Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

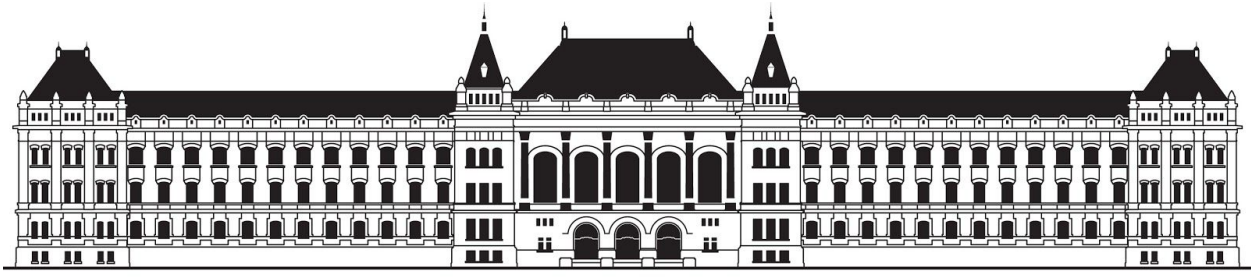
Házi feladat dokumentáció

Intelligens Elosztott Rendszerek

2018 Tavasz

Kovács-Egri Kristóf
BB85QA

Zsiga Tibor
L04P9O



Tartalomjegyzék

Feladat leírás

Jelen házi feladatunkban szerverek, szerverszobák teljeskörű, komplex biztonsági védelmét szeretnénk támogatni különböző intelligens, együttműködő ágensek segítségével. A rendszer képes jelezni az illetéktelen behatolást, valamint magas füst és tűz esetén megtenni a szükséges óvintézkedéseket a szerverek és adatok biztonságának megóvása érdekében.

Négy szerverszoba szimultán működését modelleztük, amelyek minden biztonsági és védelmi feladatra reagálnak és ezt kommunikálják mester ágens felé, hogy megtörténhessenek a szükséges óvintézkedések.
Ha füstöt, vagy magas hőmérsékletet érzékel bármely szoba ágense, akkor azt jelzi a mester ágensnek, amely jelzi a tűzoltók felé az esetet, lezárja a szoba ajtajait, hogy a többi szerverszobába ne terjedjen át a tűz, illetve a szervereken tárolt adatokról biztonsági mentést kezd készíteni egy biztonsági felhőbe. Amikor ez kész van, akkor leállítja a szobában lévő szervereket a tűz elhárításának a végéig.
Ha valaki belép a szerverszobákba bekapcsolt riasztó esetén, akkor a mozgásérzékelő ezt észleli és egy 10 másodperces időlimitet hagy arra, hogy az illető a biztonsági kódját beírja és ezzel megelőzhesse a riasztó jelzését, ellenben jelez a mester ágens felé, mely elindítja az biztonsági protokollt, amely során az ajtók lezárulnak és a rendőrségnek jelzi az esetet a rendszer.

A rendszer a következő bemenetekkel (szenzorokkal) rendelkezik:

- Füstérzékelő
- Mozgásérzékelő
- Hőmérő
- Riasztó kezelő panel

Ezeknek az adatait egy egyszerű grafikus kezelőfelületen lehet irányítani és megadni, ahol ezek alapján egyből kapunk visszajelzést az épp végrehajtott intézkedésekről és minden szoba jelenlegi védelmi és biztonsági státuszáról. Minden történést és intézkedést azonnal megjelenítünk a konzolban.

Egy adott szerverszobához tartozó szenzor mért adatait a szobához tartozó védelmi és biztonsági ágens feldolgozza, és ha bármilyen jellegű intézkedésre szükség van, akkor ezt közlik a master nevű ágenssel, amely elvégzi a szükséges külső beavatkozásokat.

Megoldás

A rendszer az alábbi ágensekkel rendelkezik:

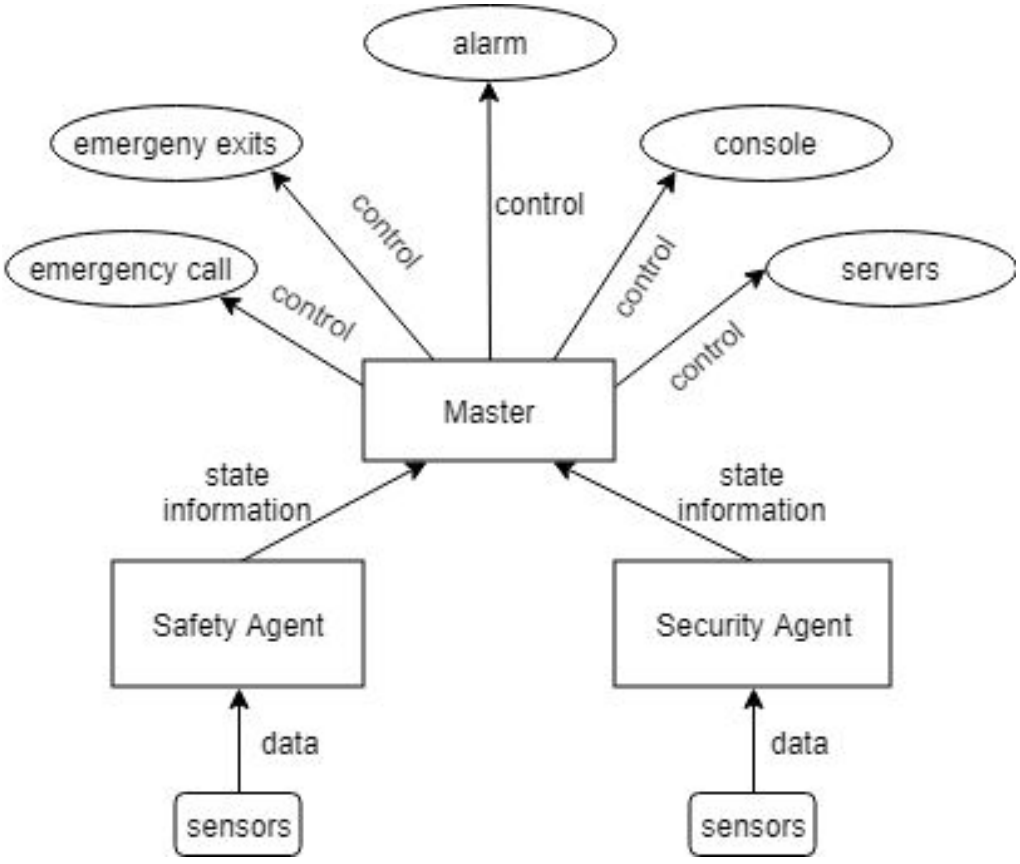
- Biztonsági Rendszer (Security Agent)
- Védelmi rendszer (Safety Agent)
- Master

Ágensek funkcionalitása

- **Biztonsági rendszer (Security Agent)**
  - Figyeli a riasztó állapotát (be- vagy kikapcsolt)
  - Figyeli, hogy van-e mozgás a szerverszobában
  - Figyeli és ellenőrzi a beírt pinkódot bekapcsolt riasztó és mozgás esetén
  - Ha a kódot nem sikerül beadni 10 másodpercen belül, akkor jelez a masternek
- **Védelmi ágens (Safety Agent)**
  - Figyeljük, ha füstöt és magas hőmérsékletet érzékelünk, és ha egy adott szobában ez a kettő egyszerre következik be, akkor jelzünk a masternek
- **Mester ágens (Master agent)**
  - Megkapja a Safety és Security ágensektől, ha valami nem megfelelő védelmi vagy biztonsági esetet észlelnek és erre reagálni tud az adott szobában az ajtók nyitásával vagy zárással, rendőrség és tűzoltók értesítésével, adatok kimentésének kezdeményezésével, szerverek leállításával és újraindításával

Minden szerverszoba rendelkezik egy biztonsági és egy védelmi ágenssel. Ezek azok, amikhez az adott szoba szenzoraitól (jelen esetben ezek a GUI-n vannak szimulálva) beérkeznek az adatok. Ők eldöntik a szenzoradatok alapján, hogy az adott szobában van-e valami nem megfelelő vagy nem elfogadható eset. Ha ilyen történik, akkor ezt jelzik a master ágensnek, amely reagál ezekre az esetekre és elindítja a szükséges protokollok szerinti óvintézkedéseket. Ha a hiba elhárításra kerül, akkor azt a safety és security szenzor észleli újra, majd a master ágens az, amely reagál ezekre is és újra helyreállítja a rendszer működését.

A rendszer összefoglaló ábrája



Fejlesztés

A jelenlegi legfrissebb Jason verziót használtuk, melyet sourceforge-ról le lehetett tölteni, ez a jason-2.2a volt.

A github-on és különböző interneten lévő doksikból informálódtunk és értettük meg az asl nyelv szintaktikáját és a Jason használatát.
Szerencsére a letöltéskor a mappában elég sok hasznos példa is csatolva volt, amelyeket tanulmányozva a struktúrát és a kódot megértve már mi is el tudtuk kezdeni az implementálást.

