내장형 시스템을 위한 아키텍처 모델링 언어 개발

(Development of Architecture Description Language for Embedded Systems)

문 영 주 † 김 문 주 †† 김 태 효 †††
(Young-Joo Moon) (Moonzoo Kim) (TaiHyo Kim)

요 약 본 논문에서는 Simplified Architecture Description Language (SADL) 이라는 새로운 아키텍처 모델링 언어를 제안하고자 한다. 이는 기존의 아키텍처 모델링 언어의 복잡한 표기법으로 인해서 업계에서 시스템 개발의 전반적인 단계에서 사용이 저조한 실정을 해결하기 위해 제안된 것으로, 내장형 시스템 도메인에 중점을 두고 기존의 아키텍처 모델링 언어 문법의 간략화와 더불어 실제 구현 단계에서 필요한 정보를 모델링 단계에서 명세할 수 있는 표기법을 제안한다. 이와 더불어, 독립적인 다수의 명세 관점을 제안하여, SADL이 표현할 수 있는 다양한 시스템 정보를 각 명세 관점에 따라 제공할 것을 제안함으로써, 사용자에게 혼란을 야기시킬 수 있는 상황을 방지하고자 한다. 본 논문에서는 제안된 아키텍처 모델링 언어와 명세 관점을 바탕으로, 내장형 시스템인 Anti-lock Braking System을 모델링한 결과를 예제로 보여준다.

키워드: 내장형 시스템, 아키텍처 모델링 언어, 명세 관점, Anti-lock Braking System

Abstract This paper proposes a new Architecture Description Language (ADL), called Simplified Architecture Description Language (SADL), to increase the utilization of ADL throughout the development phases of software systems. There have been a number of ADLs with various characteristics for different domains. However, in practice, their complex grammar confines their utilization to the communication among stakeholders in a design phase of software development. To overcome this limitation, SADL provides compact and concise grammar compared to the grammar of existing ADLs. In addition, SADL focuses on an embedded system, that is, its grammar is customized to describe the architecture of embedded systems and specific information related to the implementation of an embedded system. Such an SADL is provided with a number of independent specification views, which enables us to avoid ambiguous interpretation of a given architecture model, by filtering system architecture information according to each view. Using SADL, we show the architecture model of Anti-lock Braking System with regard to the proposed several specification views.

Keywords: embedded systems, architecture description language (ADL), specification views, anti-lock braking system

· 본 연구는 민군겸용기술사업(UM11014RD1)의 연구비 지원으로 수행하였음
· 이 논문은 제40회 추계학술발표회에서 '내장형 시스템을 위한 아키텍처 모델링 언어 개발'의 제목으로 발표된 논문을 확장한 것임

† 비 회 원 : 한국과학기술원 전산학과 교수
youngjoo.k.moon@gmail.com
†† 종신회원 : 한국과학기술원 전산학과 교수
moonzoo@cs.kaist.ac.kr
(Corresponding author임)
††† 비 회 원 : (주) 포멀웍스
taihyo.kim@formalworks.com

논문접수 : 2013년 11월 26일
심사완료 : 2014년 2월 12일

정보과학회논문지: 소프트웨어 및 응용 제41권 제4호(2014.4)

1. 서 론

일상 생활에서 우리는 다양하고 많은 내장형 시스템을 마주하고 있다. 예를 들어, 간단한 스마트폰과 가전 용품뿐만 아니라 자동차와 같은 안전성이 주요 쟁점이 되는 시스템의 내부에도 장착되어 사용된다. 따라서, 내장형 시스템의 행위의 정확성과 안전성을 검증하는 문제는 그 중요성이 점점 더 강조되고 있다.

이를 위해 다양한 명세 및 검증 기법이 제시되었다[1,2]. 그러나, 시스템의 행위가 올바르게 명세 및 검증되더라도, 이를 바탕으로 구현한 결과물이 시스템의 요구사항을 반드시 만족시킨다는 것을 보장할 수 없다. 이는 시스템의 행위 단위요소들이 요소들이 실제 시스템의 구성요소에 전개(deployment)되는 과정에서 발생할 수 있는 예상치 못한 상황들을 고려하지 않기 때문에 발생한다. 따라서, 시스템 구성요소의 추상화, 시스템 구성의 스타일, 표준에 대한 표기법을 제시를 목표로, Acme [3], Wright[4,5], Architecture Analysis and Design Language(AADL)[6] 등과 같은 다양한 아키텍처 모델링 언어(Architecture Description Language)들이 제안되었다. 그러나 기존의 언어들은 사용 도메인의 제한이나 비 기능적 속성(non-functional properties) 표현의 제한으로 인해 내장형 시스템의 아키텍처 명세에는 적합하지 않다. 또한, 복잡한 문법으로 인해 업계에서 그 사용 측면에서 활용성인 저조한 실정이며, 사용된다고 하더라도 시스템 개발의 이해관계자들 사이의 의사소통 수단으로만 사용되고 있다는 단점이 있다.

