병렬 소프트웨어 프러덕트 라인을 위한 컴포넌트 생성기

장정아° 최승훈
덕성여자대학교 전산및정보통신대학원
{jjung°, csh}@duksung.ac.kr

Component Generator for Concurrent Software Product Lines

Jeong Ah Jang°, Seung Hoon Choi
Dept. of Computer Science, Duksung Women's University

요 약

소프트웨어 프러덕트 라인 개발 방법론이란 개발 초기에 시스템의 공통적인 부분과 가변적인 부분을 명확히 하여 소프트웨어 자산을 구축한 후 다양한 요구 사항에 따라 가변적인 부분을 커스터마이징하여 목표 시스템을 생성하는 소프트웨어 개발 패러다임이다. 일반적인 소프트웨어 프러덕트 라인에 대한 연구는 활발히 진행되고 있지만, 병렬성을 지원하는 소프트웨어 프러덕트 라인에 대한 연구는 상대적으로 미약하다. 본 논문에서는 병렬 소프트웨어 프러덕트 라인 구축에 있어서 특성 모델을 통해 기능적 가변성을 지원하고 상태 다이어그램을 통해 동시성을 지원하는 컴포넌트의 코드를 자동 생성하는 도구를 제안한다. 본 연구 결과는 병렬성이 중요한 실시간 임베디드 소프트웨어 프러덕트 라인 구축에 활용될 수 있다.

1. 서론 및 연구 배경

소프트웨어 프러덕트 라인이란, 개발 초기에 시스템의 공통적인 부분과 가변적인 부분을 명확히 하여 소프트웨어 자산을 구축한 후 다양한 요구 사항에 따라 가변적인 부분을 커스터마이징하여 목표 시스템을 생성하기 위해서 개발된 소프트웨어 집약 시스템들의 집합을 의미한다.[1]

소프트웨어 프로덕트 라인의 목적은 특정 도메인에서 주로 사용되는 공통된 핵심 자산들을 초기 단계에 먼저 개발한 후, 소프트웨어 생산 시 정해진 방식에 의해 이들을 조립함으로써, 비슷한 특성을 가지고 있지만 특정 부분이 다른 일련의 다양한 소프트웨어들을 보다 빠르고 좋은 품질을 갖도록 생산하고자 하는 데에 있다. 즉, 소프트웨어 개발 단계 초기에 소프트웨어 패밀리에 속하는 멤버들 사이의 차이점과 공통점을 미리 예측하고 분석함으로써 보다 전략적인 재사용이 가능하도록 하여 소프트웨어 개발 생산성을 향상시키고자 하는 것이다.

현재 일반적인 소프트웨어 프러덕트 라인에 대한 연구는 활발히 진행되고 있지만, 실시간 임베디드 소프트웨어에서 가장 중요한 개념인 병렬성(concurrency)을 지원하는 소프트웨어 프러덕트 라인에 대한 연구는 상대적으로 미약한 상황이다.

본 논문에서는 병렬 소프트웨어 프러덕트 라인 구축을 효율적으로 지원하기 위해, 특성 모델을 통해 기능적 가변성을 지원하고 상태 전이 다이어그램(State Transition Diagram)을 통해 동기화을 지원하는 컴포넌트의 코드를 자동 생성하는 도구를 제안한다. 본 도구에서의 컴포넌트 패밀리는 자동 생성 프로그래밍의 한 기법인 GenVoca[2]의 아키텍처를 가지며 XSLT 스크립트가 컴포넌트를 구성하는 구성 요소의 코드 템플릿을 제공한다. 본 논문의 컴포넌트 코드 생성 도구는 특성 구성을 통해 재사용자의 요구에 맞는 병렬 컴포넌트 소스 코드를 자동 생성함으로써 실시간 임베디드 소프트웨어 프러덕트 라인 개발의 생산성을 향상시킨다.

