Debreceni Egyetem Informatika Intézet

Információs rendszer tervezése, fejlesztése

A National Insruments NITIL rendszere

DIPLOMAMUNKA

Témavezető:
Dr. L. Nagy Éva
számítástechnikai munkatárs

Készítette:
Orbán Tamás
programtervező matematikus
hallgató

Debrecen, 2007.

Tartalomjegyzék

1. Bevezetés

Az információtechnológia fejlődésével a vállalati folyamatok és az IT infrastruktúra eszközei egyre szorosabban integrálódnak. A növekvő vállalatok hatékony működéséhez elengedhetetlen az IT eszközök pontos és hatékony nyilvántartása, mely – túlnőve a kezdetben papíron és Excel táblázatokban vezetett listákon – egy könnyen kezelhető, gyorsan kereshető, jelentéseket generáló szoftver bevezetésében teljesedik ki.

A vállalati részről felmerülő igényekkel a National Instruments számára folytatott fejlesztői munkám során találkoztam első ízben. Diplomamunkámban bemutatom a cég számára készített konfigurációmenedzsment szoftvert, melynek célja az eszköznyilvántartáson túl az IT szolgáltatások színvonalának növelése. A rendszer vélhetően elősegíti az infrastruktúra eszközeiből fakadó hibák gyorsabb, hatékonyabb elhárítását.

A kitűzött cél szerves részét képezi az IT szolgáltatás menedzsment területének. Szolgáltatásmenedzsmentre vonatkozó ajánlások már a 80-as években megjelennek az IBM négy kötetes Yellow Books könyvében, továbbá a brit CCTA (Central Computer and Telecommunication Agency) által kibocsátott ITIL (IT Infrastructure Library) dokumentációsorozatában. Az ITIL már bevált gyakorlati tapasztalatok (best practices) egységes formában rögzített dokumentációja. A jelenleg 9 témakörbe szervezett dokumentumgyűjtemény két központi része az informatikai szolgáltatásmenedzsmentet öleli fel, a szolgáltatásbiztosítást és szolgáltatástámogatást tárgyalva. Az ajánlások egyre szélesebb körű elterjedésével az ITIL „de facto” szabvánnyá nőtte ki magát, s mára az informatikai szolgáltatásmenedzsment (IT Service Management - ITSM) területén az ISO/IEC 20000 szabvány hordozza az ITIL-ben foglaltakat.

A konfigurációkezelést az ITIL a szolgáltatástámogatás részeként tárgyalja. Az ajánlás bevezeti a konfigurációs elem fogalmát, mely lehet tetszőleges IT eszköz vagy eszközök együttese lehet. Az egyes elemek egymástól relációk mentén függnek. A konfigurációs elemeket attribútumok segítségével jellemezi (gyártási szám, leltári szám, verzió, gyártó stb.). Az elemek, attribútumok, relációk együttesét egy konfigurációmenedzsment adatbázisban (CMDB) tárolja. Az ITIL az egyszerű nyilvántartás mellett javaslatot tesz a tervezhetőség

beépítésére, azaz még nem létező, tervezett elemek, állapotok nyilvántartására. Ezzel szimulálható a vállalat rövidtávú stratégiája az infrastruktúra területén. Definiálja a vezérlést és monitorozást, azaz megfelelő ellenőrzést követően lehet konfigurációs elemet rögzíteni a rendszerben, s annak a teljes életútját nyomon lehet követni a felvételtől a törlésig.

A diplomamunkában bemutatásra kerülő szoftver az ITIL fogalomrendszere mentén felépülő, konfigurációkezelést megvalósító termék. A National Instruments Europe Kft számára készült szoftver összekapcsolja a vállalat pénzügy és IT részlegén párhuzamosan létező, gyengén konzisztens tárgyieszköz nyilvántartást, s üzleti logikájában felhasználja a vállalat működése során kialakult konvenciókat. Így a termék egy National Instruments (NI) specifikus ITIL implementáció, innen származik a NITIL elnevezés.

A cég által használt vállalatirányítási keretrendszer az Oracle9i adatbázis alatt futó Oracle Applications 11i. Így a nyilvántartórendszer fejlesztése is Oracle technológiát követ, az adatbázis Oracle9i R2, az alkalmazásszerver Oracle Application Server 10g. Azaz a rendszer a felépítése a napjainkban divatos n-rétegű architektúrát követi, webes elérhetőséggel. A fejlesztés alapja RAD technológia. Az adatbázis kezdeti fázisban Oracle Designer 2000 segítségével készült el, mely automatizmust biztosít a Table API valamint egyszerűbb felhasználói felületek generálására. A felhasználói felületek további része és a riportok manuálisan készültek Oracle Forms 6i-ben illetve Oracle Reports 6i-ben. Az üzleti logikát és felhasználói felülettel való kommunikációt megvalósító PL/SQL csomagok PL/SQL Developer 6.0-ban íródtak. Diplomamunkámban elsősorban az Oracle Forms eszközrendszerével foglalkozom, illetve Forms és Reports integrációjával. A fejlesztéshez az egyetemi tanulmányaim során szerzett PL/SQL ismereten túl szükséges volt a fent említett fejlesztőeszközök (Oracle Designer 2000, Oracle Forms 6i, Oracle Reports 6i) mélyebb ismerete, ezekhez elsősorban Oracle dokumentációk tanulmányozásán keresztül jutottam.

Munkám során a felhasználói felület elkészítésén túl a szerveroldali csomagok Module API (form modulokkal kommunikáló interfész) részével és az Engine API (üzleti logika) bizonyos részeivel foglalkoztam. A rendszer jelenleg bevezetés előtti stádiumban van. Bevezetőmben végül szeretnék köszönetet mondani Dr. L. Nagy Éva tanárnőnek, továbbá a National Instrumentsnek a szoftver fejlesztésében nyújtott segítségért és támogatásért.

2. A NITIL fogalomrendszere

Az ITIL módszertan fogalomrendszeréből a rendszer felhasználja a konfigurációs elem típus (`configuration item type`) fogalmát. A nyilvántartandó elemeket kategorizáljuk, a kategóriák pedig hierarchiába rendeződnek. A hierarchia csúcsán egy tetszőleges konfigurációs elem típus áll, neve `something`. Ez minden további kategória őse, nem törölhető. A fa tetszőleges mélységig bővíthető.

A rendszer a kategóriák jól definiált tulajdonságait tárolja. Tulajdonság típusokat különböztetünk meg (`property type`), mely lehet egy képernyőméret, IP cím stb. A tulajdonság típusok kategóriákhoz rendelhetőek. A hozzárendelt tulajdonságtípus lesz a tulajdonság (`property`). A fenti viszonyt szemlélteti az alábbi példa. A képernyő méret tulajdonságtípus mind a CRT monitorhoz, mind az LCD monitorhoz hozzárendelhető. A már hozzárendelt tulajdonságtípus kapja a tulajdonság nevet. A hozzárendelés folyamán eldönthetjük, hogy a tulajdonság megadása az adott kategória minden letárolt eleme számára kötelező-e. A tulajdonságok öröklődnek: az elemtípusok fájában egy elem leszármazottja örökli ősének minden tulajdonságát.

Bizonyos tulajdonságok esetén célszerű definiálni egy értékhalmazt, ahonnan az adott tulajdonság értéket kaphat. Ez lesz a `property value`. Egy lehetséges példa lehet az USB portok számszerűsítése: 1, 2, 4, 8. Az értékhalmaz tulajdonságfüggő, azaz egy laptop azonos tulajdonságához más értékhalmaz tartozhat, mint egy asztali PC esetén. Lehetőség van az egyes értékhalmazokon belül alapértelmezett érték kijelölésére.

A konfigurációs elem típusok konkrét előfordulásai a konfigurációs elemek (`configuration item`, röviden `ci`). Egy konfigurációs elem minden esetben rendelkezik egy névvel, státusszal, továbbá egy érvényességi idővel. A név-státusz pár csak az érvényességi időn belül él. A konfigurációs elemek neve mindig egyedi. A státusz a változásmenedzsment és nyomkövetés szempontjából fontos információ. Egy elem megrendelt (`ordered`) vagy tervezett (`planned`) státuszú. A tervezett állapotból egy vezetői jóváhagyást követően jóváhagyott (`approved`) állapotba kerül. Jóváhagyást követően egy tesztelési periódus veszi kezdetét, mely után vagy használatba kerül (`in use`)

vagy ismételten tervezett lesz. Egy elem használatba kerülhet a rendelést követően is. Miután a megrendelést teljesítették, az elem raktáron van (`in stock`), ahonnan használatba kerül. Az elem életútjának bejefezését jelenti a halott (`end of life`) állapot.

A konfigurációs elem típusok tulajdonságainak egy előfordulása az attribútum (`ci attribute`). A tulajdonság sohasem hordoz konkrét értéket, az attribútum ellenben mindig. A tulajdonság-attribúrum viszonyra példa ismét az USB portok száma. A laptophoz mint kategóriához rendeljük hozzá az USB portok száma tulajdonságtípust. A hozzárendelés innentől egy tulajdonságot definiál. Egy nyilvántartott laptop esetén ez a tulajdonság attribútum formájában megkapja a számszerűsített 2 értéket. Az attribútum tehát rendelkezik egy értékkel (`value`), mely származhat a tulajdonság értékhalmazából, valamint státusszal és érvényességi idővel. A státusz attribútumok esetén `planned`, `approved`, `tested` vagy `done` lehet. Az attribútum értéke és státusza is csak az érvényességi időn belül él. Egy esetleges változtatást követően archiválásra kerül az adott attribútum, azaz a régi változat továbbra is az adatbázisban marad. Az archivált attribútumok segítségével rekonstruálható a konfigurációs elem teljes életútja.

A kategória, tulajdonságtípus, tulajdonság fogalmak bevezetése nagyfokú rugalmasságot biztosít: Szűkíthetjük a kategóriák számát, ezzel párhuzamosan megnövelhetjük a kategóriákhoz rendelt tulajdonságok számát, vagy a fordítottját alkalmazzuk. Egy részletes kategorizálás során (ahol pl. a monitorvezérlő kártya, a hálózati adapter, a processzor önálló kategóriát képez) problémát jelenthet a teljes számítógépes konfiguráció rekonstruálása. Ismernünk kell, hogy mely elemek tartoznak azonos konfigurációba. A rendszer ennek feloldására kapcsolattípus fogalmat vezet be (`relation type`). A kapcsolattípus a tulajdonságtípushoz hasonlóan lehetséges kapcsolatokat nevez meg (pl. tartalmazás). A definiált kapcsolattípusokat kategória párokhoz rendelhetjük. A hozzárendelést követően típusrelációkról beszélünk (`type relation`). A típusrelációra példa az `installed on` reláció típus hozzárendelése a `software` és `computer system` kategóriákhoz. Ezzel írható le, hogy egy szoftvertermék egy számítógépes rendszerre van telepítve.

A típusrelációk előfordulásai a relációk (`relation`). Reláció két létező konfigurációs elem között létesíthető, amennyiben a két elem kategóriája között létezik a megfelelő típusreláció.

A reláció az attribútumhoz hasonlóan státusszal és érvényességi idővel bír. Változás esetén archiválásra kerül.

A konfigurációs elemek nyilvántartása során fontos szempont volt az elemek keresésének egyszerűsítése. Így a kategóriákhoz hasonlóan hierarchikusan tartjuk nyilván az elemek lehetséges helyeit, a lokációkat (`location`). Az ős lokáció az `earth` nevet kapta. Minden további lokáció ennek a leszármazottja. Speciális szerepű továbbá a `stockroom`, ahová a raktáron lévő elemek kerülnek. Az üzleti logikai megkötés raktáron lévő elemnél a módosíthatóság letiltása. Azaz `in stock` állapotú elem attribútumai nem módosíthatóak. Bizonyos elemek (pl. projektorok) funkciójukból adódóan állandó mozgásban vannak. Ezek helyét jelzi a `virtual` lokáció. Más elemeknél a hely nem meghatározható (pl. szoftverek), ennek jelzésére az `na` lokáció szolgál.

Az elemhez rendelt lokáció a pozíció (`position`) nevet viseli. Egy elem életútja folyamán több pozícióban is tartózkodhat, így a pozíció a státuszon kívül egy érvényességi idővel is bír. A státusz teljesen analóg az attribútumoknál megismerttel. Azaz lehet tervezett, jóváhagyott, tesztelt és kész állapot. A pozíció az attribútumokhoz hasonlóan archiválódik.

Párhuzam vonható az objektum-orientált világ osztály - példány fogalma és a törzsadat - nyilvántartott elem adatai között. Az osztály mint konfigurációs elem típus OO értelemben attribútumokat tartalmaz. Ezek az attribútumok lesznek a tulajdonságok ebben a fogalomrendszerben. Az egyes attribútumok típusa szintén egy jól definiált halmazból kerülhet ki, a tulajdonság típusok halmazából. A leszármazott osztály örökli ősének minden attribútumát, így a leszármazott konfigurációs elem típus is örökli ősének minden tulajdonságát. Az osztály egy konkrét példánya lesz a konfigurációs elem – attribútum páros.

3. A NITIL táblaszerkezete

A piacon előforduló, hasonló funkcionalitással bíró szoftverek előre definiált kategóriákat használnak, úgymint hálózati csatoló, monitor, processzor, stb. Az egyes kategóriák elemeit önálló táblákban tárolják, így relatíve kis adatmennyiség kerül az egyes alaptáblákba (pl. Microsoft SMS). Ez a megvalósítás feltételezi a kategóriák definiálását már a fejlesztés kezdetekor.

A NITIL táblaszerkezete ezzel szemben törzsadatokra és elemhez kötődő adatokra korlátozódik. A fogalomrendszerben szereplő konfigurációs elem típus, tulajdonság típus, tulajdonság, értékhalmaz, reláció típus, típusreláció és lokáció törzsadatok. A konfigurációs elemek, attribútumok, relációk, pozíciók megadásához a külső kulcs hivatkozások miatt előzetesen szükséges a törzsadatok definiálása. Az Oracle Designer által generált táblák kivételével az elsődleges kulcsot a táblák `id` oszlopa adja. Értéket a `nit_seq` szekvenciából kap. A táblák része továbbá négy audit oszlop: `creation_date`, `created_by`, `last_update_date`, `last_updated_by`. Célja egyszerű logging információ tárolása, azaz nyilvántartjuk a rekord létrehozásának dátumát és a létrehozó adatbázis-felhasználót, továbbá az utolsó módosítás dátumát és az azt végző felhasználó nevét. A táblákhoz opcionálisan tartozhat egy `description` és `comments` oszlop. Előző a tábla rekordjához kötődő részletes leírást tartalmaz, utóbb opcionális megjegyzést.