따라서, 본 논문에서는 좀더 직관적으로 이해하기 쉽고 간결한 표기법을 제공하며, 실제 업계에서 전반적인 시스템 개발 단계에서 사용될 수 있는 새로운 아키텍처 모델링 언어로, Simplified Architecture Description Language(SADL)를 제안하고자 한다. 기존의 아키텍처 모델링 언어는 언어의 표현력을 높이기 위해 제안된 다양한 문법요소들을 사용자가 숙지하는데 많은 노력과 시간을 요구할 뿐만 아니라, 시스템의 도메인 특성 상 사용하지 않는 일부 요소들까지 포함하고 있다. 따라서, SADL은 기존의 아키텍처 모델링 언어의 문법요소를 내장형 시스템 도메인(특히, 웹 기반을 제외한 내장형 시스템)에 중점을 두고 불필요한 문법요소들을 제외하여 사용자가 숙지하기 쉽도록 간결화하였다. 또한 개발자가 내장형 시스템 구현에 필요한 정보를 모델링단계에서 명세할 수 있는 표기법을 제공한다. 이러한 SADL은 시스템을 명세하기 위해 소프트웨어적인 구성요소의 명세뿐만 아니라, 소프트웨어 구성요소들이 수행되는 물리적인 플랫폼인 하드웨어와 시스템이 상호작용하는 환경요소에 대한 명세를 위한 표기법을 포함하고 있다. 이를 통해, 전반적인 소프트웨어 시스템 개발 과정에서, 모델 작성의 시기를 좀더 앞당김으로써 빠른 문제점 발견을 촉구할 수 있다. 이를 통해, [7]의 그림 1과 2에서 살펴볼 수 있는 것처럼 시스템 개발 비용을 줄일 수 있다는 장점이 있다. 이와 더불어, 제안된 SADL에서 제공할 수 있는 정보를 다양한 관점(view)에 따라 독립적으로 살펴보기 위해 다수의 명세 관점을 제안하고, 각 명세 관점 따라 제안된 SADL 표기법을 분류했다. 이를 통해, SADL을 이용하여 시스템을 명세할 때에는 각 명세 관점 별로 표현 가능한 시스템 정보의 제약 사항 및 가이드라인을 사용자에게 제공할 수 있으며, 명세된 모델을 해석할 때에는 다양한 정보로부터의 혼란 야기를 방지할 수 있다는 장점이 있다. 제안된 아키텍처 모델링 언어와 명세 관점을 이용한 모델링 예로, Anti-lock Braking System(ABS)을 각 명세 관점에 맞추어 SADL로 모델링 한 결과를 보여준다.

따라서, 본 논문의 기여도는 다음과 같이 요약될 수 있다.

1) SADL 제안: 내장형 시스템의 소프트웨어와 하드웨어 및 환경 요소와 더불어 시스템 구현에 필요한 정보를 표기할 수 있는 간결한 표기법을 제공함으로써 시스템 개발의 전 단계에서 사용 가능성을 높일 수 있는 새로운 아키텍처 모델링 언어인 SADL을 제안한다.
2) 다수의 명세 관점 제공: 복잡하고 다양한 시스템 정보를 독립적인 명세 관점(view)에 따라 제공함으로써, 혼란을 야기시키지 않고 사용자에게 시스템 정보를 제공할 수 있다.

본 논문은 다음과 같은 구성으로 이루어져 있다. 2장에서는 본 연구와 관련된 연구를 살펴봄으로써 본 연구의 동기를 보여준다. 3장에서는 본 논문에서 제안하는 SADL의 표기법을 살펴본다. 이를 바탕으로 시스템의 속성 명세를 위한 부가적인 표기법은 4장에서 살펴본다. 5장에서는 본 연구에서 제안된 다수의 명세 관점에 대한 소개와 이에 따른 SADL 표기법의 사용법 등을 살펴본다. 6장에서는 SADL과 제안된 각각의 명세 관점을 이용한 아키텍처 모델링 예제를 ABS를 통해서 보여준다. 7장에서는 본 논문의 내용을 요약하고 향후 연구 방향에 대해 살펴본다.

2. 관련 연구

시스템 아키텍처 모델링의 필요성은 내장형 시스템 개발에서 비롯된 것은 아니다. 기존의 시스템 개발 과정에서도 그 필요성이 대두되어 왔으며, 이에 따라 Acme, Wright, AADL과 같은 다양한 아키텍처 모델링 언어들이 제안되었다[3,4,6,8-10].

Acme는 소프트웨어 개발에 필수적인 아키텍처 요소와 기존의 다양한 아키텍처 모델링 언어들의 공통된 요소들을 통합하기 위해 제안된 것으로, 범용 도메인에서 사용 가능하다. 컴포넌트와 그 사이의 결합자(connector) 등의 문법 요소를 표현하는 텍스트와 그래픽 표기법을 제공하고 있다. 그러나 범용 도메인 지원 및 다른 아키텍처 모델링 언어들과 통합을 목표로 제안된 만큼 내장형 시스템과 같은 특정 도메인에 적합한 내용의 명세에 제한이 있다는 단점이 있다.

Wright는 시스템 컴포넌트간의 상호작용(interaction)을 중점적으로 모델링하는 아키텍처 모델링 언어로, 주로 병행 시스템(concurrent system)의 통신 수행을 명세하기 위해 사용된다. 컴포넌트와 결합자 등의 문법 요소를 위한 텍스트 표기법을 제공하고 있다. 결합자에 의해 연결되는 컴포넌트의 명세는 Communicating Sequential Processes(CSP)[11]의 형식으로 표현한다. 또한 timing 요구 사항과 같은 일부 비 기능적 속성의 기술을 지원할 뿐만 아니라, Acme와 결합하여 Acme에서 비 기능적 속성 표현이 가능하도록 한다. 그러나 이는 시스템 구성요소의 통신 수행에만 중점을 두고 있으므로, 실시간 내장형 시스템의 모델링에 제한이 있다는 단점이 있다.

