FSP Model을 이용한 C&C View 아키텍처의 검증

김정호
SK C&C S/W공학 센터

강성원
한국정보통신대학교 공학부

An Approach to Verifying C&C View Architecture with FSP Modeling

Jungho, Kim
SK C&C
S/W Engineering Center
E-mail : kimjh@skcc.com

Sungwon Kang
School of Engineering
Information and Communications University
E-mail : kangsw@icu.ac.kr

요 약

시스템의 동적 요소를 기술하고 분석하기 위해 C&C view 아키텍처를 주로 사용하지만 시스템의 실행 순서나 상태에 대한 정보가 부족하기 때문에 단지 C&C view 아키텍처로만 시스템의 동적 요소를 분석하기가 어렵다. FSP model은 시스템의 동작을 표현하기 위한 모델링 언어로써 시스템의 동작을 모델링 할 뿐만 아니라 LTSA라는 FSP 검증 툴을 이용하여 시스템의 특성을 자동으로 파악하기에도 용이한 특성을 가지고 있어 시스템 동적 요소 분석에 도움을 준다. 하지만 관련 정보를 얻기 어려워 FSP model을 구현하는데 어려움이 있다. 이 논문은 C&C view 아키텍처와 요구 사항(Use Case Scenario 혹은 Sequence diagram)을 근간으로 시스템의 동적 요소를 FSP로 모델링 하는 방법을 정의한다. 또한 LTSA 툴을 이용하여 자동적으로 시스템의 특성을 검증할 수 있고, 이를 통해 시스템 구현 전에 시스템의 에러를 찾아내고 해결하는 도움을 줄 수 있다. C&C view 아키텍처는 시스템의 컴포넌트와 커넥터에 대한 정보를 제공하여 주고 요구 사항은 소프트웨어 시스템의 행동에 대한 정보를 제공하여 줄 수 있다. 만약 우리가 컴포넌트와 커넥터에 대한 정보가 있고 시스템 동적 요소에 대해 알고 있다면 우리는 이것을 기반으로 메시지 흐름도를 구현할 수 있다. 메시지 흐름도는 Harel, Kugler 와 Pnueli에 의해 개발된 Synthesized 알고리즘을 이용하여 Statechart로 만들어지고 이것은 바로 FSP model로 변환될 수 있다. 우리가 소프트웨어 시스템에 대한 FSP model을 가지고 있다면 이것을 기반으로 시스템 행동 특성을 체크할 수 있으며 이는 향후 발생할 수 있는 결함을 미리 찾아내는데 유용한 도움이 될 수 있다. 우리는 간단한 예제를 통해 본 논문의 이론을 수립하였으며 이를 기반으로 실제 어플리케이션에 적용하여 그 효용성을 검증하였다.

1. 서론

1.1 연구 동기

C&C view 아키텍처(Component & Connector view architecture)는 소프트웨어 시스템의 런타임 아키텍처를 기술 및 분석하기 위해 사용되는 제반 모델링 언어 중 하나이다. 그러나 C&C view는 런타임 아키텍처 용도로 사용됨에도 불구하고 그것만으로는 소프트웨어 시스템의 동작 및 상태를 쉽게 확인할 수 없다는 단점을 안고 있다. 본 연구에서는 소프트웨어 시스템의 동작 및 속성 요소를 기술하기 위하여, 유한 상태 프로세스(FSP: Finite State Process) 모델링 언어를 사용한다. FSP를 활용할 경우, 컴포넌트의 동작 순서 및 상태를 쉽게 확인할 수 있고 또한 컴포넌트의 FSP 모델에 기초하여 컴포넌트가 갖는 속성을 쉽게 검증할 수 있다.

본 논문에서는 소프트웨어 시스템의 동작을 기술하기 위한 목적에서 C&C view 아키텍처 및 제반 요구 사항(혹은 유즈케이스)을 하나의 FSP 모델로 번역하기 위한 방법을 제안하고 있다. 또한 FSP 모델을 구축하면 소프트웨어 시스템을 구현하기에 앞서 병행성(concurrency) 등의 소프트웨어 시스템의 제반 동작 속성을 FSP 모델을 통하여 확인해볼 수 있다.

1.2 의의

본 연구의 첫 번째 의의는 하나의 소프트웨어 시스템에서 C&C view 아키텍처 및 제반 요구 사항을 갖는 컴포넌트 및 커넥터들(connectors)의 FSP 모델을 구성하는데 있다. 어떤 소프트웨어 시스템의 유즈케이스 또는 sequence diagram 등 시나리오 단편 및 C&C view 아키텍처의 형태로 된 제반 시스템 요구 사항이 있을 경우, 메시지 순서도(MSC)를 얻을 수 있으며 MSC는 합성 알고리즘(synthesized algorithm)을 통해 상태도(state chart)로 변환할 수 있다[2]. 이러한 상태도는 하나의 FSP 모델로 쉽게 변환할 수 있다. 즉, 커넥터 및 컴포넌트의 동작은 FSP를 통해 모델링 할 수 있는 것이다.

