Journal of The Korea Society of Computer and Information
Vol. 20 No. 11, pp. 119-126, November 2015
www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2015.20.11.119

Development of an Arden Syntax Translator for Building a Clinical Decision Support System with XML

Sung-Hyun Doo*, Chai Young Jung**, Jong-Min Bae***

Abstract

CDSS provides clinical doctors with knowledge to be required when they diagnose or make decision about treatment strategy. Arden Syntax is one of the language with which we write MLM that is a component of CDSS. It was designated as a standard by HL7/ANSI. ArdenML is an XML version of Arden Syntax. In this paper we propose a tool which translates Arden Syntax MLMs into ArdenML MLM. To this end we first defines the corresponding relation between two languages. Next we presents a modified version of Arden Syntax grammar to improve performance of lexical analysis and minimize parsing conflicts. Finally we presents syntax and semantics gaps between the both languages, which are a structural representation problem, a data type problem, and a disrelation problem. Our translator resolves such issues and generates exact ArdenML codes for an arbitrary Arden Syntax MLM.

▸ Keyword : Arden Syntax, ArdenML, XML, Compiler, CDSS

I. Introduction

임상의사결정시스템(CDSS: Clinical Decision Support System)은 진료에 임하는 의사가 진단이나 치료 방침을 결정 및 판단할 때 필요한 지식을 제공하고 올바르게 추론할 수 있도록 도와주는 기능을 하는 시스템이다[1]. CDSS를 활용하면 임상에서 의사가 진단하고 치료방침을 판단할 때 올바르게 추론할 수 있도록 도움을 준다. 실제로 많은 의사들은 CDSS의 필요성을 인식하고 있고 많은 병원에서 병원정보시스템을 개발할 때 CDSS를 도입하는 사례가 많다. 그럼에도 불구하고 의사의 진단에서 CDSS를 활용하는 사례는 현실적으로 그리 많지 않다. CDSS가 실용성을 가지기 위해서는 양질의 의료지식이 축적되어 있어야 하고 축적된 지식을 바탕으로 정확한 의사 결정 지원을 할 수 있어야 한다.

CDSS의 성능을 향상시키기 위해서는 다수의 임상 지식 정보가 요구된다. 그런데 각 병원에서 구축한 임상 지식 정보를 공유할 수 있으면, 많은 의료 지식을 쉽게 축적할 수 있을 것이다. 그러나 병원마다 병원 정보시스템의 자료구조와 언어가 달라 임상 지식 정보를 공유하기 어렵다. 이에 따라 HL7과 ANSI에서 임상 정보 공유를 위해 임상 지식 정보를 표현하는 언어로 Arden Syntax를 표준으로 정하였다. 그 이후 Arden Syntax는 계속 발전하여 2013년 3월에 2.9 버전이 발표 되었다[2].

Arden Syntax는 의료지식과 논리를 명시적 표현하고 관련 기관 내에 지식 공유를 가능하게 하고 의료 지식을 병원 정보 시스템에 통합하는 방법을 표준화할 목표로 만들어 졌다[3]. Arden Syntax는 임상적인 의사결정방법을 표현하는 모듈 MLM(Medical Logic Module)을 작성하는데 사용될 수 있는 컴퓨터 언어이다. MLM에는 유지 보수 정보, 지식의 다른 소스에 대한 링크, 그리고 하나의 결정을 내릴 수 있는 논리가 포함되어 있다.

그런데 Arden Syntax 표기법에는 몇 가지 표현력 상의 문제가 있다[4]. 첫째, Arden Syntax에서 중괄호는 병원

• First Author: Sung-Hyun Doo, Corresponding Author: Jong-Min Bae
*Sung-Hyun Doo(dooshs85@gmail.com), Korea Aerosoft
**Chai Young Jung (zeuschaeng@daum.net), Dept. of Prevent of Medicine, The Catholic University
***Jong-Min Bae (jmbae@gnu.ac.kr), Dept. of Computer Science, Gyeongsang National University
• Received: 2015. 09. 15, Revised: 2015. 10. 12, Accepted: 2015. 11. 10.

데이터를 읽어오는 부분에 사용한다. 병원마다 자료구조와 용어가 다르기 때문에 병원의 데이터를 변수에 대입하는 별도의 작업을 거쳐야 사용할 수 있다. 변수 명에 병원 데이터가 연결되면 MLM은 병원 데이터를 사용할 수 있게 된다. 즉, 중괄호로 묶인 부분은 병원마다 다르게 표현되는데 이것은 Arden Syntax의 목적에 벗어난다. 이 문제를 해결하기 위해 많은 연구가 진행 중이다[5, 6]. 둘째, 병원에서 MLM을 사용하려면 병원 환경에 맞게 컴파일 해주어야 한다. 병원에 따라서 C++, Mumps, JAVA 등 다양한 언어를 사용하기 때문에 MLM을 해당 언어에 맞게 소스코드로 1차 컴파일하고 소스코드를 실행환경에서 동작하도록 2차 컴파일해야 한다. 이는 Arden Syntax가 보급 되는데 방해 요소이다[4].

