Advanced Science and Technology Letters
Vol.63 (NGCIT 2014), pp.118-121
http://dx.doi.org/10.14257/astl.2014.63.24

# A Study of Adaptive Software Task Model

Jinhong Kim[1], Leesang Cho[2]

Department of Computer Engineering, Hansung University
Department of Mechanical Systems Engineering, Hansung University
{ppome815, jinhkm6}@naver.com

**Abstract.** Self-adapting is a basic element of Autonomic Computing [1]. A system called adaptive system should provide the ability to adapt the external and internal environment changes. As a critical part in self-adaptive system, adaptive software is a function of how well the software supports and adapts the changing environment, so it is important for developers to acquire the knowledge of user's roles, task will affect software changing. In this paper, we focus on the idea of task model as a representation to support software adaptation, including task model compression, task model construction, using task model to support software adaptation, and users interact with their computing environments in terms of high-level tasks.

**Keywords:** Autonomic Computing, Self-Adaptive System, Adaptive Software

## 1 Introduction

In an increasing number of domains software is expected to do more for us, in more situations, in more complex environments in which there are more users, more connected systems and more interactions among systems and more resources and more goals [1,2]. All these take adaptation an important requirement in a software system. Adaptability in most applications has been implemented at the component / module level. Accordingly, our approach is to investigate how context knowledge can be represented, and how to build presents the point of task model supporting for software adaptation. Many engineering solutions are inspired by nature. One of these inspirations is adaptation; that is, the way living organisms change their structure and/or behavior in order to retain or achieve a better compatibility with their environment. Adaptation can occur either instantly or over time. Organisms may adapt themselves to changes in their environment as these changes take place. This type of alteration is limited to an organism's adaptiveness; that is, the degree to which an organism is able to live and reproduce in its environment. Thermoregulation and learning are examples of these kinds of adaptation in living organisms [3,4]. On the other hand, evolution is a change process that happens continuously over time throughout generations. The result of evolution is a new individual that carries most of the properties of its base, to the extent that it remains in the same class as its base. Evolution happens gradually, and the resulting newer generations are expected to be better individuals in terms of functionality and quality. Therefore, adaptation refers to both the current organism being adapted and to the evolutionary process that leads to the adapted generations. Evolution can result in adaptation, but not all evolution changes adapt an organism to its surroundings, even though evolution may increase adaptiveness. This phenomenon is known as an organism's adaptive trait; i.e., an aspect of the developmental pattern of the organism that enables or enhances the probability of that organism's survival and reproduction

## 2 Conceptual Modeling

The use of simplified abstract models through finding generic solutions and decisions is a common general strategy for dealing with complexity in software engineering. Models specify abstractions of real-world phenomena, the abstraction level can vary and depends on the application. Models are used for a variety of purposes and are used during different stages of the software life-cycle [5]. For example, in software construction, abstract models serve as the basis for software design and development. During software evolution, appropriate models of software can assist in understanding the software, and serve as the basis for model transformations that result in the generation of the evolved software. One of the main challenges in engineering self-adaptive software systems is defining the problem space, and ultimately, setting the scope of a software solution based on an understanding of what will be built, as well as domain knowledge and a conceptualization of how information will follow through the solution. Moreover, maintaining self-adaptive software or migrating non-adaptive software systems to behave in an adaptive manner demand a comprehensive understanding of the internals of the software at hand. Such an understanding can be obtained with the help of appropriate domain-specific models of the software. Difficulties arise especially in the area of model-centric runtime adaptation, where models of software need to be generated, manipulated, and managed at runtime. As discussed in the following chapter, providing conceptual models of the problem space of self-adaptive software is essential to understanding the required solution, and how adaptation changes must be implemented and positioned in self-adaptive software.

### 2.1 Software Adaptive Architecture

This system structure includes three parts as shown Figure 1. (1) Task Module (TM) is responsible for setting the overall system objectives that determines the quality of service requirements for the tasks. In addition, this module focuses on task model, which is responsible for managing/describing user’s intention or running task, task representation, services that task required and the desired performance profile for the application service. (2) Architecture Module (AM) is responsible for interpreting system observations, integrating adaptation rules, making adaptation plans, evaluating the plan and finally propagating these to Implementation Module, it also includes architecture model and model manager. (3) Implementation Module (IM) that consists of the system and application is responsible for system running, this module is also responsible for monitoring running system and sending all the information to TM and AM.

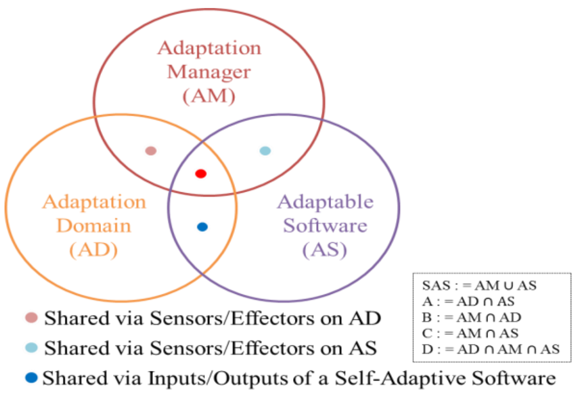


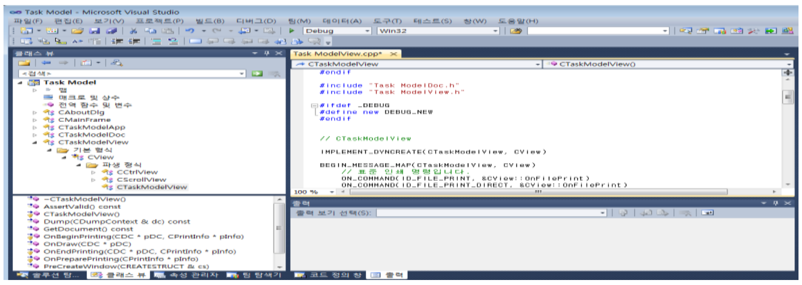
**Fig. 1**. Phenomenon Sets in a Self-adaptive Software System.

