목표와 시나리오를 이용한 적응형 S/W 개발 방안

김동선 °, 박수용
서강대학교 컴퓨터학과
darkrsw@selab.sogang.ac.kr, sypark@ccs.sogang.ac.kr

An Approach to Self-Adaptive Software using Goals and Scenarios

Dong-sun Kim °, Soo-yong Park
Dept. of Computer Science, Sogang University

요 약

소프트웨어가 수행되는 환경은 점점 복잡해지고, 예측이 어려워지고 있지만, 소프트웨어의 자율성과 견고성을 달성하는 것은 여전히 어려운 문제 중에 하나이다. 적응형 소프트웨어는 소프트웨어의 복잡성을 해결하고 자율성과 견고성을 달성하는데 기여할 새로운 소프트웨어 개발 패러다임으로 떠오르고 있다. 적응형 소프트웨어는 환경과 자신의 상태를 인지하고 현재의 성능을 향상시킬 방법을 스스로 판단하여 소프트웨어의 구조 및 행위를 변경할 수 있는 소프트웨어이다. 본 논문에서는 적응형 소프트웨어의 개발 초기단계에서 요구사항 분석 단계에서 추출된 목표와 시나리오를 이용하여 어떻게 적응형 소프트웨어의 각 기능을 정의하는 방안을 제시한다.

1. 서 론

최근 소프트웨어 공학이 직면하고 있는 문제는 크게 세가지로 분류될 수 있다. 소프트웨어의 복잡도(complexity) 증가, 소프트웨어의 자율성(autonomy) 부여, 소프트웨어의 견고성(robustness) 달성이 세가지 문제로서 이 들은 각기 따로 해결할 수 있는 문제들이 아니라 서로 연관되어 있어 해결이 어렵다[1]. 적응형 소프트웨어는 위의 문제들을 해결할 수 있는 새로운 패러다임으로 떠오르고 있다.

적응형 소프트웨어란 '자신의 행위를 평가하고 있다가, 원래 의도했던 목적으로 잘 수행하지 못하거나 더 나은 성능을 기대할 수 있다고 판단될 경우에 자신의 행위를 변경하는 소프트웨어[2]' 이다. 수행을 멈추지 않고 실행시간(run-time)에 자신의 구조 및 행위를 변경할 수 있으므로 복잡하고 변하기 쉬운 요구사항을 수용할 수 있어 복잡도 문제를 해결할 수 있다. 스스로 행위의 결과를 평가하고 좀 더 나은 결과를 얻기 위해 행위를 변경할 수 있어 소프트웨어에 자율성을 부여할 수 있다. 지속적으로 더 나은 결과를 얻기 위해 행위를 변경하므로 다양하고 변화하는 환경에서도 견고성을 달성할 수 있다.

적응형 소프트웨어를 구성하기 위해서는 크게 세가지 기능이 필요하다. 첫째, 적응형 소프트웨어가 실행되고 실행도중 수행한 각 행위들의 결과를 인지할 '환경인지' 기능이 필요하다. 둘째, 환경으로 받은 결과를 평가하고 결과가 기대치에 못 미치거나, 더 나은 성능을 기대할 수 있을 때 소프트웨어의 구조 및 행위 변경을 결정하는 '적응판단' 기능이 필요하다. 셋째, 구조 및 행위의 변경이 필요하다고 판단되었을 때 실제로 구조 및 행위의 변경을 수행할 '동적재구성' 기능이 필요하다. 본 논문에서는 목표(goal)와 시나리오(scenario)로 모델링된 요구사항으로부터 적응형 소프트웨어를 구성하는 세가지 기능을 구축하는 방안을 제안한다.

2장에서는 기존의 강화학습 이론을 기초로 적응형 소프트웨어의 구조를 서술하고 관련연구를 소개하여 본 연구의 필요성을 제시한다. 3장에서는 실제로 목표와 시나리오를 이용하여 2장에서 제시한 함수 및 집합을 정의하는 방법을 제안한다. 4장은 결론과 앞으로 수행해야 하는 향후 연구를 서술한다.

2. 연구 배경 및 관련연구

적응형 소프트웨어를 구성하는 세가지 기능인 환경인지, 적응판단, 동적재구성은 서로 독립적으로 수행된다. 다음의 적응형 소프트웨어에 관한 기술은 강화학습(Reinforcement Learning)의 모델이 없는 경우[3]을 확장한 것이다.

