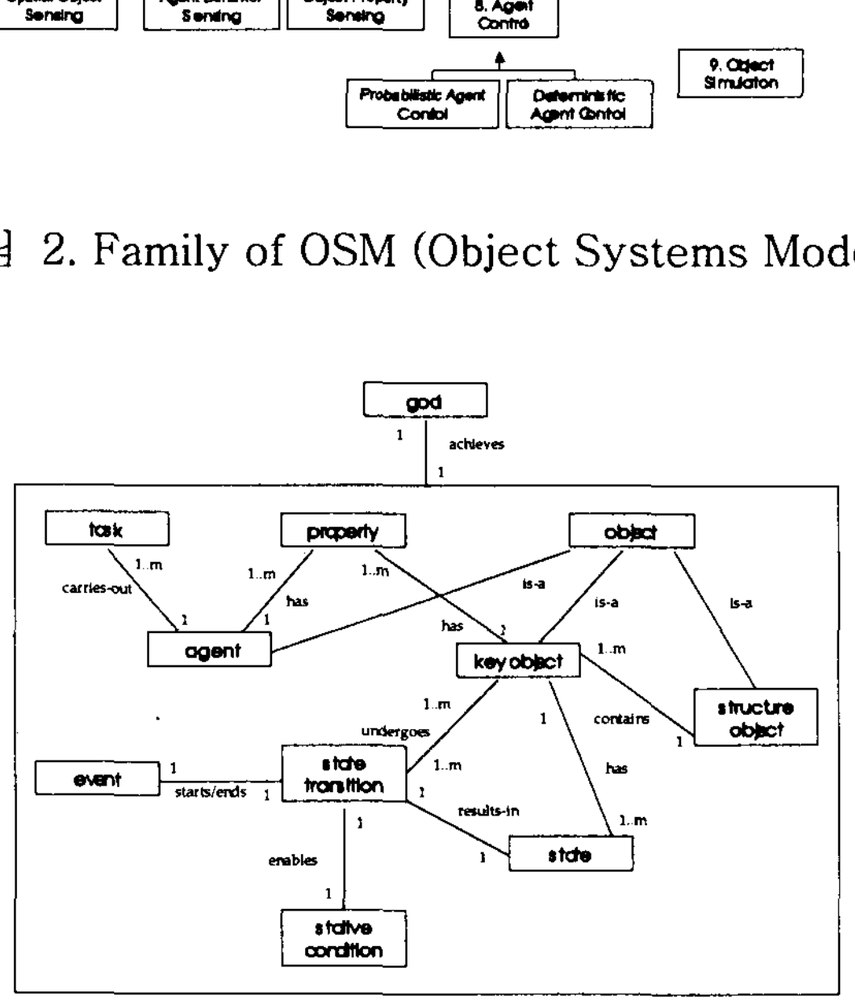

그림 3. Schema of knowledge type

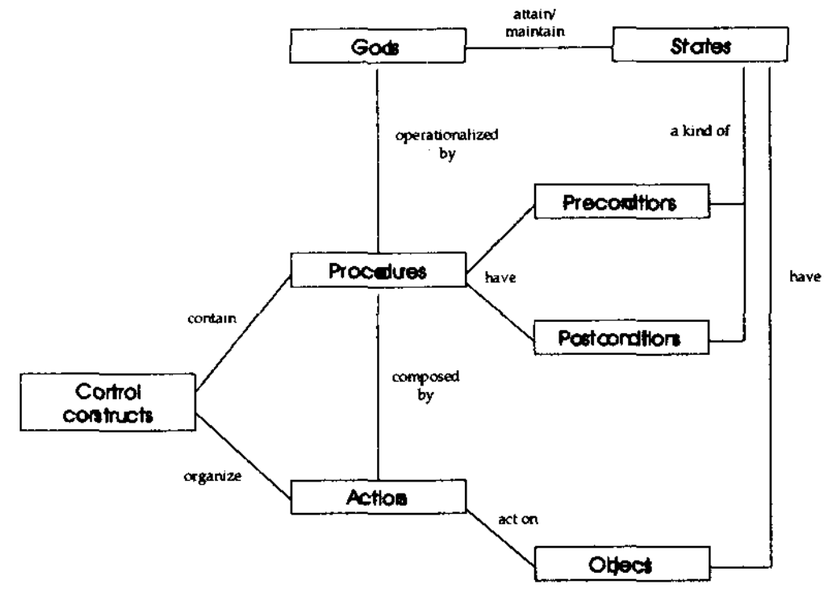

그림 4. Schema of task knowledge

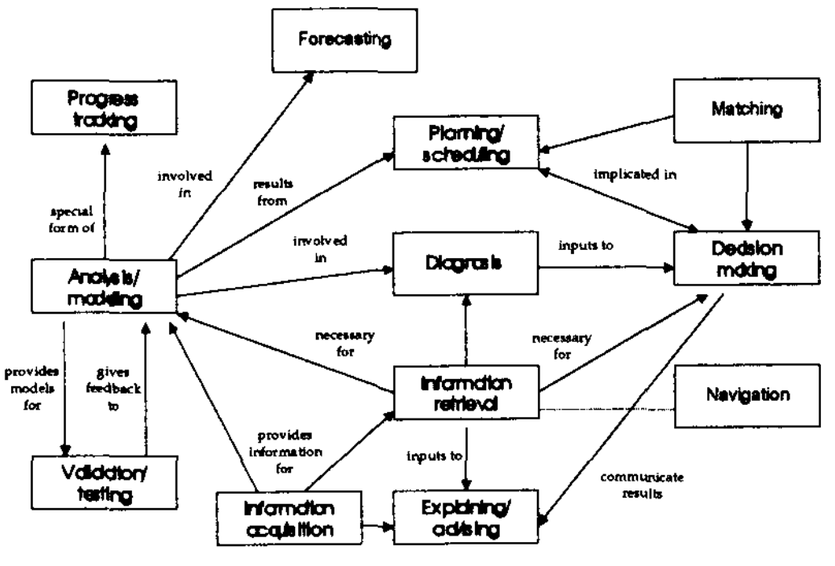

그림 5. Generalized task들간의 관련성

위한 것이다. 즉, grounded domains은 시스템 목적을 달성하는 관측 가능한 객체, 행위 및 직무로 구성되면서 물리적인 존재를 갖는 실세계 문제의 추상화라고 정의 내릴 수 있다. 하나의 grounded domain 예로 hiring application을 생각할 수 있는데 이는 도서관, 자동차 대여, 비디오 대여 등에 공통적으로 관련되는 개념이다. Grounded domain은 다시 두 개의 하위 클래스로 나뉘게 된다. 첫 번째는 OSM (Object System Model)으로 일련의 목표를 달성하기 위해 상호 협력하는 객체와 그들의 행위로 표현되는 핵심 트랜잭션을 모델화한 것이다. 그림 2는 domain theory에서 제공하는 OSM family이다. 각 OSM은 네 개의 속성으로 기술이 된다: goal state, agents, key object, and transitions. 일례로 object containment는 다음과 같이 기술될 수 있다.

Goal state: to satisfy all resource requests by supply of resource objects.
Agents: client (requester), supplier.
Key object: resource.
Transitions: request, supply transfer.

두 번째는 ISM (Information System Model)으로 OSM에 관한 정보를 제공하고 보고하는 등의 일련의 프로세스에 관한 모델이다.

그림 3은 일종의 metaschema로 OSM의 근본적인 구성요소를 형성하는 11개의 지식 형태들과 그들 간의 관련성을 ER 다이어그램 형태로 표현한 것이다. 응용 영역의 지식을 OSM으로 표현

