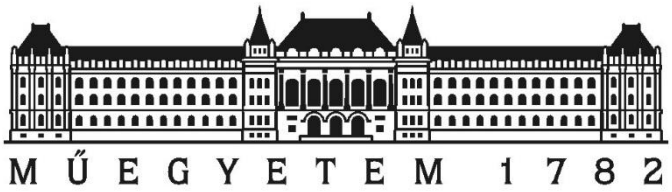


Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

Inline programozó állomás vezérlésének automatizálása

Készítette	*Belső Konzulens*	*Külső Konzulens*
Németh Dávid	Dr. Bank Balázs	Péczeli Viktor

2016

TARTALOMJEGYZÉK

HALLGATÓI NYILATKOZAT

Alulírott **Németh Dávid**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző, cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulensek neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2016. 05. 28.

.....

Németh Dávid

Összefoglaló

A felületszerelt gyártástechnológiát (SMT) széles körben alkalmazzák, mert számos előnye van a furatszerelt technológiával (THT) szemben. Az alkatrészek mérete jóval kisebb a furatszerelt alkatrészeknél, és a kivezetések elmaradásával a villamos paraméterek is javulnak. További előny, hogy az alkatrészek tokozása szabványosított, így egyszerűsödik a tárolás és az adagolás. Ezzel a technológiával a gyártási folyamatok automatizálhatók, és így olcsóbbá válnak.

A MES – Manufacturing Execution System – termelési folyamatokat felügyelő számítógépes rendszer, melynek lényege a termékek életciklusának követése, és a rajtuk végzett gyártási és megmunkálási folyamatok pontos dokumentálása. A MES elősegíti, hogy lehetőség szerint a gyártás minden egyes pontjában áttérhessünk a kézi adatbevitelről az automatizált módszerekre.

Szakdolgozatom célkitűzése az SMT sorokon található inline programozó (RoadRunner) átállási, és alkatrész ellenőrzési folyamatainak fejlesztése volt a SMED módszer támogatásával. A SMED módszer lényege, hogy minimálisra csökkentse a belső átállás folyamatait. Belső átállásnak nevezzük az átállás azon részeit, melyeket csak akkor lehet elvégezni, ha a gépet megállítjuk.

Az elkészített alkalmazás képes figyelemmel kísérni, hogy éppen milyen termék gyártásának kell a soron következnie. Ennek megfelelően tudja a RoadRunner munkamenetét leállítani, törölni, módosítani és elindítani. Az alkalmazás képes továbbá a gyártott termék MES rendszerbeli tesztelési adatainak megtekintésére, és a programozott alkatrész tartalmának ellenőrzésére egy referencia fájl alapján.

A végeredményként kialakult szoftver hozzájárul az emberi mulasztásból fakadó hibák minimalizálásához, és támogatja a termékek közötti gyors átállás megvalósulását.

Abstract

The use of Surface Mount Technology (SMT) is widespread, because it has several advantages over Through-Hole Technology (THT). The electronic components are much smaller and without the wire leads much better electrical properties can be achieved. SMT components also have standardized packaging, which makes their storage and distribution easier. With this technology the production can be automated, making it cheaper.

MES – Manufacturing Execution System – is a computer system that supervises the production process. Its purpose is to follow the lifecycle of the product and to document the production and elaboration processes that it goes through. MES helps make it possible to switch from manual data input to automated solutions at every step of the manufacturing process.

The goal of my thesis is to improve the changeover and component inspection process of the inline programmer (RoadRunner) used on the SMT production lines, with support of the SMED method. The SMED method's purpose is to minimize the internal changeover processes. Internal changeover is the part of the changeover that can only be performed with the stopping of the machine.

The application prototype is capable of monitoring which product should be next on the given SMT. Based on this information it can stop, delete, modify or start the RodeRunner's current session. Furthermore the application also allows the survey of the product's MES system test data and the examination of the programmed components based on a reference file.

The final software helps minimize the faults caused by human error and supports the fast changeover between products.

1. Bevezetés

A Continental konszern 2011-ben ünnepelte fennállásának 140. évfordulóját. Alapítása óta a hannoveri székhelyű vállalat gumiabroncs- és kaucsuk-specialistából a világ egyik vezető autóipari beszállítójává fejlődött.

A vállalat autóelektronikai részegységek és mikroelektronikai áramköri modulok gyártását végzi a járműelektronika szinte minden területén, a Continental Chassis & Safety (Futómű és Biztonsági Egységek), Powertrain (Hajtómű) és Interior (Belső tér) divíziói, valamint a legfőbb autógyárak számára világszerte.

A vállalat piaci pozíciójának megőrzése érdekében fontos, hogy megfeleljen a piaci igényeknek, tehát alacsony költségszint mellett jó termékminőséget és minél alacsonyabb szállítási időt nyújtson vevői számára. Ennek biztosítása érdekében arra törekszik, hogy az emberi mulasztásból fakadó hibákat a minimálisra csökkentse úgy, hogy – ahol lehet – automatizálja folyamatait.

Feladatom a cég budapesti telephelyének felületszerelt gyártástechnológiát alkalmazó sorain használt inline programozóberendezés automatizálása, és a programozott termék megfelelőségének vizsgálata volt, standard szoftveres környezetben.

A dolgozat következő fejezetében bemutatom a megismert szerelési technológiák előnyeit és hátrányait, a technológia során alkalmazott főbb alkatrész és tokozási típusait, valamint a felületszerelt gyártástechnológia meghatározó folyamatait.

A 3. fejezetben bemutatom a Budapesten használt traceability rendszer felépítését, ismertetem az alapvető fogalmakat és a főbb modulok feladatát, majd egy valódi SMT sor segítségével szemléltetem a rendszer működését.

A 4. fejezetben először röviden ismertetem az egyes alkatrész-programozási technológiákat, valamint az egyes technológiák előnyeit és hátrányai. Ezek után bemutatom a kiírásban szereplő SMT beültető gépekre szerelhető inline alkatrész-programozó és –kezelő berendezést, a RoadRunnert. A fejezet későbbi szakaszaiban pedig a feladatmegoldás során alkalmazott automatizálást támogató webszolgáltatás főbb funkcióit mutatom be.

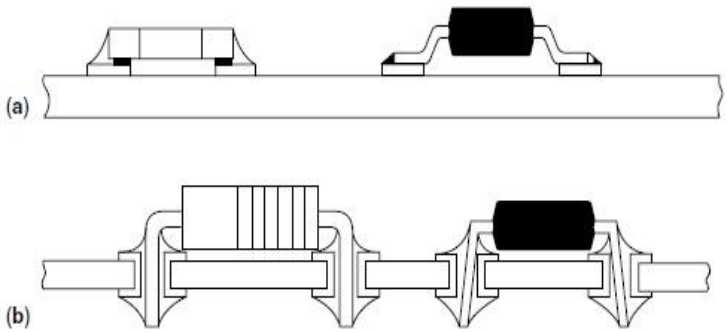
Az 5. fejezet tartalmazza a feladatom megvalósítását. Először ismertetem a munkafolyamat korábbi lefolyását, és az ezzel kapcsolatosan felmerülő problémákat. Majd bemutatom a vezérlési folyamatot megvalósító funkciók működését, a szoftver konfigurációs lehetőségeit, a program felépítését és az alkalmazott standard szoftveres környezetet. Végül megmutatom az elkészült program felhasználói felületét.

A 6. fejezetben az elkészített szoftverek értékelésére, valamint néhány fejlesztési lehetőség bemutatására kerül sor.

2. A felületszerelt gyártástechnológia

2.1. Szerelési technológiák

Az elektronikai szereléstechnológiában két szerelési módszert különböztetünk meg egymástól: a furatszerelési technológiát (THT –Through-Hole Technology) és a felületszerelési technológiát (SMT – Surface Mount Technology).



2.1. ábra. THT (a) és SMT (b) [2]

Az alkatrészek kivezetései lehetnek merev vagy hajlékony kialakításúak. A furatszerelési technológiánál ezeket a szerelőlemez furataiba helyezik, majd a lemez másik oldalán (tipikusan hullámforrasztás segítségével) elektromosan bekötik. Az előbbit alkatrészoldalnak, az utóbbit forrasztási oldalnak nevezzük.

A furatszerelési technológia egyre inkább háttérbe szorul. Napjainkban a beültetésre kerülő alkatrészek közel 95%-a SMD, és csupán 5% furatszerelt [3]. Hátránya, hogy az alkatrészek helyigénye nagy a furatok miatt, valamint az, hogy a szerelőlemez mindkét oldala felhasználásra kerül.

A felületszerelési technológiában nem különböztetünk meg alkatrész-, illetve forrasztási oldalt, mivel az alkatrészek ugyanazon az oldalon találhatók, ahol azok rögzítése történik a panelhez.

Az SMT-nek sok előnye van a furatszerelési technológiával szemben. Alkalmazása során nincs szükség furatokra, az alkatrészek mérete jóval kisebb a furatszerelt alkatrészeknél, és a kivezetések elmaradásával a villamos paraméterek is javulnak. További előny, hogy az alkatrészek tokozása szabványosított, így egyszerűsödik a tárolás és az adagolás. Ezzel a technológiával a gyártási folyamatok olcsóbbá válnak és automatizálhatók lesznek.

Sok előnye mellett a felületszerelt technológiának van néhány hátránya is. Az alkatrészszám és a méretcsökkenés miatt az SMT bonyolultabb tervezést igényel, és az alkatrészek beültetése rendkívül nagy pontosságot követel meg. A rengeteg alkatrész miatt a hibák keresése és feltárása jóval nehezebb, mint a furatszerelt technológiánál.

2.2. Szerelési változatok

Két szerelési változatot különböztethetünk meg egymástól: a tiszta felületi szerelést és a vegyes szerelést.

A tiszta felületi szerelés használja ki legjobban az SMT nyújtotta előnyöket. Ennél a szerelési változatnál csak felületre szerelhető alkatrészeket alkalmaznak, és az elektromos kötés megvalósítása forrasztópaszta alkalmazásával történik.

Vegyes szerelés esetén a felületre szerelhető alkatrészek mellett furatba szerelhető alkatrészeket is használnak. Az elektromos kötések megvalósítása itt hullámforrasztással történik. Ennél a technikánál megnő a technológiai folyamatok száma, de sokkal nagyobb alkatrész-szerelési sűrűség érhető el, mint tiszta felületi szerelés esetén.

Mindkét szerelési technológiára szükség van, hiszen bizonyos alkatrészeket nem lehet vagy nem érdemes felületszerelt alkatrészként megvalósítani. Ilyen alkatrészek például a nagy kapacitású kondenzátorok, teljesítménytranzisztorok, induktivitások, transzformátorok, mechanikai elemek, valamint a kapcsolók nagy része.

2.2.1. A tiszta felületi szerelés fajtái és technológiai lépései

A tiszta felületi szerelésnek két fajtáját különböztetjük meg annak függvényében, hogy a panelnak csak az egyik, vagy mindkét oldalára kerülnek alkatrészek.

Egyoldalas felületszerelés (Reflow) esetén három technológiai lépés követi egymást.

1. Forraszpaszta felvitele a hordozóra.
2. Felületre szerelhető alkatrészek beültetése.
3. Újraömlesztéses forrasztás a paszta kikeményítésére.

Kétoldalas felületszerelés (Dupla Reflow) esetén a folyamat már hét lépésből áll.

1. Forrasztópaszta felvitele a hordozó „A” oldalára.
2. Felületre szerelhető alkatrészek beültetése az „A” oldalra.
3. Újraömlesztéses forrasztás.
4. Panel fordítása.
5. Forrasztópaszta felvitele a hordozó „B” oldalára.
6. Felületre szerelhető alkatrészek beültetése a „B” oldalra.
7. Újraömlesztéses forrasztás.

2.2.2. A vegyes felületi szerelés fajtái és technológiai lépései

Vegyes felületi szerelés esetén is két technológiát különböztetünk meg egymástól. Az első esetben a panel egyik oldalára felületszerelt, a másik oldalára pedig furatszerelt alkatrészek kerülnek. A folyamat során hét lépés követi egymást.

1. Furatszerelt alkatrészek beültetése, mechanikai rögzítése.
2. Panel fordítása.

3. Ragasztó felvitele.
4. Felületre szerelhető alkatrészek beültetése.
5. Ragasztó kikeményítése kemencében.
6. Panel fordítása.
7. Hullámforrasztás, tisztítás.

A vegyes felületi szerelés második esete, amikor a panelnek van olyan oldala, amin felületszerelt és furatszerelt alkatrészek is megtalálhatók. A folyamat kilenc lépésből áll.

1. Forrasztópaszta felvitele az „A” oldalra.
2. Felületszerelt alkatrészek beültetése az „A” oldalon.
3. Újraömlesztéses forrasztás.
4. Huzalkivezetéses alkatrészek beültetése az „A” oldalon.
5. Panel fordítása.
6. Ragasztó felvitele.
7. Felületszerelt alkatrészek beültetése a „B” oldalon.
8. Ragasztó kikeményítése.
9. Panel fordítása, hullámforrasztás.

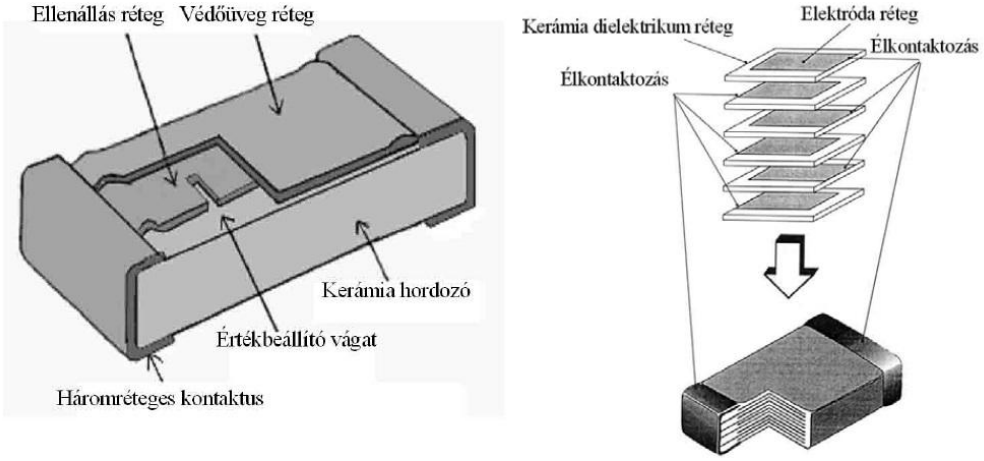
2.3. Felületre szerelhető alkatrészek

Az felületszerelt alkatrészek alapkövetelményei, hogy méretcsökkenést tudjunk elérni, valamint, hogy kialakításuk és alakjuk egységes legyen, ezzel segítve az automatizált beültetés folyamatát, és a villamos ellenőrzést. Már szinte minden alkatrész előállítható felületszerelt változatban, így nagyon sokféle kialakítás létezik. Érdemes azonban két csoportba sorolva megvizsgálni a legjellegzetesebbeket:

- Passzív elemek;
- Integrált áramkörök.

2.3.1. Passzív elemek

A passzív alkatrészek csoportjába az ellenállások, kondenzátorok, potenciométerek és induktivitások tartoznak. Az alkatrészek végein elhelyezkedő forrasztásra alkalmas fémezést használjuk a kötések kialakítására. Így tud az alkatrész mechanikailag és villamosan is kapcsolódni a szerelőlemezen kialakított pad-ekhez. A passzív SM alkatrészek közül az úgynevezett chip méretű diszkrét alkatrészek a legelterjedtebbek. Az ilyen típusú ellenállásokat leggyakrabban kerámia hordozón kialakított rétegellenállás formájában (2.2. ábra) valósítják meg. Értékük jellemzően néhány Ω-tól egészen 10 MΩ-os nagyságrendig változhat.

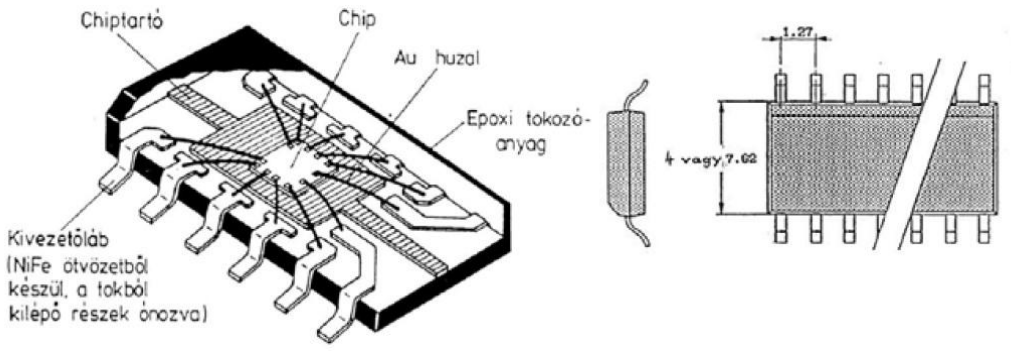


2.2. ábra Felületszerelt vastagréteg-ellenállás és a többrétegű kerámia blokk-kondenzátor felépítése [5]

A felületszerelhető kondenzátoroknak is több fajtája létezik, leggyakoribbak a kerámia blokk-kondenzátorok, melyeknek kialakítása hasáb alakú (2.2. ábra), valamint az elektrolit kondenzátorok, melyek jellemzően hengeres kialakításúak. Utóbbinál az alumínium ház tölti be a védőréteg szerepét, előbbinél kerámia. Meg kell említenünk még a tantál kondenzátorokat, melyek hasáb alakúak, szigetelőanyaguk tantál, védőrétegük műanyag ház. A chip kondenzátorok értéke pF-os nagyságrendtől néhány µF-ig változhat. Ezen értékek felett már jellemzően furatszerelt alkatrészeket használnak, mert felületszerelt kivitelben méretük annyira nagy lenne, hogy alkalmatlanná tenné őket technológia előnyeinek használatára.

2.3.2. Felületszerelt IC-k

Az első felületszerelt IC tokok az SO (Small Outline) tokok voltak (2.3. ábra). A kivezetések a tok hosszabbik oldala mentén egyenletesen helyezkednek el, a tipikus távolság közöttük 1.27 mm. A chip fém hordozólemezre van beültetve, kivezetései a tok lábaihoz mikrohuzalkötéssel (vékony aranyhuzallal) vannak kötve. A tok kivezetései úgynevezett sirályszárny (Gull Wing) formájúak.



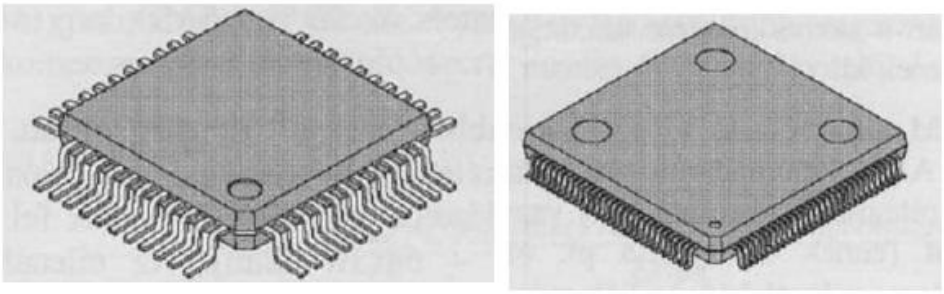
2.3. ábra. SO IC-k felépítése [5]

Kezdetben az SO tokok legfeljebb 28 kivezetéssel rendelkezhettek, majd a rasztertávolság fokozatos csökkenésének következtében akár 56 kivezetést is lehetővé tettek, ami így méretnövekedéssel sem járt.

A sirályszárny alakú kivezetések azonban könnyen deformálódtak, ezért kifejlesztették a J alakú kivezetéssel szerelt tokokat, melynél a kivezetés a chip tokozása alá hajlik. Ahogy az SO tokok fejlődtek, számos új tokozás jelent meg: TSOP (Thin Outline Package), PTP (Paper Thin Package), vagy az UTSOP (Ultra Thin SOP).

Az SO-nál leggyakrabban alkalmazott tokozóanyag a műanyag és a kerámia, ezeknél pedig létfontosságú a jó hővezető képesség, a nagy megbízhatóság, és hogy kevés parazita tulajdonságokkal rendelkezzenek. A tokok hermetikus lezárása is megoldható, ehhez általában fém-üveg tokokat használnak. Ilyenkor a tok anyaga kovar (vas, kobalt és nikkel ötvözet), melynek a hőtágulási tényezője megegyezik az üvegével, ami nagyon fontos, hogy elkerüljük a különböző hőtágulási együtthatók okozta mechanikai feszültségeket.

