Smart Media Journal / Vol.14, No.5 / ISSN:2287-1322
https://dx.doi.org/10.30693/SMJ.2025.14.5.34

스마트 IoT 시스템의 신뢰성 및 안전성 검증을 위한 프로세스 대수 방법론

(A Process Algebraic Methodology for Verifying Reliability and Safety of Smart IoT Systems)

송준섭*, 이문근**
(Jun Sup Song, Moon Kun Lee)

요 약

디지털 트윈 기술은 물리적 객체와 사이버 세계 간의 동적 상호작용을 가능하게 하며, 스마트 IoT 시스템의 신뢰성과 안전성을 보장하는 데 필수적이다. 그러나 기존의 IoT 통합 플랫폼은 시스템 구현에 초점이 맞춰져 있어, 스마트 IoT 시스템의 요구사항 분석과 검증에는 한계가 있다. 본 논문은 이러한 문제를 해결하기 위해 DeViLL(Delta Visual Language and Logic) 방법론을 제안한다. 이 방법론은 dTP-Calculus 프로세스 대수와 SAVE 도구 모음을 활용하여 스마트 IoT 시스템의 요구사항을 정형적으로 명세하고, 분석 및 검증한다. 또한, 이는 검증된 시스템을 openHAB 플랫폼에 통합하여 실제 환경에서의 신뢰성과 안전성을 보장한다. 본 논문에서는 스마트 홈-수영장 안전 모니터링 시스템을 설계 및 구현하는 사례 연구를 통해 DeViLL의 유효성과 실용성을 입증하였다.

■ 중심어 : 프로세스 대수 ; dTP-Calculus ; IoT 통합 플랫폼 ; openHAB ; SAVE

Abstract

Digital twin technology enables dynamic interaction between physical objects and the cyber world and is essential to ensure the reliability and safety of Smart IoT System. However, existing IoT integration platforms focus on system implementation, which has limitations in analyzing and verifying the requirements of Smart IoT systems. To address the issues, this paper proposes the DeViLL (Delta Visual Language and Logic) Methodology. The methodology utilizes dTP-Calculus process algebra and the SAVE tool suite for the formal specification, analysis and verification of the requirements of Smart IoT systems. Furthermore, it integrates the verified systems into the openHAB platform to ensure their reliability and safety in real-world environments. In this paper, both effectiveness and practicality of DeViLL are demonstrated by designing and implementing a smart home-swimming pool safety monitoring system as a case study.

■ keywords : Process Algebra ; dTP-Calculus ; IoT Integration Platform ; openHAB ; SAVE

Ⅰ. 서 론

물리적 객체와 사이버 세계를 연결하는 디지털 트윈은 정보 통신 기술을 활용하여 현실 세계의 물리적 객체를 디지털 형태로 가상화하는 기술이다[1, 2]. 이는 단순한 정적 모델링을 넘어, 물리적 객체와 대응되는 디지털 형태 간의 동적 상호작용을 가능하게 하며, 시스템을 통해 물리적 객체를 제어 및 관리할 수 있다[3].

디지털 트윈 기술은 스마트 IoT 시스템의 요구사항을 충족하지 못하는 상황을 분석할 수 있는 메커니즘을 제공함과 동시에 스마트 IoT 시스템

* 정회원, 전북대학교 전자정보공학부(컴퓨터공학)
** 정회원, 전북대학교 컴퓨터인공지능학부

접수일자 : 2025년 05월 15일
게재확정일 : 2025년 05월 28일
교신저자 : 이문근 e-mail : moonkun@jbnu.ac.kr

의 요구사항을 충족시킴으로써 물리적 시스템의 신뢰성과 성능을 보장해야 한다[4-6].

이러한 분석은 복잡한 상호작용을 하는 스마트 IoT 시스템의 신뢰성과 안전성을 유지하는 데 필수적이며, 이는 물리적 배치 전, 스마트 IoT 시스템의 안전성을 미리 검증함으로써, 오작동으로 인한 위험을 줄일 수 있게 한다.

디지털 트윈을 위해 Home Assistant[7], Domoticz[8], openHAB[9]과 같은 다양한 IoT 통합 플랫폼이 개발되었다. 이러한 IoT 통합 플랫폼은 다양한 IoT 장치를 통합하여 사용자에게 편의성과 효율성을 제공한다. 그러나 이러한 IoT 통합 플랫폼은 디지털 트윈 기술을 통해 스마트 IoT 시스템의 구현에만 초점이 맞춰져 있어, 스마트 IoT 시스템의 요구사항을 사전에 분석하기 어렵다.

이러한 IoT 통합 플랫폼의 문제점을 해결하기 위해, 본 논문에서는 IoT 통합 플랫폼에 프로세스 대수 정형 기법과 이를 기반으로 하는 설계 도구 모음을 적용한 DeViLL(Delta Visual Language and Logic) 방법론을 제안한다.

정형 기법 중 프로세스 대수는 복잡한 시스템의 상호작용을 수학적으로 모델링하고 분석하는 수학적 구조이며, 이는 복잡한 시스템이 갖는 동시성, 비결정성, 재귀성과 같은 복잡한 상호작용을 명세하고, 분석할 수 있도록 도와준다[10]. 프로세스 대수는 다음과 같은 방식으로 스마트 IoT 시스템을 효과적으로 모델링 할 수 있다:

- IoT 장치의 동작을 하나의 프로세스로 표현함으로써 시스템의 각 구성 요소와 이들 간의 상호작용을 명확히 모델링 한다.
- 시스템의 불확실한 상황에서 발생하는 비결정적 동작을 확률 선택 연산자를 활용하여 다룰 수 있다.

이러한 특성들은 스마트 IoT 시스템의 동작과 상호작용을 체계적으로 분석하고 관리하는 데 도움을 준다.

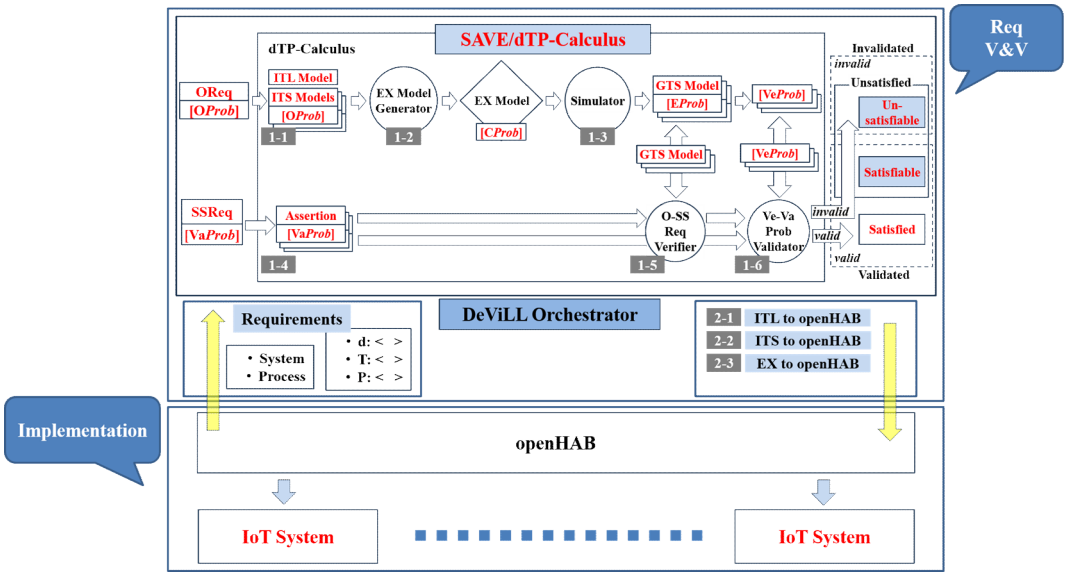

그림 1. DeViLL 방법론

그림 1은 DeViLL 방법론의 개요를 보여준다. DeViLL은 기본적으로, 저자가 개발한, dTP-Calculus[11, 12]라는 프로세스 대수와 이를 기반으로 하는 SAVE(Specification, Analysis, Verification and Evaluation) 도구 모음 및 openHAB을 기반으로 한다.

DeViLL의 접근방법은 크게 2단계로 구성된다. 첫 번째 단계는 스마트 IoT 시스템의 요구사항을 명세, 분석 및 검증하는 단계로 스마트 IoT 시스템의 신뢰성과 안전성을 확인하는 단계이다. 이 단계의 절차는 다음과 같다:

1-1) 스마트 IoT 시스템의 요구사항은 dTP-Calculus로 명세 되며, 이를 SAVE 도구를 통해 ITL 모델(In-The-Large Model)과 ITS 모델(In-The-Small Model)로 시각화한다.
1-2) 실행 모델(Execution Model)은 ITL 및 ITS 모델을 기반으로 스마트 IoT 시스템의 모든 실행 경로를 생성한다.
1-3) 실행 모델에서 각 실행 경로의 시뮬레이션 결과는 GTS 모델(Geo-Temporal Space Model)로 표현된다.
1-4) 안전 및 보안 요구사항은 GTS Logic(Geo-Temporal Space Logic)으로 정의된다.
1-5) GTS Logic으로 정의된 안전 및 보안 요구사항은 GTS 모델에서 분석 및 검증된다.
1-6) 이를 기반으로 안전 및 보안 요구사항에 대한 충족도를 검정한다.

두 번째 단계는 첫 번째 단계를 통해 신뢰성과 안전성이 확인된 시스템을 IoT 통합 플랫폼인 openHAB에 배치하는 단계로, SAVE 도구와 openHAB의 연동을 수행한다. 이 단계의 절차는 다음과 같다:

2-1) SAVE 도구에서 ITL 모델의 특정 프로세스를 openHAB의 Item으로 생성한다.
2-2) SAVE 도구에서 ITS 모델의 특정 프로세스에 대한 특정 통신 동작을 openHAB의 Rule로 생성한다.
2-3) SAVE 도구에서 실행 모델의 특정 실행 경로를 openHAB에서 시뮬레이션한다.

서술한 바와 같이, 디지털 트윈 기술은 스마트 IoT 시스템의 운영을 안전하고 신뢰성 있게 만들기 위한 필수 요소로 자리 잡고 있다. 그러나 기존의 IoT 통합 플랫폼은 시스템의 구현에만 초점을 맞추고 있어, 스마트 IoT 시스템의 요구사항을 분석하고 검증하는 데 한계가 있다. 이를 해결하기 위해, 본 논문은 그림 1에서 제안한 DeViLL 방법론을 통해 IoT 통합 플랫폼에 정형 기법을 적용하였다.

이 방법론은 dTP-Calculus와 SAVE 도구 모음, 그리고 openHAB을 기반으로 스마트 IoT 시스템의 요구사항을 명세, 분석, 검증함으로써 시스템의 안전성과 신뢰성을 보장한다.

나아가, DeViLL 방법론은 스마트 IoT 시스템의 복잡한 상호작용과 비결정적 요소를 체계적으로 관리함으로써, 물리적 배치 이전 단계에서 시스템의 안전성과 신뢰성을 효과적으로 확보할 수 있도록 한다.

본 논문의 구성은 다음과 같다. II 장에서는 IoT 통합 플랫폼과 확률 프로세스 대수에 관한 관련 연구 및 비교 연구를 기술한다. III 장과 IV 장에서는 본 논문의 저자가 개발한 dTP-Calculus와 SAVE 도구를 소개한다. V 장에서는 본 논문의 저자가 제안하는 DeViLL 방법론의 접근방법을 기술한다. VI 장에서는 DeViLL 방법론의 유효성과 실용성을 입증하기 위한 스마트 홈-수영장 안전 모니터링 시스템 예제를 제시한다. 그리고 마지막으로 VII 장에서는 결론과 향후 연구를 기술한다.

Ⅱ. 관련 연구

1. IoT 통합 플랫폼

스마트 IoT 시스템의 자동화는 다양한 기기와 센서를 통합하여 사용자에게 편의성과 효율성을 제공하는 기술로, 최근 몇 년간 IoT(사물인터넷)의 발전과 함께 급격히 성장하고 있다. 이번 장

에서는 다양한 스마트 IoT 통합 플랫폼들을 살펴본다:

- Home Assistant[7]: Home Assistant는 Python으로 개발된 오픈소스 스마트 홈 자동화 플랫폼이다. 사용자가 원하는 모든 장치를 하나의 대시보드에서 제어할 수 있도록 하며, 강력한 커뮤니티와 통합 옵션을 제공한다. Home Assistant는 비교적 사용이 간편하며, 주기적인 업데이트를 통해 새로운 기능이 추가된다. Docker를 이용한 손쉬운 설치가 가능하며, 특히 로컬 서버에서의 데이터 처리를 지향하여 보안성을 높이는 데 초점을 두고 있다.
- Domoticz[8]: Domoticz는 경량의 오픈소스 홈 자동화 플랫폼으로 다양한 센서와 스위치를 지원한다. Domoticz는 저사양 하드웨어에서도 실행할 수 있으며, 사용자가 간단하게 설정할 수 있도록 직관적인 UI를 제공한다. 그러나 다른 플랫폼에 비해 통합할 수 있는 장치의 수가 제한적이라는 단점이 있다.
- openHAB[9]: openHAB(Open Home Automation Bus)은 다양한 스마트 IoT 기기를 통합하여 하나의 플랫폼에서 제어할 수 있도록 하는 오픈소스 자동화 시스템이다. Java 기반으로 개발되었으며, 다수의 프로토콜 및 제조사의 장치를 지원한다. 사용자는 복잡한 설정을 통해 다양한 자동화 시나리오를 구현할 수 있으며, 강력한 커뮤니티 지원이 특징이다.

