Ada와 Statemate를 이용한 소프트웨어 설계모듈의 재사용

김창진° 최진영
고려대학교 컴퓨터학과 컴퓨터이론 및 정형기법 연구실
{cjkim°, choi}@formal.korea.ac.kr

Reuse Of Software Design Module Using Ada and Statemate

Changjin Kim° Jin-Young Choi
Computer Theory & Formal Method Lab, Department of Computer Science, Korea University

요 약

Ada 언어의 일반화(Generic) 메커니즘은 복잡한 대형 시스템의 설계에 있어 소프트웨어의 재사용성을 높이기 위한 효과적인 방편으로 사용되어 왔다. 일반화 모듈은 기능 또는 알고리즘은 정의되었으나 파라미터의 구체적인 속성이 아직 정해지지 않은 한 개의 소프트웨어 모듈을 다양한 파라미터를 적용하여 여러 곳에서 반복적으로 사용할 수 있도록 한 개념이다. 정형이론 및 구조적 설계기법에 기반한 설계도구인 Statemate 또한 설계 모듈의 반복적인 재사용을 위해 일반화 차트(Generic Chart)를 제공하고 있다. ATCS(Air Traffic Control System) 설계 모듈에 적용한 결과 두 가지 모두 소프트웨어의 재사용과 설계의 효율성 향상을 위한 도구로 활용할 수 있으며 Statemate의 경우 일반화 차트를 다양한 자료형으로 실례화(Instantiation)할 수 있다면 보다 효과적인 설계모듈의 재사용이 가능할 것이다.

1. 서 론

1973년 당시 미 국방성이 인식한 소프트웨어 위기(software crisis)의 핵심은 미 국방성 전체 컴퓨팅 비용의 46%에 해당하는 소프트웨어 비용과 그 중 56%에 달하는 임베디드 소프트웨어의 비용 문제였다. 그에 대응하여 미 국방성(DoD)은 소프트웨어의 재사용성 및 프로그래밍의 생산성 향상을 목적으로 사용된 이전의 모든 프로그래밍 언어와 패러다임을 대체하는 표준 언어로서 Ada를 개발하게 된다[1].

본 논문에서는 Ada가 전통적으로 존재했던 엄격한 자료형과 재사용성 간의 비양립성 (incompatibility) 문제를 해결한 언어라는 사실에 주목하였다. 그리고 그 구체적인 도구로서 Ada가 제공하는 일반화 단위(generic unit)의 개념과 구현방법 상의 특성을 검토한다.

기존의 Ada 시스템 설계 방식을 Statemate의 설계모듈로 변환하고자 한 것은 전통적인 설계기법 적용 시 시스템의 이해/분석 과정에서 발행할 수 있는 해석의 차를 최소화하고 Ada 언어의 장점을 시스템에 최대한 반영하기 위해서이다. 또한 프로그램의 모듈화가 비교적 용이한 Ada의 속성에 비추어 Statemate가 효과적인 설계도구가 될 수 있었기 때문이다. 본 연구에 앞서 Kjell Nielsen[4]이 실제 개발 경험을 토대로 구조적 설계, 객체지향(Object-Oriented), DARTS (Design Approach for Real- Time Systems) 등의 설계기법들을 적절히 응용한 Ada Real- Time Design Methodology(ARTDM)가 큰 변형 없이 Statemate의 설계 절차로 활용될 수 있음을 확인한 바 있다.

설계에 사용된 I-Logix사의 Statemate Magnum(이하 Statemate)은 설계 대상의 기능적(functional) 측면을 표현하는 activity chart와 그 행위적(behavior) 측면을 표현하는 statechart를 통해 보다 직관적인 시스템의 이해를 도와준다. 또한 전체 시스템 설계의 완성 여부에 관계없이 설계 모듈 별로 설계의 일관성(consistency), 정확성(correctness) 등을 검증할 수 있는 다양한 도구를 제공하므로 설계가 진행 중인 경우에도 조기에 오류를 발견, 수정할 수 있다.

본 논문을 통해 전통적인 Ada의 generic program을 Statemate의 재사용 메커니즘 즉, 일반화 차트(generic chart)로 변환할 수 있는지 검토하고 Statemate를 통해 자동 생성된 generic 모듈의 소스코드를 분석한다.

2. Ada 일반화(generic)의 개념 및 구현

Generic unit은 Ada 프로그램의 한 단위로서 generic *subprogram* 또는 *generic package*의 형태를 가진다. Generic unit은 매개변수화 될 수 있는 일종의 틀(template)이며 그 자체는 실행될 수 없는 프로그램이지만 그 틀로부터 generic 하지 않으면서 실행 가능한 프로그램을 생성할 수 있다.