Jason és ASL szinten a három ágenstípust dolgoztuk ki, amelyek felelősek azért, hogy eldöntsék a beérkező szenzoradatokból, hogy ez kritikus vagy normális esetnek minősül-e.
A beérkező szenzor adatokat egy refresh függvényben kapjuk meg a Java GUI felületről. Ezeket szobánként vizsgálja a szobához tartozó biztonsági és védelmi ágens, majd egy hiedelmet felállítva az esetről szól a master ágensnek, amely lekezeli az adott esetet és a szükséges óvintézkedéseket illetve a helyreállítást megteszi.
Itt ahhoz, hogy szeparálni tudjuk a szobákhoz tartozó védelmi és biztonsági ágenseket, minden ágensből készítettünk egy példányt, amely egy adott szobának az állapotáért felel és ehhez határoz meg hiedelmeket, melyeket a masternek továbbküld.
A smartServerRoom.mas2j fogja össze az asl-ben írt ágenseket és a Java environmentet, majd ő indítja el ezeknek a működését. Itt beállítottuk még, hogy centralizált infrastruktúrával rendelkezünk.

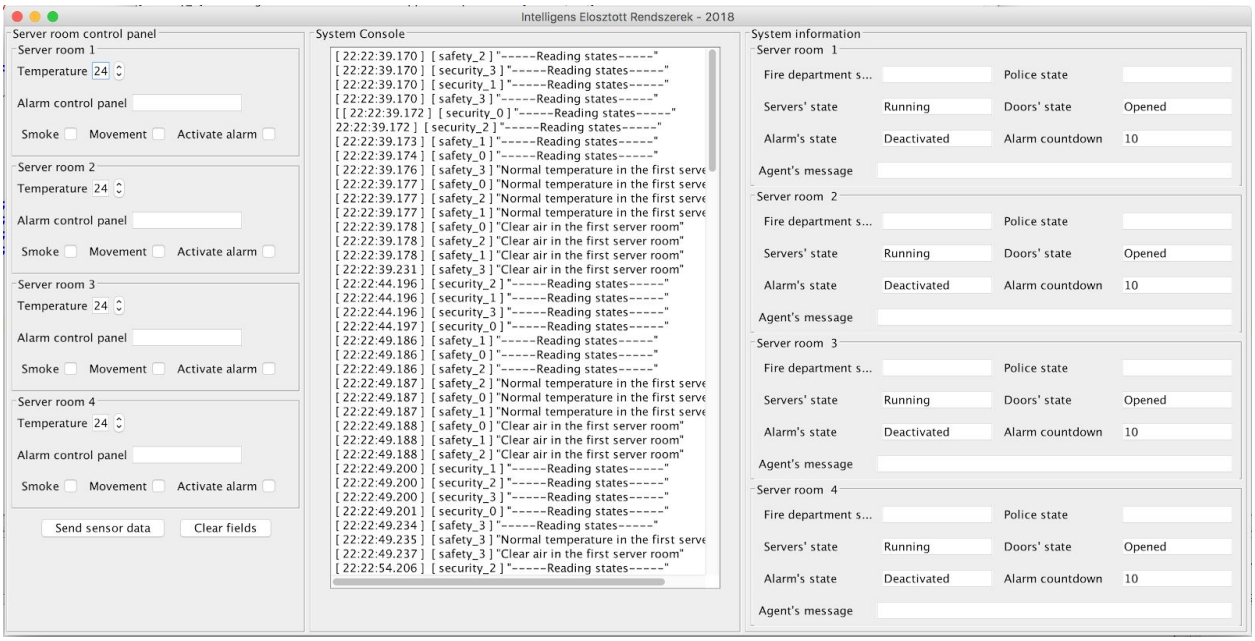
Java szinten egy grafikus felületet állítunk fel, mely szépen szeparálva és modulokra bontva került implementálásra az objektumorientáltság elveinek megfelelően.
Itt valósítottuk meg a szobákhoz tartozó szenzorok bemeneteit TextFieldek és Checkboxok használatával, emellett egy konzolfelületet is, ahova az ágensek logolják az épp folyamatban lévő státuszüzeneteket, emellett megvalósítottunk még egy felületet, ahol szobánként tudjuk megjelenítani Label-ekben az adott szobák éppeni státuszát.
Szükségünk volt még egy Timer-re is, amely az élesített riasztó visszaszámlálásának a megjelenítésére szolgál. Ennek is a master ágens jelez, hogy mikor kezdje el a számolást, majd jelez ha le kell állítani, vagy reset-elni kell.