2. 확률 프로세스 대수

확률 프로세스 대수는 스마트 IoT 시스템의 비결정성에 대한 모델링과 분석을 위해, 확률적 동작과 시간적 요소를 포함한 프로세스 대수이다. 기존 제안된 확률 프로세스 대수는 다음과 같다:

- TPCCS (Timed Probabilistic Calculus of Communicating Systems)[13]: 확률적 선택 연산자와 이산 시간 모델을 사용하여 시스템의 확률적 및 시간적 동작을 모델링 한다. 이 대수는 확률적 선택, 이산 시간의 진행, 그리고 공리화를 통해 시스템의 동작을 정형적으로 기술하고 분석하는 방법을 제공한다.
- PACSR (Probabilistic Communicating Shared Resources)[14]: 실시간 시스템에서 자원의 신뢰성과 사용을 평가하기 위해 자원 고장 확률을 모델에 포함하는 수학적 구조를 제공한다. PACSR은 확률적 전이와 비결정적 전이를 통해 시스템의 자원 상태와 동작을 명확하게 모델링하고 분석할 수 있도록 한다.
- PALOMA (Process Algebra for Located Markovian Agents)[15]: 공간적으로 분산된 에이전트의 상호작용을 모델링 하여, 위치 정보와 확률적 동작을 결합한 분석을 가능하게 한다. 이를 통해 공간적으로 분산된 시스템의 복잡한 동작을 효과적으로 다룰 수 있다.
- dTP-Calculus[11, 12]: dTP-Calculus는 분산 이동 실시간 시스템 내에서 프로세스의 이동과 상호작용, 그리고 시스템의 비결정성을 확률로 모델링 하기 위해 설계된 프로세스 대수 기반의 정형 기법이다.

3. 비교 분석

표 1은 본 논문에서 제안하는 DeViLL 방법론과 기존 IoT 통합 플랫폼을 시스템 및 소프트웨어공학 관점에서 비교 분석한 결과를 보여준다.

표 1. DeViLL 방법론과 기존 IoT 통합 플랫폼의 비교 분석(시스템 및 소프트웨어 공학적 관점)

구분	DeViLL 방법론 (정형 기법+openHAB)	Home Assistant	Domoticz	openHAB
시스템 모델링	dTP-Calculus 기반 정형 명세 ITL/ITS 모델 제공 openHAB과 연동시 프로세스->Item, 액션->Rule 변환	YAML/GUI 기반 구성	센서 기반 UI 구성	DSL(Domain-Specific Language) 기반 내부 모델 제공 (Thing, Item, Rule)
운영 요구사항 분석 (시뮬레이션)	EX 모델 기반 모든 실행 경로 생성 및 시뮬레이션 지원 openHAB과 연동시 각 실행 경로->Rule 조건 트리거	제한적인 자동화 테스트 지원	미지원	Rule 트리거 기반 제한적인 시나리오 테스트 지원
안전 요구사항 검증	GTS 모델 기반 안전 요구사항 정형 검증 지원	미지원	미지원	미지원
적용 범위 및 목적	복잡한 IoT 시스템 정형 명세, 분석 및 검증, 구현	스마트 홈 중심 시스템 구현	단순 센서/스위치 기반 제어 시스템 구현	복잡한 IoT 시스템 연동 및 Rule 기반 자동화 시스템 구현

기존 IoT 통합 플랫폼인 Home Assistant, Domoticz, openHAB은 IoT 장치의 통합과 자동화를 주요 목적으로 설계되었다. 따라서 표 1에서 제시된 바와 같이, 이러한 플랫폼들은 시스템 및 소프트웨어공학 관점에서 스마트 IoT 시스템의 요구사항을 정형적으로 분석하고 검증하는데 한계가 있다.

반면, DeViLL 방법론은 정형 기법을 통해 스마트 IoT 시스템의 요구사항을 사전에 명세, 분석, 검증할 수 있는 기반을 제공한다. 또한, 복잡한 IoT 시스템 구현이 가능한 openHAB과 연동함으로써, 안전성과 신뢰성이 보장된 스마트 IoT 시스템을 구현할 수 있다.

표 2. dTP-Calculus와 기존 확률 프로세스 대수의 비교 분석

구분	dTP-Calculus	TPCCS	PACSR	PALOMA
이동 속성	능동적 및 수동적 이동 표현	이동성에 대한 명시적 정의 없음	이동성에 대한 명시적 정의 없음	에이전트의 공간적 배치 기반 이동성 표현
시간 속성	이산 시간 (준비 시간, 실행 시간, 타임아웃, 데드라인)	이산 시간 (임의 대기 시간, 최소 지연 시간)	이산 시간	이산 및 미분 시간
확률 속성	이산, 정규, 지수, 균등 분포 지원	이산 분포 지원	이산 분포 지원	지수 분포 지원

표 2는 dTP-Calculus와 기존 확률 프로세스 대수를 이동, 시간, 확률 속성 관점에서 비교한 결과를 보여준다. 표 2에 제시한 바와 같이, dTP-Calculus는 능동적 및 수동적 이동 속성을 모두 지원하여 스마트 IoT 시스템의 유연한 동적 이동을 모델링할 수 있으며, 준비 시간, 실행 시간, 타임아웃, 데드라인 등 다양한 시간 속성을 통해 실시간 제약 조건을 표현할 수 있다. 또한, 이산, 정규, 지수, 균등 분포 등 다양한 확률분포를 활용함으로써 시스템 내 불확실성을 정량적으로 다룰 수 있다.

이러한 특징을 기반으로, DeViLL 방법론은 스마트 IoT 시스템의 복잡한 동작 특성을 정형적으로 명세하기 위해 dTP-Calculus를 핵심 정형 기법으로 사용하였다.

Ⅲ. dTP-Calculus

dTP-Calculus는 시간에 따른 프로세스의 다양한 이동을 명세 하기 위한 프로세스 대수인, d-Calculus[16]에 시간과 확률 속성을 추가하여 확장 시킨 프로세스 대수이다. dTP-Calculus는 정규, 지수, 이산, 균등 확률 모델을 지원하여 다양한 확률 모델을 적용한 IoT 시스템을 명세하고 분석할 수 있다. dTP-Calculus의 주요 속성은 다음과 같다:

- 이동 속성: dTP-Calculus는 프로세스의 이동을 표현하기 위해, 이동의 주체와 이동의 방향을 기반으로 In, Out, Get, Put이라는 4가지의 주요 이동을 표현한다.
- 시간 속성: dTP-Calculus는 시스템을 구성하는 프로세스의 시간적 제약 조건을 명세 하기 위해 준비 시간, 실행 시간, 타임아웃, 데드라인, 주기와 같은 다섯 가지의 시간 속성을 표현할 수 있다.
- 확률 속성: dTP-Calculus는 시스템의 비결정성을 이산, 정규, 지수, 균등 분포와 같은 네 가지의 확률 모델을 기반으로 표현할 수 있다.

$P ::= A$	Action	$A ::= \phi$	Empty
$\mid A^{per,n}_{[r,to,e,d]}$	Timed Action	$\mid ch(\overline{msg})$	Send
$\mid P^{per,n}_{[r,to,e,d]}$	Timed Process	$\mid ch(msg)$	Receive
$\mid P_{(pri\_n)}$	Priority	$\mid M$	Movement
$\mid P[Q]$	Nesting	$\mid C$	Control
$\mid P\langle ch\rangle$	Channel	$M ::= m^{pri}(k)\ P$	Movement Request
$\mid P + Q$	Choice	$\mid P\ m(k)$	Movement Permission
$\mid P\{pc\} +_F Q\{pc\}$	Probabilistic Choice	$m ::= in$	In Movement
$\mid P \parallel Q$	Parallel	$\mid out$	Out Movement
$\mid P\backslash E$	Exception	$\mid get$	Get Movement
$\mid A \cdot P$	Sequence	$\mid put$	Put Movement
$F ::= D$	Discrete Distribution	$C ::= new\ P$	Create Process
$\mid N(\mu,\sigma)$	Normal Distribution	$\mid kill\ P$	Kill Process
$\mid Ex(\lambda)$	Exponential Distribution	$\mid exit$	Exit Process
$\mid U(l,u)$	Uniform Distribution		

그림 2. dTP-Calculus의 문법

1. Syntax

dTP-Calculus의 문법은 그림 2와 같으며, 각 문법에 대한 자세한 의미는 다음과 같다:

1) 액션(Action): 액션은 프로세스가 수행하는 동작을 나타낸다. 액션의 종류는 공백(Empty), 통신(Send, Receive), 이동(Movement), 제어(Control)가 있다.

a) 공백(Empty): 공백 액션은 프로세스가 어떠한 동작도 수행하지 않는 상태를 나타낸다.

b) 통신(Send, Receive): 통신 액션은 프로세스 간의 동기화에 의한 통신을 나타내기 위해 사용된다. 통신 액션의 종류는 Send 및 Receive가 있으며, Send 및 Receive가 반드시 한 쌍이 존재해야만 동기화에 의한 통신이 수행된다. Send 및 Receive가 동기화하려면 동기화하려는 두 프로세스 간에 채널이 존재해야 하며, 채널을 통해 주고받는 메시지가 같아야 한다.

c) 이동(Movement): 이동 액션은 이동을 수행하는 요청자(Request) 및 허가자(Permission)로 구분된다:

- In: 액션을 수행하는 프로세스가 다른 프로세스의 내부로 이동한다.

- Out: 액션을 수행하는 프로세스가 다른 프로세스의 외부로 이동한다.
- Get: 액션을 수행하는 프로세스가 다른 프로세스를 자신의 내부로 이동시킨다.
- Put: 액션을 수행하는 프로세스가 자신의 내부에 있는 프로세스를 외부로 이동시킨다.

d) 제어(Control): 제어 액션은 New, Kill, Exit로 구분된다:

- 생성(New): 생성 액션은 프로세스의 내부에 새로운 프로세스를 생성하는 액션이다. 생성되는 프로세스는 자신을 생성한 프로세스보다 높은 우선순위를 지닐 수 없다.
- 강제 종료(Kill): 강제 종료 액션은 다른 프로세스를 종료시키는 액션이다. 종료된 프로세스는 자신보다 높지 않은 우선순위를 지녀야 강제 종료 액션을 수행할 수 있다.
- 종료(Exit): 종료 액션은 액션을 수행하는 프로세스 자신을 종료시키는 액션이다. 종료될 시 내부의 모든 프로세스 또한 같이 종료된다.

2) 시간제한 액션(Timed Action): 시간제한 액션은 시간 속성을 지닌 액션을 나타낸다. 시간제한 액션에서 사용할 수 있는 시간 속성은 [r, to, e, d]이며, 각각 준비 시간, 타임아웃, 실행 시간, 데드라인을 나타낸다. 시간제한 액션의 p와 n은 액션의 주기적 실행을 위한 시간 속성이며, p는 period, n은 액션의 반복 횟수를 나타낸다.
3) 시간제한 프로세스(Timed Process): 시간제한 프로세스는 시간 속성을 명세 한 프로세스를 의미한다.
4) 우선순위(Priority): 우선순위는 프로세스의 우선순위를 표현한다. 우선순위는 0 이상의 정수로 표현되며, 숫자가 높을수록 높은 우선순위를 가진다. 예외적으로 0은 최상위 우선순위를 나타내며, 이러한 우선순위를 가진 액션은 비동기 이동 액션을 수행하기 위해 사용된다.
5) 포함(Nesting): 포함은 P가 Q를 포함하고 있음을 나타낸다. 내부에 있는 프로세스는 외부의 프로세스에 의해 제어된다. 만일 내부 프로세스가 외부 프로세스보다 높은 우선순위를 지닌다면, 허가 없이 외부 프로세스의 밖으로 나갈 수 있다. 포함 관계에 있는 두 프로세스의 동작은 내부 프로세스의 이동을 제외하고 서로 독립적으로 수행된다. 외부의 프로세스가 이동하는 경우, 내부의 프로세스도 같이 이동하게 된다.
6) 채널(Channel): 채널은 프로세스와 연결된 채널의 목록을 나타낸다. 채널을 통해 프로세스는 다른 프로세스와 통신할 수 있다.
7) 선택(Choice): 선택 연산은 여러 액션/프로세스 관계 중 하나를 선택하여 실행시키는 연산을 나타낸다. 선택 연산은 P와 Q 둘 중의 하나의 액션만이 비결정적으로 수행된다.
8) 확률 선택(Probabilistic Choice): 확률 선택 연산은 선택 연산과 유사하지만 하나의 선택을 결정할 때 명시한 확률분포에 의한 확률에 의해 선택이 결정된다. 확률 선택에 대한 확률 모델 유형은 이산 분포, 정규 분포, 지수 분포 및 균등 분포가 있다. 확률 선택 연산을 명세 할 때 주의할 점은 확률의 합이 반드시 1이 되도록 명세를 수행하여야 한다.
9) 병렬(Parallel): 병렬 연산은 병렬 관계에 있는 두 프로세스가 같은 시간대에서 동시에 병렬 실행됨을 의미한다. 병렬 관계에 있는 두 프로세스는 완전히 독립적이다. dTP-Calculus는 프로세스가 수행하는 액션에 대한 병렬 실행은 지원하지 않는다.

10) 시퀀스(Sequence): 시퀀스 액션은 액션 A가 실행된 후, 다음 액션이 실행되는 것을 구분하고 명세 하기 위해 사용된다.
11) 예외 처리(Exception): 예외 처리 연산은 액션이 실행 중에 데드라인 또는 타임아웃에 의해 실패(fault)가 발생하였을 경우, 현재 실패한 액션을 종료한 후, 예외 처리를 통해 다른 액션이 수행되는 것을 명세 하기 위해 사용된다.

2. Semantics

dTP-Calculus의 의미론은 전이 규칙으로 표현된다:

$$\frac{p_1,\ldots,p_k}{q}(r) \quad (1)$$

$p_1,\ldots,p_k$는 전제(premise), q는 전제로부터 도출한 결론(conclusion), r은 부가 조건(side condition)을 나타낸다. 전이 규칙은 부가 조건이 충족되는 조건에서 전제 조건이 성립하는 경우, 전제 조건으로부터 도출한 결론으로 전이할 수 있다.

