평면 영상 분석을 통한 상황 정보 획득 기반의 적응형 소프트웨어 프레임워크

(Adaptive Software Framework based on Acquiring Context Information using Plane Image Processing)

김 기 문 † 정 우 성 † 이 병 정 †† 우 치 수 †††
(Kimun Kim) (Woosung Jung) (Byungjeong Lee) (Chisu Wu)

요 약 오늘날 소프트웨어가 다양한 환경에서 광범위하게 사용됨에 따라 적응형 소프트웨어에 대한 요구가 증가하고 있다. 적응형 소프트웨어는 환경의 변화에 반응하여 스스로의 행동을 변화시키는 견고하고 유연한 소프트웨어이다. 그러나 환경으로부터 상황 정보를 획득하는 데 있어서의 시간 제약이나 계산 복잡도가 높은 등의 어려움으로 인해, 실제 구현 시 보다 획득이 쉬운 데이타로 환경을 한정하는 경우가 많다. 따라서 본 연구에서는 충분한 복잡도를 지니면서 획득이 쉬운 평면 영상을 환경으로 가정, 상황 정보를 획득하고 행동 규칙 정보를 바탕으로 추론하여 행동하는 적응형 소프트웨어의 프레임워크를 제안한다. 이를 바탕으로 간단한 게임을 자동으로 조작하는 소프트웨어를 구현하였다.

키워드 : 적응형 소프트웨어, 상황 인식 시스템, 영상 분석, 서술 논리 추론

Abstract As software is widely used on various environments today, there is an increasing need for adaptive software. Adaptive software is robust and flexible software that modifies its own behavior in response to the changes in its environment. Due to time constraints, high complexity and so on, it is hard to acquire context information from environment. So, when implementing software, it is common to think easily acquired data to be the environments. This research proposes an adaptive software framework that assumes plane images to be environments. Plane images are easy to acquire and have enough complexity. From this, our framework is able to acquire context information, reasons with action rule, and acts on the result of reasoning. Stand on this framework, implements software that plays a simple game automatically.

Key words : Adaptive Software, Context Aware System, Image Processing, Description Logic Reasoning

1. 서 론

오늘날 소프트웨어는 일반적인 컴퓨터 환경 위에서 동작하던 것에서 벗어나 다양한 기기를 기반으로 그 적용 범위를 점차 넓혀가고 있다. 즉, 가전기기에서부터 휴대용 전자 기기, 그리고 차량에 사용되는 시스템 등에 이르기 까지 기존에 전자 회로 기반으로 처리하였던 많은 부분을 소프트웨어가 대체하고 있다. 이렇게 소프트웨어가 다양한 기능을 가지고, 복잡하고 변화가 많은 환경에서 수행하도록 변함에 따라 기존 소프트웨어 보다 더욱 견고하고(robust) 유연한(flexible) 소프트웨어에 대한 요구가 증가하고 있다. 이러한 요구에 가장 잘 맞는 것이 적응형 소프트웨어(adaptive software)이다.

적응형 소프트웨어란 "동작하는 환경의 변화에 반응하여 자신의 행위를 수정하는 소프트웨어"를 말한다 [1]. 즉, 소프트웨어가 실행되는 동안 시스템 내부 또는 외부의 동작 환경을 감지하여, 감지한 환경으로부터 인식 가능한 상황 정보를 추출하고, 이로부터 가지고 있는 규칙에 따라 행동방식을 판단하여 스스로의 행동을 변화시키는 소프트웨어를 말한다. 이로 인해 적응형 소프

· 본 연구는 한국과학재단 특정기초연구(R01-2006-000-11150-0)지원으로 수행되었음
† 학생회원 : 서울대학교 컴퓨터공학부
gildream@selab.snu.ac.kr
wsjung@selab.snu.ac.kr
†† 종신회원 : 서울시립대학교 컴퓨터과학부 교수
bjlee@venus.uos.ac.kr
††† 종신회원 : 서울대학교 컴퓨터공학부 교수
wuchisu@selab.snu.ac.kr
논문접수 : 2007년 4월 25일
심사완료 : 2007년 6월 28일

트웨어는 수행 도중 발생하는 여러 상황에서 스스로의 동작을 종료하거나 오류를 발생하지 않고 실행을 지속할 수 있는 견고성과, 환경의 변화에 맞추어 스스로의 동작을 변화시킴으로써 어디에서나 잘 적응하는 유연성의 특성을 가진다.

그러나 적응형 소프트웨어가 시스템 외부의 환경으로부터 상황 정보를 획득하는 데는 많은 어려움이 존재한다. 우선 환경으로부터 정보를 추출할 만한 데이타를 가공하는 것이 쉽지가 않고, 데이타를 가공한 후에도 이로부터 상황정보를 추출하는 경우에도 계산의 복잡도(complexity)가 크거나 시간의 제약이 존재하는 등 어려움이 많다. 이러한 제약으로 인해 실제로 구현된 적응형 소프트웨어는 시스템 내부에서 수치로 얻을 수 있는 데이타만을 환경이라고 가정하여 구현하는 경우가 많다. 그러나 이 경우 실제 환경에 적용하기에는 다소 어려움이 있다.

따라서 본 논문에서는 평면 영상을 시스템 외부의 환경이라고 가정하여 상황 정보의 계산 복잡도를 보장하면서 획득이 용이하도록 하고, 이로부터 상황 정보를 추출하고 평가하여 스스로의 행동을 변화시키는 적응형 소프트웨어의 프레임워크를 제안한다. 또한 이를 바탕으로 실제 구현 사례를 보이고, 프레임워크의 도메인 개별 모듈을 변경함으로써 소프트웨어를 변경하는 데 적용이 가능함을 보인다.

