모바일용 상황이해엔진의 설계 및 구현

Design and Implementation of a Context-Aware Engine for Mobile Platforms

이선아*, 이건명**, 이지형**, 김종태**, 전재욱**
*충북대학교 전기전자컴퓨터공학부, **성균관대학교 정보통신공학부

Seon A Lee, Keon Myung Lee, Jee Hyong Lee, Jong-Tae Kim, Jae-Wook Jeo
School of Electrical and Computer Engineering, Chungbuk National University
School of Information and Communication, Sungkyunkwan University
E-mail : kmlee@cbnu.ac.kr

요 약

지능로봇이나 유비쿼터스 환경에서는 주변의 상황정보를 파악하여 이를 바탕으로 적당한 서비스를 제공하는 것이 필요하다. 상황에 따른 서비스를 탐색하여 제공하는 유용한 플랫폼으로 규칙기반 시스템을 활용하는 연구가 많이 진행되어왔다. 이 논문에서는 상황이해 서비스를 위한 규칙기반 시스템의 특성을 살펴보고, 이를 바탕으로 설계하여 구현한 규칙기반 상황이해 서비스 엔진에 대해서 소개한다. 구현한 상황이해 엔진은 임베이드 환경을 비롯한 모바일 환경에서도 동작할 수 있도록 설계하였다.

1. 서론

로봇이 독거 노인의 주거생활을 지원하는데 활용되고, 유비쿼터스 환경에 대한 인프라가 구축되어가면서 유비쿼터스 서비스가 일상에 도입되어감에 따라 주변 상황에 따른 지능적인 서비스에 대한 요구는 크게 증가할 것이다. 이러한 지능적인 시스템을 효과적으로 구축하기 위해서는 다음과 같은 특성을 갖는 시스템이 요구된다. 우선 상황의 변화 또는 환경의 변화에 따라 이에 대응할 수 있는 지능서비스를 제공할 필요한 경우, 복잡한 시스템의 재구성을 피하면서 이러한 처리를 가능하게 하는 융통성(flexibility)을 갖는 것이 필요하다. 한편, 제공되는 지능서비스를 점진적으로 축적하여 지능서비스의 범위와 품질을 개선할 수 있도록 하는 확장성(extensibility)를 지원하는 것이 필요하다. 한편, 지능로봇이나 자원제한이 큰 휴대 단말기를 통해서 지능 서비스가 이루어지는 경우에서는 시스템을 이러한 플랫폼에 집어넣을 수 있는 내장성(embeddedness)를 지원하는 것이 바람직하다. 또한 이러한 서비스는 여러 응용 프로그램에서 활용되기 때문에, 다른 시스템과 연계하여 사용될 수 있는 상호작용가능성(interoperability)를 갖는 것이 요구된다.

지능로봇이나 유비쿼터스 서비스에서는 상황(context) 정보를 이용하는 것이 필수적인데, 상황은 개체의 상태를 특징짓는데 사용될 수 있는 모든 정보를 말한다. 개체는 사용자와 어플리케이션을 포함해서 사람, 장소 또는 사용자와 어플리케이션 간의 상호작용에 관련된 모든 것이 될 수 있다. 전형적인 상황 정보에는 위치, 신원, 시간, 행위 등이 있는데, 이러한 정보를 이용하여 컴퓨터는 *who*, *what*, *when*, *where*에 관련된 서비스를 할 수 있게 된다. 상황을 고려하여 사용자에게 서비스나 정보를 제공하는 시스템을 상황인식 시스템이라 한다. 상황정보는 상황 인식 어플리케이션 개발하는데 있어 여러 가지 형태으로 사용될 수 있다. Dey[1]는 상황 인식 어플리케이션 서비스의 특징을 다음과 같이 분류하여 제시하였다:

- 상황에 따라 자동으로 사용자에게 정보나 서비스를 제공하는 상황 기반의 정보 및 서비스 제공(contextual presentation)
- 상황에 따라 자동으로 서비스를 실행하거나 수정하는 상황 기반의 서비스 자동 실행(contextual execution)
- 디지털 데이터를 사용자의 상황과 연관시켜서 차후 검색을 위한 상황 기반의 정보 증강

* 본 연구는 충북대 산학협력단 부설 유비쿼터스 바이오정보기술연구센터의 지원을 받아 수행된 것임.