A konfigurációs elem típus mint objektum-orientált világ osztálya a **`nit_ci_types`** táblában tárolódik. A táblába adatai között fennálló hierarchiát a `cit_id` külső kulcs biztosítja, mely a tábla elsődleges kulcsát, az `id` oszlopot hivatkozza. Célja az öröklődés biztosítása, azaz segítségével tudjuk hivatkozni a közvetlen őst (pl. `workstation` közvetlen őse a `computer system`). Egy kategóriáról a néven (`name`) és rövid néven (`abbreviation`) kívül tárolásra kerül az üzleti logika szempontjából fontos három adat: az adott kategóriába tartozó elemek raktározhatóak-e (`could_in_stock`), leltárba vehetőek-e (`could_in_inventory`) illetve rendelkeznek-e megrendelési számmal (`need_po`). A `naming_pattern` oszlopnak a kategóriához tartozó konfigurációs elemek elnevezésénél van szerepe. A megjelenítésnél játszik szerepet az `order_in_list` sorszám és az

icon_name. A közös őssel rendelkező kategóriák esetén a sorszám határozza meg a megjelenítés sorrendjét a felhasználói felületen, így a gyakran használt elemekhez célszerű alacsony sorszámot rendelni. Az icon_name az elemet reprezentáló ikon nevére utal.

A nyilvántartandó tulajdonság típusokat szintén metaadatok formájában rögzítjük. A **nit_ci_property_types** tábla szolgál a lehetséges tulajdonság típusok megadására. A name oszlop értéke minden tulajdonságtípusra nézve egyedi. A type_name oszlop értéke adatbázis adattípus neve lehet: varchar2, number, boolean. Értékétől függően végrehajtható egy alkalmas típuskonverzió az attribútum által képviselt érték feldolgozásánál.

A **nit_ci_properties** tábla kapcsolótábla szerepét tölti be, a fogalomrendszerben tárgyalt tulajdonságokat tárolja. A már korábban definiált kategóriák és tulajdonság típusok közötti M:N kapcsolatot teremti meg. A tábla külső kulcsai a nit_ci_types táblát hivatkozó cit_id és nit_ci_property_types táblát hivatkozó cipt_id és az elsődleges kulcsot hivatkozó cip_id és cip_id_ancient. Utóbbi két oszlop a kategóriák közötti öröklődés miatt indokoltak. A cip_id a közvetlen ős kategória, míg a cip_id_ancient a legtávolabbi ős kategória azonos tulajdonság típushoz tartozó tulajdonságát azonosítja. A required oszlop értékétől függően lesz az adott tulajdonság megadása kötelező vagy opcionális. Ha valamely tulajdonság esetén a mező 'Y' értékű, úgy a kapcsolódó kategóriába tartozó konfigurációs elemek felvitele nem végezhető el anélkül, hogy az adott tulajdonság számára ne szolgáltatnánk értéket (pl. kötelezővé tesszük a leltári szám megadását). Egyes tulajdonságok egy előre definiált halmazból kaphatnak értéket. Ennek jelzésére szolgál a set_like oszlop.

A nit_ci_properties tábla set_like oszlopának 'Y' értéke esetén a hivatkozott tulajdonság **nit_ci_property_values** táblából kaphat értéket. A tábla cip_id oszlopa a külső kulcs, így kategóriánként lehet értékhalmazokat definiálni (pl. míg CRT monitorok esetén a képernyőméret lehet 14, 15, 17, 19, 21 inch, laptop megjelenítő esetén a 17, 19, 21 értékek nem szerepelnek). A value oszlop hordozza a konkrét értéket. Az is_default oszlop egy alapértelmezett érték kijelölésére szolgál ('Y', 'N' lehetséges

értékekkel). Az `order_in_list` oszlop sorszáma definiálja az elemek megjelenítésének sorrendjét a felhasználói felületen.

A fogalomrendszerben szereplő kapcsolattípusok a **`nit_cit_relation_types`** táblában tárolódnak. A kapcsolattípusok neve mindig egyedi. Reláció típusokon belül három előre definiált típus mögött húzódik üzleti logika: `contains`, `installed on` és `connected to`. A `contains` reláció esetén például a tartalmazó konfigurációs elem pozícióváltása magával vonja a tartalmazott elem pozícióváltását is.

A **`nit_ci_type_relations`** tábla tölti be a kategóriák és reláció típusok közötti kapcsolótábla szerepét. Egy reláció típust egy kategória párhoz kapcsol. A külső kulcsokat a `citrt_id`, `cit_id_from`, `cit_id_to` oszlopok adják. Az `order_in_list` oszlop a megjelenítés sorrendjét befolyásolja azonos kategória párok esetén.

A kategóriákhoz hasonló felépítésű a lokációkat tartalmazó **`nit_locations`** tábla. Felépítését tekintve teljesen azonos a kategóriák alaptáblájával. A hierarchiát az `lcn_id` külső kulcs biztosítja, mely a tábla `id` oszlopát hivatkozza. Mivel minden elem őse az `earth` lokáció, így az `lcn_id` egyedül ezen elem esetén lesz `NULL` értékű.

A rendszer felhasználóinak adatait a **`nit_users`** táblában tartjuk nyilván. A felhasználói néven (`username`) és teljes néven (`realname`) kívül e-mail címet (`email`) és szerepkört (`role`) tárolunk. Az e-mail cím tárolása képessé teszi a rendszert elektronikus levél küldésére egy esetleges változás jóváhagyása érdekében vagy bizonyos hibák elhárítása esetén. A szerepkör definiálása a NITIL további fejlesztésére ad lehetőséget, bizonyos formok és műveletek elrejthetővé válnak a jogkörrel nem rendelkező felhasználók számára.

Az OO világ példány fogalmával analóg adatokat tartalmazó táblák mindegyike érvényességi időt tart nyilván, ezek a `valid_from`, `valid_until` oszlopokban tárolódnak. Az archivált és élő adatok közötti különbség jelzésére egy `obsolete` oszlopot használunk. A táblák külső kulcsa továbbá egy `opn_id` oszlop, mely a nit_operations tábla elsődleges kulcsát hivatkozza.

A nyilvántartott elemeket a **nit_config_items** táblában találjuk. A tábla külső kulcsa a cit_id oszlop, mely az elem kategóriáját azonosítja. Az élő konfigurációs elemeket az obsolete oszlop 'L' (live) értéke jelzi. A konfigurációs elem státuszának end of life állapota az elem életútjának végét jelenti, az obsolete oszlop ekkor 'D' (deleted) értéket kap. A név és státusz változása esetén az archiválás folyamata az attribútumok táblájába helyezi az elavult értékeket.

A **nit_ci_attributes** tábla tárolja a fogalomrendszerben megnevezett attribútumokat. Az attribútum mindig egy tulajdonság számára szolgáltat értéket és egy konfigurációs elemhez kötődik. Ezért a tábla külső kulcsai között találjuk a nit_config_items és nit_ci_properties táblákat hivatkozó ci_id és cip_id oszlopokat. Az attribútum értéke értékhalmazból is származhat, amit a cipv_id hivatkoz (nit_ci_property_values tábla valamely elsődleges kulcs értéke vagy NULL). Az érték a data oszlopban tárolódik. Az obsolete oszlop értéke a konfigurációs elemeknél megismerteken felül lehet még 'A', melynek jelentése archív.

Az elemeket összekapcsoló relációk a **nit_ci_relations** táblában találhatóak. A tábla külső kulcsa a típusrelációt hivatkozó citrt_id, továbbá a kapcsolatban résztvevő elemeket hivatkozó ci_id_from és ci_id_to oszlopok. Az obsolete oszlop által felvehető értékek megegyeznek az attribútumoknál megismertekkel. Az érvényét vesztett adatok archiválva tovább élnek az adatbázisban.

Az elemek pozícióit a **nit_positions** tábla hordozza. Mivel a pozíció konfigurációs elemet és lokációt kapcsol össze, így a tábla külső kulcsai a nit_config_items tábla elsődleges kulcsát hivatkozó ci_id és a nit_locations tábla elsődleges kulcsát hivatkozó lcn_id. A pozícióknál alkalmazott obsolete értékek az attribútumokéval azonosak. A tábla rekordjai módosítás előtt minden esetben archiválódnak.

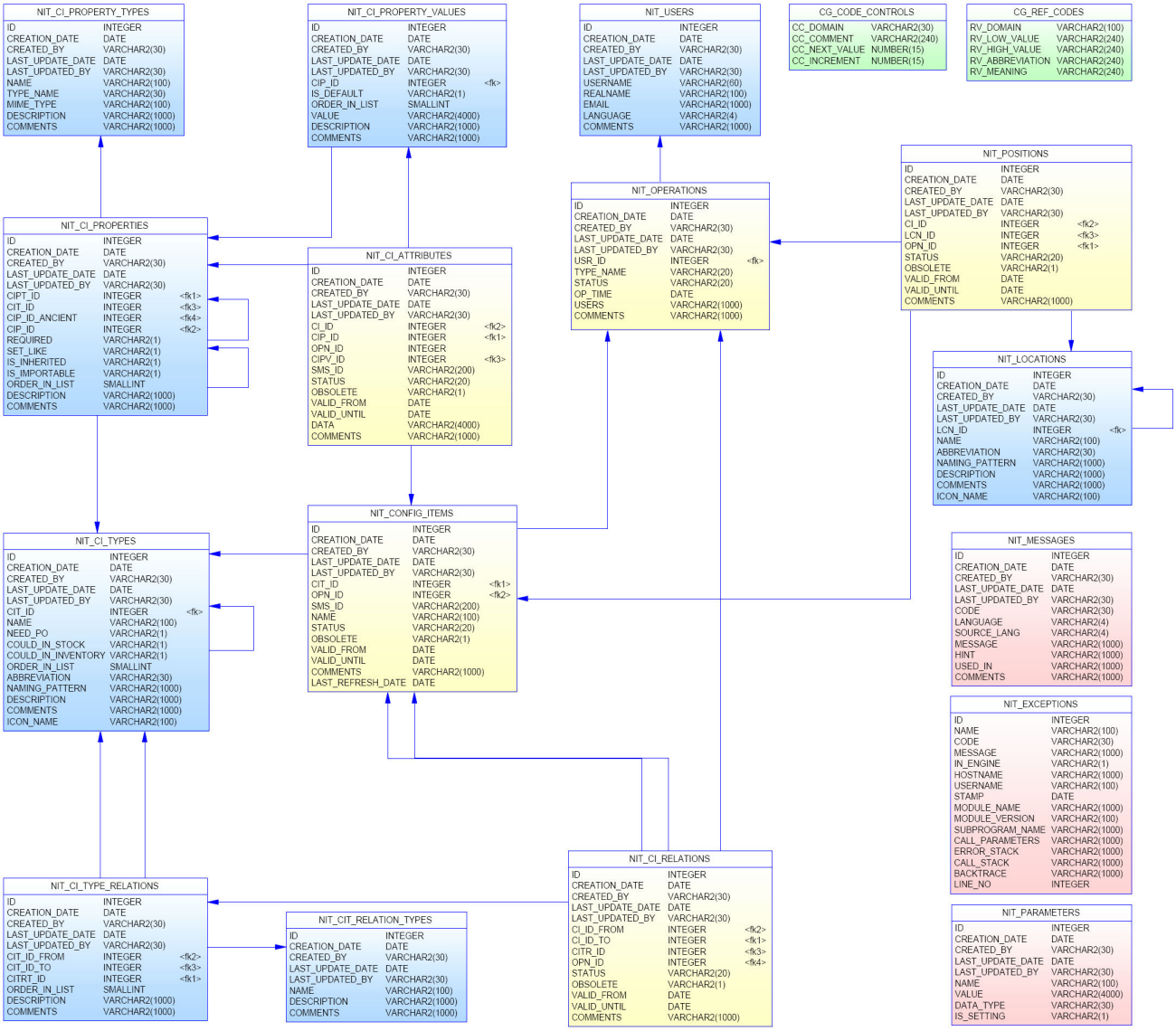
Speciális szerep jut a **nit_operations** táblának. A rendszer a nem metaadatokkal kapcsolatos műveleteket operációkba szervezi. Az operációt minden esetben egy felhasználó indítja, azonosítására alkalmas a tábla usr_id külső kulcsa. Az operáció típusa

(type_name) lehet rutinszerű (routine), tervezett (requested) vagy vészhelyzet elhárítás (emergency). A típus hatással van az üzleti logika működésére. Az operáció rendelkezik továbbá státusszal (status). Indítást követően elindított (started) állapotba kerül, befejezését követően pedig sikeres (finished) vagy sikertelen (failed) lehet. Az op_time oszlop mindig az indítás időpontját hordozza.

A kivételek loggolására alkalmas a **nit_exceptions** tábla. A Table API alaptáblát kezelő csomagja és az Engine API csomagjai által dobott kivételekhez kötődő rekordokat tárolja. A rekord alapján beazonosítható az alkalmazást futtató számítógép (hostname) és felhasználó (username), a hívó modul (module_name, module_version) és alprogram (subprogram_name). A code és message oszlopok tartalma az SQLCODE és SQLERRM értékével azonos.

A kivételekhez kötődő, felhasználói felületen megjelenített üzenetek tárolására szolgál a **nit_messages** tábla. A code oszlop tárolja a kivételkódot (NIT-xxxxx alakban), a language oszlop azonosítja a message és hint oszlopban tárolt üzenet és tipp nyelvét. A used_in oszlop információt ad a kivétel előfordulásáról, azaz mely csomagtörzsben váltódik ki.

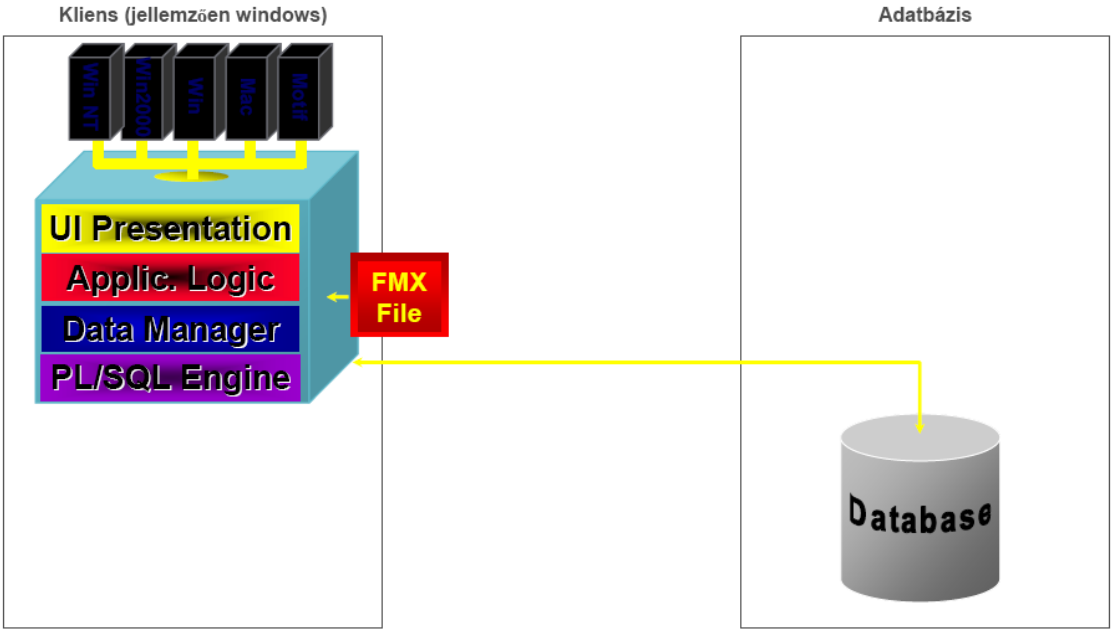
Az Engine API-ban és form modulokban használt paramétereket és értékeit tárolja a **nit_parameters** tábla. A kulcs-érték pár bevezetésével elkerülhető a hard-coding az Engine API csomagjaiban illetve azok újrafordítása a paraméter értékének változása esetén.