Sokáig a nagy kivezetőszám elérésének egyetlen lehetősége a QUAD elrendezés, ami a tok négy oldalán elhelyezkedő kivezetésekre utal. Ide tartoznak például a QFP (Quad Flat Package, 2.4. ábra), a TQFP (Thin Quad Flat Package) és a QFJ (Quad Flat J-leaded Package) tokozású alkatrészek melyeknél a kivezetések száma akár 500 is lehet. Az első QUAD elrendezésű tokok esetén a lábtávolság még 1.27 mm volt, ez az érték mára 0.3 mm-re csökkent.



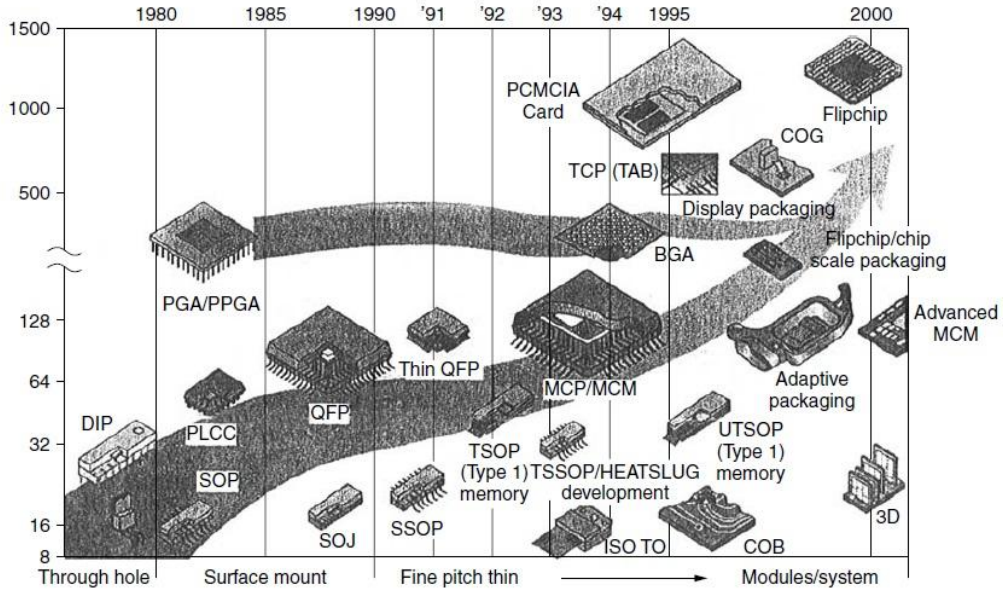
2.4. ábra. A QFP tokozás (bal oldalt 48 kivezetéses tok, jobb oldalt 120 kivezetéses tok) [3]

Finom raszterosztású (fine pitch) alkatrészekről akkor beszélünk, ha a kivezetések osztástávolsága nem haladja meg a 0.63 mm-t [7]. Ezeknél az alkatrészeknél különös figyelmet kell fordítanunk a pasztafelvitel mennyiségére és a forrasztásra, mivel a kivezetések kis távolsága miatt nagy a forraszhíd, illetve nem megfelelő nedvesítés kialakulásának valószínűsége.

A 90-es évek közepén már nagyon nagy volt az igény olyan tokozásra, ami megfelelően nagyszámú kivezetéssel rendelkezik, ugyanakkor az is fontos volt, hogy se a technológia költsége, se a tokozás mérete ne növekedjen jelentős mértékben. Az új elv lényege az volt, hogy a kivezetések a tok alján helyezkedjenek el, úgynevezett forraszgolyók formájában. Ezt a tokozási formát BGA-nak (Ball Grid Array) nevezzük. A BGA ki-

alakítás legnagyobb hátránya az, hogy a hibaanalízis, valamint a javítás folyamata sokkal bonyolultabb lett, mivel a kivezetések szabad szemmel nem láthatóak, így a kötések csak röntgenes vizsgálattal ellenőrizhetők. A BGA tokok kézzel már nem pozícionálhatóak, csak fine pitch beültetőgépek segítségével.

A ma használatos CSP (Chip Scale Package) tokozású alkatrészek legelterjedtebb formája a QFN (Quad Flat No leads) tok. Ez a típus távolról hasonlít a QUAD tokozásra, hiszen a kivezetések a tok négy oldalán helyezkednek el, csakhogy itt nem beszélhetünk hagyományos értelemben vett kivezetésekről. A tok alján elhelyezett fémfólia kivezetések biztosítják a kontaktus felületeket a chipen, de ezek nem állnak el a toktól, mint a hagyományos lábak. A QFN tokozás gyakori hibája a forrasztás utáni rövidzár képződés, melyet a forraszpaszta mennyiségének optimalizálásával lehet elkerülni.

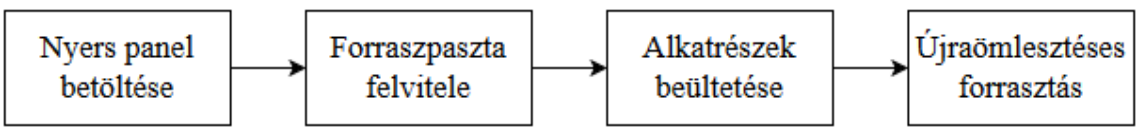


2.5. ábra. Szerelhető alkatrészek fejlődése/megjelenése [2]

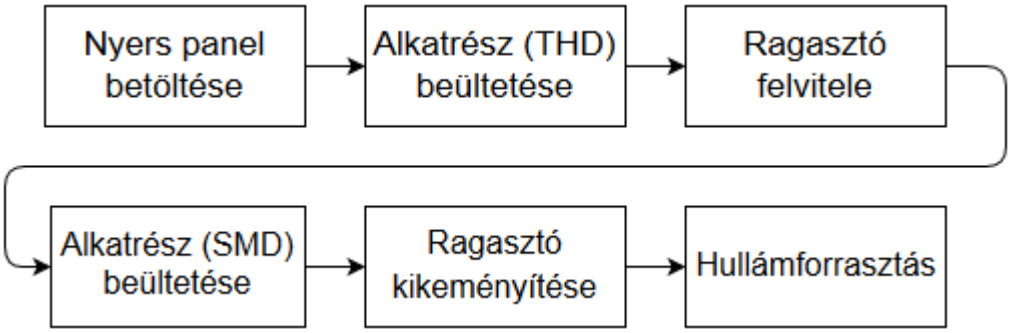
2.4. A felületszerelési technológia munkafolyamatai

Mint ahogy azt már korábban tárgyaltuk, megkülönböztetünk egymástól tiszta és vegyes szerelési technológiát. Különböző szerelési változatok esetén más-más munkafolyamatokon keresztül jutunk el a nyers paneltől a késztermékig (2.6.-2.8. ábra). A beültetett alkatrész rögzítése szempontjából két eljárást különböztethetünk meg egymástól:

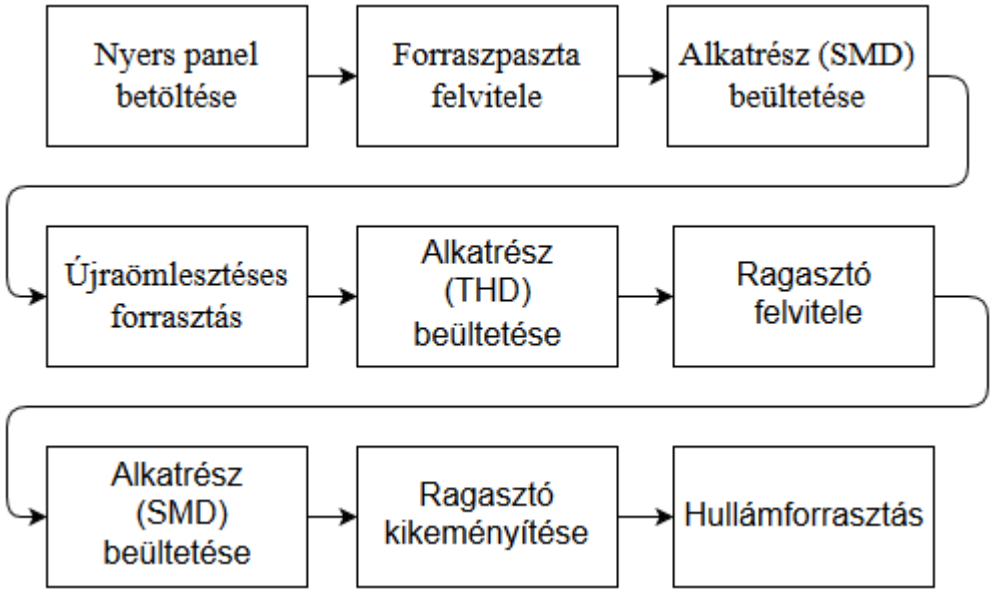
- ragasztással történő rögzítést;
- forrasztópaszta segítségével történő rögzítést.



2.6. ábra. A felületszerelési technológia folyamatainak sorrendje tisztán felületi szerelés esetén



2.7. ábra. A felületszerelési technológia folyamatainak sorrendje vegyes szerelés esetén (egyik oldalon csak SMD, másikon csak THD)



2.8. ábra. A felületszerelési technológia folyamatainak sorrendje vegyes szerelés esetén (legalább egyik oldalon SMD és THD is)

2.4.1. Ragasztásos technológia

Bizonyos esetekben szükségünk lehet az alkatrészek ragasztására, például mikor vegyes szerelésnél hullámforrasztást alkalmazunk. A vegyes szerelés első lépése, hogy a felületre szerelhető alkatrészeket felragasztjuk a szerelőlemezre. Ezután megfordítjuk a szerelőlemezt és beültetjük a furatszerelt alkatrészeket, végül a hullámforrasztás következik. Ha nem alkalmaznánk ragasztást, akkor a beültetett alkatrészek leesnének a szerelőlemez megfordításakor, valamint a forraszhullám is könnyen lesodorhatná az alkatrészeket.

A felhasznált ragasztók lehetnek egy vagy több komponensűek. A ragasztó kiválasztása során figyelembe kell venni a térhálósítási időt, a tárolási időintervallumot és azt, hogy a ragasztott kötés bontható-e vagy sem. A többkomponensű ragasztók hátránya, hogy összekevert állapotban néhány napon belül fel kell őket használni, míg az egy komponensűek több hónapig is megőrzik felhasználhatóságukat. Az SMT-ben leggyakrabban alkalmazott ragasztók hő hatására szilárduló műanyagokból készülnek. A ragasztószer

panelre történő felviteléről adagológépek gondoskodnak. Három adagolási módszert különböztethetünk meg egymástól:

- Auger adagolás: egy menetes tengely segítségével történik meg a ragaszó kipréselése a tubusból;
- Sűrített levegővel történő adagolás: a tubusban uralkodó nyomás változtatásával érjük el a ragasztó kiszivárgását;
- Pisztolyszerű adagolás: a tubusban lévő ragasztó kilövődik a panelre egy rugó segítségével.

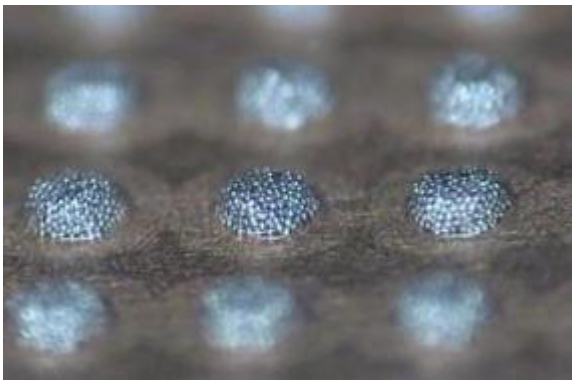
Az adagológépeknek a ragasztópöttyök felvitelét századmilliméter pontossággal kell elvégezniük. Minden adagoló a ragasztás megkezdése előtt próbaragasztást végez a hordozó nem használt területén. A tesztragasztás során letett pöttyöket egy kamera segítségével leellenőrzi, és megfelelőség esetén megkezdi a ragasztó felvitelét. Az ellenőrzés rendkívül fontos, hiszen alkatrész-beültetés után már bonyolult az esetleges hiba kijavítása. A ragasztásos technológia előnye, hogy kevés lépésből áll, és bár a ragasztóanyag költséges, de nagyon kis mennyiség is elegendő belőle a mechanikai kötés létrejöttéhez.

2.4.2. Újraömlesztéses (Reflow) technológia

A ragasztásos technológiával ellentétben, ahol a ragasztó csak a mechanikai rögzítést biztosítja, és a hullámforrasztás hozza létre az elektromos kötést, az újraömlesztéses technológia során alkalmazott úgynevezett forrasztópaszta nem csak az SM alkatrészek rögzítését szolgálja, hanem megolvasztásával létrehozhatóak az elektromos kontaktusok is. A két technológia közötti különbséget jól mutatja a tény, hogy a ragasztásos technológia során a kemencében még csak a ragasztó kikeményítése történik meg, míg a reflow eljárás során a kemence a technológia utolsó fázisa.

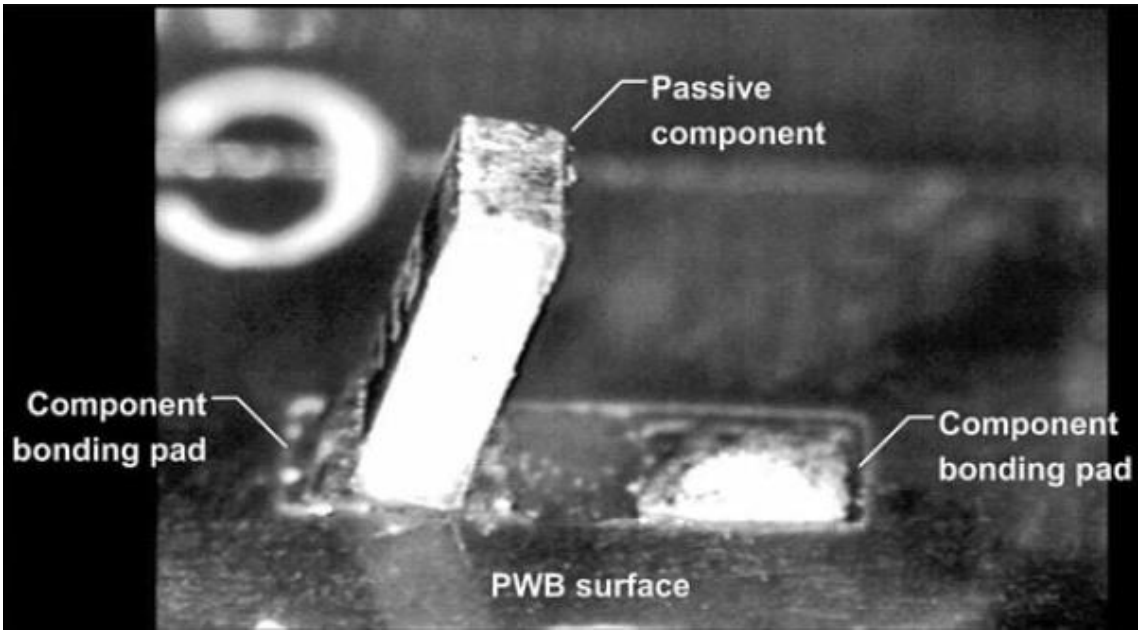
2.4.3. A forraszpaszta

A felületszerelt gyártástechnológiában alkalmazott forrasztópaszták (2.9. ábra) összetételüket tekintve fémporból, folyasztószerből és különböző szerves adalékanyagokból tevődnek össze. Először a fémport állítják elő, létrehozzák a megfelelő ötvözetet, felfűtik az olvadási hőmérséklete fölé, és porlasztással létrehozzák a forraszgömböket, melyek szokásos átmérője 10-50 μm, fémtartalma pedig körülbelül 90 tömegszázalék. A gömb alak elérése nagyon fontos, mivel adott térfogat mellett a gömbformának a legkisebb a felülete, és a kisebb felület kisebb mértékű oxidációt eredményez, ami pozitív irányban befolyásolja a forrasztás minőségét. Forraszpaszta választáskor figyelembe kell azt az általános elvet, miszerint a szemcsék átmérőjének legalább ötször kisebbnek kell lennie a stencilapertúra szélességénél.



2.9. ábra. Felvitt forraszpaszta [10]

A forraszpaszták másik alkotóeleme a folyasztószer, mely lehet szerves, szervetlen vagy gyantaalapú. A folyasztószer fő feladata, hogy a kemencében végbemenő megömlesztéskor megfelelően benedvesítse a felületeket. A nem megfelelő nedvesítés az alkatrész elcsúszásához, vagy az úgynevezett sírkő jelenséghez (tombstone effect, 2.10. ábra) vezethet. Aktivátor anyagként általában valamilyen gyenge savat alkalmaznak, amely nélkülözhetetlen a megfelelő nedvesítéshez. Az elsődleges szempont az, hogy a forrasztási hőmérséklet elérése előtt eltűnjön a pad-ek és az alkatrészek kivezetései között kialakult oxidréteg.



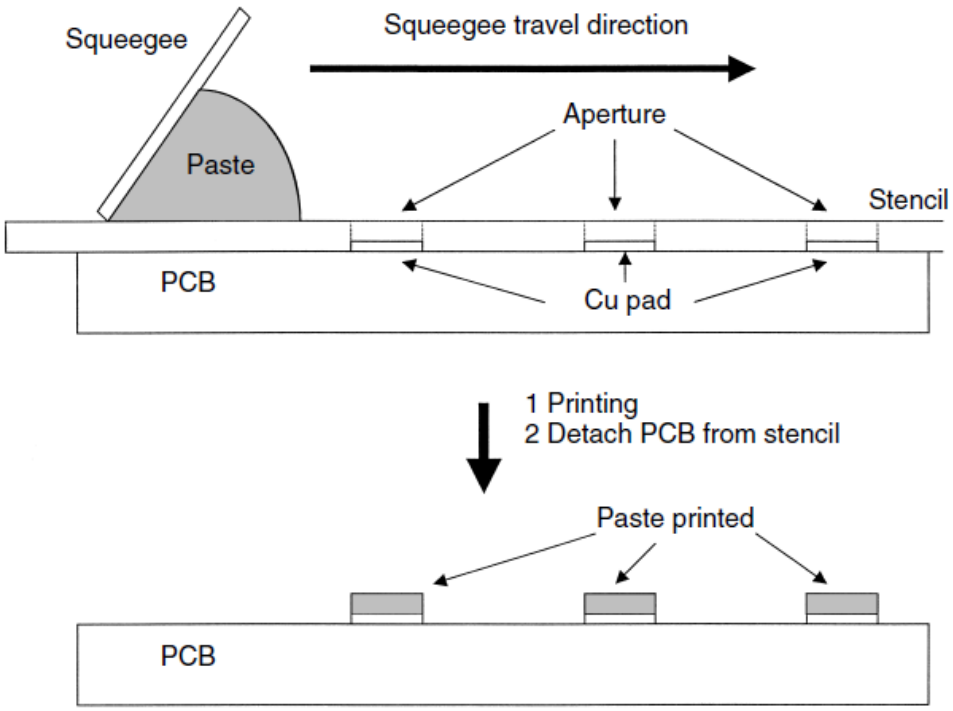
2.10. ábra. Példa a sírkő effektusra [4]

A forraszpaszták tárolásuk során hűtést igényelnek, mivel hűtés nélkül felhasználhatóságuk pár hétre korlátozódna (a levegő páratartalmától függően), míg hűtéssel hónapokig vagy akár egy évig is felhasználhatók maradnak minőségromlás nélkül. A pasztákat felhasználásuk előtt szobahőmérsékletre kell felmelegíteni [9].

2.4.4. Forraszpaszta felvitele

A paszta felvitele több módon is megtörténhet. Az egyik ilyen mód a diszpenzálás, más néven a cseppadagolás. Ilyenkor a pasztát egy fecskendő segítségével viszik fel a kontaktus felületre. Ezt a módszert kis darabszámú termékek esetén alkalmazzák. A másik technológia, amelyet SMT soroknál már nagy darabszám esetén is használható a stencilnyomtatás (2.11. ábra). Ilyenkor az adott nyomtatott áramköri lemez tervei alapján egy 75-200µm vastagságú fémfóliát ún. stencilt (maszkot) készítenek. A stencilen a nyomtatott áramköri hordozón található pad-eknek megfelelően nyílásokat, más néven apertúrákat alakítanak ki.

A stencilnyomtató berendezésbe helyezett panelre a forraszpasztát nyomtatókések nyomják át az apertúrákon a kontaktusfelületre. A nyomtatás történhet egy-, illetve két fázisban. Az egyfázisú pasztázás során a kés csak egyszer halad át a felvitel helyén, míg két fázis esetén oda-vissza történik meg a paszta felvitele. A megfelelő mennyiségű forraszpaszta felvitelét az apertúrák mechanikai méretének és a stencilfólia vastagságának megfelelő megválasztásával érik el.