(contextual augmentation)
이러한 다양한 상황기반 서비스를 효과적으로 제공하기 위한 플랫폼에 대한 연구가 활발히 진행되고 있다. 이들 플랫폼의 형태는 규칙기반 시스템의 형태를 갖거나, 라이브러리 함수형태의 API로 구축되는 것이 대부분이다.

이 연구에서는 상황이해 서비스를 효과적으로 제공하기 위한 엔진 구조로서 규칙기반 표현방법과 추론기법을 제공하는 구조를 설계하고 개발하였다. 규칙기반 표현 방법에 따른 서비스 저작을 함으로써, 서비스 구성에 대한 융통성과 확장성을 확보할 수 있다. 이는 규칙의 변경, 추가 등만을 통해서 서비스에 대한 재구성이 가능하기 때문에 API를 사용하여 프로그램을 재구성해야 하는 접근방법에 대해서 융통성과 확장성이 훨씬 크다. 한편, 상황이해 서비스의 대표적인 형태인 상황기반 정보제공, 상황기반 서비스 자동실행, 상황기반 정보증강 등을 구현할 때, 규칙기반 표현을 이용하는 것이 확장성 측면에서 유리하다. 따라서 이 연구에서는 규칙기반의 상황이해 엔진을 모바일 환경등과 같이 자원 제약이 있는 상황을 상정하여 설계하고 이에 대한 초기 버전의 엔진을 구현하였다.

이 논문은 다음과 같이 구성된다. 2절에서는 개발한 상황이해 엔진을 설계할 때 고려한 사항, 시스템의 구성, 서비스 규칙의 효과적인 매칭을 위해 구현한 Rete 네트워크의 구조에 대해서 설명한다. 3절에서는 구현된 서비스 지식 표현 구문, 컴파일러 및 엔진의 구현, 최적화의 방향에 대해서 기술하고, 4절에서는 결론을 맺는다.

2. 모바일용 상황이해 엔진

2.1 상황이해 서비스의 표현 및 구현

제안한 시스템에서는 특정상황에 어떤 작성을 할 것인지에 대한 서비스를 시스템에 등록하기 위해서는 다음과 같은 규칙형태를 이용한다.

```
(defrule rule-name "comment"
  (condition-1) (condition-n)
=>
  (action-1) (action-m) )
```

규칙에서 condition 부분은 해당 서비스를 제공할 상황에 대한 조건을 나타내고, action은 해당 조건이 만족될 때 서비스할 내용이 된다. action 부분은 서비스의 종류에 따라 정보제공, 자동실행, 정보증강 등의 동작을 위한 내용이 된다. 한편, 상황정보를 비롯한 시스템이 관리할 정보는 각 정보에 대응하는 클래스를 다음과 같이 정의하고, 각 정보는 이들 클래스에 대한 객체로 나타낸다. 아래에서 slot은 해당객체를 구성하는 속성을 나타낸다.

```
(deftemple fact-class-name (slot-1) (slot-n))
(fact-class-name (slot-1 value-1) (slot-n value-n))
```