전체적인 구성은 다음과 같다. 2장에서는 상황 정보를 기반으로 한 적응형 소프트웨어의 프레임워크와 평면 영상 분석을 통한 상황 정보 획득의 각 분야에 대해 기존 진행된 연구를 살펴본다. 3장에서는 평면 영상 분석을 통한 상황 정보 획득 기반의 적응형 소프트웨어를 구현하기 위한 프레임워크를 정의하고 설명한다. 4장에서는 이를 바탕으로 한 실제 구현 사례를 소개하고 마지막으로 5장에서는 결론과 앞으로 수행해야 하는 향후 연구를 서술한다.

2. 관련 연구

본 논문은 상황 정보를 기반으로 한 적응형 소프트웨어의 프레임워크를 구성하는 방법에 대해 제안하는 부분과, 평면 영상을 분석 및 처리하여 하여 상황 정보를 획득하는 상황 인식 기법에 대한 연구의 두 가지로 크게 나눌 수 있으며, 다음에서 그 각각에 대해 이야기하도록 하겠다.

2.1 상황 정보를 기반으로 한 적응형 소프트웨어 프레임워크 연구

상황 정보를 이용한 적응형 소프트웨어의 프레임워크에 대한 연구로는 우선 상황 인식 어플리케이션을 적은 비용으로 빠르게 개발하기 위한 서비스 기반의 SOCAM (Service-oriented Context-Aware Mobile) 아키텍처에 관한 연구가 있다[2]. SOCAM은 온톨로지(Ontology)를 이용하여 여러 상황을 기술할 수 있고, 상황 추론이나 지식 공유를 지원하며, 이 기종간의 메시지 교환이 가능하다. 상황 제공자, 상황 해석기, 서비스 위치 서비스, 상황 인식 모바일 서비스로 구성되어 있으며, OSGi(Open Service Gateway Initiative) 서비스 플랫폼상에서 구현하였다. 하지만, 주로 상황 인식 정보를 관리하기 위한 커뮤니케이션 프로토콜 및 아키텍처에 대한 내용으로 상황 인식 정보를 획득하기 위한 구체적인 방법에 대한 연구나 실현화에 대한 부분은 미흡하며, 동적으로 규칙을 변경하는 방법은 제시하지 못했다.

그리고 STEAM(Scalable Timed Event And Mobility) 미들웨어는 추론엔진으로 ECA(Event-Condition-Action) 규칙에 기반한 CLIPS(C Language Integrated Production System)를 사용하며, 복수개의 센서로부터 입력된 상황 조각(context fragments)을 통합하기 위하여 계층구조로 이루어진 베이지안 네트워크(Bayesian network)를 이용한다[3]. 하지만, 복수개의 센서가 사용된다는 가정으로부터 설계를 시작하여, 실제적인 통합 사례를 보여주지 못하고 추상화 수준에 머물러 있기 때문에 실현화에 대한 가능성이나 접근법에 대한 부분이 미흡하다.

뿐만 아니라 추론 방식으로 전략(tactic)을 사용하여 자기 적응성(self-adaptation)을 가지도록 한 Rainbow 프레임워크가 있지만, 상황 정보의 획득에 관한 부분에는 초점을 맞추고 있지는 않다[4].

3. 평면 영상 처리를 통한 상황 인식 기법 연구

일반적인 영상의 처리를 통해 정보를 획득하려는 시도는 많이 되고 있는 편이다. 그러나 이를 상황 정보로 가정하여 적응형 소프트웨어에 이용하려는 시도는 많지 않다. 그 중 풍경 영상을 분석하여 해당 영상의 방향에 대한 상황 정보를 획득하기 위한 연구가 있다. 이를 위해 우선 풍경 영상의 지평선(skyline)을 캡쳐하여 연속된 코드(series of code)로 변환하고, 동적인 임계값(threshold)을 사용하여 변환점(turning point)을 분석함으로써 빠른 처리 시간에 비해 높은 정확도를 보여준다[5]. 이와 유사하게, 주변 정보를 이용하여 동적인 상황 정보를 모니터링 하고, 이를 실시간에 처리하기 위한 연구가 있었다. 하지만, 대부분이 상황 정보 획득에 집중한 연구들이며 특정 어플리케이션에 통합하기 어렵기 때문에 해당 컴포넌트의 효율적인 재사용을 위해서는 이를 지원하는 적응형 소프트웨어의 프레임워크에 대한 별도의 연구가 필요하다.

3. 평면 영상 분석을 통한 상황 정보 획득 기반의 적응형 소프트웨어 프레임워크

적응형 소프트웨어의 기능은 크게 다음과 같이 네 가지로 크게 나눌 수 있다. 첫 번째는 자기 설정(self-configuration)으로 소프트웨어가 환경의 변화에 적응하거나 다른 시스템의 목표에 부합하도록 스스로를 자동으로 재조정 하는 기능이다. 두 번째는 자기 치유(self-healing)로 오류가 발생할 가능성이 예측되거나, 오류가 실제로 발생했을 경우 오류를 인지하고 그것을 스스로 효과적으로 복구하는 기능이다. 세 번째는 자기 최적화(self-optimization), 즉 소프트웨어가 현재의 성능을 측정하고 알려진 최적 수준이 되도록 기존에 정의된 정책에 의해 성능의 향상을 시도하는 기능이다. 마지막 네 번째 자기 보호(self-protection)는 소프트웨어가 악의적인 외부의 공격으로부터 자신을 보호하도록 스스로를 관리하는 기능이다[6].

위의 네 가지 기능을 위해 적응형 소프트웨어는 상황의 인식을 위한 모니터링(monitoring), 상황에 대한 평가(evaluation), 평가된 상황에 대한 행동(action)의 세가지 요소로 구성되어 있다[7]. 본 논문에서 구현한 프레임워크는 이러한 세가지 구성 요소 각각을 서비스의 형태로 구성하였다. 즉, 평면 영상 분석을 통해 상황 정보를 추출하는 모니터 서비스(monitor service)와 모니터 서비스에서 추출된 상황 정보와 각 도메인에 적합하도록 제공된 규칙 정보를 바탕으로 논리적으로 추론하여 소프트웨어의 행동을 결정하는 평가자 서비스(evaluator service), 결정된 행동을 실제 수행하는 행위자 서비스(actor service)의 세가지 서비스로 구성한다. 또한 시스템의 행동에 대한 규칙을 정적으로 작성하여 평가자 서비스로 전달하거나, 소프트웨어 사용자의 행동 패턴을 분석, 측정하여 해당 도메인의 규칙을 동적으로 변경하여 평가자 서비스로 전달하도록 하는 행동 규칙 생성자 서비스(action rule maker service)도 포함하였다.