2.11. ábra. Stencilnyomtatás vázlatosan [2]

2.4.5. Alkatrészek beültetése

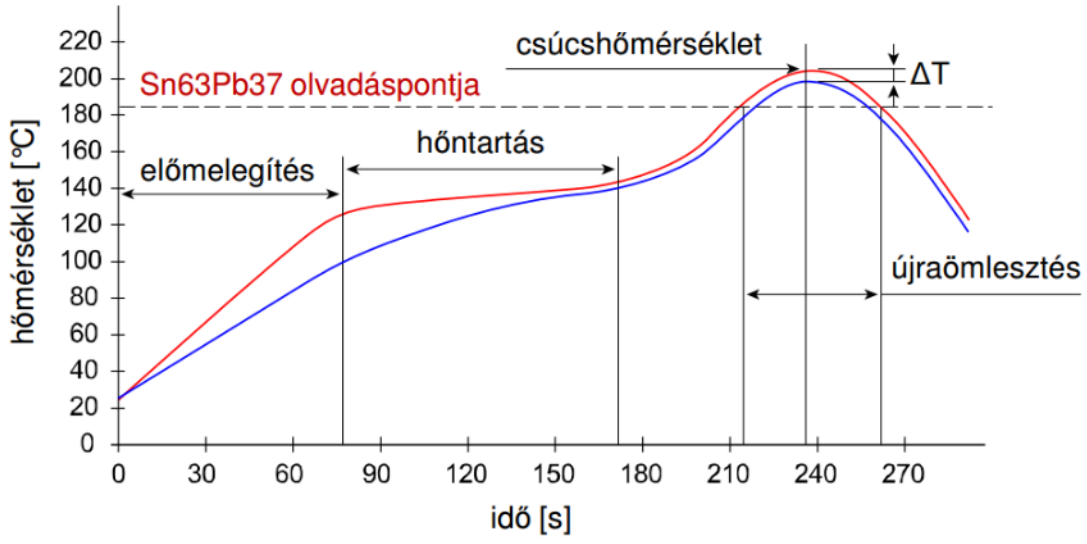
Az alkatrészek beültetése az a munkafázis, melyben nyilvánvalóvá válik az SM technológia lényege, a tökéletes automatizálhatóság. A legmodernebb beültető berendezések már akár 130000 [1] SMD beültetésére képesek egyetlen óra leforgása alatt. Ezek a gépek a pick and place (PNP), azaz „megfog és beültet” technológia alkalmazásával működnek. Ezt tipikusan vákuumpipettás technológiával valósítják meg, mely azt jelenti, hogy az alkatrészeket a tárolóból egy az adott alkatrésztípushoz készített vákuumos szí-

vófej (noozle) veszi fel és teszi le a hordozóra. A gép letétel előtt egy kamera vagy lézer segítségével ellenőrzi, hogy pontosan milyen orientációval és elfordulási szöggel sikerült felvennie az alkatrészt, és ezeket korrigálja. Mivel különféle tokozásokhoz más-más szívófejre van szükség, ezért szükség van a szívófejek cseréjére. Emiatt alakítanak ki minden gépben egy ún. noozle állomást, melynek segítségével „önállóan" (programban definiálva) tudja a szívófejeket cserélni. A mai berendezések forgófejes P&P fejegységgel rendelkeznek.

2.4.6. Újraömlesztéses forrasztás

Az alkatrészek felhelyezése után a forraszanyag megömlesztése következik. Az újraömlesztéses forrasztás megbízhatósága annak a függvénye, hogy milyen eredményesen lehet a fűtést irányítani és a fűtési variációkat a különböző panelekre alkalmazni. Az irányított fűtést hőprofilnak nevezzük (2.12. ábra). A hőprofil jellemző szakaszai a következő táblázatban láthatóak:

Hőprofil szakasz	Ólmos paszta esetén	Ólommentes paszta esetén
Előmelegítés (ramp)	Hőmérséklet tartomány: 1-150 °C Hőmérséklet változás: <2 °C/s Szakaszon töltött idő: 60-150s	Hőmérséklet tartomány: 1-160 °C Hőmérséklet változás: 2-4 °C/s Szakaszon töltött idő: 60-150s
Hőntartás (soak)	Hőmérséklet tartomány: 150-185 °C Szakaszon töltött idő: 60-90s	Hőmérséklet tartomány: 160-200 °C Szakaszon töltött idő: 60-120s
Újraömlesztés (reflow)	Csúcshőmérséklet: 205-230 °C Szakaszon töltött idő: 45-90s	Hőmérséklet tartomány: 230-255 °C Szakaszon töltött idő: 20-60s
Hűtés (cool down)	Hűtés 130 °C-ig Hőmérséklet változás: 3-4 °C/s	Hűtés 130 °C-ig Hőmérséklet változás: 4-5 °C/s

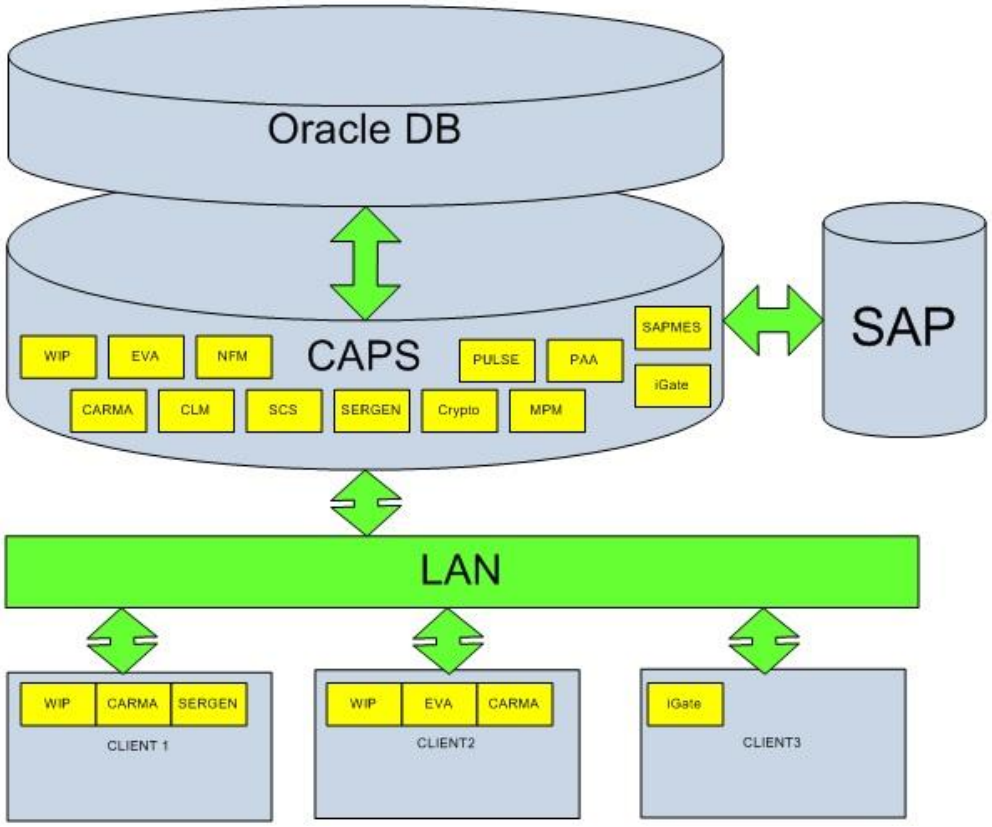


2.12. ábra. Újraömlesztéses forrasztás tipikus hőprofilja [6]

3. A traceability rendszer [11]

3.1. Általános leírás [12]

Annak érdekében, hogy egy termék előállítással foglalkozó cég magas termékminőség és alacsony költségszint mellett is megfeleljen a kritikusan csökkent szállítási idők teljesítésének, lehetőség szerint a gyártás minden egyes pontjában át kell térnie a kézi adatbevitelről az automatizált módszerekre. Ennek érdekében termelés-végrehajtás felügyeleti szoftverrendszereket fejlesztettek ki, melyeket a nemzetközi szaknyelvi terminológiában MES – Manufacturing Execution System – rendszernek neveznek. A MES tulajdonképpen egy, a termelési folyamatokat felügyelő számítógépes rendszer, amely a vállalat gyártási folyamatainak valós idejű felügyeletét jelenti. A rendszer információt biztosít a gyártási rendelések állapotáról, a gyártásközi anyagszükségletről, üzemzavarokról, gyártási veszteségekről, gyártóberendezések kapacitásának kihasználtságáról, ütemezett karbantartási periódusokról, üzemórákról. A MES szolgáltatása az egymással párhuzamosan futó termelési tevékenységekről gyűjtött adatok rendszerezése, valamint összegzett információ biztosítása a termelést végrehajtó dolgozók és az azt irányítók részére, támogatva és gyorsítva a szükséges döntések meghozatalát, intézkedések végrehajtását. A MES rendszer felépítése a 3.1. ábrán látható.



3.1. ábra. A MES felépítése [11]

3.2. Alapvető fogalmak

A fejezet további pontjainak ismertetése előtt fontos, hogy tisztázzunk néhány alapfogalmat.

3.2.1. Egységek azonosítása

Traceability rendszerben az egyik legalapvetőbb igény, hogy a megfigyelt termékeket egyedileg be tudjuk azonosítani. Minden egységhez két paraméter tartozik, ami alapján pontosan be lehet azonosítani. Az egyiket egység azonosítónak (Unit Id), a másikat egység azonosító típusnak (Unit Id Type) nevezik (3.2. ábra). Utóbbi bevezetésére azért volt szükség, mert néhány megrendelő pontosan meghatározza, hogy milyen azonosító kerüljön a nekik gyártott termékekre, és ennek bevezetésével elkerülhető, hogy bonyodalmat okozzon, ha két különböző megrendelő ugyanazokat az azonosítókat szeretné látni.

3.2.2. Pass, Fail, Scrap

Egy egységhez három besorolási típus tartozhat: pass (megfelelt), fail (hibás), scrap (selejt). Egy egység akkor kap pass besorolást, ha minden teszten átment, vagy ha egy hibát sikeresen javítottak. A fail és a scrap között az a különbség, hogy az előbbit még lehet javítani, az utóbbit pedig már nem. Gyártási folyamatban csak a pass besorolású egység léphet tovább – kivételt képez, ha egy javító állomás következik.

3.2.3. Megrendelés

Minden egység egy megrendeléshez (order) tartozik. A megrendelés tartalmazza a legyártandó egység típusát (anyagszámát), mennyiségét és a hozzá tartozó munkatervet.

3.2.4. Munkaterv

Egy gyártási megrendelés alapvető része a gyártási folyamatot leíró munkaterv. Ez adja meg a gyártási folyamat lépéseit állomásról állomásra. A teljes lista birtokában adható csak meg a gyártásra fordítandó ember- és gépidő szükséglet, amely a variábilis gyártási költségek számításának alapját képezi.

3.2.5. Állomás típusok

Az állomásoknak két típusa van: kötelező és opcionális. Kötelezőek azok az állomások, amelyeken az egységeknek mindenképp át kell haladniuk. Ilyen lehet például egy beültető állomás SMT sor esetén. Az opcionális állomások azok, amelyeken az egységeknek nem kötelező járniuk ahhoz, hogy a késztermékeket megkapjuk. Ilyen lehet például egy javító állomás, hiszen oda egy egység csak akkor kerül, ha egy teszt során hibás besorolást kapott. Továbbá opcionális lehet egy mérőállomás is, hiszen egyes folyamatok annyira stabilnak mondhatóak, hogy a termékeknek csak egy bizonyos százalékát szükséges ellenőrizni.

3.2.6. Checkin, Checkout

Mikor egy egység egy állomásra kerül, az egységet be kell jelentkeztetnünk (Checkin), majd mielőtt kikerül, ki kell jelentkeztetnünk (Checkout). Amennyiben az egység egy állomásról nem lett kijelentkeztetve, nem járhat a következő állomáson, tehát oda nem tudjuk bejelentkeztetni.

3.2.7. Tesztfutás

Minden egységtípus méréseihez létre lehet hozni teszttterveket (testplan). Egy testplan egy vagy több tesztlépést (teststep) tartalmazhat. A tesztlépések három típusba sorolhatók:

1. változó (variable);
2. attributív (attributive);
3. kiegészített attributív (attributive ext).

Változó típusnál az eredmény egy valós szám, ami megfelelt, ha egy bizonyos előre meghatározott tartományon belül van, és hibás, ha azon kívül. Az attributív típus a mérés kimenetelét két értékre szűkíti le: ’P’ mint Pass és ’F’ mint Fail. A kiegészített attributívnál az eredmények egy előre meghatározott lista (ún. attributív lista) elemei lehetnek.

3.3. Kliens-szerver architektúra [13]

A MES kliens-szerver architektúrán alapul, melynek lényege, hogy a feladatokat elosszuk olyan számítógépek között, amelyek csak a hálózaton keresztül érintkeznek egymással, megkönnyítve a karbantartás elvégzését. Megoldható például, hogy javítsunk, frissítsünk, áthelyezzük vagy akár kicseréljünk egy szervert anélkül, hogy klienseire ez bármilyen hatással lenne. Ezt a változtatásoktól való függetlenséget információ elrejtésnek nevezik. Az összes adat a szervereken tárolódik, amelyek általában sokkal hatékonyabb biztonsági ellenőrzéssel rendelkeznek, és jobban tudják szabályozni az erőforrásokhoz és adatokhoz való hozzáférést.

3.4. A MES főbb alkalmazásszervereinek feladata

3.4.1. WIP (Work In Progress)

A Work In Progress (gyártási folyamat vagy folyamatvezérlés) az egyik legfontosabb modul, szinte minden MES-be kötött állomás használja. A WIP segítségével egyebek mellet a következő műveleteket tudjuk elvégezni.

Egység létrehozása (WIP-ben);
Egység állomások közötti mozgatása (checkin, checkout), az állapotok pontos tárolása időbélyegekkel;
Egység állapotának megváltoztatása (pass, fail, scrap);
Egységhez tartozó információk lekérdezése.

3.4.2. MAMA (Material Managment)

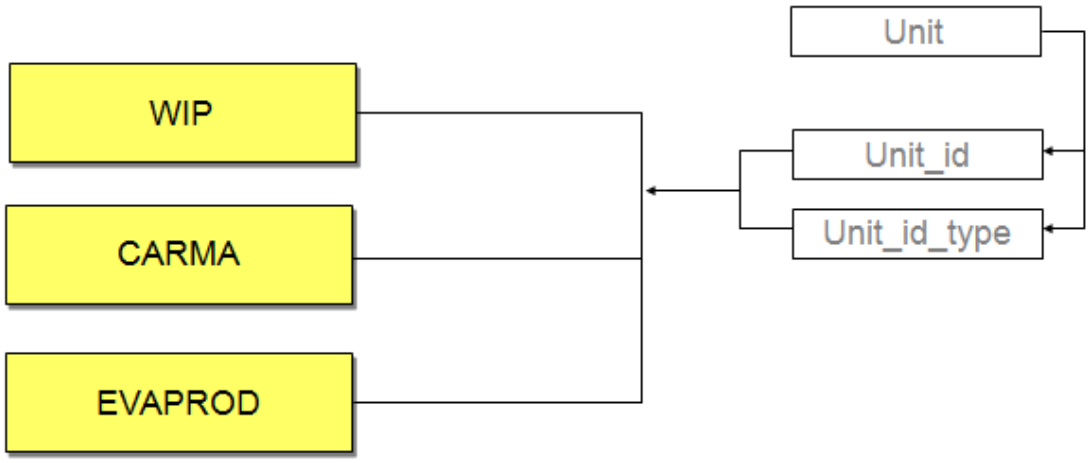
A MAMA modul a félkész, negyedkész és egyéb beépülő anyagok nyomonkövetéséért felelős. Amikor egy termék egy adott komplett gyártási folyamaton végigmegy félkész vagy negyedkész termék lesz belőle. Ekkor a terméket már a WIP modul helyett a MAMA modul segítségével követjük nyomon, „átkerül MAMA-ba”.

3.4.3. EVAPROD (Evaluation of Product)

Az EVAPROD tárolja a létrehozott teszterveket, segítségével lehet egy tesztfutás eredményét feltölteni az adatbázisba, a tesztfutást kiértékelni és az eredményét lekérdezni.

3.4.4. CARMA (Carrier Management)

A CARMA feladata a több egységből álló panelek (multiboard, nutzen) és a csomagolási listák kezelése. Multiboardoknál mindig van egy kitüntetett egység, mely alapján az összetartozó egységeket be lehet azonosítani. A WIP modul csak ezt az egy egységet kezeli, azt pedig, hogy ehhez az egységhez milyen alegységek (subunit, miniboard) tartoznak, a CARMA modulban tartjuk nyílván.



3.2. ábra.Táblaterek közötti kapcsolat [11]

3.4.5. SERGEN (Serial Number Generator)

A SERGEN feladata reguláris kifejezések alapján egyedi azonosítók generálása.

3.4.6. PULSE

A PULSE a többi modullal ellentétben nem rövidítés. Nevét onnan kapta, hogy ún. pulzusokat küld a szervernek, melyek lehetővé teszik gyártósori információk, állásidők, sorkihasználtság vizsgálatát. Ezek a pulzusok gyakorlatilag időbélyegek, melyek a PULSE akkor küld, amikor egy egység elhagy egy adott állomást. Minden állomásnál be lehet állítani, hogy mekkora legyen az a két egység között eltelt időintervallum, amit a PULSE már állásnak érzékel.

3.4.7. MPM (Master Process Monitoring)

Lehetőséget nyújt, hogy bizonyos minőségügyi határértékek megszegése esetén riasztást küldjünk, és a gyártósort még időben automatikusan megállítsuk, hogy megakadályozzuk a nagymennyiségű selejt keletkezését, a hiba továbbgyűrűzését. Ilyen szabály lehet például az utolsó 100 gyártott termékből 5 darab hibás egység jelenléte.

3.4.8. PAA (Part Average Analyse)

Egy mérési sorozaton belül vizsgálja a tesztfutások eredményét, és zárolja azokat az egységeket, amelyek valamilyen okból kifolyólag ugyan a határértékeken belüli, de a gyártási mennyiségre vetített átlagértéktől kiugróan eltérő mérési értéket mutattak.

3.4.9. SAPMES

A vállalatirányítási rendszer (SAP) és a traceability rendszer (MES) közötti kapcsolatot teremti meg.

3.4.10. IGate

WEB-es alapú termékadat-visszakeresést biztosít. Az IGate-en keresztül gyakorlatilag hozzáférünk a MES adatbázistáblák adataihoz. A 3.3. ábrán egy megrendelés munkafolyamatait láthatjuk az IGate felületén.

Operations Report for Order = 168741210 ; Product Family = VED_ER100_SMT ;Type = SEMI ; Number = 2851557822200

Page size of 10 Rows of 10 page 1 of 1

Order	Search	Operation		Workcenters	History	Inventory							Runs			Timestamps		Lock	Expand
	J	No	Name			Total	Checkin	Start	End	Fail	Yield	Scrap	Start	Fail	Yield	Start	End		
168741210	J	110	MES_START_TOP	M_E100_T	history	0	0	0	0	0	0	0	1800	0	1800	19-May-2016 22:11:23	20-May-2016 05:13:42	no	expand
168741210	J	112	MES_HEAT_SINK_PRESSING	MES_HEAT	history	960	0	0	0	0	960	0	3600	0	3600	19-May-2016 22:12:02	20-May-2016 05:15:11	no	expand
168741210	J	115	MES_HEAT_SINK_ANALYSIS	MES_HE_A	history	0	0	0	0	0	0	0	0	0	0	--	--	no	expand
168741210	J	117	MES_CONVEYOR	MES_CONV	history	30	0	0	0	0	30	0	840	0	840	20-May-2016 04:15:53	20-May-2016 13:23:29	no	expand
168741210	J	120	MES_PASTE_PR	MES_P_PR	history	3	0	0	0	0	0	3	3	3	0	20-May-2016 03:28:43	--	no	expand
168741210	J	130	MES_PASTE_AOI	MES_P_AO	history	0	0	0	0	0	0	0	0	0	0	--	--	no	expand
168741210	J	140	MES_PLACEMENT	MES_PLAC	history	72	66	0	0	0	6	0	741	0	741	20-May-2016 04:32:03	20-May-2016 13:23:17	no	expand
168741210	J	150	MES_OVEN	MES_OVEN	history	102	0	0	0	0	102	0	747	0	747	20-May-2016 04:32:14	20-May-2016 13:23:26	no	expand
168741210	J	160	MES_SORTING	MES_SORT	history	0	0	0	0	0	0	0	0	0	0	--	--	no	expand
168741210	J	170	MES_INSP	MES_INSP	history	24	24	0	0	0	0	0	609	0	609	20-May-2016 04:51:33	20-May-2016 13:23:39	no	expand

3.3. ábra. Egy megrendelés munkafolyamatainak áttekintése IGate-en

3.5. Traceability az SMT sorokon

Mivel MES rendszerhez kapcsolt állomásoknál alapvetően ellenőrizendő, hogy az egység járt-e az előző állomáson, így ezt az egyes folyamatok részletezésénél külön nem emelem ki. Az SMT-MES folyamatábra (3.6. ábra) a fejezet végén található.