환경인지는 물리적인 외부환경 또는 외부 시스템으로부터 센서가 인지한 데이터를 기반으로 적응형 소프트웨어에 관련된 상태를 기술하는 기능이다. 외부환경은 시간의 흐름에 따라 상태가 변화하며 환경인지 기능이 인지하는 상태도 외부환경에 따라 변화한다. 외부환경의 변화는 다음과 같이 전이함수(Transition Function) T 로 정의된다.

$$T : S \times A \times F \rightarrow S$$

S 는 외부상태가 가질 수 있는 상태의 전체 집합을 나타낸다. A 는 적응형 소프트웨어가 현재 상태에서 취할 수 있는 적응행위(actions)을 나타낸다. F 는 적응형 소프트웨어 이외의 요인으로 현재 환경의 상태에 영향을 주는 외부요인을 나타낸다. 외부환경은 현재상태와 적응형 소프트웨어 취한 적응행위, 그리고 외부요인에 의해서 다음 상태로 전이한다. 외부환경의 전이는 결정적(deterministic)일 수도, 확률적(stochastic)일 수도 있다. 불행하게도 적응형

소프트웨어는 외부환경의 모든 요소를 고려하려 행동할 수 없다. 적응형 소프트웨어는 관심의 대상이 되는 환경의 요소만을 관찰하여 현재 상태를 기술해야 한다. 외부환경의 관찰은 다음과 같이 관찰함수(Observation Function) O로 정의된다.

$$O: S \times A \rightarrow Z$$

S는 외부환경이 가질 수 있는 가능한 상태 집합이며, A는 적응형 소프트웨어가 취할 수 있는 적응행위의 집합이고, 적응형 소프트웨어가 Z는 관찰할 수 있는 상태의 집합이다. 환경인지 기능에서 실제로 적응형 소프트웨어에서 구현되는 부분은 집합 A와 Z이다. 기존의 환경인지에 대한 연구[4]는 환경을 묘사하는 방법에 대해서만 다루었다. 어떤 요구사항으로부터 적응형 소프트웨어의 적응행위가 도출되어야 하는지, 어떤 상태들을 인지해야 하는지에 대한 지침은 존재하지 않았다. 요구사항으로부터 적응행위의 집합을 도출하기 위해서는 해당 적응형 소프트웨어가 달성하고자 하는 목적이 무엇인지, 가능한 시나리오는 무엇인지 알아야 한다. 해당 목표에 대해서 순기능을 하는 적응행위를 시나리오에서 식별하여 적응행위의 집합을 정의하고, 해당 적응행위가 이루어질 수 있는 상황의 집합을 시나리오로부터 식별해야 한다.

적응판단 기능은 환경인지 기능으로부터 현재상태 $z \in Z$와 z에서 수행 가능한 적응행위의 집합 $A(z)$를 받아 그 중에서 가장 최적의 결과를 낼 수 있는 적응행위 $a \in A$를 선택하는 기능을 담당한다. 적응형 소프트웨어의 적응행위의 결과는 환경과 소프트웨어 자신으로부터 모두 받는다. 이들은 다음과 같이 보상함수 R과 P로 정의된다.

$$R: Z \times A \rightarrow \Gamma$$
$$P: S_S \times A \rightarrow \rho$$

보상함수 R은 현재 관찰한 상태 $z \in Z$에서 적응형 소프트웨어가 적응행위 $a \in A$를 취했을 때 환경으로부터 보상 $r \in \Gamma$을 측정한다. 소프트웨어 내부의 성능을 측정하는 보상함수 P는 현재 소프트웨어의 상태 $x \in S_S$에서 적응행위 $a \in A$를 취했을 때 자신의 성능 $p \in \rho$를 측정한다. 적응형 소프트웨어를 개발할 때 적응판단 기능에서 구현되어야 하는 부분은 보상함수 R의 식과 보상 값의 집합 Γ, 소프트웨어의 상태 집합 S_S, 소프트웨어의 성능 집합 ρ이다. 기존의 순차 결정 구조에 대한 연구들[3](Q-Learning, Temporal Difference Learning 등)은 이론적인 결정구조만을 다루었을 뿐 어떤 요구사항으로부터 보상함수가 구성되어야 하는지에 대한 연구는 미비하였다.

동적재구성 기능은 적응판단 기능이 결정한 적응행위를 기초로 구조 변경 및 재구성, 행위변경 등을 수행하는 기능을 담당한다. 적응행위를 수행한 후에는 소프트웨어의 상태가 변화하며 상태에 따라 성능 또한 달라진다. 이는 다음의 소프트웨어의 상태 전이 함수 D에 의해 결정된다.

$$D: S_S \times A \rightarrow S_S$$

기존의 동적재구성에 대한 연구들[5][6][7]은 구조 및 함수의 기능 변경에만 초점을 맞추었을 뿐, 어떤 요구사항으로부터 적응행위를 추출하고, 요구사항으로부터 소프트웨어의 어떤 성능에 초점을 맞추어야 하는지를 결정하는 지침은 다루지 않았다.

