Somogyi Csongor

TUDÁSFA ALAPÚ DIAGNOSZTIKAI ALKALMAZÁS INTELLIGENS TERMÉKDOKUMENTÁCIÓHOZ

TDK dolgozat

1998.

Konzulens: **Mészáros Tamás**
Tóth András

Bevezetõ

A négy egyetem és két ipari partner együttmûködését magába foglaló intelligens termékdokumentáció (IPM) projekt diagnosztikai alkalmazásainak fejlesztésében veszek részt. Az IPM megfogalmazza azt a régóta áhított igényt, hogy megkönnyítve egy termék üzemeltetését, karbantartását, javítását segítséget nyújthassunk a gép üzemeltetõinek, karbantartóinak és bizonyos döntések meghozatalát nagy mértékben elõkészíthessük. Ezt az igényt a mai Internetes, multimédiás és mesterséges intelligenciára alapuló technológiák alkalmazásával igyekszünk kielégíteni. Célunk nemcsak az elméleti fejtegetés, munkánkat párhuzamosan a koncepció kidolgozásával egy prototípus megalkotásával is kiegészítjük, ahol szembesülhetünk az elviekben megfogalmazott feladatok implementálási kérdéseivel. A prototípus az ipari partnerek egy-egy termékének a termékdokumentációját foglalja magába.

Ehhez a munkához én a diagnosztikai alkalmazások fejlesztésével igyekszem hozzájárulni. Kezdõlépésként a tudásfa alapú koncepció implementálása vetõdik fel annak egyszerûsége és hasznossága által. Egyszerûsége az implementációs kérdéseknél és a problémakör átláthatóságánál jelenik meg, hasznossága pedig a konkrét diagnosztikai esetek megvizsgálása során válik nyilvánvalóvá.

Dolgozatom négy fõ részre tagolódik. Az elsõ fejezet röviden ismerteti az IPM projektet, mit takar az elnevezés, mik a projekt céljai, követelményei, kik a résztvevõi. A második részben az IPM prototípusáról szeretnék beszélni, ebben elhelyezni a diagnosztikai alkalmazásomat. A harmadik fejezet szól az alkalmazás tényleges megvalósításáról, bemutatja mûködését, az alkalmazott adatstruktúrát, részletezi megírásának körülményeit. Végül az utolsó fejezet ismerteti a diagnosztikai alkalmazások továbbfejlesztésének fõbb irányvonalait.

Tartalomjegyzék

Bevezetõ 1
Tartalomjegyzék 2
I. Az IPM projekt 3
II. Az IPM prototípusa 4
2.1 A kitûzött feladat 4
III. A diagnosztikai alkalmazás 5
3.1 Követelmények 5
3.2 Az alkalmazás megépítéséhez szükséges technológiák 5
3.3 Tudásfa koncepció 6
3.4 A fejlesztés 7
3.4.1 A fejlesztés lépései: 7
3.4.2 A fejlesztés menete 7
3.4.2.1 Egybefejezett, mûködõ, de a követelményeknek nem megfelelõ munkadarab 7
3.4.2.2 A végleges verzió 8
3.4.2.3 Az adatstruktúra 9
3.4.2.4 A felhasználói kezelõ felület 10
3.4.2.5 Az információs fájlok formátuma 11
3.4.2.6 Az információs fájlok tartalma 12
3.4.2.7 A java applet környezete (indítása, indítási paraméterei) 12
IV. Továbbfejlesztési irányok 13
Összefoglaló 13
Irodalomjegyzék és linkgyüjtemény 13

I. Az IPM projekt

Az intelligens termékdokumentáció projekt* [IPM] célja, hogy kidolgozzon egy módszertant egy olyan rendszer kialakítására, amely támogatást ad intelligens döntéshozatalok elõkészítésére egy termék teljes életciklusa alatt, kezdve az üzembe helyezéstõl, a karbantartáson és a diagnosztikán keresztül a szétszedésig és újrahasznosításig. Az angol nevén Intelligent Product Manual (IPM, ProManual, ProMan) projekt fejlesztése négy európai egyetem és két ipari partner közremûködésével történik: a Cardiffi Egyetem (Wales), a Clausthalli Mûszaki Egyetem (Németország), a Rousse-i Egyetem (Bulgária), a Budapesti Mûszaki Egyetem, a SPARKY-Agromachina (Bulgária) és az Excel-Csepel. A termékdokumentáció készítéséhez szükséges módszerek kidolgozása mellett feladata a termékdokumentáció készítésének konkrét demonstrálása a két ipari partner egy-egy gépének termékdokumentációjának prototípusán keresztül. [1] [2]

Az IPM projekt három fõ technológiát kíván ötvözni:

- *elektronikus* formátumban tárolás megnövekedett információ elérést tesz lehetõvé, ide értve a dokumentációban történõ keresés lehetõségét, az egyidejûleg többszörös hozzáférést, a hordozhatóságot (méretbeli csökkenés), stb.
- *multimédia* eszközök alkalmazása kibõvült audió-vizuális megjelenítést nyújt a felhasználó számára, ami elõsegíti a gép funkcionális mûködésének megértését, segítséget nyújt a karbantartási, összerakási, szétszedési, stb. mûveletek elsajátításában
- *tudásalapú* technikák interaktívvá teszik a diagnosztikai folyamatokat, és strukturált formában tárolják a diagnosztikai információinkat, elõsegítve ezzel a bennük való keresést

A BME a Gépgyártástechnológia Tanszék és a Méréstechnika és Információs Rendszerek Tanszék fejleszti az egyik IPM prototípust, az Excel-Csepel cég SL 320/600 CNC típusszámú esztergagép intelligens termékdokumentációját.