3.5.1. Gyártógépek közötti kommunikáció [15]

Az SMT gyártógépek az ún. SMEMA kommunikációs protokoll segítségével kommunikálnak egymással, ami nevét a protokollt kidolgozó egyesület (Surface Mount Equipment Manufacturers Association) után kapta. A SMEMA lehetővé teszi, hogy a gyártógépek és a konvejorok (anyagmozgatók) tudassák egymással, ha elfoglaltak, vagy ha épp készen állnak, hogy átadjanak vagy fogadjanak egy áramkört. Az információátadás szekvenciálisan működik, nem tartalmaz címzést.

3.5.2. A panel betöltése és lézergravírozás

Mielőtt egy termék gyártása megkezdődne, a gyártósoron át kell állítani a konvejorok szélességét az adott termék fizikai méretének megfelelően. Ezt általában minden egyes gépnél külön-külön meg kell tenni, de bizonyos esetekben a gépek képesek egymásnak átadni ezt az információt. A paneleket ún. magazinokba (3.4. ábra) kell helyezni, a betöltő gép (loader) ezekből veszi ki és helyezi őket a lézerező állomás konvejorára. A lézergravírozás során egy koncentrált lézersugár hatására az anyag elpárolog vagy elég. Ezzel rendkívül tartós, jól látható vonalkód (barcode) készíthető szinte bármilyen anyagra. A lézerező állomás feladata, hogy lézergravírozással a SERGEN modul által generált egyedi azonosítót vonalkód formában felvigye a hordozóra, valamint az, hogy a lézerezett egységet létrehozza WIP-ben. Több egységből álló panelek esetében minden egyes miniboardot külön azonosítóval lát el, hiszen a szétválasztás után már külön egységként is tudni kell kezelni őket. A lézerező állomás nem mindig van jelen, mert bizonyos multiboardok már vonalkóddal ellátva kerülnek a gyárba, valamint kétoldalas termékeknél a második oldalra nem minden esetben kell az azonosítót felvinni, például akkor sem, ha a gépek vonalkódolvasói képesek a panel mindkét oldalát figyelni.

3.4. ábra. Magazin [15]

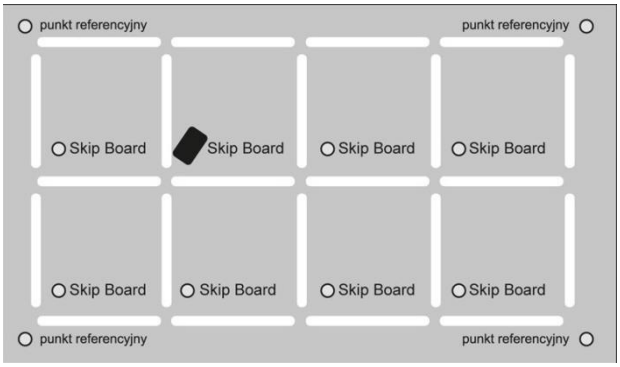
3.5.3. Pasztanyomtatás

A lézerezőállomás után a pasztanyomtatás következik. Az állomáson lévő MES szoftver a vonalkód alapján azonosítja a terméket és ellenőrzi, hogy az egység járhat-e az állomáson, a pasztanyomtatóban a megfelelő stencil van-e, és volt-e tisztítva. A stencileket kétféle módon kell tisztítani. Az egyik az ún. gyorstisztítás, melyet a gép mindig elvégez magának 4-5 panel nyomtatása után. A másik az ún. nagytisztítás, melyet egy műszakban általában egyszer kell elvégezni, de gépmegállás esetén a forraszpaszta lehetséges beszáradása miatt előfordulhat, hogy többször is. Amennyiben mindent megfelelőnek ítél a szoftver, az egységet beengedi az állomásra, és elvégzi a paszta felvitelét.

3.5.4. Optikai ellenőrzés

Pasztanyomtatás után egy, a pasztanyomtatást optikailag ellenőrző gép, az ún. AOI (Automated Optical Inspection) berendezés következik, melynek eredménye feltöltésre kerül az EVA-ba. Amennyiben hibát érzékel, megállítja a sort, és ezt jelzi a sor mellett dolgozó operátornak. Az AOI által készített képek segítségével az operátor eldöntheti, hogy ez egy téves jelzés (false call) vagy valódi hiba. Előbbi esetén felülírhatja az AOI

döntését, utóbbi esetben pedig két lehetősége marad. Ha több miniboard is hibás, akkor az egész panelt kiveszi a gyártásból, ha csak egy a hibás, akkor olcsóbb megoldás az, ha informálja a beültető gépet, hogy arra a miniboardra ne ültessen SM alkatrészeket. Ezt úgy tudja megtenni, hogy a minden alegységen található „Skip board" mezőn inkspotot (tintafoltot) helyez el, azaz lefesti azt (3.5. ábra).



3.5. ábra. Multiboard inkspottal [16]

3.5.5. SMD beültetés

SMD beültetés megkezdése előtt ellenőrizzük, hogy az adott anyagszámhoz megfelelő program és egységek vannak-e betöltve. Ezeket az SMD saját adatbázisából tudjuk kinyerni. Az alkatrészbeültető általában több kisebb modulból ál, melyek mind más-más alkatrészek beültetésére alkalmasak. Ugyan az SMD beültető gépnek csak az elején és a végén van vonalkódolvasó, de az egyes modulok képesek egymásnak eljuttatni az információt, hogy épp milyen típusú terméket adnak át, így egy időben egy beültető gépen többféle termék is járhat. A modulok egymás közötti kommunikációját whispering-nek (suttogás) nevezik.

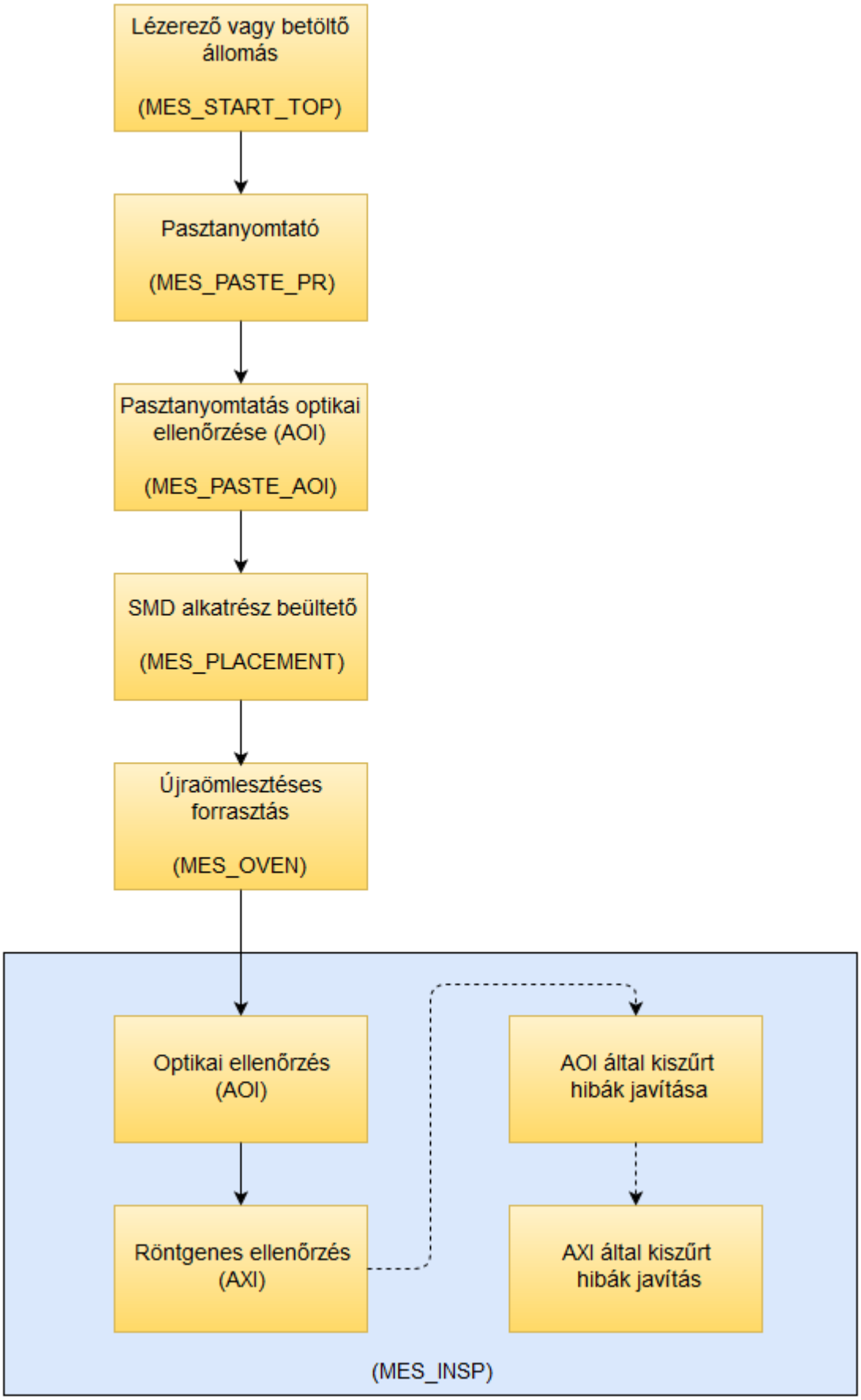
3.5.6. Újraömlesztéses forrasztás

Az SMD beültetés után az újraömlesztéses forrasztás következik. Mielőtt beengedjük az egységet az állomásra, ellenőriznünk kell, hogy a megfelelő hőprofil van-e a gépen beállítva, valamint azt is, hogy a pasztázás óta nem telt-e el túl hosszú idő. Ekkor ugyanis a paszta beszáradhatott, ami alkalmatlanná teszi az újraömlesztéses forrasztásra.

3.5.7. AOI és az AXI

A kemence után a végső ellenőrzések következnek. Az AOI (optikai ellenőrző) és az AXI (röntgen-es ellenőrző) egymás után helyezkednek el. Az egység mindenképpen végighalad mindkét ellenőrzési folyamaton, akkor is, ha az adott egységnél szükségtelen a röntgenes vizsgálat. A gépek hiba esetén jeleznek az operátornak, aki a korábbiakhoz hasonlóan dönthet arról, hogy a gép tévesen jelzett-e vagy valódi hiba áll fent. Amennyiben valódi hiba van, az egység a javítóállomásra kerül. Ha nem lehet kijavítani, az egység selejt lesz. Amennyiben ki lehet javítani, javítás után a terméket vissza kell helyezni az ellenőrző állomások elejére, ugyanis a gépeknek el kell fogadni a javítás eredményét is. Az AOI és az AXI a mérési adatait logfájlokba menti (naplózza). Minden géphez tartozik egy ún. parser szoftver, ami értelmezi a logfájlok tartalmát és

feltölti tesztfutásként a MES adatbázisba, így később visszakereshető, hogy mely alkatrészekkel volt a hiba. Ellenőrzések után a pass besorolású egységek sikeresen végigmentek az SMT gyártási folyamaton, és mint félkész termékek átkerülnek a MAMA modulba. Ezután a termékek az ún. backend sorra kerülnek, ahol megtörténik a technikai sáv levágása, azaz a darabolás.



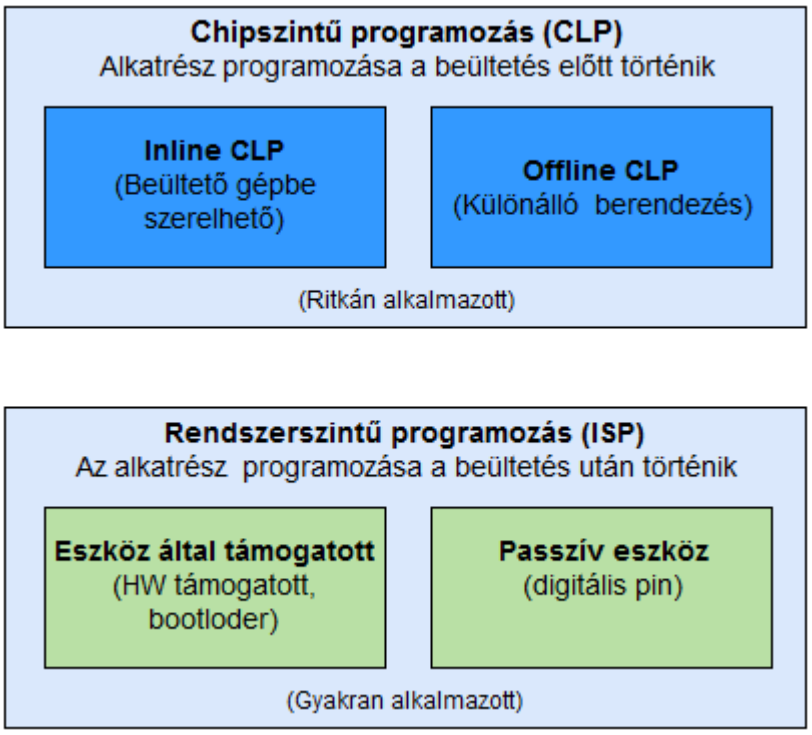
3.6. ábra. SMT-MES folyamatábra (zárójelben az állomások MES rendszerben található neve)

4. Alkatrész programozás

4.1. Az alkatrész-programozási lehetőségek áttekintése [20]

Az alkatrészek programozási folyamatait két kategóriába sorolhatjuk:

1. Chipszintű programozás (Chip-Level Programming - CLP);
2. Rendszerszintű programozás (In-System Programming - ISP).



4.1. ábra. CLP és ISP

4.1.1. Chipszintű programozás (Chip Level Programming – CLP)

Chipszintű programozásról akkor beszélünk, amikor az IC annak felprogramozásakor még nincs beültetve a nyomtatott áramkörbe. Ez a leggyorsabb módja az alkatrész-programozásnak, a programozás sebessége csak az integrált áramkörtől függ. Néhány beszállító cég ingyen vagy nagyon kedvező áron vállalja, hogy előre beprogramozza a termékeit. Ezzel rengeteg idő megtakarítható, mindazonáltal a módszer nagyon rugalmatlan. Csak akkor alkalmazható, ha nincs variáns-specifikus szoftver, és a szoftveres változások valószínűtlenek. A chip szintű programozást két módon lehet megvalósítani:

1. Inline – az alkatrész-programozási folyamat az SMT soron belül történik a beültető gépre szerelhető speciális eszköz segítségével;
2. Offline – az alkatrész programozása egy külön erre a célra dedikált gépben történik a gyártósoron kívül.

Mindkét programozási folyamatnak vannak előnyei és hátrányai is. Az inline programozás kevesebb helyet foglal, olcsóbb és ciklusideje is kisebb, mint az offline programozá-

sé, de nagy hátránya, hogy csak üres alkatrész programozására alkalmas és a tartalom ellenőrzése nehézségeket okoz.

4.1.2. Rendszerszintű programozás (In-System Programming – ISP)

Rendszerszintű programozásról akkor beszélünk, ha a céleszköz annak felprogramozásakor már be van ültetve a nyomtatott áramkörbe. Ebben az esetben a nyomtatott áramkör tervezési korlátai miatt a programozás sebessége lényegesen lassabb, mint CLP esetében. ISP alkalmazásának feltétele, hogy ezt a készülék valamilyen formában támogassa. Például nyomtatott áramköri szinten a tesztelési pontokon, vagy tesztelés alatt CAN buszon keresztül lehetőség legyen az alkatrész felprogramozására. Az ISP legnagyobb előnye a rugalmasság, alkalmazása esetén lehetőségünk van arra, hogy a gyártás utolsó szakaszáig módosítsuk a termék kimenetelét, hiszen vannak olyan termékek, melyek csak szoftverben különböznek.

4.2. RoadRunner [21]

A RoadRunner (4.2. ábra) egy inline alkatrész-programozó és -kezelő berendezés, melyet közvetlenül az SMT beültető gépekre lehet felszerelni. Üres alkatrészek és előkészített programozási instrukciók felhasználásával a RoadRunner megfogja, felprogramozza és elszállítja a terméket az SMT beültető gép alkatrészfelvevő régiójába.

4.2. ábra. ProLine-RoadRunner [21]

4.2.1. A RoadRunner tartozékai

- Kezelőpanel: a RoadRunner felhasználói felülete, mely vezérlési funkciókat lát el nyomógombok és egy kijelző segítségével.
- Alkatrésztároló: az üres alkatrészek tárolására használt szalagtekercs.
- Szalag-bemeneti modul: feladata a szalagtekercs léptetése és felvágása a perforáció mentén.
- Szalag-szétválasztó modul: célja a szalagtekercs borítójának levétele.
- Négyszondás szállítófej: egy linárisan mozgó PNP fej, melynek feladata az alkatrészek szállítása a RoadRunneren belül.

- Programozó modul: feladata, hogy adatokat írjon az eszközökre. Négy foglalattal, más néven adapterrel rendelkezik.
- Szállító modul: a sikeresen programozott alkatrészeket a szállítófej a programozó modulból ide helyezi. Tartozik hozzá egy futószalag és egy szenzor, mely érzékeli, ha az alkatrész elérte az SMT-beültető gép felviteli pontját.
- Selejttároló: azokat az alkatrészeket, amelyeket nem sikerült felprogramozni, a szállítófej ebbe a tárolóba dobja.

4.3. A RoadRunner vezérlése a FIS segítségével

4.3.1. A FIS áttekintése

A FIS (Factory Integration Software) egy webszolgáltalás (webservice), amely lehetővé teszi olyan szoftverek készítését, amelyek segítségével mendzselhetjük gyártásba integrált RoadRunnereket. A FIS-t úgy tervezték, hogy minden programnyelven lehessen kliens-programot írni hozzá, amely támogatja a webszolgáltatásokat. A FIS három modult tartalmaz:

1. Konfigurációs modul (Configuration Module): segítségével tudunk menedzselni kívánt RoadRunnereket hozzáadni, eltávolítani, valamint a beállításaikat módosítani;
2. Ellenőrző modul (Auditor Module): segítségével lekérdezhetők és törölhetők bizonyos információk a gyártott termékekről;
3. Feladatvezérlő modul (Job Controller Module): segítségével irányítani lehet az összes berendezést, ami a konfigurációs modulhoz hozzá lett adva.

4.3.2. A feladatvezérlő modul

Feladatom elvégzése során a FIS moduljai közül csak a feladatvezérlő modult használtam. A fejezet további szakaszaiban ennek funkcióit és adattípusait tekinthetjük át.

4.3.2.1. A feladatvezérlő modul adattípusai

Az webszolgáltatás által használt adattípusok tartalmát a könnyebb áttekinthetőség érdekében táblázatos formába rendeztem. A táblázatok az F1. függelékben találhatók

4.3.2.1.1. Vezérlési adattípusok

- **DataIOResponse:** a webszolgáltatás minden metódusa a DataIOResponse objektum egy példányával tér vissza.

- **DataIOSystem:** a DataIOSystem tartalmazza a programozóval kapcsolatos konfigurációs információkat.

- **DataIOSystemInfo:** a DataIOSystemInfo egy programozó eszközzel kapcsolatos lényegi információkat tartalmaz.

4.3.2.1.2. Vezérlési adattípusok státuszai

- **DataIOHardwareStatus:** DataIOHardwareStatus típusa enum (felsorolás), a programozó lehetséges állapotait tartalmazza.

- **DataIOConnectionStatus:** DataIOConnectionStatus típusa enum, tartalmazza a FIS és a programozó eszköz közötti lehetséges kapcsolati állapotokat.

- **DataIOHardwareType:** az eszköz lehetséges típusait tartalmazó enum.

- **DataIOOperationStatus:** a műveletek lehetséges kimeneteit tartalmazó enum.

4.3.3. A feladatvezérlő modul funkciói

A vezérlési modul egyes funkcióinak magyarázata szintén egyszerűbb táblázatos formába rendezve. A táblázatok az F2. F2. F2. függelékben találhatók.