1. ábra: Az alkalmazás táblaszerkezete

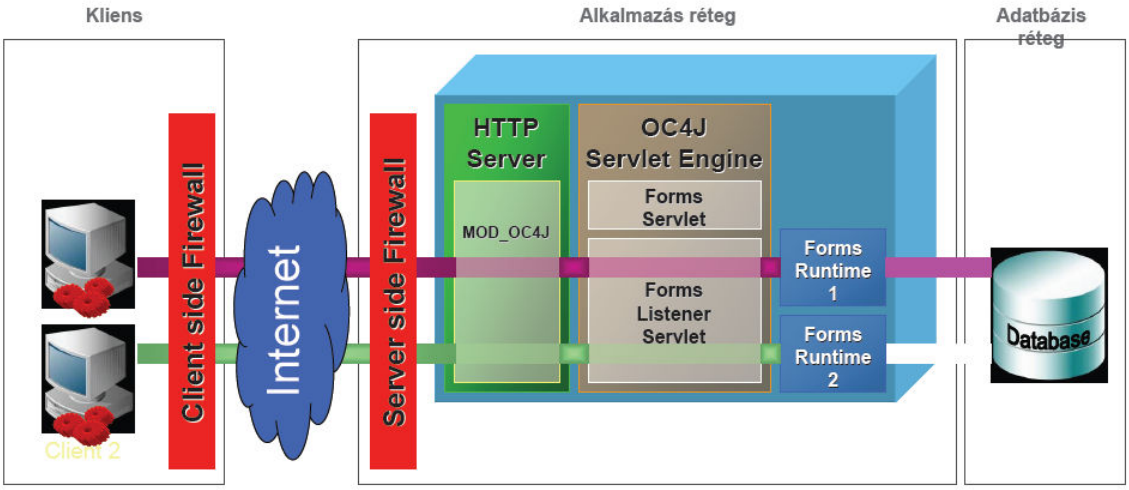
4. Oracle Forms alkalmazások környezete

A Form Builder 6i verziójával bezárólag az Oracle Forms alkalmazások kétrétegű architektúrában futnak. A kliens-szerver modell esetén a szerver feladata az adatbázis-szerver futtatására korlátozódik. Az alkalmazás üzleti logikája szerveroldali adatbázis-objektumokban (triggerek, csomagok, tárolt alprogramok) vagy a form modulokban kerül implementálásra (az Oracle az üzleti logika szerveroldali megvalósítását javasolja). A felhasználói felület C++ alapú, mely a 6i Release 2 verziótól kiegészül Java támogatással.



2. ábra: Forms alkalmazások felépítése kétrétegű architektúra esetén

A háromrétegű architektúránál egy, a kliensoldali JVM alatt futó applet feladata a felhasználói felület megjelenítése. A form modulok futtatása egy köztes réteg (middle-tier) feladata, melyen az Oracle Application Server fut. A beérkező igények kiszolgálása mellett ez a réteg tartja a kapcsolatot az adatbázissal, továbbá ide kerül a form modulhoz kapcsolódó üzleti logika. Az applet alapú megjelenítés tetszőleges platformon lehetővé teszi a Forms alkalmazások futtatását (alkalmas böngésző segítségével). Csökken a fenntartási költség, elegendő egyetlen Forms Server konfigurálása és menedzselése.

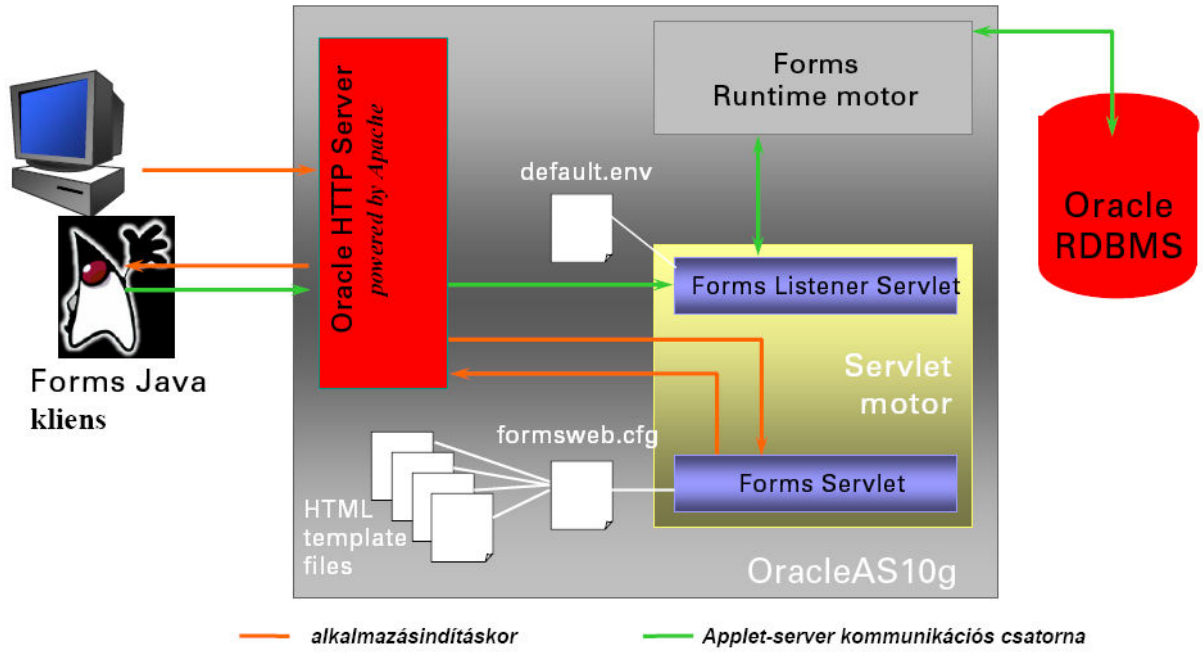


3. ábra: Forms alkalmazások felépítése háromrétegű architektúra esetén

Webes környezetben nem működnek a kétrétegű architektúránál még alkalmazható Visual Basic és ActiveX vezérlők. Ezeket célszerű JavaBeanre vagy Pluggable Java Class-ra cserélni. Triggerekre vonatkozó korlátozás a WHEN-MOUSE-MOVE, WHEN-MOUSE-ENTER, WHEN-MOUSE-LEAVE triggerek elhagyása (oka a generált hálózati forgalom). Beépített alprogramok működése módosul: a HOST, TEXT_IO, READ_IMAGE_FILE ezúttal az Application Serveren fut, a kliensoldali funkciók elérésére a CLIENT_HOST, CLIENT_TEXT_IO alkalmazható.

5. Forms alkalmazások futtatása Oracle Application Server 10g-vel

Az Oracle Application Server 10g (továbbiakban OAS) része az Apache alapú Oracle HTTP Server (OHS). A kliensoldalról érkező igényeket az OHS fogadja, mely a mod_oc4j modul segítségével kommunikál az Oracle Components for J2EE-vel (OC4J). A mod_oc4j egy load balancer, feladata az OHS és OC4J közötti routing biztosítása, mely lehet round robin, véletlenszerű és metrika alapú. A OC4J önálló, szabadon felhasználható J2EE konténer. Az OAS 10g-be integrált verzió J2EE 1.4 kompatibilis. A szabadon felhasználható változat szolgáltatásai (web alkalmazások, webszolgáltatások) mellett elérhetők további osztálykönyvtárak, finomított loggolás, teljes integráció a mod_oc4j-vel. Az OC4J Servlet motorjában két servlet kapcsolódik a Forms szolgáltalásokhoz: egy Forms Servlet, mely a kliensoldali alkalmazás elindításakor aktív, továbbá egy Forms Listener Servlet, mely az alkalmazás futása közben szolgálja ki az beérkező igényeket. Ez utóbbi kapcsolódik a Forms Runtime Engine-hez, mely az adatbázissal tartja a kapcsolatot.



4. ábra: A kommunikáció útja az architektúra komponensei között

6. Röviden a Table API-ról

Az adatbázistáblák fölött közvetlenül a Table API (továbbiakban tapi) réteg helyezkedik el. Az Oracle Designer által generált Table API egy adatbázis objektumokból álló halmaz, mely tartalmaz egy, az alaptáblát kezelő csomagot, továbbá 12 adatbázis triggert (sor- és utasításszintű triggerek INSERT, DELETE, UPDATE utasításra) és egy nem generált táblát a felmerülő kivételek tárolására. A kivételek táblája (`cg$errors`) szükség esetén szkript segítségével létrehozható, de lehet szinonima is. A triggerek törzsében az alaptábla fölé definiált csomag alprogramjai kerülnek meghívásra. A Designer továbbá lehetőséget biztosít a DML utasítást megelőző, illetve azt követő tevékenységek definiálására (PRE-EVENT, BEFORE-EVENT, POST-EVENT, AFTER-EVENT). A tevékenység egy kódrész, melyet automatikusan beilleszt a generált csomagba. Alternatív megoldás az üzleti logika triggertörzsben történő implementálására. A felépülő hívási lánc miatt a tapi hívások determinisztikusak ellentétben a triggerek végrehajtásával, az alaptábla fölé generált csomagok felépítése pedig táblától függetlenül azonos szerkezetű.

Az alaptábla fölé generált tapi neve minden esetben `cg$<alaptábla_név>` alakú. A csomagfej fontosabb elemei:

- `cg$row` típus, mely `<alaptábla>%ROWTYPE` típusú rekord.

```
cg$row nit_users%ROWTYPE;
```

- `cg$row_type` típus, mely az alaptábla egy sorát reprezentálja. A `cg$row` típus használatához képest az egyetlen különbség a `the_rowid` mező megjelenése, melyben opcionálisan megadható a sorazonosító. A tapi alprogramjainak tipikus bemenő paramétere a `cg$row_type` típusú rekord, melyben átadásra kerülnek az új vagy módosított értékek.

```
TYPE cg$row_type IS RECORD
     (ID cg$row.ID%TYPE
     ,CREATION_DATE cg$row.CREATION_DATE%TYPE
```

```
      ,CREATED_BY cg$row.CREATED_BY%TYPE
      ,LAST_UPDATE_DATE cg$row.LAST_UPDATE_DATE%TYPE
      ,LAST_UPDATED_BY cg$row.LAST_UPDATED_BY%TYPE
      ,USERNAME cg$row.USERNAME%TYPE
      ,REALNAME cg$row.REALNAME%TYPE
      ,EMAIL cg$row.EMAIL%TYPE
      ,LANGUAGE cg$row.LANGUAGE%TYPE
      ,COMMENTS cg$row.COMMENTS%TYPE
      ,ROLE cg$row.ROLE%TYPE
      ,the_rowid ROWID);
```

- `cg$ind_type` típus, mely a DML műveletekhez tartozó indikátor szerepét tölti be. A rekord mezői rendre `BOOLEAN` típusúak. A mező TRUE értéke jelzi a tapi alprogramjai számára, hogy az adott mezőérték módosult.

```
 TYPE cg$ind_type IS RECORD
      (ID BOOLEAN DEFAULT FALSE
      ,CREATION_DATE BOOLEAN DEFAULT FALSE
      ,CREATED_BY BOOLEAN DEFAULT FALSE
      ,LAST_UPDATE_DATE BOOLEAN DEFAULT FALSE
      ,LAST_UPDATED_BY BOOLEAN DEFAULT FALSE
      ,USERNAME BOOLEAN DEFAULT FALSE
      ,REALNAME BOOLEAN DEFAULT FALSE
      ,EMAIL BOOLEAN DEFAULT FALSE
      ,LANGUAGE BOOLEAN DEFAULT FALSE
      ,COMMENTS BOOLEAN DEFAULT FALSE
      ,ROLE BOOLEAN DEFAULT FALSE);
```

- `cg$pk_type` típus, mely az alaptábla elsődleges kulcsát és sorazonosítóját tartalmazza. Tipikusan a törlésnél van jelentősége.

```
TYPE cg$pk_type IS RECORD
     (ID cg$row.ID%TYPE
     ,the_rowid ROWID);
```

- `cg$table_type` típus, mely alaptábla sortípusú értékek index alapú táblája

```
TYPE cg$table_type IS TABLE OF NIT_USERS%ROWTYPE INDEX
BY BINARY_INTEGER;
```

- `cg$tableind_type` típus, mely indikátor típusú értékek index alapú táblája

```
TYPE cg$tableind_type IS TABLE OF cg$ind_type INDEX BY
BINARY_INTEGER;
```

A tapi DML műveleteket megvalósító fontosabb alprogramjai:

- `ins`: A `cg$rec` paraméterként átadott táblasor beszúrását végzi el. Amennyiben a `cg$ind` indikátor adott mezője `TRUE` értékű, úgy a `cg$rec` ugyanezen mezőjének értéke kerül beszúrásra az alaptábla adott oszlopába. `FALSE` vagy `NULL` esetén egy opcionális kezdőérték kerül hozzárendelésre, ennek hiányában `NULL` érték szúródik be. Az `INSERT INTO` utasítást követően a `cg$rec` a beszúrt táblasor értékeit hordozza, beleértve a `the_rowid` sorazonosítót.

```
PROCEDURE ins(cg$rec IN OUT cg$row_type
              ,cg$ind IN OUT cg$ind_type
              ,do_ins IN BOOLEAN DEFAULT TRUE);
```

- `upd`: Az eljárás feladata az `update` végrehajtása. A `cg$rec` paraméterben átadott értékekre frissíti az alaptábla egy sorát. A táblasort a `cg$pk` paraméter `the_rowid` mezője, `NULL` esetén `id` mezője alapján hivatkozzuk. Amennyiben a paraméter értéke `NULL`, úgy a `cg$rec` `the_rowid`, `id` mezőit veszi alapul. A `cg$ind` indikátor

adott mezőjének TRUE értéke esetén módosul az alaptábla sorának ugyanezen nevű oszlopa, FALSE vagy NULL esetén ezt az oszlopot figyelmen kívül hagyja.

```
PROCEDURE upd(cg$rec IN OUT cg$row_type
             ,cg$ind IN OUT cg$ind_type
             ,do_upd IN BOOLEAN DEFAULT TRUE
             ,cg$pk  IN cg$row_type DEFAULT NULL);
```

