정제 프로세스 상에서 컴포넌트 모델의 합성과 검증

경북대학교 전자전기컴퓨터학부
소프트웨어 안전공학 연구실

장 훈

2010. 01. 08

발표순서

1. 서론
2. 추상 컴포넌트의 행위일관성 검증
3. 특성 기반 검증
4. 향후 과제

1.1 서론

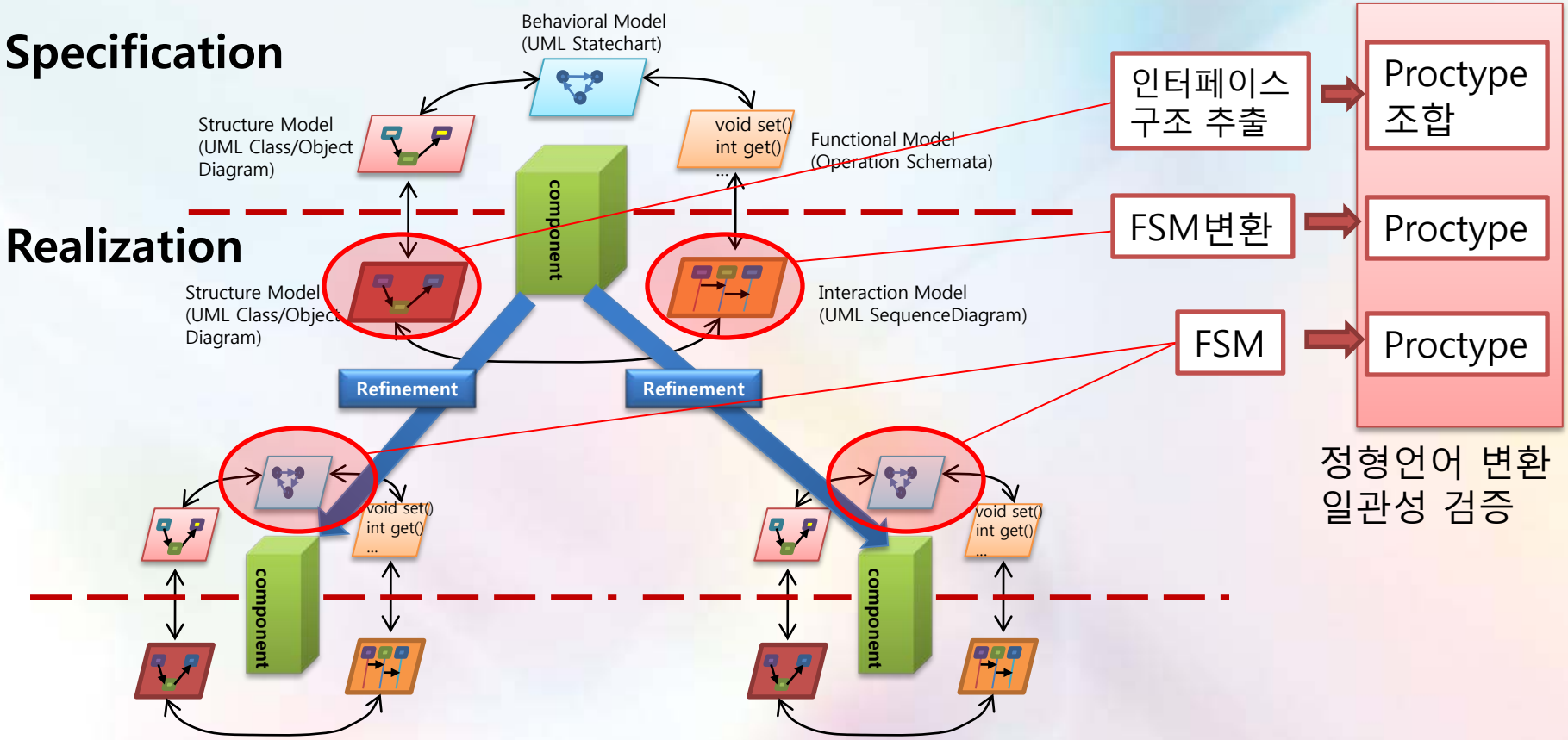
- 컴포넌트 기반 개발 방법
  - 시스템 개발과정의 복잡도의 해소와 질적향상, 높은 재사용성 보장
  - Catalysis, Select Perspective, UMLComponents

- 주요 문제점
  - 행위 복잡도
    - 컴포넌트의 수가 늘어날 수록 시스템 행위 복잡도 증가
    - 정제 과정에서 정제 전 후 컴포넌트간의 행위 일관성 검증의 부재
  - 컴포넌트의 선택 및 합성 검증
    - 컴포넌트간 행위의 호환성을 만족하는 컴포넌트의 선택
    - 독립적인 환경을 갖는 컴포넌트들의 합성 및 검증

1.2 컴포넌트 검증 관련연구

- 컴포넌트 행위 호환성 관련 연구
  - UML 컴포넌트의 인터페이스 호환성 검증
  - UML로 설계된 다이어그램간의 일관성 검증
- 특성 검증 관련 연구
  - Assume-guarantee
    - 특성을 컴포넌트 별로 나누어 검증하고 이를 조합
  - 컴포넌트를 포트로 조합하여 검증
    - 포트 행위의 오토마타를 이용하여 컴포넌트를 조합 검증
  - 특성을 패턴으로 분류하여 시제논리로 변환
    - 공통적인 특성을 패턴으로 분류하고 이를 시제 논리로 변환

2.1 추상 컴포넌트의 행위 일관성 검증



2.2.1 컴포넌트의 정형적 표현

	Type	Formal description
1	Abstract component	Comp_spec(i, o, op_set, action_set) = new u, v(I(i, o, u, v) | $Spec_0$ (u, v, op_set, action_set))
2	Interface	I(i, o, u, v) = i?x.u!x.I(i, o, u, v) + v?y.o!y.I(i,o,u,v)
3	Component behavior	$Spec_i$(u,v,op_set,action_set) = u?x.[x=op_k].$Action\_spec_k$(u, v, op_set, action_set) + u?x.[x!=op_k].$Spec_i$ (u, v, op_set, action_set)
4	Abstract Implementation	$Action\_spec_i$ (u, v, op_set, action_set) = (v!a_j)*.$Spec_j$(u, v, op_set, action_set)
5	Component realization	Comp_real(i, o, op_set, action_set) = new u, v, $\{(u_i, v_i)\}_i$ (I(i, o, u, v) | ! $_i$SubComp_i(u_i,v_i, $sub\_op\_set_i$, $sub\_action\_set_i$) | Composit_Rel(u, v, $\{ (u_i, v_i)\}_i$)
6	Component Relations	Composit_Rel(u, v, $\{(u_i, v_i)\}_i$) = $\sum v_i$?x.f(v_i)!x.Composit_Rel(u, v, $\{(u_i, v_i)\}_i$)