컴파일러 문제를 해결하기 위해 Arden Syntax를 XML 표현으로 만들자는 요구가 있었고 그 결과, Arden Syntax를 XML로 표현한 ArdenML이 발표되었다[7, 8, 9]. ArdenML은 기존의 Arden Syntax에서의 문제점을 상당히 개선한 것으로 평가되고, 향후 Arden Syntax를 대체할 것으로 예상된다. 그런데 Arden Syntax가 ArdenML로 대체되면 기존의 이미 개발되어 있는 Arden Syntax로 구현된 MLM을 계속해서 사용하기 위해서는 Arden Syntax로 표현된 MLM을 ArdenML로 변환시키는 도구가 필요하다. 이에 본 논문에서는 Arden Syntax로 표현된 MLM을 ArdenML로 변환하는 도구를 개발하고 그에 대한 평가를 제시한다. 이를 위하여 먼저 두 언어 사이의 대응관계를 정의한다. 그리고 어휘분석의 효율성과 신뢰성을 높이기 위하여, 또한 파싱 충돌을 최소화하기 위하여 수정된 Arden Syntax 문법을 제시한다. 그리고 두 언어 사이의 문법적, 의미론적인 차이점을 제시하고 구현상에서의 해결책을 제시한다.

II. Arden Syntax와 ArdenML

1. Arden Syntax

Arden Syntax로 작성된 MLM의 기본 구조는 Fig. 1과 같다.

```
maintenance:
slotname: slot-body;;
slotname: slot-body;;
...
library:
slotname: slot-body;;
...
knowledge:
slotname: slot-body;;
...
resources: <optional>
slotname: slot-body;;
...
end:
```

Fig. 1. The basic structure of MLM written in Arden Syntax[2]

MLM은 4개의 카테고리 Maintenance, Library, Knowledge, Resource로 구성된다. 각 카테고리는 여러 슬롯들로 구성된다. 각 슬롯은 몸체부분의 형에 따라서 textual slots, coded slots, textual list slots, structured slots로 나누어진다.

Arden Syntax 데이터 타입으로 Null, Boolean, Number, Time, Duration, String, Term, List, Query Results, Object, Time-of-day, Day of week등이 있다. 어휘와 데이터 타입에는 시간과 기간에 관한 항목이 많은데 이는 임상에서 발생시점과 경과시간이 중요하기 때문이다. 연산자는 List 연산자 , Where 연산자, 논리 연산자, 단순 비교 연산자, is 비교 연산자, Occur 비교 연산자, 문자열 연산자, 산술 연산자, 시간 연산자, 기간 연산자, 집합 연산자, Query 집합 연산자, 변환 연산자, Query 변환 연산자, 숫자 함수 연산자, 시간 함수 연산자, 객체 연산자 등이 있다.

명령문에는 Assignment문, If-then문, Switch-Case문, Conclude문, Call문, While-loop문 For-Loop문, New문, Read문, Reas As문, Event문, MLM문, Arqument문, Message문, Message As문, Destination문, Destination As문, Interface문, Object문 Include문, Write문, Return문, Simple Trigger문, Delayed Event Trigger문, Constant Time Trigger문, Periodic Trigger문, Constant Periodic Trigger문등이 있다. 명령문은 Knowledge 카테고리의 Data 슬롯, Evoke 슬롯 Logic 슬롯, Action 슬롯에서 각각 사용된다.

Arden Syntax로 작성된 MLM의 예를 살펴보면 Fig. 2와 같다. Fig. 2에서 Maintenance 카테고리에는 title, mlmname, arden, version 등의 슬롯이 포함되어 있다. Maintenance 카테고리는 MLM에 대한 메타데이터를 기술하고, Knowledge 카테고리에는 MLM의 실제 임상 지침 내용이 들어간다.

2. ArdenML

ArdenML은 Arden Syntax 문법의 XML 버전이다. ArdenML의 구조는 XML Schema로 정의되어 있다[10]. 여기서는 ArdenML Schema의 내용 일부를 간단히 설명한다.

최상위 엘리먼트로 <ArdenMLs> 엘리먼트가 있고 <ArdenMLs> 엘리먼트는 하위 엘리먼트로 <ArdenML> 엘리먼트가 1개 이상 있다. <ArdenML> 엘리먼트의 하위 엘리먼트 <Maintenance>, <Library>, <Knowledge>, <Resources>가 순서대로 하나만 나타난다. <Knowledge> 엘리먼트는 <Type>, <Data>, <Priority>, <Evoke>, <Logic>, <Action>, <Urgency> 엘리먼트가 순서대로 하나만 존재하고, 이 중 <Priority> 엘리먼트, <Evoke> 엘리먼트, <Urgency> 엘리먼트는 생략 가능하

```
maintenance:
      title: HgA1C Reminder;;
      mlmname: Diabetes_HgA1C_Language;;
      arden: Version 2.7;;
      version: 1.00;;
      institution: Intermountain Healthcare;;
      author: Peter Haug (Peter.Haug@imail.org);;
      specialist: Peter Haug (Peter.Haug@imail.org);;
      date: 2008-11-19;;
      validation: testing;;
library:
      purpose: Alert for needed HgA1C testing and diabetics;;
      explanation: Diabetics need to have their HgA1C measured routinely to monitor glucose control.
          This is a simple example of an alert when the last HgA1C was LE 7 but occurred more
          than six months ago.
          A reminder is issued to order a HgA1C;;
      keywords: diabetes; HgA1C;;
      citations: to be added;;
      link: to be added;;
knowledge:
      type: data_driven;
      data:
          let Last_HgA1C be read latest {"HgA1C Value"};
          let Diabetic_Patient be read latest {"Problem: Diabetes"};
          let LngCode be read latest {"Language Coe"};
          let Scheduled_Visit be event {"Scheduled_OP_Visit"};;
      evoke: Scheduled_Visit;;
      logic:
          if
               Diabetic_Patient is not null
               and Last_HgA1C occurred not within pas 6 months
               and Last_HgA1C is less than or equal 7
          then
               conclude true;
          else
               conclude false;
          endif;;
      action:
          write localized 'message' by LngCode;;
resources:
      default: en;;
      language: en
          'message': "All patients should have a HgAIC at least every 6 months.";;
      language: ko
          'message': "모든 환자는 적어도 6개월 마다 HgAIC 검사를 해야 합니다.“;;
end:
```