표 3. dTP-Calculus의 의미

ChoiceL	$\frac{-}{P+Q \rightarrow P}$	ChoiceR	$\frac{-}{P+Q \rightarrow Q}$
Probability ChoiceL	$\frac{-}{P\{pc_1\} +_F Q\{pc_2\} \rightarrow P\{pc_1\}}(pc_1+pc_2=1)$	Probability ChoiceR	$\frac{-}{P\{pc_1\} +_F Q\{pc_2\} \rightarrow Q\{pc_2\}}(pc_1+pc_2=1)$
ParallelL	$\frac{P \rightarrow P'}{P \parallel Q \rightarrow P' \parallel Q}$	ParallelR	$\frac{Q \rightarrow Q'}{P \parallel Q \rightarrow P \parallel Q'}$
Parallel Com	$\frac{P \xrightarrow{A} P', Q \xrightarrow{\overline{A}} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}$	Nesting Com	$\frac{P \xrightarrow{A} P', Q \xrightarrow{\overline{A}} Q'}{P[Q] \xrightarrow{\tau} P'[Q']}$
NestingO	$\frac{P \rightarrow P'}{P[Q] \rightarrow P'[Q]}$	NestingI	$\frac{Q \rightarrow Q'}{P[Q] \rightarrow P[Q']}$
In	$\frac{P \xrightarrow{in(k)\ Q} P',\ Q \xrightarrow{P\ in(k)} Q'}{P \parallel Q \xrightarrow{\delta} Q'[P']}$	Out	$\frac{P \xrightarrow{out(k)\ Q} P',\ Q \xrightarrow{P\ out(k)} Q'}{Q[P] \xrightarrow{\delta} P' \parallel Q'}$
Get	$\frac{P \xrightarrow{get(k)\ Q} P',\ Q \xrightarrow{P\ get(k)} Q'}{P \parallel Q \xrightarrow{\delta} P'[Q']}$	Put	$\frac{P \xrightarrow{put(k)\ Q} P',\ Q \xrightarrow{P\ put(k)} Q'}{P[Q] \xrightarrow{\delta} P' \parallel Q'}$
InP	$\frac{P \xrightarrow{in^{pri}(k)\ Q} P'}{P_{(n_1)} \parallel Q_{(n_2)} \xrightarrow{\delta} Q_{(n_2)}[P'_{(n_1)}]}(((n_1 > n_2) \wedge (n_2 \neq 0)) \vee ((n_1 = 0) \wedge (n_2 \neq 0)))$		
OutP	$\frac{P \xrightarrow{out^{pri}(k)\ Q} P'}{Q_{(n_2)}[P_{(n_1)}] \xrightarrow{\delta} P'_{(n_1)} \parallel Q_{(n_2)}}(((n_1 > n_2) \wedge (n_2 \neq 0)) \vee ((n_1 = 0) \wedge (n_2 \neq 0)))$		
GetP	$\frac{P \xrightarrow{get^{pri}(k)\ Q} P'}{P_{(n_1)} \parallel Q_{(n_2)} \xrightarrow{\delta} P'_{(n_1)}[Q_{(n_2)}]}(((n_1 > n_2) \wedge (n_2 \neq 0)) \vee ((n_1 = 0) \wedge (n_2 \neq 0)))$		

표 3. dTP-Calculus의 의미 (계속)

PutP	$\frac{P \xrightarrow{put^{pri}(k)\ Q} P'}{P_{(n_1)}[Q_{(n_2)}] \xrightarrow{\delta} P'_{(n_1)} \parallel Q_{(n_2)}} (((n_1 > n_2) \wedge (n_2 \neq 0)) \vee ((n_1 = 0) \wedge (n_2 \neq 0)))$		
TickTimeR	$\frac{-}{A^{per,\ n}_{[r,\ to,\ e,\ d]} . P \xrightarrow{\rhd 1} A^{per,\ n}_{[r-1,\ to,\ e,\ d-1]} . P} (r \geq 1)$	TickTime TO	$\frac{A . P \parallel \overline{A} . Q \xrightarrow{\tau \vee \delta} P \parallel Q}{A^{per,\ n}_{[0,\ to,\ e,\ d]} . P \xrightarrow{\rhd 1} A^{per,\ n}_{[0,\ to-1,\ e,\ d-1]} . P} (to \geq 1)$
TickTime SyncE	$\frac{A . P \parallel \overline{A} . Q \xrightarrow{\tau \vee \delta} P \parallel Q}{A^{per_1,\ n_1}_{[0,\ to_1,\ e_1,\ d_1]} . P \parallel \overline{A}^{per_2,\ n_2}_{[0,\ to_2,\ e_2,\ d_2]} . Q \xrightarrow{\rhd 1} A^{per_1,\ n_1}_{[0,\ to_1,\ e_1-1,\ d_1-1]} . P \parallel \overline{A}^{per_2,\ n_2}_{[0,\ to_2,\ e_2-1,\ d_2-1]} . Q} (e_1 \geq 1, e_2 \geq 1)$		
TickTime AsyncE	$\frac{-}{A^{per,\ n}_{[0,\ to,\ e,\ d]} . P \xrightarrow{\rhd 1} A^{per,\ n}_{[0,\ to,\ e-1,\ d-1]} . P} (e \geq 1)$		
TickTime End	$\frac{-}{A^{per,\ n}_{[0,\ to,\ 0,\ d]} . P \xrightarrow{\rhd 1} P}$	Timeout	$\frac{-}{\left(A^{per,\ n}_{[0,\ 0,\ e,\ d]} \backslash E\right) . P \xrightarrow{\rhd 1} E . P}$
Period	$\frac{-}{A^{per,\ n}_{[r,\ to,\ e,\ d]} . P \xrightarrow{\rhd per} A^{per,\ n-1}_{[r,\ to,\ e,\ d]} . P} (n \geq 1)$	Period End	$\frac{-}{A^{per,\ 0}_{[0,\ to,\ 0,\ d]} . P \xrightarrow{\rhd 1} P}$
Deadline	$\frac{-}{\left(A^{per,\ n}_{[r,\ to,\ e,\ 0]} \backslash E\right) . P \xrightarrow{\rhd 1} E . P}$	Sequence	$\frac{-}{A . P \xrightarrow{A} P}$

표 3은 dTP-Calculus의 의미를 정리한 표이다. 각 의미에 대한 세부 내용은 다음과 같다:

1) ChoiceL, ChoiceR: ChoiceL과 ChoiceR은 같은 조건에서 둘 중 하나가 선택되는 행동의 전이 규칙을 나타낸다. 선택되지 않은 프로세스는 실행되지 않는다.
2) Probability Choice: Probability Choice는 명세된 확률에 의해 선택되는 행동의 전이 규칙을 나타낸다. dTP-Calculus에 의해 명세된 확률밀도함수 및 선택에 의해, 확률 선택에 기반한 전이가 수행된다.
3) ParallelL, ParallelR: ParallelL과 ParallelR은 서로 간에 독립적이므로, 서로의 실행 상태와는 관계없이 각 프로세스가 독립 시행되는 전이 규칙이다.
4) ParallelCom: ParallelCom은 서로 간에 동기화되는 행동이 발생하는 경우 반드시 두 프로세스가 동시에 전이되어야 함을 나타내는 전이 규칙이다. 프로세스 P와 Q가 다음 상태로 전이하기 위해 실행되어야 할 액션이 서로 동기화되는 경우, 두 프로세스는 통신에 의해 먼저 동기화된 후에 다음 상태로 전이를 수행한다.
5) NestingCom: NestingCom 전이 규칙은 서로 간에 동기화되는 행동이 발생하는 경우 두 프로세스가 동시에 전이됨을 나타내는 전이 규칙이다. 그러므로 NestingCom은 두 프로세스 P, Q 사이에 동기화가 먼저 수행되어야 한다.
6) NestingO, NestingI: Nesting 전이 규칙은 두 프로세스 P 및 Q가 각각 독립적으로 동작을 수행할 수 있음을 의미한다. 하지만 두 프로세스 사이에 동기화가 요구될 경우, 독립적으로 실행되지 않는다.
7) In, Out, Get, Put: In, Out, Get 및 Put은 프로세스의 상호작용에 의한 이동을 의미한다. In 및 Get 액션은 액션을 수행하는 프로세스가 대상이 되는 프로세스의 내부로 들어가거나, 반대로 액션을 수행하는 프로세스가 대상이 되는 프로세스를 내부로 들여온다. Out 및 Put 액션은 액션을 수행하는

프로세스가 대상이 되는 프로세스의 외부로 나가거나, 반대로 액션을 수행하는 프로세스가 대상이 되는 프로세스를 외부로 내보낸다.

8) InP, OutP, GetP, PutP: InP, OutP, GetP 및 PutP는 우선순위를 적용하기 위해 정의된 전이 규칙이다. 요청을 수행하는 프로세스의 우선순위가 대상 프로세스보다 높을 경우, 이동의 허가가 필요하지 않고 비동기 이동을 수행한다.

9) TickTimeR: TickTimeR은 액션의 준비 시간에 대한 전이 규칙이다. 액션의 준비 시간이 진행되는 동안 준비 시간 r과 데드라인 d가 시간이 흐름에 따라 감소한다. $\triangleright_t$는 t 단위 시간만큼 시간이 지나감을 의미한다.

10) TickTimeTO: TickTimeTO는 액션의 대기 시간에 대한 전이 규칙이다. 액션이 실행되지 않고 대기 중일 때, 시간의 흐름에 따라 타임아웃 시간 to와 데드라인 d가 감소함을 의미한다.

11) TickTimeSyncE: TickTimeSyncE는 동기화 액션의 실행 시간에 대한 전이 규칙이다. 상호작용 액션에서, 수행 요청 액션과 대응하는 수행 허가 액션이 준비되면, 시간의 흐름에 따라 액션의 실행 시간 e와 데드라인 d가 감소한다.

12) TickTimeAsyncE: TickTimeAsyncE는 비동기 액션의 실행 시간에 대한 전이 규칙이다. 비동기 액션의 경우 별도의 대기 시간 없이 바로 실행됨으로 액션의 준비 시간 r이 끝난 뒤 바로 실행 단계가 수행된다. 그 후 실행 시간 e와 데드라인 d가 시간이 지남에 따라 감소 된다.

13) TickTimeEnd: TickTimeEnd는 액션에 종료에 대한 전이 규칙이다. 액션은 실행 시간 e가 완료된 후에 종료가 된다.

14) Timeout: 타임아웃 발생에 대한 전이 규칙을 나타낸다. 타임아웃은 타임아웃 시간 to의 값이 0이 될 경우, 시스템 실패가 발생된다. 만약 예외 처리가 정의되어 있다면, 시스템 실패 후에 정의한 예외 처리에 따라 예외 처리가 수행된다.

15) Period: 주기 실행에 대한 전이 규칙을 나타낸다. 정의한 Period만큼 액션이 반복 실행되며, 반복 실행에 대한 반복 횟수 n이 감소한다.

16) PeriodEnd: 주기 실행의 종료에 대한 전이 규칙을 나타낸다. 반복 횟수 n이 0인 경우, 액션을 더 이상 반복하지 않고 다음 상태로 전이하게 된다.

17) Deadline: 데드라인 위반에 대한 전이 규칙을 나타낸다. 데드라인은 데드라인 d의 시간 값이 0이 되면 발생한다. 타임아웃과 마찬가지로 예외 처리가 정의되어 있다면, 시스템 실패 후에 정의한 예외 처리에 따라 예외 처리가 수행된다.

18) Sequence: Sequence 전이 규칙의 의미는 전제 없이 액션 A가 수행된다면 액션 A에 의해 프로세스 P가 실행된다는 것을 나타낸다.

Ⅳ. SAVE

이번 장에서는 SAVE (Specification, Analysis, Verification and Evaluation) 도구를 소개한다. SAVE는 ADOxx 메타-모델링 플랫폼을 사용하여 개발되었다. SAVE는 저자가 OKRC (OMiLAB Korea Research Center)에서 개발한 시스템 분석 및 검증 도구이며, SAVE 구성 요소는 모델러, 시뮬레이터, 분석기 및 검증기로 구성되어 있다. 그림 3은 SAVE의 아키텍처와 구성 요소를 보여준다.