컴포넌트 구조와 행위 및 정제과정의 정형적 표현

2.2.2 정형명세 변환규칙

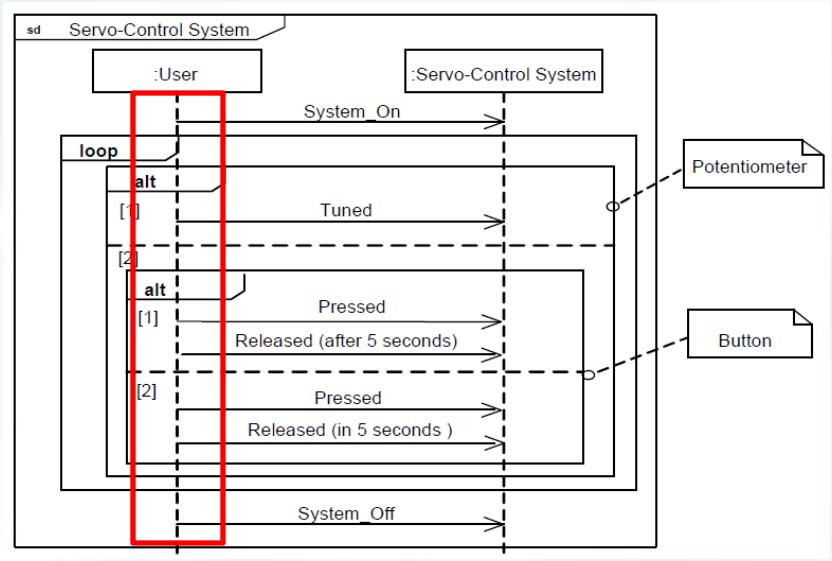
	MARMOT construct	PROMELA construct
messages	O = ∪ op_set, action_set { n | n∈ op_set or n ∈action_set}	mtype = { n_1, n_2, … , n_k}, where n_i ∈ O
channels	new u	chan u = [1] of mtype
Processes	I(i, o, u, v) Comp_spec(i, o, op_set, action_set) Comp_real(i, o, op_set, action_set)	proctype Interface(chan i, o, u, v) proctype Comp_spec(chan i, o) { … } proctype Comp_real(chan i, o) {… }

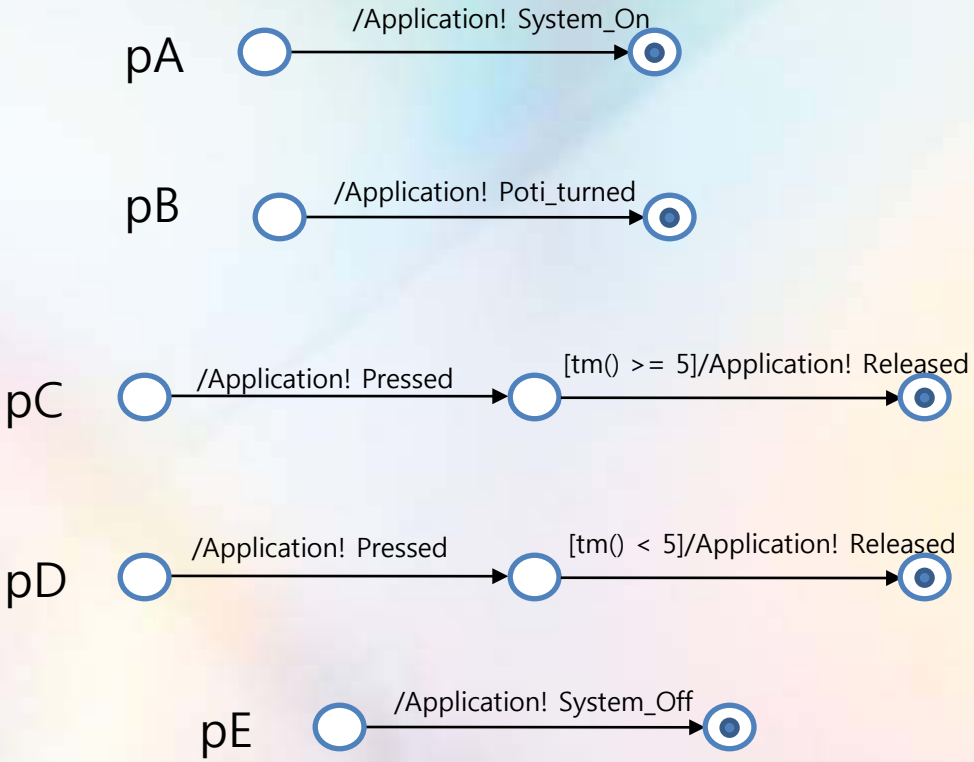
Process Activation	Comp_Spec(i, o, O, A) = new u, v I(i, o, u, v) | Spec(u, v, O, A)	proctype Comp_Spec(chan i, o) {
		chan u = [1] of mtype;
		chan v = [1] of mtype;
		run Interface(i, o, u, v);
		run Spec(u, v); }

actions	u?x u!y	mtype x; u?x; mtype y; u!y;
states	$Spec_i$(u,v,op_set,action_set)	State = i;
transitions	π.$Spec_i$(u,v,op_set,action_set)	π; State = i;
conditionals	u?x.[x=a].$Spec_i$(u, v, O, A)	if :: u?[eval(a)] → State = i; fi;

정형언어 변환규칙

2.2.3 환경 모델의 Statechart 변환 (1)



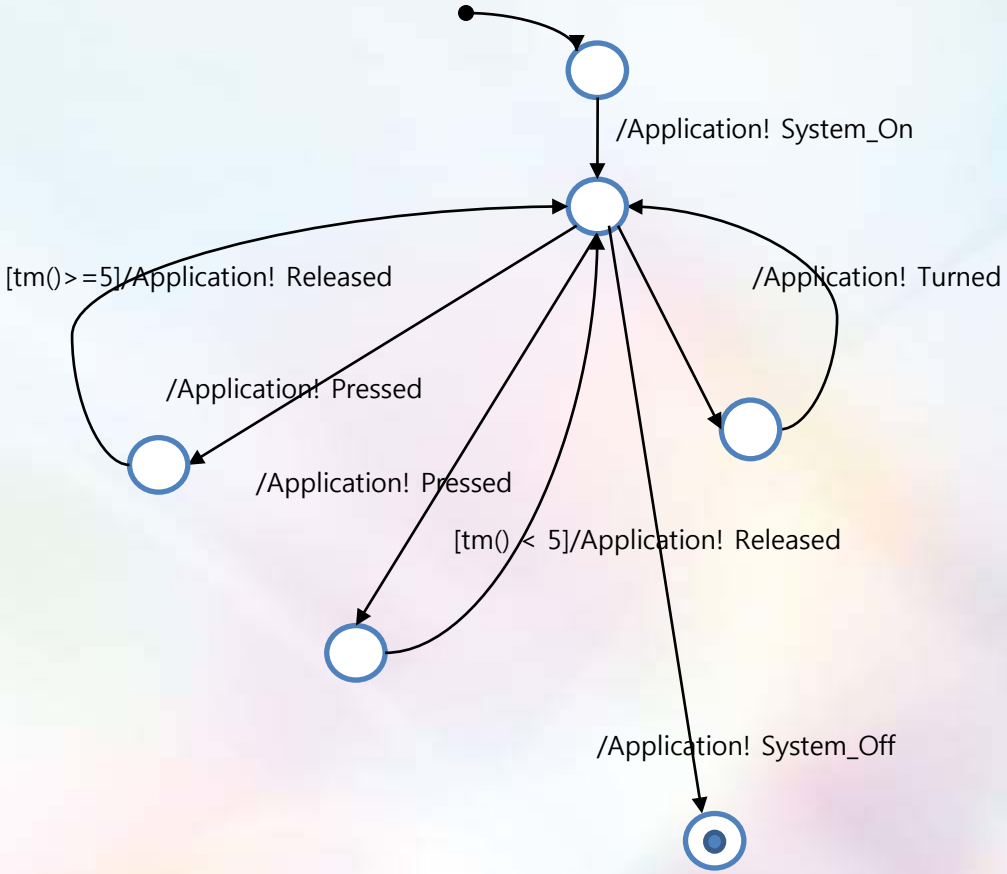


합성 규칙 :
(pA *seq* (*loop* (pB *alt* ((pC *seq* pD) *alt* (pC *seq* pD)))) *seq* pE

Reference : Tewfik Ziadi, Lo¨ıc Helou¨et, and Jean-Marc Jezequel. Revisiting statechart synthesis with an algebraic approach. In 26th International Conference on Software Engineering, 2004.

