Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

Blockchain alapú digitális iker modellvezérelt megvalósítása

TDK dolgozat

Készítette:

Bálint Sándor
Gyönki Bendegúz

Konzulens:

dr. Kocsis Imre
dr. Vörös András

2020

Tartalomjegyzék

Kivonat

A kiberfizikai rendszerekben (Cyber-physical systems, CPS) a fizikai világot számítógépes rendszerek figyelik, vezérlik, továbbá az intelligenciát biztosítják. Annak érdekében, hogy mindezen szolgáltatásokat biztosítani tudjuk, a fizikai entitásoknak egy digitális képét kell tárolnunk: ezt szokás digitális ikerpárnak nevezni.

A digitális iker karbantartásához szükséges a megfelelő infrastruktúra, amely lehetővé teszi a szükséges adatok, információk megbízható összegyűjtését. Erre nyújt megoldást a Data Distribution Service (DDS), amely egy nagy teljesítményű, magas rendelkezésre állású kommunikációs middleware. Az adatgyűjtés során nyert reprezentáción sokféle műveletet, optimalizációt vagy predikciót végezhetünk, továbbá ezeket a digitális ikerpárokat hordozni tudjuk szolgáltatók között is.

A digitális ikerpárokat azonban nem elég tárolni, de lehetőleg biztosítani kell a tárolt adatok hitelességét, a műveletek és az események letagadhatatlanságát is. Ez különösen fontos kritikus alkalmazásokban, ahol a rendszer hibája vagy résztvevők mulasztása komoly anyagi kárt okozhat, vagy akár emberéleteket is követelhet.

Ezen problémákra nyújtanak megoldást a napjainkban népszerű blokklánc (blockchain) alapú technológiák. Az adattárolás mellett az okosszerződések segítségével tranzakciókat is megvalósíthatunk, és ezen műveletekre is teljesül a blokkláncok legfontosabb tulajdonsága: a műveletek, tevékenységek letagadhatatlanok a résztvevők számára.

Dolgozatunk célja, hogy kombináljuk a blokklánc és a digitális ikerpár technológiák előnyeit, és magas szintű nyelvekkel támogassuk a blokklánc alapú digitális ikerpárok fejlesztését. Munkánk során a digitális iker interfészeit automatikusan, magas szintű adatmodellekből származtatjuk, amely segítségével könnyen integrálható a digitális iker az adatgyűjtő DDS hálózatra. Ugyanezen leírásokat használjuk fel a blokklánc interfészeinek definiálására is. Az okosszerződések tervezésére állapotgép alapú viselkedés leíró nyelvet vezetünk be, amelyből az okosszerződés implementációja is automatikusan származtatható. Az általunk bemutatott megközelítés célja a megbízható, blokklánc alapú digitális iker rendszerek fejlesztésének támogatása. Munkánk alkalmazhatóságát egy kritikus CPS esettanulmány segítségével szemléltetjük.

Abstract

In the case of cyber-physical systems, the physical world is monitored and controlled by computer systems and these systems also provide intelligence. In order to provide these services, the digital image of physical entities has to be stored. These are commonly called digital twins.

The maintenance of a digital twin requires a suitable infrastructure that enables reliable data collection. A solution to this problem is provided by Data Distribution Service (DDS), which is a high performance, high availability communication middleware. Different kinds of operations, optimizations and predictions can be performed on the representation obtained by data collection. Also, these digital twins can be transferred between service providers.

In addition to storing digital twins, the authenticity of the stored data and non-repudiation of the operations and events has to be provided. This is especially important in critical applications, in which the error of the system or a user may cause significant damage to property or even cause human death.

The nowadays popular blockchain technologies provide a solution for these problems. In addition to data storage, smart contracts can also be used to implement transactions. The most important property of blockchains is also true for these operations: the occurrences of operations, activities cannot be denied later by the participants.

The aim of our thesis is to combine the benefits of blockchain and digital twin technologies and to support the development of blockchain-based digital twins with high level languages. In our work, the interfaces of the digital twin are derived automatically from high level data models. This means that the digital twin can be easily integrated to the DDS network used for data collection. The same descriptions are used to define blockchain interfaces. A state machine based behavioral language for designing smart contracts is also introduced, from which the implementation of the smart contract can be automatically derived. The aim of the presented approach is to support the development of reliable, blockchain-based digital twin systems. The applicability of our work is illustrated through a critical CPS case study.

1. fejezet

Bevezetés

A kiberfizikai rendszerekben (Cyber-physical systems, CPS) a fizikai világot számítógépes rendszerek figyelik és vezérlik. A döntéshozáshoz és a beavatkozáshoz szükséges a fizikai világra vonatkozó adatok gyűjtése és a folyamatosan gyűjtött adatok alapján a fizikai entitás digitális képének tárolása és karbantartása. A fizikai entitás digitális képét digitális ikerpárnak szokás nevezni.

A digitális iker karbantartásához megfelelő infrastruktúra szükséges, amely biztosítja a szükséges adatok megbízható összegyűjtését. A Data Distribution Service (DDS) egy nagy teljesítményű, magas rendelkezésre állású kommunikációs köztesréteg, ami megoldást nyújt erre a problémára.

A digitális ikerpárok felhasználhatóak kritikus alkalmazásokban, ahol a rendszer hibája vagy résztvevők mulasztása komoly anyagi kárt okozhat, vagy akár emberéleteket is követelhet. A digitális ikerpárok tárolásán kívül ezért fontos követelmény a tárolt adatok hitelességének, a műveletek és az események letagadhatatlanságának biztosítása.

Ezen problémára nyújtanak megoldást a napjainkban népszerű blokklánc alapú technológiák. Az adattárolás mellett az okosszerződések segítségével tranzakciókat is megvalósíthatunk, és ezen műveletekre is teljesül a blokkláncok legfontosabb tulajdonsága: a műveletek, tevékenységek letagadhatatlanok a résztvevők számára.

Dolgozatunk célja, hogy a blokklánc és a digitális ikerpár technológiákat együttesen alkalmazva kombináljuk azok előnyeit. Célunk továbbá, hogy magas szintű nyelvekkel támogassuk a blokklánc alapú digitális ikerpárok fejlesztését.

Munkánk során a digitális iker interfészeit automatikusan, magas szintű adatmodellekből származtatjuk, ezáltal a digitális iker könnyen integrálható az adatgyűjtő DDS hálózatra. A blokklánc interfészeket is ezekből a modellekből állítjuk elő. Az okosszerződések tervezésére állapottérkép alapú, viselkedést leíró nyelvet vezetünk be, amelyből az okosszerződés implementációja is automatikusan származtatható.

Az általunk bemutatott megközelítés célja a megbízható, blokklánc alapú digitális iker rendszerek fejlesztésének támogatása. Munkánk alkalmazhatóságát egy konkrét, kritikus kiberfizikai rendszer esettanulmányával szemléltetjük.

A dolgozat felépítése

A 2. fejezetben bemutatjuk a munkánk során használt eszközöket és technológiákat. A 3. fejezetben ismertetjük munkánk motivációját, illetve bemutatjuk az általunk készített eszközöket. A 4. fejezetben egy konkrét CPS esettanulmányon keresztül mutatjuk be az általunk készített eszközök használatát.

2. fejezet

Háttérismeretek

Ebben a fejezetben bemutatjuk munkánk elméleti háttérét, illetve ismertetjük az általunk készített eszközök megvalósításához használt technológiákat, eszközöket.

2.1. Kiberfizikai rendszerek

A kiberfizikai rendszerek [1] a fizikai világ folyamatait figyelik meg és vezérlik. Ezen rendszerekre jellemző, hogy a fizikai folyamatok és a döntéshozás kölcsönösen befolyásolják egymást. A kiberfizikai rendszerek gyakran kritikus feladatokat látnak el, elég csak az önvezető autókra gondolni. Ez azt jelenti, hogy a hibás működés nem megengedhető: ha nem teljesíti az elvárt követelményeket a CPS, akkor az komoly anyagi kárt okozhat, vagy akár emberéleteket is követelhet. Emiatt fontos feladat a kiberfizikai rendszerek szolgáltatásbiztos működésének biztosítása.

A kiberfizikai rendszerek a beágyazott világra épülnek, azaz szenzoroktól érkező adatokat dolgoznak fel. A szenzor réteg fölött a terepi eszközök között nagyobb számítási kapacitású komponenseket is találhatunk már. A hierarchia tetején a felhő szolgáltatások vannak, amelyek a nagy távolság miatt nem valósidejű beavatkozást végeznek, viszont a nagy számítási kapacitásuknak hála képesek intelligenciát hozni a rendszer működésébe.

Napjaink kiberfizikai rendszerei egyre több feladatot látnak el, amely kiterjed a vezérlés, optimális működés vagy a külső hatóságok integrálására a rendszerbe. Ezen új követelmények jelentik munkánk motivációját.

2.2. Data Distribution Service

A Data Distribution Service (DDS) egy kommunikációs köztesréteg. Használatával elérhető, hogy az adatküldés megbízható, valósidejű, skálázható és nagy teljesítményű legyen.

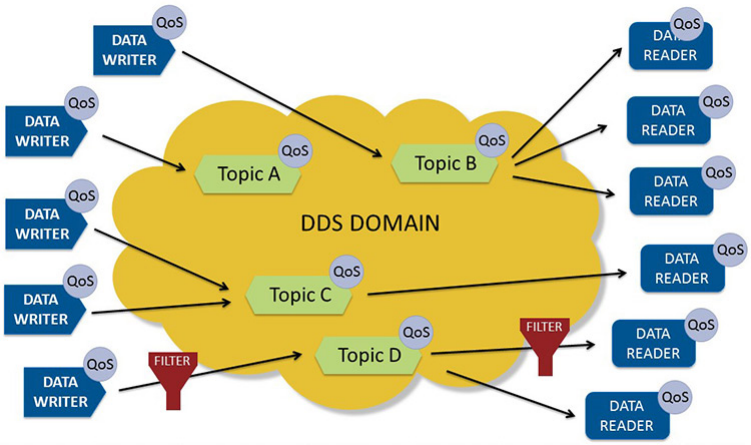
Publish-Subscribe kommunikációs séma

A DDS a kommunikáció megvalósításához a Publish-Subscribe sémát használja.