### 2.2 Adaptation Requirements

The requirements of a system describe what is expected from the system in terms of its effects on its environment. Functional requirements specify the desired changes to those phenomena of the AD that are shared with the system through input and output interfaces. The system observes a change in the AD's phenomena via an input interface, and in turn changes some shared phenomena via an output interface. In contrast, non-functional requirements specify changes to phenomena that are either (i) part of the AD, but are not shared phenomena and are changed as a side-effect of the application's behavior (e.g., performance requirements and quality of service requirements), or (ii) part of other domains (e.g., business domain, test domain, and process domain). Based on these requirements, engineers analyze the boundaries of a software system with its environment and develop specifications. Specifications describe the system to be built, which must fit and interact with its environment.

## 3 Main System

The Task Model includes two parts: task specification and task adaptation. Task specification is a representation of user's tasks, it includes user's intention, task requirement, task constraints, task procedures and task actions and objects, that gives a profile description of task. Task adaptation is an ability to adapt to changing task requirements and changing work context, all the changes will reflect in task specification. We can describe the task as follow together with the five arguments in task model and we showed implementation as following Figure 2.


Task Model - Microsoft Visual Studio
파일(F) 편집(E) 보기(V) 프로젝트(P) 빌드(B) 디버그(D) 팀(M) 데이터(A) 도구(T) 테스트(S) 창(W) 도움말(H)
Debug
Win32
클래스 뷰
<검색>
Task ModelView.cpp*
CTaskModelView
CTaskModelView()
100 %
출력
출력 보기 선택(S):
솔루션 탐... 클래스 뷰 속성 관리자 팀 탐색기 코드 정의 창 출력

```
Task Model
    맵
    매크로 및 상수
    전역 함수 및 변수
    CAboutDlg
    CMainFrame
    CTaskModelApp
    CTaskModelDoc
    CTaskModelView
        기본 형식
            CView
                파생 형식
                    CCtrlView
                    CScrollView
                    CTaskModelView
```

```
~CTaskModelView()
AssertValid() const
CTaskModelView()
Dump(CDumpContext & dc) const
GetDocument() const
OnBeginPrinting(CDC * pDC, CPrintInfo * pInfo)
OnDraw(CDC * pDC)
OnEndPrinting(CDC * pDC, CPrintInfo * pInfo)
OnPreparePrinting(CPrintInfo * pInfo)
PreCreateWindow(CREATESTRUCT & cs)
```

```
#endif

#include "Task ModelDoc.h"
#include "Task ModelView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif


// CTaskModelView

IMPLEMENT_DYNCREATE(CTaskModelView, CView)

BEGIN_MESSAGE_MAP(CTaskModelView, CView)
    // 표준 인쇄 명령입니다.
    ON_COMMAND(ID_FILE_PRINT, &CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, &CView::OnFilePrint)
```

**Fig. 2**. Implementation of Task Model.

## 4 Conclusion

The aims of the research in which we are engaged are ultimately to develop architecture, methods and tools to the ability of self-adaptation in software system by MFC Component design. Task model support for software adaptation is a part of work in our ongoing project that offers an approach involving external context changes in self-adaptive can be derived and incorporated in system adaptation. This paper has shown how task knowledge encapsulated in task model and discussed the basic composition of task model. Moreover, we'll actually approach this project to need to develop more explicit mechanisms to provide 'Adaptiveness' for software system.

**Acknowledgments.** This research was financially supported by Hansung University.

## References

1. Basili, V. R., Briand, L. C., and Thomas, W. M., 1994a, "Domain Analysis for the reuse of software Development Experience," Proceedings of the 19th Annual Software Engineering Worshop, NASA/GSFC, Greenbelt, Maryland. December. Pp. 1-14.
2. Lemus-Olalde, C., and Belkhouche, B., 1996a, "Multiple View Analysis of Designs," to appear in Viewpoints of Software Development Workshop, Symposium on Foundations of Software Engineering. ACM SIGSOFT'96. San Francisco, CA. October 14-18.
3. Luckham, D., Kenney, J. J., Augustin, L. M., Vera, J., Bryan, D., and Mann, W., 1995a, "Specification and Analysis of System Architecture Using Rapide," IEEE Transactions on Software Engineering, 21(4), April. pp. 336-353.
4. Mehdi Amoui, Mahdi Derakhshanmanesh, Jurgen bert, and Ladan Tahvildari. Software evolution towards model-centric runtime adaptivity. In Proceedings of the 15th European Conference on Software Maintenance and Reengineering, pages 89-92, 2011.
5. Mehdi Amoui, Mazeiar Salehie, Siavash Mirarab, and Ladan Tahvildari. Adaptive action selection in autonomic software using reinforcement learning. Autonomic and Autonomous Systems, International Conference on, 0:175-181, 2008.