이렇게 구성된 프레임워크의 각 모듈 가운데 구현하고자 하는 소프트웨어의 도메인에 상관 없이 모두 적용 가능한 부분과, 구현하고자 하는 소프트웨어의 도메인에 따라 다르게 구현하여야 하는 부분이 각각 나누어진다. 즉, 모니터 서비스에서는 각기 다른 알고리즘을 이용하며 입력으로 영상을 받고 상황 정보를 텍스트 형태로 출력하는 영상 분석 모듈과 분석된 영상을 상황 규칙(context rule)으로 만들어 주는 상황 규칙 생성자를 교체함으로써, 각 다른 도메인에 적용할 수 있으며, 행위자 서비스에서도 행동 모듈을 교체함으로써 각기 다른 도메인에서 다른 행동을 하는 소프트웨어를 구현함으로써 본 프레임워크를 적용할 수 있을 것이다. 또한 행동 규칙 생성자 서비스에서도 패턴 학습 모듈과 행동 규칙 생성자를 수정함으로써 다른 알고리즘을 사용한 패턴 학습이나 다른 도메인에서의 동적 규칙 생성이 가능할 것이다. 평가자 서비스의 경우 최대한 종속성을 제거하여 도메인 개별 모듈이 존재하지 않으므로 프레임워크를 적용하는 도메인이 변경 될 경우에도 평가자 서비스는 변경하지 않고 재사용이 가능하다.

여기에서 언급한 서비스의 구성도가 그림 1에 나타나 있으며, 다음에서 이들 서비스 각각에 대해 설명하도록 하겠다.

3.1 모니터 서비스

적응형 소프트웨어의 가장 기본이 되는 기능이 바로 환경 및 소프트웨어의 상황을 인식하는 기능이다. 상황 인식의 경우, 범용으로 사용되는 시각이나 압력 등을 측정하는 센서를 이용하여 이로부터 실 세계에 근접한 정보를 분석하여 얻는 방식과, 소프트웨어 및 소프트웨어를 구동하는 하드웨어에 한정하여 여기로부터 얻어진 정보를 분석하는 방식으로 크게 나누어진다.

이 가운데 소프트웨어와 하드웨어에서 얻을 수 있는 수치 및 데이타로 환경 및 상황 정보를 한정하는 경우, 적응형 소프트웨어로서의 구현에는 어려움이 없으나, 환경 및 상황 정보를 한정하는 데 대한 소프트웨어의 상황 인식의 한계가 발생한다. 따라서 실 세계와는 차이가 있어 실제적 적용이 힘들다. 이와는 반대로 범용 센서를 이용한 방식은 가장 실 세계와 가까워 실제로 구현되었을 경우 바로 사용이 가능하다는 장점이 있지만, 센서의 데이타로부터 환경 및 현재 상황에 대한 정보를 얻어내기가 어려우며, 또한 데이타를 정보로 가공 하는 데 많은 계산 시간과 노력이 소요되고, 실제 장치들과 연계되어 구현 및 테스트 되어야 한다는 단점이 있다. 이를 표 1에 정리하였다.

따라서 소프트웨어와 하드웨어의 내부에서 구성 가능한 정보 중 실제로 센서로부터 얻어지는 데이타와 유사한 정도의 복잡도를 가지는 환경의 구성이 필요하며, 이를 통해 실제 상황 정보 획득을 위한 모니터 서비스의 구현 및 테스트를 용이하게 하면서, 구현한 프레임워크가 큰 변화 없이 로봇이나 홈 네트워크 같은 실생활에 적용되도록 할 수 있을 것이다. 또한 소프트웨어가 실행되는 시스템에 한정하여 적용이 될 경우에도, 다른 소프트웨어의 동작에 영향을 미치기 위해서 기존의 경우 영향을 받는 소프트웨어에 상황 정보를 얻어내기 위한 특정한 정보를 삽입하여야 했던 것에 비해, 이미 그러한 고려 없이 구현되어 있는 범용 소프트웨어로부터 사람이 시각으로부터 얻어낼 수 있는 정보들 중 일부를 얻어내는 방식을 통하여 다른 소프트웨어와 상호작용하는 시스템도 구성할 수 있다.