2.2.3 환경 모델의 Statechart 변환 (2)

- Statechart 변환 (합성 결과)



2.2.4 행위모델의 PROMELA 변환

- **액션언어 (Lex/Yacc 문법)**

input: /* empty string */
| input line

line: action

action : elementary_action | control_action

elementary_action : assignment_action
| send_action
| call_action
| return_action

control_action : IF simple_expression THEN line ENDIF
| IF simple_expression THEN line ELSE line ENDIF

assignment_action : attribute_name ASSIGNMENT simple_expression
| attribute_name ASSIGNMENT call_action

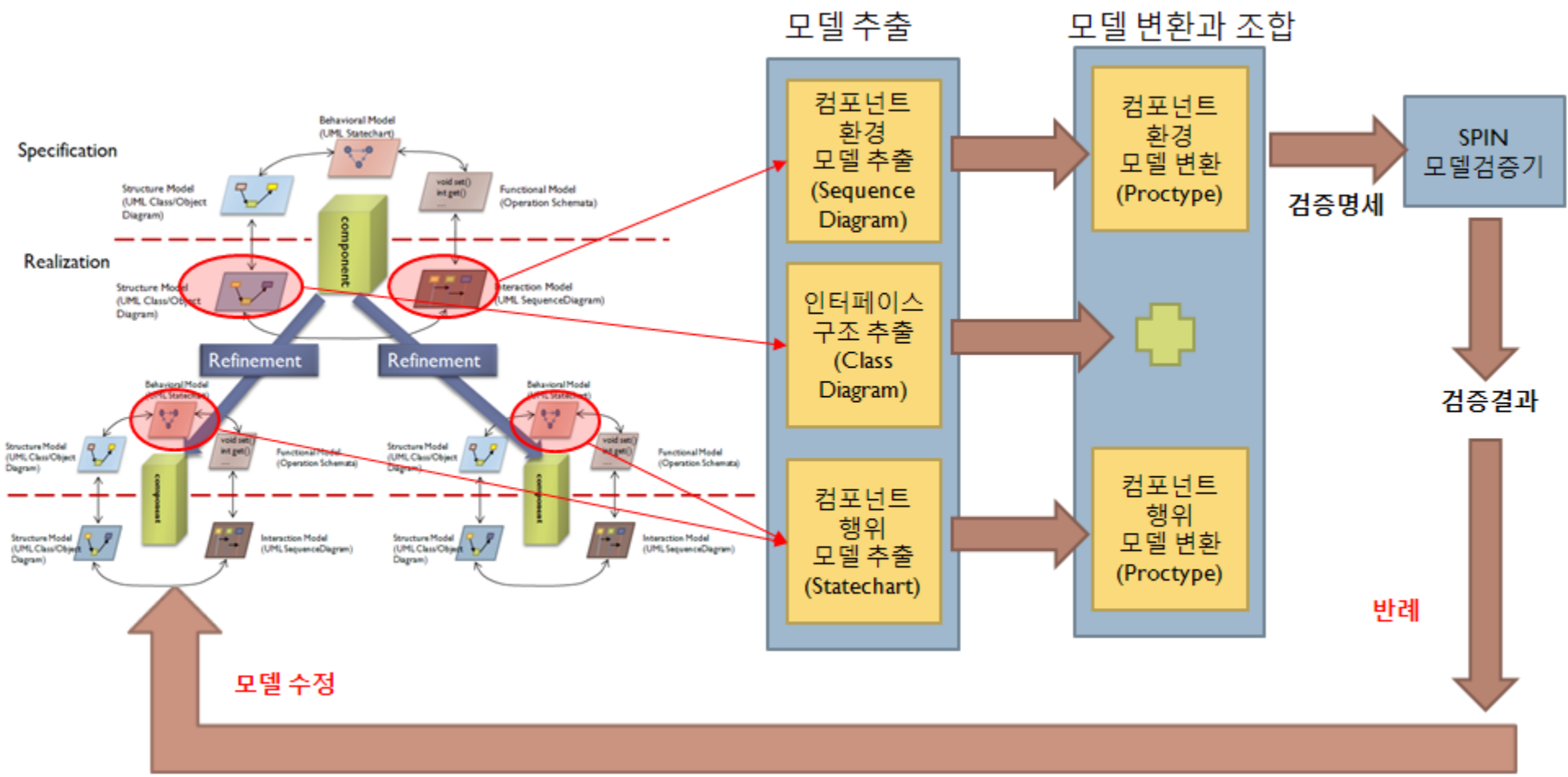
call_action : CALL call_expression ;
call_expression : classifier_name CLASSIFIER_DELI operation_name '(' paramList ')'
| operation_name '(' paramList ')'

send_action : SIGNAL classifier_name CLASSIFIER_DELI signal_name '(' paramList ')'

return_action : RETURN simple_expression | RETURN paramList

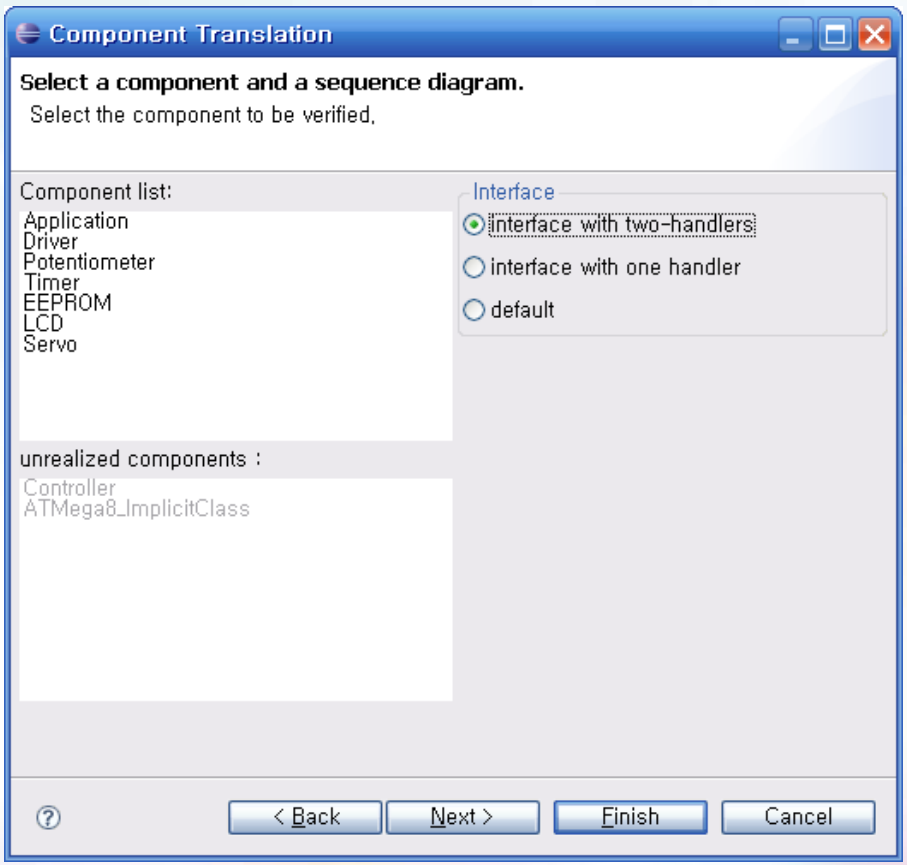
literal : FALSE | TRUE | INTEGER_LIT | NULL | attribute_name | enumeration_item ;