AADL은 항공기 컨트롤 시스템 모델링을 위해서 제안된 것으로, 실시간 내장형 시스템 모델링을 염두에 두고 개발되었다. 앞에서 언급한 Acme와 Wright와 같이 컴포넌트-결합자 패러다임을 따르고 있으며, 이를 표현하기 위한 텍스트 및 그래픽 표기법을 제공하고 있다. 또한, timing과 신뢰성(reliability)과 같은 시스템의 비 기능적 속성 기술을 지원한다. timing 정보의 경우는 각 컴포넌트의 속성으로 표현이 되고, 다른 속성들은 기본적인 AADL 문법의 확장을 고려해 개발된 라이브러리 형태의 annex[12,13]로 표현된다. 이러한 관점에서 살펴보았을 때, 내장형 시스템을 위한 아키텍처 모델링 언어로 가장 적합한 언어로 볼 수 있다. 그러나 이는 언어 표기법의 복잡성을 야기할 뿐만 아니라 사용자에게 가파른 학습 곡선이 요구된다는 단점이 있다.

3. SADL(Simplified Architecture Description Language)

이 장에서는 내장형 시스템의 전반적인 개발단계에서 사용할 수 있는 새로운 아키텍처 모델링 언어인 SADL을 정의한다. SADL은 엔티티(entity)와 연결 관계(relation)의 표기법으로 구성되어 있으며, 이들은 각각 시스템을 구성하는 구성요소들과 그들 사이의 관계(relation)를 명세하기 위해 사용된다. 기본적으로 SADL은 다음과 같은 원칙을 바탕으로 개발되었다.

- 그래픽 인터페이스를 통한 모델 작성을 염두에 둔다.
- 기존의 아키텍처 모델링 언어의 복잡한 표기법을 지양하고, 각 표기법을 간결화한다.
- 시스템 개발 단계에 전반적으로 사용 가능하도록, 시스템의 구조적 명세와 더불어 구현에 필요한 정보를 모델링 단계에서 명세할 수 있는 표기법을 제안한다.

즉, SADL의 표기법은 기존의 아키텍처 모델링 언어에 비해 구성요소 명세는 내장형 시스템의 특성을 기반으로 하여 간결화하고, 구성요소간의 의존성 및 속성 정의 부분은 강화시켰다. 예를 들어, AADL과 비교했을 때, 원거리 서버에서 작동하는 프로그램과 같은 명세는 SADL에서 제외시키고, 공유 데이터 사용(데이터 의존성)과 프로세스간의 의존성 등에 대한 명세를 첨가하여 시스템 구현에 좀더 근접한 명세가 가능하도록 하였다.

3.1 엔티티(Entity)

SADL의 문법 요소 중 하나인 엔티티는 시스템의 구성요소를 표현하기 위해 제안되었다. 총 10개의 엔티티가 존재하며, 엔티티의 특성과 모델 작성 시 효율적인 선택 및 사용을 위해 3개의 그룹으로 분류되어 있다. 각각의 그룹은 물리적 특성이나 시스템의 내부 혹은 외부에서의 접근에 관련된 특성에 따라 소프트웨어, 하드웨어, 리소스의 3개의 그룹으로 분류된다. 표 1은 이러한 3개의 그룹과 이에 따른 엔티티의 분류 및 각 엔티티의 그래픽 표현형을 보여준다. 하드웨어 그룹은 프로세서(processor), 메모리(memory), 연결(connection), 디바이스(device)와 같은 4개의 엔티티를, 소프트웨어 그룹은 유닛(unit), 프로세스(process), 유닛 그룹(unit group), 프로세스 그룹(process group), 데이터(data)와 같은 5개의 엔티티를, 리소스 그룹은 시스템 구현 시 영향을 미치는 외부 리소스(external resource) 엔티티 1개를 포함하고 있다.

하드웨어 그룹에 속하는 엔티티들은 물리적인 시스템 구성요소로써, 소프트웨어 구성요소들이 수행되는 플랫폼과 시스템 외부의 물리적인 환경 요소를 표현하는 센서와 구동장치(actuator)를 포함하는 디바이스 엔티티로 구성되어 있다. 소프트웨어 그룹은 하드웨어 플랫폼 위에서 실행되는 실행 단위로써, 독립적인 기능뿐만 아니라 실제 구현 단계에서 개별적으로 개발되어야 하는 유닛 엔티티와 유닛 엔티티 내부에서 독립적으로 수행되는 프로세스 엔티티, 그리고 소프트웨어 실행에 직접적으로 영향을 받는 객체로써의 데이터 엔티티가 이에 속한다. 유닛 그룹, 프로세스 그룹 엔티티는 각각 유닛과 프로세스 엔티티의 모임으로 공통된 특성을 가진 유닛과 프로세스 엔티티들을 모아서 묶은 단위로써, 모델의 가독성을 높이기 위해 제안되었다. 리소스 그룹은 실제 시스템을 구성하는 요소는 아니지만, 실제 구현에 있어서 영향을 미치는 외부 요소를 표현하기 위해서 제안된