- del: Táblasor törlésére alkalmazható. A cg$pk paraméter the_rowid mezője, NULL esetén id mezője azonosítja az alaptábla sorát.

```
PROCEDURE del(cg$pk  IN cg$pk_type
             ,do_del IN BOOLEAN DEFAULT TRUE);
```

- lck: Az eljárással az alaptábla egy sora zárolható. A cg$old_rec paraméter tükrözi a táblasor zárolást megelőző állapotát. Az eljárás egy sikeres zárolást követően ellenőrzi, hogy a zárolt táblasor állapota megegyezik-e a paraméterben kapott állapottal. Az összehasonlítás a cg$old_rec mezői alapján történik, a cg$old_ind indikátor figyelembe vételével. Eltérés esetén kivétel váltódik ki. A nowait_flag paraméter TRUE értéke kikényszeríti a FOR UPDATE NOWAIT záradékot a lefutó update utasításban.

```
PROCEDURE lck(cg$old_rec  IN cg$row_type
             ,cg$old_ind  IN cg$ind_type
             ,nowait_flag IN BOOLEAN DEFAULT TRUE);
```

- slct: Az eljárás szemantikája egy SELECT INTO utasítás végrehajtása. A cg$sel_rec paraméter the_rowid sorazonosítójú, NULL értéke esetén az id azonosítójú táblasort szolgáltatja.

```
PROCEDURE slct(cg$sel_rec IN OUT cg$row_type);
```

A tapi csomagfejben definiálásra kerül egy `cg$table_type` és `cg$tableind_type` típusú változó. Nagy rekordszámú insert és update esetén célszerű a `do_ins`, `do_upd` paraméterek értékét `FALSE`-ra állítani, ekkor ugyanis a tényleges dml művelet elmarad, a paraméterként kapott beszúrandó / frissítendő rekord és a kapcsolódó indikátor az előbb említett táblákban kerülnek tárolásra. A beszúrást követően egy `FORALL` ciklussal célszerű végigmenni a beágyazott táblákon, s elvégezni a tényleges dml tevékenységet. Az ilyen módon végrehajtott `BULK BIND` elkerüli a folyamatos kontextusváltást az SQL és PL/SQL motor között.

A tapi további alprogramjai lehetőséget teremtenek cascade tevékenységek definiálására, külső kulcs hivatkozások ellenőrzésére, értékek validálására.

7. Röviden az Engine API-ról

Az Engine API (továbbiakban eapi) feladata az alkalmazás üzleti logikájának megvalósítása. A törzsadatokat tartalmazó alaptáblák fölé egy csomag épül az `<alaptáblanév>_p` elnevezési konvenciót követve. Feladata az üzleti logika által diktált megszorítások kikényszerítése, esetleges cascade tevékenységek végrehajtása, elnevezési konvenciók és alapértelmezett értékek biztosítása.

Az alaptáblák audit oszlopainak kitöltése automatizált. A dml utasítás végrehajtása előtt lefut az eapi `pre_dml` eljárása, mely a felhasználónév, létrehozás és módosítás dátuma oszlopok kitöltése mellett beállítja az indikátor rekordot. Feladata továbbá az esetlegesen kitöltetlen érvényességi idő meghatározása.

Az általános tevékenységek mellett bizonyos törzsadatoknál insert és update estén megjelenik egy `pre_insert`, `pre_update` eljárás. Specifikusan az adott műveletre fókuszálva hajtanak végre extra tevékenységet. A konfigurációs elemek, attribútumok, pozíciók esetében update előtt arhiválás történik. Így biztosítható a teljes életút nyilvántartása a konfigurációs elemeknél. Az archiválás egy új rekord beszúrását jelenti a még módosítatlan, régi adatokat felhasználva. Csak miután megjelent az archív másolat az adatbázisban, fut le az update utasítás.

A dml-t követően szükséges lehet cascade tevékenységek végrehajtása. A tulajdonságok mint törzsadatok esetében egy végrehajtott változtatás hatással lehet az összes gyermek tulajdonságra. A változásokat egy `cascade_upd` eljáráshívás vezeti át az érintett rekordokra. További esetek lehetségesek insert és delete esetén.

Az eapi tartalmaz továbbá a form modulok által felhasznált logikai (visszatérési értéke `CHAR`, tartománya `'Y','N'`), valamint kollekcióval visszatérő függvényeket (a hierarhizált törzsadatok esetében visszatér az összes leszármazott elemmel).

Speciális szerepe van a `nit_core_p` csomagnak. A több csomag által hívott alprogramok gyűjteménye. Eljárásai között szerepel az audit értékeket szolgáltató `set_audit`, az

érvényességi időt beállító `set_validity`. Függvényei között szerepelnek paraméter értékeket szolgáltató `getp`, `getpi`, `getpn`, `getpd`, `getpts`. Feladatuk megkeresni az átadott paramétert a nit_parameters táblából, majd a paraméter aktuális értékét `VARCHAR2`, `INTEGER`, `NUMBER`, `DATE`, vagy `TIMESTAMP` típusúra konvertálni.

5. ábra: Kommunikáció a Table API és Engine API csomagok között

8. Röviden a Module API-ról

A Table API és Engine API rétegek fölött helyezkedik el a Module API (továbbiakban mapi). Csomagjai átjárást biztosítanak a formok által használt illetve Designer által generált rekordszerkezetek között. Az elnevezési konvenció `form_modul_név>_p`. A csomagfejben definiált rekordtípus mezői tipikusan a formokon használt, adatbázis adatmezők párjai. A mezők így gyakran több alaptáblát is felölelnek (pl. konfigurációs elemek form). A tapi-hoz hasonlóan definiálunk egy indikátor típust is.

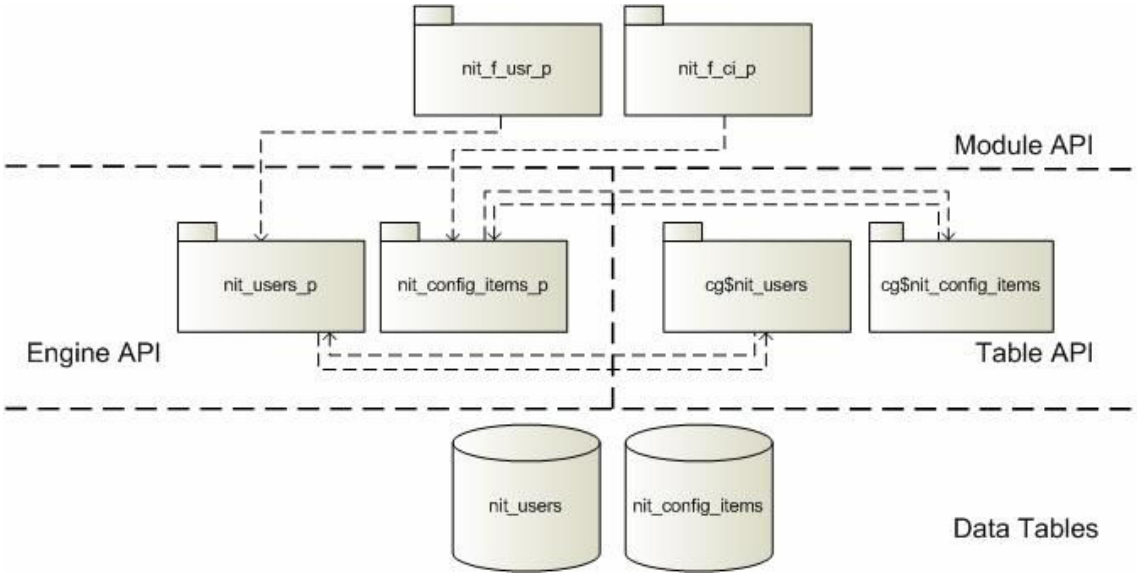
```
TYPE rt_usr IS RECORD(
         id       nit_users.id%TYPE
        ,username nit_users.username%TYPE
        ,realname nit_users.realname%TYPE
        ,language nit_users.language%TYPE
        ,email    nit_users.email%TYPE
        ,comments nit_users.comments%TYPE);

TYPE it_usr IS RECORD(
         id       BOOLEAN
        ,username BOOLEAN
        ,realname BOOLEAN
        ,language BOOLEAN
        ,email    BOOLEAN
        ,comments BOOLEAN);
```

A form modul és az adatbázis kapcsolatát a dml műveletek során egy konverzió biztosítja, melynek során a fenti rekord típusú változó mezőit megfeleltetjük a tapi által definiált `cg$row_type` típusú változó mezőinek. A folyamat teljesen analóg az indikátor típusok esetében. A tapi és mapi rekordszerkezetei között rendre a `convert_for_tapi`, `convert_for_mapi` eljárások biztosítják a konverziót értékadó utasítások segítségével.

A dml műveleteket az `ins`, `upd`, `del`, `lck` eljárások végzik, egy konverzió és egy alkalmas tapi alprogramhívás keretében. A dml műveletet követően az esetleges változások jelzésére a mapi irányában is végrehajtunk egy konverziót (pl. insertet követően értéket kap az id mező, mely a form számára lényeges információ).

Bizonyos modulok esetében szükség lehet az eapi alprogramjainak hívására (esetleges alapértelmezett név hozzárendelése, adott rekord törölhető státuszban van-e stb.). Mivel az eapi csomagjai a tapi rekordszerkezetét használják, így a hívások menete a fentiekkel azonos konverzióval párosul.



6. ábra: Az API-k csomagjainak rétegeződése

9. A form modulok felépítése

A fejlesztés során a modularitást szem előtt tartva a form modulok által használt adatforrás egyetlen esetben sem közvetlenül adatbázis tábla. Az alaptáblák fölé mindig nézeteket definiálunk, mely gyakran több alaptábla összekapcsolásából áll. A modulok adatblokkjai a nézeteket használják fel adatforrásként. Így az adatblokkok `DML adatcél` tulajdonsága szükségképpen a tranzakciós trigger értéket kapja.

A form modul a `COMMIT_FORM` beépített eljárás kiadását követően a definiálás sorrendjében végighalad az adatblokkokon. Az adatblokkban található rekordok státusza alapján dönt a dml utasítás (`INSERT`, `UPDATE`) végrehajtásáról. A rekordtörlés már a kérést követően a `DELETE` utasítás lefutását eredményezi. Tranzakciós trigger választása esetén a dml utasítás helyett a megfelelő `ON-INSERT`, `ON-DELETE`, `ON-UPDATE` trigger aktivizálódnak. A triggertörzs feladata a Module API-ban definiált `ins`, `upd`, `del` függvények hívása.

A konkrét megvalósítást a `nit_f_usr` felhasználókat adminisztráló form esetében vizsgáljuk:

A form által kezelt adatok egyetlen táblában, a `nit_users`-ben tárolódnak. A tábla fölé egy `nit_f_usr_main_v` nézetet definiálunk, mely az audit oszlopok értékeinek kivételével minden értéket lekérdez a táblából.

```
CREATE OR REPLACE VIEW nit_f_usr_main_v AS
SELECT usr.id
      ,usr.username
      ,usr.realname
      ,usr.language
      ,usr.email
      ,usr.role
      ,usr.comments
FROM nit_users usr
```

A modulban egyetlen adatblokk (usr) szerepel, melynek adatforrása a fenti nézet. DML adatcél-típusnak tranzakciós triggereket adunk meg. A form modul továbbá új programegységgel, a usr_dml csomaggal bővül. Az elnevezési konvenció az <adatblokknév>_dml formából származik. A csomag feladata, hogy előállítsa a mapi alprogramjainak hívásához szükséges rekordokat, és meghívja azt. Fontosabb alprogramjai:

- BLOCK_TO_REC: Feladata egy mapi-ban definiált rekord típusú változóba másolni az adatblokk tartalmát, valamint egy indikátor rekord készítése a dml művelet számára.

- BLOCK_TO_REC_LOCK: Az előző eljáráshoz hasonlóan a blokk tartalmát egy rekordba helyezi, az indikátor rekord minden mezőjét TRUE értékűre állítja. A zárolást készíti elő.

- REC_TO_BLOCK: Az eljárás aktualizálja az adatblokkot a kapott, mapi-ban definiált rekord típusú változó alapján.

- INS, UPD, LCK, DEL: Az eljárások a mapi azonos nevű alprogramjainak hívását végzik, a fenti eljárásokat használva. Hívás előtt az adatblokk tartalmát egy paraméterként átadott rekordba másolja, átadja továbbá a beállított indikátor rekordot. A hívást követően frissíti az adatblokk tartalmát.

A csomag alprogramjait a tranzakciós triggerek hívják. A modularitás a modulok esetében PL/SQL könyvtárak csatolásával érhető el. Egy önálló PL/SQL könyvtár kerül létrehozásra minden modulhoz, melynek neve a <modulnév>_L elnevezési konvenciót követi. Egyetlen eventhandler nevű csomagot tartalmaz. A csomag szerepe a triggertörzsek kódjainak közös helyen való tárolása, s az fmb állomány újrafordításának elkerülése a triggertörzs kódjának módosítása esetén.

Az eventhandler csomagban eljárások reprezentálják az egyes triggerek törzsét. Egyetlen paraméterük a p_event, mely paraméter a bekövetkezett eseményre utal (gyakorlatilag a trigger neve, pl. WHEN-BUTTON-PRESSED). A form szintű eseményekre az frm alprogram hívásával reagálhatunk, adatblokk szintű eseményekhez egy adatblokk nevével megegyező

alprogramot definiálunk. Elem szintű események alprogramjának neve <adatblokknév>_<elemnév> alakú. A következő példa a WHEN-NEW-FORM-INSTANCE triggerre vonatkozik:

```
PROCEDURE frm(p_event VARCHAR2) IS
BEGIN
    IF (p_event = 'WHEN-NEW-FORM-INSTANCE') THEN
    ... – triggertörzs kódja
    END IF;
END;
```

A form modulban található trigger törzsének kódja egy anonim PL/SQL blokk, mely meghívja az eventhandler csomag eljárását:

```
BEGIN
    eventhandler.frm('WHEN-NEW-FORM-INSTANCE');
END;
```

A csatolt PL/SQL könyvtár használatával elvész a lehetőség a formon elhelyezett elemek tartalmának kapcsolt változó-szerű hivatkozására (:<adatblokknév>.<elemnév>). A változó értékének lekérdezésére a NAME_IN beépített függvény használható. A függvény NULL értékkel tér vissza, ha az adott változó nem létezik. Bemenő paramétere a változónév. A COPY eljárás segítségével lehet a változók számára értéket biztosítani. Bizonyos változók (pl. SYSTEM.FORM_STATUS) értéke azonban nem módosítható.