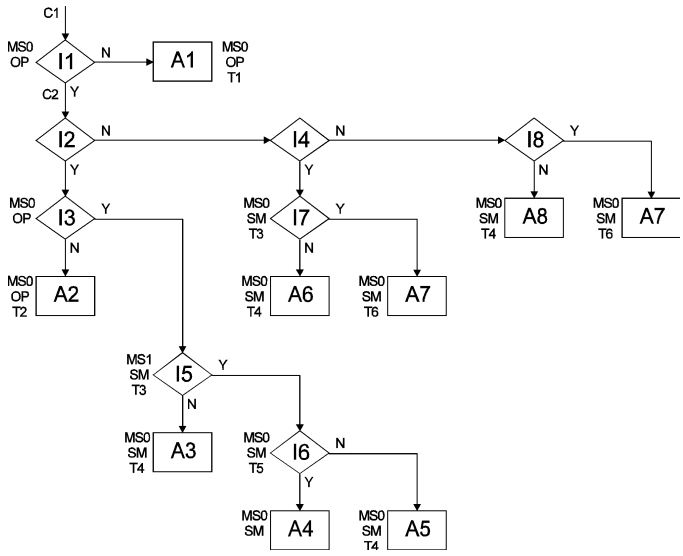
* INCO-Copernicus 96/0231

II. Az IPM prototípusa

Az prototípus [ExProMan] fejlesztésének egyik fõ irányvonala Webes környezetben történik. Elsõdleges feladat a meglévõ papíron írott dokumentáció átvitele elektronikus formába. A projekt által kitûzött fejlesztési menetrend alapján a módszerek kidolgozásával párhuzamosan kerül sor a gyakorlati megvalósításra is. A projekt jelenlegi fázisában egyre nagyobb hangsúly terelõdik a különbözõ mesterséges intelligencia technikákra is, amelyek különösen a diagnosztikai alkalmazások fejlesztéseinél fontosak. A jelenlegi fejlesztõ gárda egyrészt az IPM HTML dokumentumainak formai és strukturális kialakítását végzik, másrészt a diagnosztikai alkalmazások terén történik fejlesztés, amiben én veszek részt.
A Webalapú fejlesztésen kívül létezik egy ún. Multimedia Toolbook nevû fejlesztõi környezet, ez lényegében hasonlít a HTML-hez, de sokkal gazdagabb vizuális elemkészlettel rendelkezik.

2.1 A kitûzött feladat

Feladatom a fejlesztés alatt álló rendszer megismerése után egy diagnosztikai alkalmazás készítése Java programozási nyelven, amely az Exceltõl kapott diagnosztikai ismereteket reprezentálva interaktív technikai segítséget nyújt a végfelhasználóknak. A diagnosztikai alkalmazás a hibakeresés folyamatát segíti elõ, annak a lépéseit tárja a felhasználó elé. Az Exceltõl kapott adatok hibafára vannak redukálva (ld. 1. ábra).



1.ábra – Excel diagnosztikai hibafa

Minden egyes csomóponthoz egy-egy leírás tartozik, pl.:

I1: The adjustment of the chuck jaws is according to the size of the workpiece? Visual investigation.

A2: Fixing the loose contact.

És az egyes jelölésekhez a következõ megjegyzések tartoznak:

T Tools to be used:

T1 hex socket head screw driver
T2 slim slot screwdriver

MS Machine Tool's status:

MS0 machine switched off
MS1 machine switched on

OP/SM Person in action

OP Operator
SM Service man/electrician

III. A diagnosztikai alkalmazás

3.1 Követelmények

A diagnosztikai alkalmazással szemben támasztott követelmények a következõk:

- felhasználóbarát grafikus kezelõi felület létrehozása
- az alkalmazás beágyazása az intelligens termékdokumentációba
- flexibilis, dinamikusan módosítható, bõvíthetõ tudásbázis

Az elsõ szempontnál fontos annak vizsgálata, hogy a gépet üzemeltetõ mérnök, technikus milyen szemmel látja a diagnosztikai információt, amelynek a struktúrája kézzelfoghatóan látszódnia kell az alkalmazásban. A második szempont azért fontos, hogy egyrészt az alkalmazás szinte eggyé váljon a termékdokumentációval, azaz a kezelõi felületen keresztül ne legyen észrevehetõ, hogy külön alkalmazás, másrészt pedig megteremtõdjön a kapcsolat a termékdokumentáció adatbázisa és az alkalmazás között, tehát az alkalmazásnak paraméteren keresztül átadható legyen, hogy milyen konkrét diagnosztikai esetet tanulmányozunk, ill. lehetõség legyen az alkalmazásból külsõ információkra hivatkozni. A harmadik követelmény arról szól, hogy az alkalmazás képes legyen egy adatbázisból információt nyerni, és ennek az adatbázisnak a struktúrája legyen minél egyszerûbben kezelhetõ.

3.2 Az alkalmazás megépítéséhez szükséges technológiák

- Java™ programozási nyelv ahhoz, hogy az alkalmazást (appletet) beágyazhassuk HTML dokumentumokba

A Webhez kötõdõ technológiák közül nélkülözhetetlen a Java [Java] [3] technológia. A nyelv filozófiája, hogy "írd meg egyszer, futtasd akárhol". Ezzel a filozófiával milliárdos költségeket tudunk megtakarítani, másrészt rengeteg programozói veszõdségtõl mentesülhetünk belegondolva abba, hogy az egyszer jól megírt programozói könyvtár segítségével nem kell minden alkalommal újraírni egy programot. A Sun™ [Sun] cég fejlesztése ezért terjedhetett el az Interneten és jelentõsége ma is nõ. Természetesen meg kell említeni a Java nagy vetélytársait, a különbözõ script nyelveket, amelyek szintén platformfüggetlenségre törekszenek, de mégis megvan az a hátrányuk, hogy kevésbé támogatják úgy a mai korszerû programozói elveket (típusosság, objektum-orientáltság), mint a Java. Emellett a Java talán az elsõ olyan széles körben ismertté vált programozási nyelv, ami nem akadémiai, tehát nem egyetemi kutatók és az általuk megalkotott szabványok segítségével jött létre, hanem egy cég pénzügyi befektetésének és fejlesztésének eredménye. Fontos leszögeznünk, hogy a Java nemcsak programozói nyelv, hanem a hozzátartozó futtatási környezet is, amit Java virtuális gépnek nevezünk (JVM).
A Java röviden ismertetve úgy mûködik, hogy a java forráskódot egy ún. byte-kódra fordítjuk, ami tulajdonképpen egy hordozható, futtatható kódot jelent, amit egy JVM futtatni