A Publish-Subscribe sémában nem direkt üzenetváltás történik a feladó és a címzett között. Az adatokat típusokba rendszerezi, a típusokat pedig úgynevezett Topic-okhoz köti. Egy Topic-on belül csak a regisztrált adattípusok használatával történhet üzenetváltás. Az üzenetváltás a következőképpen működik:

- az üzenet küldője elküldi az üzenetét a megfelelő DDS Topic-ba,
- az üzenet fogadója pedig a Topic-ra feliratkozva értesül az üzenet érkezéséről.

A 2.1. ábrán látható a DDS Publish-Subscribe sémával történő működése.



2.1. ábra. DDS Publish-Subscribe architektúra

DDS komponensei

A DDS működéséhez szükséges komponensek röviden részletezve:

- **Domain**: Minden DDS alkalmazás egy domainhez tartozik, csak azok az alkalmazások tudnak egymással kommunikálni, amelyek egy domainhez tartoznak.
- **DomainParticipant**: Egy adott domain tagja. Az összes többi entitás hozzá van rendelve.
- **Topic**: Adatobjektum, amelynek az értékeit publikálják a Publisher entitások és/-vagy olvassák a Subscriber entitások.
- **Publisher**: A DataWritereket magában foglaló entitás.
- **Subscriber**: A DataReadereket magában foglaló entitás.
- **DataWriter**: Adatot küld a Topic-ba.
- **DataReader**: Topic-ba érkező adatot fogad.

2.3. Digitális iker

A digitális ikrek ötlete legelőször 2003-ban, a NASA egyik előadásán merült fel, ami a Product Lifecycle Managementről szólt. Azt képzelték el, hogy a jövőben a termékeknek lesz egy virtuális modellje, ami eltárolja az adatait, és lehetővé teszi a termék fejlesztését, optimalizálását.

A digitális ikrek használata különböző ágazatokban 2003 után terjedt el, azonban a különböző alkalmazási területek miatt minden területen másképpen definiálják, más célokra használják fel. Ezért a digitális ikreknek nincs még konkrét, szabványosított definíciója. A legáltalánosabb definíció, hogy a digitális iker egy valós entitás digitális reprezentációja.

CPS rendszerek esetén a digitális iker feladatai a következők:

- Lehetővé teszi, hogy műveleteket hajtsunk végre a fizikai entitásokon.
- Tárolja a fizikai párjának aktuális állapotát.
- Számításokat végez, és ha szükséges, beavatkozik a fizikai párjának működésébe.

A közelmúltban elkezdődtek olyan nyílt forráskódú megoldások fejlesztései, amelyek lehetővé teszik digitális ikrek létrehozását a fizikai eszközeinkhez. Ezek közül az egyik legelterjedtebb megoldás az Eclipse Ditto.

Eclipse Ditto

Az Eclipse Ditto egy nyílt forráskódú digitális iker implementáció. Alább olvasható, hogy milyen funkciókat nyújt a felhasználó számára:

- Lehetővé teszi fizikai eszközök aktuális állapotának tárolását.
- Lehetővé tesz műveleteket az eltárolt párokon.
- Lehetővé tesz beavatkozásokat a fizikai eszközökön.

A fizikai eszközök aktuális állapotának tárolására a Ditto úgynevezett Thing-eket használ. [4] Egy Thing a következőképpen épül fel:

- **thingId** - Az adott Thing egyedi azonosítója.
- **definition** - A Thing leírása Vorto nyelven.
- **policyId** - A Thing adataira vonatkozó policy-k. Ez hozzáférési adatokat határoz meg.
- **attributes** - A Thing-ek statikus adatai.
- **features** - A Thing-ek aktuális állapotai.

A műveleteket a Ditto HTTP API-kon keresztül teszi lehetővé. A következő adatok elérhetőek:

- **Things** - A /things endpointot hívva le tudjuk kérdezni egy Thing adatait, törölhetünk Thing-et, és frissíthetjük az adatait egy Thing-nek.
- **Features** - Lehetőségünk van adott Thing feature-jeinek a változtatására, lekérdezésére, törlésére.
- **Policies** - Lehetőségünk van policy-ket létrehozni, törölni, változtatni.
- **Things-Search** - Lehetőségünk van szűrni az eltárolt Thing-einket a Ditto által biztosított szűrőkkel.
- **Messages** - Üzenetküldési lehetőséget biztosít.

A Ditto számos kommunikációs protokollt támogat, így sokfajta kommunikációs rétegre illeszthetjük rá a Ditto-t. Ezek a kommunikációs protokollok a következőek:

- AMQP 0.9.1
- AMQP 1.0
- MQTT 3.1.1
- MQTT 5
- HTTP 1.1
- Kafka 2.x

A Ditto az ezekkel a protokollokkal érkező üzeneteket a saját Ditto Protocol-jára képezi le. A Ditto Protocol üzenetei a következő felépítésűek:

- **topic** - Ennek a mezőnek a segítségével tudjuk megadni, hogy milyen cselekvést szeretnénk végrehajtani és min. Például változtatni szeretnék egy a topicban ID-val megadott Thinget.
- **path** - A Path változó segítségével tudjuk specifikálni például, hogy a Thing pontosan melyik részét szeretnénk megváltoztatni.
- **value** - A value-ban adjuk meg az értéket, amire szeretnénk megváltoztatni.

2.4. Eclipse Vorto

A Vorto egy magasszintű modellezési nyelv, aminek a segítségével bizonyos domain-ben lévő eszközök által szolgáltatott adatokat, illetve rajtuk vagy általuk végzett cselekvéseket tudunk leírni. Informationmodel-eket hozhatunk létre rendszerekhez, amelyek a komponenseikről functionblock-okat tartalmaznak. Egy functionblock felépítése:

- **azonosítók** - Ide kerül a verziószám, namespace, a functionblock neve.
- **configuration** - A configuration részhez kerülnek az adott komponens statikus metaadatai.
- **status** - A status részhez kerülnek a komponens állapotai, változói.
- **fault** - A fault részhez kerülnek a meghibásodáshoz kapcsolható részek.
- **operations** - Ide kerülnek az adott komponensen végrehajtható cselekvések leírásai.
- **events** - Ide kerülnek a komponens által végzett, illetve kiváltott cselekvések leírásai.

2.5. Yakindu

A Yakindu eszközkészlet az alábbi feladatokhoz használható:

- Állapotgépek modellezése
- Állapotgépek szimulációja
- Kódgenerálás: futtatható állapotgépek generálása

A Yakindu a felsorolt feladatok elvégzéséhez a következő szolgáltatásokat nyújtja:

- **Grafikus szerkesztő** az állapottérképek létrehozásához és szerkesztéséhez
- **Szimulátor** az állapottérkép viselkedésének szimulálásához
- **Kódgenerátorok** Java, C, C++ és Python nyelvekhez
- **Egyedi generátor projektek** saját kódgenerátorok fejlesztéséhez
- **Beépített validátor** az állapottérkép szintaktikai és szemantikai helyességének ellenőrzéséhez
- **Tesztelési keretrendszer** unit tesztekhez

2.6. Blokklánc technológiák

Elosztott főkönyvi technológiák

A főkönyv egy tranzakciók nyilvántartására szolgáló adatbázis, amelyhez előre meghatározott szabályok szerint fűzhetőek hozzá új tranzakciók. Az elosztott főkönyvi technológiák (Distributed Ledger Technology, DLT) [5] főbb jellemzői az alábbiak:

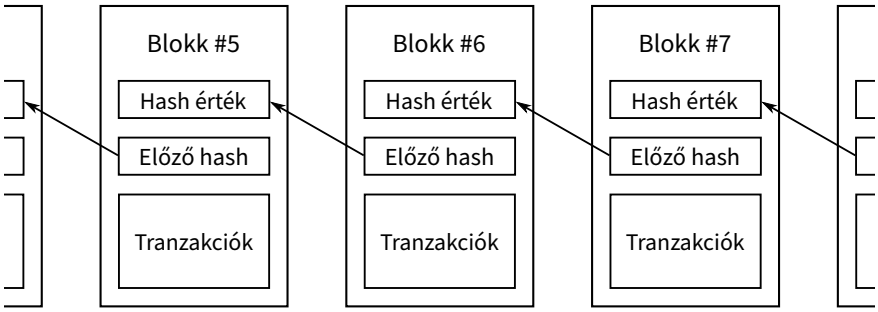
- A főkönyvet nem egy központi, sok esetben harmadik fél által üzemeltetett csomópont tárolja.
- Egy adott hálózat résztvevői mind rendelkeznek a főkönyv egy-egy példányával.
- Lehetővé teszik, hogy egy adott hálózat résztvevői között konszenzus jöjjön létre a főkönyvben tárolandó, sorba rendezett tranzakciókról.

A DLT-k előnyei az alábbiak:

- Nincs dedikált csomópont, melynek meghibásodása a teljes rendszer működésképtelenségét okozza.
- Kiküszöbölhető a főkönyvben rögzített tranzakciók meghamisítása.

Blokklánc

A blokklánc [3] esetében a főkönyvben nyilvántartandó tranzakciók a 2.2. ábrán látható blokkokba kerülnek. A kezdeti blokk kivételével minden blokkban eltárolásra kerül az azt megelőző blokk kriptográfiai hash értéke, tehát az összeállított blokkok egy blokkláncot alkotnak.



2.2. ábra. Egy blokkláncrészlet

A főkönyv magát a blokkláncot tárolja. A blokklánc biztosítja a blokkokba foglalt tranzakciók utólagos megmásíthatatlanságát: ha egy korábban közzétett blokk megváltozna, akkor a hash értéke is megváltozna, és ezáltal az összes további blokk hash értéke is megváltozna, mivel mindegyik tartalmazza az előző blokk hash értékét.

Okosszerződések

Nick Szabo az okosszerződést a következőképpen definiálta 1994-ben: az okosszerződés egy számítógépesített tranzakciós protokoll, ami végrehajtja egy szerződés feltételeit. [6]

A gyakorlatban az okosszerződések olyan programkódok, amelyeket egy blokklánc hálózat résztvevői futtatnak. Az okosszerződések futtatásának eredménye a blokkláncban kerül eltárolásra. Az okosszerződéseket is a blokklánc tárolja.