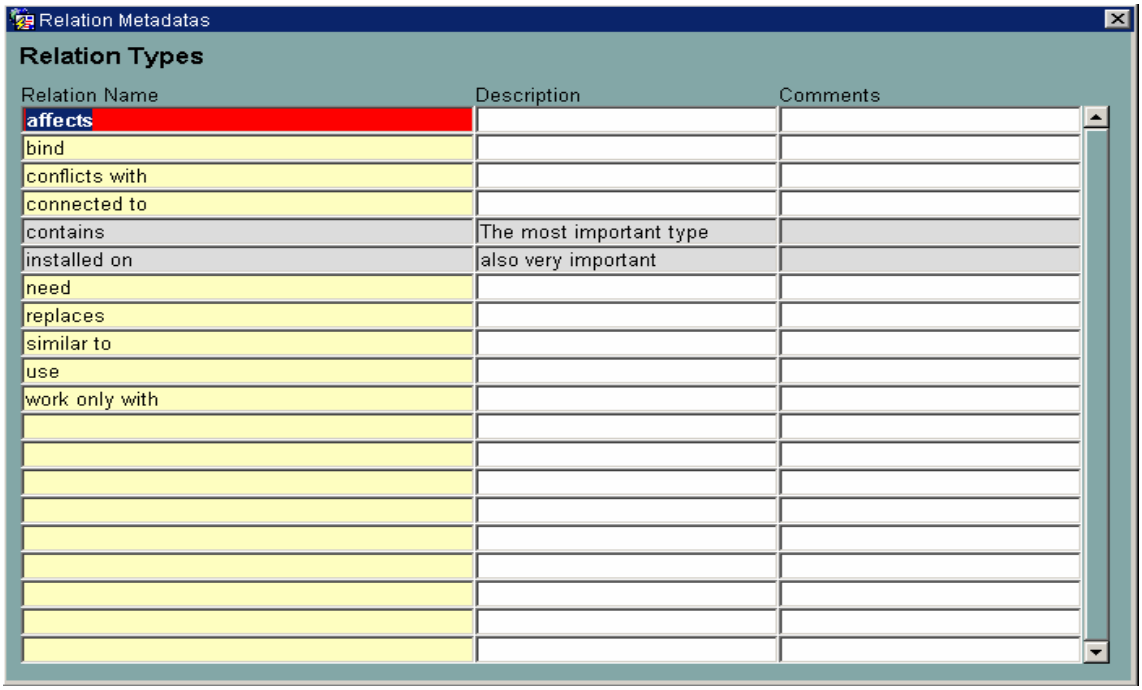
10. A megvalósított form modulok

A rendszer az adatkezelés és navigáció megkönnyítésére a Form Builder által biztosított menürendszert és eszköztárat használja fel. A változások véglegesítése a mind eszköztárban, mind menüben fellelhető `Save` funkció segítségével érhető el. Az adatblokkokban az `Enter Query` és `Execute Query` műveletek biztosítják a lekérdezés finomítását (további szűrést lehetővé téve) és ismételt lefuttatását. Lehetőség van továbbá blokk, rekord és mező szinten az egyes egységek közötti navigációra illetve adatok eltávolítására (`Clear`).

A navigálás alapja a `nit_f_main` form modul. A rendszer által nyilvántartott adatok adminisztrációja a felületen elhelyezett, nyomógombok segítségével nyíló form modulokban lehetséges. A meghívott modul a már megnyitott ablakban kerül megjelenítésre, elrejtve a hívót, mely az ablak bezárását követően válik ismét aktívvá. A hívó modul elrejtése elkerülhető a nyomógombok eseménykezelőjében hívott `CALL_FORM` eljárás alkalmas paraméterezésével (a `display` bemenő paraméter értéke legyen `NO_HIDE`).

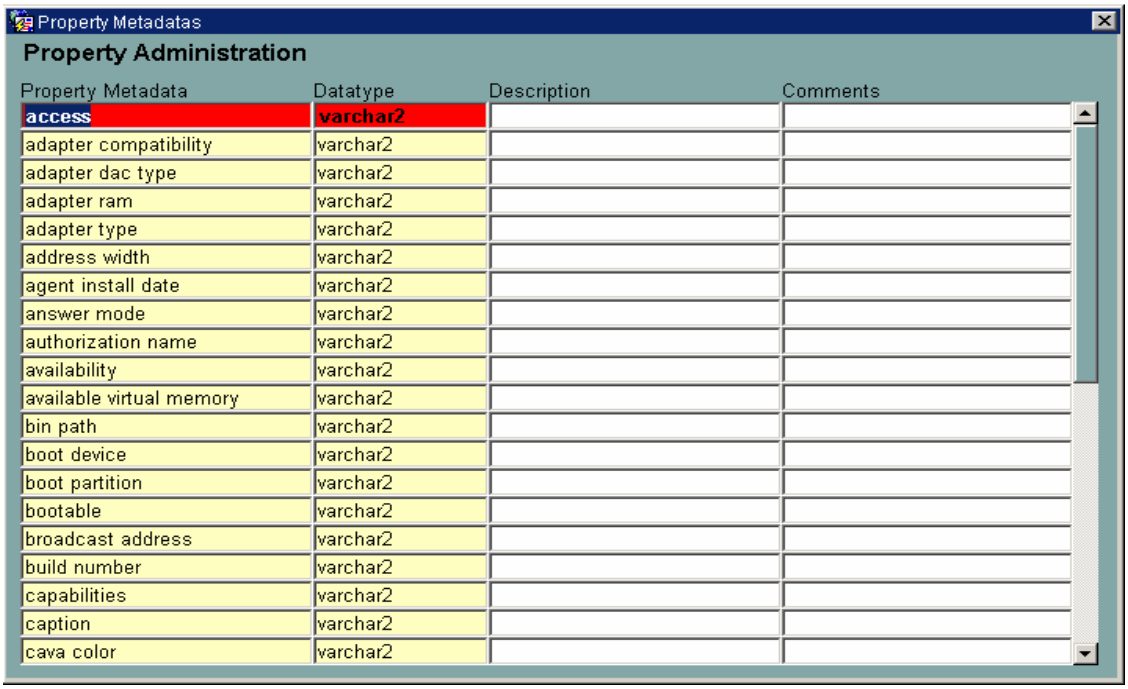
Azonos felépítést követnek a reláció típusokat, tulajdonság típusokat és felhasználókat kezelő `nit_f_citr`, `nit_f_cipt` és `nit_f_usr` form modulok. A modulok mindegyike egyetlen, több rekordot megjelenítő adatblokkot tartalmaz, mely egyetlen vásznon kerül elhelyezésre a görgetősávval együtt. A formok vizuális attribútumcsortot használnak a kötelezően megadandó illetve nem módosítható adatok megjelenítésére, így a színek újradefiniálása egyszerűen megoldható. A kötelezően megadandó mezők háttérszíne sárga, a nem módosítható rekordoké szürke.

A `nit_f_citr` form modul feladata a fogalomrendszerben szereplő reláció típusok adminisztrálása. Kötelezően megadandó a reláció neve, opcionális a leírás és megjegyzés mezők kitöltése.



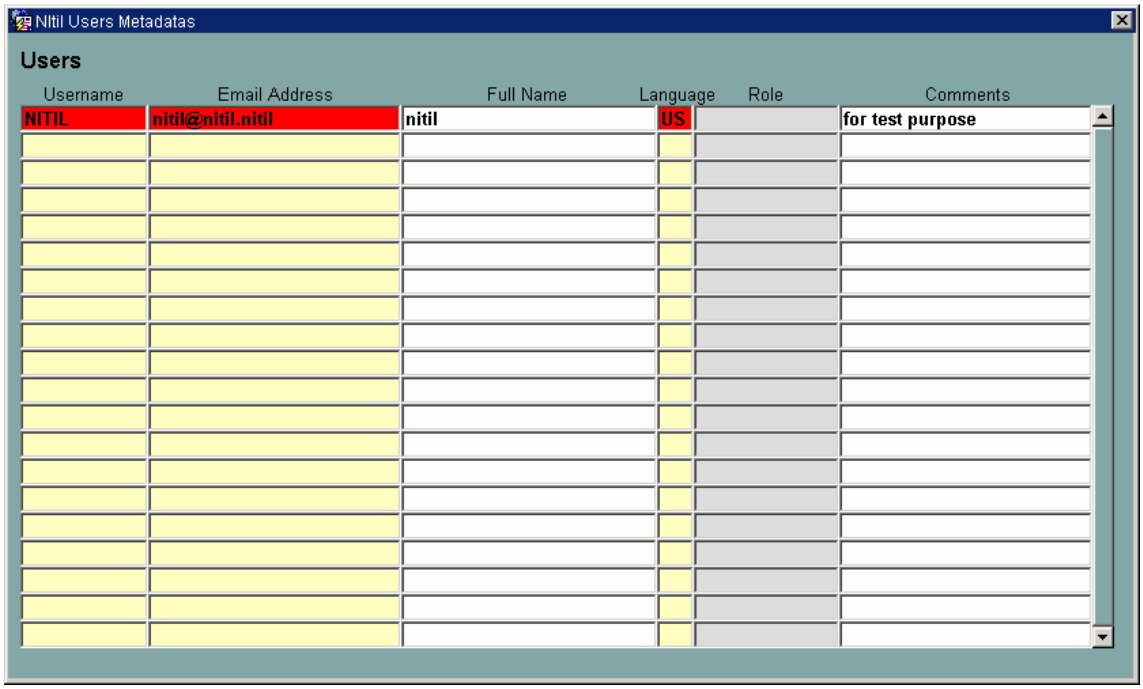
7. ábra: A nit_f_citr form képernyőképe

A `nit_f_cipt` form modul kezeli a fogalomrendszer tulajdonság típusait. Név mellett kötelező egy adattípus megadása, mely az adatfeldolgozásnál és számításoknál bír hasznos információval. Opcionális a leírás és megjegyzés mezők kitöltése.



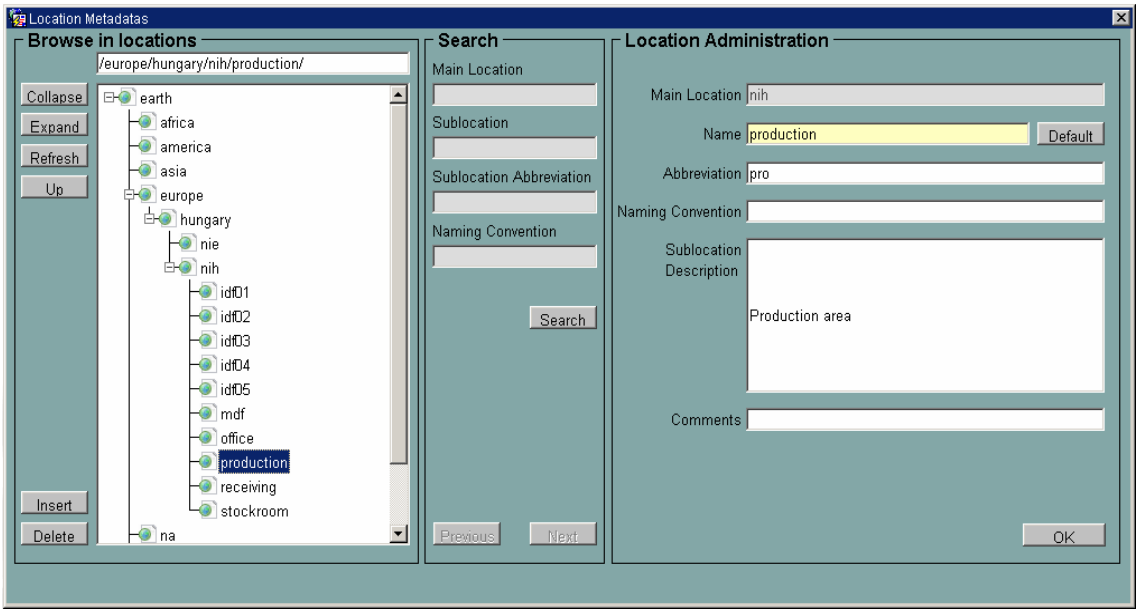
8. ábra: A nit_f_cipt form képernyőképe

A felhasználókat kezelő `nit_f_usr` form modul esetén kötelező egy név, e-mail cím és nyelv megadása. A rendszer funkcionalitása lehetővé teszi, hogy szükség esetén az érintett felhasználók elektronikus levélben értesüljenek egy változtatás bekövetkezéséről. A felhasználó teljes nevének és a megjegyzés mezőnek kitöltése nem kötelező.



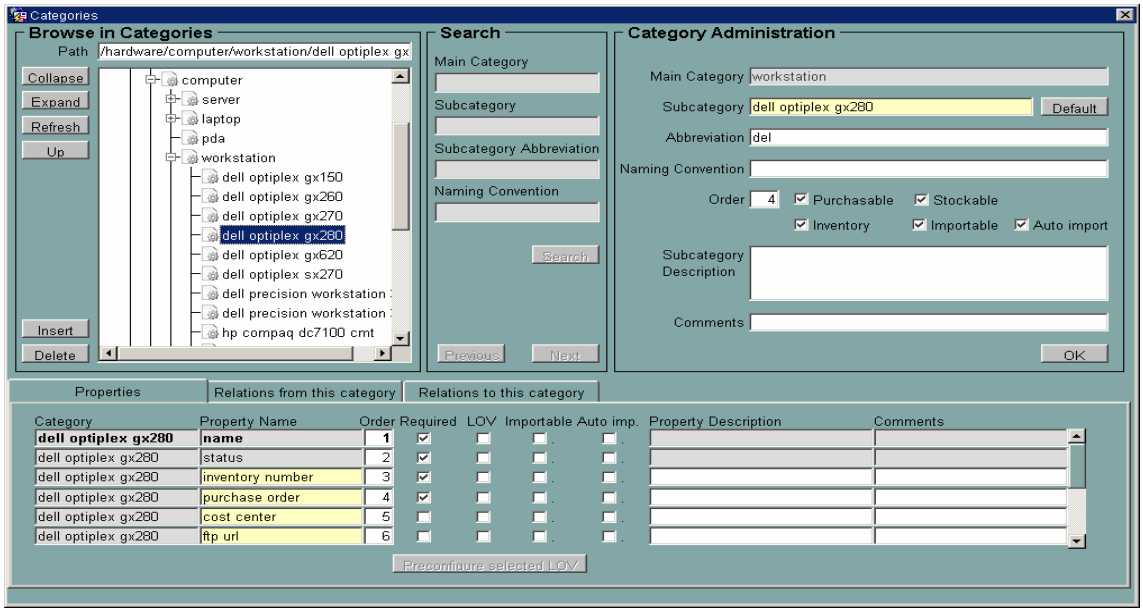
9. ábra: A nit_f_usr form képernyőképe

A rendszer lokációit a `nit_f_lcn` form adminisztrálja. A lokációk hierarchikus fája a képernyő bal oldalán elhelyezkedő Browse in locations területen látható. A form lehetővé teszi lokáció keresését a szülő neve, név, rövidítés és elnevezési konvenció alapján. A találatok között a Previous és Next gombokkal tallózhatunk. Az aktuális elemről név, rövidítés, elnevezési konvenció, leírás és megjegyzés kerül tárolásra.