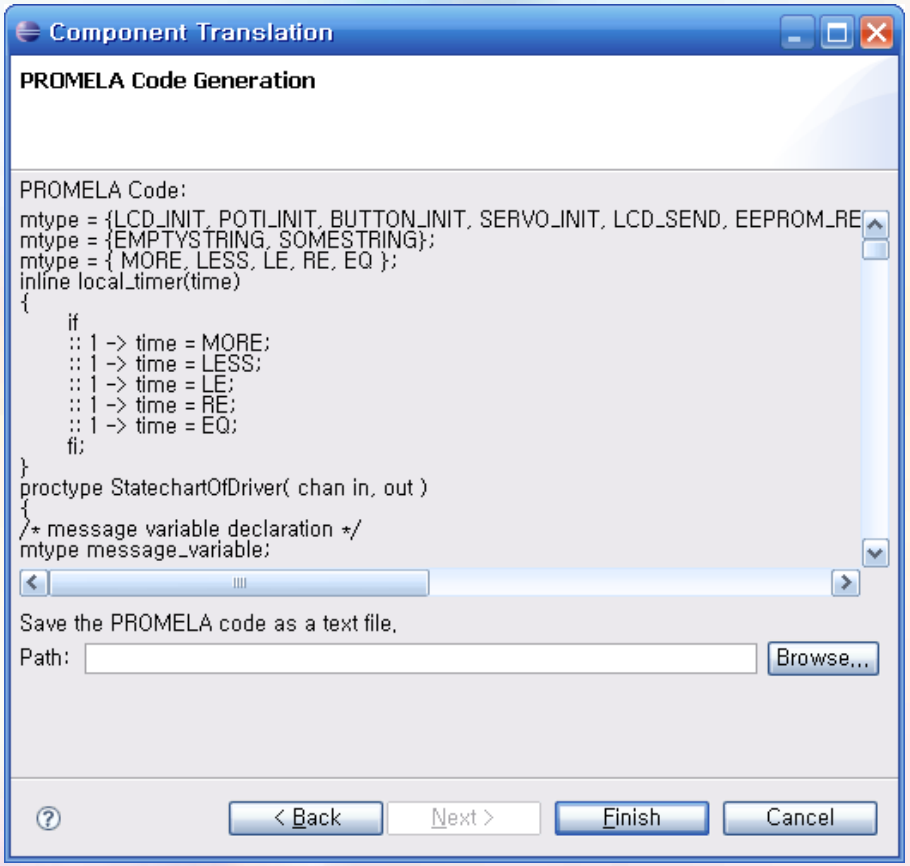
2.4 모델 변환기 구성


모델 추출
모델 변환과 조합
Specification
Realization
Behavioral Model (UML Statechart)
Structure Model (UML Class/Object Diagram)
Functional Model (Operation Schemata)
Interaction Model (UML SequenceDiagram)
component
Refinement
Refinement
컴포넌트 환경 모델 추출 (Sequence Diagram)
인터페이스 구조 추출 (Class Diagram)
컴포넌트 행위 모델 추출 (Statechart)
컴포넌트 환경 모델 변환 (Proctype)
컴포넌트 행위 모델 변환 (Proctype)
검증명세
SPIN 모델검증기
검증결과
반례
모델 수정

2.4 정형분석 모델 변환기 구현

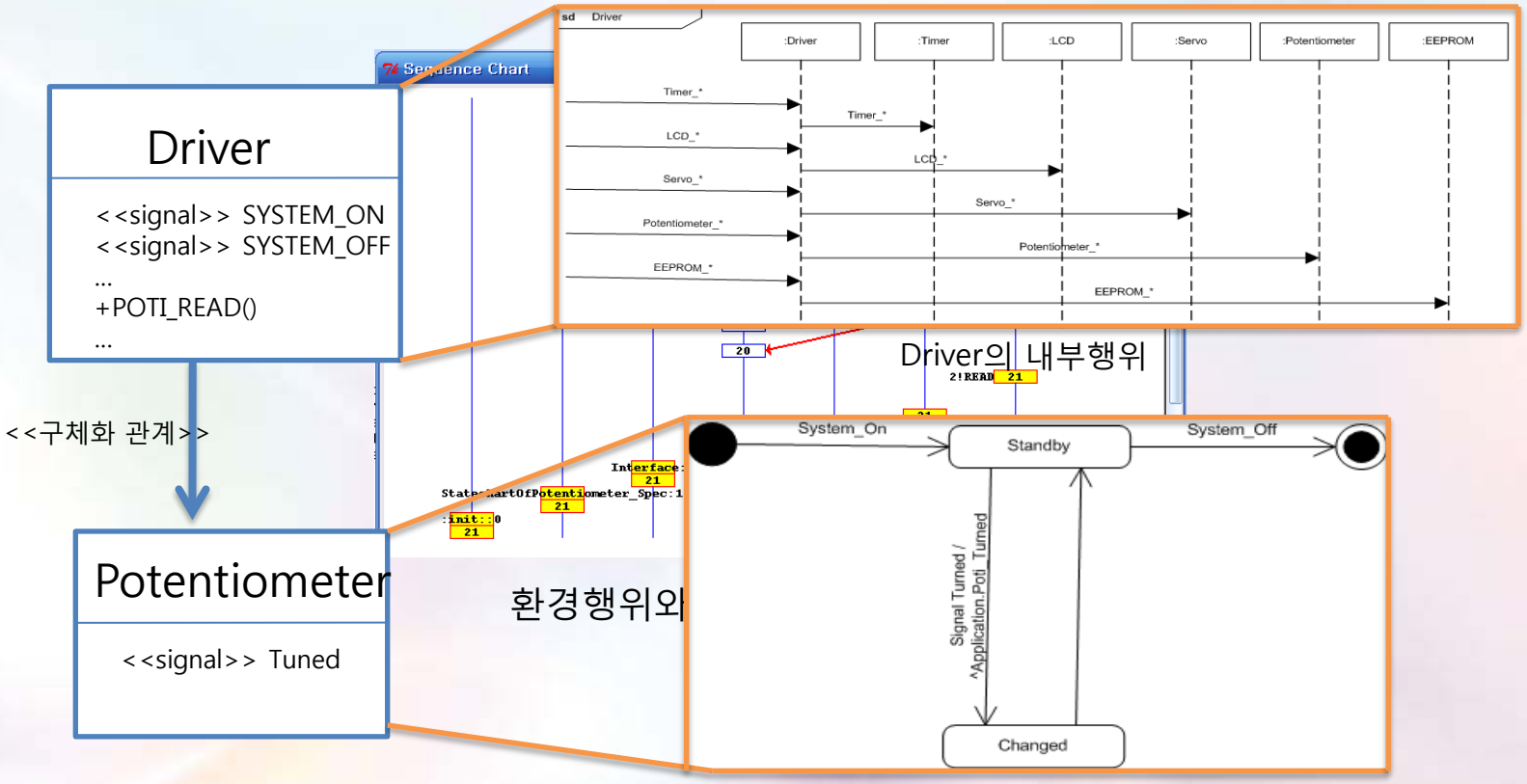
- 실행화면 *(**website:** http://m80.knu.ac.kr/~sselab/marmot.html)*





2.4 적용 결과

- 자동차 거울 조정 시스템
  - 컴포넌트 서비스 이름의 불일치
  - 컴포넌트 환경과 컴포넌트 외부행위간 행위 불일치



3.1 시스템 특성의 정의

- 시스템 특성
  - 시스템이 만족해야 할 요구사항
  - 정형검증에서 시제 논리로 표현
- 서비스 호출에 대한 시스템 특성
  - 컴포넌트간의 서비스 호출에 관한 제약조건
  - 서비스 호출이나 수행 시 다음 상황에서 일어나야 할 컴포넌트의 서비스 호출이나 수행을 표현
  - 구성요소
    - 서비스 호출 컴포넌트(Caller)
    - 서비스 수행 컴포넌트(Callee)
    - 서비스 이름

3.2 서비스 호출 특성

- 특성 문법(Lex&Yacc)
  - 기본구문은 컴포넌트에서 함수 호출(Caller)과 컴포넌트에서 함수 수행(Callee)으로 구성
  - 예 : x!a and not(y?b) -> y?b
- 변환
  - 컴포넌트의 send 메시지 변수와 take 메시지 변수로 변환
  - 시제논리 :

    □ ((*p*) -> ◇(*q*))

Property → *Compound_expression* '->' *Compound_expression* ';'

Compound_expression → *And_expression*
| *Compound_expression* 'or' *And_expression*

And_expression → *Unary_expression*
| *And_expression* 'and' *Unary_expression*

Unary_expression → *Primary_expression*
| 'not' *Primary_expression*

Primary_expression → *Term*
| '(' *Compound_expression* ')'