Fig. 2. An Example of MLM written in Arden Syntax

다. <Type> 엘리먼트의 데이터 타입은 임의의 문자열이고 <Priority> 엘리먼트의 데이터 타입은 1에서 99의 숫자이며 기본 값은 50이다.

<Urgency> 엘리먼트의 데이터타입은 1에서 99의 숫자이다. <Data>, <Evoke>, <Logic>, <Action> 엘리먼트는 복합적인 데이터 타입이 있다. LogicStatementType은 <Assignment>, <Object>, <If>, <Conclude>, <Call>, <While>, <For>, <New>, <Switch>, <Breakloop> 엘리먼트가 순서에 관계없이 1개 이상 나타날 수 있다.

<Condition>과 <Then> 엘리먼트 순서쌍은 하나 이상 나타난다. 먼저 <Condition> 엘리먼트의 하위 엘리먼트로는 ExprGroup타입의 엘리먼트가 있다. 다음 <Then>과 <Else> 엘리먼트의 하위 엘리먼트는 LogicStatementType 엘리먼트이다. 마지막으로 <Else> 엘리먼트는 생략 가능하다.

3. Arden Syntax와 ArdenML 사이의 대응관계

Arden Syntax와 ArdenML은 그 문법이 방대하여 지면 관계상 모두 표현하기는 어렵다. 따라서 여기서는 간단한 예로써 대응관계를 보이고자 한다. Fig. 3은 ArdenML로 작성한 문서의 예인데, Fig. 2의 logic 부분을 ArdenML로 표현한 것이다. Fig. 2의 Logic 슬롯은 <Logic> 엘리먼트로 표현되고, Logic 슬롯의 내용은 <Logic> 엘리먼트의 하위 엘리먼트로 표현된다. Logic 슬롯의 내용인 If문은 <If> 엘리먼트로 표현된다. <If> 엘리먼트의 하위 엘리먼트로는 조건을 기술하는 <Condition>, 조건이 참일 때 수행하는 항목을 기술하는 <Then>, 조건이 거짓일 때 수행하는 항목을 기술하는 <Else>가 있다. 이중 <Else>는 생략 가능하다. <If> 엘리먼트의 첫 번째 하위 엘리먼트 <Condition>에는 3개의 조건이 있다. 이를 <And> 엘리먼트로 묶고 <And> 엘리먼트의 하위 엘리먼트로 3개의 조건을 표현한다.

```
<Logic>
 <If>
  <Condition>
   <And>
    <Not>
     <IsNull type="is">
      <Identifier var="Diabetic_Patient" otype="string"/>
     </IsNull>
    </Not>
    <Not>
     <OccurWithinPast type="occurred">
      <Identifier var="Last_HgA1C" otype="number"/>
      <Value otype="duration" unit="months">6</Value>
     </OccurWithinPast>
    </Not>
    <IsLE type="is">
     <Identifier var="Last_HgA1C" otype="number"/>
      <Value otype="number">7</Value>
    </IsLE>
   </And>
  </Condition>
  <Then>
   <Conclude>
    <Value otype="boolean">true</Value>
   </Conclude>
  </Then>
  <Else>
   <Conclude>
    <Value otype="boolean">false</Value>
   </Conclude>
  </Else>
 </If>
</Logic>
```

Fig. 3. An Example MLM written in ArdenML

첫 번째 조건은 “Diabetic_Patient is not null”이다. 여기서 “Diabetic_Patient”는 변수이고, “is not null”은 연산자이다. 변수는 <Identifier> 엘리먼트로 표현되며, 그 자료형 속성 otype은 스트링이다. 이것을 ArdenML로 표현하면 다음과 같다.

```
<Not>
 <IsNull type="is">
   <Identifier var="Diabetic_Patient" otype="string"/>
 </IsNull>
</Not>
```

두 번째 조건은 “Last_HgA1C occurred not within past 6 months”이다. 여기서 “Last_HgA1C”는 변수이고, “occurred not within past”는 연산자이고, “6 months”는 기간이다. 이것을 ArdenML로 표현하면 다음과 같다.

```
<Not>
 <OccurWithinPast type="occurred">
   <Identifier var="Last_HgA1C" otype="number"/>
   <Value otype="duration" unit="months">6</Value>
 </OccurWithinPast>
</Not>
```

여기서 기간은 <Value> 엘리먼트로 표현한다. <Value> 엘리먼트에는 값의 타입을 나타내는 otype 속성과 부가 설명을 나타내는 unit 속성이 있다. otype의 속성값은 “number”, “string”, “duration”, “time”, “boolean”, “null”이 있다. 이 경우 값의 타입은 duration이고, 단위는 months이며, 값은 6이다.