2.7. DAML

A DAML egy nyílt forrású programozási nyelv és platform okosszerződések fejlesztéséhez, amely például jól használható többszereplős üzleti folyamatok megvalósításához. [2]

Egy DAML főkönyvben szerződések, szerződéspéldányok tárolhatók. A szerződések sablonokból hozhatók létre, és egy sablon a következő részekből áll:

- **Adattagok**: Egy sablon különféle típusú, akár összetett adattagokat is definiálhat.
- **Szerepkörök**: Egy személyt vagy jogi entitást reprezentáló résztvevő az alábbi szerepkörökkel rendelkezhet:
  - Signatory: jóvá kell hagynia a szerződéspéldányok létrehozását.
  - Observer: láthatja a szerződéspéldányt és az azzal kapcsolatos információkat.
  - Controller: végrehajthat egy adott choice-t egy adott szerződéspéldányon.
  - Maintainer: része a szerződéskulcsnak.
- **Choice-ok**: Meghívásukkor végrehajtódik a törzsükben lévő kód (az általános célú programozási nyelvek függvényeihez hasonlóak).

A létrehozott szerződéspéldányok archiválhatók. Az egyes szerződéspéldányok a létrehozásuk után nem módosíthatók, ezért az adattagok módosítása csak az alábbi lépések végrehajtásával lehetséges:

1. Új szerződéspéldány létrehozása a módosított adattagokkal.
2. Az eredeti példány archiválása.

A szerződéspéldányok opcionálisan rendelkezhetnek szerződéskulccsal, amely a relációs adatbázisok elsődleges kulcsához hasonlóan működik. A DAML nyelven írt forráskódok teszteléséhez szcenáriók definiálhatók.

3. fejezet

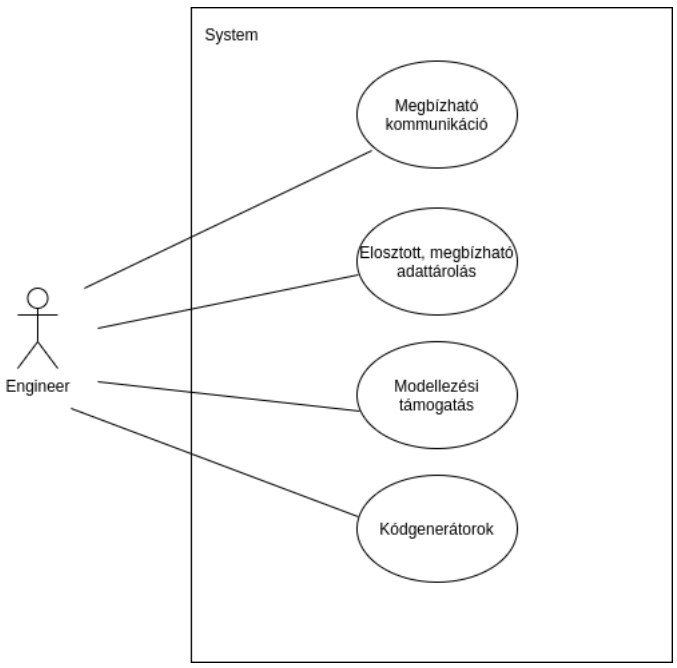
Blokklánc alapú digitális iker modellvezérelt fejlesztése

Ebben a fejezetben bemutatjuk munkánk motivációját, áttekintjük építőelemeit és tárgyaljuk, hogy az általunk készített eszközök hogyan támogatják a mérnököket abban, hogy ki tudják használni a digitális ikerpár és elosztott főkönyv megközelítések előnyeit kritikus CPS rendszerek fejlesztése és üzemeltetése során.

3.1. Motiváció

A kritikus kiberfizikai rendszerek megbízható működésének biztosítása elengedhetetlen. A megbízhatóságot nem csak futásidőben kell biztosítani, hanem érdemes a fejlesztés során is lehetőleg már a tervezés korai fázisában kiszűrni a hibákat.

Továbbá a biztonságkritikus funkciók működése felelősségbeli kérdéseket is felvet: egy elosztott CPS-ben célunk annak biztosítása, hogy a rendszer által hozott egyes döntések mindenki számára látható módon, visszakövethetően és megmásíthatatlanul kerüljenek tárolásra.



3.1. ábra. Use case-ek

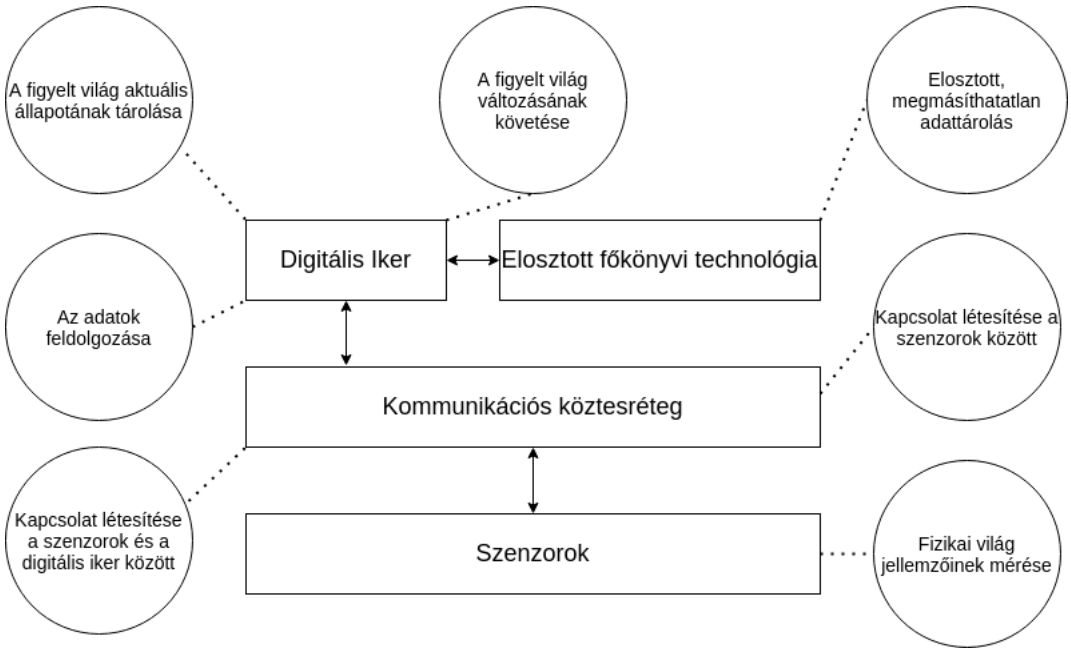
Ezen követelmények teljesítésének érdekében az általunk bemutatásra kerülő megközelítés a 3.1. ábrán látható szolgáltatásokat nyújtja. Ezek részletesebben a következők:

- A terepi kommunikáció a nagy megbízhatóságú és teljesítményű, szabványos DDS technológia segítségével kerül megvalósításra.
- Digitális ikerpár technológiát csatolunk a kommunikációs köztesréteghez, amely reprezentálja a valóság egy képét. Ezen lehetőségünk van többféle műveletet végezni és felhasználhatjuk a rendszer optimalizációjára.
- Modellezési támogatást nyújtunk a kommunikációs köztesréteg és a digitális ikerpár tervezéséhez, és automatikus kódgenerátorokat adunk a fejlesztés támogatására.
- Elosztott főkönyvi technológiát illesztünk a rendszerhez, amely lehetővé teszi egyrészt a kritikus adatok, másrészt a vezérlés lépéseinek letagadhatatlan módon történő eltárolását, ezáltal biztosítva az adatintegritást és a nyomonkövethetőséget a teljes rendszerben.
- Modellezési támogatást nyújtunk ahhoz, hogy az elosztott főkönyvi technológia egyszerűen illeszthető legyen a rendszerünkhöz, és lehetővé tesszük az okosszerződések magas szintű nyelven alapuló modellvezérelt fejlesztését. Az okosszerződések modellből történő generálásához automatikus kódgenerátort biztosítunk. Az általunk választott technológia lehetőséget biztosít az okosszerződések további analízisére és formális vizsgálatára is.

A 3.2. ábrán látható magas szintű architektúra segítségével támogatjuk a fentebb említett kritériumok teljesítését: ezen rendszerek modellalapú tervezését és megvalósítását támogatjuk.

3.2. Megközelítés áttekintése

A megközelítésünk blokklánc alapú digitális ikrek kiberfizikai rendszerekhez való illesztését segíti. Ebben az alfejezetben ismertetjük az általunk elképzelt kiberfizikai rendszert alkotó, a 3.2. ábrán látható komponensek feladatait és az ezekkel kapcsolatos követelményeket.



3.2. ábra. Architektúra feladatmegjelölésekkel

Szenzorok

A szenzorok feladata a fizikai világ előre meghatározott jellemzőinek mérése. A szenzorok azáltal, hogy továbbítják a mérések eredményeit, információt szolgáltatnak a fizikai világról a kiberfizikai rendszer számára. Az összegyűjtött szenzoradatok felhasználása teszi lehetővé, hogy a kiberfizikai rendszer döntéseket hozzon és a fizikai világ egyes részeit vezérelje. A kiberfizikai rendszerekben szereplő szenzorok nagy mennyiségű adatot szolgáltatnak, így olyan kommunikációs köztesrétegre van szükségünk, ami alkalmas nagy mennyiségű adat kezelésére.

Kommunikációs köztesréteg

A kommunikációs köztesréteg feladata a szenzorok és a digitális iker közötti kommunikáció biztosítása, a szenzoroktól érkező adatok megbízható továbbítása. A kommunikációs köztesréteg megfelelő működése elengedhetetlen a digitális iker naprakészen tartásához és ezáltal a valósidejű döntéshozás megfelelő működésének biztosításához. A kommunikációs köztesrétegnek biztosítania kell a valósidejű beavatkozások továbbítását is.

A DDS egy olyan kommunikációs köztesréteg megoldás, amely teljesíti az általunk és a kiberfizikai rendszerek által a kommunikációs köztesréteg felé támasztott követelményeket. Ezért a DDS-t alkalmazzuk a megoldásunkban. Továbbá mivel a megoldásunkban alkalmazott Eclipse Ditto nem támogatja a DDS-t, így a DDS üzeneteket Ditto Protocol üzenetekké kell alakítanunk a digitális iker illesztéséhez.

Digitális iker

A digitális iker egy valós entitás digitális reprezentációja. Megfelelő absztrakciót nyújt, hogy segítségével különböző műveleteket végezzünk és megfelelően beavatkozzunk a valós folyamatok futásába.