tud. Tehát nekem elég csak a JVM-et megírom egyszer és máris Java kódok futtatására alkalmas környezetet kapok.
Az alkalmazás fejlesztése szempontjából fontos említést tennünk a különbözõ verziókról ill. a programozói könyvtárakról (package-ekrõl). A Java-nak két különbözõ verziója jelent meg, az 1.0 és az 1.1. Az 1.0 a legelsõ igen széles körben elterjedt verzió volt, szinte minden egyes platformra készült virtuális gép, de hátránya volt, hogy még kevés programozói könyvtárral rendelkezett és azok között is akadt olyan, amit késõbb teljesen újrastrukturáltak, emiatt az elsõ verziót hamar követte a második. A másodiknál már sajnos problémák akadtak, mert bár sokkal többmindenre fel van/volt készítve, és sokkal strukturáltabb, ezáltal rugalmasabb programozói környezettel rendelkezett, a virtuális gépfejlesztõk nehezen tudták követni a változásokat és emiatt viszonylag nehezebben terjedt/terjed el. Természetesen az 1.1 verzió visszafele kompatíbilis az 1.0-ssal. Azt nagyos fontos leszögezni, hogy az 1.0-s verzió fejlesztését leállították és az 1.1 verzió fejlesztését elõtérbe helyezték, emiatt valószínûsíteni lehet az 1.1 jövõbeni szélesebb körû elterjedését.
A Java kezdeti fejlesztésénél az volt a koncepció, hogy egy bizonyos grafikus elemkészlet (gomb, ablak, görgetõ, stb.) ne a programnyelvvel leírt elemi rajzolási mûveletekkel rajzolódjék ki a képernyõre, hanem a virtuális gépet futtató operációs rendszer natív elemkészlete hívódjon meg. Ez azt jelentette, hogy különbözõ platformokon különbözõ módon rajzolódtak ki az alakzatok ugyanazon függvényhívásra. Manapság azonban a virtuális gép gyorsabbá válása következtében lehetõség adódik arra, hogy a Java maga rajzolja meg ezeket az elemeket, ezáltal robosztusabbá téve a továbbra is hordozható kódot. Ezt a technológiát lightweightnek nevezzük. Ezen a technológián alapul a Sun Java Foundation Classes (JFC) [JFC] elemkészlete, ami arra törekszik, hogy a grafikus elemkészlet független legyen az operációs rendszertõl, és ezáltal nagyobb elemkészlet álljon a programfejlesztõk rendelkezésére. A JFC csak 1.1-es Java-val mûködik és ott is csak meghatározott alverziószám fölött. A mi szemszögünkbõl nézve a JFC azért fontos, egyrészt mert szabványos, másrészt mert gazdagabb és strukturáltabb grafikus elemkészlettel rendelkezik, harmadrészt pedig az alakzatok minden platformon ugyanúgy fognak kirajzolódni.
A Java programjainkat az ún. applet technológia segítségével ágyazhatjuk be Webes dokumentumainkba. A dokumentum letöltésekor a Webböngészõ meghívja a java virtuális gépet futtató rendszert és átadja neki a java applet kódját. A JVM a Weblapon elhelyezkedõ kis ablakba fogja rajzolni kimenetét. Így az a látszat adódik, mintha az applikáció fizikailag a dokumentum szervesrésze lenne.

3.3 Tudásfa koncepció

- az Exceltõl jelenleg rendelkezésre álló adatokra alkalmazható
- egyszerû és flexibilis struktúra
- a megoldáskeresése a fa gyökere éslevelei közötti útkeresésen alapul

Az applikáció elkészítésére két hónap volt megadva, ezért egy olyan módszert kellett alkalmazni, ami viszonylag egyszerû, de mégis flexibilis és a rendelkezésre álló adatokra alkalmazható. Ezért döntöttem a tudásfa-struktúra mellett. A felépítésébõl is világosan tükrözõdik, hogy egy olyan fáról van szó, amelynek vannak elágazó csomópontjai, ágai és végpontjai, levelei. Az elágazó csomópont egy kérdésnek vagy vizsgálatnak feleltethetõ meg, amire a felhasználónak válaszolnia kell, és ezután a választól függõen kell a következõ csomópontokra lépdelni, egészen addig, amíg egy levélig el nem jutunk. A levelek tartalmazzák az adott diagnosztikai problémára a választ, vagy megoldást. Technikailag tehát egy utat kell keresnünk a fa gyökerétõl addig a levélig, amely levél a megoldást tartalmazza az adott problémára.

3.4 A fejlesztés

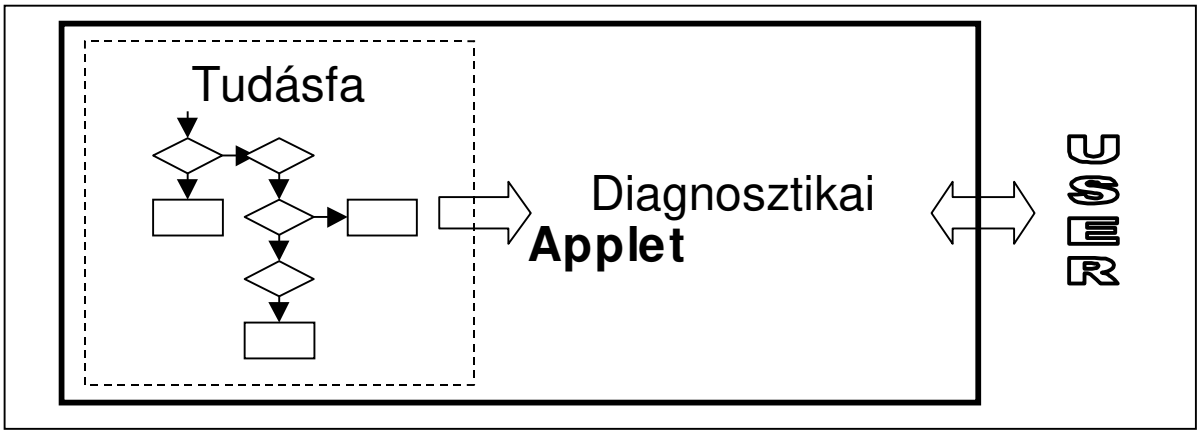
A tudásfa alapú diagnosztikai applet:

- teljesíti a követelmények közül az alkalmazás beágyazhatóságát az IPM-be valamint a grafikus felhasználói felület létrehozását
- a fa struktúrájú diagnosztikai információtárolás implementált

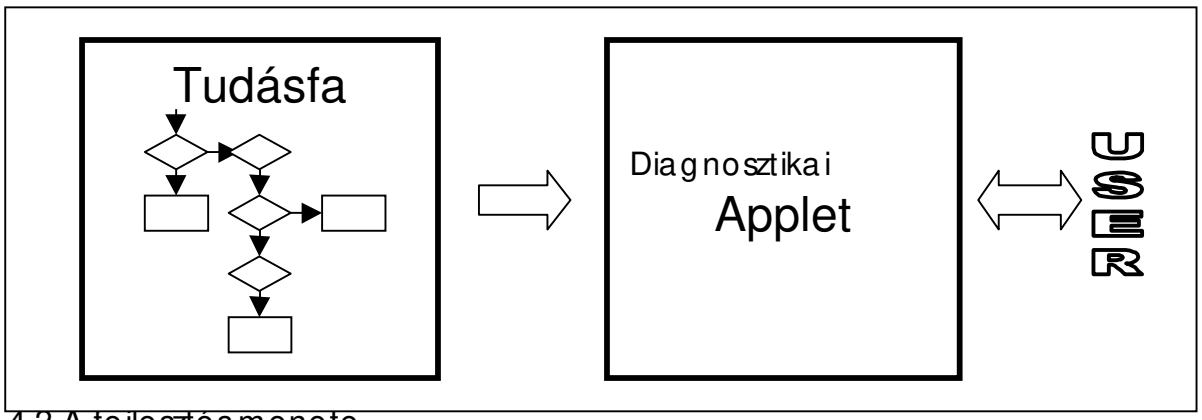
3.4.1 A fejlesztés lépései:

A fejlesztési tervben az a koncepció szerepelt, hogy elõször az applikáció belsõ vázát építsük fel, ehhez a fastruktúrát tároló objektumok létrehozását, valamint a kezelõi felület alapkomponenseinek megépítését szükséges megtenni. Tehát magának az appletnek elõször nincs szüksége külsõ adatok betöltésére, de fenn kell hagyni annak a lehetõségét, hogy késõbb az adatokat kívülrõl feltölthessük. A külsõ adatfeltöltés elsõ megközelítésben valamilyen szöveges fájl feldolgozásával történhetett volna meg, de mint késõbb látható ennél valamivel összetettebb, mégis sokkal átláthatóbb adatstruktúrát alkalmaztam.

1. lépés:



2. lépés:



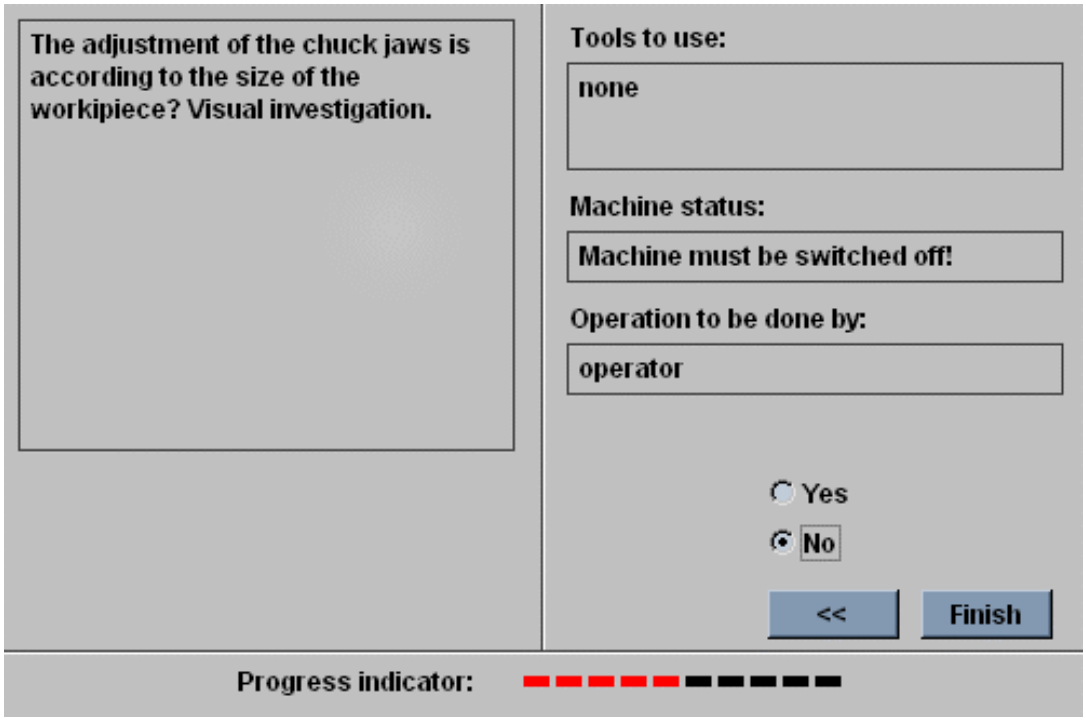
3.4.2 A fejlesztés menete

3.4.2.1 Egybefejezett, mûködõ, de a követelményeknek nem megfelelõ munkadarab

Elsõ tervem az volt, hogy az appletet a Java 1.0 verziójával építem meg. Ennek az elõnye, mint említettem, hogy széles körben elterjedt (még a 16 bites PC-s platformon is) mûködõképes programot készítettem. Ez a nagy elõny viszont többletmunkával járt, hiszen a sok programozói elem kezdetlegessége miatt a meglévõket ki kellett egészítenem, ill. újakat

kellett létrehoznom. Így pl. szükségem volt a grafikus elemek kibõvítésére (keretes szövegablak, pozíció indikátor). Saját elgondolás alapján kellett osztályokat létrehoznom, amelyek segítségével a fastruktúrát tudom belsõ programozói objektumokkal tárolni. Végül az applet kicsisége ellenére is elérte az átláthatatlanság küszöbét. Egyrészt bebizonyosodott, hogy programozói tevékenységem túl elhamarkodott volt, jobban át kellett volna gondolnom a program felépítését. Ezen kívül a további korlátok adódtak:

- az applet csak egyszerû szöveges információ megjelenítését tette lehetõvé (szegényes elemkészlet)
- a bemenõ adatok beolvasása egy szövegfájl alapján történtek, amiben túlságosan is megkötöttem az adatok jellemzõit (tipizáltam)
- ez magával vonta volna azt is, hogy az információ formai bõvítéséhez a szövegfájl struktúráját is meg kellett volna változtatnom.



2. ábra – Az elsõ változat felhasználói interfésze

Ezért ezen a szálon futó fejlesztés befejezõdött és rátértem egy sokkal egyszerûbb, átlátható struktúra alkalmazására.

3.4.2.2 A végleges verzió

Az ötlet egy a fastruktúrára épülõ könyvtárstruktúra felhasználása volt. Ezzel a módszerrel az adatok sokkal rendezettebben tárolhatóak, rugalmasan módosíthatóak és sokkal általánosabb struktúrát eredményez, amelyet lényegesen egyszerûbb a késõbb esetlegesen felmerülõ igényekhez igazítani. Az applet megépítésére a Java 1.1-es verzióját használtam, aminek hátránya, hogy nem minden böngészõben fut rendesen, viszont ezt a hátrányt a Java plug-in technológia alkalmazásával lehet kiküszöbölni (ld. késõbb). A felhasználói interfész tervezésekor a programozói szabadságot lényegesen megnövelte a JFC, és a tudásfa felépítéséhez szükséges belsõ objektumok is adottak voltak a JFC-ben rejlõ

famegjelenítõhöz kapcsolódó csomópont objektumok bõvítésével. Az igazi nagy segítséget viszont a JFC HTML komponense adta, amivel lehetõvé vált HTML dokumentumok megjelenítése az applet ablakban. Ezzel teljes beágyazottságot érhettem el.

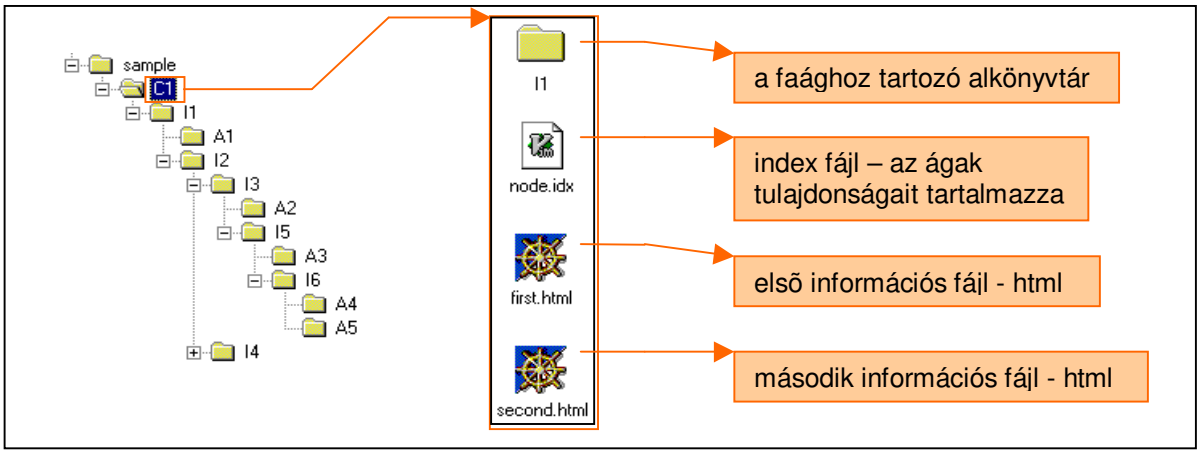
A szükséges eszközök:

- Java fordító (Sun JDK 1.1.6)
- Java Foundation Classes (JFC, Swing 1.1.3)
- esetlegesen vizuális fejlesztõi környezet

A fent említett elgondolások alapján megépített applet által használt adatstruktúrát és részletes mûködését a következõ fejezetek ismertetik:

3.4.2.3 Az adatstruktúra

Az adatstruktúra felépítése az 3. ábrán látható. Mivel az operációs rendszerek könyvtárstruktúrája faszerkezetû, ezért célszerû kihasználni ezt az elõnyös tulajdonságot. Így minden egyes fa csomóponthoz egy könyvtárat lehet rendelni, és egy csomópontból elágazó ágakon levõ további csomópontok a szülõ csomóponthoz tartozó könyvtár alkönyvtáraiban tárolódnak, ill. azoknak felelnek meg.



Ahhoz, hogy egy könyvtárban, mint csomópontban elhelyezkedõ alkönyvtárakat, mint ágakat jellemezhessük, minden egyes alkönyvtárba egy indexfájlt (node.idx) helyezünk el. Az indexfájl leírja többek között az ágakat, ezek az ágak a csomópontra, mint kérdésre (vizsgálatra) adandó lehetséges válaszokat reprezentálják, leírja továbbá az ágakhoz csatlakozó csomópontok neveit, amely nevek megnevezik az alkönyvtárakat, mint következõ csomópontokat, ezen kívül megadja minden egyes kapcsolódó csomópont típusát, hogy további kérdés/vizsgálat-e vagy egy falevél, utóbbi esetben ugyanis a hozzátartozó alkönyvtárban már nincs újabb alkönyvtár és így index fájl is szükségtelen. Ezek alapján az index fájl felépítése a következõ:

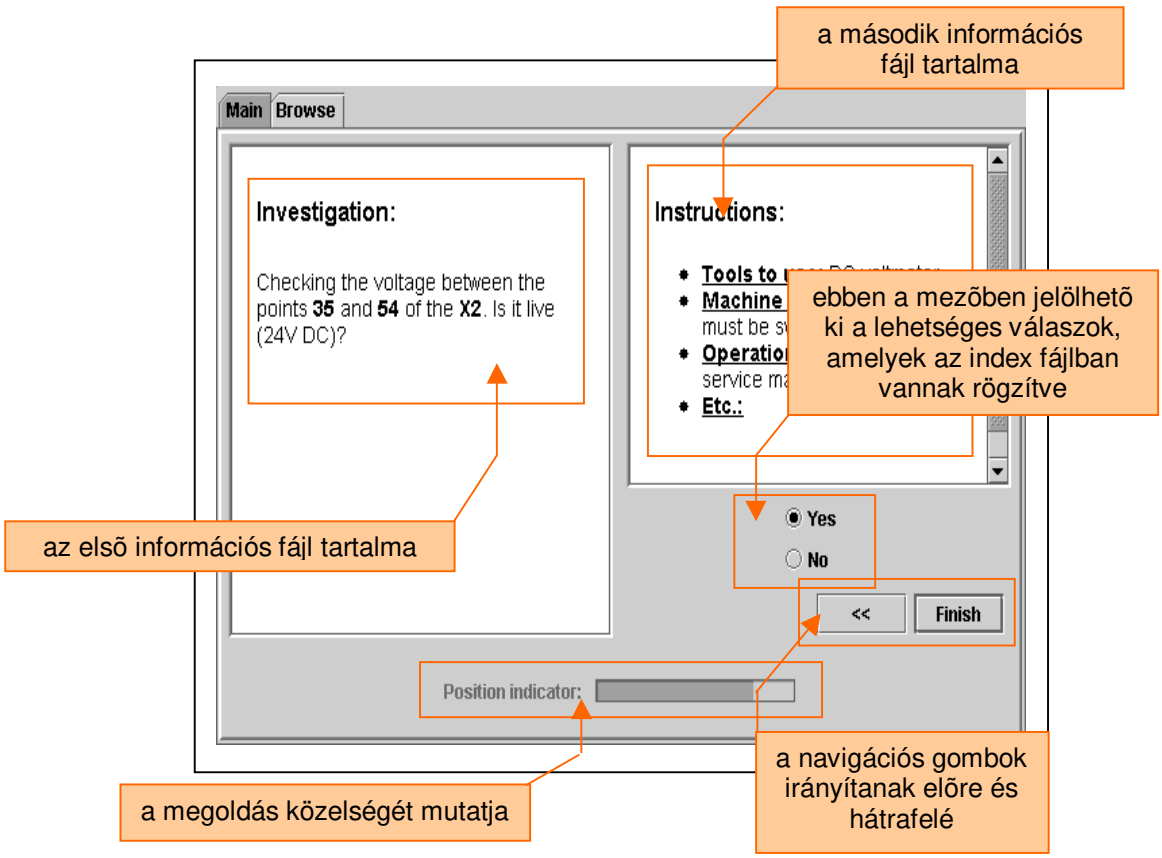
```
node|leaf:<alkönyvtár(csomópont) neve>:<a kapcsolódó
válasz(ág)>:<megjegyzés>
```

Tehát az egyes sorok jelölik az egyes ágakat. A sor mezõi kettõsponttal vannak elválasztva, az elsõ mezõ az ághoz kapcsolódó csomópont típusát adja meg, node mint csomópont egy további kérdést/vizsgálatot jelent, míg a leaf levelet jelent, és egy választ/megoldást jelez. A második mezõ a kapcsolódó csomópont (alkönyvtár) neve, a harmadik mezõ a kérdésre adandó, a második mezõben megadott csomóponthoz kapcsolódó választ adja. A negyedik mezõben megjegyzést helyezhetünk el, melynek szerepérõl késõbb beszélek. Ezek alapján egy valóságos indexfájl tartalma így néz ki:

```
node:I2:Yes:checking illumination of 10PRS16 and 10PRS17 LEDs
leaf:A1:No:adjust the jaws
```

Az egyes könyvtárakban a csomóponthoz, mint kérdéshez/válaszhoz tartozó összes információt tárolhatjuk. Jelen esetben két HTML formátumú információs fájl található (first.html, second.html), de attól függõen, hogy milyen megjelenítési módszereket alkalmazunk, másfajta adatokat is tárolhatnánk benne, én itt elsõsorban a csomóponthoz tartozó információ megjelenítésére fektettem a hangsúlyt, ezért választottam a HTML formátumot.