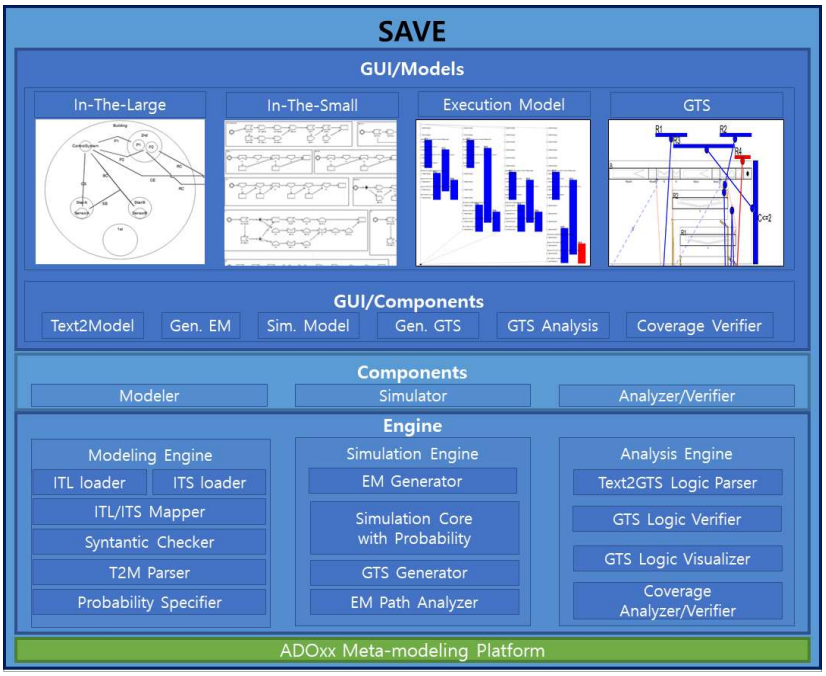

그림 3. SAVE의 아키텍처 및 구성 요소

1. ADOxx 메타-모델링 플랫폼

ADOxx는 OMiLAB에서 개발된 메타-모델링 플랫폼으로, 비즈니스와 엔지니어링 도메인에서 시스템 모델링 도구를 개발할 수 있는 기능을 제공한다[17]. 지난 10년 동안, OMiLAB 커뮤니티는 교육, IoT, 비즈니스 등 다양한 분야에 걸쳐 57개 이상의 새로운 시각화 방법론을 제시하며, 단순히 시각적 표현에 그치지 않고 이해관계자 간의 소통을 위한 다양한 기능을 제공해 왔다. 예로는 햅틱 모델링, IoT 시뮬레이션, 로봇 제어 등이 있다. ADOxx 메타-모델링 플랫폼의 주요 구성 요소는 다음과 같다:

- 모델링 언어 (Modeling Language): 다양한 도메인에 적합한 모델링 언어의 표기법, 문법, 의미를 정의하는 라이브러리와 기능을 제공한다.
- 모델링 기법 (Modeling Technique): 모델링 언어를 활용해 모델을 구성하고, 그 모델을 통해 예측된 결과를 얻기 위한 절차를 정의하는 라이브러리와 기능을 포함한다.
- 메커니즘 및 알고리즘 (Mechanisms and Algorithms): ADOxx 플랫폼에서 제공되는 메커니즘과 알고리즘을 활용해 새로운 모델링 방법론을 개발할 수 있는 라이브러리와 기능을 지원한다.

ADOxx의 기본 기능과 메타-모델 개념을 이용하여 특정 도메인에 맞는 모델링 도구를 개발할 수 있으며, 그림 4는 ADOxx의 구성 요소를 보여준다.

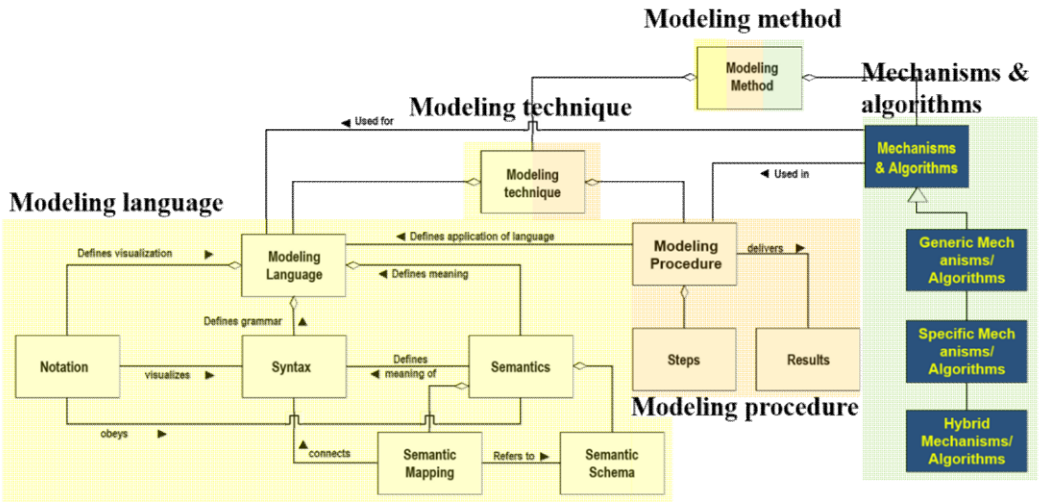

그림 4. ADOxx의 기본 구성 요소

2. 모델러

모델러는 SAVE의 주요 도구 중 하나로, 스마트 IoT 시스템의 대상 시스템을 위한 프로세스를 모델링하는데 사용된다. 모델러는 ITL (In-The-Large) 또는 ITS (In-The-Small) 모델을 생성하며, 각각 시스템 뷰 (System View)와 프로세스 뷰 (Process View)로 표현된다:

- ITL 모델 (In-The-Large Model): 시스템 내의 프로세스들이 서로 통신하는 채널을 통해 연결된 시스템 뷰 (System View)를 시각적으로 보여준다.
- ITS 모델 (In-The-Small Model): 순차적 또는 선택적 순서로 수행되는 상호작용, 즉 통신이나 이동의 집합으로 구성된 프로세스 뷰 (Process View)를 시각적으로 보여준다.

3. 시뮬레이터

시뮬레이터는 SAVE의 주요 도구 중 하나로, ITL 모델과 ITS 모델로 명세된 시스템의 모든 실행 경로를 시각적으로 표현하는 실행 모델 (Execution Model)을 생성한다. 또한, 시뮬레이터는 지리적 및 시간적 속성에 관련된 모든 정보, 즉 GTS 모델 (Geo-Temporal Space Model)을 생성한다:

- 실행 모델 (EX Model: Execution Model): 실행 모델은 도달 가능성 그래프와 유사한 시스템 전이 모델로, 대상 ITL 모델과 포함된 ITS 모델의 모든 실행 경로를 시각적으로 보여준다.
- GTS 모델 (Geo-Temporal Space Model): 시스템 내의 프로세스와 그 상호작용, 즉 통신 및 이동을 2차원 공간인 Geo-Temporal Space에 시각적으로 나타낸 모델이다.

4. 분석기 및 검증기

SAVE 도구의 핵심 요소인 분석기 및 검증기는 스마트 IoT 시스템의 안전 및 보안 요구사항을 분석하고 검증하며, 결과를 GTS 모델로 시각화하여 표현한다. 이 도구는 여러 내부 구성 요소로 이루어져 있으며, 각 구성 요소의 역할은 다음과 같다:

- T2G 논리 파서: 안전 및 보안 요구사항의 문법적 유효성을 확인하고, 분석 및 검증

과정에 필요한 데이터를 미리 생성하는 역할을 한다.

- GTS 논리 검증기: T2G 논리 파서에서 전달받은 데이터를 바탕으로, GTS 모델이 주어진 안전 및 보안 요구사항을 만족하는지를 분석하고 검증한다.
- GTS 논리 시각화기: GTS 논리 검증기를 통해 얻은 분석 및 검증 결과를 시각적으로 표현하여 사용자가 쉽게 이해할 수 있도록 돕는다.
- 커버리지 분석기/검증기: 시스템의 모든 실행 경로에 대해 안전 및 보안 요구사항과 관련된 일련의 분석 및 검증 작업을 수행하고, 이를 시각적으로 보여주는 기능을 제공한다.

이러한 구성 요소들은 대상 시스템의 안전성과 보안성을 검증하기 위한 전 과정을 체계적으로 지원하며, 이를 시각적으로 표현하여 사용자가 시스템의 상태를 쉽게 파악할 수 있게 돕는다.

Ⅴ. DeViLL 접근방법

DeViLL 접근방법은 스마트 IoT 시스템의 요구사항을 명세하고 검증하며, 이를 실제 시스템에 배치하는 과정을 크게 두 단계로 나누어 진행한다. DeViLL은 dTP-Calculus 기반의 프로세스 대수와 이를 활용한 SAVE 도구, 그리고 openHAB을 중심으로 작동한다. 그림 5는 DeViLL 방법의 개요를 보여준다.

첫 번째 단계는 스마트 IoT 시스템의 요구사항을 정형적으로 명세하고, 이를 분석 및 검증하는 과정으로, 시스템의 신뢰성과 안전성을 확인하는 데 중점을 둔다. 이 과정의 절차는 다음과 같다:

1-1) 스마트 IoT 시스템의 요구사항을 dTP-Calculus를 통해 명세하고, 이를 SAVE 도구로 시각화하여 ITL(In-The-Large) 모델과 ITS(In-The-Small) 모델을 생성한다.

1-2) ITL 및 ITS 모델을 기반으로 시스템의 모든 실행 경로를 포함하는 실행 모델을 생성한다.

1-3) 각 실행 경로의 시뮬레이션 결과를 GTS(Geo-Temporal Space) 모델로 표현하여 시스템의 동작을 검토한다.

1-4) 안전 및 보안 요구사항을 GTS Logic을 통해 정의한다.

1-5) GTS 모델에서 이러한 요구사항을 분석하고 검증한다.

1-6) 분석 및 검증 결과를 바탕으로 안전 및 보안 요구사항이 충족되는지 평가한다.

두 번째 단계는 첫 번째 단계에서 검증된 시스템을 실제 IoT 환경인 openHAB에 배치하는 과정을 다룬다. 이 단계에서는 SAVE 도구와 openHAB의 연동을 통해 시스템의 통합을 수행하며, 절차는 다음과 같다:

2-1) ITL 모델에서 명세된 특정 프로세스를 openHAB의 Item으로 생성한다.

2-2) ITS 모델에서 명세된 통신 동작을 openHAB의 Rule로 생성한다.

2-3) 실행 모델에서 파생된 특정 실행 경로를 openHAB 환경에서 시뮬레이션한다.

이러한 두 단계의 접근방법을 통해 DeViLL은 스마트 IoT 시스템의 요구사항을 명세하고 검증하며, 최종적으로 신뢰성과 안전성을 보장하는 시스템을 실세계에 적용할 수 있는 기반을 제공한다. DeViLL 접근방법은 IV 장에서 소개한 SAVE 도구의 주요 기능들을 사용하여 실현할 수 있다.

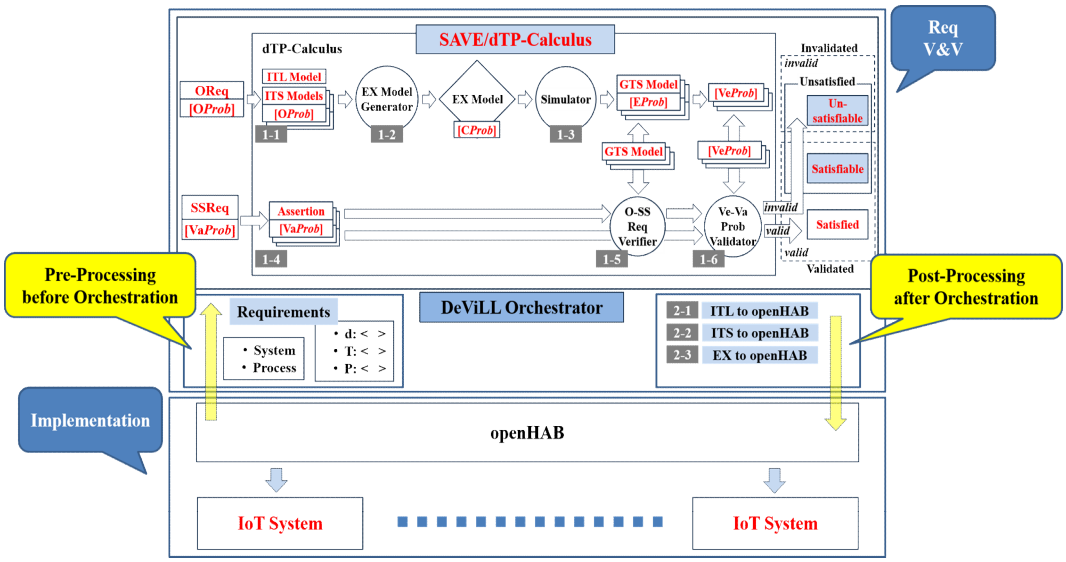

그림 5. DeViLL 접근방법

1. 1단계: 스마트 IoT 시스템의 명세, 분석 및 검증

스마트 IoT 시스템의 명세, 분석 및 검증 단계에서는 SAVE의 주요 기능들을 사용한다. 그림 6은 SAVE의 주요 기능을 보여주며, 각 기능의 내용은 다음과 같다:

- Text to Model(Specification): dTP-Calculus를 기반의 문자 명세를 시각화 명세로 변환하는 기능으로 DeViLL 접근방법의 (1-1)에 해당한다.
- Generate Execution Model: 명세한 ITL 및 ITS 모델을 기반으로 실행 모델을 생성하는 기능으로 DeViLL 접근방법의 (1-2)에 해당한다.
- Simulation Model: ITL 및 ITS 모델에 대한 시뮬레이션을 애니메이션 형태로 출력하는 기능이다.
- Generate GTS: 실행 모델 중 하나의 경로를 선택하여 GTS 모델을 생성하는 기능으로 DeViLL 접근방법의 (1-3)에 해당한다.
- GTS Analysis: GTS Logic을 기반으로 안전 요구사항을 명세하고, 시스템을 분석 및 검증하는 기능으로 DeViLL 접근방법의 (1-4)부터 (1-6)에 해당한다.