본 연구의 두 번째 의의는 LTSA(Labeled Transition Systems Analyzer)라는 FSP의 분석기를 이용해 소프트웨어 시스템의 속성을 파악한다는데 있다. 제반 컴포넌트 및 커넥터를 보여주는 FSP 모델이 있을 경우, 제반 요구 사항들로부터 소프트웨어 시스템에 요구되는 속성들을 얻을 수 있고 이러한 속성들은 FSP 속성 프로세스(property process)로 모델링 될 수 있다. 따라서 LTSA 같은 FSP 도구에서는 소프트웨어 시스템의 FSP 모델을 이용해 제반 컴포넌트 및 커넥터의 속성을 파악할 수 있는 것이다. 특히 시스템 속성들의 검증을 소프트웨어 시스템을 구현하기에 앞서 LTSA를 통해 확인할 수 있다는데 큰 의의가 있다.

1.3 개요

본 논문의 나머지 내용 구성은 다음과 같다. 제 2 장에서는 본 논문의 연구 절차에 대해 설명하고 있으며, 제 3 장에서는 제반 컴포넌트 및 커넥터를 보여주는 FSP 모델에 관한 설명한다. 이를 위하여 하나의 적용 사례를 살펴본 다음, LTSA를 사용하여 FSP 속성을 활용해 소프트웨어 시스템의 제반 속성을 파악하고 있다. 마지막으로 제 4 장에서는 본 논문의 의의를 설명하고 결론을 맺는다.

2. FSP 모델을 이용하여 C&C view 아키텍처를 검증하는 방법

이 논문에서 우리는 C&C 아키테처를 FSP 모델로 어떻게 표현하는지 살펴보고 이러한 FSP 모델을 LTSA라는 툴로 시스템 행위를 체크하는 방법에 대해서 알아본다. 그림 1은 이러한 방법에 대한 흐름을 보여준다.

