■ MetaH. Spec. 분석

MetaH는 미국의 Honeywell사에서 개발한 신뢰성 있는 실시간 다중 프로세서 항공공학 시스템을 개발하기 위한 언어이면서 툴셋이다. MetaH 는 소프트웨어와 하드웨어 구조를 명세하기 위한 시스템 아키텍처를 제공하고 있는데, 소프트웨어 아키텍처의 개념은 컴포넌트 기반 소프트웨어 개발 방법과 분리되어 연구되었다. 소프트웨어 아키텍처의 정의는 현재까지 표준화되어 있지 않으며, 일반적으로 소프트웨어 아키텍처는 시스템의 구조적 측면에 중점을 두어 왔다. 또한, 지금까지의 연구는 아키텍처를 기술하기 위한 정형명세에 치중하였다. 이러한 관점들을 기술하기 위해 많은 아키텍처 기술언어들이 제안되어 왔는데, 그 중 하나가 MetaH이다.

MetaH는 이러한 아키텍처 기반의 명세를 바탕으로 한, 다시 말해서 Architecture Oriented Development Process를 지원하면서 보다 복잡하고 Hard Real Time System에서의 다양한 시스템 개발을 위한 모델링과 분석방법을 통한 시스템 설계 및 구현을 이룬다. MetaH가 가진 언어로서의 특징은 전통적인 언어로 쓰인 코드 모듈을 어플리케이션을 형성하기 위해 조립하는 방법을 규정한다. 그리고, 특정 하드웨어 시스템의 구조를 규정하며 소프트웨어가 하드웨어에 할당되는 방법도 규정한다. 또한, 특정한 종류의 객체를 제공하며 다양한 분석을 수행한다.

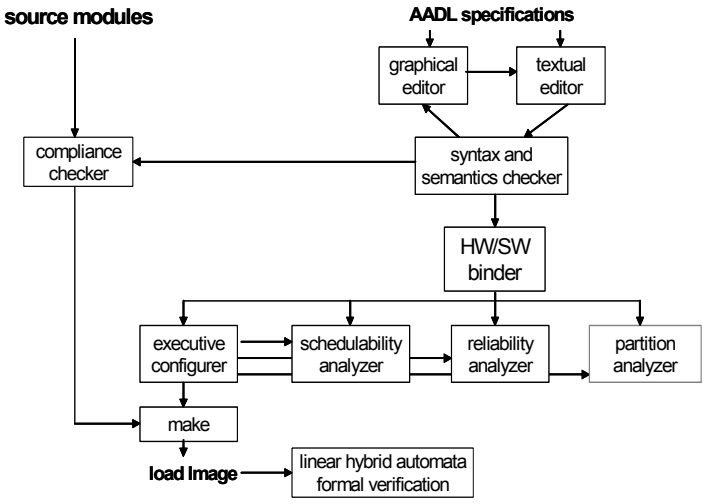
MetaH의 툴셋으로서의 특징을 보면, 단순한 CASE 도구가 아니라, Hard real-time, fault tolerant, safely /securely partitioned, scalable multi-processor systems 을 구현하기 위해서 초점이 맞추어진 툴셋이다. 또한 Domain-specific generators, Reengineering tools, module libraries 등등의 오픈 인터페이스를 가진 통합 툴셋이며, 설계와 구현이 각각 정형화 된 모델링(Modeling)과 분석(analysis)의 과정으로 밀접하게 짝을 이루어 구현되어진다는 특징을 지니고 있다.

MetaH 툴셋은 그래픽, 텍스트 포맷으로 MetaH 명세를 변환하며 자동으로 실행 이미지를 생성한다. 또한, 실시간 스케줄 가능성, 신뢰성, 안전, 보안 모델링 분석 도구 등을 사용하여 다양한 분석을 수행한다. MetaH 툴셋에서 사용하는 MetaH 언어는 코드 모듈의 인터페이스와 속성을 기술하며 코드 모듈들이 상위 레벨의 객체로 조립되는 방법도 기술한다. 객체는 두 개의 그룹(소프트웨어, 하드웨어 객체)으로 분류된다. 소프트웨어 객체는 두 개의 서브그룹으로 나뉘어 지는데 소스객체와 소스객체의 그룹화를 기술하는 상위 레벨 객체이다. 유사하게 하드웨어 객체는 하드웨어의 특정 부분을 기술한 하드웨어 컴포넌트 객체와 그룹화를 기술한 상위 레벨 객체로 나뉘어 진다. [표 16]는 MetaH에서 사용하는 객체의 리스트를 보여준다.