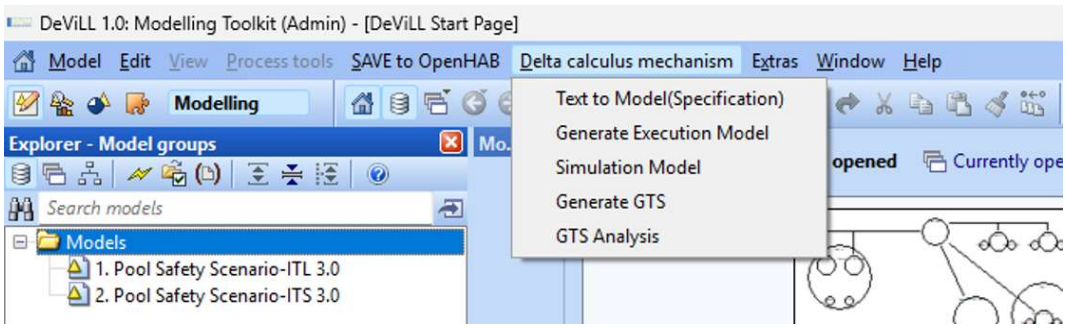

그림 6. SAVE의 주요 기능

2. 2단계: openHAB 연동

openHAB 연동 단계에서는 DeViLL의 주요 기능들을 사용한다. 그림 7은 openHAB 연동과 관련된 주요 기능을 보여주며, 각 기능의 내용은 다음과 같다:

- ITL to openHAB: 이것은 ITL 모델의 프로세스를 openHAB의 Item으로 생성하는 기능으로, ITL 프로세스에 openHAB이라는 이름의 프로세스를 반드시 명세해야 한다. 또한, openHAB이라는 이름의 프로세스와 Channel을 통해 연결된 다른 프로세스만이 openHAB의 Item으로 생성된다. 이 기능은 DeViLL 방법론의 (2-1)에 해당한다.
- ITS to openHAB: 이것은 ITS의 통신 블록(Communication Block), 즉, Send와 Receive 표기법을 openHAB의 Rule로 생성하는 기능이다. SAVE의 ITS 모델은 ITL 모델에서 정의된 프로세스의 상세한 동작을 명세한다. 따라서, ITL 모델에서 openHAB이라는 이름의 프로세스의 상세한 동작은 ITS 모델의 openHAB이라는 이름의 프로세스에서 명세할 수 있다. 이는 openHAB의 실제 동작을 명세할 수 있다는 것을 의미하며, 이곳에 정의된 통신 블록만 openHAB의 Rule로 생성될 수 있다. 이 기능은 DeViLL 방법론의 (2-2)에 해당한다.
- EX to openHAB: 이 기능은 실행 모델의 데이터를 기반으로 openHAB을 기반으로 한 스마트 IoT 시스템을 시뮬레이션하는 기능이다. SAVE에서 실행 모델은 ITL 모델과 ITS 모델을 기반으로 생성되며, 시스템의 모든 실행 가능한 경로에 대한 정보를 포함한다. 즉, 특정 openHAB을 기반으로 한 스마트 IoT 시스템의 시나리오를 명세한 ITL 및 ITS 모델로부터 생성된 실행 모델은 해당 시나리오의 스마트 IoT 시스템에 대한 모든 실행 가능한 경로를 포함하고 있다. 결과적으로 이 기능은 실행 모델의 특정 실행 경로에 저장된 정보를 기반으로 openHAB을 기반으로 한 스마트 IoT 시스템을 제어하는 기능이다. 이 기능은 DeViLL 방법론의 (2-3)에 해당한다.
- Edit openHAB Configuration file: 이 기능은 openHAB과 연동하기 위한 openHAB의 URL 및 openHAB의 계정 권한에 대한 정보를 입력하기 위한 기능이다.

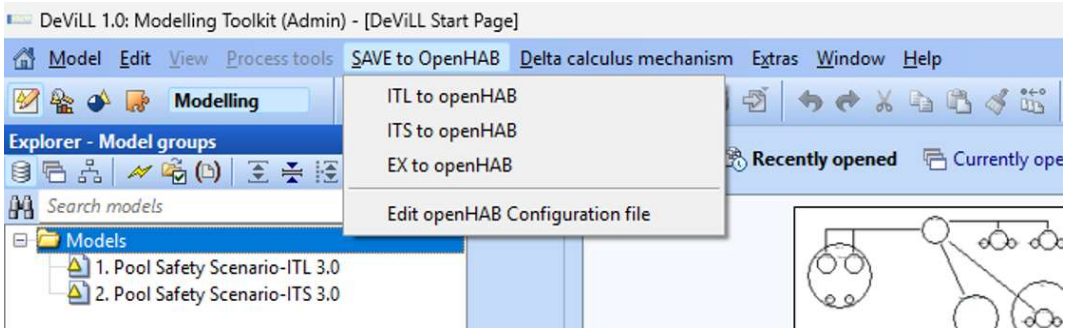

그림 7. openHAB 연동 주요 기능

Ⅵ. 사례 연구: 스마트 홈-수영장 안전 모니터링 시스템

1. 스마트 홈-수영장 안전 모니터링 시스템의 요구사항

스마트 홈-수영장 안전 모니터링(Smart Home-Swimming Pool Safety Monitoring: SwiPSaM) 시스템은 수영장 근처에 아이들을 모니터링 하여, 아이들만 수영장에 있을 경우, IoT 장치의 지원을 받아 수영장의 문을 자동으로 잠그는 시스템이다. SwiPSaM 시스템의 운영 요구사항은 다음과 같다:

1) SwiPSaM 시스템은 openHAB 플랫폼(openHAB), 집(Home), LED(GreenLED, RedLED), 수영장(Pool, PoolGate_Servo, NearPool_RFIDReader), 부모와 아이(Parent, Child)로 구성된다.
2) 집(Home)에 부모와 아이(Parent, Child)가 있는 초기 위치를 의미한다.
3) 수영장은 Pool, PoolGate_Servo, NearPool_RFIDReader로 구성된다:

- Pool은 수영장을 의미한다.
- PoolGate_Servo는 수영장 문을 제어하는 Iot 장치로, openHAB 플랫폼이 보내는 신호에 따라 자동으로 닫힐 수 있다.
- NearPool_RFIDReader는 수영장 근처에 부모 또는 아이가 있는지 감지하는 IoT 장치이며, 이는 아이만 있는지, 부모만 있는지, 아이와 부모가 함께 있는지, 아무도 없는지에 대한 정보를 openHAB 플랫폼으로 송신한다.

4) openHAB 플랫폼은 NearPool_RFIDReader로부터 온 정보를 수신하고, 수신한 정보에 따라 PoolGate_Servo와 각 LED에 신호를 보낸다:

- 아이만 있다는 신호를 받은 경우, openHAB 플랫폼은 PoolGate_Servo에 수영장의 문을 제어하는 신호와 LED에 신호를 보낸다.
- 부모만 있다는 신호를 받은 경우, openHAB 플랫폼은 PoolGate_Servo에 수영장의 문을 제어하는 신호와 LED에 신호를 보낸다.
- 아이와 부모가 함께 있다는 신호를 받은 경우, openHAB 플랫폼은 PoolGate_Servo에 수영장의 문을 제어하는 신호와 LED에 신호를 보낸다.
- 아무도 없다는 신호를 받은 경우, openHAB 플랫폼은 PoolGate_Servo에 수영장의 문을 제어하는 신호와 LED에 신호를 보낸다.

2. DeViLL 방법론을 사용한 시나리오 모델링

이번 절에서는 SwiPSaM 시스템의 운영 요구사항을 기반으로 DeViLL 방법론을 적용한다. 해당 시나리오에 대한 DeViLL 방법론은 다음과 같이 수행된다:

1) 1단계: SwiPSaM 시스템 명세, 분석 및 검증

- 명세 단계: 이 단계에서는 해당 시나리오의 요구사항을 dTP-Calculus로 명세하고, ITL 모델과 ITS 모델을 생성한다.
- 분석 단계: 이 단계에서는 ITL 모델과 ITS 모델을 기반으로 실행 모델을 생성하고, 각 실행 경로를 분석한다.
- 검증 단계: 이 단계에서는 각 실행 경로를 기반으로 요구사항을 만족하는지 검증한다.

2) 2단계: SwiPSaM 시스템을 openHAB과 연동

- openHAB의 Item 생성 단계: 이 단계에서는 ITL 모델을 기반으로 Item을 생성한다.
- openHAB의 Rule 생성 단계: 이 단계에서는 ITS 모델을 기반으로 Rule를 생성한다.
- openHAB 시뮬레이션 단계: 이 단계에서는 실행 모델을 기반으로 openHAB의 Item을 제어하여 시뮬레이션을 수행한다.

가. 명세 단계

명세 단계에서는 dTP-Calculus와 SAVE 도구를 활용하여, 요구사항을 기반으로 SwiPSaM 시스템을 명세한다.

그림 8은 SwiPSaM 시스템의 요구사항을 dTP-Calculus로 명세한 것이다. 여기에서 핵심 프로세스는 openHAB 프로세스이다.

그림 9는 openHAB 프로세스 동작의 일부를 해석하기 위한 것이다. 해당 동작을 해석하면, openHAB 프로세스는 NP_HAB 이라는 채널을 통해 NearPool_RFIDReader 프로세스로부터 수영장 근처에 아이만 있다는 신호(Child)를 수신한 경우, openHAB 프로세스는 HAB_GATE 라는 채널을 통해 PoolGate_Servo 프로세스에게 수영장 문을 닫는 신호($\overline{Servo\_On}$)를 보내고, 아이가 안전하기 때문에 openHAB 프로세스는 HAB_GL 이라는 채널을 통해 GreenLED 프로세스에게 LED를 켜라는 신호($\overline{Green\_Light\_On}$)를 보낸다. 그러나 시스템의 비결정성에 의해, openHAB 프로세스가 HAB_GATE 라는 채널을 통해 PoolGate_Servo 프로세스에게 수영장 문을 여는 신호($\overline{Servo\_Off}$)를 보낼 경우, 아이가 안전하지 않기 때문에 openHAB 프로세스는 HAB_RL 이라는 채널을 통해 RedLED 프로세스에게 LED를 켜라는 신호($\overline{Red\_Light\_On}$)를 보낼 수도 있다.

$$
\begin{aligned}
&Pool\_Safety\_System ::= openHAB \parallel Home[Parent \parallel Child] \parallel \\
&\qquad NearPool\_RFIDReader \parallel Pool[PoolGate\_Servo \parallel Gate\_Open \parallel Gate\_Close] \parallel \\
&\qquad GreenLED \parallel Green\_ON \parallel Green\_OFF \parallel RedLED \parallel Red\_ON \parallel Red\_OFF;
\end{aligned}
$$

$$
\begin{aligned}
openHAB ::= &\Big(NP\_HAB(Child)\backslash\Big(\Big(NP\_HAB(Parent)\backslash\Big(HAB\_GATE(\overline{Servo\_On}) + HAB\_GATE(\overline{Servo\_Off})\Big)\\
&\cdot HAB\_RL(\overline{Red\_Light\_Off}) \cdot HAB\_GL(\overline{Green\_Light\_On}) \cdot exit\Big) \cdot \Big(HAB\_GATE(\overline{Servo\_On}) + HAB\_GATE(\overline{Servo\_Off})\Big)\\
&\cdot HAB\_RL(\overline{Red\_Light\_Off}) \cdot HAB\_GL(\overline{Green\_Light\_On}) \cdot exit\Big)\Big)\\
&\cdot \Big(NP\_HAB(Parent)\backslash\Big(HAB\_GATE(\overline{Servo\_On}) \cdot HAB\_RL(\overline{Red\_Light\_Off}) \cdot HAB\_GL(\overline{Green\_Light\_On})\\
&+ HAB\_GATE(\overline{Servo\_Off}) \cdot HAB\_RL(\overline{Red\_Light\_On}) \cdot HAB\_GL(\overline{Green\_Light\_Off})\Big) \cdot exit\Big)\\
&\cdot \Big(HAB\_GATE(\overline{Servo\_On}) + HAB\_GATE(\overline{Servo\_Off})\Big) \cdot HAB\_RL(\overline{Red\_Light\_Off}) \cdot HAB\_GL(\overline{Green\_Light\_On}) \cdot exit;
\end{aligned}
$$

$$
\begin{aligned}
&Home ::= exit;\\
&Parent ::= (Empty\ + in\ NearPool\_RFIDReader) \cdot exit;\\
&Child ::= (Empty\ + in\ NearPool\_RFIDReader) \cdot exit;\\
&NearPool\_RFIDReader ::= \big(Child\ in \setminus (Parent\ in \setminus exit) \cdot NP\_HAB(\overline{Parent}) \cdot exit\big) \cdot NP\_HAB(\overline{Child}) \cdot\\
&\qquad (Parent\ in \setminus exit) \cdot NP\_HAB(\overline{Parent}) \cdot exit;\\
&Pool ::= exit;\\
&PoolGate\_Servo ::= (HAB\_GATE(Servo\_On) \setminus HAB\_GATE(Servo\_Off) \cdot get\ Gate\_Open \cdot put\ Gate\_Close \cdot exit) \cdot\\
&\qquad get\ Gate\_Close \cdot put\ Gate\_Open \cdot exit;\\
&Gate\_Open ::= (PoolGate\_Servo\ get \setminus PoolGate\_Servo\ put) \cdot exit;\\
&Gate\_Close ::= (PoolGate\_Servo\ get \setminus PoolGate\_Servo\ put) \cdot exit;\\
&GreenLED ::= (HAB\_GL(Green\_Light\_On) \setminus HAB\_GL(Green\_Light\_Off) \cdot get\ Green\_OFF \cdot put\ Green\_ON \cdot exit) \cdot\\
&\qquad get\ Green\_ON \cdot put\ Green\_OFF \cdot exit;\\
&Green\_ON ::= (GreenLED\ get \setminus GreenLED\ put) \cdot exit;\\
&Green\_OFF ::= (GreenLED\ get \setminus GreenLED\ put) \cdot exit;\\
&RedLED ::= (HAB\_RL(Red\_Light\_On) \setminus HAB\_RL(Red\_Light\_Off) \cdot get\ Red\_OFF \cdot put\ Red\_ON \cdot exit) \cdot\\
&\qquad get\ Red\_ON \cdot put\ Red\_OFF \cdot exit;\\
&Red\_ON ::= (RedLED\ get \setminus RedLED\ put) \cdot exit;\\
&Red\_OFF ::= (RedLED\ get \setminus RedLED\ put) \cdot exit;
\end{aligned}
$$

그림 8. SwiPSaM 시스템의 dTP-Calculus 코드

$$
\begin{aligned}
openHAB =:: & \Big(NP\_HAB(Child)\backslash\Big(\big(NP\_HAB(Parent)\backslash\big(HAB\_GATE(\overline{Servo\_On}) + HAB\_GATE(\overline{Servo\_Off})\big)\\
& \cdot HAB\_RL(\overline{Red\_Light\_Off}) \cdot HAB\_GL(\overline{Green\_Light\_On}) \cdot exit\big) \cdot \big(HAB\_GATE(\overline{Servo\_On}) + HAB\_GATE(\overline{Servo\_Off})\big)\\
& \cdot HAB\_RL(\overline{Red\_Light\_Off}) \cdot HAB\_GL(\overline{Green\_Light\_On}) \cdot exit\big)\Big)\\
& \cdot \big(NP\_HAB(Parent)\backslash\big(HAB\_GATE(\overline{Servo\_On}) \cdot HAB\_RL(\overline{Red\_Light\_Off}) \cdot HAB\_GL(\overline{Green\_Light\_On})\\
& + HAB\_GATE(\overline{Servo\_Off}) \cdot HAB\_RL(\overline{Red\_Light\_On}) \cdot HAB\_GL(\overline{Green\_Light\_Off})\big) \cdot exit\big)\\
& \cdot \big(HAB\_GATE(\overline{Servo\_On}) + HAB\_GATE(\overline{Servo\_Off})\big) \cdot HAB\_RL(\overline{Red\_Light\_Off}) \cdot HAB\_GL(\overline{Green\_Light\_On}) \cdot exit;
\end{aligned}
$$

그림 9. openHAB 프로세스의 동작(1): 수영장 근처에 아이만 있는 경우

마찬가지로 그림 10은 부모(Parent)만 수영장 근처에 있는 경우, 그림 11은 부모(Parent)와 아이(Child)가 함께 있는 경우, 그림 12는 수영장 근처에 누구도 없는 경우를 dTP-Calculus로 표현한 것이다.