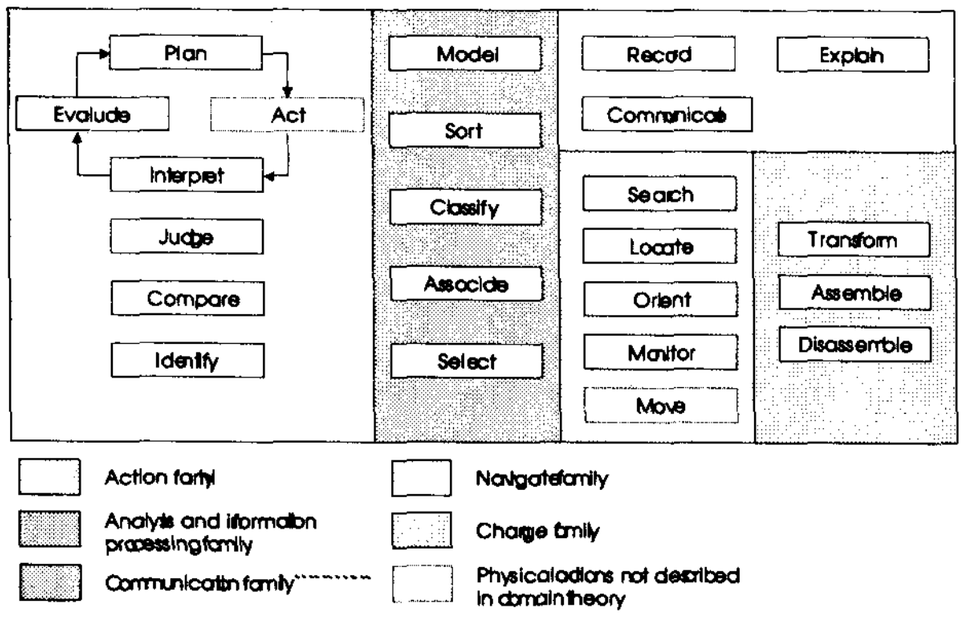

그림 6. Generic task의 종류

하기 위해 해당 시스템 혹은 직무의 goal, object, agent, state, event, stative condition 등을 파악할 필요가 있음을 알 수 있다. 또한 그림 4는 domain theory에서 가정하고 있는 인간이 수행하는 직무에 대한 schema 구조를 보여준다. 하나의 직무는 goal, pre and post condition, action, agent, object 등을 모두 포함하는 개념임을 알 수 있다.

두 번째의 generic model은 metadmains이다. 실세계를 통제하고 이해하는 시스템의 상위 수준의 아키텍쳐로 생각할 수 있다. 많은 인간의 행위는 물리적으로 관측 가능한 행위에 의해 특징지어지지 않는다. 이러한 현상을 다루기 위해 metadomains의 개념이 필요한데 이는 grounded domain에 적용되는 generalized task의 개념으로 구성된다. Generalized task는 인간이 소프트웨어를 기반으로 한 시스템에서 일을 할 때의 직무들을 추상화 하여 대표할 수 있는 몇 개의 직무로 구분한 것이다. 이런 의미에서 metadomains은 하나의 응용 영역에서 grounded domain과 직무에 작용하면서 knowledge와 artifact 등을 변환하기 위한 복잡한 goal을 달성하는 인지적 직무에 의해 특징 지어진다고 할 수 있다. 대표적인 metadomains은 education, management & governance, research, and design이다.

그림 5는 domain theory에서 설명하는 12개의 generalized task 및 그들 간의 관련성을 보여준다. 그림에서 나타나는 직무들은 모든 응용 영역에서 나타날 수 있는 것으로 모두 각각의 목적, 기능, 정보의 입출력, 목적을 달성하기 위한 세부 행위들 및 이들간의 계층적 구조 등을 갖고 있다

또한 generalized task는 세부적인 인지적 직무 혹은 행위로 다시 구성되어지는데 이들이 세 번째의 generic model인 generic task이다. 그림 6은 23개의 generic task 형태를 보여준다. Generic task는 postcondition 상태에 기술된 goal을 달성하기 위해 수행되는 하나의 단위 행위로 정의 할 수 있다. 이 task들은 크게 5개의 그룹으로 나뉘어지게 된다. 일례로 navigation family에 속하는 generic task에는 search, locate, orient, monitor, move가 포함된다. 앞에서 설명했듯이 하나의 generalized task는 다른 generalized task와 정보처리 관점에서 관련을 맺으면서 다수의 generic task를 포함하고 있다. 일례로 diagnosis는 identify, interpret, evaluate, classify, compose, select 등을 포함한다.

이 외에 domain theory에는 세부적인 user interface 및 dialogue 설계에 도움이 되면서 generic task의 communication family를 보다 체계화한 generic dialogue pattern 및 generic argumentation pattern이 제공된다. 또 설계 과정에서 design component를 설계할 때 발생하는 design decision making에 대한 정보를 제공하는 일종의 design rationale인 Claims이 제공이 되어 추후 design reuse에 도움이 되도록 도와준다.