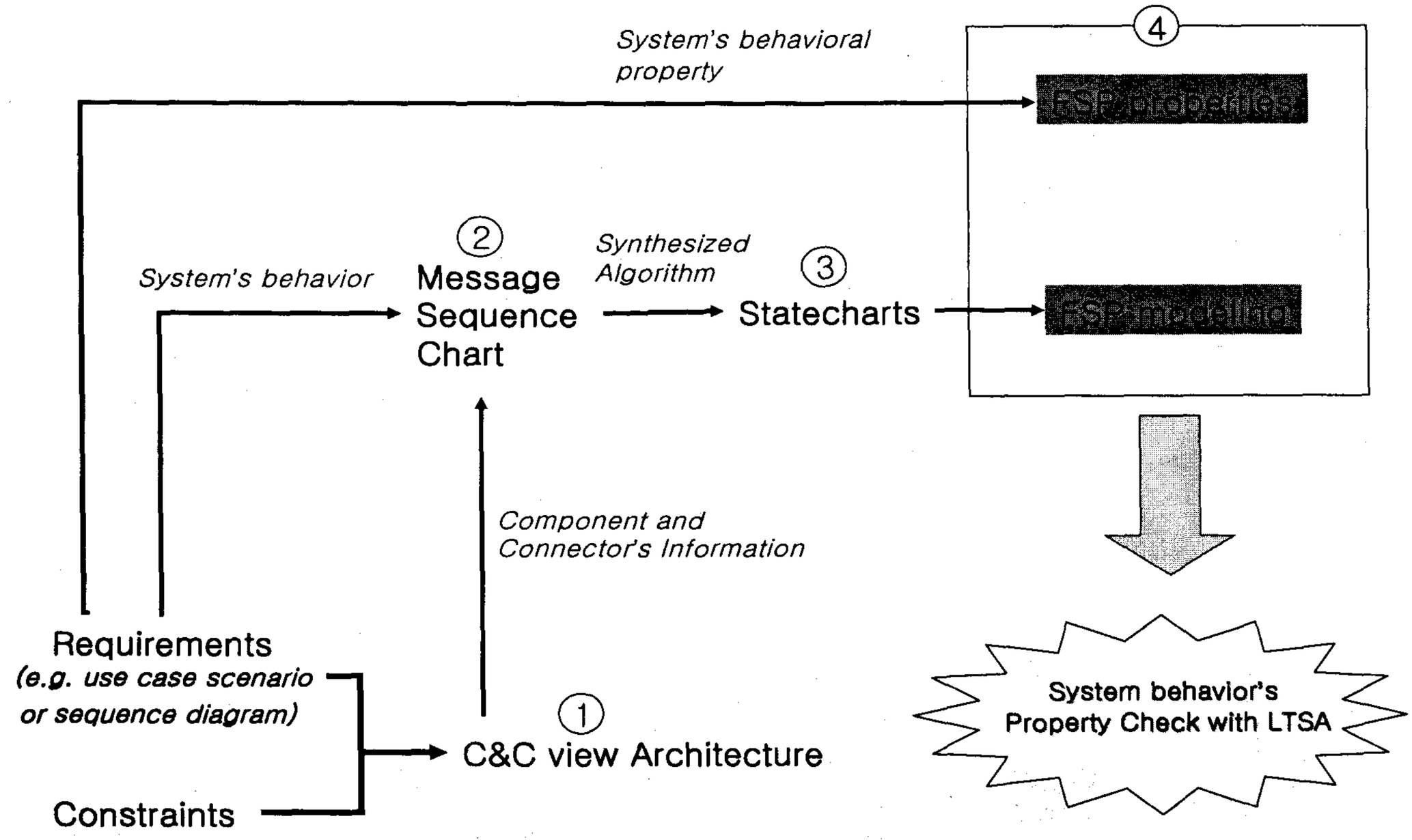


그림 1. 시스템 행위를 체크하는 방법의 순서

그림 1에 대한 자세한 소개는 아래와 같다.

① C&C view 아키텍처를 요구사항과 제약을 통하여 개발한다. [1, 2]

② C&C view 아키텍처는 컴포넌트와 커넥터를 묘사하고 시스템의 런타임 정보를 제공하여 준다. 또한 요구사항은 시스템의 흐름을 표시한 Message Sequence Chart (MSC)을 만들 수 있는 정보를 제공한다. 예를 들어 유즈케이스 혹은 sequence diagram 을 통하여 우리는 원하는 MSC 를 얻을 수 있다. [3, 4]

③ MSC 를 만들어지면, synthesis 알고리듬을 이용하여 MSC 를 컴포넌트와 커넥터의 statechart 를 만들수 있다. 이러한 컴포넌트와 커넥터의 상태 정보는 바로 FSP 로 표현될 수 있다. [2, 3, 4]

④ 시스템의 특성은 요구사항을 통하여 FSP property 로 표시할 수 있다. 더욱이 컴포넌트와 커넥터에 대한 행위와 상태를 표시한 FSP 모델을 가지고 있으면, 이러한 FSP 모델과 시스템의 특성을 표시한 FSP property 를 이용하여 LTSA 툴을 이용하여 시스템의 개발 전에 시스템의 행위를 체크하여 시스템의 아키텍처를 검증할 수 있다.

3장에서는 이러한 단계들을 실제의 예를 통하여 보여준다.

3. 어플리케이션 예제

여기서 사용하는 예제는 TTCN-MP라는 테스트 언어로 작성된 테스트 케이스를 실행이 가능한 C 언어로 변경하는 어플리케이션이다. 이 시스템은 실제로 프로토콜 테스트에 사용될 수 있는 실제의 어플리케이션이다. 그림 2는 변환 어플리케이션의 context diagram이다. 테스트 엔지니어는 TTCN-MP 언어로 작성된 테스트 케이스를 작성하고 변환 어플리케이션을 통하여 C 언어로 변경을 한다. 사용자는 테스트 대상에 맞게 조금의 변경을 거쳐 실제로 테스트 대상(예, 블루투스)을 테스트하게 된다.

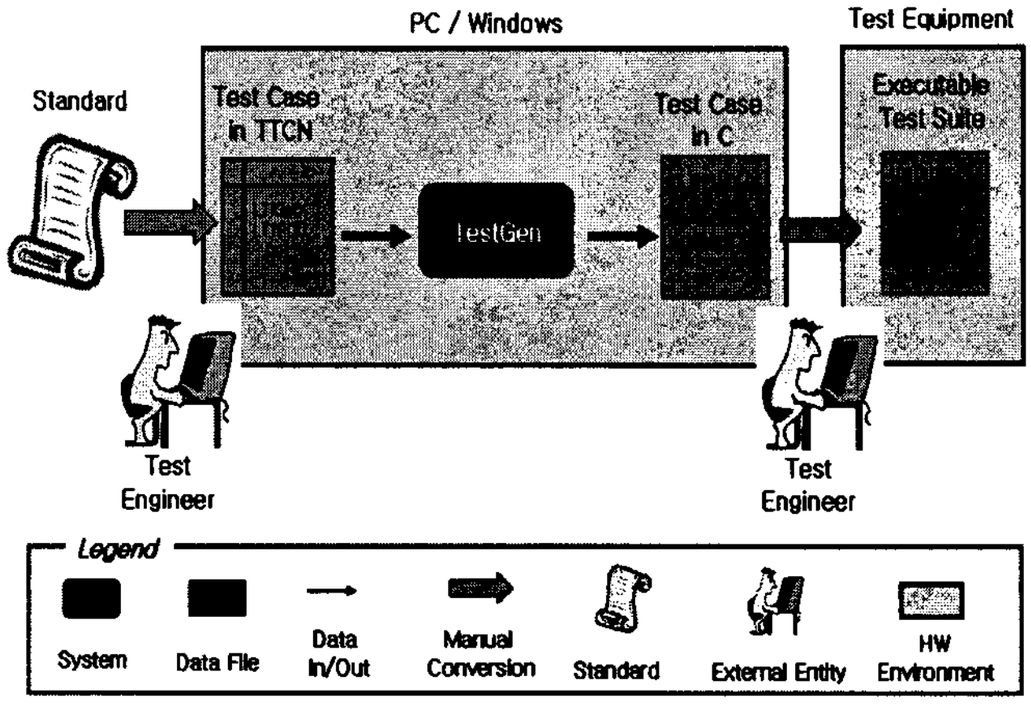


그림 2. 변환 어플리케이션의 Context diagram

이 경우에 사용되는 커넥터는 단순한 동기화 커넥션(즉. call-return과 data flow 타입)이므로 FSP 모델에서는 제외하고 컴포넌트만 적용하기로 한다.