표 1 그룹 별 엔티티의 그래픽 표현형 및 의미
Table 1 Graphical representation and its meaning for entities w.r.t. groups

Groups	Graphical representation	Meaning
Hardware	processor	a physical platform for running software entities
	memory	a memory
	connection	a physical platform for dataflow between entities
	device	a physical platform corresponding to a sensor or an actuator outside of a system
Software	unit	an independent execution entity in a sense of structure and functionality
	process	an independent execution entity inside a unit entity
	unit group	a collection of unit entities
	process group	a collection of process entities
	data	an object for software entity execution
Resource	external resource	a resource given from outside of a system

엔티티로 현재 외부 리소스 엔티티를 포함하고 있다.

앞에서 언급한 것과 같이 이러한 SADL의 문법은 기존의 아키텍처 모델링 언어의 문법을 간략화시킨 것으로, 이에 대해 간단히 살펴보기 위해 기존의 아키텍처 모델링 언어 중에서 상대적으로 학계와 업계에서 활발하게 사용되고 있는 AADL과 비교해보았다. AADL은 시스템 구성요소를 표현함에 있어서 프로세스의 하위 레벨의 구성요소로서 스레드(thread)와 원거리 서버 등에서 동작하는 서브프로그램(subprogram) 등과 같은 소프트웨어 구성요소들을 포함하고 있다. 그러나 이러한 AADL의 구성요소들은 각각 개발 단계에서 개발자가 자의적으로 결정해야 하는 문제(즉, 멀티 스레드들의 동시성 쟁점)들을 야기하거나 혹은 웹 기반 내장형 시스템과 같이 특화된 경우를 명세하기 위해 사용되는 문법 요소로써 언어의 높은 복잡도를 야기하기도 한다. 따라서, 이와 같은 요소들을 문법 요소에서 제외하여 간략화 하였다. 이는 언어 사용자의 학습 곡선을 완만하게 하므로 이를 통해 아키텍처 모델링 언어의 사용을 높일 수 있다.

또한 제외된 문법요소들의 명세는 기존의 소프트웨어 명세 언어와의 연계를 통하여 독립적으로 수행할 수 있을 것이다. 연계에 관련된 부분은 본 논문의 범위에서 벗어나므로 다루지 않지만, 향후 연구로써 수행될 것이다.

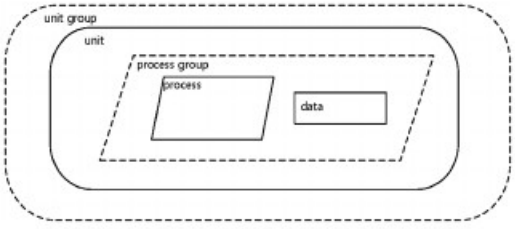

그림 1 소프트웨어 그룹 엔티티의 포함 관계
Fig. 1 Inclusion relation between entities in Software group

SADL의 엔티티를 구성하는 3개의 그룹은 서로 독립적이다. 즉, 임의의 엔티티는 자신이 속한 그룹 이외의 다른 그룹에 속한 엔티티를 포함하거나 그 내부에 포함될 수 없다. 다만, 소프트웨어 그룹의 엔티티들은 일부 다른 소프트웨어 엔티티들을 내부에 포함할 수 있다. 이와 같은 포함 관계는 그림 1에서 살펴볼 수 있다. 각각의 상위 엔티티는 하위의 엔티티를 하나 이상 포함할 수 있으며, 하위 엔티티없이 상위 엔티티만 존재할 수 없다. 예를 들어, 아키텍처 모델 작성 시, 유닛 엔티티는 내부에 반드시 하나 이상의 프로세스 엔티티를 포함해야 한다.

3.2 연결 관계(Relation)

엔티티 사이의 연결 관계는 시스템 구성요소 사이의 관계를 명세하기 위한 것으로, 구성요소 사이의 정보 흐름(data flow)과 의존성(dependency) 등의 관계를 표현한다. Role은 이러한 연결 관계를 발생시키는 행위들의 집합으로 다음과 같이 정의된다.

$$\text{Role} = \{\text{creates, uses, transmitted, writes}\}$$

Role의 각 요소들은 차례로 생성, 사용, 전송, 쓰기에 해당된다. 생성은 시스템 초기화 단계에서 시스템 구성요소들 사이의 생성에 관한 정보를 표현하기 위해서 정의되었고, 사용은 시스템 구성요소들 사이의 접근 및 구조적 결합을 표현하기 위해 정의되었다. 전송 및 쓰기는 시스템 구성요소들 사이의 정보 흐름의 방향성을 표현하기 위해 정의하였다.

이를 바탕으로 표 1에서 살펴본 엔티티들의 집합을 Entity라고 했을 때, 연결 관계 R은 다음과 같이 정의된다.

$$R \subseteq \text{Entity} \times \text{Role} \times \text{Entity}$$

이 때, 왼쪽과 오른쪽 엔티티는 Role 수행 관점에 있어서 각각 주체와 객체에 해당된다. 즉, 왼쪽의 엔티티는 오른쪽 엔티티에 주어진 Role을 수행하고, 오른쪽 엔티티는 왼쪽 엔티티 수행의 결과로 인해 영향을 받는다고 해석될 수 있다.