Software Source Objects	Hardware Component Objects	Software Higher-Level Objects	Hardware Higher-Level Objects
event, port, type, type package, monitor, subprogram, package.	event, port, type, type package monitor, memory processor, device channel	application, error model, connection, process macro, mode, path	application, error model, connection, system

[표 16] MetaH 객체 리스트

MetaH 툴셋의 시스템 구조는 다음 [그림 61]와 같다. 각 부분의 기능을 간략히 살펴보면 다음과 같다. 첫째, 그래픽과 텍스트 엔트리(graphical and textual entry) 기능으로 MetaH 언어는 그래픽, 텍스트 문법(Syntax) 모두를 가지며, 그래픽, 텍스트 버전 각각은 그래픽, 텍스트 편집기를 사용해 편집 가능하고 두 표현사이에는 자동 변환을 지원한다. 둘째, 소프트웨어, 하드웨어 바인더 기능으로서 다중 프로세서 시스템의 각 프로세스를 특정 프로세서에 바인딩 시키며 프로세스 사이에 공유되는 패키지, 모니터는 특정 메모리에 바인딩 시킨다. 또한 객체를 특정한 하드웨어 객체에 바인딩 하는 역할을 수행한다. 셋째, 스케줄 가능성 모델링과 분석으로서 MetaH 툴셋은 자동적으로 스케줄 가능성 모델을 생성하고 분석을 수행한다. 이는 어플리케이션이 스케쥴링 되는지 여부와 모든 프로세스가 규정된 빈도와 마감시간안에 실행되는지 여부를 분석한다. 넷째, 신뢰성 모델링과 분석 기능으로 결함 이벤트와 에러 상태의 집합을 규정하기 위해 사용하며 자동으로 Markov 신뢰성 모델을 생성한다. 마지막으로 안전, 보안 모델링과 분석 기능으로서 각 프로세스를 위한 시공간 보안을 제공한다. 예를 들어, 불안전한 객체의 부적절한 동작이 안전한 객체의 동작에 영향을 끼칠 수 없게 확인하며 객체가 권한을 갖고 있지 않은 데이터를 포함하거나 접근할 수 없도록 확인한다.



[그림 61] MetaH 툴셋의 시스템 구조

MetaH를 통해서 시스템을 설계하고 구축할 때 이러한 시스템에 대한 구성요소가 되는 소프트웨어 및 하드웨어 Object 들에 대한 명세를 통해 구축해야 할 시스템에 대한 특성 및 보다 자세한 시스템 요소들을 파악하고 각 Object 들에 대한 기술내용을 명세한다. 특

히, 크게 Software 와 Hardware 두 부분으로 분류하고 이를 구성하는 각 Object 들의 Architecture를 명세하고 표현함으로써 그 내용을 기술하게 되는데, 즉 다시 말하자면 MetaH 에서의 Specification를 보면 Software Object 뿐만 아니라 Hardware Object 에 대해서도 명세가 가능하다.

Hardware Architecture Specification는 사용하기 편리하게 하기 위해서 두 레벨로 나누어지는데, High Level 에서는 예전에 이미 선언된 프로세서 타입이라던지, 프로세서와 관련된 포트, 이벤트, 모니터, 디바이스 및 채널 등을 결합하여 멀티프로세서 시스템 구성을 이룬다. 대부분의 시스템 개발에 대한 노력은 이미 기존에 개발되어진 Interface Code, low-level Hardware architecture specification에서의 Processor, channels, devices 의 Instance 들을 이용한 High-level
System hardware architecture를 구성하는 것으로 전개가 된다. 다시 말하면, 큰 사이즈의 Multi-Processor System에 대한 구성은 이미 만들어진 하드웨어에 의존적인 Interface Code인 Hardware 부품들의 모음으로부터 컴포넌트들을 선택, 이를 함께 조립하는 것으로 구성을 할 수 있다. 이러한 구성에 대한 마지막 부분에서는 대상이 되는 하드웨어의 실제 물리적인 인스턴스를 조립함으로써 전체적인 시스템 아키텍처를 구성하게 된다.
여기에서 새로운 종류의 low-level hardware object 들을 추가하는 데에는 다음과 같은 세 단계가 필요하다. 첫째, Device Driver 와 같은 Hardware Object Specification에 대한 Source file 목록을 준비한다. MetaH 에 의해서 다시 재구성이 될 때 이러한 Low-level의 Source file 은 그 인터페이스에 대한 명세를 반드시 다시 해주어야 한다. 둘째, MetaH specification은 반드시 Hardware Object 들을 기술한 Specification을 포함해야한다. 셋째, MakeH 는 특정 프로세서 나 Target System에 대한 개발을 하려할 때 특정 개발 Language에 CROSS-Development 한 Toolset 이다. MetaH 가 새로운 타입의 프로세서를 추가하려 했을 때 스크립트(scriopt)나 makefile이 다시 작성되어 MakeH Library에 추가된다.