A digitális iker feladata tehát a fizikai, valós entitás állapotának figyelése és tárolása. További feladata az állapotváltozások követése, majd a változásokból kinyert információ alapján döntések és vezérlés számítása, küldése a fizikai entitásoknak.

A digitális iker a fizikai világból származó információkat gyakran feldolgozva teszi elérhetővé a felhasználók számára. Ez alapján a felhasználók üzleti döntéseket hozhatnak, beavatkozhatnak a rendszer működésébe vagy optimalizálhatják a folyamataikat. Mivel a digitális ikrek valós időben avatkoznak be a fizikai eszközök működésébe, így a kommunikációs köztesrétegnek alkalmasnak kell lennie ilyen beavatkozások továbbítására.

Digitális ikrek kezelésére létezik egy nyílt forráskódú megoldás, az Eclipse Ditto. A Ditto úgynevezett Thing-ekben tárolja el a fizikai eszközök aktuális állapotát. Ezeken a Thing-eken keresztül kérdezhetőek le az eszközeink adatai valós időben. Ezáltal a Ditto használatával tudjuk kezelni fizikai eszközök aktuális állapotát, ezek le is kérdezhetőek, illetve támogatja üzenetek, parancsok küldését eszközök felé. Az előbb felsoroltak miatt ezt a technológiát építettük be a megoldásunkba.

Elosztott főkönyvi technológia

Az elosztott főkönyv feladata a megbízható, elosztott módon történő adattárolás megvalósítása, méghozzá magas szintű adatbiztonságot és hibatűrést biztosítva. Az elosztott főkönyvvel kapcsolatos legfontosabb elvárás az adatok megmásíthatatlansága, illetve az elvégzett műveletek letagadhatatlansága.

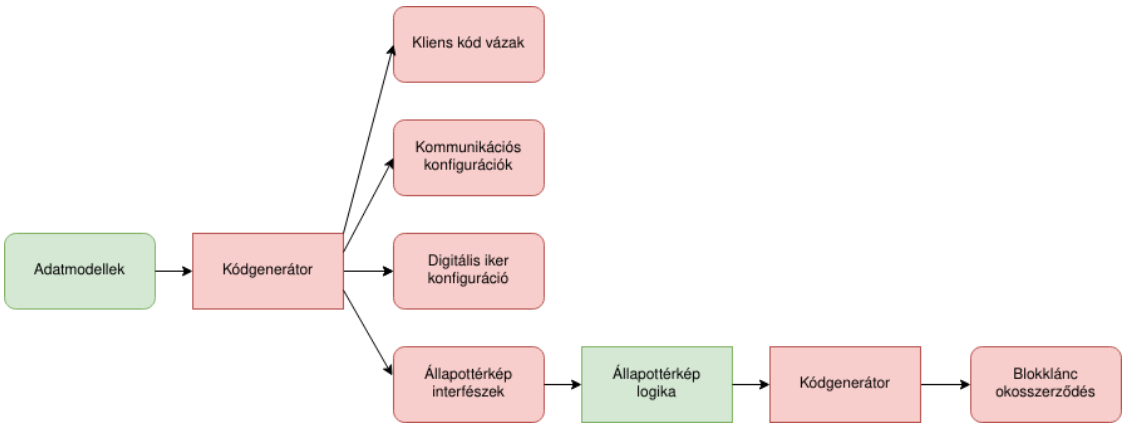
Napjainkban az elosztott főkönyvek alkalmazási esetei többnyire valamely blokklánc technológia felhasználásával kerülnek megvalósításra. A blokklánc technológiák általában lehetővé teszik okosszerződések végrehajtását, és ezáltal támogatják például üzleti tranzakciók vagy döntéshozás megvalósítását.

3.2.1. Modellalapú tervezési támogatás

A fenti felsorolásban bemutattuk az általunk elképzelt kiberfizikai rendszer komponenseit az információk forrásától egészen az elosztott adattárolásig és vezérlésig. Azonban egy ilyen összetett rendszert megtervezni és elkészíteni nehéz feladat, ezért munkánk során célunk modellezési és kódgenerálási támogatás nyújtása blokklánc alapú digitális iker rendszerekhez megbízható kommunikációs köztesréteg felett.

A modellezéssel célunk, hogy konzisztensen egy modell segítségével konfiguráljuk a kommunikációt, definiáljuk az egyes komponensek interfészeit, továbbá modellalapú tervezési támogatást nyújtsunk a blokklánc okosszerződések elkészítéséhez.

A teljes modellalapú fejlesztési folyamatot a 3.3. ábra mutatja be. Az ábrán a zöld szín jelzi a felhasználó által elkészítendő részeket, a piros szín pedig a rendszerünk által biztosított részeket.



3.3. ábra. Modellalapú fejlesztés folyamatábrája

A folyamat során először az általunk biztosított generátor az adatmodellekből előállítja a kommunikációs konfigurációkat, a digitális iker konfigurációt, a kliens kód vázakat és az állapottérkép interfészeket. Ezután a felhasználónak az állapottérkép interfészek felhasználásával el kell készítenie az állapottérkép-alapú logikát. Végül az általunk biztosított generátor az állapottérkép alapján előállítja a blokklánc okosszerződést.

3.3. Digitális iker fejlesztésének modellalapú támogatása

Ebben az alfejezetben a fizikai eszközök pillanatnyi állapotának tárolásáért és a rajtuk végrehajtható cselekvések végrehajtásáért felelős digitális ikrek modellalapú fejlesztését mutatjuk be.

3.3.1. Modellezés

A digitális ikrek a fizikai párjukhoz DDS kommunikációs köztesrétegen keresztül kapcsolódnak. Kutatásunk során olyan magas szintű modellezési nyelvet kellett keresnünk, aminek a segítségével a Ditto által használt Thing-ek konfigurációját és a DDS-en való kommunikációhoz szükséges konfigurációkat, illetve az ikrek megfigyeléséhez és a kommunikációhoz szükséges kliens kódokat is tudjuk generálni.

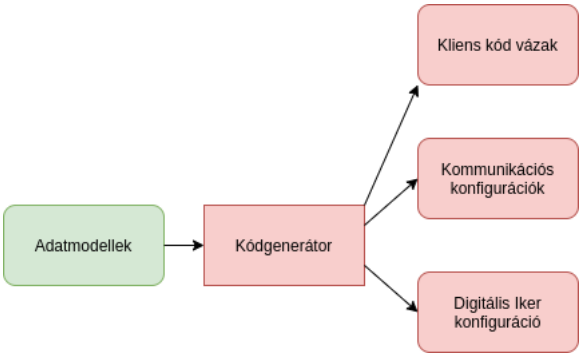
Az Eclipse Vorto modellezési eszközre esett a választásunk. A Vorto segítségével leírhatóak a mi területünkön szereplő eszközök adatai, opcionálisan az általuk végrehajtott, vagy a rajtuk végrehajtandó tevékenységek. A megoldásunkban a Vorto functionblock leírásban definiált ötfajta kategóriájú adattagokból generáljuk a számunkra szükséges részeket.

A status és configuration kategóriába tartozó részekből hozzuk létre a digitális iker konfigurációkat, illetve a kommunikációs konfigurációkat és a kliens kód vázak bizonyos részeit, mivel az ebbe a kategóriába eső adatok írják le az eszközeink állapotait, illetve konfigurációs adatait. Az operations és event kategória segítségével hozzuk létre az állapottérkép modellek előállításához szükséges interfész leírásokat, mivel ezek a kategóriák tartalmaznak olyan részeket, amelyek az eszközeink által kiváltható eseményeket, illetve az eszközeinken végrehajtható eseményeket írják le. A fault kategóriát nem használjuk fel a megoldásunkban.

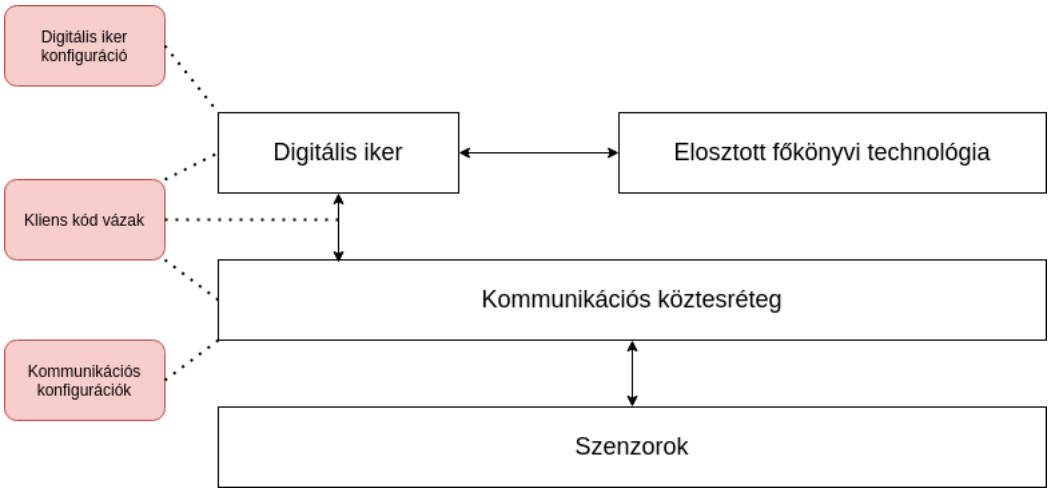
Mivel egyszerre tudunk modellezni eseményeket és adatokat ezzel az eszközzel, így tervezésünk során úgy döntöttünk, hogy Vorto leírásokból fogjuk a szükséges komponenseinket generálni.

3.3.2. Kódgenerálás

A kódgenerátor absztrakt folyamatábrája a 3.4. ábrán látható. Látható az ábrán, hogy a kódgenerátor adatmodelleket vár és ezek segítségével generál kliens kód vázakat, kommunikációs konfigurációkat és digitális iker konfigurációkat. A 3.5. ábrán látszik, hogy az előbb említett generált erőforrások az architektúránk melyik komponenséhez tartoznak. A kliens kód vázak a digitális iker figyeléséhez, a digitális iker és kommunikációs köztesréteg közötti kommunikációhoz, illetve a kommunikációs rétegen történő kommunikácóhoz szükségesek. A digitális iker konfigurációk a digitális ikrek kezeléséhez kellenek. A kommunikációs konfigurációk pedig a kommunikációs köztesréteg létrehozásához szükségesek.