- **DownloadJob:** a DownloadJob segítségével lehet egy feladatot feltölteni a programozó belső tárhelyére.
- **DeleteAllJobsOnSystem:** a DeleteAllJobsOnSystem függvény segítségével lehet törölni az összes programozóra töltött munkamenetet.
- **StartJob:** a StartJob függvény segítségével lehet elindítani egy munkamenetet a programozón.
- **PauseJob:** a PauseJob függvény segítségével lehet egy munkamenetet szüneteltetni.
- **StopJob:** a StopJob függvény segítségével lehet egy munkamenetet leállítani.
- **ResumeJob:** a ResumeJob függvény segítségével lehet egy szüneteltetett munkamenetet újraindítani.
- **GetSystems:** a GetSystems függvény segítségével kérdezhetjük le, hogy milyen eszközök vannak hozzáadva a FIS-hez.
- **AdjustPassQuantity**: az AdjustPassQuantity függvény segítségével be tudjuk állítani, hogy egy feladat hány darab passos egység legyártásáig fusson.
- **ClearBelt:** a ClearBelt függvény a szállítószalag ürítési módjának módosítását szolgálja.
- **GetLastResponse:** egy művelet azonosítójának ismeretében lekérdezhető a művelet eredménye.
- **GetStatus:** a GetStatus függvény egy programozó jelenlegi állapotának lekérdezésére szolgál.
- **GetSystemInfo:** a GetSystemInfo függvénnyel lekérdezhetőek egy programozó rendszerinformációi.

5. Inline programozó állomás vezérlésének automatizálása

5.1. Bevezetés

Feladatom az SMT sorokon található programozó eszközzel, a RoadRunnerrel kapcsolatos szoftveres fejlesztési lehetőségek megvalósítása volt, melyek a következők:

1. A felprogramozott IC-k tartalmának ellenőrzése a sor megállítása nélkül;
2. Átállási idő minimalizálása a programozó automatizálásával, SMED módszer támogatása.

A feladat megvalósításához szükséges volt a jelenlegi állapotok és a SMED módszer megismerésére.

5.1.1. Tartalom ellenőrzése

Amennyiben nem ellenőrizzük gyártás közben, hogy a megfelelő adatok kerültek-e fel az IC-kre, akár több száz vagy ezer terméket gyárthatunk le hibásan, és beültetés után azokat már nem, vagy csak nagy időbefektetéssel lehet újraprogramozni. Az ellenőrzés korábbi lefolyása nagyon kezdetleges. A RoadRunnert be lehet állítani, hogy minden munkamenet indításakor az első sikeresen programozott egység tartalmát kiolvassa és feltöltse a saját tárhelyére. A Continental központilag minden terméktípushoz kibocsát egy referencia fájt, mely tartalmazza az IC elvárt adatait is. Az ellenőrzésre azt a módszert alkalmazzák, hogy Total Commander segítségével összehasonlították a két fájlt, és ha azt látták, hogy a tartalom bizonyos részei egyeznek, akkor megfelelőnek ítélték. Ezzel a módszer nagyon sok a probléma. Egyrészt nem garantált, hogy észreveszik a hibát, másrészt senki sem figyelmezteti az operátorokat a művelet elvégzésére, így a folyamat akár ki is maradhat, és csak az SMT sor utáni ellenőrzésnél derül ki a probléma.

5.1.2. A RoadRunner kezelése

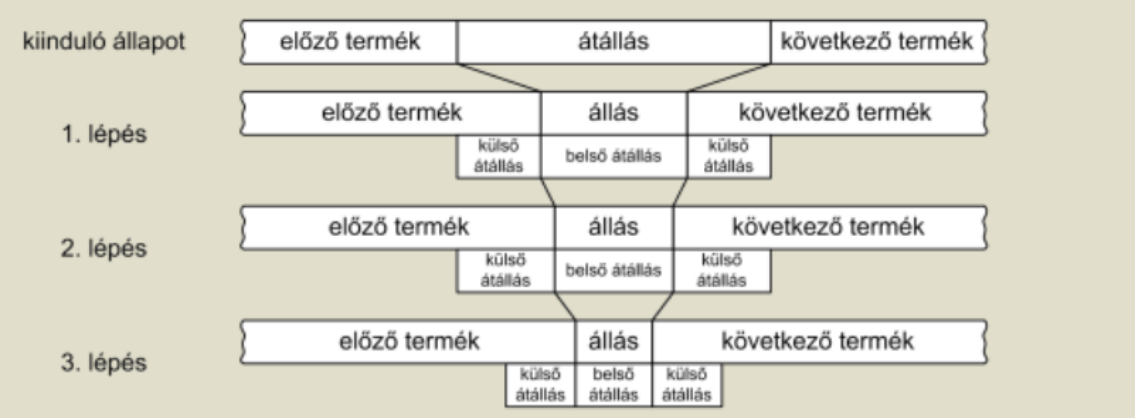
A programozó kezelése jelenleg manuálisan történik. Az operátorok termékváltáskor először leállítják a RoadRunnert, és megnézik a gyártási tervben, hogy milyen termék gyártása következik. Ezek után egy szoftver segítségével kiválasztják az induló terméknek megfelelő munkamenetet és feltöltik a RoadRunnerre, majd átállítják az IC-ket befogó adapter szélességét a terméknek megfelelően, és elindítják a feladatot. Ez a folyamat így nagyon lassú, és sok helyen tartalmaz olyan elemeket, ahol nagyon könnyű hibát elkövetni. Ilyen elem például az anyagszám kiválasztása. Az egyes termékek anyagszámai lehetséges, hogy csak egy-két karakterben térnek el egymástól. Így átálláskor előfordulhat, hogy nem a megfelelő munkanetet választják ki.

5.1.3. SMED [19]

Az átállási időnek azt az időintervallumot nevezzük, amely ahhoz szükséges, hogy a gépek bizonyos paramétereit vagy részeit átállítsuk vagy lecseréljük annak érdekében,

hogy egy másik termék gyártása megkezdődhessen. A SMED (Single Minute Exchange of Die) ennek az időnek drasztikus lecsökkentését célul kitűző módszer. Ha szó szerint akarjuk lefordítani, azt jelenti, hogy egy számjegyű - azaz kevesebb, mint 10 – percen belül kell tudnunk átállni. A definíciót szabadon kell értelmeznünk, hiszen vannak olyan átállások, ahol nem lehet kitűzni célnak a 10 percet, mivel ez technológiai okokból nem elérhető, és vannak olyanok, ahol a módszer használata nélkül is át lehet állni 4-5 percen belül. A módszer lényege három pontban leírható:

1. Válasszuk szét az átállást külső és belső átállási műveletekre:
    a. belső átállás: az átállás azon része, melyet csak akkor lehet elvégezni, ha a gép már áll;
    b. külső átállás: az átállás azon része, melyet akkor is el lehet végezni, mikor a gép dolgozik.
2. Minden belső műveletet, amit csak lehet, helyezzünk át a külső átállásba.
3. Fejlesszük mind a külső, mind a belső átállást úgy, hogy az idejük minél jobban lecsökkenjen.



5.1. ábra. SMED lépései [19]

5.1.4. A szoftverek megvalósítása

A munkám során a feladatkiírásban szereplő két szoftvert valósítottam meg: az offline tesztelő állomást, és a teljes automatizálást végző szoftvert. Az utóbbi szoftver tartalmazza az előbbi minden funkcióját – a tesztelő alkalmazást csak azért kellett külön is megvalósítanom, mert annak kritikus volta miatt sokkal rövidebb határidőt követeltek meg –, így a továbbiakban a szoftverek funkciót csak a teljes vezérlőszoftveren keresztül ismertetem. A programokat C#, SQL, és XML nyelvek felhasználásával írtam.

5.2. Az átállási folyamatok felosztása

A SMED módszert szem előtt tartva fel kell osztanunk az átállási folyamatokat külső és belső átállásra. Az átállási folyamat lépései a következők:

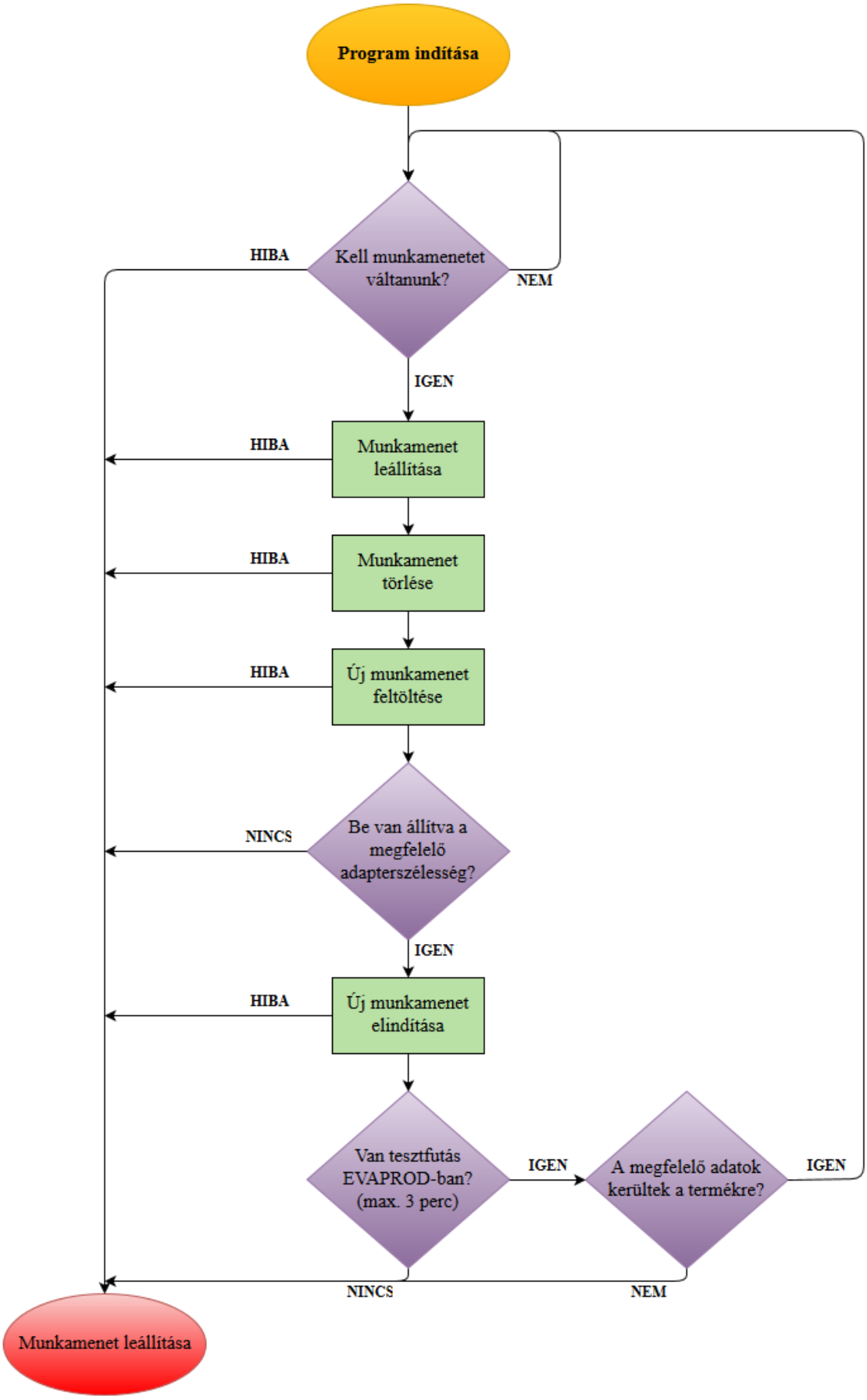
1. Előző munkamenet leállítása;
2. Munkamenet törlése;

3. Új feladat feltöltése;
4. Adapter szélességének módosítása;
5. Munkamenet elindítása.

Szigorúan véve ebben mindegyik folyamat belső átállásnak számít, hiszen a munkamenet leállításával kezdjük a folyamatot. Viszont ha belegondolunk abba, hogy szoftveres segítséggel azalatt, hogy valaki átszereli az adapter szélességét és kicseréli a szalagtárat, minden más előkészítő műveletet el tudunk végezni, tekinthetjük úgy, hogy csak az adapter szélességének módosítása belső átállás.

5.3. A teljes munkamenet leírása

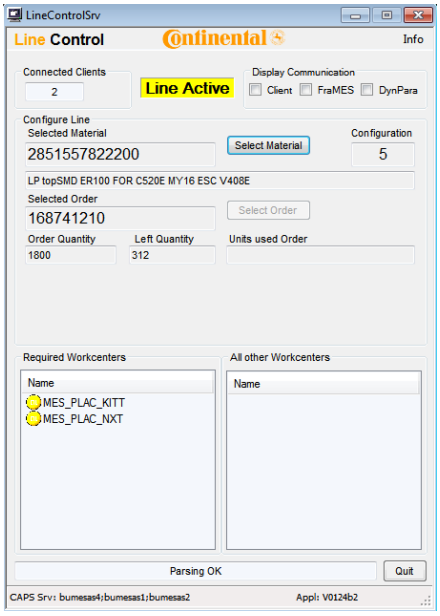
A szoftvernek elindítása után folyamatosan figyelnie kell, hogy épp milyen termék gyártása megy a soron. Amennyiben a RoadRunneren nem ennek megfelelő termék programozása folyik, leállítja annak működését, és törli a munkamenetet. Ez utóbbira valójában nem lenne mindig szükség, de mivel abban az esetben, ha a következő feladatnak nincs elég hely a RoadRunner tárhelyén, hibás működés léphet fel, praktikus megoldásnak találtam mindig elvégezni a folyamatot. Ezután következik az új munkamenet felöltése, ami után a program megáll, és operátori engedélyt kér a folytatásra. Az engedélykérésre azért van szükség, mert sajnos nincs mód arra, hogy szoftveresen ellenőrizzük, hogy a megfelelő adapterszélesség van-e beállítva. Ha az operátori engedélyt megkapjuk, elindítjuk a munkamenetet, majd elkezdjük figyelni, hogy került-e fel tesztfutás EVAPROD-ba. Mikor azt érzékeljük, hogy felkerült a tesztfutás, ellenőrizzük, hogy a megfelelő adatokat írtuk-e az IC-re. Jelenlegi koncepció szerint minden egyes hiba – továbbá egyes folyamatoknál bizonyos idő eltelte is – a munkamenet leállását eredményezi. A vezérlés folyamatábrája az 5.2. ábrán látható. A fejezet további részeiben kifejtem az egyes folyamatok, a felhasznált modellek, a konfiguráció és a naplózás megvalósítását.



5.2. ábra. A vezérlés folyamatai

5.4. A terméktípus lekérdezése

A folyamat első lépéseként le kell tudnunk kérdezni az éppen gyártani kívánt termék anyagszámát. A budapesti gyár szinte minden gyártósorán megtalálható a LineControl nevű MES szoftver, mely az adott soron éppen futó megrendeléssel kapcsolatban tartalmaz információkat (5.3. ábra). Tartalmazza többek között a megrendelés azonosítóját, a legyártani kívánt darabszámot és a termék anyagszámát.



5.3. ábra. A LineControl felhasználói felülete

5.4.1. A LineControllal való kommunikáció megvalósítása

A LineControl TCP/IP protokollstruktúrán keresztül képes egyszerű üzeneteket küldeni, és fogadni. A kommunikációt a LineControlHelper nevű osztályban valósítottam meg. A LineControlHelper egy TcpIpHandler nevű modult használ fel, mely képes TCP/IP üzeneteket küldeni és fogadni a szerver (jelen esetben a LineControl) ip címe és a kommunikációs port ismeretében. A TcpIpHandler a Continental által központilag használt .NET-es sablon része, melyről a fejezet későbbi szakaszában ejtek szót. A TcpIpHandler konfigurációs szekciója az 5.4. ábrán látható. Csak azokkal az eszközökkel képes kommunikálni, melyek abban megtalálhatók.

```
<TcpIpHandler1 LogEvents="true" LogDirectory="Logs/TcpIpEvents" LogSize="500" AutoInit="true">
    <Clients>
        <add Alias="SMT10_LineControl" Port="20001" Ip="172.20.120.73" />
        <add Alias="SMT11_LineControl" Port="20001" Ip="172.20.124.211" />
        <add Alias="SMT12_LineControl" Port="20001" Ip="172.20.126.30" />
        <add Alias="SMT13_LineControl" Port="20001" Ip="172.20.120.159" />
    </Clients>
</TcpIpHandler1>
```

5.4. ábra. TcpIpHandler konfigurációs szekciója

A LineControlHelper osztályom példányosításakor két paramétert kell átadni: az elérni kívánt LineControl nevét (5.4. ábrán Alias mező) és egy IEventAggregator nevű interface-t, mely az események kezelésére szolgál. A LineControlHelper egyetlen üzenet elküldésére, és az arra kapott válasz értelmezésére képes. Az elküldött üzenet a következő: „<STX>TS_GET:MAT<ETX>”. Az STX és az ETX az üzenet kezdetét és végét jelölő vezérlőkarakterek (start of text, end of text). A „TS_GET:MAT” a tényleges üzenet, mellyel gyakorlatilag megkérjük a LineControlt, hogy küldje el nekünk az éppen futó megrendelés anyagszámát. A válaszüzenet a következőképp néz ki: „<STX>TS_GET:MAT:ANSWER<ETX>”. Az „ANSWER” szó helyén optimális esetben az anyagszám van, amennyiben valami hiba történt, a válaszüzenet az „ERROR” szóval kezdődik. A LineControlHelper anyagszámot lekérdező függvénye az 5.5. ábrán látható. Visszatérési értéke az anyagszámot tartalmazó string, hiba esetén null.

```
2 references
public string GetLineControlMaterial()
{
    this.LineControlAnswer = string.Empty;
    this.EventAggregator.GetEvent<TcpIpClientSendMessageEvent>().Publish
        (new TcpIpClientSendMessageEventArgs(this.LineControlClientAlias, Encoding.ASCII.GetBytes(STX + "TS_GET:MAT" + ETX)));
    int waitTime = 200;
    for (int i = 0; i <= LocalModuleConfiguration.Instance.LineControlTimeOutMSec; i += waitTime)
    {
        if (!string.IsNullOrEmpty(this.LineControlAnswer))
        {
            this.LineControlAnswer = this.LineControlAnswer.Replace("<STX>", "");
            this.LineControlAnswer = this.LineControlAnswer.Replace("<ETX>", "");
            this.LineControlAnswer = this.LineControlAnswer.Replace("LC_GET:MAT:", "");
            if (this.LineControlAnswer.StartsWith("ERROR"))
            {
                return null;
            }
            return this.LineControlAnswer;
        }
        Thread.Sleep(waitTime);
    }
    return null;
}
```

5.5. ábra. LineControlHelper osztály anyagszámot lekérdező függvénye

5.5. Tesztfutás lekérdezése

A RoadRunner minden egység felprogramozása után „naplót vezet”, azaz egy logfájlba elmenti, hogy mikor, melyik programozó milyen terméket gyártott, és milyen eredménnyel. Ezeket az eredményeket egy szoftver feltölti a MES EVAPROD adatbázisába. Lévén egy egyszerű tesztfutás lekérdezéséről van szó, jogosan feltételezhetnénk, hogy az EVAPROD klienst használva könnyedén megkaphatjuk a választ. Ez általában véve így is van, csakhogy mi ebben az esetben nem ismerjük a tesztfutás azonosítószámát, csak a termék típusát és a teszterv nevét, az EVAPROD kliens pedig nem nyújt lehetőséget arra, hogy ezekből az adatokból információkat tudjunk meg az esetleges tesztfutásokról. Így az egyetlen lehetőségem az volt, hogy direkt lekérdezést írok az EVAPROD adatbázishoz, melynek eredménye az 5.6. ábrán látható IsTestrunAvailable nevű függvény. A függvénynek két bemeneti paramétert kell megadni, az egyik a teszterv neve – mely a RoadRunnereknél a budapesti gyárban „RR_FLASH”, de konfigurációs szekcióban megadható (erről később) –, a másik teszterv csoportja, mely központi megállapodás alapján mindig az adott termék anyagszáma. A lekérdezéshez három adattáblát kell összekapcsolnunk. Az egyik tartalmazza magát a tesztfutást, a másik kettővel pedig hozzá tudjuk kapcsolni teszterv nevét és a termék anyagszámát. A függvény bool típus-

sal tér vissza, melynek értéke true, ha talált tesztfutást az elmúlt 1 percben, és false, amennyiben nem talált.

```
1 references
public bool IsTestrunAvailable(string testplanGroup,string testplanName)
{
    DateTime currentTime = DateTime.Now.AddMinutes(-1);
    String currentTimestr = currentTime.ToString("yyyy.MM.dd HH:mm:ss");
    List<Dictionary<String, Object>> listResult = new List<Dictionary<String, Object>>();

    try
    {
        listResult = DB.DbInstances["MESBU"].CustomSelect(
                String.Format(@"SELECT
                                RUN.RUNID ID,
                                RUN.RUNID_TYPE TYPE,
                                TO_CHAR(RUN.RUN_DATE,'YYYY.MM.DD HH24:MI:SS') DATE_TIME,
                                RUN.RUN_STATE RESULT,
                                PAG.PRP_BEZ TP_GROUP,
                                PAG.PRP_VAR TP_NAME,
                                AUF.PDK_AUFTR JOB
                                FROM
                                  PD_LFD_RUN RUN, -- tesztfutásokat tartalmazó tábla
                                  PD_LFD_AUF AUF, -- job a tesztfutáshoz
                                  PD_LFD_PAG PAG -- név a tesztfutáshoz

                                WHERE
                                  RUN.PRD_PAG_SID = PAG.PRD_PAG_SID
                                  AND RUN.PRD_SPC_SID = AUF.PRD_SPC_SID

                                  AND PAG.PRP_BEZ = '{0}' -- testplan group
                                  AND PAG.PRP_VAR = '{1}' -- testplan név

                                  AND RUN.RUN_DATE > TO_DATE('{2}','YYYY.MM.DD HH24:MI:SS') -- from

                                ORDER BY TP_GROUP,TP_NAME, DATE_TIME", testplanGroup, testplanName, currentTimestr));
    }
    catch (Exception ex)
    {
        throw ex;
    }

    return (listResult!= null);
}
```

5.6. ábra. IsTestRunAvailable függvény direkt EVAPROD lekérdezéssel

5.6. A tartalom ellenőrzése

5.6.1. Összehasonlítandó fájlok elérése

A tartalomellenőrzés megvalósításához szükség van a RoadRunner által felprogramozott tartalom kiolvasására, valamint egy referenciafájlra, amivel össze lehet azt hasonlítani. A RoadRunner az új munkamenet indítását követően kiolvassa az első IC tartalmát, és feltölti a saját tárhelyére, így az FTP-n (File Transfer Protocol) keresztül elérhető. A referenciafájlt pedig a Continental központilag biztosítja minden anyagszámhoz. Az FTP-n lévő fájl betöltésére a budapesti MES csapatnak már volt egy megoldása, amit annyiban kellett módosítani, hogy string típus helyett byte tömbben is képes legyen visszaadni az adatokat. Erre azért volt szükség, mert sok bináris kombinációt a string típus nem tud értelmezni, és ezeket a helyeket ’?’ karakterekkel tölti fel, melyek ellehetetlenítik az ellenőrzést. A referenciafájlok egy hálózati meghajtón találhatóak. Ezeket a System.IO namespace-ben található FileStream és File osztályok segítségével töltöm be a programba.