3.1 C&C view 아키텍처 수립

아래는 변환 어플리케이션의 요구사항 및 제약의 일부를 예시로서 제시한 것이다. 요구사항 중 기능 요구사항은 TTCN-MP의 문법을 기준으로 작성되었고 품질 요구사항과 제약 사항은 고객과 상의하여 작성하였다. 이러한 요구사항 및 제약을 통하여 C&C view 아키텍처를 작성할 수 있고 시스템의 행동 특성을 추출할 수 있다.

기능 요구사항

FR1. ‘ timer’ 행위를 개발해야 한다.

FR2. ‘ test suite type definitions’ 행위를 개발해야 한다.

….

품질 요구사항

QR1. 변경된 C 언어를 사용자가 쉽게 읽을 수 있도록 한다.

QR2. 기본 테스트 케이스를 기준으로 변경 시간은 3초 내에 해결되어야 한다.

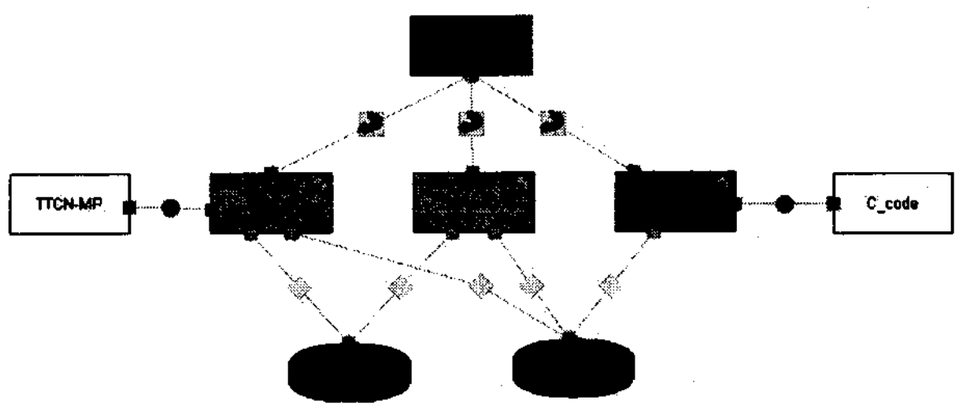
…

제약 사항

C1: 사용자는 변환 어플리케이션을 본인의 PC에서 작동할 수 있어야 한다.

…

이러한 요구사항 및 제약사항을 C&C view 아키텍처를 작성할 수 있지만 변환 어플리케이션의 참조 아키텍처는 아키텍처를 선택함에 있어서 매우 중요한 역할을 할 수 있다. 자바를 이용한 변환 어플리케이션의 참조 아키텍처 중 가장 유명한 것은 ‘ *Javacc*’ 이다. 이것은 자바 언어로 개발된 변형 어플리케이션을 컴화일 해주는 어플리케이션으로서 자바 컴화일러를 만들 때 널리 사용되는 어플리케이션이다. 이것을 기본으로 한 변형 어플리케이션의 C&C view 아키텍처는 그림 3과



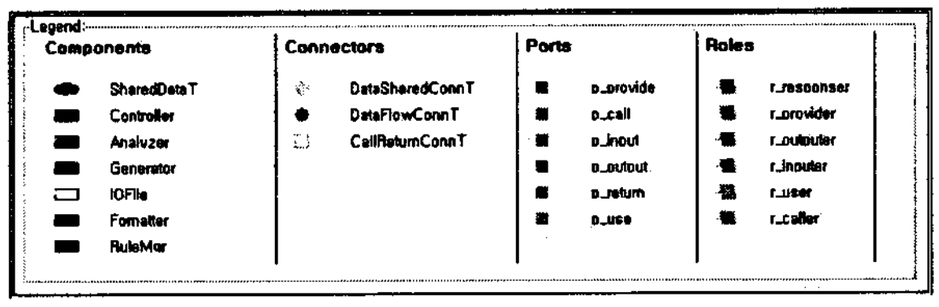


그림 3. 변형 어플리케이션의 C&C view 아키텍처

그림 3은 변형 어플리케이션의 C&C view 아키텍처는 시스템의 런타임 정보를 제공한다. 여기에서 표시되는 컴포넌트는 Parser와 Semantic Analyzer (이하 Semantic), Code Generator (이하 Code_Gen), Symbol_Table, Tree, 그리고 Main Controller (이하 Main_Controller)가 있다. Main_Controller는 변환 프로세스를 위하여 Parser, Semantic, Generator를 차례로 실행시키는 역할을 한다. Parser는 TTCN-MP로 작성된 테스트 케이스를 읽어들여 이것을 토큰으로 구분한다. Symbol_Table은 이러게 구분된 토큰을 컴화일 과정 사용하는 심볼로 매핑하는 테이블을 관리한다. Parser는 다시 Symbol_Table로 문법을 체크한 후, 소스를 구성하는 트리를 구성한다. Semantic은 의미상으로 트리가 맞는지 확인한다. 마지막으로 Code_Gen은 트리의 내용을 C 언어로 변형한다.