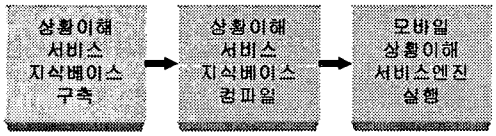
2.2 모바일용 상황이해엔진에 대한 고려사항

상황이해 엔진을 개발할 때는 자원의 제약이 큰 임베디드 시스템이나 모바일 단말기에서 상황이해엔진이 동작하는 것을 전제하고, 다음과 같은 설계결정을 하였다. 제안한 시스템은 서비스에 대한 규칙을 효율적으로 처리하기 위해서 Rete네트워크를 구현한다. Rete 네트워크를 이용할 경우에는 우선 규칙이 네트워크 형태의 데이터구조로 컴파일되어 이를 이용한 규칙에 대한 매칭연산이 수행된다. 서비스를 위한 규칙에 대한 매칭과 실행이 일어날 때는 Rete 네트워크를 구성하는 컴파일러가 불필요하다. 따라서, 기존의 규칙기반 시스템에서와는 달리 개발된 엔진에서는 컴파일러를 별도로 분리하여서, 임베디드 시스템이나 모바일 단말기에서는 엔진만 동작하도록 해서, 시스템 자원에 대한 요구량을 줄이도록 하는 방법을 택하고 있다. Rete 네트워크는 신속한 매칭연산을 위해 규칙들의 조건부를 네트워크에 대응하는 데이터구조로 나타내고, 중간 연산결과를 이 데이터구조 내에 저장해 두게 된다. 따라서 경우에 따라서는 실행시에 메모리 요구량이 많아질 수 있다. 따라서, 개발한 시스템에서는 메모리 사용을 최적화하기 위해서는 동적메모리 할당을 통해서, 필요시에만 메모리를 요구하여 사용하고 반환하도록 구현하였다. 또한 많은 경우의 속성값으로 사용되는 문자열값에 대해서 코딩을 통해서 메모리 요구량을 줄이는 것을 지원하는 것을 설계에 고려하였다. 기존의 규칙기반 시스템은 작업메모리(working memory)에 사실(fact)을 추가(assert), 삭제(delete), 수정(modify)하고, 출력하는 정도의 action만을 규칙의 결론부에서 제공하고 있다. 그런데 상황이해 엔진이 이러한 작업메모리 관련 action뿐만 아니라 다양한 서비스를 위한 action이 필요할 수 있다. 따라서 이러한 사용자 또는 시스템 설계자가 작성한 action을 서비스를 제공하는 규칙에서 호출하여 사용할 수 있도록 하는 것이 필요하다. 따라서 제안한 시스템에서는 사용자가 정의한 함수를 규칙의 결론부에서 호출할 수 있도록 하였다. 한편, 개발하는 상황이해엔진이 다른 응용프로그램에 쉽게 통합할 수 있도록 해서 전체 시스템의 구성이 원활히 이루어질 수 있도록 엔진을 라이브러리 형태로 제공할 수 있도록 하였다. 또한 C++를 개발언어로 선택하므로써 시스템의 다른 응용프로그램에서 쉽게 활용할 수 있도록 하였다. 상황이해 서비스에서는 특정 상황이 발생한 후 경과시간을 모니터링하여 특정시간을 경과하면 특정 서비스를 개시하는 경우가 종종있다. 이러한 서비스를 위해서는 규칙기반 시스템 자체가 시간정보를 작어메모리에 갖고 있는 것이 필요하다. 이러한 필요에 따라 개발된 시스템에서는 시간정보를 나타내는 time이라는 사실을 시스템 차원에서 생성하여 관리하는 방법을 지원하고 있다.

2.3 시스템 구성

상황이해엔진 시스템은 컴파일러와 실행엔진으로 구성된다. 컴파일러는 입력으로 주어진 사실 클래스 파일과 규칙 파일로부터, Rete 네트워크 구성정보와 규칙의 결론부 정보를 만들어 낸다. 실행엔진은 컴파일러가 생성한 정보를 입력으로 받아들인 다음, 시시각각 입력되는 상황관련 정보를 분석하여, 해당 상황에 적합한 서비스를 결정하여 이를 알려주는 역할을 한다. 자원제약이 큰 모바일 환경이나 임베디드 환경에서는 실행엔진만을 가지고 있으면 되

고, 컴파일러는 다른 시스템에 설치되어 있어도 무관하다.