<If> 엘리먼트의 두 번째 하위 엘리먼트 <Then>의 내용은 “conclude true”이다. 이것을 ArdenML로 표현하면 다음과 같다. 여기서 true는 <Value> 엘리먼트의 내용으로 표현된다.

```
<Then>
  <Conclude>
    <Value otype="boolean">true</Value>
  </Conclude>
</Then>
```

III. Implementation Methods

Arden Syntax를 ArdenML로 변환하는 과정은 전통적인 컴파일러 구성의 전단부에 해당하는 방법을 사용한다. 어휘분석기는 Arden Syntax로 작성된 MLM을 입력으로 받아서 토큰을 생성하고, 구분분석기는 Arden Syntax 문장에 대하여 추상구문트리를 생성하며, 코드생성기는 추상구문트리를 순회하면서 ArdenML 코드를 생성한다. 본 논문에서는 Arden Syntax 버전 2.8을 ArdenML로 변환한 결과를 제시한다. 컴파일러 생성도구로는 어휘분석기 생성기 JFlex[11]와 구문분석기 생성기 Cup[12]을 사용한다.

HL7에서 발표한 Arden Syntax에 대한 문법은 구문부분과 어휘부분으로 나누어서 BNF로 표현되어 있다. Fig. 4는 파싱과 ArdenML 코드 생성의 핵심 자료구조인 추상구문트리 노드 구조이다. Fig. 4에서 Node 클래스의 child 필드는 자식노드를 가리키고 pnumber는 생성규칙번호를 저장하는데 사용한다. codeGen() 메소드는 해당 노드형에 대한 코드를 생성하는 메소드이다. 모든 논터미널에는 그에 대응되는 새로운 노드형이 정의되고, 이 새로운 노드는 Node의 서브클래스로 정의된다.

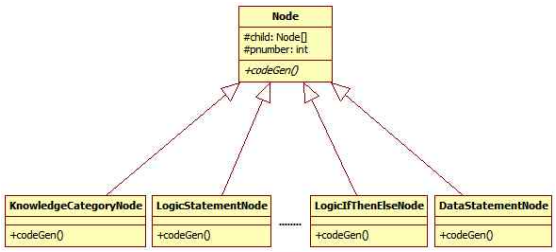

Fig. 4. Node structure of abstract syntax tree

예를 들어 Fig. 5와 같은 Cup 명세에서 세 번째 생성규칙은, Arden Syntax의 문법에서 논터미널 logic_statement에 대한 구문트리의 노드형인 LogicStatementNode 형의 객체를 생성하는 규칙이다. 이 클래스 생성자의 첫 번째 매개변수는 해당 논터미널 정의에 대한 생성규칙번호이고, 두 번째 매개변수는 자식노드에 대한 객체이다. LogicStatementNode의 자식노드는 LogicIfthenelse이다. 이와 같이 모든 생성규칙에 대하여 구문트리를 만든 후, 코드 생성시에는 구문트리를 순회하면서 노드의 형과 생성규칙번호에 따라서 그에 알맞은 ArdenML 코드를 생성한다.

```
logic_statement ::= /* empty */
   {: RESULT = new EmptyNode(); :}
   |
   logic_assignment:e1
   {: RESULT = new LogicStatementNode(NodeType.TYPE1, e1); :}
   |
   IF logic_if_then_else2:e1
   {: RESULT = new LogicStatementNode(NodeType.TYPE2, e1); :}
   |
   FOR identifier:e1 IN expr:e2 DO logic_block:e3 SEMICOLON ENDDO
   {: RESULT = new LogicStatementNode(NodeType.TYPE3, e1,e2,e3); :}
   |
   WHILE expr:e1 DO logic_block:e2 SEMICOLON ENDDO
   {: RESULT = new LogicStatementNode(NodeType.TYPE4, e1,e2); :}
   |
   logic_switch:e1
   {: RESULT = new LogicStatementNode(NodeType.TYPE5, e1); :}
   |
   BREAKLOOP
   {: RESULT = new LogicStatementNode(NodeType.TYPE6); :}
   |
   CONCLUDE expr:e1
   {: RESULT = new LogicStatementNode(NodeType.TYPE7, e1); :}
   ;
```

Fig. 5. Part of Cup specification

개발된 변환기의 정확성을 평가하기 위하여, 먼저 ArdenML로 작성된 MLM과 ArdenSyntax로 작성된 MLM을 수집한다. 웹사이트[10]에서는 ArdenML로 작성된 MLM 자료와 함께, ArdenML을 Arden Syntax로 변환하는 XSLT 파일을 제공한다. 이 웹사이트에서 제공하는 ArdenML파일과 XSLT파일을 사용하여, 먼저, XSLT로 구현된 변환기를 사용하여

ArdenML 파일을 Arden Syntax파일로 변환한 다음, 생성된 Arden Syntax파일을 본 변환기의 입력자료로 활용하여 생성된 ArdenML파일을 원래의 ArdenML과 비교하여 정확성을 분석한다.