멀티프로세서 시스템을 조립할 때, 현재의 MetaH 툴셋은 프로세서 상호간의 Communication Buffer를 선언한 소스 코드에 대해서 손수 재구성하는 작업이 필요하다. 이 소스코드는 타겟 패키지 내에 있으며, Software Design Document for the Honeywell Aerospace Compiled Kernel 이라는 도큐먼트를 참고한다. 멀티 프로세서 시스템에 대한 예비 명세 및 분석은 실제적인 물리적 대상 시스템이 없거나, Hardware specific interface code가 작성되어지기 전에 수행되어 질 수 있다. 명세를 하기 위한 대상이 되는 Hardware Object들은 MetaH tool 에 의해서 자동으로 Load 되는 표준 Predeclaration 파일에 기술한다.

유닉스 시스템에서의 표준 Predeclaration File은 $METAH_ROOT/src/targets/standard.mh 이다. MetaH Hardware Specification을 구성할 때 Specification을 바탕으로 실제 물리적인 하드웨어에 대한 조립을 하게 되는데, 이때 충분한 분석을 통해서 그 정확성이 검증되어야 한다. 만약, 이러한 Hardware architecture specification이 정확하지 않는다면, 이에 대한 다양한 분석결과 역시 정확하지 않거나, 그렇지 않으면 MetaH 툴셋을 통해 조립되어진 자동적으로 생성한 실행 가능한 이미지 파

일의 실행이 정확하지 않게 된다. 하드웨어를 이루는 객체들에 대한 선언에서 그 문법과 전체적인 구조는 소프트웨어 선언과 동일하다. Interface와 Implementation Specification은 memory, channel, device, processor, system 들을 위해 정의된다. Port, event, type 과 같은 Object 들은 다른 Object 들로부터 그 기본이 되는 Object 들이며, 다른 Object들에게 하나의 컴포넌트처럼 선언되어질 수 있다. 하지만, 이들 세 Object 들은 그 자체적으로 더 이상 분해될 수 없다.

Software Architecture Specification은 크게 Textual Architecture 명세와 Graphical Architecture 명세 둘로 나누어진다. 그 중 먼저 Textual Architecture Specification을 살펴보면, Lexical Level 에서의 Textual MetaH는 거의 Ada와 같다. 특히, 문자 셋(character set)과 문자 상수(character literal)를 대신하는 Lexical elements, 분리자(separators), 경계자(delimiters)와 주석은 똑같으며 식별자, 숫자문자, 기본문자, 문자열은 정확히 Ada 와 일치한다. 그러나 MetaH에서는 문자 상수와 프라그마(pragma)를 사용하지 않으며, 또한 허용하지지도 않는다.

MetaH Software Architecture에서의 각 Object들에 대한 Interface 및 Implementation Specification에 대해서는 다음과 같다. Interface와 Implementation은 Error model, Modes, Macros, Processes, Monitors, Packages, Subprograms, Type package 등에 대해서 명세 되어 질 수 있다. Interface Specification과 하나나 그 이상의 Implementation Specification에서의 Object 들에 대한 각 클래스로의 선언이 가능하다.
각각의 interface Specification은 정확히 하나의 Class를 가진다. Error model, Mode, Macro, Process, Monitor, Package, Subprogram, Type package 가 모두 그 하나의 Class 가 될 것이다. [그림 61]는 Interface 와 Implementation Specification에서의 Class를 명시하기 위해 사용하는 문법을 나타내고 있다. 이러한 클래스들 중에서 Port, Event, Type 들은 기본 클래스가 되며, Object 들의 다른 클래스들은 Interface 와 Implementation Specification을 가진다. Interface Specification 은 서로 관련 있는 0 또는 그 이상의 Implementation Specification을 가진다.

각각의 Implementation 은 정확히 하나의 관련 있는 Interface를 가지며, 인터페이스 식별자는 주어진 클래스와 별개로 반드시 명확하게 구분되어져야 한다. 하지만 이러한 인터페이스의 클래스가 서로 다르다면, 인터페이스 식별자는 Overloading 될 수 있다.
Interface 와 Implementation Specification은 명세 할 Object 의 클래스에 기초한 두 개의 Main Group으로 나눌 수 있다. 이 두 Object 는 Source Object 와 Higher-level Object이며, Source Object의 경우, 현재 지원되는 Ada와 같이 전형적인 프로그래밍 언어로써 작성된 Source code module의 특징을 설명하거나 모델링 하고 있다. Higher-level Object는 어떻게 Source Object 와 다른 Higher-level Object가 전체 Application 안에서 조립되어지는지를 명세한다. Graphical Architecture Specification의 내용을 보면, Domain Modeling Environment 툴셋인 DoME에서 제공하는 Graphic Editor 와 같은 Editor를 사용해서 Software Architecture 에 대한 명세를 한다. 앞서 이미 이야기 한 각 구성 Object 들에 대한 윈도우 툴바 아이콘들을 통해서 Architecture 명세를 정의하며, 어떤 메뉴와 어떤 프로퍼티 박스를 이용해서 그 내용을 기술하고 설계할 수 있는지에 대해서 나타낸다.