3.4. ábra. Digitális iker és a DDS-hez szükséges részek generálása



3.5. ábra. Az architektúra összekapcsolva a generált részekkel

Kommunikációs konfigurációk A megközelítésünkben DDS kommunikációs köztesréteget használunk, mivel a DDS megbízható kommunikációt nyújt elosztott rendszerekben. A kiberfizikai rendszerben szereplő eszközök ezen keresztül kommunikálnak, illetve az eszközök az ikerpárjukkal is ezen keresztül kommunikálnak. A DDS beállításához meg kell adnunk egy (XML vagy IDL) konfigurációs fájlt. A megoldásunkban Python és Javascript klienseket használunk a DDS alkalmazására, ezek a kliensek az XML formátumú konfigurációs fájlokat támogatják, ezért a DDS konfigurációhoz XML-t használunk. A megfelelő beállításhoz generálandó elemek az alábbiak:

- DDS Típusok
- DDS Topic-ok
- DDS DataReader-ek
- DDS DataWriter-ek

Ezek a komponensek a Vorto leírások status és configuration adattagjaiból generálhatóak. A Vorto-ban definiált adattagok mellett generálódik egy Message típus és topic. Ezen keresztül küld a Ditto üzeneteket az eszközök felé. Azonban a Ditto aktuális verziója nem támogatja a DDS-en való kommunikációt, ezért létrehoztunk egy közvetítőt, ami a DDS üzeneteket a Ditto által használt Ditto Protocol üzenetekre alakítja át. Erről részletesebben a Kliens kód vázak alfejezetben írunk.

Digitális iker konfiguráció

A digitális ikrek konfigurációjához Ditto Thing struktúrákat kell generálnunk. A Ditto Thingek beállításánál lehetőségünk van megadni a tárolandó adatokat, illetve Policy-ket szabhatunk meg arra tekintettel, hogy melyik adattaghoz ki férhet hozzá és milyen műveleteket hajthat végre. Jelen megoldásunk az adatbeállításokat használja, a Policy által nyújtott lehetőségeket nem aknázza ki. A Ditto Thing-ek azonosítója a Vorto leírásban található namespace és functionblock nevéből áll elő. A Thing attribútumai a Vorto configuration adattagjaiból, a feature-jei pedig a status adattagokból generálódnak.

Kliens kód vázak

Az adatmodelleinkből kliens kód vázak is generálódnak, amelyek vagy a digitális iker és a kommunikációs köztesréteg közötti összekötésért felelősek, vagy a digitális ikrekben végbe-

ment változások követéséért. A digitális ikert figyelő kód egy vázat nyújt, ami önmagában csak az ikrek változását figyeli. Azonban ez kiegészíthető a felhasználó által egyéb funkciókkal, például az eszközök felé történő üzenetküldéssel. Továbbá generálódik egy példa kód DDS Topic-okba való íráshoz, amelyet ki lehet egészíteni például szenzorokról való adatlekérdezéssel. A kommunikációs köztesréteget és a digitális ikreket összekötő kliens teljes egészében generálódik. Az Eclipse Ditto széleskörű kommunikációs protokoll támogatása nem tartalmazza a DDS-t. Ennek érdekében szükség volt egy olyan kliens generálására, amely ezt az összekötést elvégzi. Lényegében az előbbiekben említett generált XML-t, a Vorto-ban definiált adattagokat és a létrejött Thing struktúrát alkalmazza a létrejövő kliens. Működésekor feliratkozik az XML-ben definiált DDS Topic-okra és amint üzenet érkezik a Topic-on, Ditto Protocol szerinti üzenetet küld a küldőhöz tartozó digitális iker számára.

A második kliens a konfigurált Ditto Thingek figyelését végzi. Ennek a generálásához csak a Vorto leírásokban definiált namespace-re és azonosítókra van szükség.

A harmadik kliens pedig egy példa kódot mutat a DDS köztesrétegbe való íráshoz, az adott Topic-okra. Ehhez a generált XML-re és a Vorto-ban definiált adattagokra van szükség.

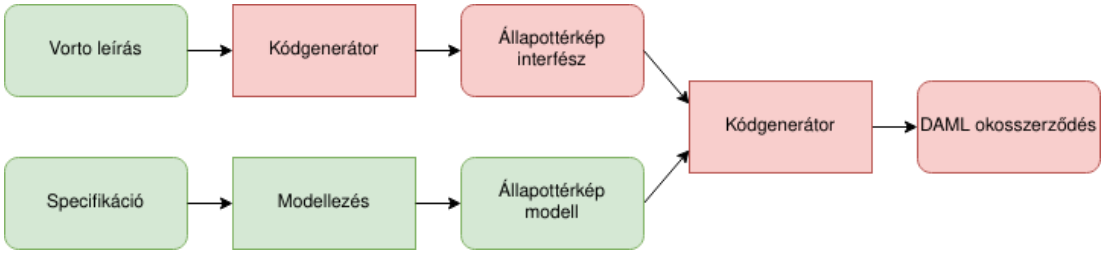
3.3.3. DDS és Digitális iker futtatása

A futtatáshoz készítettünk egy scriptet, ami argumentumként várja a a definiált Vorto functionblock-okat tartalmazó fájlokat. Az argumentumok feldolgozása után futtatja a kódgenerátort, amivel létrejön minden szükséges fájl. Ezek után elindítja az Eclipse Ditto-t lokálisan, majd az Eclipse Ditto-t és a DDS-t összekötő klienst. Ezután pedig futtatja a Ditto Thing-eket figyelő klienst. Ezt követően a rendszer ezen része készen áll az adott adatmodellekkel rendelkező eszközök adatainak fogadására, és aktuális állapotainak digitális ikrekben való tárolására.

3.4. Blokklánc alapú vezérlés fejlesztésének modellalapú támogatása

Ebben az alfejezetben az adatok tárolásáért és a tranzakciók végrehajtásáért felelős blokklánc okosszerződések modellalapú fejlesztését mutatjuk be.

A fejlesztési folyamatot a 3.6. ábra foglalja össze. Ahogy az ábrán is látszik, az okosszerződések fejlesztésének folyamata jól illeszkedik a digitális iker koncepciójához, azaz egy olyan blokklánc alapú vezérlőt kapunk, amely kompatibilis a korábbi fejezetben bemutatott rendszerrel.



3.6. ábra. Az okosszerződések modellalapú fejlesztésének folyamata

3.4.1. Modellezés

A CPS rendszerek adatokat gyűjtenek a fizikai világról, majd döntéseket hoznak és ez alapján irányítják a fizikai világ egyes részeit. Mivel a döntéshozáshoz sok esetben szük-

séges valamilyen belső állapot nyilvántartása, a rendszer viselkedése jól modellezhető állapottérképekkel. Ezért a megközelítésünk részét képezi az okosszerződések állapottérképek alapján történő származtatása.

A viselkedést modellező állapottérképek elkészítéséhez többféle szoftver is elérhető. Az Eclipse fejlesztőkörnyezetre épülő Yakindu modellező eszköz támogatja az állapottérképekből történő kódgenerálást, valamint lehetővé teszi egyedi kódgenerátorok fejlesztését is Xtend vagy Java nyelven. A Yakindu többek között azért ideális választás az okosszerződésfejlesztés támogatásához, mert az állapottérképek megjelenítéséhez és szerkesztéséhez egy egyszerűen kezelhető grafikus modellt biztosít a mérnökök számára.

A tervezés során úgy döntöttünk, hogy egy olyan Yakindu kódgenerátort fejlesztünk, amely képes blokklánc okosszerződések származtatására Yakindu állapottérképekből, és ezáltal nagymértékben elősegíti az okosszerződések modellalapú fejlesztését.

Ahogyan a 3.6. ábrán látható, a Yakindu kódgenerátor bemenete egy állapottérkép modell, illetve a hozzá tartozó interfész leírás. Az állapottérkép modellt a kiberfizikai rendszer specifikációja alapján a mérnököknek kell elkészíteni, de olyan módon, hogy a modell kompatibilis legyen a Vorto leírásból generált interfész leírással. A kiberfizikai rendszert célszerű úgy fejleszteni, hogy egy-egy fizikai entitáshoz egy-egy állapottérkép tartozzon.

3.4.2. Kódgenerálás

A kódgenerátor elkészítéséhez választanunk kellett egy megfelelő célnyelvet. Mivel a napjainkban népszerű blokklánc technológiák különböző nyelveken írt okosszerződéseket támogatnak, fontos elvárás a célnyelvvel szemben, hogy támogassa a generált okosszerződések különböző blokklánc platformokkal történő integrációját. Ezáltal bizonyos mértékben biztosítható az egyes blokklánc platformok közötti hordozhatóság.

A tervezés során úgy döntöttünk, hogy célnyelvnek a DAML-t választjuk és ehhez a nyelvhez készítünk kódgenerátort. A kódgenerátor fejlesztéséhez meg kellett határozni, hogy a Yakindu állapottérképek egyes elemei hogyan kerülnek leképezésre DAML nyelvi elemekké, konstrukciókká.

Az állapottérkép alkotóelemei és a DAML nyelv közötti leképezéseket a 3.1. táblázat foglalja össze. A következőkben ismertetésre kerülnek az egyes állapottérkép elemek leképezésének részletei.

Állapottérkép	**DAML**
állapot	enum érték
állapotátmenet	choice
összetett állapot	enum érték + choice
műveletek (belépés/kilépés/átmenet)	choice body
őrfeltételek	if/else elágazás
összetett állapot előzménye (history)	változó

3.1. táblázat. Yakindu állapottérkép elemeinek leképezése DAML nyelvre

Állapot Az állapottérkép állapotai enum értékekként kódolva kerülnek tárolásra az okosszerződésben. A példányosított okosszerződés egy változóban nyilvántartja az aktuálisan aktív állapotokat. Erre az engedélyezett átmenetek meghatározásához van szükség.

Állapotátmenet Az okosszerződés az állapotátmenetet (azaz az aktuális állapot frissítését és az átmenethez tartozó műveletek végrehajtását) egy choice meghívásaként való-

sítja meg. Mivel a DAML szerződéspéldányok megváltoztathatlanok, ezért a belső állapot változásakor egy új szerződéspéldány jön létre.

Összetett állapot Az okosszerződés az összetett állapotokat egyrészt enum értékekként kódolva tárolja, másrészt az összetett állapotokhoz choice-ok is tartoznak. Ezek a choice-ok felelnek az összetett állapotba történő belépéskor, illetve az összetett állapotból történő kilépéskor az aktív állapotok frissítéséért.