Term → *COMPONENT_NAME* '!' *SERVICE_NAME*
| *COMPONENT_NAME* '?' *SERVICE_NAME*

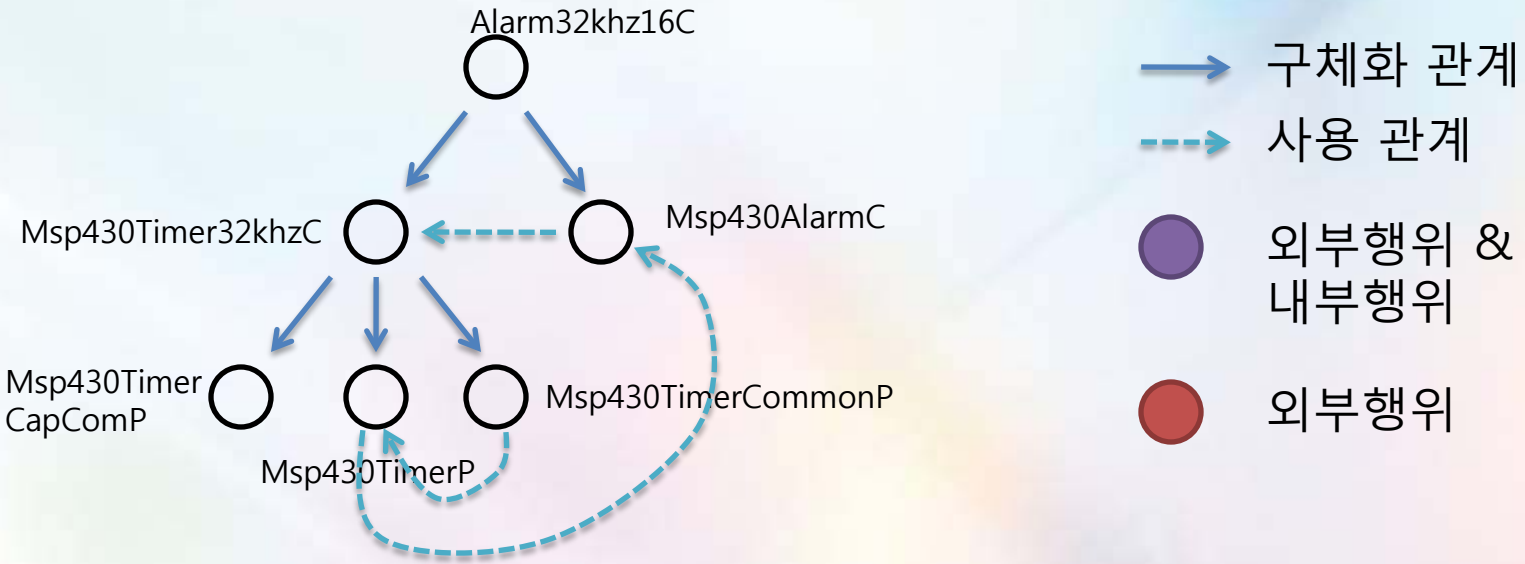
서비스 호출 특성 문법

3.3 컴포넌트 추출 알고리즘

- 컴포넌트 추출
  - 사용자가 작성한 특성 구문을 분석
  - 구체화 관계/사용관계의 Transitive Closure를 구함
  - 함수 호출에 관여하지 않는 컴포넌트 제거
- 컴포넌트 참여 마킹
  - 외부행위(**S**pecification Part)
  - 내부행위(**R**ealization Part)

3.3 컴포넌트 추출 알고리즘

- 예]

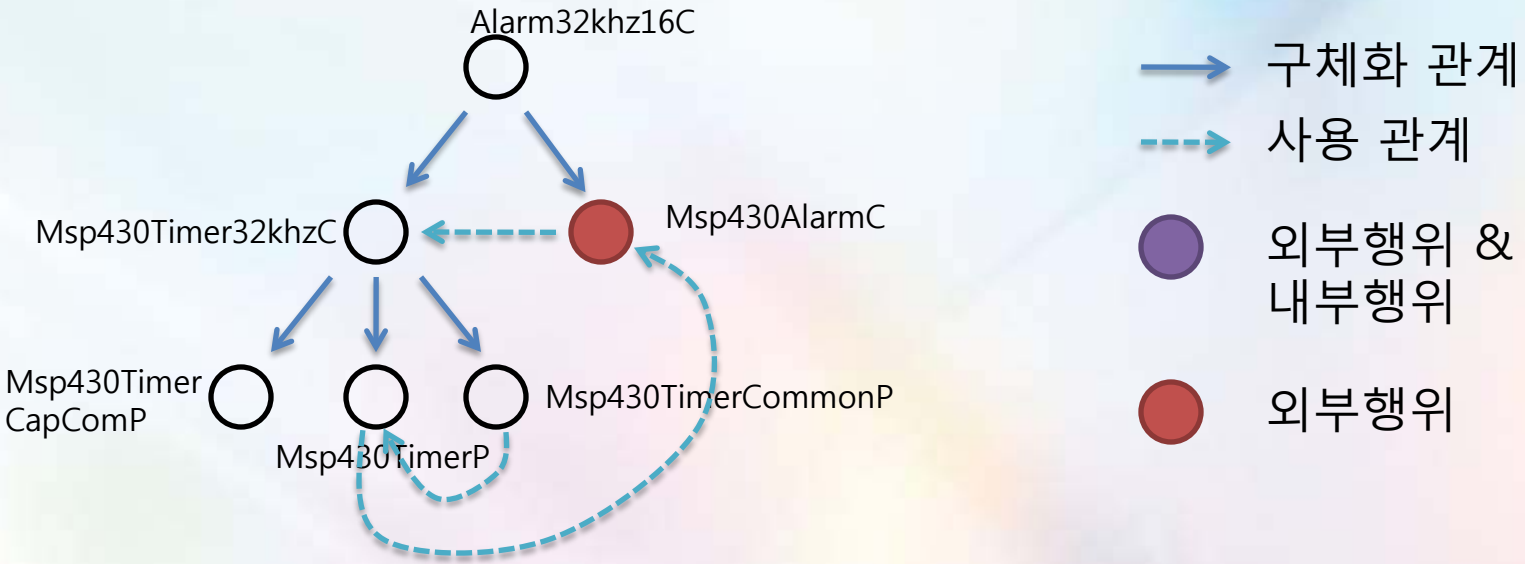


특성 : Msp430AlarmC!GET -> Msp430TimerP?GET;

Msp430AlarmC의 GET은 내부행위에 의해 구체화 되지 않는 함수
Msp430TimerP의 GET은 내부행위에 의해 구체화 되지 않는 함수

3.3 컴포넌트 추출 알고리즘

- 예]