일련의 프로그램들이 알고리즘은 동일하고 각각 사용하는 자료구조 또는 자료의 객체(object)만 다르다고 가정하자. 그러한 프로그램을 하나의 틀 즉, generic unit으로 만들 수 있는데 Ada에서 generic unit은 그 자체가 마치 시스템의 자원과 같이 관리된다. 필요한 자료구조를 generic unit에 적용하여 실행 가능한 프로그램으로 반복해서 만들 수 있다면 동일한 틀에서 다양한 자료를 사용하는 다수의 프로그램을 생성한 효과를 얻을 수 있을 것이다. 이것이 Ada의 generic 개념인데 프로그램의 시스템 자원화를 통하여 양질의 프로그램을 재사용함으로써 프로그램의 생산을 보다 쉽게 할 수 있도록 한 것이다[2].

Generic unit의 또 하나의 장점은 아직 결정되지 않은

타입의 객체를 포함하는 프로시저 또는 함수를 미리 작성할 수 있다는 데 있다. 즉 그 프로시저 또는 함수의 개발자는 실제로 사용될 구체적인 객체의 속성 또는 자료형에 대해 미리 고민할 필요가 없다는 것이다.

그러나 무엇보다 generic unit이 필요한 이유는 검증된 모듈의 재사용을 통해 개발 노력 감소 효과와 프로그램의 신뢰성을 동시에 획득할 수 있기 때문이다.

```
generic
    type Track_Count is range <>;
    type Coordinates is private;
    -- for the use of private type Coordinates
    with function "<=" (Left, Right : in Coordinates)
        return Boolean is <>;

package Track_File_Manager is
    subtype Track_Number is Track_Count
                            range 1 .. Track_Count'Last;
    procedure Add_Track (x, y : in Coordinates);
    Track_Store_Full : exception; -- raised by Add_Track
    procedure Initialize_Track_Reading;
    procedure Next_Track (x, y : out Coordinates;
                          Track_Id : out Track_Number);
    No_More_Tracks : exception; -- raised by Next_Track
    procedure Update (x, y : in Coordinates;
                      Track_Id : in Track_Number);
    procedure Correlate (x, y : in Coordinates);
    procedure Extrapolate;
end Track_File_Manager;
```

그림 2-1. Generic package 명세

위의 그림 2-1은 본 연구에 사용된 실시간 항공관제 체계인 ATCS 모델[4]의 track file 관리를 위해 작성된 generic package의 소스코드이다.

코드를 살펴보면 패키지의 명세 이전에 'Track_Count'와 'Coordinates'를 generic type parameter로 선언함으로써 형식매개변수(formal parameter)를 정의하였다. 또한 프로시저 Add_Track, Next_Track, Update 및 Correlate 등은 형식 매개변수를 입력 또는 출력 값으로 사용하면서 항적(track) 정보를 관리하는 기능을 수행하도록 하였다.

Generic 프로그램의 기능을 활성화하기 위해서는 형식 매개변수 대신 사용될 실제 매개변수(actual parameter)가 generic 프로그램을 사용하는 해당 프로그램 내에서 결정되어야 한다. 그 과정을 실례화(instantiation)라고 하며 예제는 다음의 그림 2-1과 같다.

```
with Track_File_Manager;

separate (Track_File_Monitor)
    task body Monitor is                    Instantiation
        package Track_File_Mgr is
                    new Track_File_Manager
                      (Coordinates => Coordinates,
                       Track_Count => Track_Count);
    begin
    -- ...
        accept Initialize_Track do
            Track_File_Mgr.Initialize_Track_Reading;
    -- ...
    end Monitor;
```

그림 2-1. Generic 패키지의 실례화 (insanitation)

위의 예제는 generic type parameter를 선언함으로써 서로 다른 자료형의 실제 파라미터를 사용하여 다양한 프로그램으로 실례화할 수 있는 방법을 보여주고 있다.

Ada는 다른 형태의 generic 메커니즘도 제공하고 있는데 자료형 매개변수(type parameter) 외에도 값(value), 객체(object) 또는 subprogram을 generic unit의 매개변수로 전달할 수 있다.

그런데 다음 장에서 설명할 Statemate의 generic chart는 형식매개변수와 동일한 자료형의 실제매개변수만을 허용한다. 따라서 어떤 설계 모듈을 generic type parameter를 사용하여 Ada generic unit으로 만들었다고 해도 현재의 Statemate 기능으로는 그와 정확히 일치하는 generic chart를 만들 수 없다.