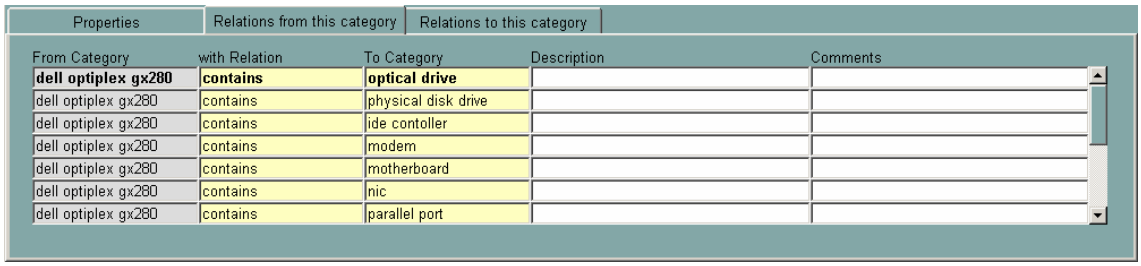
10. ábra: a nit_f_lcn form képernyőképe

A kategóriákat nyilvántartó `nit_f_cit` form modul több törzsadatot is felhasznál működése során. A kategóriák adminisztrációja a lokációval analóg módon történik. Újdonság az `Order` mező megjelenése, mely a megjelenítés sorrendjét befolyásolja a hierarchikus fában. A képernyő alján elhelyezett fülesvásznon (`tabbed canvas`) nyílik lehetőség az aktuális kategóriához tartozó tulajdonságok és típusrelációk megadására. A Purchasable és Inventory jelölőnégyzetek bejelölt értéke esetén automatikusan felvételre kerülnek a purchase order illetve inventory number tulajdonságok.



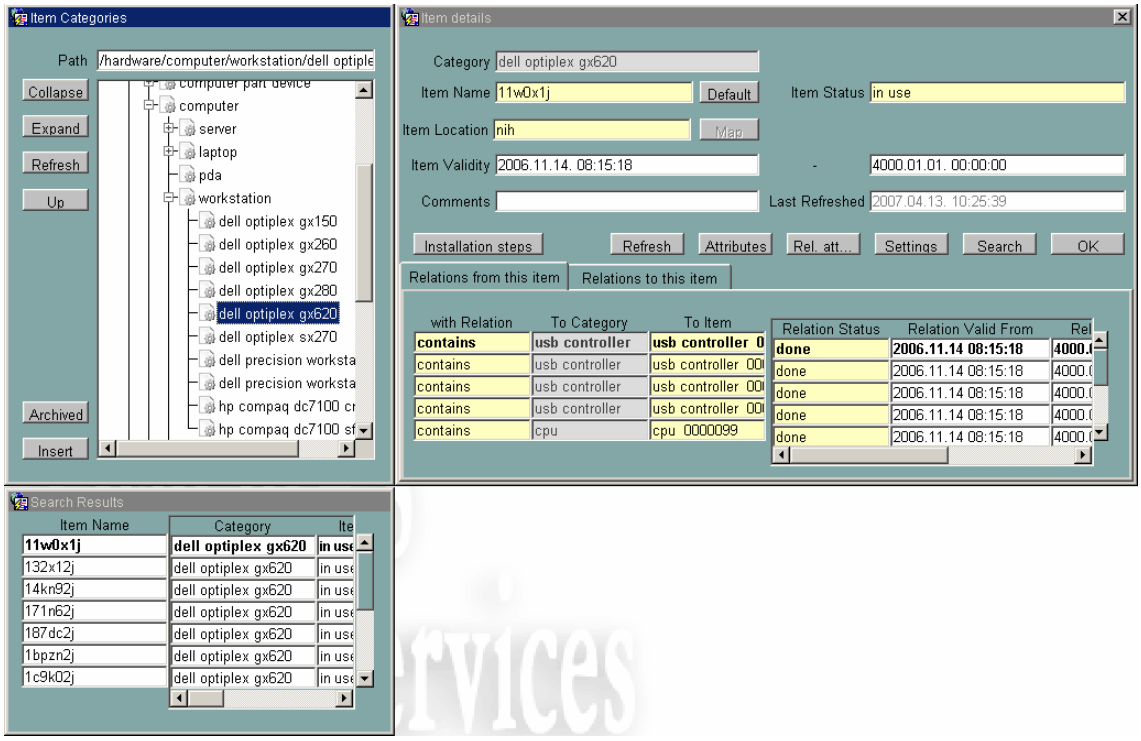
11. ábra: A nit_f_cit form képernyőképe

A kategória típusrelációit a Relations from this category, Relations to this category füleken találjuk.



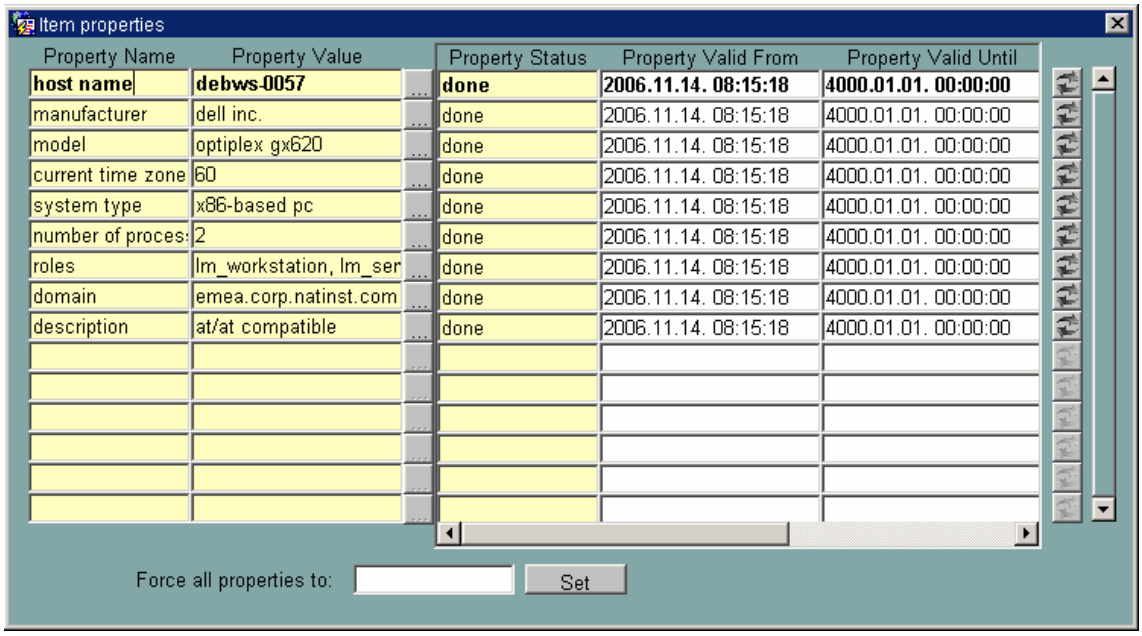
12. ábra: Relációkat kezelő fül a nit_f_cit formon

A törzsadatok megadását követően a nit_f_ci form modul teszi lehetővé a konfigurációs elemek adminisztrációját. A képernyő bal oldalán megjelenő kategóriák fájának tetszőleges levélelemét aktuálissá téve listázásra kerül a kategória összes konfigurációs eleme. A lista aktív eleméről a képernyő jobb oldalán megjelenő panelben kaphatunk részletes információt (elem neve, státusza, helye, érvényességi ideje, megjegyzés). A relációkkal kapcsolódó konfigurációs elemek között a Relations from this item, Relations to this item fülekre kattintva tallózhatunk.



13. ábra: A nit_f_ci form képernyőképe

Az Attributes és Related attribs gombokra kattintva az aktuális, illetve a relációval kapcsolódó elem attribútumait tekinthetjük meg (tulajdonság neve, aktuális értéke, státusza, érvényességi ideje, megjegyzés).



14. ábra: A konfigurációs elemek attribútumai

Beállítástól függően az ablakokban megjelenítésre kerülhetnek a már érvényességüket vesztett konfigurációs elemek és attribútumok, lehetővé téve az életút teljes nyomkövetését.

11. Hierarchikus fa kezelése Oracle Formsban

A törzsadatok felépítéséből adódóan a lokációk és típusok esetében célszerű a fa struktúrában történő megjelenítés. A felhasználó számára nagyobb áttekinthetőséget és könnyebb kezelhetőséget biztosít az egyes ágak legördítésének, elrejtésének a lehetősége. Emellett alapvető cél volt az interaktivitás biztosítása, mely a következőket takarja: Az egyes csomópontokra kattintva a képernyő egy, a fától független területén jelenjen meg az adott csomópont összes, felhasználó számára lényeges tulajdonsága (jelen esetben a lokáció megnevezése, rövidítése, esetleges megjegyzés), továbbá egy elérési út. A név változása esetén automatikusan frissüljön a csomópont neve a fában. Rekord törlésekor a csomópont, s az esetleges gyermekei kerüljenek eltávolításra. Új rekord felvitele közben a fában jelenjen meg egy, az új adatot jelképező csomópont.

A hierarhikus fa része az Oracle Forms beépített elemtípusainak. A formon való elhelyezés feltétele, hogy az elem különálló adatblokkba kerüljön, melyhez további elemek már nem kerülnek hozzáadásra. Futásidőben történik a hierarchikus fa inicializálása, mely magában foglalja az adatokkal való feltöltést, a Form Builder terminológiájában „populálást”. A hierarchikus fa alapja lehet lekérdezés, rekordcsoport, illetve lehetőség van a fa elemeinek manuális hozzáadására. A lekérdezés választása jelen esetben nem szerencsés, mivel a felhasználói interakció gyakran igényli a fa frissítését. Minden ilyen esetben egy hierarchikus lekérdezés fut le az alaptáblára vonatkozóan. A rekordcsoport esetén csak szélsőséges esetben vagy külön felhasználói kérésre indul el a lekérdezés.

A hierarchikus fa adatforrása rekordcsoport és lekérdezés esetén is a következő módon épül fel:

- `STATE`: A csúcs állapota. 1 esetén a csúcs gyermekei látszanak (`EXPANDED`), -1 esetén elrejti azokat (`COLLAPSED`). A 0 jelentése levélelem (`LEAF_NODE`).

- `LEVEL`: Az elem szintje a hierarchiában. A gyökér szintje 1.

- `LABEL`: A felhasználói felületen megjelenő név

- ICON_NAME: Ikonfájl megnevezése (NULL esetén a csúcshoz nem tartozik ikon)

- VALUE: A csúcs által reprezentált érték. A formokon az alaptábla elsődleges kulcs értéke került felhasználásra, a későbbiekben így egyszerűen hivatkozható a csúcshoz kötődő táblasor.

11.1. Nem-statikus rekordcsoportok kezelése

A Form Builder lehetővé teszi rekordcsoportok futási időben történő definiálását. A rekordcsoport létrehozását, manipulálását, törlését a fejlesztő az alábbi beépített alprogramok segítségével végezheti el:

- CREATE_GROUP: A beépített függvény létrehoz egy, a paraméterben megadott nevű üres rekordcsoportot. A scope paraméterben szabályozható, hogy lokális vagy globális rekordcsoportot hozunk létre, azaz a továbbiakban meghívott formok számára látható-e.

- CREATE_GROUP_FROM_QUERY: Visszatérési értéke egy rekordcsoport, melynek struktúráját a paraméterként megadott lekérdezés definiálja. A rekordcsoport tartalma a lekérdezés által behozott rekordok halmaza. Ajánlott az alias nevek használata, elhagyása esetén a Form Builder automatikusan generál nevet: ICRGGQ_<sorszám> alakban.

- DELETE_GROUP: Az eljárás törli a paraméterben megadott rekordcsoportot.

- ADD_GROUP_COLUMN: A függvény egy már létező rekordcsoporthoz ad oszlopot. Paraméterben adható meg az oszlop neve és típusa (CHAR, DATE, LONG, NUMBER). Karakter típusú oszlop esetén kötelező a hossz megadása.

- FIND_COLUMN: Visszatérési értéke a paraméterben megadott nevű oszlop (az oszlopnév a rekordcsoport nevével minősített)

- GET_GROUP_RECORD_NUMBER: Paraméterként egy rekordcsoport oszlopát és egy értéket vár. A függvény visszatér azon sor számával, ahol először fordul elő az adott érték. Ha az érték nem fordul elő az adott oszlopban, 0 a visszatérési érték.

- GET_GROUP_ROW_COUNT: Visszatér a paraméterben megadott rekordcsoport sorainak számával.

- ADD_GROUP_ROW: Új sor hozzáadása a paraméterben megadott rekordcsoporthoz a kívánt pozícióra. A rekordcsoport végére való beszúrást az END_OF_GROUP konstans átadásával lehet kérni.

- DELETE_GROUP_ROW: Az eljárás törli a paraméterként kapott rekordcsoport adott sorát.

- SET_GROUP_CHAR_CELL, SET_GROUP_DATE_CELL, SET_GROUP_NUMBER_CELL: Az eljárás a már létező (esetlegesen üres) sorok módosítására alkalmas. Paraméterként átadásra kerül az oszlopnév, a sor száma valamint az új érték.

- GET_GROUP_CHAR_CELL, GET_GROUP_DATE_CELL, GET_GROUP_NUMBER_CELL: A függvény a paraméterként kapott oszlopnév és sor száma alapján visszaadja a sor-oszlop metszetben lévő értéket.

11.2. Az FTREE csomag ismertetése

A Form Builder hierarchikus fa elemtípusa esetén - ellentétben egyéb alaptípusokkal - nem épül ki automatikusan a kapcsolat az adatbázisban tárolt rekordokkal. A fa rekordokból történő létrehozását explicit függvény- és eljáráshívásokkal teheti meg a fejlesztő. Az alprogramok a beépített FTREE csomagban kaptak helyet:

- SET_TREE_PROPERTY: Az eljárás segítségével beállítható a hierarchikus fa adatforrása. A felépítéshez szükséges adatokat rekordcsoportból szeretnénk kinyerni:

```
FTREE.SET_TREE_PROPERTY(<hierarhikus_fa>
                        ,FTREE.RECORD_GROUP
                        ,<rekordcsoport>);
```

- GET_TREE_PROPERTY: A függvény a paraméterként átadott hierarchikus fa adatforrásáról nyújt információt (rekordcsoport vagy lekérdezés). Lekérdezhető továbbá a rekordcsoport neve (RECORDGROUP), a lekérdezés szövege (QUERY_TEXT). Jól használható a kijelölt elemek számának meghatározására (SELECTION_COUNT).