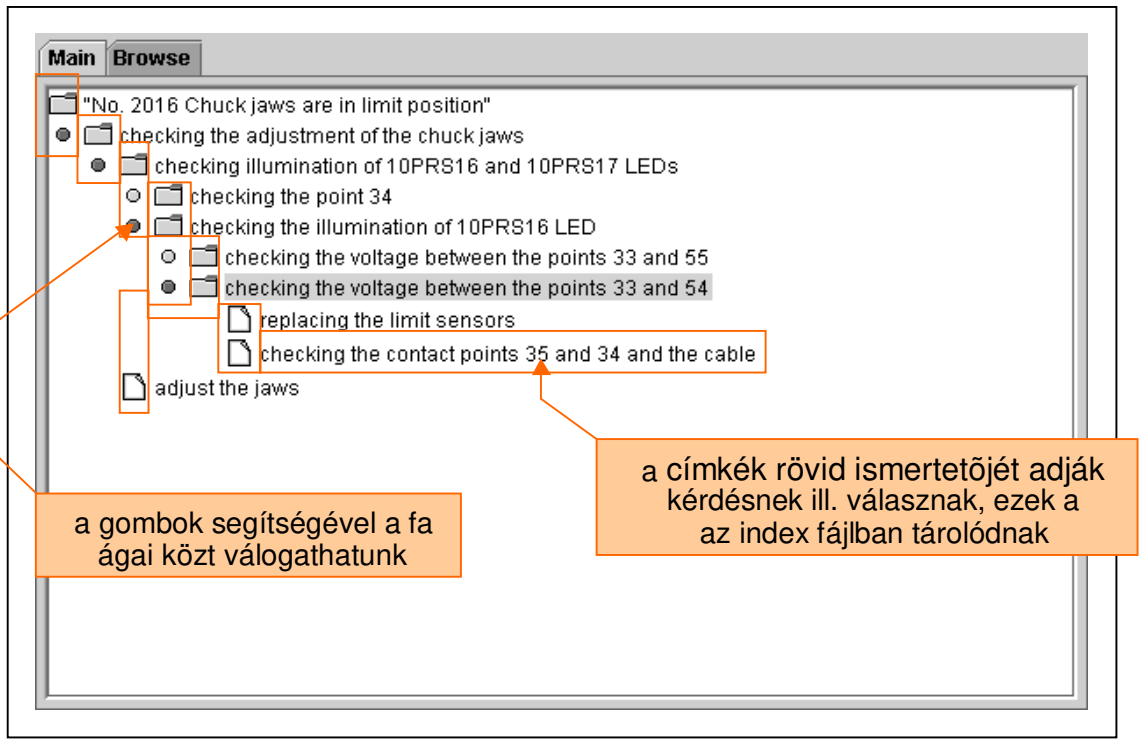
3.4.2.4 A felhasználói kezelõ felület



A felület a JFC grafikus elemkészlet (Swing toolkit, widget-set) segítségével lett kialakítva. Az ablakban két lap lett egymáson elhelyezve, amelyek közötti váltás a bal felsõ sarokban található két fülön történõ egérkattintás segítségével oldható meg. A két lap közül az elsõ (4. ábra) Main-nek lett elnevezve, ezen a lapon található a két információs fájl tartalma, két fõ részre osztva az ablakot. Itt található továbbá a navigációt segítõ eszközök, így a kérdések közötti válaszok kiválasztásának lehetõsége, valamint itt vannak az elõre-

hátranavigáló gombok is, amelyek segítségével a fán a gyökér és valamely levél közötti útvonalon tudunk mozogni. A pozíció indikátor azt mutatja meg, hogy ezen az útvonalnak hányad részét tettük meg a gyökértõl számítva.

Amennyiben a második fülre (Browse) kattintunk, akkor a fának strukturális képét (5. ábra) kapjuk egy ún. faböngészõben. Itt jelennek meg az egyes csomópontokhoz tartozó megjegyzések, amelyek a node.idx fájlban vannak eltárolva. Itt lehetõség van kiválasztani egy adott címkét, amelyre rákattintva az ablak visszavált a Main lapra, ahol a kiválasztott kérdés vagy válasz részletes leírása található. A címke kiválasztásával a gyökértõl az aktuális csomópontig tartó útvonal beállítódik, vagyis ezen az útvonalon oda-vissza tudok mozogni a navigációs gombok segítségével, és meg tudom vizsgálni, hogy az adott pont eléréséhez milyen vizsgálatokat kell végeznem, kérdéseket megválaszolnom, és azokra milyen válasznak kell érkeznie.



3.4.2.5 Az információs fájlok formátuma

A fájlok HTML formátumúak, így bennük bármilyen, a szabványban megadott vizuális kifejezõ eszköz használható, így pl. képek, listák, táblázatok, linkek, stb. Ha egy linkre kattintunk a dokumentumon belül, akkor egy új böngészõ ablak nyílik meg, amiben azon dokumentum tartalma jelenítõdik meg, amelyre a link mutat. Ez a link számos új információra mutathat az adott kérdéskörrel kapcsolatban, ami felhasználható arra, hogy a tárgykörrel kapcsolatban más részletes adatokat is hozzáfûzhessünk az alapinformációhoz vagy esetlegesen egy rajz mérete túl nagy az ablakhoz képest, ekkor egy új böngészõablakban nagyobb léptékben szemlélhetjük meg.

3.4.2.6 Az információs fájlok tartalma

Az applet a viszonylag tartalomfüggetlen felépítése miatt alkalmas általános fastruktúrájú kérdezz-felelek típusú rendszerben való alkalmazására is, de a diagnosztikai ismeretek tudatában a két információs fájl tartalmát is részletesebben kell meghatároznunk. Az elsõ információs fájl az ablak bal felében jelenik meg. Itt fogalmazhatjuk meg a kérdésünket/vizsgálódásunkat ill. válaszainkat/ megoldásainkat. A jobb oldalon megjelenített második információs fájlban segítséget tudunk nyújtani a baloldalon feltett kérdés megválaszolásához, ill. utasításokat tehetünk a felderített hiba megoldásának konkrét lépéseire. Ilyen segítségek vagy utasítások lehetnek, hogy milyen eszközökre van szükségünk, milyen állapotban kell lennie a gépnek a mûvelet elvégzéséhez, ki az, aki elvégezheti a mûveletet, stb.

3.4.2.7 A java applet környezete (indítása, indítási paraméterei)

A java applet egy HTML [HTML] dokumentumban elhelyezett ablakot jelent, amely ablakban látszik a programka felhasználói felülete. Ezt a beágyazást az APPLET nevû taggel lehet elvégezni:

```
<APPLET  CODE = DiagApp.class WIDTH = 600 HEIGHT = 400 >
<PARAM NAME = "root" VALUE  = "./sample/">
<PARAM NAME = "background" VALUE  = "FFCCFF">
<PARAM NAME = "foreground" VALUE  = "CCFFFF">
</APPLET>
```