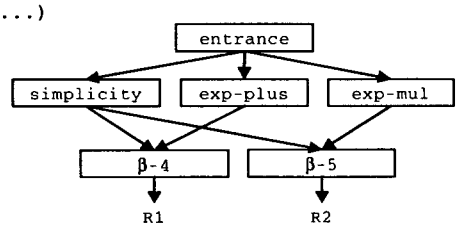


[그림 1] 상황이해 시스템 개발과정

2.4 Rete 매칭 구조

개발한 시스템에서는 서비스 규칙과 주어진 상황정보 사이의 매칭을 효율적으로 하기 위해 Rete 네트를 사용한다. Rete 네트는 규칙의 조건만의 정보를 추출하여, 중복된 비교연산을 피하고, 기존 계산결과를 활용할 수 있도록 하기 위해 네트워크 형태의 데이터구조를 만들고 여기에 새로이 주어지는 사실을 통과시키는 방법으로 실행가능한 규칙을 선정하는 역할을 한다. Rete 네트는 하나의 루트(root) 노드, 규칙의 각 조건에 대응하는 알파노드, 조건 패턴간의 비교연산에 대응하는 베타노드로 구성된다. 각 노드는 자신에 대응하는 비교연산을 수행하는데, 조건을 만족하면 해당 정보를 자신의 메모리에 저장하게 된다. 그림 2는 두 개의 규칙 R1과 R2에 대한 Rete 네트를 보여준다.

```
(R1 (has-goal (a ?x) (b simplicity))
    (exp-plus (a ?x) (b 0) (c ?y))
    =>
    ...)
(R2 (has-goal (a ?x) (b simplicity))
    (exp-mul (a ?x) (b 0) (d ?y))
    =>
    ...)
```



[그림 2] Rete 네트 구조

3. 상황이해서비스 엔진구현

3.1 서비스 지식 표현 구문

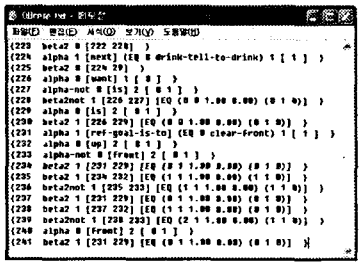
개발된 상황이해서비스 엔진은 다음과 같은 규칙 과사실의 구문을 이용하여 기술된 서비스를 처리한다. defrule은 서비스 규칙을 정의할 때 사용하며, deftemplate는 사실 클래스를 정의할 때 사용한다. 규칙에는 패턴의 속성값에 대응하는 변수 및 패턴자체에 대응하는 변수를 사용할 수 있으며, 마지막 쪽에 위치한 패턴에는 not 패턴을 사용할 수 있다. 아래 그림은 사용할 수 있는 규칙 및 사실클래스 구문을 BNF 형식으로 표현한 것이다.

```
<program> ::= {<template>} [<rule>]
<template> ::= "(deftemplate" <templ-name>
{"("<slot-name>")"} ")"
<temp-name> ::= <identifier>
<slot-name> ::= <identifier>
<identifier> ::= <letter> { <letter> | <digit> }
<rule> ::= "(defrule" <rule-name> ""<comment>""
<pattern>{<pattern>}
=>
<action>{<action>} ")"
<rule-name> ::= <identifier>
<comment> ::= {<letter>|<digit>}
<pattern> ::= <norm-pattern>{<norm-pattern>}
 {<not-pattern>}|
{<norm-pattern>}<not-pattern>{<not-pattern>}
<norm-pattern> ::= [<pattern-variable> "<-"] <fact>
<not-pattern> ::= "(not" <fact> ")"
<variable> ::= "?"<identifier>
<pattern-variable> ::= "?"<identifier>
<fact> ::= "("<predicate> {<slot>}")"
<predicate> ::= <identifier>
<slot> ::= "("<slot-name> (<slot-condition> ")"
<slot-condition> ::= <constant> |
<variable> |
<variable> "("<test> <constant> ")" |
<variable> "("<test> <expression> ")" |
"~"<constant> |
"~"<variable> |
"("<test> <constant> ")" |
"("<test> <expression> ")"
<test> ::= "=" | "~"| ">=" | "<="
<expression> ::= [<numeric> "*"] <variable>
 [+ <numeric>]
 <constant>::= <identifier>
 <slot-name>::= <identifier>
 <action> ::= <assert> | <retract> | <modify> |
<printout> | <user-defined-func>
<assert> ::= "(" "assert" "("<temp-name>
   {"(" <slot-name> <slot-value> ")"} ")"
   <slot-value> ::= <constant> | <variable> |
<func-call>
   <fact-id> ::= <variable>
   <numeric> ::= <digit> {<digit>}
   <func-call> ::= "(" "funcall" <func-name>
   {<parameter>} ")"
   <func-name> ::= <identifier>
   <parameter> ::= <constant> | <variable>
   <retract>::= "(" "retract" <fact-id> ")"
   <modify> ::= "(" "modify" <pattern-variable>
   "(" <slot-name> <slot-value> ")"
   <user-defined-func> ::= "(" <identifier>
   {<parameter>} ")"
   <printout> ::= "(" "printout" "t"
   {<variable>|<constant>} ")"
```