3.2 MSC 작성

그림 4는 요구사항 및 제약사항과 참조 아키텍처를 기준으로 작성된 변형 어플리케이션의 C&C view 아키텍처를 기준으로 MSC를 작성한 그림이다.

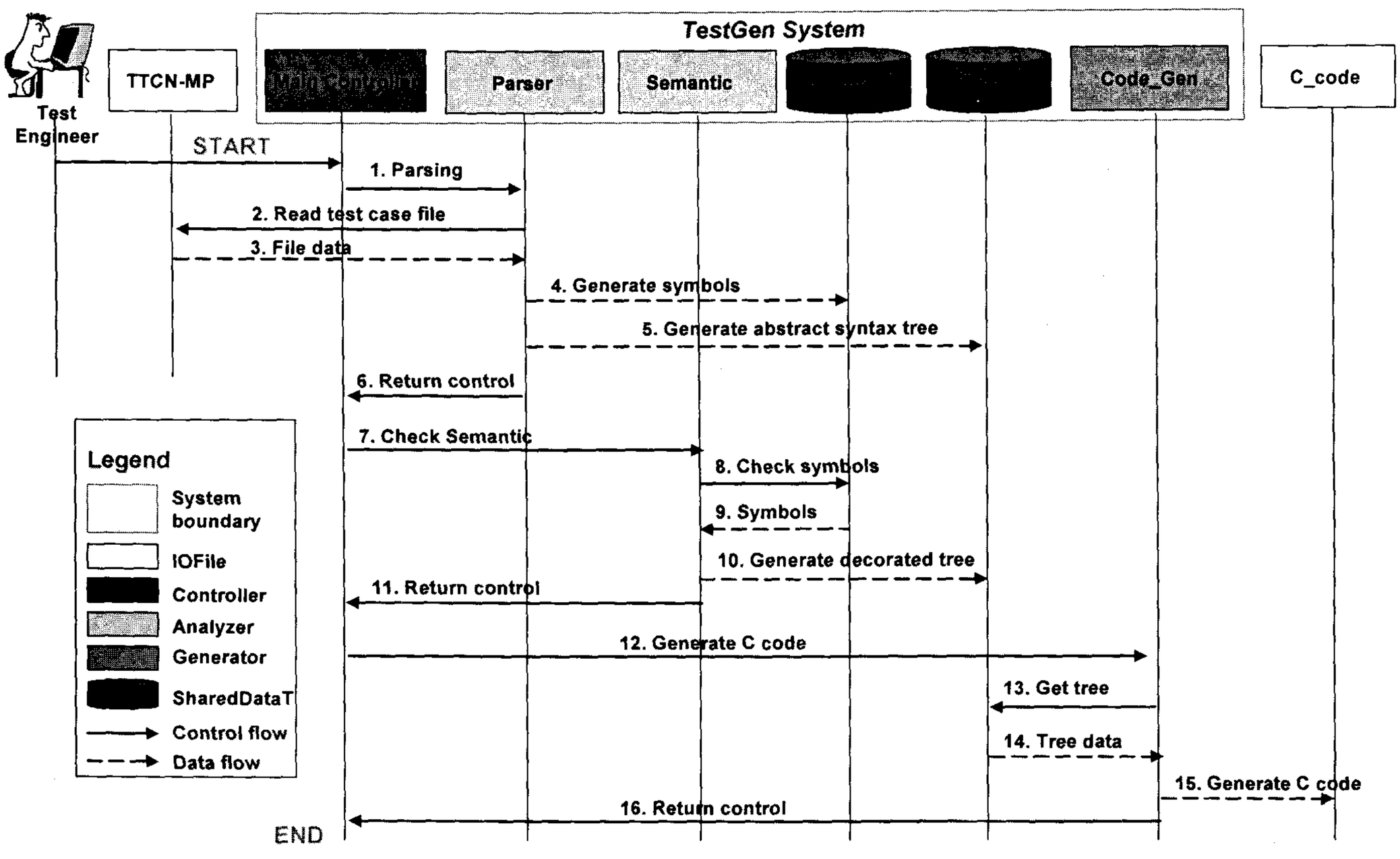


그림 4. 변형 어플리케이션의 MSC

MSC 의 컴포넌트는 C&C view 아키텍처에서 가지고 왔고 Parser 와 Semantic, Code_Gen 의 흐름은 전형적인 컴화일러를 기준으로 작성되었다.