이러한 연결 관계 R은 방향성을 지닌 화살표를 이용하여 표현할 수 있다. 즉, 왼쪽과 오른쪽 엔티티를 각각 화살표의 소스(source)와 타겟(target)에 대응시키고, Role의

표 2 엔티티 사이의 연결 관계의 그래픽 표현형 및 의미
Table 2 Graphical representation and its meaning for each relation between entities

Types	Graphical representation	Meaning
creates	creates →	A source entity creates a target entity.
uses	uses →	A source entity uses a target entity.
transmitted	transmitted →	A source entity is transmitted to a target entity.
writes	writes →	A source entity writes down on a source entity.

값은 화살표의 레이블(label)로 사용하여 그래픽하게 표현한다. 표 2는 이러한 R의 그래픽 표현형과 각 연결 관계의 의미를 보여준다.

4. 엔티티 및 연결 관계 속성

3장에서는 시스템 아키텍처의 엔티티와 그들 간의 연결 관계를 정의하고, 해당 그래픽 표현형을 살펴보았다. 그러나 이를 바탕으로 한 시스템 명세는 여전히 시스템 구현 단계에서 개발자가 자의적으로 결정해야 할 부분을 남겨두고 있다. 예를 들어, 다수의 프로세스가 존재하는 경우, 3장을 바탕으로 한 시스템 명세는 이러한 프로세스 들에 관련된 정보, 즉, 스케줄링 방침이나 각 프로세스의 활성 주기와 같은 속성들은 고려하지 않고 있다. 이와 같은 속성 정보들은 3장에서 주어진 표기법만으로는 표현하는데 한계가 있으므로, 그 외의 부가적인 표기법으로 명세할 필요가 있다. 이를 위해, 모델 작성 시 표기할 필요가 있는 속성들을 먼저 살펴보고, 각 속성에 적합한 표기법을 살펴보도록 하겠다.

표 3은 부가적인 표기법이 요구되는 엔티티 속성들의 종류와 의미를 보여준다. 이와 같은 부가적인 정보들은 표 1에서 주어진 그래픽 표현형의 내부에 속성 이름과 속성 값의 쌍(pair)의 텍스트 형식으로 부가적으로 표현되는 것을 기본으로 한다. 단, 그래픽 표현형의 변형을 통해 표현이 가능한 일부 속성들은(예를 들어, 데이터 엔티티의 공유 여부 속성) 표 3의 *속성* 열에 부가적으로 표현된 괄호 내에 변형된 해당 그래픽 표기법을 이용하여 표현된다.

표 3에서 살펴본 바와 같이 모든 엔티티가 부가적인 속성을 표현할 필요는 없다. 즉, 시스템 아키텍처의 구성에 영향을 미치지 않는 엔티티의 속성들은 아키텍처 명세에서 제외시켰다. 이와 더불어 속성 표현을 통해 실제 시스템 구현 시 개발자에게 필요한 정보 등을 엔티티 속성으로 규정하여 표기할 수 있도록 하였다. 예를

표 3 그룹 별 엔티티 속성 및 의미
Table 3 Properties and their meaning for entities w.r.t. group

Groups	Entities	Properties (graphical representation)	Meaning
Hardware	processor	scheduling policy	scheduling policy for processes ex) FIFO, LIFO
	memory	access policy	access policy for memory ex) read & write, read only
	device	data format	data type which a current device entity handles
Software	unit	inclusion relation	process or data entities included by a unit entity
	process	priority	priority for process execution
		period	period for process execution
		deadline	deadline for process execution
		execution time	bound for process execution
	unit group	inclusion relation	unit entities included by a unit entity
	process group	inclusion relation	process entities included by a process group entity
	data	data format	data type
		shared data (using a dashed outline)	a data required to be shared

들어, 디바이스 엔티티의 데이터 포맷은 센서나 구동장치에서 사용되는 데이터의 포맷 정보를 미리 결정할 수 있도록 하였다. 이는 시스템의 아키텍처 자체에 영향을 미치지는 않지만, 개발자에게 정해진 포맷에 해당하는 API나 라이브러리 사용에 대한 정보를 미리 제공할 수 있고, 후에 개발자가 자의적으로 결정할 필요가 없다는 장점이 있다. 또한, SADL은 프로세스 엔티티들의 속성으로 우선순위, 활성화 주기, 실행 시간, 데드라인 등을 고려함으로써, 시스템의 비 기능적 속성 표현을 가능하게 하였다. 이는 속성 표현의 측면뿐만 아니라 후에 이를 통해 스케줄링 분석과 같은 모델 검증 및 시뮬레이션의 가능성을 열어두고 있다. 이와 같이 텍스트 표기법이 필요한 경우 외에, 데이터 엔티티의 속성 중 공유 데이터 부분은 그래픽 표현형의 변형을 통해 표기할 수 있도록 하였다. 즉, 공유 데이터의 경우, 기본적인 데이터 엔티티의 그래픽 표현형의 테두리를 점선으로 표현함으로써 해당 데이터 엔티티가 다른 엔티티와 공유된다는

표 4 Role 요소들의 속성
Table 4 Properties and their meaning for Role elements

Types	Properties (graphical representation)	Meaning
transmitted	asynchronous sending (using a dashed line)	Data is transmitted asynchronously.
writes	asynchronous writing (using a dashed line)	Writing/updating data is conducted asynchronously.