본 연구와 관련하여, 문서 표현의 표준 언어인 XML을 소프트웨어 자동 생성에 이용하기 위한 연구가 많이 진행되고 있다. 특히 [3]에서는 문서 표현의 표준 기술인 XML을 이용하여 프로그램의 추상적 명세서를 작성하고, 이를 받아들여 프로그램 코드를 자동 생성하는 프로그램 생성기를 제안하였다. [4]에서는 프러덕트 패밀리를 구성하는 자산들을 가변성을 수용하기 위한 방법으로 구조화하고, XML을 기반으로 한 명세서를 해석하여 자산의 커스터마이징을 반자동화하는 도구에 대한 연구를 수행하였다. [5]에서는 병렬 소프트웨어 컴포넌트를 의미하는 수동적 객체와 일반 컴포넌트를 의미하는 능동적 객체 개념과 동기화를 통해 병렬성을 지원하는 프로그래밍 방법을 제안하였다.

본 논문의 구성은 다음과 같다. 제 2 장에서 병렬 소프트웨어에서의 컴포넌트를 정의한다. 제 3 장에서 특성 모델을 통한 가변성 지원 과정과 상태 전이 다이어그램을 통한 동기화 지원 과정을 기술한다. 제4장에서 결론 및 향후 연구 과제를 설명한다.

2. 병렬 소프트웨어에서의 컴포넌트

병렬 소프트웨어란(concurrent software), 동시에 다수의 작업을 수행하도록 설계된 소프트웨어를 말한다. 병렬 소프트웨어는 능동 객체, 수동 객체, 제어 객체로 이루어진다. 여기서 능동 객체는 독자적인 제어를 가지고 독립적인 작업을 수행하는 객체(thread 또는 task)를 의미하고, 수동 객체는 여러 능동 객체들 사이에 공유되는 자원으로서 능동 객체들 사이의 동기화(synchronization)를 지원하여 공유 자원의 데이터 일관성을 유지해주는 컴포넌트를 의미하며, 제어 객체는 객체들 사이의 연결 및 생성을 제어한다.[5] 본 논문에서는 동기화를 지원하는 수동 객체의 코드 생성에 초점을 맞춘다.

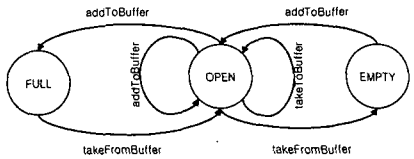
병렬 처리를 지원하는 병렬 컴포넌트는 다수의 작업이 동시 발생할 때 생기는 위험을 줄이고 효율적으로 관리하기 위해 동기화(synchronization) 지원이 필수적이다.

[5]는 모니터 자료구조를 이용하여 상태 다이어그램으로부터

동기화를 지원하는 수동 객체 구현 메커니즘을 제시하였다.

상태 다이어그램은 코드로의 변환이 가능하며, 병렬 소프트웨어의 코드 변환 과정에는 전제조건(pre-condition)과 후행조건(post-condition)이 포함된다. 전제조건에서는 객체가 실행 불가능한 상태일 때 wait() 명령을 수행하도록 함으로써 객체를 충돌로부터 보호한다. 후행조건에서는 notifyAll() 명령을 통해 후 객체의 상태가 전이되었음을 대기 중인 쓰레드들에 알리고, 다음 작업을 수행하도록 한다.

본 논문에서는 *BoundedBuffer*를 예제로 사용하였다. *BoundedBuffer*는 원형 큐(Queue)와 같은 배열로, 버퍼로 원소(element)를 저장하는 *Producer*와 원소를 순서대로 가져와 출력해주는 *Consumer*를 갖는다. 여기서 *Producer*는 버퍼가 가득 찼을 경우, *Consumer*는 버퍼가 비었을 경우 작업을 수행할 수 없다. 때문에 wait() 명령을 통한 객체의 제어를 필요로 한다. [그림 1]은 *BoundedBuffer*의 상태 전이 다이어그램과 상태 이전에 대한 명세서이다.