이를 SAVE 도구를 통해 ITL 모델과 ITS 모델로 생성하면 그림 13과 그림 14와 같다.

$$
\begin{aligned}
openHAB =:: & \Big(NP\_HAB(Child)\backslash\Big(\big(NP\_HAB(Parent)\backslash\big(HAB\_GATE(\overline{Servo\_On}) + HAB\_GATE(\overline{Servo\_Off})\big)\\
& \cdot HAB\_RL(\overline{Red\_Light\_Off}) \cdot HAB\_GL(\overline{Green\_Light\_On}) \cdot exit\big) \cdot \big(HAB\_GATE(\overline{Servo\_On}) + HAB\_GATE(\overline{Servo\_Off})\big)\\
& \cdot HAB\_RL(\overline{Red\_Light\_Off}) \cdot HAB\_GL(\overline{Green\_Light\_On}) \cdot exit\big)\Big)\\
& \cdot \big(NP\_HAB(Parent)\backslash\big(HAB\_GATE(\overline{Servo\_On}) \cdot HAB\_RL(\overline{Red\_Light\_Off}) \cdot HAB\_GL(\overline{Green\_Light\_On})\\
& + HAB\_GATE(\overline{Servo\_Off}) \cdot HAB\_RL(\overline{Red\_Light\_On}) \cdot HAB\_GL(\overline{Green\_Light\_Off})\big) \cdot exit\big)\\
& \cdot \big(HAB\_GATE(\overline{Servo\_On}) + HAB\_GATE(\overline{Servo\_Off})\big) \cdot HAB\_RL(\overline{Red\_Light\_Off}) \cdot HAB\_GL(\overline{Green\_Light\_On}) \cdot exit;
\end{aligned}
$$

그림 10. openHAB 프로세스의 동작(2): 수영장 근처에 부모만 있는 경우

$$
\begin{aligned}
openHAB =:: & \Big(NP\_HAB(Child)\backslash\Big(\big(NP\_HAB(Parent)\backslash\big(HAB\_GATE(\overline{Servo\_On}) + HAB\_GATE(\overline{Servo\_Off})\big)\\
& \cdot HAB\_RL(\overline{Red\_Light\_Off}) \cdot HAB\_GL(\overline{Green\_Light\_On}) \cdot exit\big) \cdot \big(HAB\_GATE(\overline{Servo\_On}) + HAB\_GATE(\overline{Servo\_Off})\big)\\
& \cdot HAB\_RL(\overline{Red\_Light\_Off}) \cdot HAB\_GL(\overline{Green\_Light\_On}) \cdot exit\big)\Big)\\
& \cdot \big(NP\_HAB(Parent)\backslash\big(HAB\_GATE(\overline{Servo\_On}) \cdot HAB\_RL(\overline{Red\_Light\_Off}) \cdot HAB\_GL(\overline{Green\_Light\_On})\\
& + HAB\_GATE(\overline{Servo\_Off}) \cdot HAB\_RL(\overline{Red\_Light\_On}) \cdot HAB\_GL(\overline{Green\_Light\_Off})\big) \cdot exit\big)\\
& \cdot \big(HAB\_GATE(\overline{Servo\_On}) + HAB\_GATE(\overline{Servo\_Off})\big) \cdot HAB\_RL(\overline{Red\_Light\_Off}) \cdot HAB\_GL(\overline{Green\_Light\_On}) \cdot exit;
\end{aligned}
$$

그림 11. openHAB 프로세스의 동작(3): 수영장 근처에 부모와 아이가 함께 있는 경우

$$
\begin{aligned}
openHAB =:: & \Big(NP\_HAB(Child)\backslash\Big(\big(NP\_HAB(Parent)\backslash\big(HAB\_GATE(\overline{Servo\_On}) + HAB\_GATE(\overline{Servo\_Off})\big)\\
& \cdot HAB\_RL(\overline{Red\_Light\_Off}) \cdot HAB\_GL(\overline{Green\_Light\_On}) \cdot exit\big) \cdot \big(HAB\_GATE(\overline{Servo\_On}) + HAB\_GATE(\overline{Servo\_Off})\big)\\
& \cdot HAB\_RL(\overline{Red\_Light\_Off}) \cdot HAB\_GL(\overline{Green\_Light\_On}) \cdot exit\big)\Big)\\
& \cdot \big(NP\_HAB(Parent)\backslash\big(HAB\_GATE(\overline{Servo\_On}) \cdot HAB\_RL(\overline{Red\_Light\_Off}) \cdot HAB\_GL(\overline{Green\_Light\_On})\\
& + HAB\_GATE(\overline{Servo\_Off}) \cdot HAB\_RL(\overline{Red\_Light\_On}) \cdot HAB\_GL(\overline{Green\_Light\_Off})\big) \cdot exit\big)\\
& \cdot \big(HAB\_GATE(\overline{Servo\_On}) + HAB\_GATE(\overline{Servo\_Off})\big) \cdot HAB\_RL(\overline{Red\_Light\_Off}) \cdot HAB\_GL(\overline{Green\_Light\_On}) \cdot exit;
\end{aligned}
$$

그림 12. openHAB 프로세스의 동작(4): 수영장 근처에 누구도 없는 경우

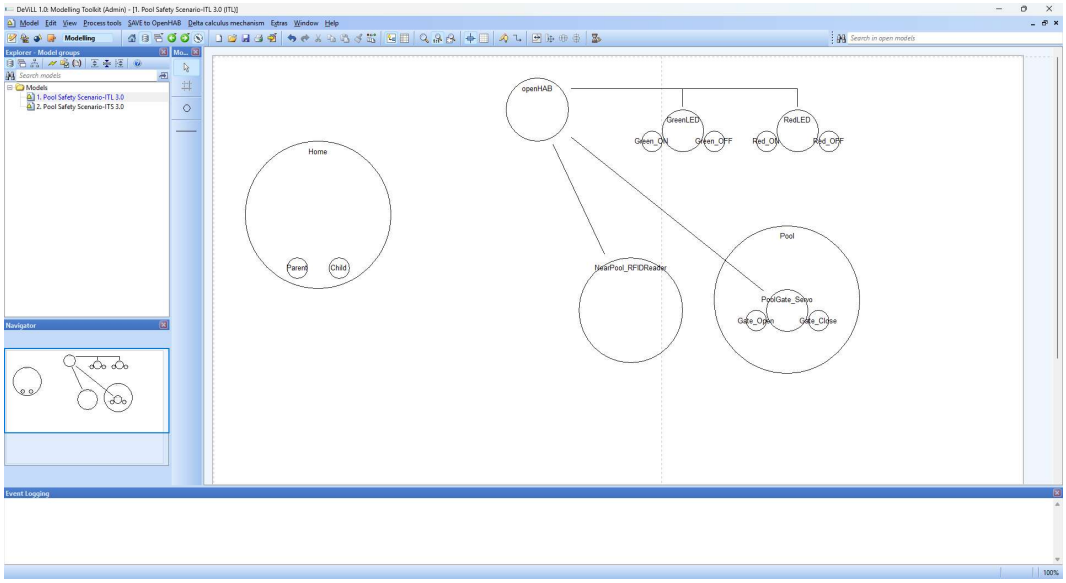

그림 13. SwiPSaM 시스템의 ITL 모델

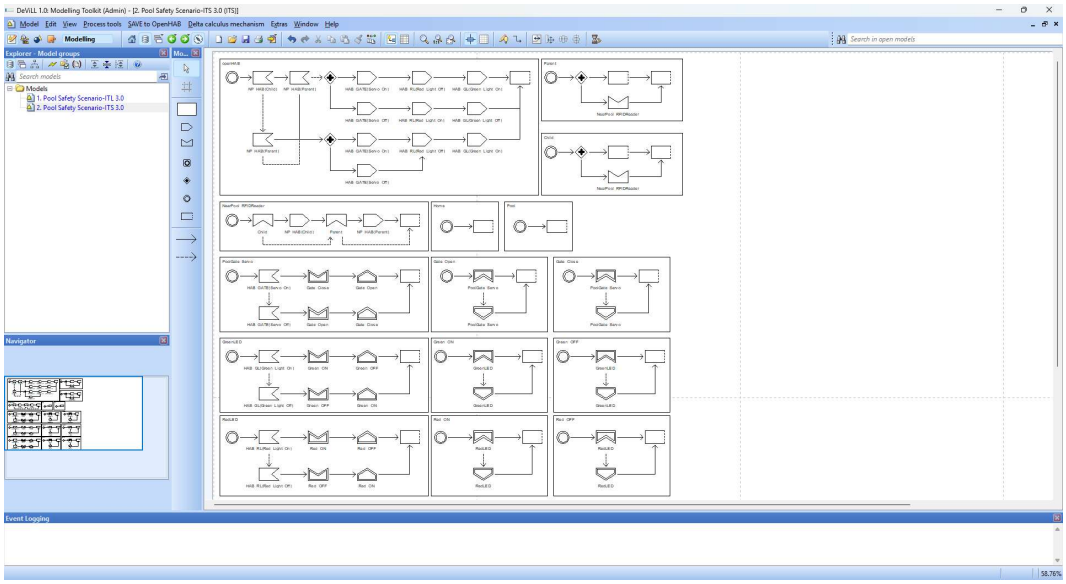

그림 14. SwiPSaM 시스템의 ITS 모델

나. 분석 단계

분석 단계에서는 명세 단계에서 생성한 ITL 모델과 ITS 모델을 기반으로 실행 모델을 생성한다. 실행 모델은 스마트 홈-수영장 안전 모니터링 시스템에서 발생할 수 있는 모든 실행 경로를 확인하여, 스마트 홈-수영장 안전 모니터링 시스템의 실행 상태를 분석할 수 있다.

그림 15는 그림 13 ITL 모델과 그림 14 ITS 모델을 기반으로 생성한 실행 모델을 보여준다. SwiPSaM 시스템에 대한 각 실행 경로를 분석한 결과는 다음과 같다:

- Path 1: 이것은 수영장 근처에 누구도 없는 경우이다. 이때 수영장 문은 열려 있지만 수영장 근처에 누구도 없으므로 안전한 상태이다.
- Path 2: 이것은 수영장 근처에 누구도 없는 경우이다. 이때 수영장 문은 닫혀 있으므로 안전한 상태이다.

- Path 3: 이것은 수영장 근처에 아이만 있는 경우이다. 이때 수영장 문이 열려 있고, 수영장 근처에 아이만 있으므로 안전하지 않은 상태이다.
- Path 4: 이것은 수영장 근처에 아이만 있는 경우이다. 이때 수영장 문은 닫혀 있으므로 안전한 상태이다.
- Path 5: 이것은 수영장 근처에 부모만 있는 경우이다. 이때 수영장 문은 열려 있지만 수영장 근처에 부모만 있으므로 안전한 상태이다.
- Path 6: 이것은 수영장 근처에 부모만 있는 경우이다. 이때 수영장 문은 닫혀 있으므로 안전한 상태이다.
- Path 7: 이것은 수영장 근처에 부모와 아이가 함께 있는 경우이다. 이때 수영장 문은 열려 있지만 아이와 부모가 함께 있기 때문에 안전한 상태이다.
- Path 8: 이것은 수영장 근처에 부모와 아이가 함께 있는 경우이다. 이때 수영장 문은 닫혀 있으므로 안전한 상태이다.