3.3 FSP 모델과 Statechart 작성

우리는 Synthesized 알고리듬을 이용하여 MSC로부터 FSP 모델을 구축할 수 있다[2]. 앞에서 애기했듯이 커넥터는 간단한 역할을 하여 FSP 모델을 만들 필요가 없어 컴포넌트에 한하여 FSP 모델을 만들었다. 우선, Main_Controller에 대한 FSP 모델을 만들어 보자. MSC에서 Main_Controller 컴포넌트는 Parser를 호출한다. 그리고 난 후, Semantic을 호출하고 마지막으로

Code_Gen을 호출한다. 그러므로 Main_Controller의 statechart는 그림 5와 같다.

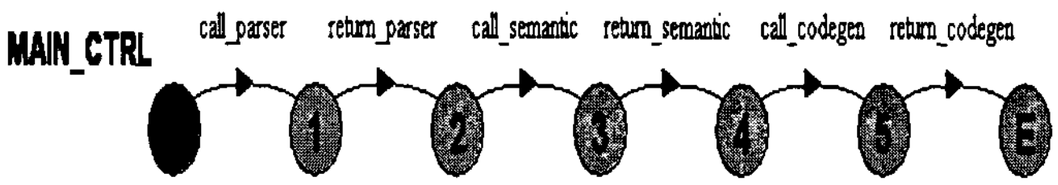
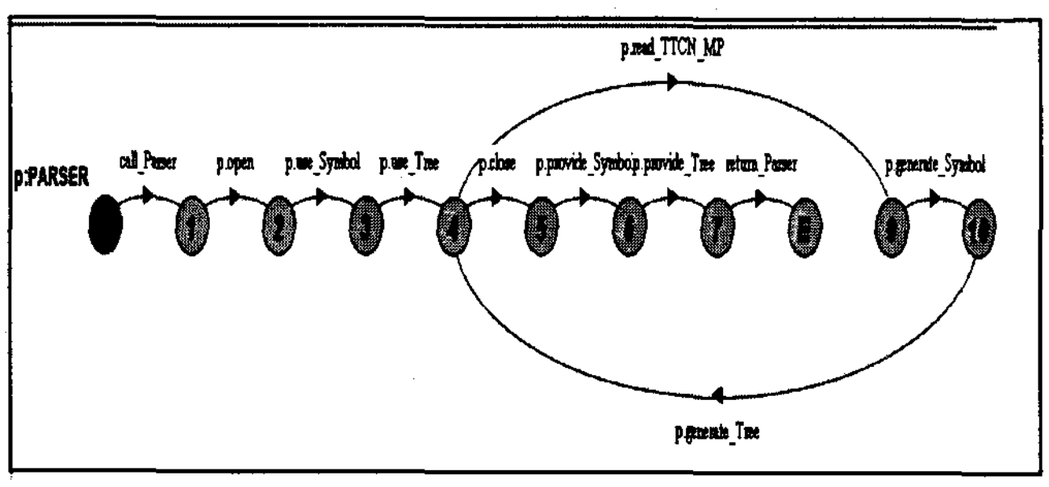


그림 5. Main_Controller 의 statechart

Statechart가 표시되면 이것은 바로 아래와 같은 FSP 모델로 변형될 수 있다.

MAIN_CTRL = (callToParser -> returnFromParser -> callToSemantic -> returnFromSemantic -> callToCodegen -> returnFromCodegen -> END).

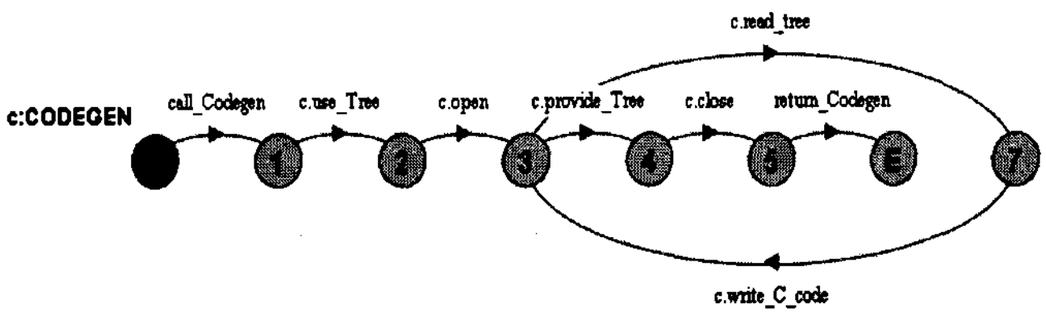
다음으로 Parser 역시 그림 6과 같이 statechart를 구성하고 이를 FSP 모델로 표현할 수 있다. Parser가 Main_Controller에 의해 호출될 때 이것은 TTCN-MP를 우선 호출하여 정보를 받아들이고 이를 구분하여, Symbol_Table을 호출한다. 그 이후, Tree에 정보를 제공하고 행위를 마친다.



PARSER = (callFromMainCotroller -> CallToTTCNMP-> callToSymbol -> callToTree -> PARSER_WORK),

PARSER_WORK = (readFromTTCNMP -> generateToSymbol -> generateToTree-> PARSER_WORK | returnFromTTCNMP -> returnFromSymbol -> returnFromTree -> return -> END).