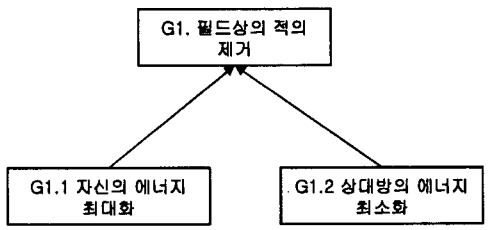
본 논문에서는 목표와 시나리오를 이용하여 환경인지 기능에서 필요한 관찰함수 O와 적응판단 기능에서 필요한 보상함수 R과 P, 그리고 동적재구성 기능에서 필요한 상태전이 함수 D를 기술하는 방법을 제안한다. 각 함수의 정의역, 공역에 해당하는 집합 A, Z, Γ, ρ, S_S를 설정하는 방법도 제안한다.

3. 목표와 시나리오를 이용한 적응형 S/W개발 방안

목표는 이해 관계자가 개발될 시스템에서 성취하고자 하는 것이며, 추상화된 최상위 수준의 요구사항이다[8]. 적응형 소프트웨어를 개발할 때 목표의 일부는 보상함수를 정의하는데 사용된다. 시나리오는 예제나 실세계의 경험을 바탕으로 추출한 요구사항이다. 시나리오는 관찰함수, 보상함수, 전이함수의 정의역과 공역에 해당하는 집합을 구성하는데 사용된다. 본 논문에서는 IBM에서 개발한 Robocode[9]에서 작동하는 로봇을 적응형 소프트웨어로 개발할 때 필요한 함수 및 집합을 정의하는 사례에 제안한 방안을 적용한다.

3.1 목표로부터 보상함수 및 보상값의 집합 추출

Robocode의 가장 기본적인 목표는 적 로봇을 필드에서 제거하는데 있다. 이를 최상위 목표로 하여 하위 목표들을 추출하면 [그림 1]과 같다.



[그림 1] 로봇의 목표 모델

보상 함수 R과 P는 앞서 2장에서 기술하였듯이, 관찰한 상태에서 최적이라고 판단한 적응행위를 수행한 직후에 측정한 값을 기록하는 역할을 한다. 현재 상태와 최적행위는 환경인지 기능과 판단 알고리즘에 의해서 선택되어 주어진다. 이들은 3.2장에서 시나리오를 이용하여 도출되므로 목표로부터 보상값의 집합만을 구성하면 된다. 보상값의 집합은 측정가능한 하위목표들로 구성한다. 일반적으로 실수집합으로 구성하는 것이 차후의 최적의 행위를 결정할 때 편리하다. 실수집합으로 구성이 불가능하면 덧셈, 곱셈 연산에 대해서 닫힌 집합이 적응판단 기능이 Q-Learning과 같은 강화학습형태로 구현될 때 유리하다. 소프트웨어 성능값의 집합도 같은 방식으로 표현가능하며, 주로 하위목표 중 소프트웨어의 비기능적 요구사항이 이에 해당된다. 측정 가능한 하위목표가 식별되면 다음과 같이 보상값과 성능값의 집합을 정의할 수 있다.

$$\Gamma = \{r \mid r = \sum w_i r_i\}$$

$$\rho = \{p \mid p = \sum e_i p_i\}$$

w_i와 e_i는 각각 개별적으로 측정한 보상값과 성능값의 가중치이다. 측정할 수 있는 값의 개수가 많은 경우 가중치를 두어 그것들을 합산하여 하나의 보상값 또는 성능값으로 이용한다. 개별적으로 측정된 값이 작아지는 것이 목표라면 가중치를 음수로 한다. [그림 1]의 하위목표 중 G1.1을 성능값, G1.2를 보상값으로 정의하면 다음과 같다.

$$\Gamma = \{r \mid r = -n, n \geq 0\}$$
$$\rho = \{p \mid p = h, h \geq 0\}$$

n는 측정된 적의 에너지이다. 적의 에너지가 최소화되는 것이 목적이므로 보상값의 가중치는 -1이 되며 범위는 0보다 큰 실수이다(0이 되면 로봇이 파괴되므로). h는 자신의 에너지이고 최대화하는 것이 목적이므로 그대로 사용하며 역시 0보다 크다.