- FIND_TREE_NODE: Visszatérési értéke a keresett csúcs. A keresés történhet név vagy reprezentált érték szerint (NODE_LABEL, NODE_VALUE). Kereshető az első előfordulás vagy az első gyermek csúcs (FIND_NEXT, FIND_NEXT_CHILD). Megadható továbbá a kiindulási csúcs.

- GET_TREE_NODE_PARENT: A paraméterként kapott csúcs szülőjével tér vissza.

- SET_TREE_SELECTION: Feladata a paraméterként kapott csúcs kijelölése vagy a kijelölés megszűntetése. Az alábbi példa egy érték alapján megkeres egy csúcsot a hierarchikus fában, s kijelöli azt:

```
v_selected_node := FTREE.FIND_TREE_NODE(
                                   <hierarchikus_fa>
                                   ,<csúcs_értéke>
                                   ,FTREE.FIND_NEXT
                                   ,FTREE.NODE_VALUE
                                   ,FTREE.ROOT_NODE
                                   ,FTREE.ROOT_NODE);
```

```
IF (NOT FTREE.ID_NULL(v_selected_node)) THEN
    -- csúcs kijelölése
    FTREE.SET_TREE_SELECTION(<hierarchikus fa>
                             ,v_selected_node
                             ,FTREE.SELECT_ON);
END IF;
```

- GET_TREE_SELECTION: A függvény visszatér a paraméterben kapott sorszámú kijelölt elemmel. A kijelölt elemek számozása 1-től kezdődik.

- SET_TREE_NODE_PROPERTY: Az eljárás segítségével frissíthető egy már létező csomópont paraméterként kapott tulajdonsága: a csomópont neve, értéke, ikonja valamint állapota.

- GET_TREE_NODE_PROPERTY: A SET eljárás párja. Tipikus használata a WHEN-TREE-NODE-SELECTED triggerben a kijelölt csúcs értékének meghatározása:

```
FTREE.GET_TREE_NODE_PROPERTY(
                    <hierarhikus_fa>
                    ,NAME_IN('SYSTEM.TRIGGER_NODE')
                    ,FTREE.NODE_VALUE);
```

- ADD_TREE_NODE: Csúcs manuális hozzáadása a fához. A form ritkán alkalmazza, általában az adatforrást képező rekordcsoportot módosítja. A paraméterek között szerepelnie kell a csúcs nevének, értékének, az ikonnévnek, egy már létező csúcsnak. A fa gyökere FTREE.ROOT_NODE segítségével hivatkozható s egyben ez a csúcs minden további őse.

- DELETE_TREE_NODE: Az eljárás törli a paraméterként kapott csúcsot az összes gyermekével együtt.

- POPULATE_TREE: Az eljárás törli a paraméterként kapott hierarchikus fa összes elemét, s a beállított adatforrásból ismételten felépíti azt.

11.3. A TREE_CONTROL csomag ismertetése

A formok által használt hierarchikus fa kezelésére született a tree_control csomag. Az újrafelhasználást szem előtt tartva önálló PL/SQL könyvtárban került elhelyezésre. A rekordcsoport mint adatforrás vonalat követve a csomag alprogramjai végzik a fa inicializálását, frissítését, új csomópontok hozzáadását vagy az elemtörlést.

- CREATE_TREE: Az eljárás inicializálja a paraméterként kapott hierarchikus fát. A p_need_order TRUE értéke esetén a fában az azonos szülőhöz tartozó csúcsok egy order numerikus oszlop értékei alapján kerülnek rendezésre. A fa adatforrását képező rekordcsoportot egy nézet alapján hozza létre, melynek neve bemenő paraméter. A lokációkat kezelő formok esetén a nézetben szereplő lekérdezés a következő:

```
SELECT
   (CASE
       WHEN LEVEL <= nit_core_p.getpi('tree depth')
          THEN 1
       ELSE -1
       END) AS NODE_STATE
  ,LEVEL AS "LEVEL"
  ,name AS LABEL
  ,NULL AS ICON_NAME
  ,TO_CHAR(id) AS VALUE
FROM nit_locations
CONNECT BY PRIOR id = lcn_id
START WITH lcn_id IS NULL
ORDER SIBLINGS BY name.
```

- `REFRESH_TREE`: A hierarchikus fa ismételt felépítését teszi lehetővé az eljárás. Paraméterei a fenti függvényhez hasonlóan a hierarchikus fa neve, `p_need_order`, továbbá `p_from_db`. Utóbbi értékétől függően lefuttatja a nézetben szereplő lekérdezést, majd a rekordcsoport aktuális tartalma alapján frissíti a fát.

- `INSERT_NODE`: Az eljárás beilleszt egy új csúcsot a hierarchikus fa adatforrását képező rekordcsoportba. A szülő csúcs érték alapján kerül azonosításra, mely a `p_parent_value` bemenő paraméterben adható meg. A további bemenő paraméterek a csúcs neve, értéke, ikon neve és egy opcionális sorszám.

- `UPDATE_NODE`: Paraméterezése az INSERT_NODE eljárással megegyező. Létező csúcs frissítésére alkalmazható. A csúcs szülője nem módosítható.

- `DELETE_NODE`: Az eljárás érték alapján azonosított csúcs első előfordulását törli a hierarchikus fa adatforrásából. A törlés során a csúcs gyermekei is eltávolításra kerülnek.

- `EXPAND_NODE`: A `p_node_value` paraméterként kapott érték alapján azonosított csúcsot bontja ki: az adatforrásként használt rekordcsoportban a megfelelő bejegyzés `NODE_STATE` értékét állítja 1-re.

- `COLLAPSE_NODE`: Az előző eljáráshoz hasonlóan a beazonosított csúcs gyermekeinek elrejtésére szolgál a felhasználói felületen. A `NODE_STATE` érték -1-re állítódik a rekordcsoportban.

- `GET_SELECTED_NODE_VALUE`: Bemenő paramétere a felhasználói felületen található hierarchikus fa neve. A függvény a fában aktuálisan kijelölt csúcs értékével tér vissza.

- `SET_NODE_SELECTED`: Az eljárás `a p_tree` paraméterben kapott hierarchikus fában jelöli ki a `p_node_value` értékkel rendelkező csúcsot.

- REMOVE_SELECTION: A paraméterként kapott hierarchikus fában minden kijelölést megszűntet.

- ONE_LEVEL_UP: Az eljárás bemenő paramétere a hierarchikus fa neve. Ez az aktuálisan kijelölt elem szülőjét teszi aktuálissá. Gyökérelem esetén nem történik változás.

- GET_NODE_PATH: Az aktuálisan kijelölt csúcs elérési útjával tér vissza. Az elérési út a hierarchikus fa gyökerétől értendő, az elválasztó karakter alapértelmezésben „/”. Az eljárás hívása nem eredményez adatbázis lekérdezést, tisztán a csúcs rekordcsoportban elfoglalt helye alapján dolgozik. Alternatívát jelenthet az Oracle 9i-ben megjelent SYS_CONNECT_BY_PATH függvénnyel szemben, amely hierarchikus lekérdezések esetén megadja a szelektált sor elérési útját a gyökérelemhez képest.

11.4. Felhasználói interakciók a hierarchikus fában

A felhasználói interakció tipikus esete a csomópont kijelölése. Az esemény lekezelését a beépített WHEN-TREE-NODE-SELECTED trigger végzi. A Form Builder SYSTEM.TRIGGER_NODE változójából nyerhető ki az eseményt kiváltó csomópont. A GET_TREE_NODE_PROPERTY függvény használható a csomópont tulajdonságainak lekérdezéséhez. A form reakciója részletes információk megjelenése az aktuális csomópontról:

```
GO_BLOCK(pl_details_block); -- aktuális blokk váltása
IF (NOT FORM_SUCCESS) THEN
    RAISE FORM_TRIGGER_FAILURE;
END IF;
v_message_level := NAME_IN('SYSTEM.MESSAGE_LEVEL');
COPY(25,'SYSTEM.MESSAGE_LEVEL');
EXECUTE_QUERY; -- lekérdezés indítása
COPY(v_message_level,'SYSTEM.MESSAGE_LEVEL');
```

```
IF (NOT FORM_SUCCESS) THEN
    CLEAR_BLOCK;
    tree_is_invalid;
END IF;
```

A példában szereplő pl_details_block csomagváltozó egy, a fához szorosan kötődő adatblokk nevét hordozza. Az adatblokk alapját az alaptáblára épülő nézet képezi, a kettő közötti kapcsolatot pedig az elsődleges kulcs biztosítja (mely a fában a csúcsokhoz rendelt érték). A cél minden esetben a hierarchikus fa és az adatblokk szinkronizálása, amennyiben bármelyikben változás történik.

A formon való navigáció procedurálisan a GO_BLOCK, GO_ITEM beépített függvények segítségével végezhető el. A kért művelet azonban nem minden esetben hajtható végre sikerrel (pl. a navigációt megelőzően az aktuális elem validálása sikertelen), melyről a felhasználó egy hibaüzenetből értesül. A kódban a FORM_SUCCESS beépített logikai változó értéke alapján tájékozódhatunk a művelet kimeneteléről. A RAISE FORM_TRIGGER_FAILURE hatására a trigger működése felfüggeszthető, a triggertörzs további kódja már nem kerül végrehajtásra.

Az aktuális adatblokkban a lekérdezés az EXECUTE_QUERY eljárással indítható. Sikeres végrehajtás esetén az állapotsorban megjelenik a lekérdezés által behozott rekordok száma. Az információ zavaró lehet a felhasználók számára, ezt elkerülendő módosítjuk a SYSTEM.MESSAGE_LEVEL beépített változó értékét. Az alapértelmezett 0 érték minden üzenetet megjelenít, a lekérdezéskor alkalmazott 25 érték azonban csak a legszükségesebb üzenetekre korlátozódik.

Az EXECUTE_QUERY hatására aktivizálódik az aktuális adatblokk PRE-QUERY triggere. Tipikusan a SET_BLOCK_PROPERTY eljárás kerül meghívásra, mely beállítja az adatblokkhoz tartozó lekérdezés WHERE feltételét (a DEFAULT_WHERE paraméter átadásával). Ezzel biztosítjuk, hogy mindig az aktuálisan kijelölt faelemre vonatkozóan jelenjenek meg a részletes adatok (WHERE ID=<aktuális csomópont értéke>).

A fa valamely csúcsára duplán kattintva állapotától függően megjeleníthetőek vagy elrejthetőek a gyermek csúcsok. Az eseményre a WHEN-TREE-NODE-EXPANDED trigger segítségével lehet reagálni. A trigger jelen esetben a tree_control csomag EXPAND_NODE, COLLAPSE_NODE eljárását hívja a csúcs állapotának függvényében. Így egy esetleges frissítés esetén megelőzhető a fa összes csúcsának kibontása.

11.5. DML műveletek és a hierarchikus fa kapcsolata

A hierarchikus fát tartalmazó formok esetén a fához szorosan kötődő pl_details_block esetén letiltjuk az új rekord beszúrásának a lehetőségét. A létrehozásra a hierarchikus fa adatblokkjának KEY-CREREC triggerét használjuk. A trigger minden olyan esetben aktivizálódik, amikor a felhasználó új rekord létrehozását kéri menüből vagy billentyűkombináció lenyomásával. A pl_details_block KEY-CREREC triggerébe a következő kód kerül:

```
GO_BLOCK(pl_tree_block);
IF (NOT FORM_SUCCESS) THEN
    RAISE FORM_TRIGGER_FAILURE;
END IF;
DO_KEY('CREATE_RECORD');
```

A DO_KEY utasítás paraméterként egy beépített alprogram nevét várja, s aktivizálja az ahhoz tartozó KEY triggert. Jelen esetben a CREATE_RECORD eljáráshoz tartozó KEY-CREREC triggert (a további esetek leírása megtalálható a Form Builder referenciájában). Így biztosított, hogy az új rekord létrehozása csak egy belépési ponton keresztül történhet.

A hierarchikus fa KEY-CREREC triggere működés során törli a pl_details_block tartalmát a CLEAR_BLOCK utasítással. Sikeres végrehajtás esetén aktivizálódik az érintett adatblokk WHEN-CREATE-RECORD triggere, mely automatikusan kitölti az új rekord szülő bejegyzését (a szülő csúcs neve a fában). A blokk tartalmának törlése után beszúrunk egy új

csúcsot a fába az aktuálisan kijelölt elem gyermekeként, a csúcs neve ideiglenesen „…” lesz. Ez a szinkronizáció egyik oldala.

Miután megtörtént az új elem beszúrása, azaz lefutott a `pl_details_block` `ON-INSERT` triggere, aktivizálódik a `POST-INSERT`. A trigger minden új rekord beszúrását követően aktívvá válik, így alkalmat teremt a korábban beszúrt ideiglenes csúcs eltávolítására a hierarchikus fából, s új csúcs beszúrására a már aktualizált adatokkal (név, érték). A helyzet hasonló a már létező adatok módosítása esetén. A `POST-UPDATE` trigger aktualizálja a `tree_control` csomag segítségével a már létező csúcs nevét, értékét. Ha a tranzakció mégis visszagörgetődne, az `ON-ROLLBACK` triggerben frissítjük a fát, azonban itt már lekérdezés futtatásával biztosítjuk a konzisztenciát a felület és az adatbázis között. Ez a szinkronizáció másik oldala.

12. Dinamikus riportolási lehetőség használata

A NITIL rendszert a konfigurációs elemek nyilvántartásán túl jelentések generálására is fel kellett készíteni (pl. leltár készítése a mindenkori gépparkról, bizonyos feltételeknek eleget tevő elemek listázása felújítás vagy csere céljából, stb.). A szükséges riportok vagy jelentések felépítése és száma azonban még a rendszer fejlesztésének késői stádiumában sem volt ismert. Nem jelentett lehetséges alternatívát az egy nyomógomb – egy riport módszer, hiszen minden további jelentés integrálása fejlesztői beavatkozást igényel a felhasználói felületen.

A dinamizmus első lépése a rendelkezésre álló riportok adatainak adatbázisban történő tárolása. Az újabb táblák létrehozását elkerülve és az adatmodell dinamizmusát felhasználva a riport mint kategória jelenik meg a rendszerben, `nitil reports` néven. Új tulajdonságtípusok kerülnek felvételre, nevük prefixe mindig `report`. Minden riporthoz hozzárendelődik két alaptulajdonság: `report output type` és `report output format`. Előbbi a riport célját határozza meg (destype), utóbbi a kimenet formátumát (desformat). További tulajdonságtípusok és tulajdonságok hozzáadása a már elkészített felületen keresztül egyszerűen megoldható. Az egyes konkrét riportok konfigurációs elemként jelennek meg a rendszerben, bemenő paramétereik pedig ezen elemek attribútumai lesznek.