Ugyancsak említésre méltó még az Env.java nevű file, ami gyakorlatilag összeköti a grafikus felületet az asl-ben megírt ágensekkel. Ez segít a Java és Jason

kommunikációjában, ez értelmezi és parse-olja a függvényeket és minden kommunikált és átadott paramétert.

Az elkészült program ismertetése

Az elkészült program működését bemutató videó [ezen a linken található.](https://photos.app.goo.gl/kiZzSMo96NazNaD9A)[1]
A program felhasználói felülete tartalmazza a szenzor adatok szimulálására szolgáló felületet, amin a szerverszobák állapotát irányíthatjuk. Emellett a mellette megjelenő konzolon és a szöveges felületeken láthatjuk külön-külön a szobák jelenlegi állapotát.



A felületen lehet állítani szobánként a szoba hőmérsékletét, azt, hogy épp van-e füst a szobában, a riasztó ki- vagy bekapcsolt állapotban van és érzékelünk-e épp mozgást.
Emellett egy beviteli mező is rendelkezésünkre áll, ahol a szobákhoz tartozó riasztó PIN kódját tudjuk megadni.
A felületen az adatok bevitele után a “Send sensor data” gombra való kattintáskor küldjük el a rendszernek az új, frissített szenzor adatokat, melyekre a rendszer reagál a megfelelő módon, és ezeket a reakciókat láthatjuk középen a konzolon és a jobboldali oszlopban megjelenített státuszban szobánként. Itt láthatjuk az ajtók és szerverek állapotát, emellett az ágens kritikus esetben küldött üzeneteit, és hogy a rendőrség és tűzoltóság hívásának státusza épp milyen állapotban van.
Lehetőségünk van még visszaállítani a szenzor adatokat a kezdeti állapotra a “Clear data” gombra kattintva.

[1] https://photos.app.goo.gl/kiZzSMo96NazNaD9A

Próbáltunk a konzolon minél informatívabban minden eseménynél kiírni egy üzenetet, hogy lássa a felhasználó, vagy az “admin”, aki épp követi a szerverszobák állapotát, hogy éppen mi történik és erről egy tiszta képet kapjon.

Az egyes ágensprogramok rövid összefoglalása

Az egyes safety és security ágensek működése megegyezik, így az első szobához tartozó safety és security ágensek működését ismertetjük.

Safety_0 ágens

Szenzor adatok frissítésekor a következő ellenőrzések futnak le:

```
+!run : refresh(A,B) <- print("-----Reading states-----");
        !checkTemperature(A);
        !checkSmoke(B);
        !!end.
```

A meghívott két függvény:

```
+!checkTemperature(N) : N > 45 <- +hightemperature;
        print("High temperature in the first server room").
+!checkTemperature(N) : N <= 45 <-  -hightemperature;
        print("Normal temperature in the first server room").

+!checkSmoke(N) : N == true <- +hassmoke;
        print("Smoke detected in the first server room").
+!checkSmoke(N) : N == false <- -hassmoke;
        print("Clear air in the first server room").
```

A Masternek küldött információk pedig:

```
+!end : (hightemperature & hassmoke) <- .send(master,tell,firealert_0);
        .wait(5000); !!start.
+!end  <- .send(master,untell,firealert_0);
        .wait(10000); !!start.
```

Security_0 ágens

Szenzor adatok frissítésekor a következő ellenőrzések futnak le:

```
+!run : refresh(A,B,C) <-
        print("-----Reading states-----");
        !checkAlarm(A);
        !checkMovement(B);
        !checkAlarmAndMovement(C);
        !!end.
```

A meghívott függvények:
Az “alarmOn” hiedelmet csak akkor veszi be a tudásbázisába, ha nincs be van kapcsolva a riasztó és a PIN kód nincs beírva / nem jó.

```
+!checkAlarm(N) : N == true & not correctPin <- +alarmOn.
+!checkAlarm(N) : N == false <- -alarmOn.
```

Az “alarmOn” hiedelem felvételekor bekapcsoljuk a riasztót:

```
+alarmOn <- activateAlarm(0);
        print("The alarm is turned on in the first server room").
```

Amikor az “alarmOn” hiedelem kikerül a tudásbázisból, leállítjuk a számlálót, kikapcsoljuk a riasztót, visszaállítjuk a számálót az eredeti értékére, és értestjük a master ágenst arról, hogy nem érzékel betörést.

```
-alarmOn <- stopAlarmCounter(0);
        deactivateAlarm(0);
        .wait(500);
        resetAlarmCounter(0);
        print("The alarm is turned off in the first server room");
        .send(master, untell, intrude_0);
        -alarmReachedZero.
```