5.6.2. Az ellenőrzési idő optimalizálása

Az első dolog, ami a két összehasonlítandó fájllal kapcsolatban szemet szúrhat, hogy a méretük nem egyezik meg. A központilag biztosított referenciafájl specifikációjából kiderül, hogy az három részből tevődik össze: Flashloader + Test software + Application software. Nekünk csak a Flashloader részre van szükségünk.

FileStream osztály használata esetén előre meg kell adni, hogy hány byte-ot szeretnénk beolvasni. Ennek megállapítására a RoadRunner által kiolvasott program tartalmát vizsgáltam meg. A fájl számunkra fontos tartalma a 0x8000 hexadecimális címen kezdődik, a végét pedig ez a három byte jelzi a megadott sorrendben: 0x06, 0x01, 0x52. Az 5.7. ábrán látható rövid kódrészlet valósítja meg RoadRunner tartalmának kiolvasását és a fájl végének megkeresését. A tartalom elérési útvonala a konfigurációs fájlban módosítható.

```
//RR tartalom beolvasása
FtpHelper ftpHelper = new FtpHelper(path, RoadRunnersInfo[i].RoadRunner.FtpUser, RoadRunnersInfo[i].RoadRunner.FtpPass);
byte[] roadRunnerContent = ftpHelper.ReadBinaryFile("IMAGE.bin");

//File végének megkeresése
int endAddress = 0;
byte[] endBytes = new byte[] { 0x06, 0x01, 0x52 };
for (int p = roadRunnerContent.Length; p > 2; p--)
{
    if (roadRunnerContent[p - 2] == endBytes[0] && roadRunnerContent[p - 1] == endBytes[1] && roadRunnerContent[p] == endBytes[2])
    {
        endAddress = p - 4;
        break;
    }
}
```

5.7. ábra. RoadRunner tartalom kiolvasás, és a fájl végének megkeresése

A RoadRunner által kiolvasott tartalom kezdő és végcímének ismeretében már megoldható a referenciafájl beolvasása (5.8. ábra). A referenciafájlok elérési útvonala szintén módosítható a konfigurációs fájlban. Az adott elérési útvonalon belül pedig úgy tudjuk megtalálni, hogy a neve mindig a „termék típusa + .bin” formátumot követi.

```
path = this.Config.SampleFilePath + MaterialNumber + ".bin";

int startAddress = 0x8000;

int count = endAddress - startAddress + 1;
byte[] flashContent = new byte[count];
using (FileStream fileStream = File.OpenRead(path))
{
    fileStream.Position = startAddress;
    fileStream.Read(flashContent, 0, count);
}
```

5.8. ábra. A referenciafájl beolvasása

A két fájl összevetéséhez bájtonként vizsgálom a két fájl tartalmát mindaddig, amíg azok megegyeznek, vagy a fájlok végére nem érek. Ha a fájlok végéig elérek, a két fájl tartalma azonos, ha korábban kilépek az összehasonlítási ciklusból, akkor pedig különböző (5.9. ábra).

```
int sameContent = 0;
for (int z = 0; z < count; z++)
{
    if (flashContent[z] == roadRunnerContent[z + startAddress])
    {
        sameContent++;
    }
    else { break; }
}

if (sameContent == count)
{
    this._mainDispatcher.BeginInvoke(new Action(() =>
    this.RoadRunnersInfo[i].ControlResult = Lang.IsUpToDate));
    enableResult = true;
```

5.9. ábra. Tartalom összehasonlítás

5.7. A RoadRunner vezérlése

Amennyiben az anyagszám lekérdezése sikeres, a következő lépés a RoadRunner-el való kommunikáció megvalósítása. A 4. fejezetben már tárgyaltuk, hogy a RoadRunner vezérlése egy FIS nevű webszolgáltatás segítségével oldható meg.

5.7.1. Webszolgáltatás hozzáadása a projekthez

A webszolgáltatást, annak használata előtt hozzá kell adnunk a projektünkhöz referenciaként. Ezt úgy tudjuk megtenni, hogy a projekt referenciáinál kiválasztjuk az „Add Service Reference” opciót, és a felugró ablak „Address” mezőjébe beírjuk a webszolgáltatás elérési címét, majd elnevezzük, és rákattintunk az „OK” gombra. A feladatvezérlő modul elérési címe a következő: http:// + gépnév vagy IP cím + port (tipikusan 9000) + „/dataio/pcs/jobcontroller”. Amennyiben a hozzáadás sikeres, a szolgáltatás referencia megjelenik a „Service Reference” mappában az általunk megadott néven, valamint generálódik mellé egy konfiguráció file is, mely 5.10. ábrán látható.

```
<startup>
    <supportedRuntime version="v4.0"
       sku=".NETFramework,Version=v4.0,Profile=Client"/>
</startup>
<system.serviceModel>
    <bindings>
        <basicHttpBinding>
            <binding name="BasicHttpBinding_IDataIOJobController" />
        </basicHttpBinding>
    </bindings>
    <client>
        <endpoint address="http://budts0906:9000/dataio/pcs/jobcontroller"
            binding="basicHttpBinding" bindingConfiguration="BasicHttpBinding_IDataIOJobController"
            contract="ServiceReferenceJobController.IDataIOJobController"
            name="BasicHttpBinding_IDataIOJobController" />
    </client>
</system.serviceModel>
</configuration>
```

5.10. ábra. A webszolgáltatás automatikusan generált konfigurációs szekciója

5.7.2. A webszolgáltatás használata

Ahhoz, hogy a webszolgáltatás függvényeit használni tudjuk, példányosítanunk kell a feladatvezérlő klienst az 5.11. ábrán látható módon. A _jobControllerCClient egy privát DataIOJobControllerClient típusú változó.

```
_joBControllerCClient = new DataIOJobControllerClient("BasicHttpBinding_IDataIOJobController");
```

5.11. ábra. A feladatvezérlő kliens példányosítása

A webszolgáltatás használatát az 5.12. ábrán látható kódrészlettel mutatom be, melyen a StopJob és a GetLastResponse nevű FIS függvény használata látható. Ha a LineControllon található anyagszám nem egyezik meg az aktuálisan gyártott anyagszámmal, akkor a munkamenet futását le kell állítanunk. Ehhez a sorhoz kapcsolt minden egyes RoadRunnerre meg kell hívnunk a StopJob függvényt. Mint azt a 4. fejezetben tárgyaltuk, a webszolgáltatás legtöbb függvénye aszinkron, ezért a következőképp kell eljárnunk. Ha a webszolgáltatás által válaszként adott műveleti státusz „Pending” – azaz függőben lévő –, akkor várunk fél másodpercet, és újra lekérdezzük az művelet státuszát, melyet úgy tehetünk meg, hogy a GetLastResponse függvényt meghívjuk az adott válasz azonosítójával. A GetLastResponse függvény meghívását egészen addig megismételjük, míg válaszként „Success”-t vagy „Failure”-t nem kapunk. Amennyiben a válasz az utóbbi, hibát jelzünk, amennyiben nem, a következő folyamatba léphetünk.

```
if (this.MaterialNumber != linecontrolmaterial)
{
    MaterialNumber = linecontrolmaterial;
    foreach (var roadrunnerinfo in RoadRunnersInfo)
    {
        _dataIOResponse = _joBControllerCClient.StopJob(roadrunnerinfo.DataIOSystem);
        while(_dataIOResponse.Status.Equals(DataIOOperationStatus.Pending))
        {
            Thread.Sleep(500);
            _dataIOResponse = _joBControllerCClient.GetLastResponse(_dataIOResponse.ID);
        }
        if(_dataIOResponse.Status.Equals(DataIOOperationStatus.Failure))
        {
            throw new Exception("An error occurred while FIS.Stopjob() in RunningJob.");
        }
    }
    _processState = ProcessState.DeleteJob;
}
```

5.12. ábra. Futó munkamenet leállítása

5.8. Állapotgép megvalósítása

A RoadRunner vezérlését egy állapotgép megvalósításának segítségével oldottam meg. Létrehoztam egy enum típust, mely a következő elemeket tartalmazza: RunningJob, StopJob, DeleteJob, DownloadJob, SettingsQuestion, StartJob, IsTestRunAvailable és Compare. Minden egyes enumhoz tartozik egy hasonló nevű függvény is. A vezérlés elindításakor a program végtelen ciklusba kerül, melyből csak hiba vagy szándékos leállítás esetében lép ki. Az állapotgép (5.14. ábra) egyes állapotainak funkciója:

1. **RunningJob:** A program mindig RunningJob állapotból indul. A program ebben a szekcióban ellenőrzi, hogy a LineControl-on lévő anyagszám és a jelenleg gyártott termék anyagszáma megegyezik-e. Amennyiben igen, maradunk ebben az állapotban, és legközelebb 10 másodperc múlva végzünk ellenőrzést. Ameny-

nyiben nem, leállítjuk a jelenleg futó programot, és az állapotot átváltjuk DeleteJob-ra (5.12. ábra).

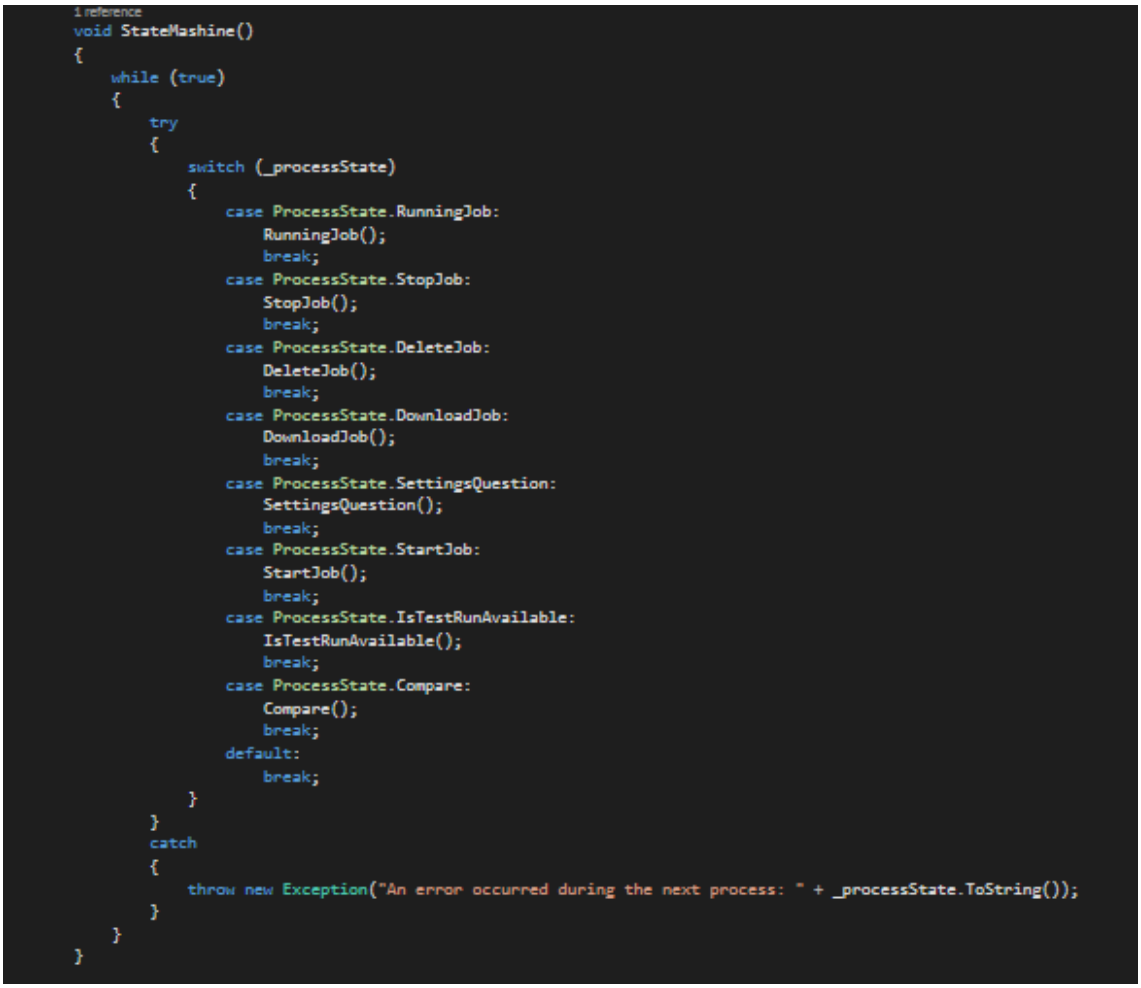
2. **DeleteJob:** A DeleteJob szekcióban töröljük a RoadRunner tárhelyén lévő összes munkatervet, majd a művelet befejezése után az állapotot átváltjuk DownLoadJob-ra.
3. **DownLoadJob:** A DownloadJob (5.13. ábra) szekcióban történik az új munkamenet feltöltése a programozóra. A DownloadJob után a SettingQuestion állapot következik.

```
1 reference
private void DownloadJob()
{
    foreach (var roadrunnerinfo in RoadRunnersInfo)
    {
        _dataIOResponse = _joBControllerCClient.DownloadJob(roadrunnerinfo.DataIOSystem, Config.JobPath + MaterialNumber);
    }
    foreach (var roadrunnerinfo in RoadRunnersInfo)
    {
        _joBControllerCClient.GetSystemInfo(out _dataIOInfo, roadrunnerinfo.DataIOSystem);
        while (_dataIOInfo.HardwareStatus == DataIOHardwareStatus.PreparingSystem || _dataIOInfo.HardwareStatus == DataIOHardwareStatus.TransferringFiles)
        {
            Thread.Sleep(500);
            _joBControllerCClient.GetSystemInfo(out _dataIOInfo, roadrunnerinfo.DataIOSystem);
        }
    }
    _processState = ProcessState.SettingsQuestion;
}
```

5.13. ábra. DownloadJob függvény

4. **SettingQuestion:** A SettingQuestion a program egyetlen része, mikor operátori engedélyt kérünk a művelet folytatásához**.** Különböző IC-k esetén lehetséges, hogy át kell szerelni az azokat befogó adapter szélességét. Ahogy korábban már említettem, a RoadRunner nem képes annak megállapítására, hogy az adapterszélesség a terméktípusnak megfelelő-e. Rossz beállítás esetén a RoadRunner selejtet termel. Amennyiben megerősítésre kerül, hogy a szélesség megfelelő, az állapotot megváltoztatjuk StartJob-ra.
5. **StartJob:** A StartJob szekcióban történik a munkamenet elindítása. Amennyiben az indítás sikeres, az állapotot IsTestRunAvailable-re változtatjuk.
6. **IsTestRunAvailable:** Az IsTestRunAvailable szekcióban történik annak ellenőrzése, hogy került-e már fel tesztfutás az EVAPROD adatbázisba**.** Ebben az állapotban maximum 3 percet vagyunk, ha ennyi időn belül nem található tesztfutás, akkor valami probléma van, és leállítjuk a munkamenetet. Ha található tesztfutás, akkor az állapotot átállítjuk Compare-re.
7. **Compare:** A Compare szekcióban történik meg az IC tartalmának ellenőrzése. Tartalomegyezés esetén a RunningJob-ra állítjuk az állapotot, ellenkező esetben StopJobra.
8. **StopJob:** Ugyan nem volt minden egyes állapotnál külön kiemelve, de a hibák esetében a program mindig StopJob állapotba kerül, és az aktuális hibaüzenet megjelenése mellett kilép a futási ciklusból.

A RoadRunner az elindított állapot mellett rendelkezik még Pause és Resume funkciókkal, melyek segítségével bizonyos időre le lehet állítani a munkamenetet, és folytatni anélkül, hogy azt újra le kéne tölteni**.**



```
1 reference
void StateMashine()
{
    while (true)
    {
        try
        {
            switch (_processState)
            {
                case ProcessState.RunningJob:
                    RunningJob();
                    break;
                case ProcessState.StopJob:
                    StopJob();
                    break;
                case ProcessState.DeleteJob:
                    DeleteJob();
                    break;
                case ProcessState.DownloadJob:
                    DownloadJob();
                    break;
                case ProcessState.SettingsQuestion:
                    SettingsQuestion();
                    break;
                case ProcessState.StartJob:
                    StartJob();
                    break;
                case ProcessState.IsTestRunAvailable:
                    IsTestRunAvailable();
                    break;
                case ProcessState.Compare:
                    Compare();
                    break;
                default:
                    break;
            }
        }
        catch
        {
            throw new Exception("An error occurred during the next process: " + _processState.ToString());
        }
    }
}
```

5.14. ábra. Az állapotgép megvalósítása

5.9. A felhasznált .NET sablon bemutatása

A Continental mindent megtesz annak érdekében, hogy a gyárai az egész világon a gyártás minden területén központilag meghatározott standardok szerint működjenek. Lehetőség szerint minden problémát úgy kell megoldani, hogy az a világ összes gyárában használható legyen. A MES szoftverek egységesítésére létrehoztak egy sablont FraMES.NET Client Shell néven, melyben elő vannak készítve a program azon részei, melyeket az esetek többségében használni kell.

5.9.1. A solution (megoldás) felosztása

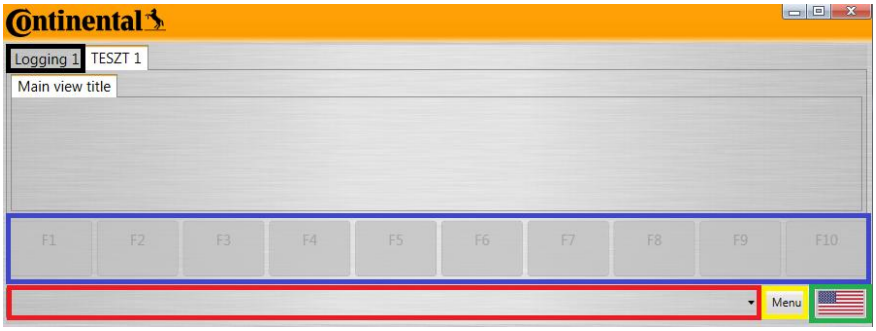
A Microsoft Visual Studioban solution-nek nevezik a több projektet összefogó struktúrát. A solution négy réteget tartalmaz:

1. Business layer (üzleti réteg);
2. Dummy business layer (színlelt üzleti réteg);
3. Contracts layer (kapcsolati réteg);
4. Modules layer (modul réteg).