3. Statemate 일반화 차트(Generic chart) 설계

Generic chart는 시스템 설계 시 동일한 설계모듈의 반복적인 사용을 위해 Statemate가 제공하는 재사용 메커니즘이다.

원래의 Kjell Nielsen의 ATCS 모델에서는 다수의 레이더를 위한 인터페이스가 구체적으로 기술되지 않았다. 그러나 실제 운영 중인 ATCS들은 여러 개의 서로 다른 레이더들과 다양한 인터페이스를 구축하고 있는 경우가 많다. 본 논문에서는 ATCS가 각기 다른 3개의 레이더로부터 자료를 수신한다고 가정하고 레이더와의 인터페이스를 위해 activity chart RADAR_HANDLER를 그림 3-1과 같은 generic chart로 작성하였다.

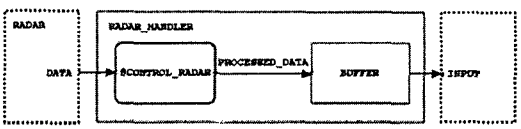

그림 3-1. Generic activity-chart for Radar_Handler

그림 3-2.는 generic activity RADAR_HANDLER의 instance인 RDR_PS_1, RDR_PS_2, RDR_PS_3와 그들의 외부 환경인 RADAR_1, RADAR_2, RADAR_3을 나타낸 top-level activity chart이다.

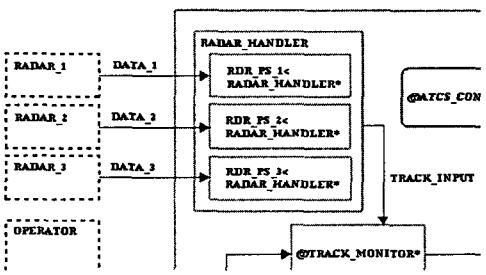

그림 3-2. Instances of Radar_Handler

'RDR_PS_1<RADAR_HANDLER'의 표기법은 RDR_PS_1이 generic activity chart RADAR_HANDLER의 인스턴스 activity라는 것을 나타내며 위 그림에 나타난 3개의 인스턴스들은 각각 기능적으로 RADAR_HANDLER와 동일하다.

여기서 각 인스턴스 activity들은 generic chart의 형식매개변수에 해당하는 DATA와 INPUT에 대해 각각

DATA_1, DATA_2, DATA_3와 TRACK_INPUT을 실제매개변수로 매칭 시키는 과정을 거쳐야만 그 기능이 활성화되는데 이를 "actual parameter binding"이라고 한다.

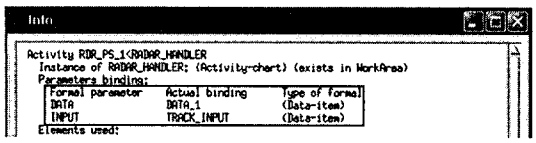

그림 3-3. Actual parameter binding

그림 3-3은 형식매개변수 DATA 및 INPUT에 해당하는 실제매개변수의 actual binding 결과를 보여주고 있다. 그런데 Statemate에서 actual binding은 반드시 형식매개변수와 같은 타입의 데이터로만 이루어져야 한다. 또한 이를 위해서는 generic chart의 생성 단계에서 반드시 형식매개변수의 타입이 구체적으로 결정되어야 한다. 즉 Ada의 generic 컴포넌트 설계에서와 같은 formal type parameter의 사용이 근본적으로 불가능하다는 것인데 이 경우 그림 3-1의 generic chart를 사용하는 레이더는 모두 같은 타입의 데이터를 인터페이스 모듈로 보내야만 한다. 이는 어떤 시점에서 설계가 종료된 이후에 다른 설계모듈을 추가하는 경우 현실적으로 상당한 제약사항이 될 수 있다. 결국 추가적인 모듈이 알고리즘은 같더라도 타입이 다른 자료를 사용해야 한다면 완전히 새로운 모듈을 다시 설계해야 한다.

Statemate에서 허용하는 actual binding의 가변성은 형식매개변수와 동일한 자료형을 필드로 가지는 구조체를 실제매개변수로 사용하는 경우 각 필드의 범위를 형식매개변수가 정하는 범위 내에서 변경할 수 있다는 점뿐이다.