그림 6. Parser 의 statechart 와 FSP 모델



CODEGEN = (callFromMainController -> CallToTree -> CallToCCode -> CODEGEN_WORK),

CODEGEN_WORK = (readFromtree -> writeToCCode -> CODEGEN_WORK | returnFromTree -> returnFromCCode-> return -> END).

그림 7. Code_Gen 의 statechart 와 FSP 모델

Symbol_Table과 Treesms 같은 상태를 유지하므로 하나의 statechart와 FSP 모델로 표현이 가능하다.

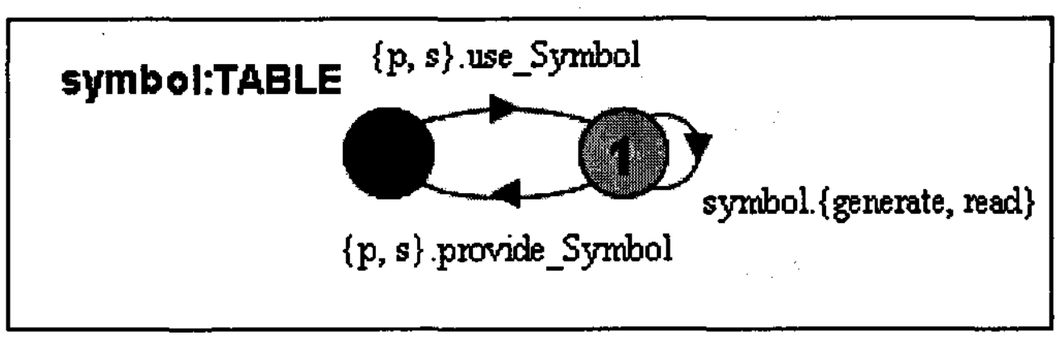


TABLE = (call-> TABLE_WORK),

TABLE_WORK = (read -> TABLE_WORK | generate -> TABLE_WORK | return -> TABLE).

그림 8. Symbol_Table 과 Tree 의 statechart 와 FSP 모델

모든 컴포넌트의 FSP 모델이 마무리 되면 우리는 시스템의 FSP 모델을 아래와 같이 구성할 수 있다.

```
||SYSTEM = (ttcnmp:FILE || ccode:FILE || symbol:TABLE
|| tree:TABLE || p:PARSER || s:SEMANTIC || c:CODEGEN
||          MAIN_CTRL)          /{callToParser/p.call,
returnFromParser/p.return,callToSemantic/s.call,
returnFromSemantic/s.return,callToCodegen/c.call,
returnFromCodegen/c.return,p.callToSymbol/symbol.call,
p.returnFromSymbol/symbol.return,p.callToTree/tree.call,
p.returnFromTree/tree.return,s.callToSymbol/symbol.call,
s.returnFromSymbol/symbol.return,s.callToTree/tree.call,
s.returnFromTree/tree.return,c.callToTree/tree.call,
c.returnFromTree/tree.return}.
```

3.4 LTSA 를 이용한 시스템 행위 특성 파악

요구사항과 제약상에서 우리는 시스템의 행위 특성을 FSP 로 표시할 수 dlTek. 아래의 FSP 특성은 TTCN-MP 표준으로 부터 C 언어를 발생하는 방식으로 qusgud 어플리케이션이 반드시 지켜야할 특성이다.

```
property P = ( p.read_TTCN_MP -> p.generate_Symbol ->
p.generate_Tree -> s.check_Symbol -> c.read_Symbol ->
c.write_C_code -> P).
```

LTSA를 이용하여 시스템의 행위를 체크해본 결과 Code_Gen이 symbol_Table을 호출할 때 그림 9와 같은 에러가 발생함을 알 수가 있다.