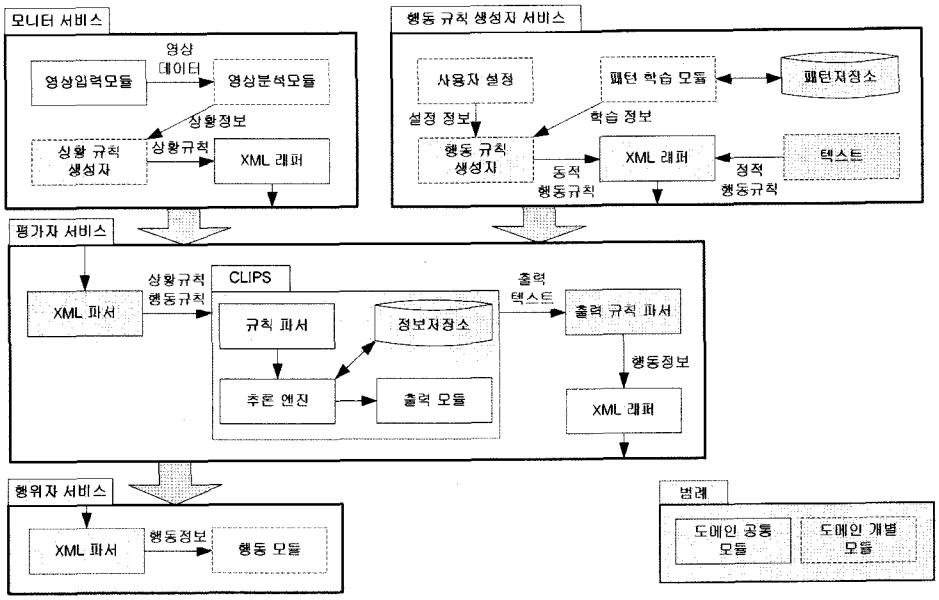

그림 1 평면 영상 분석을 통한 상황 정보 획득 기반의 적응형 소프트웨어 프레임워크

표 1 적응 환경에 따른 상황정보 획득 분류

환경 정보	시스템 내부	시스템 외부 (센서)
계산 복잡도	단순함	높음
상황 정보	획득 쉬움	획득 어려움
적용범위	좁음	넓음
구현 난이도	쉬움	어려움

이를 위해 실제로 소프트웨어 사용자의 화면의 전부 또는 일부에 해당하는 비트맵 정보를 상황 데이타로 가정하여 이를 모니터링 하고, 비트맵 데이타를 분석, 가공하여 상황 정보로 변환하는 서비스를 구현한다. 비트맵이란, 2차원의 평면 영상을 데이타화 하여 저장하는 방식을 말한다. 비트맵은 실제 영상 데이타와 그 데이타를 위한 메타데이타로 구성되어 있으며, 영상을 픽셀(pixel) 단위로 쪼개어 각 픽셀의 색상을 수치로 표현한다. 따라서 각 픽셀의 수치를 분석하고, 근접 픽셀수치까지 함께 비교하여 분석함으로써 비트맵으로부터 여러 정보를 가공하는 것이 가능하다.

비트맵으로부터 가공 가능한 정보에는 여러 가지가 있을 수 있으나, 모니터링은 일정한 시간 간격 내에 상황 정보를 도출해야 하는 시간 제한 시스템(time critical system)의 특성을 가지므로, 비트맵으로부터 가공하는 정보는 이미지에서 유효 시간 내에 계산하여 도출 가능한 정보여야 한다. 이러한 정보로는 특정 색상 또는 단순 형태의 이미지 패턴의 위치로부터의 위치 정보, 그래프나 도표 등으로부터 얻을 수 있는 수치 정보, 문자 판독 알고리즘을 이용한 문자정보 등이 있으며, 단

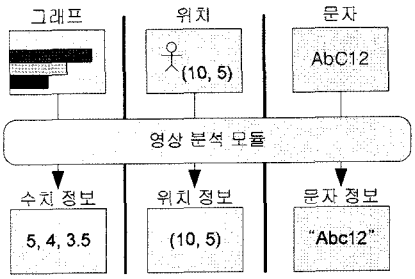

그림 2 평면 영상으로부터 획득 가능한 상황 정보

순 형태의 정보 이외에도 사람의 얼굴에서의 눈, 코, 입 등의 위치정보를 통한 신분 확인 정보나 표정 정보라든가, 기본적으로 정적으로 구성된 영상으로부터 동적 영상이 감지되었을 경우의 이벤트 정보 등이 그것이다. 비트맵 정보로부터 간단한 계산 과정을 통하여 가공 가능한 정보가 그림 2에 나타나 있다. 그 이외에도 영상 분석을 위한 많은 알고리즘 등이 존재하며, 실제 알고리즘과 하드웨어의 계산 속도의 발전에 따라 이미지로부터 획득 가능한 정보의 양은 증가할 것이다.

이렇게 영상으로부터 상황 정보가 분석이 되면 상황 정보는 다루기 쉬운 수치화된 데이타로 변환이 된다. 이 데이타는 모니터 서비스 내부의 상황 규칙 생성자에서 평가자 서비스에서 사용되는 CLIPS의 문법에 맞게 상황 정보가 상황 규칙 정보로 변환이 된다. 상황 규칙 정보는 XML(eXtensible Markup Language)의 형태로 래핑(wrapping)되어 평가자 서비스로 전송된다.

3.2 행동 규칙 생성자 서비스

시스템이 평면 영상으로부터 인지한 상황 정보에 대해 어떠한 방식으로 반응하여 행동할 것인가에 대한 일련의 행동 규칙 정보를 생성하는 서비스이다. 행동 규칙 정보는 현재 상황을 저장하고 있는 변수들과, 각 상황에 따라 취할 수 있는 일련의 행동에 대한 정보로 구성된다.

행동 규칙 정보는 프레임워크를 적용하고자 하는 도메인의 특성에 따라 정적 규칙과 동적 규칙으로 나뉘어진다. 정적 규칙은 해당 도메인에서 적응형 소프트웨어가 동작을 지속함에 따라 상황 정보에 대응하는 행동에 대한 규칙이 변화하지 않는 경우에 사용된다. 즉, 정적 행동 규칙 정보는 소프트웨어 사용자의 특성과 행위를 소프트웨어의 동작 변화에 반영하지 않는 경우 사용되며, 이는 단순히 CLIPS의 규칙 문법에 맞게 설계하여 텍스트의 형태로 작성한 다음 행동 규칙 생성자 서비스의 XML 래퍼(wrapper)로 전달하여 평가자 서비스로 전달한다.

반면, 동적 규칙의 경우는 소프트웨어나 시스템을 사용하는 사용자의 패턴을 학습하여, 이렇게 학습된 정보를 바탕으로 소프트웨어의 적응 행위 규칙(adaptive action rule)을 변경하는 방식을 말한다. 동적 규칙은 사용자의 패턴에 맞추어 소프트웨어의 적응성을 달리 하는 규칙을 말하며, 규칙 생성자 서비스로부터 사용자 패턴에 의해 학습된 규칙이 동적으로 생성되어 소프트웨어 수행 도중 수시로 변경이 되는 구조이다. 패턴에 대한 학습 알고리즘은 일반적으로 신경망(Neural Network)이나 베이지안 네트워크 등의 학습 알고리즘을 사용하여 구현이 가능하다. 이렇게 학습된 정보는 행동 규칙 생성자로 전달이 된다.