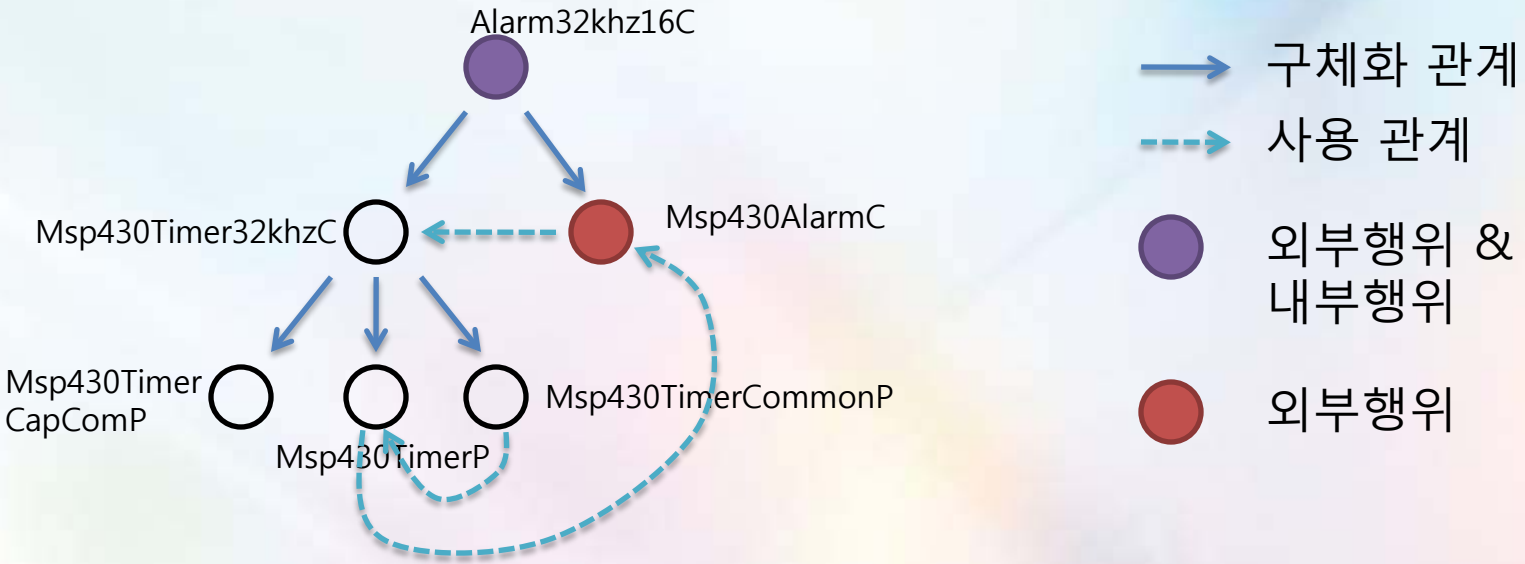


특성 : Msp430AlarmC!GET -> Msp430TimerP?GET;

Msp430AlarmC을 **S** 마킹

3.3 컴포넌트 추출 알고리즘

- 예]

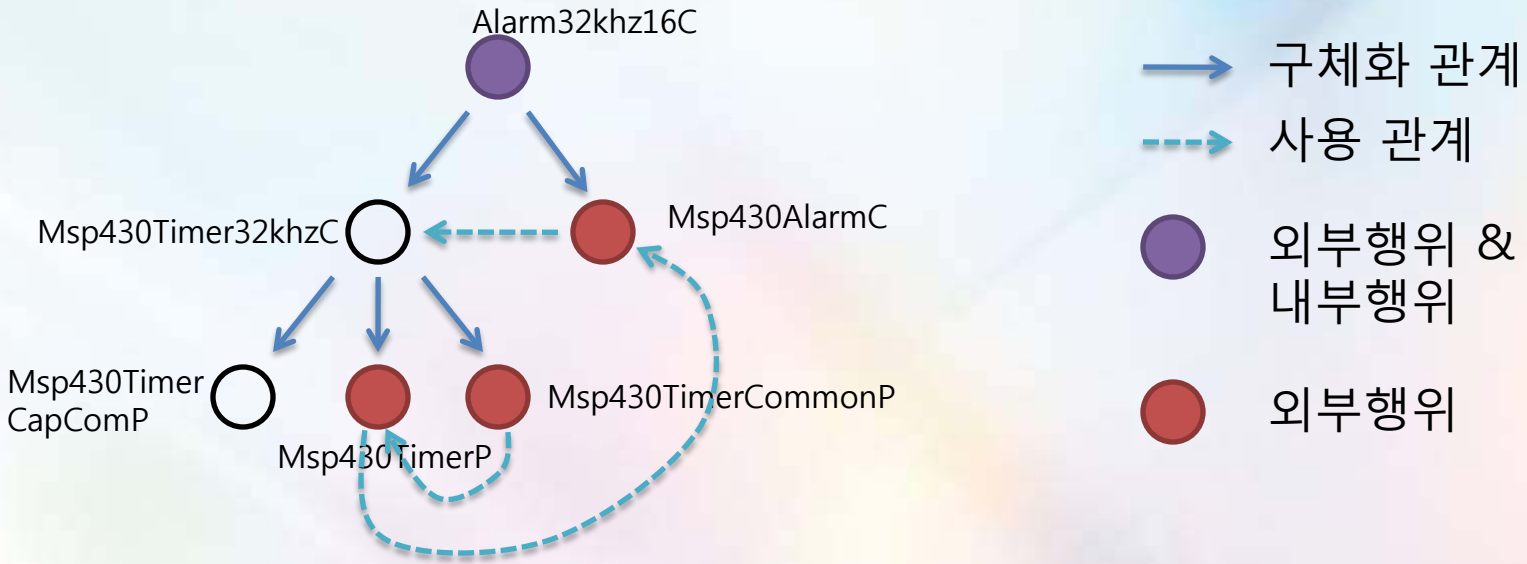


특성 : Msp430AlarmC!GET -> Msp430TimerP?GET;

Msp430AlarmC로부터 최상위 컴포넌트까지 **S**&**R** 마킹

3.3 컴포넌트 추출 알고리즘

- 예]

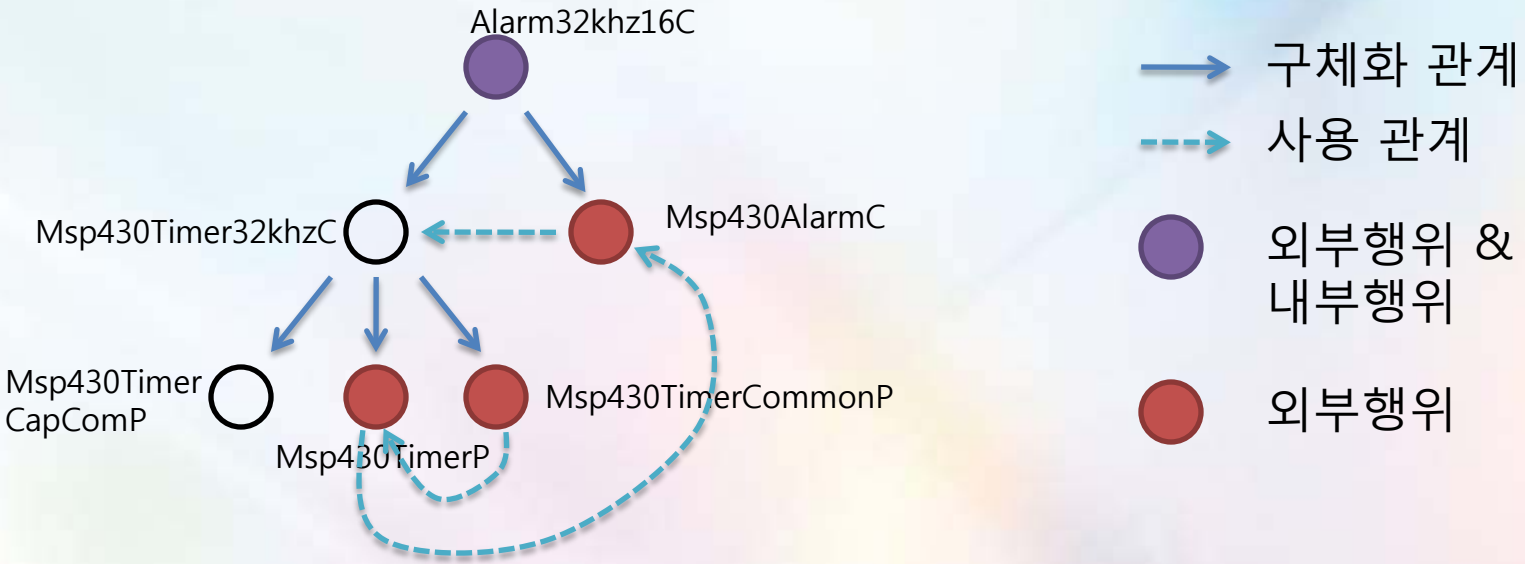


특성 : Msp430AlarmC!GET -> Msp430TimerP?GET;

Msp430AlarmC로부터의 사용관계의 순/역 Transitive Closure에 대한 **S**마킹

3.3 컴포넌트 추출 알고리즘

- 예]