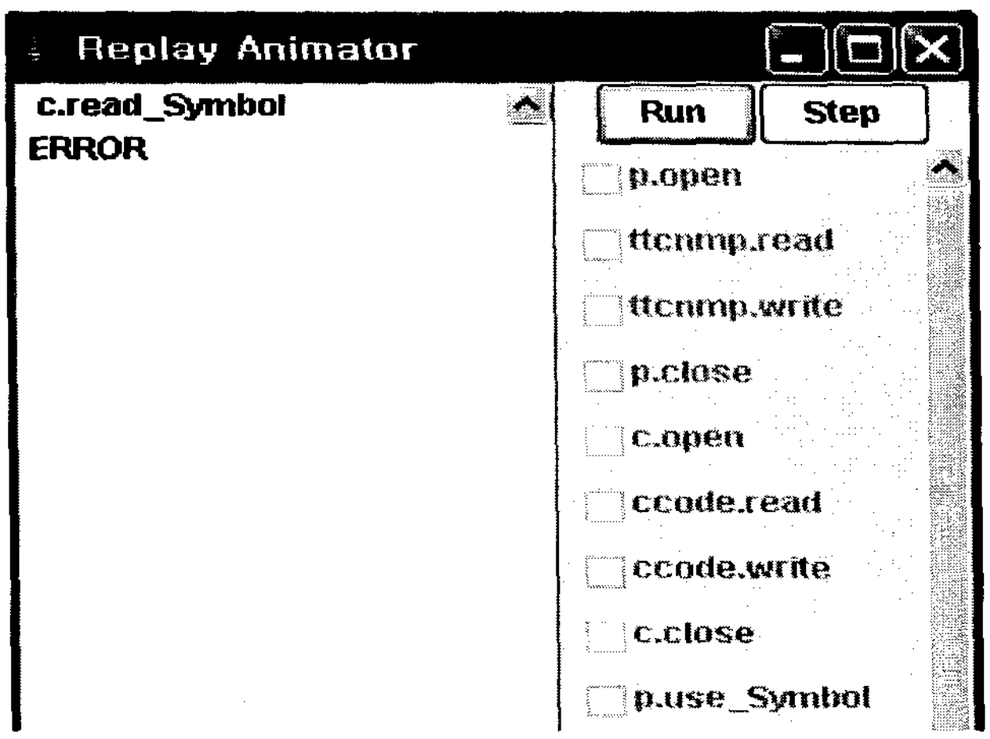


그림 9. LTSA 를 이용한 변형 어플리케이션의 FSP 특성 체크

FSP 모델 체크 결과 시스템에 치명적인 데드락이 발생함을 알 수 있다. 이것은 Symbol_Table Parser에 의해 업데이트 되는 동안 Code_Gen이 Symbol_Table의 정보 사용을 자제하도록 아키텍처를 작성해야 하는데 그렇게 하지 못해서 발생한 문제이다. 따라서, 변형 어플리케에션의 Main_Controller이 이러한 컴포넌트의 정합성 관리를 해야 한다. 결론적으로, 우리가 단순하게 C&C view 아키텍처와 MSC 모델만을 가지고 이러한 정합성 체크를 하기는 어렵다. 그러나, 변형 어플리케이션의 FSP 모델을 만들 수 있다면, 우리는 변형 어플리케이션의 행위 특성을 가지고 LTSA라는 툴을 사용하여 시스템 개발이 발생하기 전에 발생할 수 있는 시스템의 에러를 미리 체크할 수 있는 것이다.

4. 결론

모듈 뷰(Module view), C&C view 및 배치 뷰(Allocation view) 등의 아키텍처 뷰만 기술할 경우, 소프트웨어 시스템의 아키텍처 동작을 표현할 수는 있지만 검증할 수 있는 방법은 없다[1]. 따라서 시스템 동작을 기술하기 위해 때로는 시간적 순서에 따라 개체들의 상호작용을 보여주는 UML 메시지 순서도(MSC)를 작성하기도 한다. [3] 그러나 이러한 MSC는 소프트웨어 시스템의 컴포넌트 및 커넥터 각각에 대한 동작을 기술하는데 있어서 한계를 안고 있는데, 그 이유는 MSC의 경우 소프트웨어 시스템의 자체 동작을 기술하는데 주안점을 두고 있기 때문이다. 한편 MSC는 병행성(concurrency) 문제 등의 시스템 속성을 확인하는데 있어 취약한 단점을 갖고 있다. MSC에서 하나의 FSP 모델을 얻을 수 있다면 이러한 문제점들을 해결할 수 있다[2]. 본 연구에서는 하나의 소프트웨어 시스템에서 제반 컴포넌트 및 커넥터를 보여주는 상태도를 얻기 위해 합성 알고리즘을 활용하고 있는데[2],

이러한 상태도를 이용하면 FSP 모델을 쉽게 기술할 수 있다[2].

어떤 소프트웨어 시스템에서 제반 컴포넌트 및 커넥터를 보여주는 FSP 모델이 이미 존재할 경우, 제반 요구 사항 혹은 용례 시나리오에서 병행성 문제 등 소프트웨어 시스템의 FSP 속성들을 만들 수도 있다. 따라서 LTSA에서 FSP 속성을 이용하면 제반 컴포넌트 및 커넥터의 속성을 파악할 수 있다. 결국 FSP 모델을 활용하면 소프트웨어 시스템을 구현하기에 앞서 그것의 결함을 미리 파악할 수 있게 된다.

[참고 문헌]

[1] Constantinos A. Constantinides, and Tzilla Elrad. On the requirements for concurrent Software Architectures to Support Advanced separation of Concerns. OOPSLA 2000 Workshop on Advanced Separation of Concerns in OO Systems. The 14th European Conference on Object-Oriented Programming (ECOOP 2000) Workshop

[2] Gregory Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-Oriented Programming. In Proceedings of ECOOP ' 97. LNCS 1241. Springer-Verlag, pp. 220-242.

[3] Kim Mens, Cristina Lopes, Badir Tekinerdogan, and Gregory Kiczales. Aspect-Oriented Programming. Report of the ECOOP ' 97 Workshop for Aspect-Oriented Programming.

[4] Sinan Si Alhir " Extending UML"