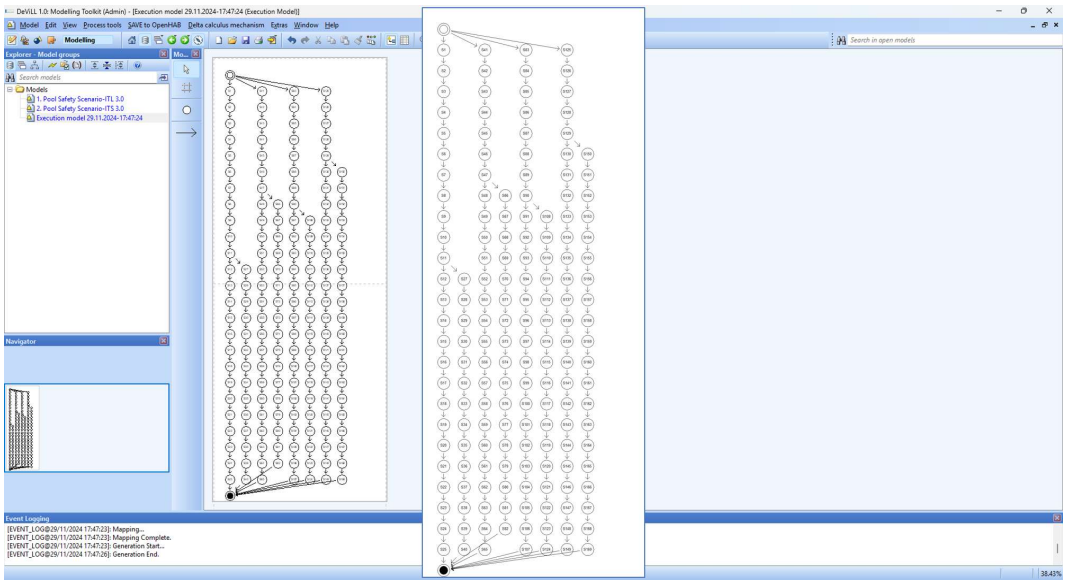

그림 15. SwiPSaM 시스템의 실행 모델

다. 검증 단계

검증 단계에서는 시스템의 안전 및 보안 요구사항을 기반으로 시스템의 안전성과 신뢰성을 검증하는 단계이다. SwiPSaM 시스템의 안전 및 보안 요구사항은 다음과 같다:

- 요구사항1(Req1): 수영장 근처에 아이만 있는 경우, 반드시 수영장 문은 닫혀 있어야 한다.
- 요구사항2(Req2): LED는 반드시 하나만 점등되어야 한다.
- 요구사항3(Req3): LED는 수영장 근처의 상황과 수영장 문의 개폐 여부에 따라 1단위 시간 안에 점등되어야 한다.

표 4. SwiPSaM 시스템의 요구사항 분석

	Path 1	Path 2	Path 3	Path 4	Path 5	Path 6	Path 7	Path 8
Req 1	O	O	O	X	O	O	O	O
Req 2	O	O	O	O	O	O	O	O
Req 3	O	O	O	O	O	O	O	O

표 4는 각 실행 경로를 기반으로 요구사항을 만족하는지 정리한 표이다.

표 4를 살펴보면, Path 4의 경우 Req1을 만족하지 못하는 것을 확인할 수 있다. Path 4의 경우, 수영장 근처에 아이만 있지만, 수영장 문이 열려 있어 Req1를 만족하지 못한다. 이런 경우, 시스템의 명세를 수정하거나, 시스템에서 Path 4의 경우가 발생한 경우, 적절한 조치를 할 수 있도록 해야 한다. 본 예제에서는 Path 4의 경우가 발생할 경우, RedLED를 점등하고 부모에게 빠르게 알려, 부모가 즉각적으로 조치를 할 수 있도록 설계하였다.

그림 16은 SAVE의 GTS 모델을 통해 SwiPSaM 시스템의 안전 및 보안 요구사항을 검증한 하나의 결과를 보여준다.

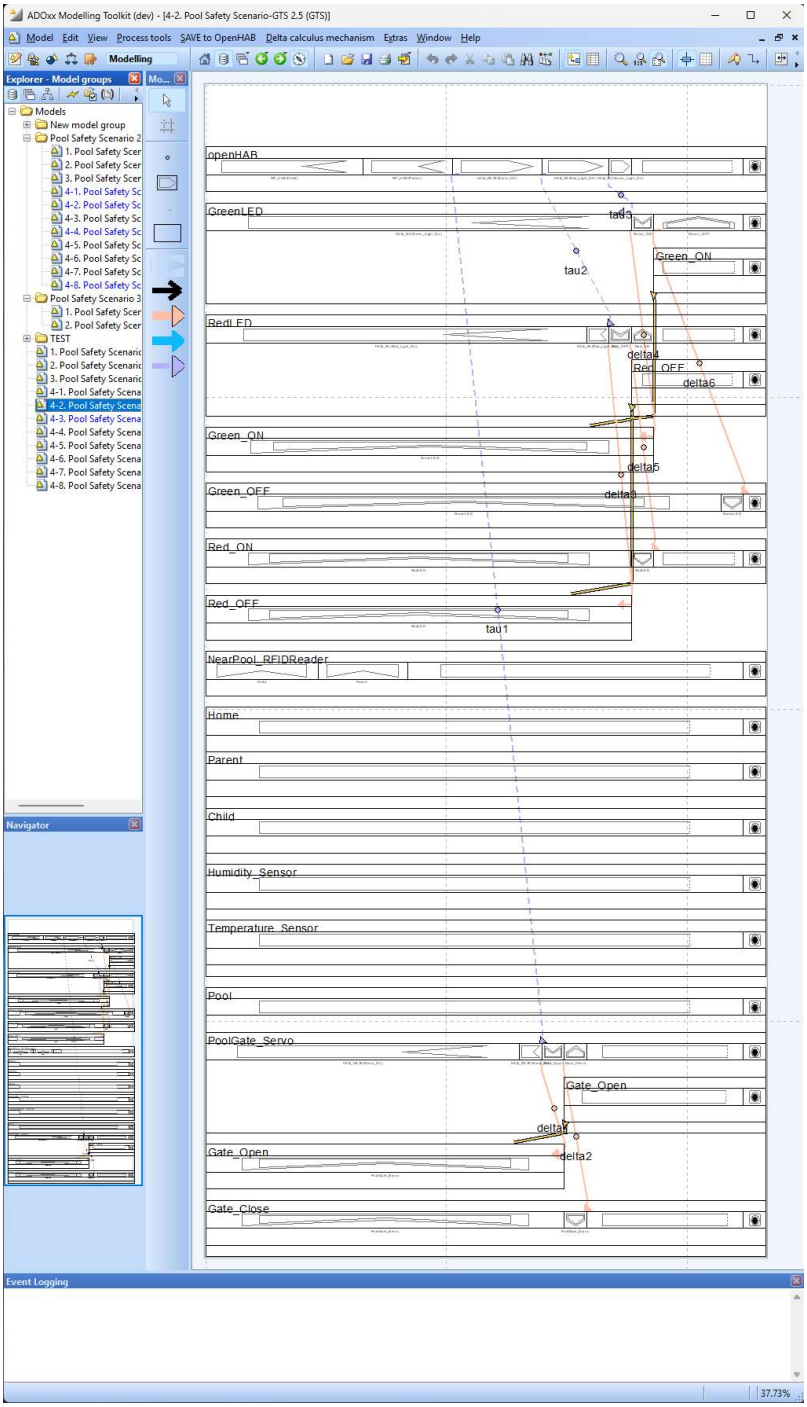

그림 16. SwiPSaM 시스템의 GTS 모델

라. openHAB의 Item 생성 단계

openHAB Item 생성 단계는 앞서 수행한 SwiPSaM 시스템의 명세, 분석 및 검증 과정을 통해 시스템의 신뢰성 및 안전성을 확인한 후, 실제 IoT 통합 플랫폼, 즉, openHAB에 배치하는 단계이다.

openHAB Item 생성 단계는 SAVE의 ITL 모델을 기반으로 openHAB의 Item을 생성한다. 이 단계에서 필수사항은 반드시 ITL 모델에 openHAB 이라는 이름을 갖는 프로세스가 존재해야 하며, 이 프로세스와 Channel로 연결된 프로세스만이 Item으로 생성된다.

openHAB의 Item을 생성하기 위한 정보는 ADOxx의 Notebook 기능을 통해 정의할 수 있으며, openHAB 프로세스와 연결된 프로세스 만이 해당 기능을 활성화할 수 있다. 그림 17은 GreenLED의 Notebook을 보여주며, GreenLED Item을 생성하기 위한 정보가 입력되어 있다.

그림 18은 스마트 홈-수영장 안전 모니터링 시스템의 프로세스를 DeViLL의 ITL to openHAB 기능을 통해 openHAB Item들을 생성하는 것을 보여준다.

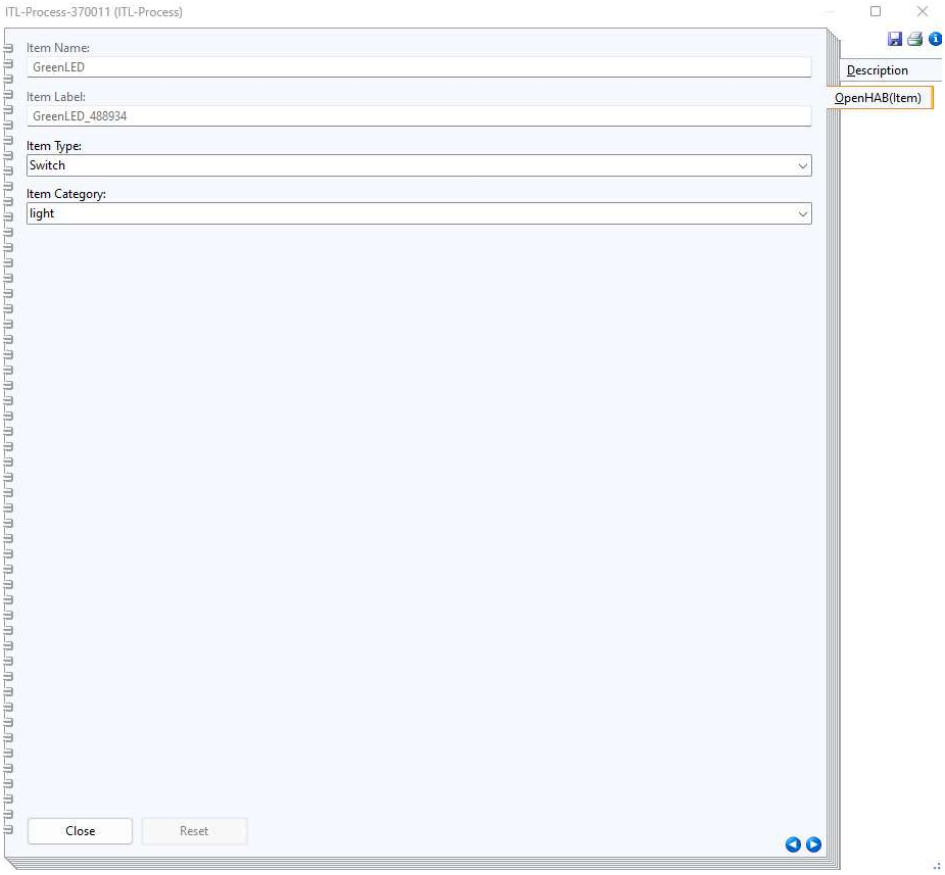

그림 17. ADOxx의 Notebook: GreenLED 프로세스

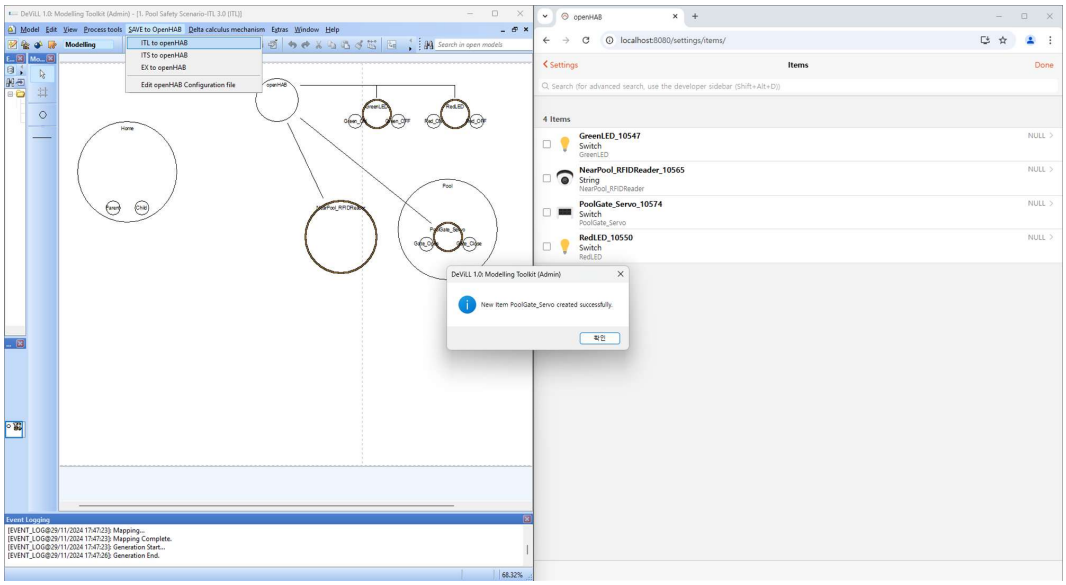

그림 18. ITL to openHAB

마. openHAB의 Rule 생성 단계

openHAB Rule 생성 단계는 openHAB에 배치하는 단계로 SAVE의 ITS 모델을 기반으로 openHAB의 Rule을 생성한다. 이 단계에서 필수 사항은 반드시 ITS 모델에 openHAB 이라는 이름을 갖는 프로세스 라인이 존재해야 한다. 이 프로세스 라인 안에 명세된 통신 채널(Send, Receive)만이 openHAB의 Rule을 생성한다.

마찬가지로 openHAB의 Rule을 생성하기 위한 정보 또한 ADOxx의 Notebook 기능을 통해 정의 할 수 있으며, 그림 19는 openHAB 프로세스 라인 안에 있는 Send 블록의 Notebook을 보여주며, openHAB의 Rule을 생성하기 위한 정보가 입력되어 있다.