Műveletek Egy adott állapotba történő belépéshez vagy egy állapotból történő kilépéshez, illetve az átmenethez végrehajtandó műveletek is tartozhatnak. Ezek egy adott choice törzsében lévő DAML nyelvű kifejezésekké képezhetők le.

Őrfeltételek Az őrfeltételekből if/else elágazások állíthatók elő. Egy adott átmenet nem kerül végrehajtásra, ha a hozzá tartozó őrfeltétel nem teljesül.

Összetett állapot előzménye Az állapotelőzményt a példányosított okosszerződés egy változója tárolja, enum érték(ek)ként kódolva.

Események Az állapottérkép (bejövő, kimenő) eseményeinek kezelése kétféleképpen képzelhető el: az első esetben az események kezelése az okosszerződés felelőssége, míg a második esetben nem. Esetünkben a kódgenerátor az első esetnek megfelelően került megvalósításra.

1. Ha az események kezelése az okosszerződés felelőssége, akkor az állapottérkép eseményeiből choice-ok állíthatók elő, és ezen choice-ok valósítják meg az esemény hatásának végrehajtását (például meghívnak egy átmenethez tartozó choice-t).

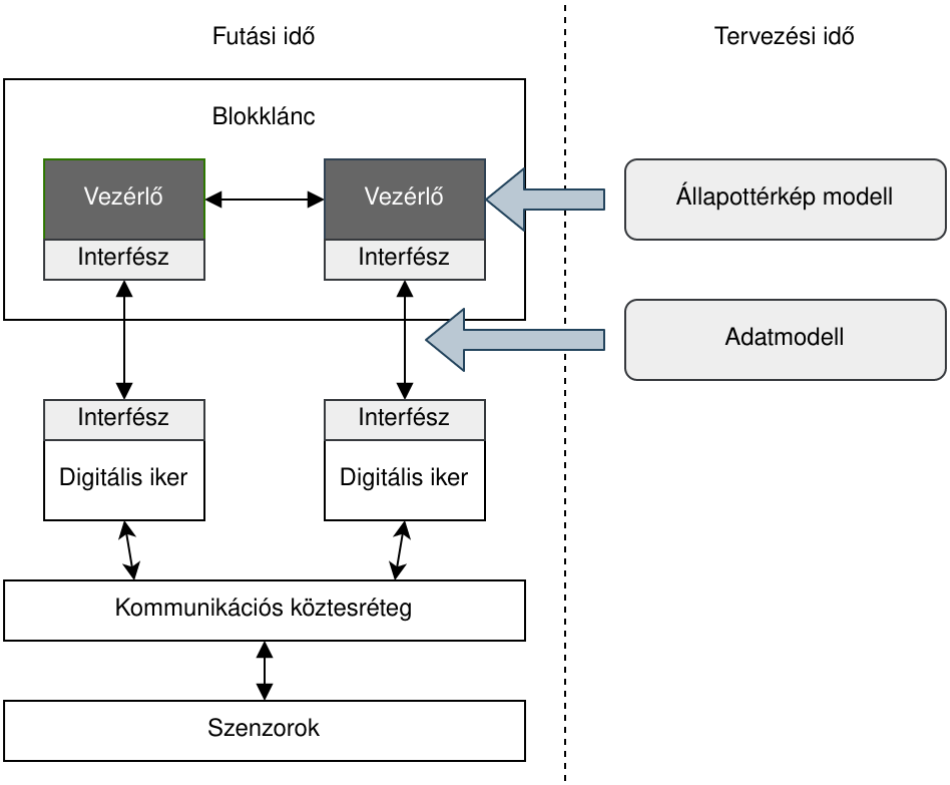
2. Ha az események kezelése nem az okosszerződés felelőssége, hanem egy másik komponensé, akkor az eseményekből nem kell choice-okat előállítani, hanem elegendő az átmenetek leképezése.

3.4.3. A generált kód végrehajtása

A 3.7. ábrán látható az általunk elképzelt blokklánc alapú digitális iker futási idejű architektúrája, illetve a tervezés és a futtatás kapcsolata.

A kódgenerátor által előállított DAML okosszerződések többféle blokkláncra platformra is telepíthetőek. Az okosszerződések példányosításával hozhatók létre a blokklánc alapú vezérlők.

Egy adott, a rendszerben bekövetkező esemény a DAML platform által biztosított interfészen keresztül jutnak el a példányosított okosszerződésekhez, méghozzá úgy, hogy meghívásra kerül az adott eseményhez tartozó choice. A meghívott choice felelőssége az esemény kezelése, azaz az aktív állapot(ok)tól függően az eseményhez tartozó megfelelő átmenet végrehajtása.



3.7. ábra. A tervezés és a futtatás kapcsolata

4. fejezet

Esettanulmány

Ebben a fejezetben egy konkrét, általunk fejlesztett esettanulmányon keresztül mutatjuk be, hogyan támogatják az általunk készített eszközök egy kritikus kiberfizikai rendszer tervezését és fejlesztését.

4.1. Esettanulmány áttekintése

Ebben az alfejezetben bemutatjuk az esettanulmányt, ami egy proof-of-concept okoskereszteződés fejlesztése. Az általunk elképzelt okoskereszteződésben egy közút egy vasútvonalat keresztez. Az okoskereszteződés az alábbi feladatokat látja el:

- szenzoradatokat gyűjt,
- az összegyűjtött adatok alapján döntéseket hoz, és
- befolyásolja a közúti forgalmat azáltal, hogy üzeneteket küld a közúton érkező autóknak.

Az okoskereszteződés az alábbi szenzorok által mért adatokat gyűjti:

- Vonatdetektáló szenzor
- Az érkező autó sebesség szenzora
- Hőmérséklet szenzor
- Eső szenzor

A vonatdetektáló szenzor feladata az, hogy értesítse az okoskereszteződést a vonat érkezéséről. Az érkező autó sebesség szenzora jelzi az autó közeledését a kereszteződés felé, illetve baleset esetén a hatóság számára hasznos információ a balesetben résztvevő autó sebessége. A hőmérséklet és eső szenzor a kereszteződés természeti körülményeiről adnak tájékoztatást, egy esetleges baleset utólagos vizsgálata során ezek az információk is hasznosak a hatóság számára.

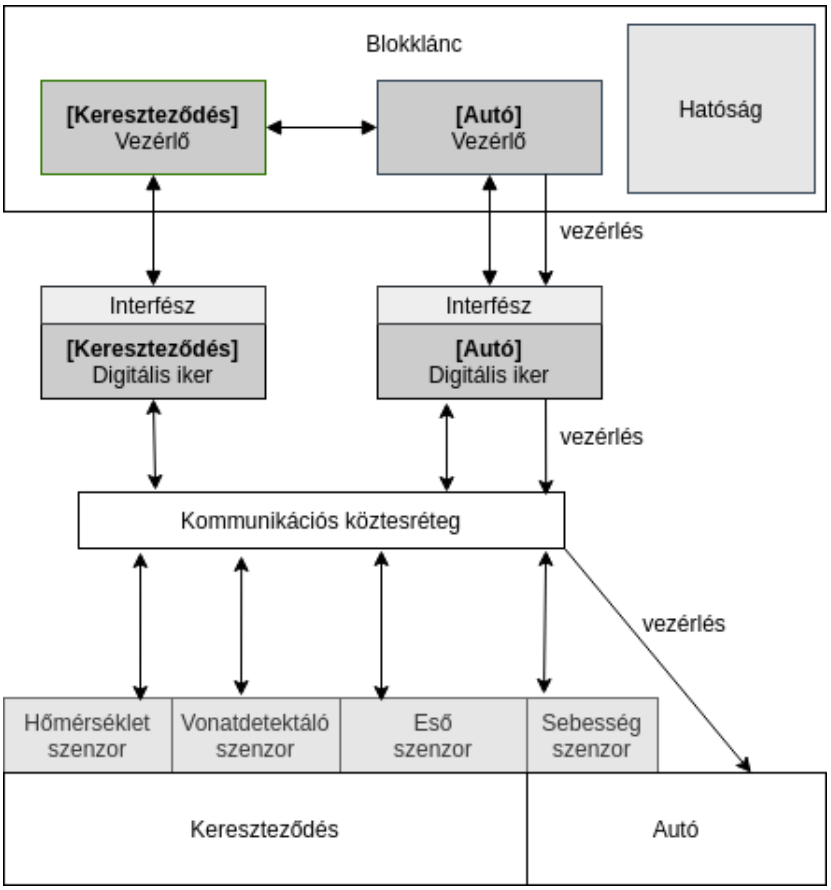
Az okoskereszteződés működésével kapcsolatban több funkcionális követelményt is megfogalmaztunk, melyek a következők:

- Az okoskereszteződésnek folyamatosan fel kell dolgoznia a szenzoroktól kapott adatokat, és ez alapján nyilván kell tartania, hogy érkezik-e autó és/vagy vonat.
- Ha az okoskereszteződés érkező autót észlel, a helyzettől függő üzenetet kell küldenie annak:

  - Ha az okoskereszteződés nem észlel érkező vonatot, akkor az autó áthaladását meg kell engednie, és ezt jeleznie is kell az autónak.
  - Ha az okoskereszteződés érkező vonatot is észlel, akkor az autó áthaladását nem engedheti meg, és ezt jeleznie is kell az autónak.
- A kereszteződést felügyelő hatóság (vasúttársaság, biztosító, egyéb jogi személy) utólag le tudja kérdezni az okoskereszteződés által hozott döntéseket.

4.2. Megközelítésünk áttekintése az esettanulmányon

Ebben az alfejezetben bemutatjuk, hogy az általunk készített megoldás hogyan alkalmazható az előző alfejezetben ismertetett esettanulmányon. A 4.1. ábrán szemléltetjük az esettanulmány részletes felépítését, az általunk kidolgozott CPS architektúrára vetítve.



4.1. ábra. Az esettanulmány futási idejű architektúrája

A megoldásunk létrehoz egy megbízható kommunikációs köztesréteget, amin keresztül a kereszteződés és az érkező autó tudja publikálni a szenzorai által mért adatokat. Ezek az adatok a megfelelő digitális ikerben frissülnek, és a változások alapján a blokklánc frissíti az adott entitás állapotát.