그림 3-4는 Statemate의 자동코드생성기능을 통해 추출된 ATCS 모델의 Ada 소스코드 중 일부분으로 generic chart RADAR_HANDLER에 해당하는 generic package이다.

```
generic
  instance_name  : in string ;
  DATAd1         : in out  PLOT_DATA_TYPE;
  INPUT          : in out  PROCESSED_RADAR_DATA_TYPE;

package g_RADAR_HANDLER is
-- Inputs
-- Outputs
  BUFFERacy       : ACTIVITY := dummy_act;
  RADAR_HANDLER   : ACTIVITY := dummy_act;
  RDPROCESSED_RADAR_DATA : PROCESSED_RADAR_DATA_TYPE_event ;
  RDRAW_DATA      : event ;
  RAW_DATA        : INTEGER32;
  PROCESSED_RADAR_DATA : PROCESSED_RADAR_DATA_TYPE;
  function G_ROOT return address;
  SCOPE_ID : INTEGER32 ;
  instance_info : scope_info ;
  debug_table : array (0..4) of address := (OTHERS => ZNIL);
  procedure dbg_g_radar_handler_init;
  procedure dbg_g_radar_handler_eval_compounds;
  procedure g_radar_handler_init;
  procedure g_radar_handler_exec_all;
end g_RADAR_HANDLER ;
```

그림 3-4. Generic chart RADAR_HANDLER의 Statemate auto-generated Ada code

그림 3-5.에서 ***표시된 부분은 자동 생성된 package rdr_hdlr의 소스코드 중 g_RADAR_HANDLER에 대한 실례화 부분을 보여주고 있다.

그림 3-5 및 3-6.에서 나타난 바와 같이 Statemate에서 generic chart로 작성된 모듈은 자동코드생성기능을 통해 Ada generic unit의 개념 및 syntax에 정확히 일치하는 코드로 추출될 수 있음을 확인하였다. 또한 Statemate에서 generic instance로 실례화한 모듈 역시 완벽한 Ada generic unit의 실례화 코드로 추출할 수 있었다.

```
UPDATE_TRACK      : TRACK_DATA_TYPE;
UPDATE_TRACK_DATA : TRACK_DATA_TYPE;
***
package Inst_RDR_PS_1 is new g_RADAR_HANDLER
                  ("RDR_PS_1A",DATA_1,TRACK_INPUT);
package Inst_RDR_PS_2 is new g_RADAR_HANDLER
                  ("RDR_PS_2A",DATA_2,TRACK_INPUT);
package Inst_RDR_PS_3 is new g_RADAR_HANDLER
                  ("RDR_PS_3A",DATA_3,TRACK_INPUT);

procedure dbg_rdr_hdlr_init;
```

그림 3-5. Statemate로 자동 생성된 generic chart RADAR_HANDLER의 실례화 코드

4. 결론 및 과제

본 논문에서는 동일한 자료형을 가지는 3개의 Statemate generic instance를 실례화하였다. 그러나 ATCS의 사례를 고려해 볼 때, 실제로 다른 기종의 레이더별로 각기 다른 타입의 입력 자료를 처리하기 위해서는 실제매개변수의 자료형을 실례화 시점에서 다양하게 사용할 수 있는 방법이 필요하다. 그러나 Statemate는 형식매개변수와 다른 타입의 실제매개변수를 사용하여 실례화했을 때 'inconsistent parameter' 사용에 따른 경고를 발생시킨다. Ada의 generic subprogram에서 매우 효율적인 유연성의 한 측면이 서로 다른 자료형의 실례화라면 Statemate의 generic chart의 부족한 특성은 동일한 자료형의 사용만 가능하다는 제약조건일 것이다. 그 외 형식매개변수의 선언과 내부 기능, 특히 activity의 동작과 관련된 속성의 공유, 그리고 실례화 방법 등에 있어서는 Ada의 generic unit과 Statemate의 generic chart가 매우 유사한 특성을 보여주고 있다. 서로 다른 자료형에 대해서도 generic chart의 사용이 가능하도록 하는 것은 Statemate를 사용한 설계에 있어 반복적인 차트의 재사용을 위해 포함되어야 할 기능으로서 지속 연구되어야 할 것이다.

참고문헌

[1] Colin Atkinson, "Object-Oriented Reuse, Concur-rency and Distribution", Addison-Wesley, 1991
[2] 김화수 외, "Ada95를 이용한 실시간시스템 입문", 집문당, 1999
[3] David Harel, "Modeling Reactive Systems with State charts: The Statemate Approach", I-Logix, 1999
[4] Kjell Nielsen, "Designing Large Real-Time Systems with Ada", Hughes Aircraft Company, 1987
[5] J. G. P Barnes, "Programming in Ada, 4th Edition", Addison-Wesley, 1993
[6] Michael A. Smith, "Object-oriented Software in Ada95 Second Edition", Mcgraw-Hill, 2001