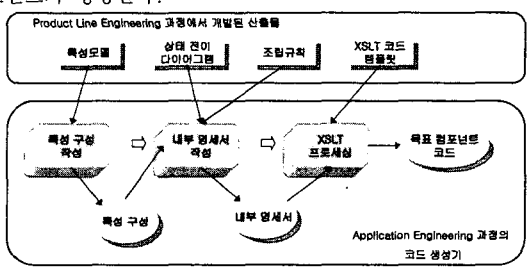
Message: addToBuffer
Parameters: Item to add to buffer
From State:
Open
Empty
Desc: Add Item to buffer using Circular Queue
To State:
1- if State = Empty **change to** Open
2- if # of Items >= buffer_size
change to Full
Return: Null

Message: takeFromBuffer
Parameters: Item to add to buffer
From State:
Open
Full
Desc: Take Item from buffer using Circular Queue
To State:
1- if State = Full **change to** Open
2- if # of Items <= 0
change to Empty
Return: Item from buffer

[그림 1] BoundedBuffer의 상태 전이 다이어그램

3. 병렬 소프트웨어 프러덕트 라인 지원 컴포넌트

병렬 소프트웨어 프러덕트 라인 구축 및 컴포넌트 코드 생성 과정 및 산출물은 [그림 2]와 같다. 본 도구는 소프트웨어 프러덕트 라인 공학을 통해 개발 초기에 공통적인 부분과 가변적인 부분을 명확히 하여 특성 모델, 상태 전이 다이어그램, 조립규칙, XSLT 코드 템플릿과 같은 소프트웨어 자산을 구축한다. 재사용시 다양한 요구 사항에 따라 특성 구성을 작성한 후 XSLT 프로세싱을 거쳐 커스터마이징된 목표 컴포넌트가 생성된다.

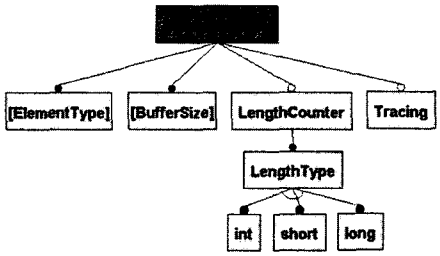


[그림 2] 컴포넌트 생산 과정 및 각 과정에서의 산출물

3.1 특성 모델을 통한 가변성지원

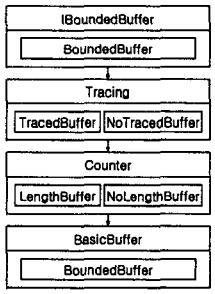
가변성을 지원하는 컴포넌트는 특성 소프트웨어 생산 시 코드 변화를 수행해야 하는 컴포넌트를 말한다. 가변성 지원은 내부 커스터마이징을 통해서 이루어지며, 본 논문에서는 GenVoca 아키텍처[2]와 XSLT를 응용한 자동 생성 기법[3]을 이용한다.

[그림 4]는 본 논문의 예제로 사용되는 *BoundedBuffer*의 특성 모델이다. 사각형은 각각의 특성을 나타내며, 사각형 위의 검은 원은 필수적 특성(ElementType, BufferSize)을, 하얀 원은 선택적 특성(LengthCounter, Tracing)을, 원호 아래에 위치한 특성들은 택일적 특성(int, short, long), 각 괄호로 나타내어진 특성은 비결정 인자 특성(ElementType, BufferSize)을 나타낸다.



[그림 4] BoundedBuffer 특성 모델

본 논문에서 컴포넌트 패밀리는 계층구조를 이루며, 상위 계층에 속하는 구현 부품은 바로 하위 계층에 존재하는 구현 클래스 중 하나를 상속받아 구현된다. 예를 들어, [그림 3]에서 *Tracing* 계층에 속하는 구현 부품 *TracedBuffer*는 하위 계층인 *Count*에 존재하는 *LengthBuffer* 또는 *NoLengthBuffer*의 구현 클래스 중 하나를 상속 받아 구현된다.