것을 표현한다.

표 4에서는 현재 총 4개의 Role 요소들 중에서 전송과 쓰기에 대한 속성의 표현을 살펴보고 있다. 정보의 동기적/비 동기적 속성의 표현은 기본 그래픽 표현형인 화살표의 연결선을 점선을 이용하여 비 동기적인 전송 및 쓰기를 표현한다.

5. 명세 관점(Specification Views)

SADL은 시스템의 구성요소와 그들 간의 연결 관계의 명세를 통해 다양한 시스템 정보를 제공한다. 그러나, 하나의 명세서에 다양한 정보들이 동시에 표현되는 경우, 이해관계자들에게 혼란만 가중시킬 수 있다. 이를 방지하기 위해, [14]에서는 의존성, 인터페이스, 결합, 상황(context), 구조, 상호작용 등 다양한 명세 관점을 제시하고 있으며, 각각의 명세 관점에 따라 여과된 시스템 정보를 이해관계자들에게 제공하는 방법을 제안했다. 그러나 각각의 명세 관점에 따른 다양한 시스템 정보 명세서를 사용하는 경우, 다양한 명세서들 간의 표기법의 일치성 문제와 각 명세서를 이해하기 위해서 개발자들의 많은 노력이 요구된다는 점에서 실효성 문제가 제기된다[15].[1] 따라서, 본 논문에서는 다양한 명세 관점 중 일부, 즉, 결합, 의존성, 정보 흐름 명세 관점을 선택하였다. 각각의 명세 관점에 대한 설명은 아래와 같다.

- 결합 명세 관점: 시스템을 구성하는 하드웨어와 소프트웨어의 구조적 연결 정보와 더불어 이들 구성요소 사이의 사용/접근 정보 표현 및 소프트웨어 구성요소들의 계층 관계를 표현한다. 또한, 시스템 구성에 직접적으로 참여하지 않지만, 시스템이 접근하여 사용하는 외부 리소스에 대한 사용/접근 표현도 포함하고 있다.
- 의존성 명세 관점: 소프트웨어 구성요소 간의 계층 관계와 구성요소들 사이의 생성 및 초기화에 따른 의존성을 표현한다.
- 정보 흐름 명세 관점: 소프트웨어와 하드웨어 구성요소 사이의 정보 흐름과 그 특성 및 이와 관련된 정보의 타입과 공유 속성 등을 표현한다.

1) [15]에서는 UML을 대상으로, 다양한 명세 관점의 실효성에 대해 언급하고 있다.

표 5 명세 관점 별 사용되는 SADL 문법 요소
Table 5 Available grammar elements in SADL for each specification view

Specification views	Grammar elements in SADL	
	Entities	Role
composition	- processor - memory - connection - device - unit & unit group - process & process group - external resource	- uses
dependency	- unit & unit group - process & process group	- creates
dataflow	- device - connection - unit & unit group - process & process group - data	- transmitted - writes

이를 바탕으로 표 5는 각 명세 관점 별 사용되는 엔티티와 Role 요소들을 정리하여 보여준다.

표 5에서 알 수 있듯이, 독립적인 명세 관점의 특성은 Role 요소의 사용에서 확연하게 드러난다. 즉, 구조적 결합 및 사용/접근 관계의 표현은 '사용'을 통해서, 의존성 관계는 '생성'을 통해서, 정보 흐름은 '전송' 및 '쓰기'을 통해서 표현하고 있으며, 각각의 Role 요소들은 해당 명세 관점 이외에는 사용되지 않는다.

6. Anti-lock Braking System(ABS) 예제

ABS는 자동차와 운전자의 안전을 도모하기 위해 개발된 장치로 자동차의 속도가 급작스럽게 변하는 경우, 브레이크의 압력을 조절하여 자동차가 안전하게 속도를 늦출 수 있도록 도와주는 역할을 한다. 즉, ABS는 자동차의 바퀴에 락이 걸려서 미끄러져 사용자가 브레이크를 제어하지 못하는 위험한 상황이 오지 않도록 막아주고 제동거리를 줄여주는 역할을 한다.

이러한 ABS[2]는 바퀴 속도 센서(wheel speed sensor), 브레이크 페달 센서(brake pedal sensor), 자동차 속도 센서(car brake sensor), 유압 장치(hydraulic motor), ABS 컨트롤 유닛(ABS control unit)으로 이루어져 있으며, 각 센서로부터의 정보는 CAN 버스 네트워크를 통해 컨트롤 유닛으로 전달된다. ABS 컨트롤 유닛은 일종의 컴퓨터로 빠른 계산을 통해 자동차 바퀴에 가해질 압력을 계산해서 그 결과값을 유압장치로 전달한다.

2) 일반적으로, ABS는 바퀴 속도 센서, 브레이크 페달 센서, 자동차 속도 센서(car speed sensor)를 포함하여, 브레이크 압력을 계산한다. 현재 아키텍처 모델링 예에서는 자동차 속도 및 바퀴 속도 센서는 통합하여 표현하였다.