학습 알고리즘을 사용하는 이외에도 직접 사용자가 행동에 대한 여러 설정(configuration)을 통하여 행동 방식을 변경할 수 있으며 이 경우 설정된 정보가 행동 규칙 생성자로 전달이 되며, 행동 규칙 생성자는 학습이나 설정의 두 가지 방식을 통하여 CLIPS 규칙 문법에 맞도록 행동 규칙을 변경한 후, 이를 XML래퍼로 전달한다.

따라서 동적 규칙의 경우 행동 규칙 생성자는 패턴 학습 모듈과 패턴 저장소로부터 학습된 행동 규칙을 전달 받거나, 사용자 설정 모듈로부터 설정된 행동 규칙을 전달 받는다. 따라서 각 도메인에 따라 서비스 구성 모듈 중 선택적으로 일부만 구현하여 적용하는 것이 가능하다.

3.3 평가자 서비스

모니터 서비스에서 수행하는 영상 분석 및 분석을 통해 획득하는 상황 규칙 정보와 행동 규칙 생성자 서비스에서 전달하는 행동 규칙 정보는 모두 서술 논리(Description Logic)로 표현이 가능하며 따라서 적절한 추론을 통해 행동을 결정한다. 서술 논리란 FOL(First Order Logic)에 속하면서, 결정 가능(Decidable)한 특성을 지니는 지식 표현 방법이다[8].

위와 같이 서술 논리로 표현된 규칙은 추론 엔진을 사용하여 추론하는데, 본 논문에서는 CLIPS를 이용하여 추론한다[9]. CLIPS는 전문가 시스템(Expert System) 구축을 위한 도구로써 C 언어로써 구현된 규칙 기반 추론 엔진이다. CLIPS의 문법을 사용하여 소프트웨어가 적응하고자 하는 도메인에 대해 작성된 행동 규칙 정보를 입력하고, 여기에 주기적으로 모니터링 된 상황 규칙 정보를 입력하여 이 두 가지 정보를 바탕으로 추론하여 결정된 행동을 행위자 서비스로 전달한다. 이와 같은 내용이 그림 3에 나타나 있다.

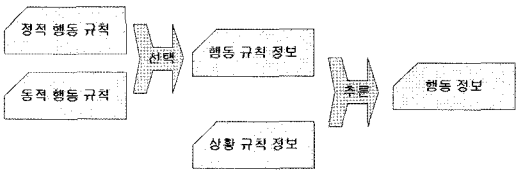

그림 3 각 규칙 정보의 추론 과정

추론 엔진의 경우 공개된 CLIPS의 소스를 변형하여 기본 추론 엔진과 규칙 파서, 정보 저장소의 기능은 그대로 두고 여기에 입출력을 위해 XML의 파서와 래퍼를 추가하여 구현하였다. 또한 CLIPS의 출력 텍스트를 분석하여 결정된 행동 정보를 추출하기 위한 파서도 추가하였다. 이렇게 추출된 행동 정보는 텍스트의 자유로운 형식으로 전달이 가능하며, 이렇게 결정된 시스템의 행동 방식은 행위자 서비스로 전달된다.

3.3 행위자 서비스

행위자 서비스의 구성은 간단하다. XML로 래핑되어 전달된 텍스트 정보를 바탕으로 행동을 결정하고, 결정된 행동을 수행한다. 이렇게 제약 조건이 적은 이유는 행위자 서비스가 취할 수 있는 행동이 소프트웨어의 도메인과 목적에 따라 다양하기 때문이다. 따라서 각기 목적에 맞는 서비스를 최대한 자율성 있게 생성하기 위해 입력은 텍스트 형식이 되고 입력에 따라 서비스는 다양한 행동을 취할 수 있게 되며, 행위자 서비스가 취한 행동이 시스템의 상황에 변화를 주어 또 다시 모니터링 서비스에서 변화된 상황에 대한 정보를 가져올 수 있게 한다.

4. 사례 연구

앞서 논의한 영상 분석 기반의 적응형 소프트웨어 프레임워크를 적용하여 자동 게임 플레이어를 구현하였다.

자동 게임 플레이어는 말 그대로 사용자가 게임을 직접 하지 않고 일정한 규칙에 의해 소프트웨어가 키보드와 마우스 입력을 흉내 냄으로써, 게임을 자동으로 진행하는 프로그램이다. 게임 소프트웨어로부터 주어진 인터페이스를 사용하여 게임으로부터 정보를 가져오는 것이 아니라, 실제로 사람이 눈으로 게임 화면을 확인하고, 게임 화면으로부터 행동을 판단하여, 그 판단을 바탕으로 키보드나 마우스를 조작하는 것과 동일하게 동작하는 것이 특징이다.

이를 위해 간단한 형태를 가지는 게임의 화면을 모니터링 하여 게임 화면으로부터 여러 정보를 추출하여 이를 규칙 기반으로 처리한 후 행동을 결정하고, 다시 결정된 행동을 키보드와 마우스의 입력을 흉내 내어 게임을 진행하고 이 화면을 다시 모니터링 하여 위의 과정을 반복함으로써 사람이 게임을 조작하는 것과 유사한 효과를 내도록 구현한다.

모니터링 하고자 하는 게임은 몇 개의 층으로 이루어진 지도 상에서 적들을 공격하여 경험치 점수를 얻는 것을 목적으로 하고 있으며, 하나의 층에서만 지속적으로 적들을 공격 하여 제거할 경우 그곳에서는 더 이상 적들이 나타나지 않게 되며, 적당한 시간 간격 동안 적이 존재하지 않아 경험치 점수를 얻지 못하면 다른 층으로 이동하여 그곳의 적들을 공격하여야 한다. 또한 게임 캐릭터에게는 체력에 해당하는 에너지가 있으며 에너지가 부족할 경우 자동으로 에너지를 보충하는 물약을 먹어야 한다. 이와 같은 목적에 맞도록 모니터 서비스를 통해 정보를 획득하며, 획득된 정보를 바탕으로 소프트웨어의 행동을 결정하고, 결정된 행동에 따라 소프트웨어가 동작한다.