IV. Implementing the Translator

HL7에서 발표한 Arden Syntax 문법은 컴파일러 제작에 최적화된 문법이라고 보기 어렵다. 여기서는 Arden Syntax 컴파일러 개발에 보다 최적화된 문법을 제시하고자 한다. 먼저, 주어진 문법으로 파서 제작을 위한 Cup 명세를 작성하면, 생성되는 메소드의 크기가 JVM이 허용하는 한도가 넘어서 파서가 생성되지 않는다. 이는 이 문제를 해결하기 위하여, 동일한 의미를 가지는 범위에서 논터미널을 터미널로 변경함으로써 논터미널 개수를 줄인다.

Fig. 6은 ValidationSlot 생성규칙에서 논터미널 개수를 줄이는 방법의 예를 든 것이다. Fig.6에서 validation_slot은 “VALIDATION:” <validation_code> “;;” 으로 구성된다. 여기서 “VALIDATION:”, “;;”은 터미널이고 <validation_code>는 논터미널이다. 논터미널 <validation_code>는 다시 터미널인 PRODUCTION, RESEARCH, TESTING, EXPRIED로 구성된다. 그런데 논터미널 validation_code는 4개의 터미널 PRODUCTION, RESEARCH, TESTING, EXPRIED로 구성하므로 validation_code는 어휘분석기에서 터미널 PRODUCTION, RESEARCH, TESTING, EXPRIED을 VALIDATIONCODE 라는 매크로 이름으로 정의하면 생성규칙은 “VALIDATION” VALIDATIONCODE “;;” 가 되어서 생성규칙의 개수를 줄일 수 있다.

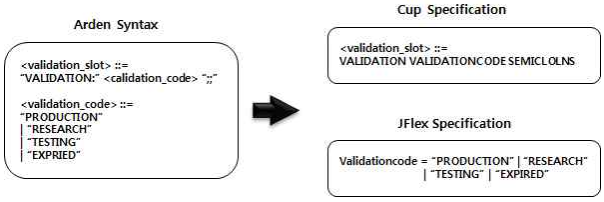

Fig. 6. The modified production rules for ValidationSlot

Fig. 7은 이와 같은 방법으로 새로 정의된 터미널에 대한 매크로 정의이다.

```
validationcode = "PRODUCTION" | "RESEARCH" | "TESTING" | "EXPIRED"
citationtype   = ( "SUPPORT" | "REFUTE" )?
linktype = ( "URL_LINK" | "MESH_LINK" | "OTHER_LINK" | "EXE_LINK" )?
```

```
trimoption     = ( "LEFT" | "RIGHT" )?
caseoption     = "UPPERCASE" | "LOWERCASE"
it = "IT" | "THEY"
is = "IS" | "ARE" | "WAS" | "WERE"
occur                  = "OCCUR" | "OCCURS" | "OCCURRED"
ofreadfuncop = "AVERAGE" | "AVG" | "COUNT" | "EXIST" | "EXISTS" | "SUM"
                | "MEDIAN"
ofnoreadfuncop = "ANY ISTRUE" | "ALL" | "ALL ARETURE" | "NO"
    | "NO ISTRUE" | "SLOPE" | "STDDEV" | "VARIANCE" | "INCREASE"
    | "PERCENT INCREASE" | "% INCREASE" | "DECREASE"
    | "PERCENT DECREASE" | "% DECREASE" | "INTERVAL" | "ARCCOS"
    | "ARCSIN" | "ARCTAN" | "COSINE" | "COS" | "SINE" | "TANGENT"
    | "TAN" | "EXP" | "FLOOR" | "INT" | "ROUND" | "CEILING"
    | "TRUNCATE" | "LOG" | "LOG10" | "ABS" | "SQRT" | "EXTRACT YEAR"
    | "EXTRACT MONTH" | "EXTRACT DAY" | "EXTRACT HOUR"
    | "EXTRACT MINUTE" | "EXTRACT SECOND"
    | "EXTRACT TIME OF DAY" | "STRING" | "EXTRACT CHARACTERS"
    | "REVERSE" | "LENGTH" | "CLONE" | "EXTRACT ATTRIBUTE NAMES"
fromoffuncop = "MINIMUM" | "MIN" | "MAXIMUM" | "MAX" | "LAST" | "FIRST"
    | "EARLIEST" | "LATEST"
indexfromoffuncop = "INDEX MINIMUM" | "INDEX MIN" | "INDEX MAXIMUM"
    | "INDEX MAX" | "INDEX EARLIEST" | "INDEX LATEST"
atleastmostop= "AT LEAST" | "AT MOST"
durationop     = "YEARS" | "MONTHS" |  "WEEKS" |  "DAYS" |  "HOURS"
    |  "MINUTES" | "SECONDS"
booleanvalue = "TRUE" | "FALSE"
weekdayliteral = "SUNDAY" | "MONDAY" | "TUESDAY" | "WEDNESDAY"
    | "THURSDAY" | "FRIDAY" | "SATURDAY"
typecode = "DATA_DRIVEN" | "DATA-DRIVEN"
```

Fig. 7. Macro definitions for new terminals

두 번째 문제로서, Arden Syntax Version 2.8 문법에는 모호한 문법이 포함되어 있다. 그 결과 파싱 과정에서 Shift/Reduce 충돌 혹은 Reduce/Reduce 충돌이 발생한다.