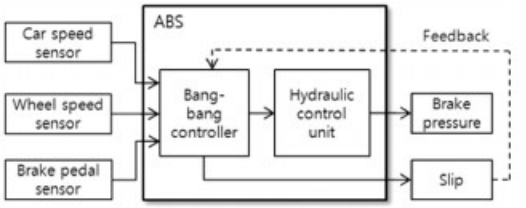

그림 2 ABS 시스템 다이어그램
Fig. 2 A diagram of ABS system

그림 2는 이와 같은 ABS의 블록 다이어그램을 보여준다.

6.1 결합 명세 관점에 따른 ABS 명세

그림 3[3]은 ABS를 결합 명세 관점으로 명세한 결과를 보여준다. 즉, ABS의 하드웨어와 소프트웨어 구성요소의 구조적 연결 정보와 구성요소들 사이의 사용/접근 정보 및 소프트웨어 구성요소들의 계층 관계를 보여주고 있다. 이를 통해 외부 환경과 상호작용을 하는 센서와 구동장치가 ABS 컨트롤 유닛과 CAN 네트워크 버스로 연결되어 있고, ABS 컨트롤 유닛은 wired로 연결된 FlashRom 메모리와 microprocessor를 바탕으로 주어진 기능을 수행하고 있다는 것을 알 수 있다. 또한, ABS 컨트롤 유닛의 내부는 init, monitoring speed, calculating pressure 프로세스 엔티티들로 구성이 되어 있음을 알 수 있다.

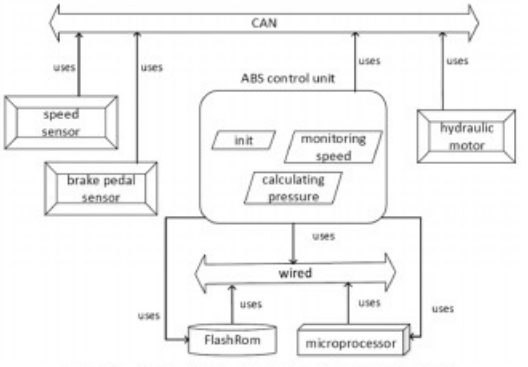

그림 3 결합 명세 관점에 따른 ABS 명세
Fig. 3 A specification of ABS w.r.t. the composition view

6.2 의존성 명세 관점에 따른 ABS 명세

그림 4는 ABS를 의존성 명세 관점으로 명세한 결과를 보여주는 것으로, ABS 컨트롤 유닛을 구성하는 3개의 프로세스의 생성 시 의존성 관계를 보여주고 있다. 즉, init 프로세스 엔티티에 의해 나머지 2개의 프로세스 엔티티가 생성됨을 보여준다.

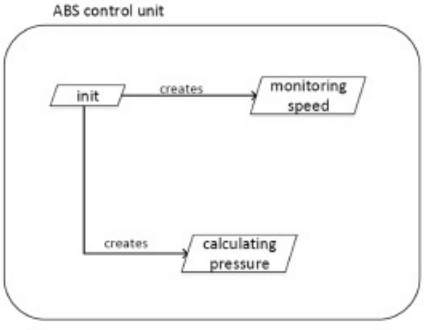

그림 4 의존성 명세 관점에 따른 ABS 명세
Fig. 4 A specification of ABS w.r.t. the dependency view

6.3 정보 흐름 명세 관점에 따른 ABS 명세

그림 5는 ABS를 정보 흐름 명세 관점으로 명세한 결과를 보여준다. 이는 구성요소들 사이의 정보 흐름의 방

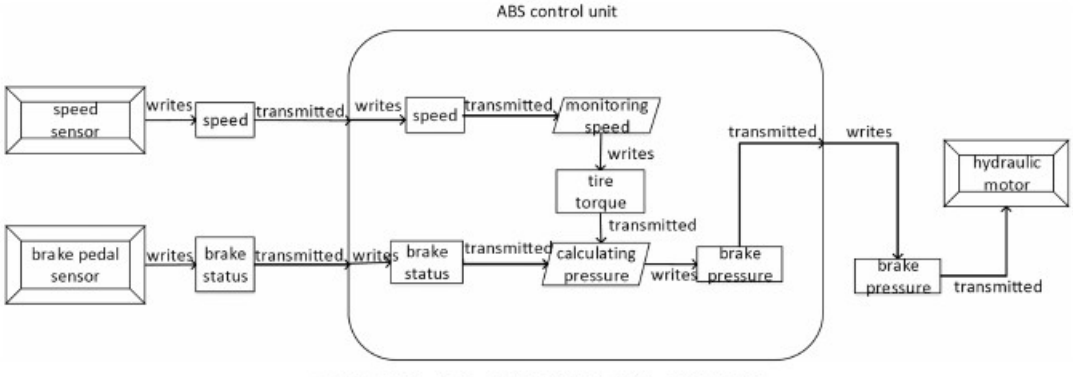

그림 5 정보 흐름 명세 관점에 따른 ABS 명세
Fig. 5 A specification of ABS w.r.t. the dataflow view

3) 그림 3에서는 SADL 표기법 사용의 이해를 돕기 위해 그래픽 표현형을 이용한 명세에 중점을 두었다. 이는 그림 4와 5에도 동일하게 적용되었다.