4.1 모니터 서비스의 구현

모니터 서비스는 윈도우 기반의 GDI(Graphics Device Interface) 프로그램을 사용하여 작성하였다. 우선 비트맵 입력 모듈에서 윈도우에서 제공되는 API(Application Program Interface)를 사용하여 분석하고자 하는 화면을 비트맵으로 얻어낸 다음, 이를 비트맵 분석 모듈로 전달한다. 분석 모듈에서는 얻어진 비트맵의 픽셀 정보를 분석하여 원하는 정보를 얻어낸다. 픽셀 정보의 분석에는 통상적으로 GDI에서 제공되는 GetPixel 함수가 사용이 되지만, 이 함수는 매번 호출될 때 마다 화면 장치 컨텍스트(Device Context, DC)로부터 얻어오기 때문에 속도가 느려 빠른 비트맵의 처리에는 적합하지 않다. 따라서 비트맵의 가공되지 않은 픽셀 데이타로부터 영역을 계산하는 함수를 작성하여야 한다. 비트맵의 가로 픽셀 수를 w, 세로 픽셀 수를 h라 하고, 픽셀당 비트 수를 b, 가로로 하나의 줄을 표현하기 위해 사용된 바이트 수를 B라 하면, 가로 x, 세로 y에 있는 픽셀 정보는 다음과 같이 구할 수 있다.

$$Pixel(x, y) = (y * B + x * b / 8) \& ((1 << b) - 1)$$

즉, 비트맵의 특정 픽셀을 구하기 위해 가로 한 줄이 차지하는 바이트 수 B에 세로 위치 y를 곱하여 세로 위치가 가리키는 가로줄의 위치로 이동하고, 거기에서 한 픽셀이 차지하는 바이트 수인 b/8을 가로 위치와 곱해 구한다. 이 때 구해진 데이타는 하나의 픽셀이며 다른 픽셀의 값과 섞이지 않도록 하기 위해 비트 수만큼 매스킹(masking) 하여 구한다. 이 방법을 사용하여 픽셀을 구하는 함수를 작성하였다.

위의 함수를 이용하여 게임 화면 이미지로부터 픽셀 데이타를 가져오고, 가져온 데이타를 통해 모니터 서비스에서 캐릭터의 모양에 해당하는 패턴의 위치 파악을 통한 지도 상에서의 현재 위치 정보, 그래프로 표시된 캐릭터의 에너지로부터 파악할 수 있는 수치화된 정보, 더 이상 공격할 수 있는 적들이 없을 경우 경험치 점수가 증가하지 않는 것으로 판단할 수 있으므로, 화면 상에서 경험치 점수를 나타내는 부분의 변화 여부에 대한 값을 알아내야 한다.

우선 캐릭터의 현재 위치는 캐릭터의 디자인이 복잡할수록 구하기 어려워진다. 그러나 여기에서 다루는 게임에서는 게임상의 작은 지도에서 노란색 사각형 모양으로 표시됨으로써 쉽게 캐릭터의 위치를 파악할 수 있다. 따라서 전체 게임 화면에서 지도의 위치를 찾아낸 다음, 그 지도 상에서 노란색 사각형에 해당하는 영역을 찾음으로써, 지도 하나에 대한 캐릭터의 절대 위치를 구할 수 있다. 또한 게임에서 캐릭터의 에너지는 그래프로 표시되어 나타나므로 이 그래프의 최대위치와 최소위치를 파악하고, 그래프의 색상범위에 해당하는 영역을 구함으로써 수치로 환산이 가능하다. 그리고 화면의 변화 여부는 예전 화면의 비트맵 픽셀 정보를 저장하였다가 픽셀 정보를 메모리 단위로 비교함으로써 판단이 가능하므로, 경험치 점수에 대한 변화 여부를 판단할 수 있다.

이와 같은 방법으로 수치 및 진리값으로 처리된 상황 정보는 상황 규칙 생성자로 전송이 되며, 여기에서 추론 엔진인 CLIPS의 문법에 따라 상황 규칙 정보로 적절히 가공되어 평가자 서비스로 전송된다.

4.2 정적 행동 규칙의 작성

이 게임은 에너지가 0이 되지 않도록 하면서, 즉 죽지 않으면서, 짧은 시간 내에 최대한의 경험치 점수를 얻는 것을 목표로 하고 있다. 따라서 목표에 따라 CLIPS의 문법으로 직접 정적 규칙을 작성 하였다. 규칙은 현재 상태와 그 상태 하에서 어떤 조건을 만족하면 취해야 할 행동으로 구성이 되어 있으므로 다음과 같은 구조를 지닌다.

• 캐릭터의 상태는 이동과 공격 두 가지로 나뉘며 초기

상태는 이동이다.

- 상태에 상관없이 에너지가 x% 이하면 에너지를 채우는 키를 누른다.
- 상태가 이동일 경우 각 위치에서 다음 층으로 이동하기 위한 키를 누른다.
- 상태가 이동이고 원하는 위치에 왔을 때 공격 상태로 변환한다.
- 상태가 공격일 경우 공격 키를 누른다.
- 상태가 공격이고 경험치가 시간 y 동안 변화가 없을 경우 이동상태로 바꾼다.