Az üzleti réteg tartalmazza a vállalati logikát, azaz a MES-es függvényhívásokat és adatbázis-lekérdezéseket. Jelen esetben tehát csak az EVAPROD-hoz írt direkt lekérde-

zést tartalmazza. A színlelt üzleti rétegnek csak a program fejlesztése során van feladata. Ugyanazokat a függvényeket valósítjuk meg benne, mint az üzleti rétegben, de a függvények kimenetelét mi szabjuk meg tesztelési céllal. A modul réteg tartalmazza az alkalmazásunk logikáját és az ahhoz szükséges modelleket, valamint a felhasználói felületeket és azok vezérlőelemeit. A kapcsolati réteg feladata, hogy megteremtse a kapcsolatot az üzleti és a modul réteg között.

5.9.2. A sablon felhasználói felülete



5.15. ábra. A sablon felhasználói felülete

A sablon felhasználói felülete az 5.15.-ös ábrán látható. A könnyebb érthetőség végett az említendő szakaszokat külön színekkel jelöltem meg. A kékkel jelölt kereten belül helyezkedhetnek el az egyes funkciókat ellátó nyomógombok, melyeket nem csak kattintással, hanem az F1-F10 nyomógombok segítségével is használhatunk. A feketével körülvett „Logging 1” feliratú fülre kattintva lehet átkapcsolni a naplózott bejegyzések megtekintésére. A pirossal körülvett lenyitható fül az ún. állapotsor. Itt jeleníthetjük meg a műveletek sikeres elvégzését, valamint az esetleges hibaüzeneteket is. Tartalma általában nagymértékben megegyezik a naplózott bejegyzéseknél láthatóakkal. A zölddel körülvett amerikai zászlóra kattintva választható ki a program nyelve. A felhasználónak a sárgával körülvett gombra kattintva lehetősége van arra, hogy kikapcsolja a naplózási fül megjelenítését.

5.9.3. Többnyelvű alkalmazás támogatása

A Continental standardja szerint minden alkalmazást angol nyelven kell megírni. Előfordulhat azonban, hogy az egyes szoftvereket olyanoknak kell használni, akik nem beszélnek idegen nyelven. Ennek a problémának a kiküszöbölésére tartalmaz a sablon egy LanguagePack nevű osztályt, ahol új tulajdonságok (property) hozzáadásával új nyelvi elemeket adhatunk a projekthez (5.16. ábra). A projekt fordításakor ezek az elemek egy XML (Extensible Markup Language, Kiterjeszthető Jelölő Nyelv) kiterjesztésű fájlba kerülnek. A fájl neve a következőképp tevődik össze: projekt neve + „lang.” + a nyelvre jellemző kétbetűs rövidítés (culture name). Ez az offline tesztelő állomás példáján szemléltetve: „RoadRunner.lang.en.xml”. Ahhoz, hogy a programban a magyar nyelv is választható legyen, le kell fordítanunk az XML elemeket, át kell írnunk a „CultureName” elemet „en”-ről „hu”-ra, valamint meg kell tennünk ugyanezt a változtatást a fájl nevével is (5.17. és 5.18. ábra).

```
private string _fileNotFound;
2 references
public string FileNotFound
{
    get
    {
        return _fileNotFound = _fileNotFound ?? "File not found(download exception): {0}";
    }
    set
    {
        _fileNotFound = value;
    }
}


private string _directoryNotFound;
1 reference
public string DirectoryNotFound
{
    get
    {
        return _directoryNotFound = _directoryNotFound ?? "Directory not found: {0}";
    }
    set
    {
        _directoryNotFound = value;
    }
}
```

5.16. ábra. FileNotFound és DirectoryNotFound, két nyelvi property

RoadRunner.lang.en.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LanguagePack xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Descriptions />
  <CultureName>en</CultureName>
  <MainViewTitle>ROADRUNNER</MainViewTitle>
  <ModuleInstanceCreated>{0} instance created.</ModuleInstanceCreated>
  <CheckFiles>Check Files</CheckFiles>
  <ExceptionLongStr>Exception: {0}</ExceptionLongStr>
  <ReadMaterialSuccesful>Read material number succesfull</ReadMaterialSuccesful>
  <ReadMaterialFail>Read material number Fail!</ReadMaterialFail>
  <FileNotFound>File not found(download exception): {0}</FileNotFound>
  <DirectionNotFound>Direction not found: {0}</DirectionNotFound>
  <IsUpToDate>File is up to date.</IsUpToDate>
  <IsNotUpToDate>File is not up to date!</IsNotUpToDate>
  <IsNotComparable>Files are not comparable!</IsNotComparable>
  <Informations>Line: {0} , RoadRunner: {1} , MaterialNumber: {2} , Result: {3}</Informations>
  <EventLoggingException>Event logging exception.</EventLoggingException>
</LanguagePack>
```

5.17. ábra. A program angol nyelvi fájlja

RoadRunner.lang.hu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LanguagePack xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Descriptions />
  <CultureName>hu</CultureName>
  <MainViewTitle>ROADRUNNER</MainViewTitle>
  <ModuleInstanceCreated>{0} instance created.</ModuleInstanceCreated>
  <CheckFiles>Ellenőrzés</CheckFiles>
  <ExceptionLongStr>Exception: {0}</ExceptionLongStr>
  <ReadMaterialSuccesful>Anyagszám lekérdezése sikeres!</ReadMaterialSuccesful>
  <ReadMaterialFail>Anyagszám lekérdezése sikertelen!</ReadMaterialFail>
  <FileNotFound>Nem elérhető a következő file: {0}</FileNotFound>
  <DirectionNotFound>Nem található a következő elérési út egy része: {0}</DirectionNotFound>
  <Informations>Sor: {0} , RoadRunner: {1} , Anyagszám: {2} , Eredmény: {3}</Informations>
  <IsUpToDate>A tartalom egyezik</IsUpToDate>
  <IsNotUpToDate>A tartalom nem egyezik!</IsNotUpToDate>
  <IsNotComparable>Nem végezhető összehasonlítás!</IsNotComparable>
</LanguagePack>
```

5.18. ábra. A program magyar nyelvi fájlja

5.10. Konfigurációs elemek

Minden program alapkövetelménye a konfigurálhatóság. Jelen szoftvert öt SMT állomáson fogják használni, így minden egyes állomáshoz be kell tudnunk állítani a hozzá tartozó RoadRunnereket és LineControllokat. Minden soron általában két darab RoadRunner található, de ez lehet egy vagy több is, így ebből a szempontból is flexibilisnek kell lennie. A program megvalósítása közben arra a következtetésre jutottam, hogy a legjobb megoldás az, ha a programhoz több sort is hozzá lehet adni, és a program indításakor lehet eldönteni, hogy melyik sort szeretnénk kezelni. Így a különböző sorokon nem kötelező különböző konfigurációs fájlokat használni, ugyanakkor megadja rá a lehetőséget.

5.10.1. Modellek

A megvalósításhoz két interface-t hoztam létre, az egyiket IRoadRunner, a másikat ILine néven (5.19. ábra). Az előbbi egy RoadRunner tulajdonságait (név/ip, ftp host, ftp felhasználónév, ftp jelszó), az utóbbi pedig egy SMT sor tulajdonságait (név, hozzá tartozó RoadRunnerek, LineControl Alias) tartalmazza.

```
namespace FraMES.Client.Modules.RoadRunnerController1.Models
{
    3 references
    public interface ILine
    {
        3 references
        string Name { get;}

        3 references
        string LineControlClientAlias { get; }

        4 references
        IRoadRunner[] RoadRunnerArr { get; }
    }
}
```

```
namespace FraMES.Client.Modules.RoadRunnerController1.Models
{
    8 references
    public interface IRoadRunner
    {
        6 references
        string Name { get; }
        2 references
        string FtpHost { get; }
        2 references
        string FtpUser { get; }
        2 references
        string FtpPass { get; }

    }
}
```

5.19. ábra. ILine és IRoadRunner interface

5.10.2. Konfigurációs elemek

A feladat megvalósítása során három darab konfigurációs elemet hoztam létre. Egyet a RoadRunnereknek (RoadRunnerConfigurationElement), egyet az SMT soroknak (LineConfigurationElement) – mely tartalmazza a RoadRunnereknek készített elemet, és egyet az Oracle adatbázis TNS szolgáltatás nevének. A sablonban található egy LocalModuleConfiguration nevű osztály, melybe beleágyaztam mind a három általam készített konfigurációs elemet (5.20. ábra). Ennek végeredményeképp jött létre az 5.21. ábrán látható felosztás. A LocalModuleConfiguration osztály tartalmazza ezen kívül a SampleFilePath és a JobPath elemet – melyek a központi referencia fájl, és a RoadRunnerre feltöltendő munkamenetek elérési címeit tartalmazzák, valamint a LineControlTimeOutMSec nevű elemet is, mely a LineControl maximális válaszidejét jelenti.

```
8 references
public class LocalModuleConfiguration : ConfigurationSection
{

    private static LocalModuleConfiguration _instance;
    3 references
    internal static LocalModuleConfiguration Instance...


    [ConfigurationProperty("LineControlTimeOutMSec", IsRequired = true)]
    1 reference
    public int LineControlTimeOutMSec...


    [ConfigurationProperty("SampleFilePath", IsRequired = true)]
    1 reference
    public string SampleFilePath...


    [ConfigurationProperty("JobPath", IsRequired = true)]
    1 reference
    public string JobPath...


    [ConfigurationProperty("EvaDBInstance", IsRequired = true)]
    1 reference
    public ConnectionStringConfigurationElement DBInstance...


    [ConfigurationProperty("Lines", IsRequired = true)]
    1 reference
    public ConfigElementCollection<LineConfigurationElement> Lines...


}
```

5.20. ábra. LocalModuleConfiguration osztály tartalma

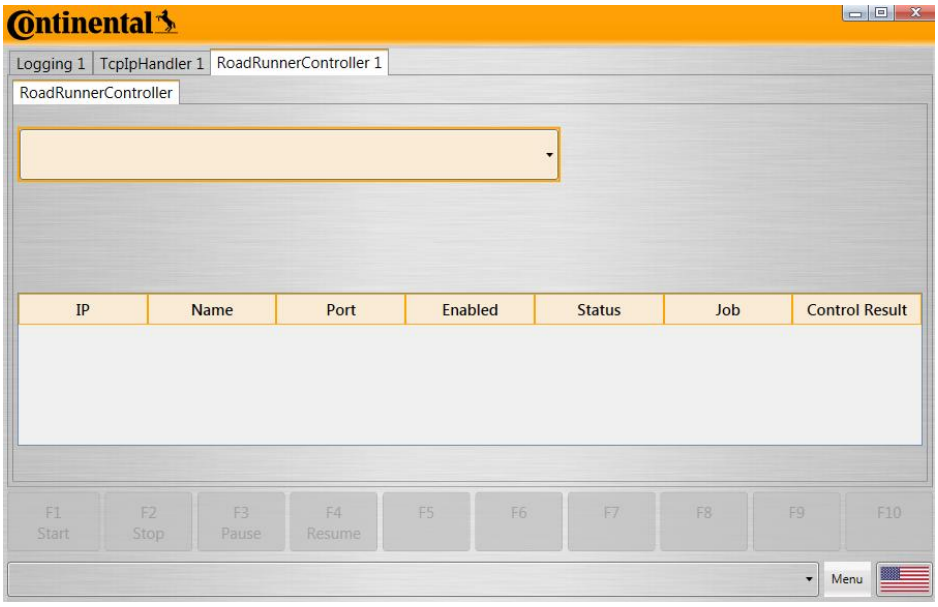
```
<RoadRunnerController1 JobPath="C:\Program Files\Data IO\FIS\FCSimulator\FDRROOT\Jobs\" LineControlTimeOutMSec="5000" SampleFilePath="Y:\">
    <!-- amelyik meghajtóra be van mappelve a "bufs026/fuji"-->
    <!--<RoadRunner1 LineControlTimeOutMSec="5000" SampleFilePath="C:\Users\uidp0245\Downloads\"> -->
    <EvaDBInstance>
        (DESCRIPTION =
            (ENABLE = BROKEN)
                (ADDRESS_LIST =
                (ADDRESS = (PROTOCOL = TCP)(HOST = budb048-vip.auto.contiwan.com)(PORT = 1521))
                (ADDRESS = (PROTOCOL = TCP)(HOST = budb049-vip.auto.contiwan.com)(PORT = 1521))
                (LOAD_BALANCE = no)
            )
            (CONNECT_DATA =
                (SERVER = DEDICATED)
                (SERVICE_NAME = REPORTING.AUTO.CONTIWAN.COM)
                    (FAILOVER_MODE =
                        (TYPE = session)
                        (METHOD = basic)
                        (RETRIES = 32)
                        (DELAY = 4)
                    )
            )
        )
    </EvaDBInstance>
    <Lines>
        <add Name="SMT10" LineControlClientAlias="SMT10_LineControl">
            <RoadRunners>
                <add Name="172.20.125.47" FtpHost="ftp://172.20.125.47//ide/FDRROOT/DUT_DUMP/" FtpUser="" FtpPass="" />
                <add Name="172.20.125.250" FtpHost="ftp://172.20.125.250//ide/FDRROOT/DUT_DUMP/" FtpUser="" FtpPass="" />
            </RoadRunners>
        </add>
        <add Name="SMT11" LineControlClientAlias="SMT11_LineControl">
            <RoadRunners>
                <add Name="172.20.120.75" FtpHost="ftp://172.20.120.75//ide/FDRROOT/DUT_DUMP/" FtpUser="" FtpPass="" />
                <add Name="172.20.120.76" FtpHost="ftp://172.20.120.76//ide/FDRROOT/DUT_DUMP/" FtpUser="" FtpPass="" />

            </RoadRunners>
        </add>
    </Lines>
</RoadRunnerController1>
```

5.21. ábra. A kész konfigurációs szekció

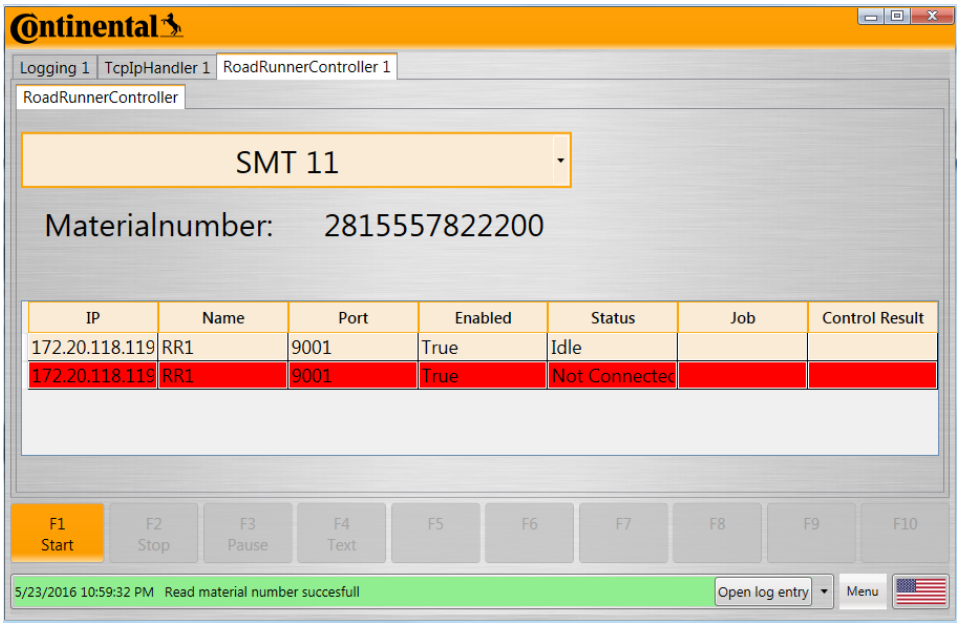
5.11. A felhasználói felület megvalósítása

Egy gyártásba készülő szoftver felhasználói felületének két feltételnek kell megfelelnie: legyen minél egyszerűbb, átláthatóbb, és pontosan annyi információt tartalmazzon, amennyit szükséges. Az 5.22. ábrán a szoftver kezdőlapja látható. A fenti legördülő menüre (combobox) kattintva tudjuk kiválasztani, hogy melyik sorhoz szeretnénk csatlakozni. Az alul található gombok az előző lépés elvégzéséig nem kattinthatók.



5.22. ábra. A kezdőlap

Miután kiválasztottunk egy SMT sort és lefutnak a fejezet korábbi szakaszaiban említett folyamatok, az 5.23. ábrához hasonló ablakot kell kapnunk. Az ábrán az látható, hogy az egyik RoadRunner feladatra várakozik, a másik pedig nincs csatlakoztatva a webszerverhez.



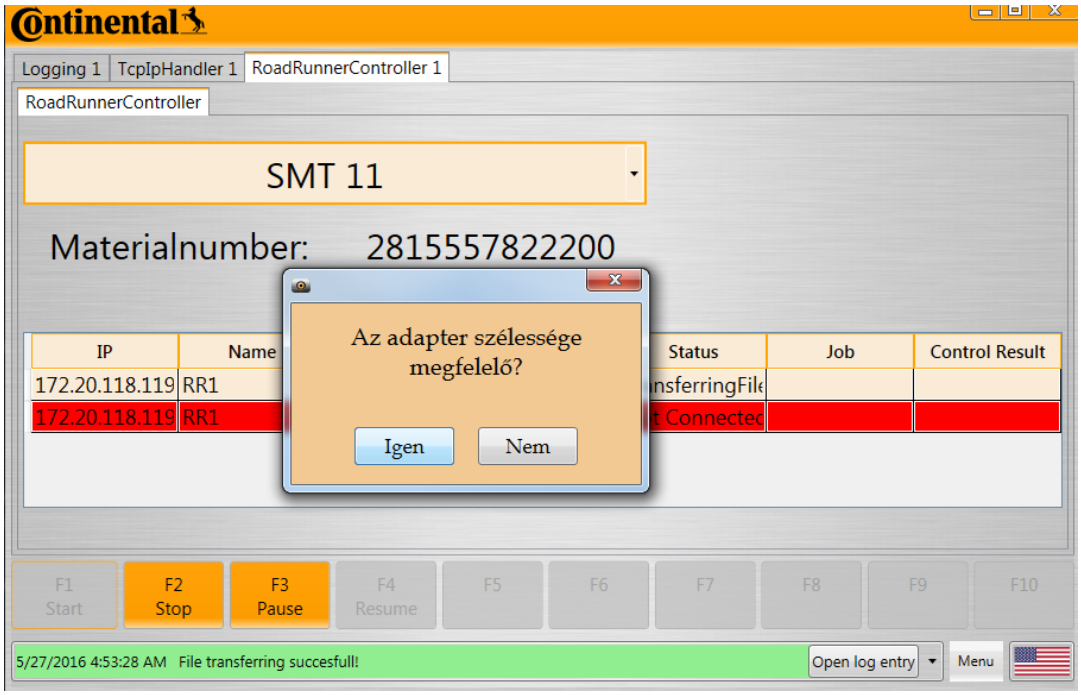
5.23. ábra. Az SMT sor kiválasztása utáni felület

A RoadRunnerek információt tartalmazó DataGrid mező XML kódját az 5.24. ábrán láthatjuk. Minden egyes adat feltöltését adat összekötéssel (data binding) oldottam meg. A nem csatlakoztatott RoadRunner piros háttere pedig az alul látható DataTriggernek köszönhető. Amennyiben a SomethingIsWrong tulajdonság értéke false, a háttér marad alapértelmezett, amennyiben értéke true, a háttér vörösre vált. Az 5.25. és 5.26. ábrán a