[그림 3] BoundedBuffer의 계층구조

특정 컴포넌트 조립 시에 각 계층에서 어떤 구현부품이 선택될지는 조립 규칙에 의해 정해진다. 예를 들어, *BoundedBuffer* 컴포넌트 조립 시 *NoLengthBuffer*가 포함될지 *LengthBuffer*가 포함될지는 [표1]과 같은 컴포넌트 조립 규칙에 의해 결정된다.

컴포넌트 카테고리	구현 부품	특성 구성 값
Counter	LengthBuffer	BoundedBuffer.Counter =LengthBuffer
	NoLengthBuffer	BoundedBuffer.Counter =NoLengthBuffer

[표 1] BoundedBuffer 컴포넌트 조립 규칙(일부분)

컴포넌트 생성 도구는 특성모델, 계층구조, 조립규칙에 따라 내부 명세서를 생성하고, 컴포넌트를 구성하는 구현 컴포넌트들의 코드 템플릿을 이용하여 각각을 위한 코드를 자동 생성한다. [그림 5]는 *Counter* 카테고리의 *LengthBuffer* 구현 컴포넌트를 위한 XSLT 코드 템플릿을 보여주고 있다. 계층 구조에 따라 구현 부품을 가져오고, 특성모델에서 선택한 *LengthType*을 가져오는 것을 알 수 있다.

```
<xsl:template match="Specification">
public class <xsl:value-of
      select="Layers/Component[@category='Counter']/ImplCom"/> <xsl:apply-
      templates select="Layers/Component[@category='Counter']"/> implements
      Counter {
      <xsl:value-of select="Features/Feature[name='LengthType']//value"/>
      length_;

      public void <xsl:value-of
      select="Layers/Component[@category='Counter']/ImplCom"/>(<xsl:apply-
      templates select="Features"/> cnt) {
            for(int i=0; i &lt; cnt; i++) length_ = length_ + 1;

            System.out.println ("Buffer Length = " + length_);
      }
}
</xsl:template>
```

[그림 5] LengthBuffer의 XSLT 코드 템플릿

[그림 6]은 특성 구성과 조립 규칙에 따라 자동 생성된 코드이다. ***LengthBuffer***는 하위 계층인 ***BasicBuffer***의 구현 클래스 ***BoundedBuffer***를 상속 받아 구현되며, ***LengthType***은 ***int*** 형으로 생성되었음을 보여준다.

```
public class LengthBuffer
 extends BoundedBuffer implements Counter {
      int length_;
      public void LengthBuffer(int cnt) {
            for(int i=0; i < cnt; i++) length_ = length_ + 1;
            System.out.println("Buffer Length = " + length_);
      }
}
```

[그림 6] LengthBuffer의 생성된 코드

3.2 상태 전이 다이어그램을 이용한 동기화 지원

2장에서 서술한 바와 같이 병렬 컴포넌트를 생성하기 위해서는 동기화 지원이 필수적이다. 본 도구에서는 상태 전이 다이어그램을 통해 동기화 기술을 지원하고자 한다.

병렬 컴포넌트 생성을 위해 프러덕트 라인 공학 과정에서는 특성 구성 이외에 상태 전이 다이어그램을 작성한다. [그림 7]은 상태 전이 다이어그램을 XML 문서로 표현한 것 이다. ***addToBuffer***를 수행할 때 아이템이 없는 조건(condition)에 부합되면, ***Buffer***를 ***Open*** 상태로 전이하도록 정의되어 있다.

```
<StateTransitionDiagram>
      <Transition>
            <Event method="addToBuffer">
                  <fromState1>OPEN</fromState1>
                  <fromState2>EMPTY</fromState2>
                  <toState condition="noItem">OPEN</toState>
                  <toState condition="overItem">FULL</toState>
            </Event>
      </Transition>
      <Transition>
            <Event method="takeFromBuffer">
                  <fromState1>OPEN</fromState1>
                  <fromState2>FULL</fromState2>
                  <toState condition="overItem">OPEN</toState>
                  <toState condition="noItem">EMPTY</toState>
            </Event>
      </Transition>
</StateTransitionDiagram>
```

[그림 7] 상태 전이 다이어그램의 XML 문서

본 도구는 내부 명세서 작성 과정에서 앞서 작성한 특성 구성과 상태 전이 다이어그램, 조립규칙을 포함하는 내부 명세서 생성한다.