[그림 3]. 규칙 및 사실클래스 구문

3.2 컴파일러

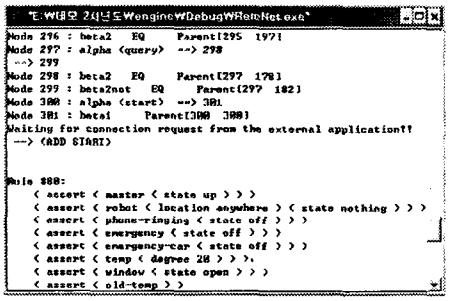
컴파일러는 서비스 규칙 파일과 서비스관련 사실 클래스 파일을 입력으로 받아서 Rete 네트워크의 구조에 대한 정보 및 규칙에서 사용된 변수에 대한 바인딩 정보를 추출하여 파일로 출력한다. 그림 4는 컴파일되어 생성된 Rete 네트워크 파일의 화면이다.



[그림 4] 컴파일된 Nete 네트 정보

3.3 엔진

엔진은 컴파일러가 생성한 Rete 네트워크 구조정보와 규칙의 action 정보 및 사실 정보를 초기화 단계에서 입력받고, 외부로부터 상황관련 정보를 온라인으로 입력받아 대응되는 서비스 수행을 위한 명령을 서비스를 실제 수행하는 시스템에 전달할 수 있게 구현되어 있다. 엔진은 C++를 사용하여 구현되었으며, LINUX 임베디드 시스템에 포팅할 경우 약 100Kbyte 정도의 메모리를 차지한다. Pocket PC에서 동작할 수 있도록 Embedded Visual C++ 환경에서 크로스 컴파일하여 실행할 수 있는 것을 확인하였다. 아래는 상황이해서비스 엔진의 실행화면이다.