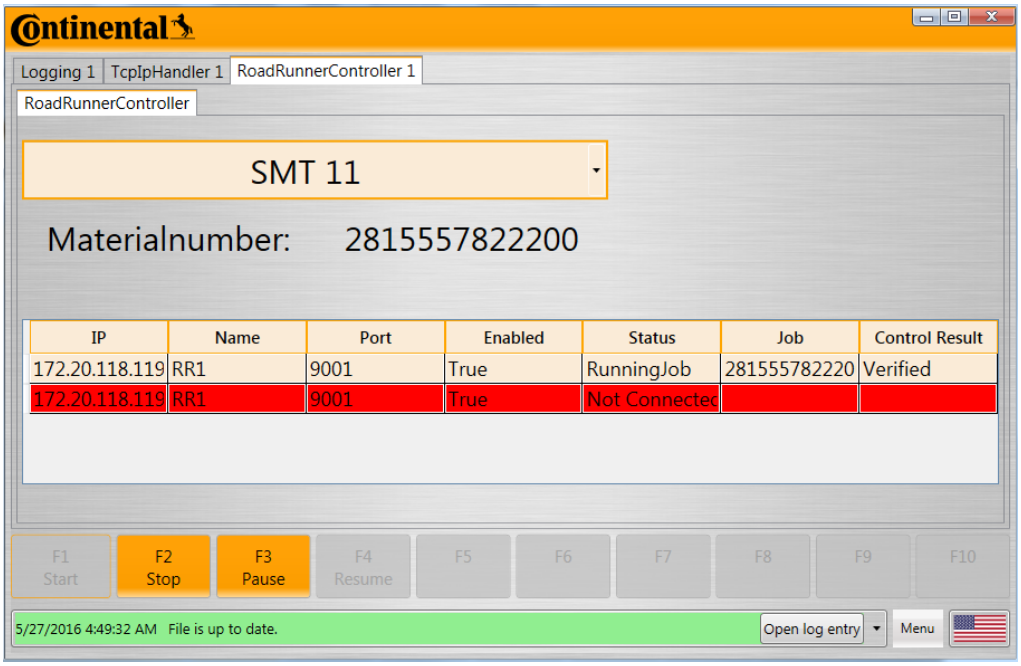
szoftver két további jellemző felületét láthatjuk: az operátori megerősítésre várakozást és futó munkameneti státuszban lévő ellenőrzött RoadRunnert.

```
<Grid Grid.Column="0" Grid.Row="1" Grid.ColumnSpan="2">
    <Grid.RowDefinitions>
        <RowDefinition Height="5*"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <DataGrid HorizontalScrollBarVisibility="Disabled" ItemsSource="{Binding RoadRunnersInfo, UpdateSourceTrigger=PropertyChanged}" AutoGenerateColumns="False">
        <DataGrid.Columns>
            <DataGridTextColumn Width="*" Binding="{Binding Ip}" Header="IP" />
            <DataGridTextColumn Width="*" Binding="{Binding Name}" Header="Name"/>
            <DataGridTextColumn Width="*" Binding="{Binding Port}" Header="Port"/>
            <DataGridTextColumn Width="*" Binding="{Binding Enabled}" Header="Enabled"/>
            <DataGridTextColumn Width="*" Binding="{Binding Status}" Header="Status"/>
            <DataGridTextColumn Width="*" Binding="{Binding Job}" Header="Job"/>
            <DataGridTextColumn Width="*" Binding="{Binding ControlResult,Mode=TwoWay}" Header="Control Result"/>
        </DataGrid.Columns>
        <DataGrid.ColumnHeaderStyle>
            <Style TargetType="{x:Type DataGridColumnHeader}">
                <Setter Property="Background" Value="AntiqueWhite"/>
                <Setter Property="BorderThickness" Value="1"/>
                <Setter Property="BorderBrush" Value="#FFA500"/>
                <Setter Property="HorizontalContentAlignment" Value="Center"/>
                <Setter Property="FontWeight" Value="DemiBold"/>
                <Setter Property="Height" Value="30" />
            </Style>
        </DataGrid.ColumnHeaderStyle>
        <DataGrid.CellStyle>
            <Style TargetType="{x:Type DataGridCell}">
                <Setter Property="FontSize" Value="18"/>
                <Setter Property="HorizontalContentAlignment" Value="Center"/>
                <Setter Property="Background" Value="AntiqueWhite"/>
                <Style.Triggers>
                    <DataTrigger Value="true" Binding="{Binding SomethingIsWrong}">
                        <Setter Property="Background" Value="Red"/>
                    </DataTrigger>
                </Style.Triggers>
            </Style>
        </DataGrid.CellStyle>
    </DataGrid>
</Grid>
```

5.24. ábra. DataGrid megvalósítása



5.25. ábra. Adapterszélesség megerősítése



5.26. ábra. Ellenőrzött RoadRunner

6. Értékelés, továbbfejlesztési lehetőségek

Munkám során megismertem az SMT gyártási folyamatait és a Continental Automotive Hungary Kft. budapesti gyárában alkalmazott traceability rendszert, valamint az alkalmazott alkatrész-programozási technológiákat. Az inline programozási folyamat fejlesztésére két alkalmazást készítettem el. Az offline tesztelőállomást, mely elősegíti a RoadRunner által programozott alkatrészek gyors ellenőrzését tartalmi szempontból, valamint a RoadRunner teljes automatizálását végző szoftvert.

Az offline tesztelőállomás tesztelési időszaka alatt nem merült fel probléma, így mindenhol kiváltotta a korábbi ellenőrzési folyamatokat. Teljes automatizálást végző szoftver egyelőre csak a termék gyártója által rendelkezésemre bocsátott szimulált környezetben volt tesztelve, de ott minden további nélkül megfelelt az elvárásoknak. Az elkészített szoftverek hozzájárulnak az emberi mulasztásból fakadó hibák minimalizálásához, és támogatják a termékek közötti gyors átállás megvalósulását.

6.1. Fejlesztési lehetőségek

6.1.1. Jobb hibakezelés

A szoftver jelenlegi legnagyobb fejlesztési lehetősége lehet a hibák specifikusabb kezelése, bizonyos esetekben a beavatkozás nélküli megoldása a programozó leállítása helyett. Azok az információk, hogy a gépnek és webszolgáltatásnak mik a gyakori, és a kevésbé gyakori hibái az éles környezetben végzett tesztelési folyamatokig nem állnak rendelkezésemre.

6.1.2. Több LineControl figyelése

Mint korábban említettem, a gyártási sorok főbb területeihez kapcsolódik LineControl, mely tartalmazza a jelenleg futó megrendelés adatait. Jelenlegi szoftverben csak kifejezetten a RoadRunnerhez tartozó LineControlt figyeltem. A RoadRunner előtti beültető géphez szintén tartozik egy LineControl, így lehetőségem nyílhat annak a figyelésére, és arra, hogy már azelőtt feltöltsem a következő munkamenetet, hogy azt a RoadRunnerhez tartozó LineControl megkövetelné. A folyamat elméleti hátterével vannak problémák, hiszen bár lehet több munkamenet egy RoadRunneren, és lehetőségünk is van arra, hogy feltöltsünk egy újat futás közben, de a tárhelye véges, és törölni csak úgy lehet, ha minden munkamenetet törlünk.

6.1.3. Folyamatfelelős értesítése e-mail üzenetben hiba esetén

A RoadRunner vezérlését elősegítő webszolgáltatás képes arra, hogy üzenetet küldjön (bizonyos feltételek esetén) a konfigurációs modulján keresztül megadott e-mail címekre. Habár erre eddig nem volt igény, és tapasztalatom alapján az egyes folyamatok felelősei nem szeretik azt, ha e-mailekkel bombázzák őket, de bizonyos kritikus hibák esetén megfontolandó lehet az alkalmazása.

6.1.4. Adapterszélesség ellenőrzése

Az adapterszélesség ellenőrzésének megteremtése az egyik legnagyobb fejlesztés lehetne. Hiszen ezzel olyan termékek átállása közben, ahol az elvárt adapterszélesség, vagy akár a teljes IC megegyezik, nem lenne szükség emberi beavatkozásra, ami a jelen folyamat legidőigényesebb része.

Ábrák jegyzéke

Irodalomjegyzék

[1] Óbudai Egyetem, Eleketronikai technológia jegyzet: Felületszerelt gyártástechnológia (Letöltés: 2016.04.17) http://uni-obuda.hu/users/grollerg/Elektronikaitechnologia/Jegyzet/2-Szerels-elm.pdf

[2] Ning Cheng Lee: Reflow Soldering Processes and Troubleshooting: SMT, BGA, CSP and Flip Chip Technologies, Newnes kiadó, 2001

[3] Pinkola János: Elektronikai technológia laboratórium, Műegyetem kiadó, 2007

[4] Clyde F. Coombs, Jr: Printed Circuits Handbook, 6th Edition, Mc Graw Hill kiadó, 2008

[5] Budapesti Műszaki Egyetem, Elektronikai technológia jegyzet 9. tétel: A furatba, illetve a felületre szerelhető alkatrészek megjelenési formái és típusai, 2009 http://vir.sch.bme.hu/dokumentumok/VillanySite/5.%20f%C3%A9l%C3%A9v/Elektronikai%20technol%C3%B3gia/elektrotech%2009-12%20tetel.pdf

[6] Budaesti Műszaki Egyetem, Elektronikai technológia IV. labor segédlete, Moduláramkör készítése újraömlesztéses felületszerelési (SMT) technológiával (2016.05.03) http://vik.wiki/images/f/f9/ETT_sillabusz4.pdf

[7] Németh Ádám: Finom raszterosztású alkatrészek forrasztott kötéseinek vizsgálata, Szakdolgozat, 2011

[8] Freescale Semiconductor, Inc. : Small Outline Integrated Circuit (SOIC) Package, 2014

[9] Solder Paste, Wikipedia (2016.05.09)

[10] Storcz Richárd: Gőzfázisú Újraömlesztéses Forrasztással Készült Pin-In-Paste Kötések Vizsgálata IPC Szabvány Szerint

[11] Continental Automotive Hungary Kft., MESWiki

[12] MES – Manufacturing Execution System, internetes cikk (2016.05.09.) http://www.scadasys.hu/hu/index.php/hu/cikkek2/88-ibv

[13] A kliens-szerver architektúra, Wikipedia (2016.05.10.)

[14] SMEMA, Wikipedia (2015.05.14)

[15] ESD SMT Magazine Rack – kép, www.conco-esd.com (2016.05.14)

[16] Skipping Damaged PCBs in the Panel (InkSpot) – kép,www.omix.pl (2016.05.11)

[17] Continental Automotive Hungary Kft., General SMT concept Budapest

[18] Pogány László, Demonstrációs alkalmazás fejlesztése operációs rendszer funkciók bemutatására, Szakdolgozat

[19] Single Minute Exchange of Die, Leanszótár (2015.05.12)

[20] Continental – Flash Download Process Specification

[21] DataIO – ProLine -RoadRunner Owner’s Manual

[22] DataIO – SDK Developer’s Guide

Függelék

F1. A feladatvezérlő modul adattípusai

F1.1. Vezérlési adattípusok

F1.1.1. DataIOResponse

A webszolgáltatás minden metódusa a DataIOResponse objektum egy példányával tér vissza.

Mező neve	Adattípus	Megjegyzés
ID	String	A webszolgáltatás összes műveletéhez generált egyedi azonosító.
Status	DataIOOperationStatus	A művelet státusza, ami lehet Pending, Success vagy Failure.
ErrorID	String	Hiba esetén a webszolgáltatás által generált egyedi azonosító.
Description	String	Hiba esetén annak szöveges leírása.

F1.1.2. DataIOSystem

A DataIOSystem tartalmazza a programozóval kapcsolatos konfigurációs információkat.

Mező neve	Adattípus	Megjegyzés
HostID	GUID	A szerver egyedi azonosítója. NULL értéke azt jelzi, hogy a kliens és a szerver ugyanazon a gépen van.
ManagementId	GUID	Egyedi azonosító, mely akkor generálódik, amikor az eszközt hozzáadjuk a rendszerhez.
HardwareID	String	A eszköz MAC címe.
NetworkAddress	String	Az eszköz IP címe.
Port	Int	A port, melyhez a FIS csatlakozni tud.
Name	String	A eszköz neve.
Enabled	Boolean	Azt jelzi, hogy az eszköz engedélyezve van-e.
Type	DataIOHardwareType	Egy enum típus, mely tartalmazza az eszköz típusát.
ConnectionStatus	DataIOConnectionStatus	Aktuális kapcsolat állapota.

F1.1.3. DataIOSystemInfo

A DataIOSystemInfo egy programozó eszközzel kapcsolatos lényegi információkat tartalmaz.

Mező neve	Adattípus	Megjegyzés
System	DataIOSystem	Programozó konfigurációs beállításai.
CurrentJob	String	Tartalmazza a jelenleg futó feladat nevét.
HardwareDetail	String	Értéke 'None', ha nincs semmi hiba. Ha hiba van, megegyezik a DataIOResponse ErrorId mezőjével.
HardwareStatus	DataIOHardwareStatus	A RoadRunner jelenlegi állapotát tartalmazza.

F1.2. Vezérlési adattípusok státuszai

F1.2.1. DataIOHardwareStatus

A DataIOHardwareStatus típusa enum (felsorolás), a programozó lehetséges állapotait tartalmazza.

Mező neve	Mező száma	Megjegyzés
Unknown	0	Az eszköz állapota ismeretlen.
Idle	1	Az eszköz arra várakozik, hogy feladatot kapjon.
RunnerJob	2	A eszközön jelenleg fut egy feladat.
TransferringFile	3	A eszköz egy feladatot másol át a saját tárhelyére.
PausedJob	4	Az eszközön lévő feladat futása szüneteltetve van.
PreparingSystem	5	Az eszköz jelenleg felkészül egy feladat elvégzésére.
InvalidConfiguration	6	Az eszköz konfigurációja hibás, így használhatatlan.

F1.2.2. DataIOConnectionStatus

A DataIOConnectionStatus típusa enum, tartalmazza a FIS és a programozó eszköz közötti lehetséges kapcsolati állapotokat.

Mező neve	Mező száma	Megjegyzés
Unknown	0	Státusz ismeretlen.
Disabled	1	A hardver nincs engedélyezve, így nem fog tudni kommunikálni a FIS szerverrel.
Connected	2	Csatlakoztatva. A hardver megtalálható a hálózaton.
NotConnected	3	Nincs csatlakoztatva. A hardver nem található a hálózaton, vagy le lett tiltva. Utóbbi esetben hamarosan Disabled lesz az állapota.
LockedByOtherHost	4	A hardvert egy másik szerver már használja.
IncopatibleFirmware	5	A hardveren lévő firmware nem kompatibilis a FIS szerverrel.(A firmware frissítésével a probléma megoldódhat.)
Pending	6	A kapcsolat létrehozása folyamatban van. Ez egy ideiglenes állapot. Az állapot ezután vagy PendingLock, vagy UnableToLock, vagy IncopatibleFirmware lesz.
PendingLock	7	Alapvető kommunikáció létrejött. Ez egy átmeneti állapot. Az állapot ezután lehet NonConnected, Connnected, LockedByOtherHost, vagy UnableToLock.
UnableToLock	8	A kommunikáció létrejött, de a zárolás sikertelen.
HardwareCannotContactHost	9	Sikertelen kapcsolódás a host számítógéphez.
InvalidSystem	10	Az eszköz érvénytelen állapotba került. A FIS ezután nem tudja használni.

F1.2.3. DataIOHardwareType

Az eszköz lehetséges típusait tartalmazó enum.

Mező neve	Adattípus	Megjegyzés
Unknown	0	Hardver típusa ismeretlen.
RoadRunner	1	A programozó a RoadRunner család tagja.
SimulatedRoadRunner	2	Az eszköz egy szimulált RoadRunner. Teszteléshez nagyon hasznos.
Ps	3	A programozó a PS család tagja.

F1.2.4. DataIOOperationStatus

A műveletek lehetséges kimeneteit tartalmazó enum.

Mező neve	Adattípus	Megjegyzés
Success	0	A művelet sikeres.
Pending	1	A művelet még nincs befejezve.
Failure	2	A művelet befejezve, és az eredménye hibás.

F2. A feladatvezérlő modul funkciói

A vezérlési modul egyes funkcióinak magyarázata szintén egyszerűbb táblázatos formába rendezve. A táblázat fejlécében helyeztem el a függvény fejlécét. A táblázat egyes soraiban pedig rendre a függvényparaméterek, a visszatérési értékek, valamint a függvényhez tartozó megjegyzések találhatók.

F2.1. DownloadJob

A DownloadJob segítségével lehet egy feladatot feltölteni a programozó belső tárhelyére.

DataIOResponse DownloadJob (DataIOSystem system, string jobPath);	
Paraméterek	DataIOSystem system – A programozót reprezentálja string jobPath – A feladatot tartalmazó mappa elérési útvonala. A mappának a szerver által elérhetőnek kell lennie.
Visszatérési típus	DataIOResponse
Megjegyzések	A letöltött munkamenetet a StartJob függvénnyel lehet elindítani. A folyamat aszinkron működésű.

F2.2. DeleteAllJobsOnSystem

A DeleteAllJobsOnSystem függvény segítségével lehet törölni az összes programozóra töltött munkamenetet.

DataIOResponse DeleteAllJobsOnSystem (DataIOSystem system);	
Paraméterek	DataIOSystem system – A programozót reprezentálja
Visszatérési típus	DataIOResponse
Megjegyzések	Ajánlott minden letöltés előtt meghívni ezt a függvényt, így elkerülhetjük, hogy probléma adódjon a programozó tárhelyének megteléséből. A folyamat aszinkron működésű.

F2.3. StartJob

A StartJob függvény segítségével lehet elindítani egy munkamenetet a programozón.

DataIOResponse StartJob(DataIOSystem system, string jobName, int passQuantity);

Paraméterek	DataIOSystem system – A programozót reprezentálja sting jobName – Az elindítandó folyamat neve int passQuantity – A kívánt pass-os egység mennyisége
Visszatérési típus	DataIOResponse
Megjegyzések	A folyamat aszinkron működésű.

F2.4. PauseJob

A PauseJob függvény segítségével lehet egy munkamenetet szüneteltetni.

DataIOResponse PauseJob(DataIOSystem system);	
Paraméterek	DataIOSystem system – A programozót reprezentálja
Visszatérési típus	DataIOResponse
Megjegyzések	A munkamenet addig szünetel, amíg újra nem indítjuk a ResumeJob függvénnyel, vagy az előlapon található start gomb megnyomásával. A folyamat aszinkron működésű.

F2.5. StopJob

A StopJob függvény segítségével lehet egy munkamenetet leállítani.

DataIOResponse StopJob(DataIOSystem system);	
Paraméterek	DataIOSystem system – A programozót reprezentálja
Visszatérési típus	DataIOResponse
Megjegyzések	A folyamat aszinkron működésű.

F2.6. ResumeJob

A ResumeJob függvény segítségével lehet egy szüneteltetett munkamenetet újraindítani.

DataIOResponse ResumeJob(DataIOSystem system);	
Paraméterek	DataIOSystem system – A programozót reprezentálja
Visszatérési típus	DataIOResponse
Megjegyzések	A folyamat aszinkron működésű.

F2.7. GetSystems

A GetSystems függvény segítségével kérdezhetjük le, hogy milyen eszközök vannak hozzáadva a FIS-hez.

DataIOResponse GetSystems(out DataIOSystem[] systems);	
Paraméterek	[out] DataIOSystem[] systems – Egy programozókat reprezentáló elemeket tartalmazó tömb
Visszatérési típus	DataIOResponse

F2.8. AdjustPassQuantity

Az AdjustPassQuantity függvény segítségével be tudjuk állítani, hogy egy feladat hány darab passos egység legyártásáig fusson.

DataIOResponse AdjustPassQuantity (DataIOSystem system, int count, out int adjustedPassQuantity);	
Paraméterek	DataIOSystem system – A programozót reprezentálja int count – A beállítási érték [out] int adjustedPassQuantitiy – A beállítás utáni pass-os mennyiség
Visszatérési típus	DataIOResponse
Megjegyzések	A beállítási érték lehet pozitív és negatív is. A függvény segítségével azt tudjuk megmondani, hogy a jelenlegi beállításnál mennyivel legyen több vagy kevesebb. A folyamat szinkron működésű.

F2.9. ClearBelt

A ClearBelt függvény a szállítószalag ürítési módjának módosítását szolgálja.

DataIOResponse ClearBelt(DataIOSystem system, BeltClearingMode mode);	
Paraméterek	DataIOSystem system – A programozót reprezentálja BeltClearingMode mode – A szállítószalag kívánt ürítési módja (automata vagy manuális)
Visszatérési típus	DataIOResponse
Megjegyzések	A szállítószalag ürítése általában 8-15 másodpercet vesz igénybe. A folyamat aszinkron működésű.

F2.10. GetLastResponse

Egy művelet azonosítójának ismeretében lekérdezhető a művelet eredménye.

DataIOResponse GetLastResponse(string operationId);	
Paraméterek	string operationId – DataIOResponse ID mezőjét kell megadni
Visszatérési típus	DataIOResponse
Megjegyzések	Egy adott művelet válaszüzenetét lehet vele megkapni. Aszinkron műveletek esetén nagyon hasznos, hogy később le tudjuk kérdezni az eredményt.

F2.11. GetStatus

A GetStatus függvény egy programozó jelenlegi állapotának lekérdezésére szolgál.

DataIOResponse GetStatus(DataIOSystem system, out DataIOHardwareStatus hardwareStatus);	
Paraméterek	DataIOSystem system – A programozót reprezentálja [out] DataIOHardwareStatus hardwareStatus – Kiemeneti paraméter, a hardware jelenlegi állapota
Visszatérési típus	DataIOResponse

F2.12. GetSystemInfo

A GetSystemInfo függvénnyel lekérdezhetőek egy programozó rendszerinformációi.

DataIOResponse GetSytsemInfo(DataIOSystem system, out DataIOSysteminfo systemInfo);	
Paraméterek	DataIOSystem system – A programozót reprezentálja [out] DataIOSystemInfo systemInfo – Jelenlegi rendszerinformációk
Visszatérési típus	DataIOResponse
Megjegyzések	A programozó állapotainak megfigyelése ezzel a függvénnyel a legkönnyebb