[그림 8]은 병렬 컴포넌트인 ***BoundedBuffer*** 컴포넌트 코드 생성을 위한 XSLT 코드 템플릿의 일부를 보여준다. 전제조건 확인에 필요한 상태 정보는 내부 명세서에서 얻어오며, 전제조건을 만족하지 않으며 ***addToBuffer()*** 를 호출한 쓰레드를 대기하도록 함을 알 수 있다. 이와 같은 방식으로 동기화를 지원하는 코드를 생성한다.

```
public synchronized void addToBuffer(int value) {
  // Pre-condition processing (method guard).
<xsl:if test="StateTransitionDiagram/Transition/Event[@method='addToBuffer']">
  while(! ((bufferState == <xsl:value-of select="StateTransitionDiagram/Transition
            /Event[@method='addToBuffer']/fromState1"/>) ||
        (bufferState == <xsl:value-of select="StateTransitionDiagram/Transition
            /Event[@method='addToBuffer']/fromState2"/>))) {
     try {
        wait();
     } catch (InterruptedException e) {
     }
  }
</xsl:if>
   ...
}
```

[그림 8] BoundedBuffer XSLT 코드 템플릿

[그림 9]는 특성 구성과 상태 전이 다이어그램에 의해 자동 생성된 병렬 컴포넌트 ***BoundedBuffer***의 소스 코드 일부를 보여준다. 동기화된 ***addToBuffer*** 메소드에서는 상태에 따라 아이템 추가 작업을 요청한 쓰레드의 실행 여부를 결정하여 소프트웨어의 동기화 및 병렬성을 지원한다.

```
public synchronized void addToBuffer(int value) {
      // Pre-condition processing (method guard).
      while(! ((bufferState == OPEN) ||
            (bufferState == EMPTY ))) {
         try {
            wait();
         } catch (InterruptedException e) {
         }
      }

      // Processing done to add item to buffer
      values[lastItem] = value;
      lastItem = (lastItem + 1) % bufferSize;
      numberOfItems = numberOfItems + 1;

      // Post-condition processing to change states
      if ((bufferState == OPEN) && (numberOfItems >= bufferSize)) {
         bufferState = FULL;
         notifyAll();
      }
      else if (bufferState == EMPTY) {
         bufferState = OPEN;
         notifyAll();
      }
}
```

[그림 9] BoundedBuffer 컴포넌트를 구현한 예(일부분)

4. 결론 및 향후 과제

본 논문에서는 병렬 소프트웨어 프러덕트 라인 구축에 있어서 특성 모델을 통해 가능적 가변성을 지원하고 상태 전이 다이어그램을 통해 동기화을 지원하는 컴포넌트의 코드 자동 생성하는 도구를 제안하였다. 본 연구 결과는 병렬성이 중요시되는 실시간 임베디드 소프트웨어 프러덕트 라인 구축에 활용될 수 있을 것이다. 향후 과제로는 보다 다양한 병렬 컴포넌트 및 임베디드 소프트웨어 프러덕트 라인에 본 논문의 기법과 도구를 적용해보는 것이다.

참고문헌

[1] P.Clements and L.Northrop," Software Product Lines: Practices and Patterns" ,Addison-Wesley, 2002.

[2] Krzysztof Czarnecki and Ulrich W.Eisenecker, " Generative Programming. Methods, Tools, and Applications" , Addison-Wesley, 2000.

[3] J.C.Cleavland, " Program Generators with XML and Java" , Prentice Hall, 2001.

[4] H.Zhang, S.Jarzab and S.M.Swe, " XVCL Approach to Separating Concerns in Product Family Assets" , Proceedings of International Conference on GCSE,2001.

[5] Charles W Kann, " Creating Components:Object Oriented, Concurrent, and Distributed Computing in Java" , Auerbach Publications, 2004.