[그림 5] 상황이해서비스 엔진 화면캡쳐

3.4 모바일용 최적화 구현

현재 개발된 상황이해 서비스 엔진 시스템은 MS 윈도우 환경이나 UNIX/LINUX 환경에서 동작할 수 있게 개발되었다. 또한 임베디드 환경과 모바일 단말기에서 운용될 수 있도록 메모리 사용을 최적화하기 위한 설계를 하였다. 메모리 사용을 효율적으로 하기 위해, 메모리가 일시적으로 필요한 경우에는 동적으로 필요한 만큼만 메모리를 할당하고, 더 이상 필요하지 않은 시점에서 바로 메모리를 반환하도록 하였다. 또한 반복해서 저장되는 문자열에 의한 메모리 소모를 줄이기 위해서 문자열을 코딩해서 사용할 수 있도록 설계에 반영하였다. 사용규칙에 대한 매칭을 효과적으로 하기 위해 Rete 네트워크를 사용하기 때문에, 엔진 자체에 대한 메모리 뿐만 아니라, Rete 네트워크를 위한 추가적인 메모리와 중간 매칭결과를 저장하기 위한 메모리 공간이 엔진 실행중에 사용되게 된다. 가상 메모리를 지원하지 않은 임베디드 시스템이나 모바일 단말기의 경우에는 메모리 부족이 발생할 수 있다. 따라서 상황이해 서비스를 개발할 때, 실제 상황에서 필요로 할 메모리 공간을 미리 분석하는 시뮬레이션 환경이 필요하다. 향후 메모리 사용량을 미리 추정하기 위해, Rete 네트워크에 대한 메모리 요구량을 추정하는 모듈을 컴파일러에 추가할 예정이다. 또한 서버용으로 실제 운용중에 메모리 사용량을 모니터링할 수 있는 시뮬레이션 환경을 개발할 것이다.

4. 결론

상황이해 서비스를 구현하기 위해서는 상황정보를 가공하여 상황에 따른 정보제공, 서비스의 자동실행, 상황정보 증강 등의 서비스를 구현하는 효과적인 프레임워크가 필요하다. 이러한 관점에서 개발한 규칙기반 상황이해 서비스 엔진은 효과적인 상황이해 서비스 환경을 제공한다. 또한 개발된 엔진은 자원제약이 큰 모바일 단말기 및 임베디드 시스템을 대상으로 개발되었다는 점에서 의미가 있다. 아직 개발된 시스템은 프로토타입 버전으로서, 시스템의 안정화 및 고도화 단계를 거칠 예정이다. 이 과정에서 메모리 사용량을 모니터링할 수 있는 환경이 개발하고, 엔진 자체를 라이브러리 형태로 패키징할 예정이다. 엔진이 라이브러리로 패키징되면, 다른 응용프로그램을 개발할 때 효과적으로 엔진을 활용할 수 있게 된다.

한편, 개발된 엔진은 온톨로지의 추론 엔진으로 활용될 수 있다. 향후 대표적인 온톨로지 언어의 하나인 OWL Lite에 대응하는 온톨로지를 규칙형태로 변환하는 컴파일러를 만들고, 규칙형태로 변환된 온톨로지에 대해서 추론하는 것을 개발된 엔진을 활용할 수 있도록 하는 연구를 할 예정이다. 또한, 병행해서 개발하고 있는 FPGA기반의 SoC 프로토타입에 대응하여 컴파일러가 생성한 Rete 네트워크 정보를 변환하는 변환기를 개발할 예정이다.

5. 참고문헌

[1]A. K. Dey, Providing Architectural Support for Building Context-Aware Applications, Ph.D. dissertation, Georgia Institute of Technology, 2000.

[2] P. J. Brown, The stick-e document: a framework for creating context-aware applications, Electric Publishing, Vol. 9, No. 1, pp. 1-14, 1996.

[3] D. Caswell, D. P. Debaty, Creating Web representations for places, Proceedings of the 2nd International Symposium on Handheld and Ubiquitous Computing, pp. 114-126, 2000.

[4] B. N. Schilit, System architecture for context-aware mobile computing, Ph. D. dissertation, Columbia University, New York, 1995.

[5] G. J. Nelson, Context-aware and Location Systems, Ph.D. dissertation, University of Cambridge, 1998.

[6] A. Schmidt, K. A. Aidoo, et al, Advanced Interaction in Context, Proceeding of HUC'99, pp. 89-101, 1999.

[7] A. Dey, J. Mankoff, G. Abowd, S. Carter, Distributed mediation of ambiguous context in aware environments, Proceeding of the 15th annual ACM symposium on User interface software and technology, 2002.

[8] S. Swan, An Introduction to System Level Modeling in SystemC 2.0. May 2001. http://www.systemc.org