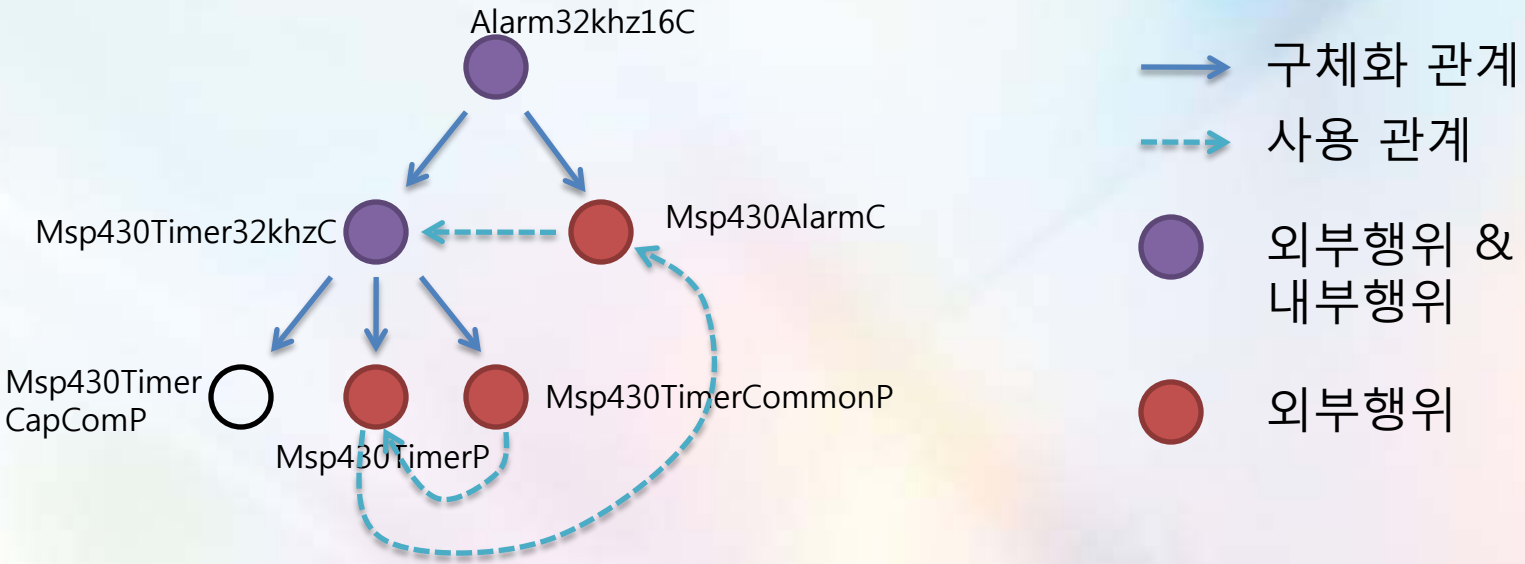


특성 : Msp430AlarmC!GET -> Msp430TimerP?GET;

Msp430TimerP를 **S** 마킹

3.3 컴포넌트 추출 알고리즘

- 예]

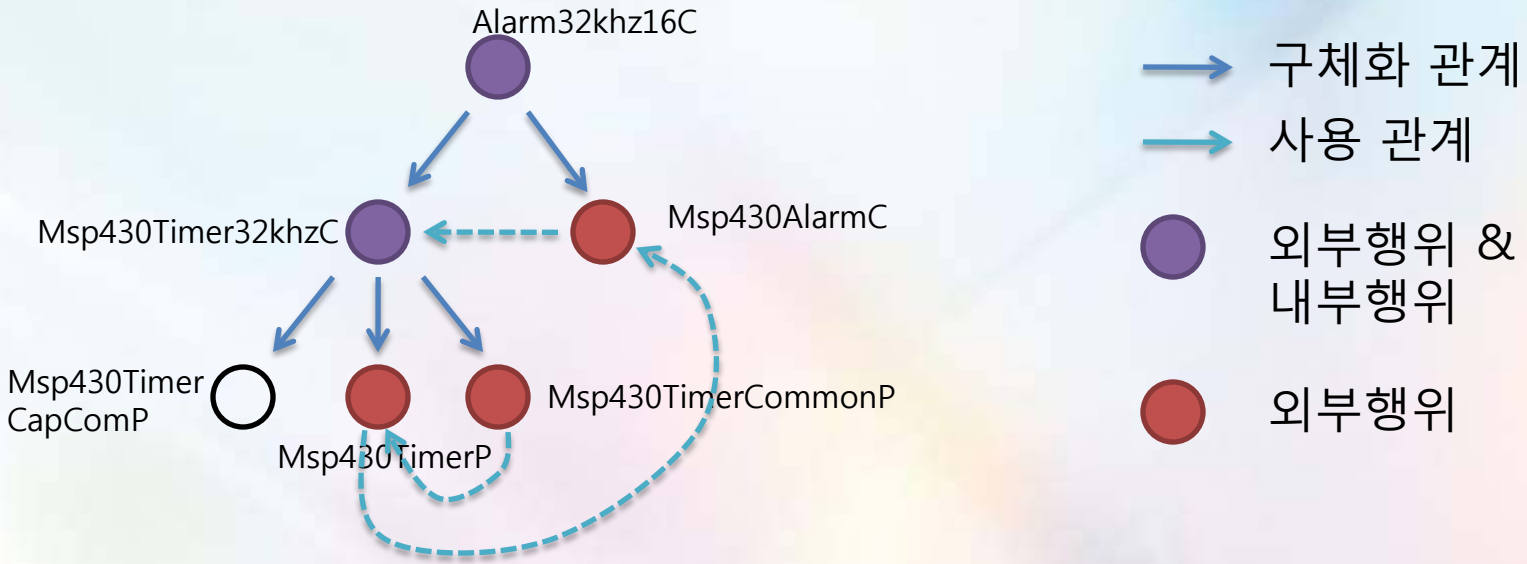


특성 : Msp430AlarmC!GET -> Msp430TimerP?GET;

Msp430TimerP로부터 최상위 컴포넌트까지 **S**&**R** 마킹

3.3 컴포넌트 추출 알고리즘

- 예]

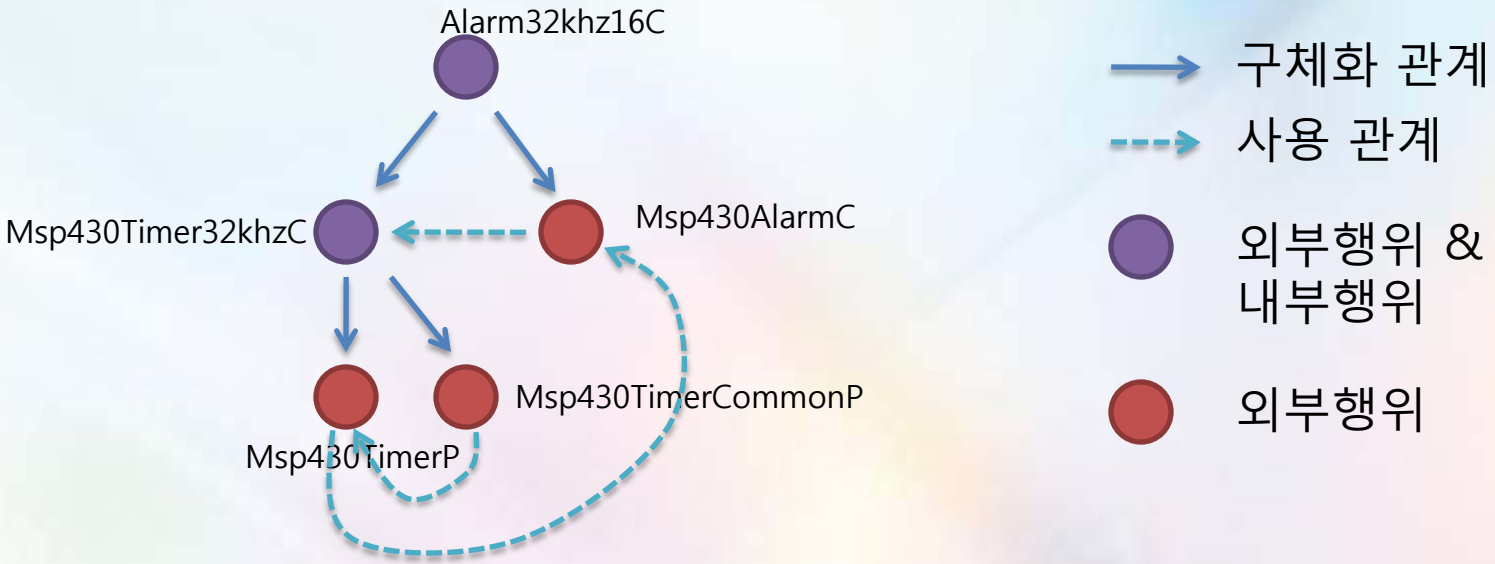


특성 : Msp430AlarmC!GET -> Msp430TimerP?GET;