향성 및 전달되는 데이터의 타입 등에 대한 정보를 제공한다. 따라서, 정보 흐름의 방향성에 대해 다음과 같이 해석할 수 있다. 속도 센서에서 모니터링 한 속도(speed data 엔티티)는 ABS 컨트롤 유닛에 보내지고 이는 다른 데이터 정보(즉, brake status data 엔티티)와 함께 ABS 컨트롤 유닛에서 브레이크 압력(brake pressure)을 계산하는데 사용되며, 그 결과는 유압 장치에 보내진다는 것을 알 수 있다.

7. 결론 및 향후 연구

본 논문에서는 내장형 시스템 구현에 적합한 아키텍처 모델링 언어로 SADL을 제안했다. 이는 기존의 아키텍처 모델링 언어를 간략화하고 동시에 개발자들의 의견을 바탕으로 한 표기법의 확대화를 통해 개발되었다. 또한 복잡한 시스템 정보가 동시에 제공될 때, 야기될 수 있는 혼란을 방지하기 위하여, 결합, 의존성, 정보 흐름 명세 관점 사용을 제안하여 각 명세 관점에 따라 여과된 시스템 정보를 사용자에게 제공할 것을 제안하였다.

그러나 아키텍처의 정보만으로는 내장형 시스템 행위의 정확성을 보장할 수 없다. 따라서 향후 연구로 SADL과 행위를 모델링 하는 다른 명세 언어와의 연계를 살펴볼 예정이다. 이를 통해 시스템의 아키텍처 정보를 바탕으로 한 시스템 행위 모델링과 더불어 시스템 행위의 시뮬레이션 및 검증을 기대할 수 있을 것이다.

References

[1] L. A. Cortés, P. Eles, and Z. Peng, "Verification of embedded systems using a petri net based representation," *Proceedings of the 13th international symposium on System synthesis*, IEEE Computer Society, 2000.

[2] R. Alur, T. A. Henzinger, and P.-H. Ho, "Automatic symbolic verification of embedded systems," Software Engineering, *IEEE Transactions on* 22(3) (1996): 181-201.

[3] D. Garlan, R. Monroe, and D. Wile, "Acme: an architecture description interchange language," CASCON 1997.

[4] R. Allen, D. Garlan, "A formal basis for architectural connection," ACM TOSEM 1997.

[5] Suleiman, Basem, Vladimir Tosic, and Eldar Aliev, "Non-functional property specifications for WRIGHT ADL," *Computer and Information Technology, 2008, CIT 2008, 8th IEEE International Conference on*, IEEE, 2008.

[6] P. H. Feiler, D. P. Gluch, J. J. Hudak, and B. A. Lewis, "Embedded system architecture analysis using SAE AADL," (No. CMU/SEI-2004-TN-005)

[7] Lars-Ola Damm, "Early and cost-effective software fault detection," Blekinge Institute of Technology (2007).

[8] Bernard Berthomieu et al., "Formal Verification of AADL models with Fiacre and Tina," 5th International Congress and exhibition ERTS2, 2010.

[9] Yves LaCerte, "AADL and MDA - Early Experience Applied to Aircraft - Weapon Integration," AADL Standardization Meeting, Seal Beach, Jan 2005.

[10] Huangjing Yu and Yang Yang, "Latency Analysis of Automobile ABS Based on AADL," Industrial Control and Electronics Engineering (ICICEE), 2012 International Conference on, IEEE, 2012.

[11] Charles Antony Richard Hoare, "Communicating sequential processes," Blekinge Institute of Technology, 2007.

[12] R. B. Frana, et al., "The AADL behaviour annex-experiments and roadmap," *Engineering Complex Computer Systems, 2007, 12th IEEE International Conference on*, IEEE, 2007.

[13] P. Dissaux, et al., "AADL behavioral annex," *Proceedings of DASIA conference, Berlin*, 2006.

[14] IEEE Std 1006™1016™-2009, IEEE Standard for Information Technology - Systems Design - Software Design Descriptions.

[15] M. PETRE, UML in practice, In: Proceedings of the 2013 International Conference on Software Engineering, IEEE Press, pp.722-731, 2013.

문 영 주

2004년 고려대학교 컴퓨터학과 졸업(학사). 2006년 고려대학교 컴퓨터학과 졸업(석사). 2011년 라이덴대학교 컴퓨터학과 졸업(박사). 2012년 INRIA postdoc. 2013년 KAIST 전산학과 연구조교수. 관심분야는 코디네이션 언어(coordination language), 정형기법, 내장형 소프트웨어

김 문 주

1995년 KAIST 전산학과 학사. 2001년 Univ. of Pennsylvania 박사. 2002년~2004년 SECUi.COM 차장. 2004년~2006 년 POSTECH 연구원. 2006년~2012년 KAIST 전산학과 조교수. 2012년~현재 KAIST 전산학과 부교수. 관심분야: Concolic 테스팅, Concurrency 테스팅, 정형검증, 내장형소프트웨어

김 태 효

1998년 카이스트 전산학과 졸업(학사) 2000년 카이스트 전산학과 졸업(석사) 2007년 카이스트 전산학과 졸업(박사) 2007년~현재 (주)포멀웍스 재직중. 관심분야는 소프트웨어 정적 분석, 정형 기법, 소프트웨어 테스팅, SCADA 보안