Ha a rendszerünk azt érzékeli, hogy egyszerre közeledik autó és vonat a kereszteződés felé, akkor üzenetet küld az autónak, hogy álljon meg. Ellenkező esetben értesíti, hogy áthaladhat a kereszteződésen. A vezérlés a blokklánctól indul a digitális iker felé. A digitális iker a kommunikációs köztesrétegen keresztül értesíti az autót.

A rendszerünkben szerepel egy hatóság is, amely felügyeli az okoskereszteződés megfelelő működését. A hatóság hozzáfér az autó és a kereszteződés blokklánc okosszerződéseihez

kapcsolódó adatokhoz, és bármilyen baleset vagy rendszerhiba esetén fel tudja használni a vizsgálathoz szükséges adatok hiteles példányát.

4.3. Adatok modellezése

Ebben az alfejezetben bemutatjuk az esettanulmányban szereplő rendszer adatainak modellezését.

A megoldásunk működéséhez szükséges a rendszer adatmodelljeinek elkészítése. A Vorto magas szintű modellezési nyelvet használva, az alábbi kódrészletekkel szemléltetett modellek készültek el.

Az autó modellje tartalmazza a járműveket egyértelműen azonosító rendszámtábla adatot. Ez az az egyedi azonosító, ami által megkülönböztethetőek a kereszteződéshez érkező járművek. Mivel ez autónként egy statikus metaadat, ezért a configuration kategóriába tartozik.

A stopping azt jelzi, hogy megállt-e az autó az utasításra. A timestamp a szenzoradatok mérésének időpontját jelzi. A speed az autó adott mérési időpontban mért sebessége. Ezek a status kategóriába tartoznak, mert az autó egy pillanatnyi állapotát írják le.

Továbbá az autó modellje tartalmazza az autón végrehajtható műveleteket. Az autó áthaladása a go() művelettel engedélyezhető, illete a stop() művelettel tiltható meg. Ezek az operations kategóriába tartoznak, mivel az autón végrehajtható műveleteket írják le.

```
vortolang 1.0
namespace org.eclipse.ditto.tdk
version 2.0
displayname "CarMetrics"
description "CarMetrics"

functionblock Car {
    configuration{
        licensePlate as String
    }
    status {
        stopping as boolean
        timestamp as long
        speed as double
    }
    operations{
        stop()
        go()
    }
}
```

A kereszteződés modellje tárolja azt, hogy éppen érkezik-e autó vagy vonat a carDetected és a trainDetected adattagokban. A temperature és humidity adattag a kereszteződésnél lévő hőmérséklet és eső szenzorok által mért adatokat tárolják. A timestamp adattag pedig ezeknek a méréseknek az időpontját tartalmazzák. Ezek mind a status kategóriába tartoznak, mivel gyakran frissülő adattagokat jelölnek.

Továbbá a modell tartalmazza a kereszteződés által észlelt eseményeket. Észleli, ha vonat érkezik a kereszteződéshez, illetve ha vonat hagyja el a kereszteződést. Ezeket a trainDetected() és trainLeft() események jelzik. Hasonlóan az autó érkezését és távozását is észleli. Ezeket a carDetected() és carLeft() események jelzik. Az előbb említett négy adattag az events kategóriába tartozik, mivel az eszköz által észlelt eseményeket jelzik.

```
vortolang 1.0
namespace org.eclipse.ditto.tdk
version 2.0
displayname "CrossingMetrics"
description "CrossingMetrics"

functionblock Crossing {
    configuration{
```

```
    }
    status {
        carDetected as boolean
        trainDetected as boolean
        timestamp as long
        temperature as double
        humidity as double
    }
    event{
        carDetected()
        carLeft()
        trainDetected()
        trainLeft()
    }
}
```

4.4. Digitális iker és kommunikáció integrálásának támogatása kódgenerálással

Ebben az alfejezetben bemutatjuk, hogy az esettanulmányunkban a kódgenerátorok mily módon támogatják a digitális iker reprezentáció integrálását a kommunikációs köztesréteghez.

A digitális iker konfigurálásához az Eclipse Ditto-ban használt Thing struktúrák generálására van szükség. Az autónak a Vorto modellból generált Thing struktúrája a következő kódrészletben látható.

```
{
    "thingId":"org.eclipse.ditto.tdk:Car",
      "attributes":{
        "licenseplate":{
          "properties":{
            "value":licenseplate
           }
        }
      },
      "features":{
        "stopping":{
          "properties":{
            "value":null
          }
        },
        "timestamp":{
          "properties":{
            "value":timestamp
          }
        },
        "speed":{
          "properties":{
            "value":speed
          }
        }
     }
}
```

A kereszteződésnek a Vorto modellből generált Thing struktúrája pedig a következő kódrészletben látható.

```
{
    "thingId":"org.eclipse.ditto.tdk:Crossing",
      "features":{
        "traindetected":{
          "properties":{
            "value":traindetected
          }
        },
        "cardetected":{
```

```
          "properties":{
            "value":cardetected
          }
        },
        "temperature":{
          "properties":{
            "value":temperature
          }
        },
        "humidity":{
          "properties":{
            "value":humidity
          }
        },
      }
}
```

A kommunikáció konfigurációjához egy XML fájlt generált a megoldásunk, ami a következő DDS komponensek beállításait tartalmazza:

- DDS Típusok
- DDS Topic-ok
- DDS DataReader-ek
- DDS DataWriter-ek

Az autó DDS Típusa:

```
<struct name="CarType">
  <member name="stopping" type="boolean"/>
  <member name="timestamp" type="long"/>
  <member name="licenseplate" stringMaxLength="128" type="string"/>
  <member name="speed" type="double"/>
</struct>
```

A kereszteződés DDS Típusa:

```
<struct name="CrossingType">
  <member name="traindetected" type="boolean"/>
  <member name="cardetected" type="boolean"/>
  <member name="temperature" type="double"/>
  <member name="humidity" type="double"/>
</struct>
```

A beavatkozó jelek Message típusú üzenetként érkeznek az eszközökhöz. A Message Topic-on keresztül tudunk beavatkozni az eszközök működésébe.

```
<struct name="MessageType">
  <member name="Message" stringMaxLength="128" type="string"/>
</struct>
```

A DDS DataWriter XML beállításai:

```
<domain_participant domain_ref="MyDomainLibrary::MyDomain" name="MyPubParticipant">
  <publisher name="MyPublisher">
    <data_writer name="MyCarWriter" topic_ref="Car"/>
    <data_writer name="MyCrossingWriter" topic_ref="Crossing"/>
    <data_writer name="MyMessageWriter" topic_ref="Message"/>
  </publisher>
</domain_participant>
```

4.5. Állapottérkép modellezés

Mint korábban bemutattuk, a kereszteződést és a résztvevőket vezérlő logikát modellvezérelten tervezzük meg az általunk nyújtott eszköztámogatás segítségével. Az elkészült

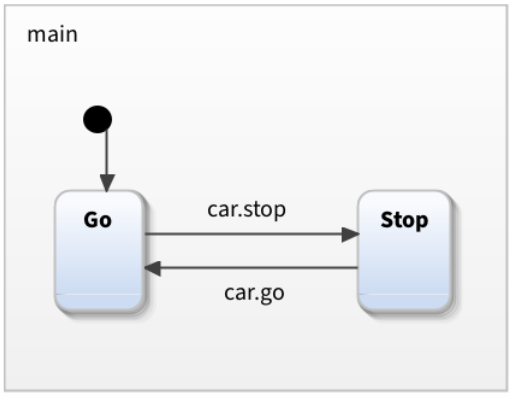
modellekből pedig az általunk fejlesztett automatikus kódgenerátorok segítségével futtatható okosszerződéseket szintetizálunk.

Ebben az alfejezetben bemutatjuk az esettanulmányhoz készített állapottérkép modelleket. A rendszerhez két Yakindu állapottérkép modell tartozik: a 4.2. ábrán az autó modellje, a 4.3. ábrán pedig a kereszteződés modellje látható. A modellek felhasználják a Vorto leírásból generált interfész leírásokat.

4.5.1. Az autó modellezése

Az autó 4.2. ábrán látható állapottérképe 2 állapotból áll, ezek jelentése a következő:

- **Go**: Az autó áthaladhat a kereszteződésen.
- **Stop**: Az autónak meg kell állnia.



4.2. ábra. Az autó állapottérkép modellje

Az állapottérkép a **car** interfészen érkező **stop** és **go** események hatására vált állapotot. Az egyes események jelentését a 4.1. táblázat tartalmazza.

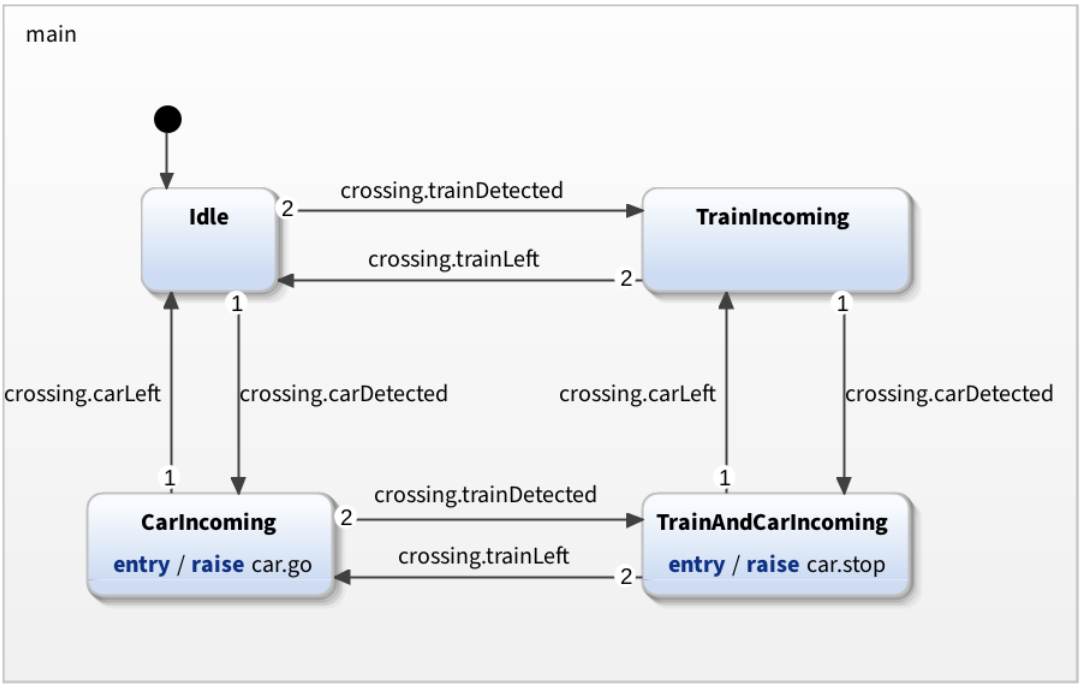
Esemény	Esemény jelentése
stop	az autónak meg kell állnia
go	az autó áthaladhat

4.1. táblázat. Az autó állapottérképének **car** interfészén érkező események

4.5.2. A kereszteződés modellezése

A kereszteződés 4.3. ábrán látható állapottérképe 4 állapotból áll, ezek jelentése a következő:

- **Idle**: A kereszteződés nem észlelte vonat vagy autó érkezését.
- **TrainIncoming**: A kereszteződés vonat érkezését észlelte.
- **CarIncoming**: A kereszteződés autó érkezését észlelte.
- **TrainAndCarIncoming**: A kereszteződés vonat és autó egyidejű érkezését észlelte.