Az elsõ sor adja meg, hogy melyik java kódot fogja futtatni és azt, hogy az ablak mérete mekkora. A paraméterek közül a legelsõ a fontosabb, mert abban adhatjuk azt az URL-t, ami a fa gyökerére mutat. Az applet innen kezdve keresi az indexfájlokat és segítségükkel az alkönyvtárakat. A másik két paraméterrel az applet színeit lehet konfigurálni.
Fontos megjegyezni, hogy a java különbözõ verziói miatt a Webböngészõk különbözõ java támogatással rendelkeznek, ami miatt elõfordulhat, hogy az applet nem fog mûködni. Ezért a Sun java fejlesztõi részlege piacra dobta az ún. java plug-int, ami a Webböngészõbe telepítve mindig az aktuális java verzió futtatásának lehetõségét nyújtja, aminek segítségével az applet biztosan mûködni fog, minden olyan böngészõben, amelyben a plug-in telepítve lett. Ehhez azonban a HTML dokumentumot módosítani szükséges a következõ formában:

```
<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
WIDTH = 600 HEIGHT = 400
codebase="http://java.sun.com/products/plugin/1.1/jinstall-11-
win32.cab#Version=1,1,0,0">
<PARAM NAME = CODE VALUE = DiagApp.class >

<PARAM NAME="type" VALUE="application/x-java-applet;version=1.1">
<PARAM NAME = "root" VALUE  = "./sample/">
<COMMENT>
<EMBED type="application/x-java-applet;version=1.1" java_CODE =
DiagApp.class WIDTH = 600 HEIGHT = 400  root =  "./sample/"
pluginspage="http://java.sun.com/products/plugin/1.1/plugin-
install.html"><NOEMBED></COMMENT>
</NOEMBED></EMBED>
```

A mûvelet elvégzésének segítésére a Sun kiadott egy ún. HTML konvertert, ami egy meglévõ dokumentumból készít java plug-int hívó dokumentumot.

IV. Továbbfejlesztési irányok

A jövõben a diagnosztikai alkalmazásokkal szemben támasztott követelmények kibõvülnek a következõvel:

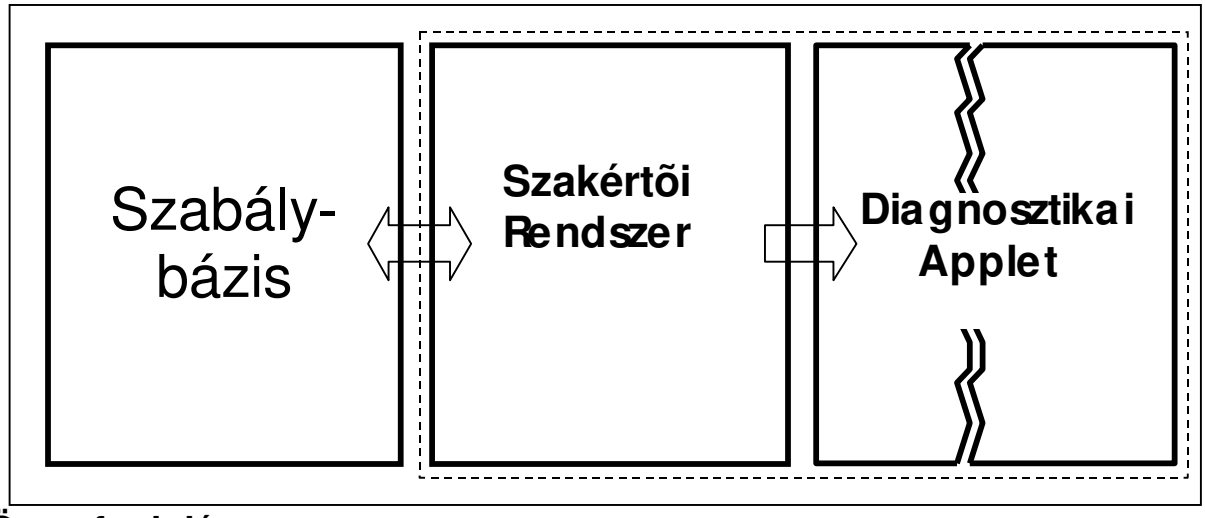
- mesterséges intelligencia technikák és eszközök felhasználása (szakértõi rendszerek, gépi tanulás, stb.)

Ehhez az itt felsorolt technológiák alkalmazása lesz szükséges:

- következtetõ gép
- tanuló algoritmusok
- szabály adatbázis

Tehát az IPM diagnosztikai alkalmazásainak jövõképe a szakértõi rendszerek irányába vezet. A diagnosztikai tudást szabályok segítségével sokkal általánosabban lehet ábrázolni, így nem fastruktúrájú diagnosztikai ismereteket is. A szabályokat táblázatokban lehet tárolni, és ez flexibilissebbé teszi ezt a módszert.

Ezek szerint a rendszer struktúrája így módosulna:



Összefoglaló

Dolgozatom röviden ismertette az intelligens termékdokumentáció projektet, majd rátért az Excel ProManual prototípuson belüli diagnosztikai alkalmazás konkrét implementációs kérdéseire. Összegezve két fõ szempont vezérelte a program fejlesztését:

- az Excel által rendelkezésre bocsátott adatok struktúrájának alkalmazása
- felhasználóbarát kezelõi felület

Úgy érzem, sikerült kellõen figyelembe venni ezeket a követelményeket, és ezen túlmenõen sikerült úgy leáltalánosítani a problémát, hogy az más diagnosztikai adatokkal is használható és emellett önmagában is mûködõképes alkalmazás készült el.

Irodalomjegyzék és linkgyüjtemény

[1] Deliverable 1.1: **Analysis of the Problem Domain**, Intelligent Product Manual, INCO-COP 96/0231, January 1998

[2] Deliverable 1.2: **Outline Solution that Illustrates the Capabilities of Intelligent Product Manuals**, Intelligent Product Manual, INCO-COP 96/0231, January 1998

[3] Ken Arnold, James Gosling, **The Java™ Programming Language Second Edition**, Addison Wesley Longman Computer & Engineering Publishing Group, ISBN 0-201-31006-6, 1998

[IPM] http://intell-lab.engi.cf.ac.uk/manufacturing/IPM/index.htm

[ExProMan] http://proman.manuf.bme.hu

[Java] http://java.sun.com

[Sun] http://www.sun.com

[JFC] http://java.sun.com/products/jfc/

[HTML] http://www.w3.org/MarkUp/