Fig. 8은 Arden Syntax Version 2.8 문법 중에서 Reduce/Reduce 충돌이 일어나는 문법 중의 하나이다. 이 경우에는 생성규칙 <expr_ago> ::= <expr_duration>과 <expr_duration> ::= <expr_funtion>은 동일한 언어를 생성하기 때문에 생성규칙 <expr_ago> ::= <expr_funtion>을 삭제하면 Reduce/Reduce 충돌문제를 해결할 수 있다.

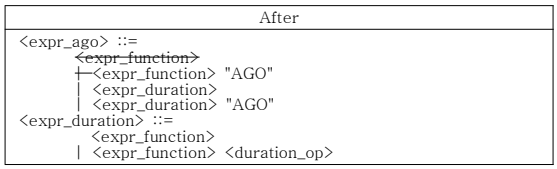

Fig. 8. A grammar with Reduce/Reduce conflict(1)

Fig. 9는 Arden Syntax Version 2.8 문법 중에서 Reduce/Reduce 충돌이 일어나는 또 다른 문법이다.

논터미널 <evoke_statement>의 세 번째 생성규칙 <evoke_statement> ::= <evoke_time>과 논터미널 <evoke_time_expr>의 두 번째 생성규칙 <evoke_time_expr> ::= <evoke_time>은 논터미널 <evoke_time>으로 동일하다. <evoke_statement> ::= <evoke_time>에 의하여 <evoke_time>이 생성될 수 있고, <evoke_statement> ::= <delayed_evoke>에 의해서도 <evoke_time>이 생성될 수 있다. 이 경우 세번째 생성규칙 <evoke_statement> ::= <evoke_time>을 생략하면 충돌문제를 해결할 수

After

```
<evoke_statement> ::=
 /* empty */
| <event_or>
| <evoke_time>
| <delayed_evoke>
| <qualified_evoke_cycle>
| "CALL"

<delayed_evoke> ::=
  <evoke_time_expr_or> "AFTER" <event_time>
| <evoke_time_expr_or>
| <evoke_duration> "AFTER" <evoke_time_or>

<evoke_time_expr_or> ::=
  <evoke_time_expr>
| <evoke_time_expr> "OR" <evoke_time_expr_or>

<evoke_time_expr>::=
  <evoke_duration>
| <evoke_time>
```

Fig. 9. A grammar with Reduce/Reduce conflict(2)

```
<expr_add_list> ::=    <expr_remove_list>
  | "ADD" <expr_where> "TO" <expr_where>
  | "ADD" <expr_where> "TO" <expr_where> "AT" <expr_where>
```

Fig. 10. A grammar with shift/reduce conflict

```
precedence left COMMA;
precedence left DOT;
precedence nonassoc USING;
precedence nonassoc AT, FROM;
precedence left DVBAR;
precedence nonassoc FORMATTED;
precedence nonassoc AFTER;
precedence nonassoc AGO;
precedence nonassoc TIME;
```

Fig. 11. Precedence options in Cup specification

Table 1. Grammars with shift/reduce conflict

순번	문 법
1	<object_init_element> ::= <identifier> ":=" <expr> <expr> ::= <expr> "," <expr_sort>
2	<identifier_or_object_ref> ::= <identifier_or_object_ref> "." <identifier_or_object_ref>
3	<expr_function> ::= <from_of_func_op> "OF" <expr_function> <expr_function> ::= <from_of_func_op> "OF" <expr_function> "USING" <expr_function>
4	<expr_function> ::= <from_of_func_op> <expr_function> <expr_function> ::= <from_of_func_op> <expr_function> "USING" <expr_function>
5	<expr_sort> ::= <expr_add_list> "MERGE" <expr_sort> <expr_sort> ::= <expr_add_list> "MERGE" <expr_sort> "USING" <expr_function>
6	<expr_sort> ::= "SORT" <sort_option> <expr_sort> <expr_sort> ::= "SORT" <sort_option> <expr_sort> "USING" <expr_function>
7	<expr_ago> ::= <expr_duration> <expr_before> ::= <expr_duration> "FROM" <expr_ago>
8	<expr_function> ::= <expr_factor> <expr_function> ::= <from_of_func_op> <expr_factor> "FROM" <expr_function> "USING" <expr_function> <expr_function> ::= <from_of_func_op> <expr_factor> "FROM" <expr_function>
9	<expr_function> ::= <expr_factor> <expr_function> ::= <index_from_of_func_op> <expr_factor> "FROM" <expr_function>
10	<expr_string> ::= "TRIM" <trim_option> <expr_string> <expr_string> ::= <expr_string> "||" <expr_plus>
11	<expr_string> ::= <case_option> <expr_string> <expr_string> ::= <expr_string> "||" <expr_plus>
12	<expr_string> ::= "TRIM" <trim_option> <expr_string> <expr_string> ::= <expr_string> "FORMATTED" "WITH" <format_string> <expr_string> ::= <expr_string> "FORMATTED" "WITH" <expr_plus>
13	<expr_string> ::= <case_option> <expr_string> <expr_string> ::= <expr_string> "FORMATTED" "WITH" <format_string> <expr_string> ::= <expr_string> "FORMATTED" "WITH" <expr_plus>
14	<evoke_time_expr>::= <evoke_duration> <delayed_evoke>::= <evoke_duration> "AFTER" <evoke_time_or>
15	<expr_duration> ::= <expr_function> <expr_ago> ::= <expr_function> "AGO"
16	<sort_option> ::= /*empty*/ <sort_option> ::= "TIME"

Fig. 10은 Arden Syntax Version 2.8 문법 중에서 Shift/Reduce 충돌이 발생하는 문법이다. Fig. 10의 두 번째 생성규칙과 마지막 생성규칙에서 "AT" 토큰이 파서에 입력 될 때 파서는 두 번째 생성규칙으로 Reduce 할지 아니면 마지막 생성규칙으로 Shift 할지 충돌이 생긴다. 이 경우에는 “AT”이 토큰으로 입력될 때 “AT”은 Reduce보다 Shift를 우선 처리 하도록 우선순위를 명시한다. 이와 유사한 문제를 가지는 문법을 전체적으로 정리하면 Table 1과 같고, 이는 Fig. 11과 같이 우선순위와 결합법칙을 명시하여 해결할 수 있다.