위의 규칙에 따라 실제 작성한 CLIPS의 규칙의 일부분이 표 2에 나타나 있다. 이 규칙에서는 에너지가 15% 이하이고, 경험치의 변화가 없는 최소 시간은 1분으로 가정하였다

표 2 CLIPS로 작성된 정적 행동 규칙 정보

```
(defglobal
   ?*exp_count* = 0 ; experence point was not
changed count
   ?*exp* = 0 ; previous experence point
   ?*mode* = 0 ; mode 0 = hunt, mode 1 = move
   ?*pos* = 0 ; my current position area
)

(defrule health
   "If health point is lower than 15, then drink health
potion."
   (declare (salience 30))
   (status hp ?hp)
   (test (< ?hp 15))
   =>
   (printout t "Press key - drink health potion" crlf)
   (printout t "[ACTION] drink potion" crlf)
)

(defrule experence
   "If experence point is not changed, then increase
'experence-point-is-not-changed' count."
   (declare (salience 30))
   (status exp ?exp)
   (test (eq ?*mode* 0)) ; current mode hunting mode
   (test (eq ?*exp* ?exp)) ; experence is not changed
   =>
   (bind ?*exp_count* (+ ?*exp_count* 1))
   (printout t "Exp point is not changed - exp count
increased to" ?*exp_count* crlf)
)
```

따라서 에너지의 임계값에 해당하는 x와 경험치 점수의 변화가 없는 시간인 y는 위의 규칙과 같이 임의로 사용자가 설정이 가능하며, 이와는 달리 사용자가 직접 게임을 수행할 경우 패턴을 파악하여 학습한 결과로 값이 자동으로 설정 되거나, 사용자 설정을 통해 지정된 값으로 설정 될 수 있다. 따라서 이 부분은 규칙 생성자를 통하여 다음 4.1.3에서 동적 행동 규칙으로 작성하도록 하겠다.

4.3 행동 규칙 생성자 서비스의 구현

행동 규칙 생성자 서비스에서는 사용자가 본 논문에서 구현된 소프트웨어를 사용하여 자동으로 게임을 진행하지 않고 직접 게임을 수행할 경우 사용자의 패턴을 분석하여 위에서 작성한 규칙에서 변경 가능한 부분의 값을 설정한다.

이 사례에서 에너지가 몇 % 이하일 때 에너지를 증가시키는가와 관련된 값 x는 실제 사용자가 게임을 진행할 때 모니터링을 통해 에너지가 몇 % 이하일 때 다시 증가하는가를 측정한다. 이 값들을 충분히 측정한 다음 평균치를 구해 x로 가정한다. 경험치 점수의 경우 사용자의 위치에 따른 경험치 증가의 지속 시간을 분석하여 특정 위치에서 얻을 수 있는 경험치의 평균값과 시간 등을 분석해둔다. 이를 통해 한 위치에서 다른 위치로 이동하기 위한 경험치 점수가 변화 없는 시간이 어느 정도가 적당한지 추정할 수 있다.

여기에서 설정되는 변수들은 실제로 영향을 받는 요인이 하나밖에 없는 단순한 구조이기 때문에 각 변인들의 가중치를 구하는 방법인 신경망이나 베이지안 네트워크 등을 사용하지 않고 단순히 평균값 만으로 산출이 가능하다.

이와 같은 방법 이외에도 사용자가 설정 할 수 있는 화면을 제공하여 학습하지 않고 수치로 설정하여 설정값에 따라 동적으로 규칙을 생성하고, 생성된 규칙을 평가자 서비스로 전송하여 사용할 수도 있다.

4.4 행위자 서비스의 구현

행위자 서비스는 단순히 문자열의 형태로 동작해야 하는 행동이 입력되면 그에 따라 단순히 키보드의 입력과 마우스의 입력을 흉내 내는 단순한 구조로 구현되었다. 마우스와 키보드의 입력은 Win32 API에서 제공되는 이벤트 함수를 사용하여 구현하였다. Win32 API인 keybd_event 함수를 사용하여 사용자가 키보드를 조작한 것과 동일한 효과를 내도록 하는 행위자 서비스의 함수 사용 부분이 표 3에 나타나 있다.

이와 같이 마우스와 키보드 입력을 통해 실제로 게임상의 캐릭터가 조작이 되며, 조작된 내용이 화면을 변화시켜 모니터 서비스가 인지하는 상황 정보를 변화시킬 수 있게 된다.

4.5 평가자 서비스의 추론 과정

평가자 서비스는 각 도메인 개별 모듈이 존재하지 않는다. 따라서 개별 모듈을 구현할 필요가 없으며, 지금까지 구현한 서비스로부터 행동 규칙 정보와 상황 규칙 정보를 통해 추론을 수행한다.

표 3 Win32 API를 사용한 키보드 이벤트

```
for(i = 0; i < m_KeyBuf.Size(); i++) {
    PKEY_ITEM pItem = m_KeyBuf.Get(i);
    DBGLOG("KEY DOWN, %08X \n",
        pItem->byVk);
    keybd_event(pItem->byVk, pItem->byScan,
        KEYEVENTF_EXTENDEDKEY, 0);
    if(pItem->dwTime == 0) {
        DBGLOG("KEY UP, %08X \n",
            pItem->byVk);
        keybd_event(pItem->byVk,
            pItem->byScan,
            KEYEVENTF_KEYUP, 0);
    }
    else {
        m_KeyDownBuf.Add(pItem->byVk,
            pItem->dwTime);
    }
    m_KeyBuf.Remove(i);
    i--;
}
```

평가자 서비스는 행동 규칙 생성자 서비스가 생성한 행동 규칙 정보를 받고, 또한 모니터 서비스로부터 상황 규칙 정보를 받는다. 상황 정보의 경우 보통 특정 이름을 가지는 변수와 값의 한 쌍의 형태로 표현이 되며, 이 정보가 상황 규칙 생성자를 거치면 CLIPS의 문법에 맞는 규칙이 된다. 행동 규칙 정보의 경우 행동 규칙 생성자를 통해 학습 또는 설정된 규칙이 전송된다. 이를 기반으로 상황 규칙 정보가 들어올 때마다, 추론을 통해 조건에 맞는 규칙들이 발화(fire)된다. 모니터 서비스가 영상으로부터 상황 정보를 추출하고 이를 상황 규칙 생성자를 통해 CLIPS 규칙으로 만든 다음, XML로 래핑하여 평가자 서비스로 전송, 추론을 통해 CLIPS에서 발화되는 과정이 다음 그림 4에 나타나 있다.