그림 20은 Notebook에 정의된 정보를 기반으로 DeViLL의 ITS to openHAB 기능을 통해 openHAB의 Rule을 생성하는 것을 보여준다.

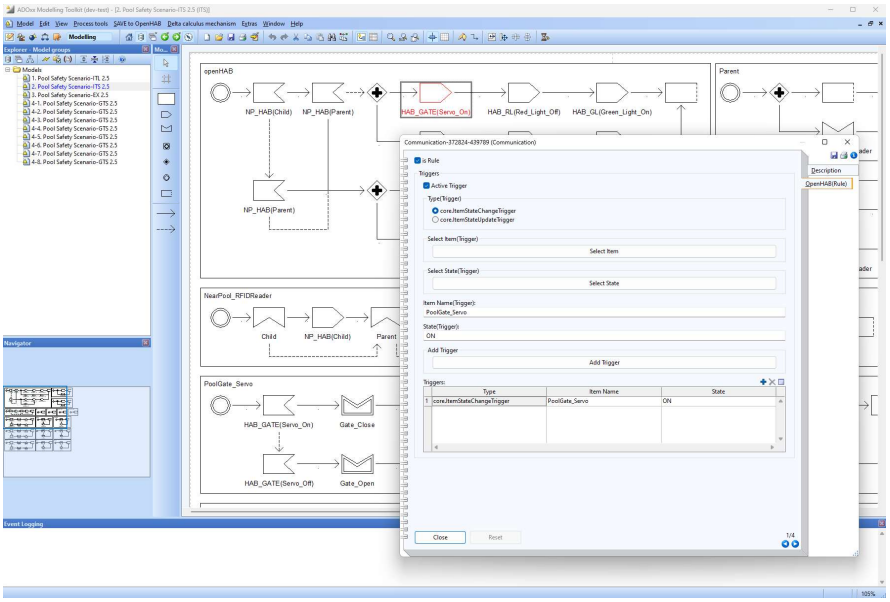

그림 19. ADOxx의 Notebook: 통신 블록

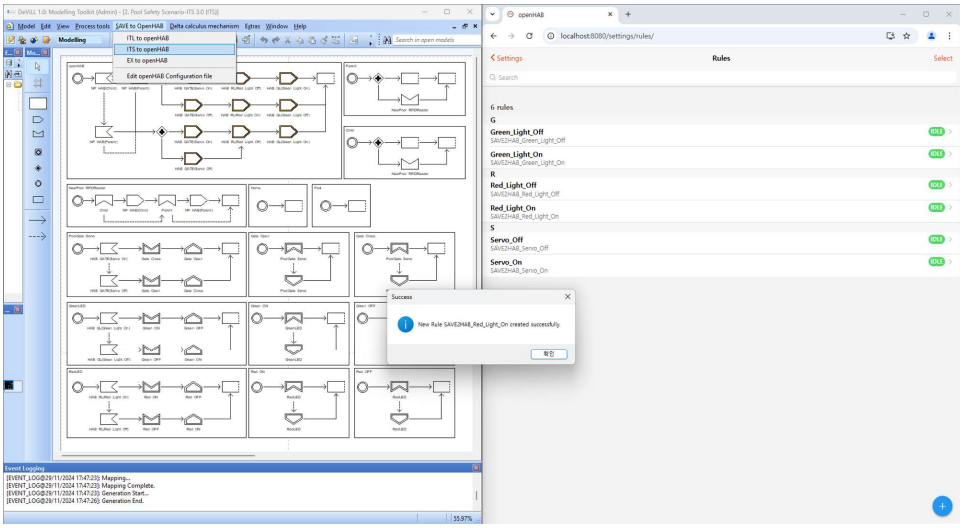

그림 20. ITS to openHAB

바. SwiPSaM 시스템 시뮬레이션 단계

openHAB 시뮬레이션 단계는 SAVE에서 생성된 실행 모델의 모든 경로가 실제 IoT 통합 시스템 플랫폼인 openHAB에서 동일한 동작으로 구동하는지 시뮬레이션하는 단계이다. 이를 통해 앞서 정의한 Item과 Rule이 올바르게 정의되었는지 확인할 수 있다.

그림 21은 실행 모델과 DeViLL의 EX to openHAB 기능을 통해 openHAB의 Item을 제어하여 시뮬레이션하는 것을 보여준다.

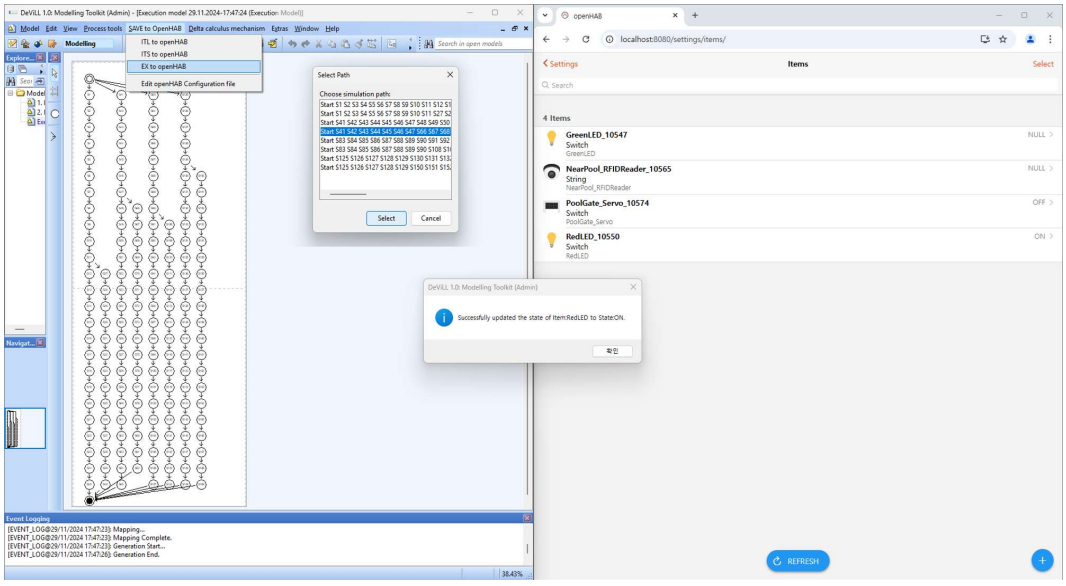

그림 21. EX to openHAB

Ⅶ. 결 론

본 연구는 스마트 IoT 시스템에서 디지털 트윈 기술의 잠재력을 최대한 활용하기 위해 기존 IoT 통합 플랫폼의 한계를 극복하고, 신뢰성과 안전성을 보장할 수 있는 DeViLL 방법론을 제안하였다. DeViLL은 dTP-Calculus 기반의 정형 기법과 이를 지원하는 SAVE 도구, 그리고 IoT 통합 플랫폼인 openHAB을 통합하여 IoT 시스템의 설계, 분석, 검증, 배치까지의 전체 공정 과정을 체계적으로 관리하였다.

사례 연구로 수행된 SwiPSaM 시스템은 아이들만 수영장 근처에 있을 때, IoT 장치를 통해 자동으로 수영장 문을 잠그는 기능을 포함했다. 이 시스템을 DeViLL로 설계 및 구현한 결과, 스마트 IoT 시스템의 요구사항을 정확히 명세하고 분석하며, 실행할 수 있는 모든 시뮬레이션 경로를 생성하여 시스템의 안전성과 신뢰성을 검증할 수 있었다. 이를 통해 DeViLL의 실용성과 유효성을 실증하였으며, 복잡한 스마트 IoT 시스템에서도 신뢰할 수 있는 성능을 제공함을 확인하였다.

향후 연구 방향은 다음과 같다:

- 스마트 시티, 산업 자동화, 헬스 케어 등과 같은, 다양한 응용 분야에서 DeViLL의 보편성을 평가하기 위한, 스마트 IoT의 실용적 시스템 요구사항을 충족할 수 있는지를 검증해야 한다.
- SAVE 도구의 기능을 고도화하여 자동화 수준을 향상하고, 특히 요구사항 분석 및 검증 과정에서 대규모 데이터 처리와 실시간 시뮬레이션 기능을 강화해야 한다.
- openHAB과 같은 IoT 플랫폼과의 통합성을 개선하여 IoT 시스템 배치 시 발생할 수 있는 기술적, 운영적 장애를 최소화할 필요가 있다.
- 시스템의 실행 과정에서 실시간 데이터를 활용하여 동적 검증 및 적응형 제어를 가능하게 하는 방법을 탐색함으로써, 변화하는 IoT 환경에서도 안정적인 시스템 동작을 보장할 수 있어야 한다.

본 연구에서 제안한 DeViLL 방법론은 스마트 IoT 시스템의 설계 및 구현 과정에서 요구되는 방대하고 복잡한 상호작용과 비결정적 요소를 효과적으로 관리함으로써, IoT 시스템의 안전성과 신뢰성을 향상하는 데 실질적인 기여를 할 것으로 예상된다.

REFERENCES

[1] K. Josifovska, E. Yigitbas, and G. Engels, "Reference framework for digital twins within cyber-physical systems," *2019 IEEE/ACM 5th International Workshop on Software Engineering for Smart Cyber-Physical Systems (SEsCPS)*, pp. 25-31, Montreal, QC, Canada, May. 2019.

[2] T. Lechler, J. Fuchs, M. Sjarov, M. Brossog, A. Selmaier, F. Faltus, T. Donhauser, and J. Franke, "Introduction of a comprehensive Structure Model for the Digital Twin in Manufacturing," *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1773-1780, Vienna, Austria, Sept. 2020.

[3] MW. Grieves, "Product lifecycle management: the new paradigm for enterprises," *International Journal of Product Development*. vol. 2, no. 1-2, pp. 71-84, Apr. 2005.

[4] A. Bumann. *Navigating the black box: generativity and incongruences in digital innovation*, Universidade Tecnica de Lisboa(Portugal), 2022.

[5] A. Fertig, M. Weigold, and Y. Chen, "Machine Learning based quality prediction for milling processes using internal machine tool data," *Advances in Industrial and Manufacturing Engineering*, vol. 4, no. 100074, pp. 1-12, Mar. 2022.

[6] S. Vandermerwe, J. Rada, "Servitization of business: adding value by adding services," *European management journal*, vol. 6, no. 4, pp. 314-324, 1988.

[7] Home Assistant Developer Docs(2025). https://developers.home-assistant.io (accessed Apr., 24, 2025)

[8] About Domoticz(2021).

https://wiki.domoticz.com/About_Domoticz (accessed Apr., 24, 2025)
[9] F. Heimgaertner, S. Hettich, O. Kohlbacher, and M. Menth, "Scaling home automation to public buildings: A distributed multiuser setup for OpenHAB 2," *2017 Global Internet of Things Summit (GIoTS)*, pp. 1-6, Geneva, Switzerland, Jun, 2017.
[10] JA. Bergstra, A. Ponse, and SA. Smolka (Eds). *Handbook of process algebra*. Elsevier. Netherlands, 2001.
[11] J.S. Song, S.H. Lee, D. Karagiannis, and M.K. Lee, "Process Algebraic Approach for Probabilistic Verification of Safety and Security Requirements of Smart IoT (Internet of Things) Systems in Digital Twin," *Sensors*, vol. 24(3), no. 767, pp. 1-35, Jan. 2024.
[12] J.S. Song, D. Karagiannis, and M.K. Lee, "A Process Algebraic Approach to Predict and Control Uncertainty in Smart IoT Systems for Smart Cities Based on Permissible Probabilistic Equivalence," *Sensors*, vol. 24(12), no. 3881, pp. 1-50, Jun. 2024.
[13] H. Hansson, and B. Jonsson, "A calculus for communicating systems with time and probabilities," *Proceedings 11th Real-Time Systems Symposium*, pp. 278-287, Lake Buena Vista, FL, USA, Dec, 1990.
[14] I. Lee, A. Philippou, and O. Sokolsky, "Resources in process algebra," *The Journal of Logic and Algebraic Programming*, vol. 72, no. 1, pp. 98-122, May-Jun. 2007.
[15] C. Feng, and J. Hillston, "PALOMA: A process algebra for located markovian agents," *International Conference on Quantitative Evaluation of Systems*, pp. 265-280, Florence, Italy, Sept, 2014.
[16] Y.B. Choe, and M.K. Lee, "Algebraic method to model secure IoT," *Domain Specific Conceptual Modeling: Concepts, Methods and Tools*, Springer, Switzerland, pp. 335-355. 2016.
[17] D. Karagiannis, and H. Kuhn, "Metamodelling platforms," *Proceedings of the 3rd International Conference EC-Web 2002*, pp. 182-195, Aix-en-Provence, France, Sept, 2002.

저 자 소 개

송준섭(정회원)

2016년 전북대학교 전기전자컴퓨터공학부(컴퓨터공학) 학사 졸업.
2018년 전북대학교 전자정보공학부(컴퓨터공학) 석사 졸업.
2025년 전북대학교 전자정보공학부(컴퓨터공학) 박사 졸업.
<주관심분야 : 소프트웨어공학, 정형기법, 프로세스 대수, 스마트 IoT 시스템, 지식 공학>

이문근(정회원)

1989년 The Pennsylvania State University, Computer Science 학과, 학사 졸업.
1992년 The University of Pennsylvania, Computer and Information Science 학과 석사 졸업.
1995년 The University of Pennsylvania, Computer and Information Science 학과 박사 졸업.
1996년 - 현재 전북대학교 컴퓨터인공지능학부 교수.
<주관심분야 : 정형기법, 소프트웨어 재/역공학, 스마트 IoT 시스템, 운영체제, 컴파일러>