3. 토의 및 결론

이 논문에서는 소프트웨어 재사용 연구에 대한 새로운 접근법으로 domain theory를 소개하였다. Domain theory는 재사용 향상을 위해서는 응용 영역에서의 지식을 추상화와 상세화의 적절한 균형 속에 어떻게 표현하는가가 핵심적인 문제라

는 가정 하에 응용 영역에서 일반적으로 사용될 수 있는 모델을 파악하기 위한 지식 표현의 틀과 실제의 유용한 모델들을 제공한다. 이를 구성하는 핵심 요소로는 개념 형성에 관련된 grounded domain model 및 metamodel, 소프트웨어 기반 시스템에서 인간의 행위 혹은 직무에 관련된 generic task model, generalized task model, generic dialogue model이 있다. Domain theory는 비교적 높은 추상화 수준의 모델들을 제공하고 있어 개발자들은 응용 영역의 특성에 영향을 받지 않고 적용할 수 있다. 또한 이 이론을 구성하는 모델과 전체 구조가 다른 기존의 재사용 이론들과는 다르게 문제해결 및 추론과 같은 인지심리학적 입장에서 개발되었기 때문에 개발자들에게 인간 중심적인 소프트웨어 개발의 관점을 부여할 수 있는 장점을 가지고 있다. 특히 재사용 과정에 핵심인 소프트웨어 개발자의 문제에 대한 추상화 과정을 심리학적 관점에서 재조명하고 이를 모델에 반영한 점은 매우 특징적인 부분이다. 이는 그대로 재사용 과정에서 domain theory가 제공하는 모델을 이용한 추상화 중심의 재사용이라는 측면과 연결된다.

그러나 domain theory가 현재까지 개념 위주로 연구가 이루어져 실제로 개발자들이 활용하기에는 쉽지 않을 것으로 예상된다. 이는 domain theory가 응용 영역의 지식 구조 및 형태, 직무의 구조 및 기능 등에 대해 개발자가 모델링 하는 것을 인지심리학 및 인간-컴퓨터 상호작용 등의 관점에서 접근했기 때문이다. 또한 domain theory를 구성하는 요소들이 상위의 추상화 수준을 설명하는 것들이어서 적용이 어려울 수 있다. 이는 현재 domain theory의 단점이 될 수 있는데 domain theory의 적용성을 높이기 위해 많은 사례연구 및 실제 적용을 통한 적용 지침이 이루어져야 할 것이다.

[참고문헌]

[1] A. Mili, S. Yacoub, E. Addy, and H. Mili (1999), Toward an engineering discipline of software reuse , IEEE Software, Vol. 16(5),pp. 22-31.

[2] A. Sutcliffe (2000), Domain analysis for software reuse , The Journal of Systems and Software, Vol. 50, pp. 175-199.

[3] A. Sutcliffe (2002), The Domain Theory: Patterns for Knowledge and Software Reuse, Lawrence Erlbaum Associates, Publishers London.

[4] C.W. Krueger (1992), Software reuse , ACM Computing Surveys, Vol. 24(2), pp. 131-183.

[5] E.R. Comer (1990), Domain analysis: a systems approach to software reuse , in Proceedings of the 9th Digital Avionics Systems Conference, pp. 224 229, Virginia Beach, VA, USA, October.

[6] H. Mili, F. Mili, and A. Mili (1995), Reusing software: issues and research directions , IEEE Trans. Software Engineering, Vol. 21(6), pp. 528-562.

[7] J.E. Cardow (1989), Issues on software reuse , in Proceedings of the IEEE Aerospace and Electronics Conference, pp. 564-570.

[8] M. Carma (1997), Software Reuse Techniques: Adding Reuse to the Systems Development Process, Prentice Hall Inc.

[9] J.L. Cybulski, Reuse in the eye of its beholder: cognitive factors in software reuse , in Proceedings of Sixth Australian Conference on Computer-Human Interaction, pp. 228-235.

[10] W.B. Frakes and C. Terry (1996), Software reuse: metrics and models , ACM Computing Surveys, Vol. 28(2), pp. 415-435.

[11] W. Lam, B. Whittel, J. McDermid, and S. Wilson (1996), An integrated approach to domain analysis and reuse for engineering complex systems , IEEE Symposium and Workshop on Engineering of Computer Based Systems (ECBS 96), pp. 102-109.