구문 분석기에서 추상구문트리를 생성한 후, 이를 순회하면서 목적코드인 ArdenML을 생성한다. 추상구문트리를 구성하는 각 노드의 코드 생성 알고리즘은 다양하게 나타난다. 이를 크게 구분하면 카테고리/슬롯의 코드 생성과 명령문/연산자의 코드 생성으로 나눌 수 있다. 카테고리/슬롯의 코드 생성 부분은 양쪽의 문법적 대응구조가 직관적으로 정의되기 때문에 비교적 쉽게 코드를 생성할 수 있으며, 명령문/연산자의 코드 생성은 전통적인 컴파일러 이론의 코드 생성기술을 그대로 이용할 수 있다.

V. Evaluation

본 변환기에 의해서 생성된 ArdenML 파일을 기존의 ArdenML과 비교한 결과 대부분 정확히 변환 되었지만, 몇 가지 문법에서 서로 일치하지 않는 부분이 있다. 먼저 Author slot, Institution slot, Author slot, Specialist slot, Keywords slot, Citation slot,

Link slot의 내용은 Arden Syntax에서 비구조적 문장, 즉 일반 텍스트 문장으로 표현되는데 반하여, ArdenML에서는 모든 문장은 구조화된 표현이기 때문에 변환이 어렵다. 예를 들어 Author_slot의 문법은 Fig. 12와 같다.

```
<author_slot> ::= "AUTHOR:" <text> ";;"
<text> ::= /* any string of
                    characters without ";;" */
```

Fig. 12. Arden Syntax grammar for author slot

여기서 <text>는 임의의 비구조적 문장이다. 그런데, 이에 대응되는 ArdenML 구조는 Fig. 13과 같다.

<author_slot>에 대응되는 ArdenML 엘리먼트는 Author 엘리먼트이다. 그리고 논 터미널 <text>는 Fig. 13에서 <Person>엘리먼트의 자식 엘리먼트로 분해되어야 한다. 즉, 비구조적 텍스트로부터 이름, 연락처, 학위 등의 구조적 정보를 추출해야 한다. 이는 또 다른 기술을 필요로 한다.

여기서는 유효한 XML문서로 만들기 위해 <Person> 엘리먼트의 자식 엘리먼트 중에서 필수 엘리먼트인 Name(FirstName, SurName) 엘리먼트를 내용 없이 변환 결과에 포함시키고 <Person> 엘리먼트의 나머지 자식 엘리먼트들은 모두 생략하는 방법으로 이 문제를 해결하였다. 이와 같이 텍스트로부터 구조적 정보를 추출해야 하는 슬롯은 Author슬롯 이외에도 Institution slot, Author slot, Specialist slot, Keywords slot, Citation slot, Link slot 등이 있다. 이들은 모두 <Author> 엘리먼트와 유사한 방법으로 ArdenML을 생성한다.

다음으로 자료형 문제이다. 예를 들어 Fig. 14에서 <Identifier> 엘리먼트의 자료형은 string이다. 그런데 Arden Syntax에는 자료형을 표시하는 문법이 없다. 이를 근본적으로 해결하기 위해서는 Arden Syntax문장으로부터 자료형을 추론해야 한다. 본 변환기에서는 ArdenML에서의 자료형 속성을 항상 생략한다.

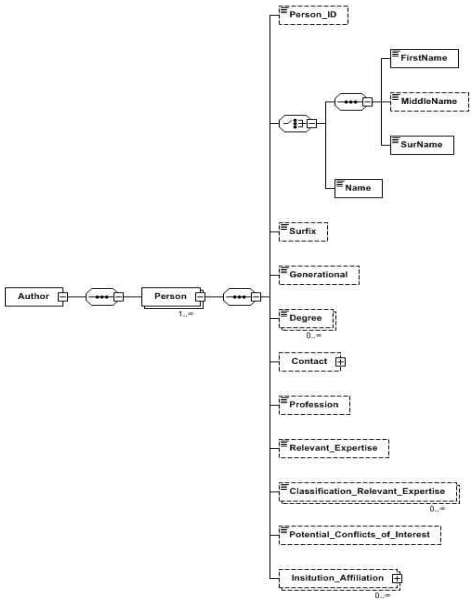

Fig. 13. ArdenML structure for <author_slot>

```
<Read>
  <Identifier var="PPD_Test" otype="string"/>
    <Assigned>
      <Last>
        <Mapping>
            <Contents>PPD Skin test Result</Contents>
              <XForms>
                <select1>
                  <label>PPD Skin Test</label>
                    <item>
                        <label>Positive</label>
                        <value>positive</value>
                    </item>
                    <item>
                        <label>Negative</label>
                        <value>negative</value>
                    </item>
                </select1>
              </XForms>
            </Mapping>
        </Last>
    </Assigned>
</Read>
```