MovementDetected hiedelem, ha mozgást érzékelnek a szenzorok:

```
+!checkMovement(N) : N == true <- +movementDetected.
+!checkMovement(N) : N == false <- -movementDetected.
```

Leellenőrzi a beütött PIN kódot:

```
+!checkAlarmPin(N) : N == 1234 <- +correctPin.
+!checkAlarmPin(N) : N \== 1234 <- -correctPin; print("Invalid pin code entered for alarm in
the first server room").
```

Ha be van kapcsolva a riasztó és mozgást érzékeltünk, megvizsgáljuk a PIN kódot, ilyenkor másodpercenként ismételjük a vizsgálatot:

```
+!checkAlarmAndMovement(X): (alarmOn & movementDetected) <- !checkAlarmPin(X);
.wait(1000); !!start.
```

Egyébkén 5 másodpercenként ellenőrizzük a szenzorokat:

```
+!checkAlarmAndMovement(X) <-.wait(5000); !!start.
```

Amennyiben a riasztó be van kapcsolva, mozgást érzékeltünk és nem jó a PIN kód, felvesszük a “timerStarted” hiedelmet:

```
+!end : (alarmOn & movementDetected & not correctPin) <- +timerStarted.
```

A timerStarted hiedelem felvételekor az ágens elindítja a számlálót, majd ha lejárt, az “alarmReachedZero” hiedelemmel bővíti tudáshalmazát:

```
+timerStarted <-
        resetAlarmCounter(0);
        startAlarmCounter(0);
        .wait(10000);
        stopAlarmCounter(0);
        +alarmReachedZero;
        .wait(20000);
        -timerStarted;
        !!start.
```

Amennyiben a riasztó számlálója elérte a nullát, be van kapcsolva még a riasztó és még mindig nem írták be a jó PIN kódot, értesíti a master ágenst a betörésről:

```
+alarmReachedZero: (not correctPin & alarmOn) <- .send(master, tell, intrude_0).
```

Ha beírják a jó PIN kódot, megállítjuk a számlálót, kikapcsoljuk a riasztót és értesítjük erről a master ágenst.

```
+correctPin <- stopAlarmCounter(0);
    -alarmOn;
    .send(master, untell, intrude_0);
    -correctPin.
```

A master ágens

Amennyiben a safety_0 ágenstől “+firealert_0” üzenet érkezik (tell), a specifikációban leírt funkciókat hajtja végre (tűzoltók értesítése, adatok mentése, biztonsági ajtók kinyitása, szerverek leállítása):

```
+firealert_0[source(A)]: true <-
        print("-----Reading states-----");
        print("Starting protocol against fire and smoke in first server room");
        callFireFighters(0, "In progress...");
        print("Saving data to the cloud...");
        emDataSave(0, start);
        print("Opening emergency exits...");
        emExit(0, open);
        print("Emergency exits are opened.");
        emSpk(0, "There is fire in the building!
                        Please follow the exit instructions.");
        .wait(2500); //2,5 mp az adatmentesre eleg lesz
        callFireFighters(0, "Firefighters on their way!");
        emDataSave(0, stop);
        print("Data saving has been finished.");
        print("Shutting down the servers...");
        stopServers(0);
        .wait(1000);
        print("Successful server shutdown.").
```

Amennyiben a safety_0 ágenstől “-firealert_0” jelzés érkezik (untell), elindítja a szervereket, a rendszer állapota vissza áll a normál működésre:

```
-firealert_0[source(A)]: true <- print("Emergency protocols successful.");
        callFireFighters(0, "");
        emSpk(0," ");
        emExit(0, open);
        print("Restarting the servers...");
        .wait(2000);
        startServers(0);
        print("Servers are up and running.").
```

Amennyiben betörés történik az első szerverszobában (intrude_0, tell), értesíti a rendőrséget és bezárja az ajtókat:

```
+intrude_0[source(A)]: true <- print("-----Reading states-----");
        print("Starting protocol against intruder in first server room");
        callPolice(0, "In progress...");
        emExit(0, close);
        emSpk(0, "!!!Intrude in first server room!!!")
        .wait(1000);
        callPolice(0, "Police is on its way!").
```

Amennyiben elhárult a veszély (intrude_0, untell), visszaáll a rendszer állapota a normál működésre:

```
-intrude_0[source(A)]: true <- print("-----Reading states-----");
        callPolice(0, "");
        emExit(0, open);
        emSpk(0, " ").
```