4.3. ábra. A kereszteződés állapottérkép modellje

Az állapottérkép a **crossing** interfészen érkező események hatására vált állapotot. Az egyes események jelentését a 4.2. táblázat tartalmazza.

Esemény	Esemény jelentése
carDetected	autó érkezik a kereszteződésbe
carLeft	az autó elhagyta a kereszteződést
trainDetected	vonat érkezik a kereszteződésbe
trainLeft	a vonat elhagyta a kereszteződést

4.2. táblázat. A kereszteződés állapottérképének **crossing** interfészén érkező események

A kereszteződés állapottérképe nemcsak bejövő eseményeket dolgoz fel, hanem maga is kibocsát eseményeket a **car** interfészén. A kibocsátott események jelentését a 4.3. táblázat tartalmazza.

Esemény	Esemény jelentése
stop	az érkező autónak meg kell állnia
go	az érkező autó áthaladhat

4.3. táblázat. A kereszteződés állapottérképének **car** interfészén kibocsátott események

4.6. DAML kód generálása

Ebben az alfejezetben bemutatjuk a DAML kódgenerátor használatát, illetve az autó állapottérképéből előállított kódrészletekkel illusztráljuk a kódgenerátor működését.

Generátor modell

A Yakindu állapottérképekből történő kódgeneráláshoz szükséges egy úgynevezett SGen generátor modell létrehozása. Az autó állapottérképéből történő kódgeneráláshoz például az alábbi generátor modell használható:

```
GeneratorModel for yakindu::generic {
    statechart Car {
      feature Outlet {
          targetProject = "TDKCaseStudy"
          targetFolder = "src-gen"
          libraryTargetFolder = "src"
      }
      feature Generator {
          generatorProject = "hu.bme.mit.sct.generator.daml"
          generatorClass = "hu.bme.mit.sct.generator.daml.DamlGenerator"
      }
    }
}
```

A kódgeneráláshoz meg kell adni a kódgenerátort tartalmazó csomagot (generatorProject paraméter), valamint magát a kódgenerátor osztályt (generatorClass paraméter).

Példa kódrészletek

A következőkben néhány DAML nyelvű kódrészletet ismertetünk.

A lehetséges állapotok listájához (**Go**, **Stop**) egy enum típus tartozik:

```
data CarState
  = Main_Go
  | Main_Stop
  deriving (Eq, Show)
```

A generált okosszerződés képes kezelni a bejövő eseményeket. Például a **car** interfészen érkező **stop** esemény kezeléséhez az alábbi kódrészlet tartozik:

```
nonconsuming Event_car_stop : CarId
  do
    if (Main_Go `elem` currentStates) then do
      exercise self Transition626793910
    else
      return self
```

A generált okosszerződés képes állapotátmenetek végrehajtására is. Például a **Go** állapotból a **Stop** állapotba vezető átmenethez az alábbi kódrészlet tartozik:

```
Transition626793910 : CarId
  do
    contractId <- create this with
      currentStates = Main_Stop :: (delete Main_Go currentStates)
    return contractId
```

5. fejezet

Összefoglaló

Kritikus kiberfizikai rendszerek (CPS) az élet egyre több területén fontos szerepet töltenek be, a fizikai világból származó információt megfelelően feldolgozva kritikus döntéseket hozhatnak meg. Nagy mértékben támaszkodunk ezen rendszerek helyes működésére, ami azt jelenti, hogy egyrészt olyan komponenseket kell használnunk, amelyek teljesítik a szolgáltatásbiztonsági, teljesítmény és funkcionális követelményeket. Kritikus döntéseket bízunk ezen kiberfizikai rendszerekre, ezen okból a döntések nyomonkövethetősége is fontos tervezési szempont.

Annak érdekében, hogy megoldjuk a fentebb vázolt problémákat és megfeleljünk a modern kiberfizikai rendszerekkel szemben támasztott követelményeknek, munkánk során egy új CPS architektúrát mutatunk be, amely:

- A DDS szabványos kommunikációs köztesrétegre támaszkodva közel valós időben képes az adatokat és beavatkozást továbbítani. Emellett a DDS köztesrétegre támaszkodva szolgáltatásbiztos megoldást nyújt.
- Digitális iker technológiát integrál a DDS fölé, ezáltal lehetővé téve a rendszer állapotának követését és a megfelelő reakciók, beavatkozások számítását. A digitális ikerben tárolt információkat szabványos módon elérhetővé tesszük a külvilág és a felhasználók számára.
- Lehetővé tesszük reaktív vezérlők integrálását a digitális ikerpáron keresztül a rendszerünkbe, így megvalósítva a rendszer vezérlését.
- Blokklánc technológiát integrálunk a digitális ikerpárhoz, így lehetővé téve az adatok és döntések elosztott tárolását és nyomonkövethetőségét. A blokklánc technológia segítségével a rendszerben történő döntések és beavatkozások letagadhatatlanul eltárolódnak minden résztvevőnél, így lehetővé téve a hatóságok és külső résztvevők integrálását a rendszerbe egy egységes platformon keresztül.

Az új CPS architektúra megtervezéséhez és az implementáció automatikus származtatásának támogatására modellező eszközöket és kódgenerátorokat integráltunk a folyamatba. A megközelítésünk alkalmazásának a támogatására az alábbi műveleteket és lépéseket integráltuk:

- Szabványos modellező nyelvet integráltunk a tervezési folyamatba a rendszerben közlekedő adatok modellezésére. A megtervezett adatmodellekből automatikusan származtathatóak a szenzor beolvasó kódok, a kommunikációs köztesréteg konfigurációja és a digitális ikerpár váza is. A modellezés támogatja a "Single Source of Truth" (SSOT) paradigmát azáltal, hogy a rendszer különböző komponenseit egy közös modell alapján konfiguráljuk.

- Automatikusan az adatmodellből származtatjuk a blokklánc illesztést és a megfelelő interfészek létrehozását is, amely által a blokklánc megközelítés előnyei könnyen integrálhatóvá válnak a rendszerünkben.

- Modellezési támogatást nyújtunk a blokkláncon futó okosszerződésalapú vezérlés modellalapú tervezéséhez. A magas szintű modellezési nyelven megfogalmazott okosszerződés modellekből az általunk fejlesztett kódgenerátor segítségével az okosszerződések automatikusan előállíthatóak. A használt eszközök lehetővé teszik továbbá az okosszerződések magasszintű modelljeinek formális vizsgálatát, továbbá tesztgenerálással támogatják az okosszerződések kódjának tesztelését.

Az általunk kidolgozott CPS architektúra és a hozzá készített tervező és fejlesztő eszközök alkalmazhatóságát egy esettanulmány segítségével szemléltettük.

Munkánk folytatásaként tervezzük integrálni a különböző modellező nyelveket és kódgenerátorokat egy egységes tervező eszközbe. Így jobban ki tudnánk használni a szinergiákat a lépések között. Emellett ez lehetővé tenné, hogy a rendszer automatikus szintetizációja során optimalizáljuk a rendszer működését szolgáltatásbiztonsági, teljesítmény vagy extra-funkcionális szempontokból. Az alkalmazhatóság növelésének érdekében érdemes lenne bővíteni a támogatott CPS komponensek körét is.

Köszönetnyilvánítás

A dolgozatban ismertetett eredmények a Budapesti Műszaki és Gazdaságtudományi Egyetem Villamosmérnöki és Informatikai Kar Balatonfüredi Hallgatói Kutatócsoport szakmai közössége keretében jöttek létre a régió gazdasági fejlődésének elősegítése érdekében. Az eredmények létrehozása során figyelembe vettük a balatonfüredi központú Rendszertudományi Innovációs Klaszter által megfogalmazott célkitűzéseket, valamint a párhuzamosan megvalósuló EFOP 4.2.1-16-2017-00021 pályázat támogatásával elnyert „BME Balatonfüredi Tudáscentrum" térségfejlesztési terveit.

Irodalomjegyzék

[1] Cyber-Physical Systems - a Concept Map. https://ptolemy.berkeley.edu/projects/cps/ [Hozzáférés dátuma: 2020.10.28].

[2] DAML SDK Documentation - Glossary of concepts. https://docs.daml.com/concepts/glossary.html [Hozzáférés dátuma: 2020.10.28].

[3] Dylan Yaga, Peter Mell, Nik Roby, Karen Scarfone: Blockchain Technology Overview. Jelentés, 2018, National Institute of Standards and Technology. https://doi.org/10.6028/NIST.IR.8202.

[4] Eclipse Ditto - Basic concepts overview. https://www.eclipse.org/ditto/1.3/basic-overview.html [Hozzáférés dátuma: 2020.10.28].

[5] Michel Rauchs – Andrew Glidden – Brian Gordon – Gina Pieters – Martino Recanatini – François Rostand – Kathryn Vagneur – Bryan Zhang: Distributed Ledger Technology Systems: A Conceptual Framework. Jelentés, 2018, Cambridge Centre for Alternative Finance. https://www.jbs.cam.ac.uk/faculty-research/centres/alternative-finance/publications/distributed-ledger-technology-systems/ [Hozzáférés dátuma: 2020.10.28].

[6] Nick Szabo: Smart Contracts, 1994. http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html [Hozzáférés dátuma: 2020.10.28].