Msp430TimerP로부터 사용관계의 순/역 Transitive Closure : **S**마킹

3.3 컴포넌트 추출 알고리즘

- 예]



특성 : Msp430AlarmC!GET -> Msp430TimerP?GET;

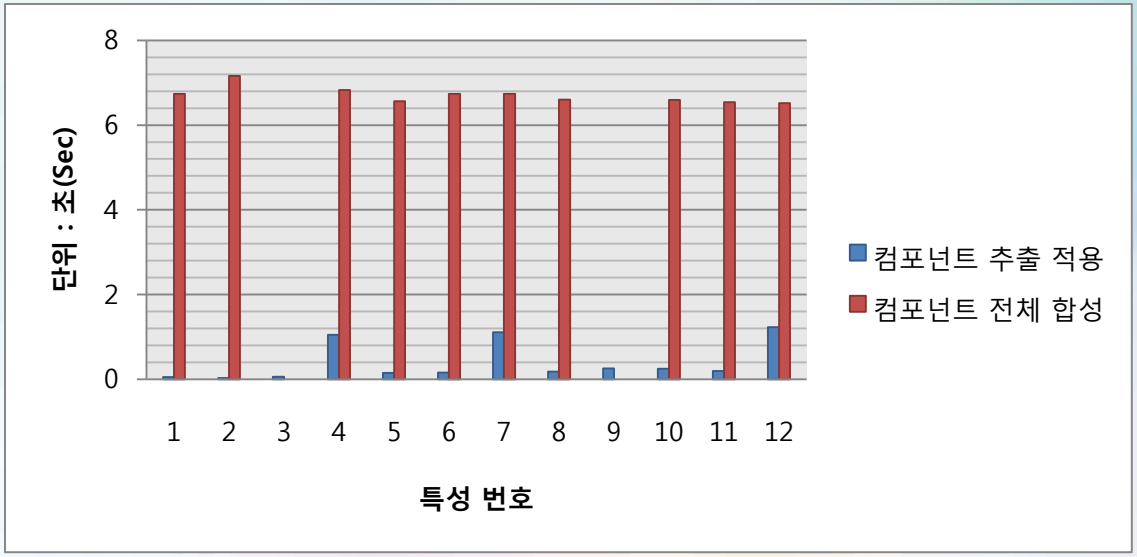
S 혹은 **R**로 마킹된 컴포넌트 집합만 추출

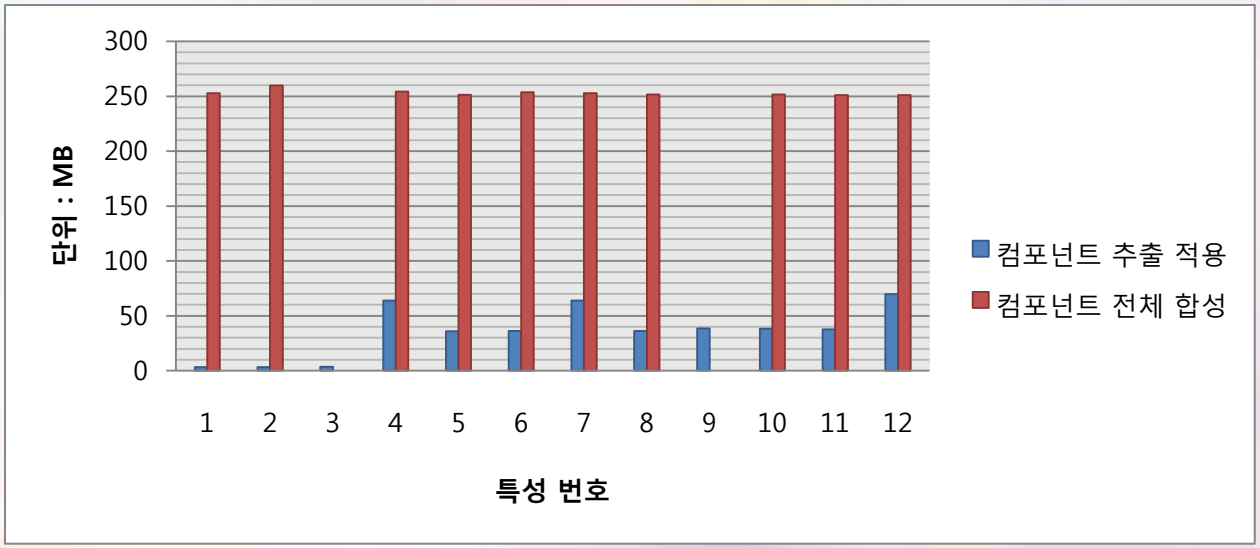
표 5.2 자동차 거울 조정시스템에 요구되는 특성

3.3 적용결과

자동차 거울 조정 시스템		특성 번호	1단계		특성 번호	여러 단계
특성	최상위 컴포넌트와 그 하위 컴포넌트	1	Application?Button_Pressed and not (Driver? Timer_Start) -> Driver?Timer_Start	최상위 컴포넌트와 마지막 컴포넌트	7	Application? Button_Pressed and not(Timer? Start or Timer? Stop) -> Timer? Start or Timer? Stop
		2	Application?Button_Released and Driver? LCD_Send -> Driver? Servo_set		8	Application? Button_Released and not (EEPROM? Store or EEPROM? Retrieve) -> EPROM? Store or EEPROM? Retrieve
		3	Application? Poti_Tuned -> Driver? Poti_Read		9	Application? Poti_tuned and not (Potentiometer? Poti_Read) -> Potentiometer? Read
	마지막 컴포넌트와 그 상위 컴포넌트	4	Driver? Timer_Start and not (Timer? Start) -> Timer? Start	마지막컴포넌트와 마지막컴포넌트	10	EEPROM? Retrieve and not (Servo? Set) -> Servo? Set
		5	Driver? Poti_Init and not (Potentiometer? Init) -> Potentiometer? Init		11	Potentiometer? Read and not (Servo? Set) -> Servo? Set
		6	Driver? EEPROM_Store and not (EEPROM? EEPROM_Store) ->EEPROM? Store		12	Timer? Stop and not (LCD? Send) -> LCD? Send

3.3 적용결과


8
6
4
2
0
단위 : 초(Sec)
1 2 3 4 5 6 7 8 9 10 11 12
특성 번호
컴포넌트 추출 적용
컴포넌트 전체 합성


300
250
200
150
100
50
0
단위 : MB
1 2 3 4 5 6 7 8 9 10 11 12
특성 번호
컴포넌트 추출 적용
컴포넌트 전체 합성

4. 향후 과제

- 서비스 호출 이외에 다양한 특성 패턴 지원
- 컴포넌트 추출 시, 효과적인 경우와 비효과적인 경우에 대한 분류 및 효율 향상
- Assume-Guarantee 방식의 적용과 비교 분석


THANK YOU!