Fig. 14. An instance of <Read> element

마지막으로 Arden Syntax와 ArdenML사이의 대응관계가 없는 경우가 있다. 예를 들면 Fig. 14는 DB에서 PPD Skin test 결과를 추출하여 가장 최근 정보를 변수 PDD_TEST에 저장하는 ArdenML 문서이다. 이는 Arden Syntax로 다음과 같이 변환될 수 있다.

```
PPD_Test := read last {PPD Skin test Result};
```

이 예에서 ArdenML의 <XForms> 엘리먼트에 대해서는 대응되는 Arden Syntax 문법이 없기 때문에 상호 변환이 불가능하다. <XForms> 엘리먼트는 필수 엘리먼트가 아니기 때문에 이를 생략하더라도 유효한 XML 문서가 될 수 있다. 따라서 이에 대한 간단한 해결책으로는 대응관계가 없는 <XForms> 엘리먼트는 생략하는 것으로 구현하였다.

VI. Conclusions

양질의 임상의사결정시스템을 구축하기 위해서는 다수의 임상지식 정보가 요구된다. Arden Syntax와 ArdenML은 병원마다 별도로 구축된 임상지식정보를 쉽게 공유할 목적으로 만들어진 HL7 표준인데, ArdenML은 향후 CDSS를 위한 표준언어로 자리 잡을 것이다. 따라서 Arden Syntax로 구축된 기존의 MLM을 ArdenML로 다시 작성할 필요가 있다.

본 논문에서는 Arden Syntax Version 2.8을 ArdenML로 변환하는 도구를 개발한 결과를 제시하였다. 구현 결과, 비구조적 텍스트로 부터 구조화된 표현을 요구하는 경우와 자료형 부분, 그리고 Arden Syntax와 ArdenML의 상호 대응되는 문법이 없는 경우를 포함해서 정확하게 ArdenML을 생성하였다. 개발된 도구는 향후 기존의 Arden Syntax로 작성된 MLM을 ArdenML로 변환시키는데 유용하게 활용될 수 있는데, 이는 향후 ArdenML을 처리하는 다양한 도구가 개발될 때 본 도구의 활용 가능성은 더 높을 것으로 기대된다.

REFERENCES

[1] Greenes R. “Clinical Decision Support: The Road Ahead”, Oxford, UK: Elsevier, 2007.

[2] Health Level Seven, Inc., Arden Syntax Histroy, http://www.hl7.org/implement/standards/ansiapproved.cfm

[3] Health Level Seven, Inc., "Health Level Seven Arden Syntax for Medical Logic Systems, Version 2 . 8 , " http://www.hl7.org/implement/standards/product_brief.cfm?product_id=268

[4] Sukil Kim and Inyoung Choi, "Arden Syntax as a standard expression language for medical knowledge", Journal of Korean Society of Medical Informatics 14(1):1-7, March 2008

[5] Health Level Seven, Inc., Virtual Medical Record, http://wiki.hl7.org/index.php?title=Virtual_Medical_Record_(vMR

[6] Health Level Seven, Inc. Reference Information Model, http://www.hl7.org/implement/standards/rim.cfm

[7] Sailors M. "ArdenML: The Arden Syntax Markup Language (or Arden Syntax: It's Not Just Text Any More!)", Proc AMIA Symp, 2001

[8] Kim S, Haug PJ, Rocha RA and Choi I. "Modeling the Arden Syntax for medical decisions in XML", International Journal of Medical Informatics 77(10), Oct. 2008.

[9] Jung CY, Sward KA, Haug PJ, "Executing medical logic modules expressed in ArdenML using Drools", Journal of the American Medical Informatics Association 19(4), April 2012.

[10] ArdenML Example, http://bmi.inyourdream.net/ardenml/

[11] Gerwin Klein, "JFlex - The Fast Scanner Generator for Java", http://jflex.de/

[12] Scott E. Hudson, "Cup Parser Generator for JAVA", http://www2.cs.tum.edu/projects/cup/

Authors

Sung-Hyun Doo received the BS and MS degree in Computer Science from Gyeongsang National University, korea, in 2011 and 2013 respectively He is currently working at Korea Aero Soft, Sacheon, korea. He interested in programming languages, compilers and open source.

Chai Young Jung received the B.S., M.S. and Ph.D. degrees in Computer Science from Gyeongsang National University, Jinju, Korea, in 1998, 2001 and 2004, respectively. Dr. Jung was the postdoctoral fellow in the Department of Biomedical Informatics at University of Utah, USA from 2008 to 2011. He is currently the researcher in Department of Preventive Medicine at College of Medicine, The Catholic University of Korea. He is interested in medical terminology, medical standardization, and Arden Syntax.

Jong-Min Bae received the B.S. degree in mathematics education, and the M.S. and Ph.D. degrees in Computer Science and Statistics from Seoul National University, in 1980, 1983 and 1995, respectively Dr. Bae joined the faculty of the Department of Computer Science at Gyeongsang National University, Jinju, Korea, in 1984. He is currently a Professor in the Department of Computer Science, Gyeongsang National University. He is interested in programming languages, computer education, and compilers.