해당 규칙들이 발화되면 CLIPS의 출력 함수를 사용하여 행위자 서비스가 취하여야 할 행동이 문자열로 출력이 되며, CLIPS 파싱 모듈에서 출력 함수로 출력된 내용만을 파싱하여 XML 형태로 행위자 서비스로 전송한다.

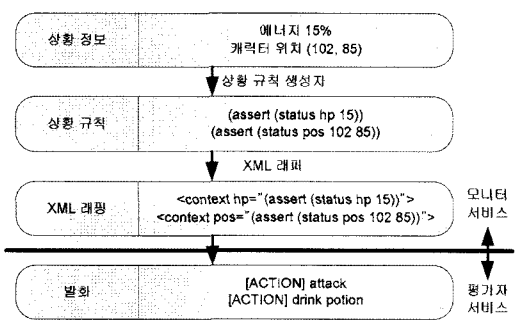

그림 4 상황 정보의 처리과정 및 발화

5. 결 론

소프트웨어가 보다 다양한 환경에서 광범위하게 사용됨에 따라 견고성과 유연성을 가지는 적응형 소프트웨어에 대한 요구가 증가하고 있다. 그러나 적응형 소프트웨어가 모니터링하고 적응하고자 하는 환경으로부터 상황 정보를 획득하는 부분에 있어서 시간 제약이나 높은 계산 복잡도 등의 어려움으로 인해, 일반적으로 구현되는 적응형 소프트웨어는 획득하기 쉬운 시스템 내부로부터 얻을 수 있는 상황 정보로 한정하는 경향이 있다.

본 논문에서는 상황 정보 획득 대상 환경을 실 세계와 유사한 복잡도를 가지는 평면 영상으로 가정한 적응형 소프트웨어의 프레임워크를 제안하였다. 즉, 평면 영상으로부터 상황 정보를 추출한 다음, 각 상황에 따라 시스템이 취해야 하는 행동을 정의한 행동 규칙 정보를 바탕으로 추론 엔진인 CLIPS를 사용하여 추론하고, 그 결과로 시스템이 취해야 하는 행동을 결정하도록 하였다. 그리고 이를 바탕으로 간단한 게임을 자동으로 조작하는 소프트웨어를 작성하였다.

한계점은 다음과 같다. 첫째, 본 연구에서 구현한 적응형 소프트웨어에서 상황으로 가정한 평면 영상은 이미 특정 위치에 지정된 형태가 존재하는 영상을 대상으로 하였다. 따라서 실 세계에서 발생 가능한 임의의 영상을 분석하여 상황 정보를 추출하는 데는 어려움이 있다. 둘째, 비록 평가자 서비스는 여러 도메인에 공통적으로 사용될 수 있지만 나머지 서비스에서 도메인이 변경될 경우 각 서비스의 도메인 개별 모듈을 교체하여야 하며, 이때 소프트웨어의 구축 비용을 줄이는 방안을 고려해야 한다.

향후에는 상황으로 가정한 평면 영상이 임의의 형태를 지니는 경우에 상황 정보를 획득하고 이 정보를 효율적으로 상황 규칙 정보로 표현 및 추론하는 방법에 대해 연구할 필요가 있다. 또한 상황을 평면 영상이 아닌 음성이나 압력과 같이 센서로부터 획득 가능하면서 충분한 계산 복잡도를 지니는 데이타로부터 상황 정보를 획득하는 방법에 대한 연구도 생각할 수 있다. 또한 현재 각 서비스 내부의 모듈들 간의 인터페이스 및 호출 프로토콜을 자세히 정의함으로써, 모듈간의 재사용성을 높이며 좀 더 다양한 적응형 소프트웨어의 개발이 가능하도록 프레임워크를 보완하도록 하겠다.

참 고 문 헌

[1] P. Oreizy, M. M. Gorlick, R. N. Taylor, "An Architecture-Based Approach to Self-Adaptive Software," *IEEE Intelligent System*, pp.54-62, 1999.

[2] T. Gu, H.K. Pung and D.Q. Zhang, "A Middleware for Building Context-Aware Mobile

Services," *In Proc. of IEEE Vehicular Technology Conference(VTC)*, 2004.
[3] G. Biegel and V. Cahill, "A Framework for Developing Mobile, Context-aware Applications," *IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 2004.
[4] S. Cheng, D. Garlan, B. Schmerl, "Architecture-based Self-Adaptation in the Presence of Multiple Objectives," *ICSE 2006 Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, May 2006.
[5] H. Liao, C. Yu, "An Image-based Approach to Generate Direction Information for Context-Aware Computing," *IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing*, pp. 152-159, 2006.
[6] M. Hinchey, R. Sterritt, "Self-Managing Software," *Software Technologies*, February 2006.
[7] Diaconescu and J. Murphy, "A Framework for Using Component Redundancy for self-Optimising and self-Healing Component Based System," *Proc. of WADS workshop in International Conference Science Engineering*, 2003.
[8] F. Baader, D. L. McGuinness, D. Nardi, P. F. Patal-Schneider, *The Description Logic Handbook*, Cambridge, 2003.
[9] http://www.ghg.net/clips/CLIPS.html.

김 기 문
2006년 서울대학교 컴퓨터공학부 졸업 (학사). 2006년~현재 서울대학교 대학원 컴퓨터공학부 석사과정. 관심분야는 소프트웨어 테스팅, 적응형 소프트웨어

정 우 성
정보과학회논문지 : 소프트웨어 및 응용 제 34 권 제 3 호 참조

이 병 정
정보과학회논문지 : 소프트웨어 및 응용 제 34 권 제 3 호 참조

우 치 수
정보과학회논문지 : 소프트웨어 및 응용 제 34 권 제 3 호 참조