3.2 시나리오로부터 관찰상태집합 및 행위집합 추출

적응형 소프트웨어의 시나리오는 사용자가 적응형 소프트웨어에 요구하는 예상치 또는 성향(bias)이다. 즉, 적응형 소프트웨어가 직면할 수 있는 방대한 양의 상태들 중 특정 상황에서는 특정 행위를 수행하도록 만드는 사용자의 욕구의 표현이 시나리오이다. 시나리오는 사용자가 예상하는 상태와 그 상태에서 수행하길 원하는 적응행위로 기술할 수 있으며 다음과 같은 구조로 표현 가능하다.

$$(z_i)\ \alpha\ (a_i)$$

환경인지 기능에서 관찰한 현재 상태 z_i에서 a_i라는 적응행위를 취하는 것을 사용자가 원하는 것을 말한다. 위와 같은 시나리오를 사용자로부터 다수 추출하여 다음과 같은 관찰상태 집합 Z와 적응행위 집합 A를 구성할 수 있다.

$$Z = \{z \mid z \in < z_1, z_2, \Lambda\ , z_n >\}$$
$$A = \{a_1, a_2, \Lambda\ , a_n\}$$

관찰상태의 집합 Z는 사용자가 시나리오에서 언급한 상태를 하나의 변수로 간주하고 모든 $< z_1, z_2, \Lambda\ , z_n >$로 생성된 벡터공간으로 정의된다. 적응행위 집합 A는 언급된 모든 행위를 원소로 갖는 집합으로 정의된다. 소프트웨어의 상태 집합 S_S도 관찰 상태 집합과 같은 방식으로 정의된다. 관찰함수 O는 환경의 실제 상태 $s \in S$는 값을 실제로 알 수 없으므로 적응행위 $a \in A$를 취했을 때, 정의된 관찰상태집합 Z 내의 하나의 값을 지정하도록 정의된다. 소프트웨어의 상태전이 함수 D는 소프트웨어의 현재상태 $x \in S_S$는 바로 측정이 가능하고 적응행위도 주어지므로 다음상태를 바로 S_S에서 찾도록 정의한다.

4. 결론 및 향후 연구

소프트웨어의 복잡도가 증가하고, 소프트웨어의 자율성과 견고성에 대한 요구 또한 증가하고 있다. 이와 같은 요구에 따라 실행도중에 외부환경을 인지하고 자신의 행위를 평가하여 더 나은 성능을 기대할 수 있다고 판단되면 실행을 멈추지 않고, 실행시간에 자신의 구조 및 정책을 변경할 수 있는 적응형 소프트웨어가 새로운 소프트웨어 개발 패러다임으로 주목받고 있다.

본 논문에서는 적응형 소프트웨어 개발의 초기단계에서 요구사항 분석 단계에서 추출된 목표와 시나리오로부터 어떻게 적응형 소프트웨어의 관찰함수, 보상함수, 상태전이 함수를 구성하는지에 대한 방안을 제시하였다.

향후에는 구성된 함수의 정의역과 공역을 소프트웨어로 어떻게 구현할지에 대한 방안과 적응형 소프트웨어의 구현을 용이하게 할 프레임워크를 어떤 구조로 개발할지에 대한 연구가 필요하다.

5. 참고문헌

[1] Robert Laddaga, "Active Software", In P. Robertson, R. Laddaga, and H. Shrobe, eds., Self-Adaptive Software(IWSAS 2000), Springer-Verlag, pp. 11-26, 2000

[2] Robert Laddaga, "Self-Adaptive Software", BAA 98-12, http://www.darpa.mil/ipto/solicitations/CBD_9812.html, 1998

[3] Richard S. Sutton, Andrew G. Barto, Reinforcement Learning: An Introduction, Bradford Books, 1998

[4] Stephen S. Yau, Yu Wang, Dazhi Huang, Hoh Peter In, "Situation-Aware Contract Specification Language for Middleware for Ubiquitous Computing", International Workshop on Future Trends of Distributed Computing Systems (FTDCS 2003), 2003

[5] David Garlan, "Model-based Adaptation for Self Healing Systems", ACM SIGSOFT Workshop on Self-Healing Systems(WOSS' 02), 2002

[6] R. N. Taylor, N. Medvidovic, K. M. Anderson, E. J. Whitehead Jr., et. al., "A Component- and Message-Based Architectural Style for GUI Software", IEEE Trans. on Software Engineering, Vol 22, No. 6, 1998

[7] I. Ben-Shaul, O. Holder, B. Lavva, "Dynamic Adaptation and Deployment of Distributed Components In Hadas", IEEE Trans. on Software Engineering, Vol. 27, No. 9, 2001

[8] E. Yu and J. Mylopoulos, "Why Goal-Oriented Requirements Engineering.", Proceedings of the 4th International Workshop on Requirements Engineering: Foundations of Software Quality (8-9 June 1998, Pisa, Italy). E. Dubois, A.L. Opdahl, K. Pohl, eds. Presses Universitaires de Namur, 1998. pp. 15-22.

[9] IBM, "Robocode", http://robocode.alphaworks.ibm.com/home/home.html, 2004