A riportolási funkciót az Oracle Reports biztosítja. A riport létrehozását követően a létrejött rdf állományt az Oracle Application Serveren kell elhelyezni. A riport hívása a szerver `rwservlet` java servlet-jén keresztül történik egy `http GET` kérés formájában. A kérés során átadott fontosabb paraméterek:

- `destype`: a céleszköz típusa (mely számára a riport elküldésre kerül). Lehetséges értékei: `CACHE`, `LOCALFILE`, `FILE`, `PRINTER`, `MAIL`, `ORACLEPORTAL`, `FTP`, `WEBDAV`.

  `LOCALFILE` esetén az igénylő számítógépen, `FILE` esetén a szerveren kerül letárolásra a riport kimenete. A gyorsítótáron kívül minden esetben megadandó a `desname` paraméter is, mely a célt konkretizálja (file megadása esetén állománynév elérési úttal, mail esetén e-mail cím).

- desformat: a riportkimenet formátuma. Lehetséges értékei: DELIMITED, DELIMITEDDATA, HTML, HTMLCSS, PDF, POSTSCRIPT, PRINTER DEFINITION, RTF, XML.
  DELIMITED megadása esetén csv állomány jön létre, a delimiter paraméter segítségével lehet specifikálni az elválasztó karaktert. A DELIMITEDDATA használata különösen nagy riportoknál célszerű, amennyiben a DELIMITED hibásan működik. Az RTF hatására rich text document formátumú állomány generálódik, szövegszerkesztővel történő további feldolgozásra.

- jobname: A riport kötegelt futtatása esetén a munkafolyamat neve a Report Queue-ban. A riport állapotát és eredményét a továbbiakban a jobname alapján lehet visszakeresni.

- module, report: A paraméter értéke a futtatandó állomány neve és kiterjesztése. (általában rdf állomány)

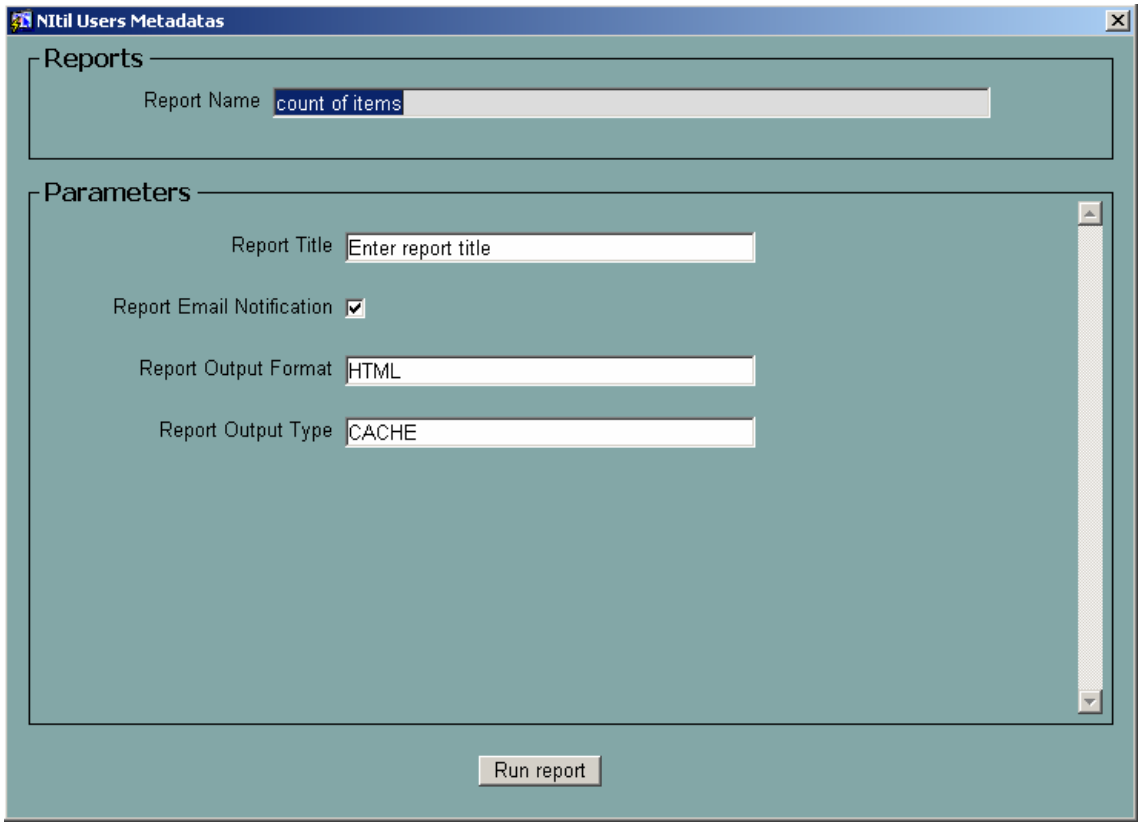
- notifyfailure, notifysuccess: A riport sikeres vagy sikertelen lefutása esetén a szerver értesítést küld a paraméterben megadott e-mail címre.

- paramform: Érvényes értékei YES, NO, HTML. YES és HTML hatására a riport futtatásakor megjeleníti a paramétereket bekérő formot.

- server: A riport futtatásához használt szerver neve.

- userid: A riport futtatásához szükséges név/jelszó@adatbáziskapcsolat.

A riportokat hívó form egyszerű felépítést követ: LOV segítségével listázhatóak a rendszerben elérhető riportok, s tetszőleges elemet kiválasztva megjelennek az adott riport paraméterei. A Run report gombra kattintva a riport futtatásra kerül az Application Serveren.



15. ábra: A nit_f_rep form képernyőképe

A modul `rep_main` adatblokkja két elemet tartalmaz: egy nem megjelenített `report_id` elemet, mely a report mint konfigurációs elem elsődleges kulcs értékét hordozza, továbbá a `report_name` elemet, melyhez a LOV kapcsolódik. A LOV hívásakor az elem `KEY-LISTVAL` triggere aktivizálódik, mely a lista megjelenítését követően a kiválasztott elem függvényében megjeleníti a paramétermezőket.

A form `rep_params` adatblokkja tartalmazza a paramétermezőket. Szöveg, szám, dátumidő és jelölőnégyzet típusú elemekből tartalmaz típusonként tíz darabot, elnevezésük `CHAR_ITEM_xx`, `NUMBER_ITEM_xx`, `DATETIME_ITEM_xx`, `CHECKBOX_ITEM_xx`, 1 és 10 között számozva. A mezők alaphelyzetben nem megjelenítettek és nem engedélyezettek.

A paraméterek későbbi átadása miatt rögzíteni kell, hogy mely felületi elem mely átadott paraméter értékét hordozza, tartozik-e az elemhez értéklista, és mi az értéklistát előállító lekérdezés. Ennek tárolása egy index alapú táblában történik, melynek definíciója a következő:

```
TYPE param_tab_rec IS RECORD(item_label VARCHAR2(100)
                            ,item_name VARCHAR2(30)
                            ,parameter_name VARCHAR2(100)
                            ,lov_select VARCHAR2(1000)
                            );
TYPE param_tab_type IS TABLE OF param_tab_rec
INDEX BY BINARY_INTEGER;
```

Minden paramétermező egy egyértelmű indexet kap a táblában a `get_hash` függvény segítségével, mely a paramétermező neve alapján 1 és 40 közé eső számot ad vissza.

A megjelenítés menete a következő: leválogatásra kerülnek azok az attribútumok, amelyek a kiválasztott riporthoz mint konfigurációs elemhez tartoznak. Az attribútumhoz kapcsolt tulajdonságtípus `type_name` mezője alapján meghatározzuk a megjelenítendő paraméter típusát (szöveg, szám, dátumidő, jelölőnégyzet). Megjelenítésre és engedélyezésre kerül az elem, s az indexalapú táblában egy bejegyzés készül a `get_hash` függvény által visszaadott indexű helyen. A táblasor a paramétermező címkéjét (pl. `Report Output Format`), nevét (pl. `CHAR_ITEM_xx`), a paraméter nevét (pl. `desformat`) és a kapcsolódó LOV-ot felépítő lekérdezést tartalmazza. A címke megegyezik az attribútumhoz kapcsolt tulajdonságtípus nevével, nagybetűvel kezdve annak minden szavát (`INITCAP` függvényt használja). A paramétermezőhöz kapcsolódó értéklista lekérdezése az alábbi konvenció szerint kerül meghatározásra: Az attribútumhoz kapcsolt tulajdonság `description` mezője hordozhat egyetlen, `VALUE` alias oszloppal visszatérő lekérdezést. Ennek hiányában az adott tulajdonsághoz tartozó értékhalmaz elemeiből (`property value`) épül fel az értéklista, nem halmazos tulajdonság esetén a paramétermezőhöz nem rendelünk LOV-ot.

A paramétermezőkhöz egyetlen közös, `PARAMETER_LOV` nevű értéklista kerül hozzárendelésre. Az értéklista alapja a `PARAMETER_GROUP` nevű rekordcsoport. Tetszőleges mezőre navigálva aktivizálódik a blokkszintű `WHEN-NEW-ITEM-INSTANCE` trigger. A törzsben meghatározásra kerül a triggert aktiviző elem (értékét a

SYSTEM.TRIGGER_ITEM rendszerváltozó hordozza), s a paramétermező neve alapján a get_hash függvény segítségével megkeressük a mezőhöz tartozó sort az index alapú táblából. A táblasor lov_selection mezője hordozza azt a lekérdezést, mely alapján frissítjük a rekordcsoportot (POPULATE_GROUP_WITH_QUERY függvény segítségével). Az értéklista által visszaadott érték a rep_ctrl adatblokk lov_selection mezőjébe kerül, s szükség esetén innen másolódik a paramétermezőbe.

A riportok futtatásához a modul egyetlen report nevű riport objektumot tartalmaz. A riport objektum paraméterei futásidőben változtathatóak, nem kötődik statikusan egyetlen rdf állományhoz sem. A form által megjelenített paramétermező értékeket a get_parameters függvény gyűjti össze és hozza a riport objektum számára értelmezhető alakra:

```
FUNCTION get_parameters RETURN VARCHAR2 IS
     v_parameters VARCHAR2(10000) := 'paramform=no';
     i            PLS_INTEGER := pl_param_tab.FIRST;
BEGIN
     LOOP
          EXIT WHEN i IS NULL;
          v_parameters := v_parameters || '&' ||
                 pl_param_tab(i).parameter_name ||
                 '=' || NAME_IN('REP_PARAMS.' ||
                 pl_param_tab(i).item_name);
          i := pl_param_tab.NEXT(i);
     END LOOP;

     RETURN v_parameters;
END;
```

A paraméterek a SET_REPORT_OBJECT_PROPERTY eljárás REPORT_OTHER paraméterezésével adhatóak át. A riport objektum futtatása a RUN_REPORT_OBJECT függvény segítségével kérhető, mely visszatérési értékként egy munkamenet azonosítót ad. Ezt követően egy időzítő kerül létrehozásra, 5 másodperces ciklusidővel:

```
pl_reportserver_job := RUN_REPORT_OBJECT(v_report_obj);

IF (pl_reportserver_job IS NOT NULL) THEN
    v_timer := CREATE_TIMER('REP_TIMER',5000,REPEAT);
END IF;
```

Valahányszor letelik az időzítő létrehozásakor megadott időintervallum, aktivizálódik a form WHEN-TIMER-EXPIRED triggere. A REPORT_OBJECT_STATUS függvény segítségével lehet tájékozódni a paraméterként megadott munkafolyamatról. FINISHED állapot esetén a riport eredménye megjelenik egy új böngészőablakban, a sikertelen végrehajtásról pedig üzenetben értesül a felhasználó:

```
v_status := REPORT_OBJECT_STATUS(pl_reportserver_job);
IF (v_status = 'FINISHED') THEN
    DELETE_TIMER('REP_TIMER');
    WEB.SHOW_DOCUMENT ('/reports/rwservlet/getjobid' ||
                       l_reportserver_job, '_BLANK');
ELSIF (v_status NOT IN
      ('RUNNING','OPENING_REPORT','ENQUEUED')) THEN
    DELETE_TIMER('REP_TIMER');
    msg_popup('Report aborted','W',FALSE);
END IF;
```

13. Összefoglalás

Diplomamunkámban egy ITIL alapokon nyugvó nyilvántartórendszer architektúráját és implementálásának bizonyos részeit mutattam be. A rendszer tervezésének részletes ismertetése meghaladja jelen diplomamunka kereteit. A fogalomrendszerrel, táblákkal és üzleti logikával kapcsolatban bővebb leírás olvasható Hodosi Szabolcs Alkalmazásfejlesztés Oracle környezetben (NITIL – ITIL szabvány alapú konfigurációmenedzsment rendszer tervezése) című szakdolgozatában. Az implementáció esetén a bővebb háttértudást igénylő, komplex eseményvezérlésű form modulok esettanulmányszerű bemutatására koncentráltam. További fejezetek témája lehetne a felhasználói felület bővítése Java osztályokkal és azok integrálása Oracle Forms környezetbe.

A megvalósított szoftver az IT tárgyi eszközök egyszerű nyilvántartásán túl – a flexibilis adatmodellnek köszönhetően – alkalmas szolgáltatások, licenszadatok, üzembehelyezési lépések tárolására. Jövőbeli fejlesztés tárgyát képezheti az ITIL változásmenedzsment koncepciójának megvalósítása, amelyhez a meglévő adatmodell és felhasználói felület jó alapokat szolgáltat.

14. Felhasznált irodalom

- Juhász István - Gábor András – PL/SQL programozás Alkalmazásfejlesztés Oracle9i-ben
  Panem, 2002
- Oracle Application Server Reports Services Publishing Reports to the Web 10g (9.0.4)
  Part Number B10314-01
  www.download-uk.oracle.com/docs/html/B10314_01/toc.htm
- Oracle Forms Developer: Form Builder Reference, Release 6i
  Part Number A73074-01
- Oracle Forms Developer 10g: Build Internet Applications
  Student Guide, Volume 1-2
  D17251GC10, Edition 1.0, June 2004, D39558
- Oracle Reports Developer 10g: Build Reports
  Student Guide Volume 1-2
  D17251GC10, Production 1.0, June 2004, D39520
- Frank Nimphius – Oracle Application Server 10g - Integrating Oracle Reports in Oracle Forms Services Application
  (An Oracle Whitepaper, May 2004)
- Frank Nimphius – Oracle Forms Services - Using Run_Report_Object() to call Reports with a parameter form
  (An Oracle Technical Whitepaper, February 2004)
- Frank Nimphius – Oracle9iAS Forms Services: HowTo Deploy Icons
  (An Oracle Whitepaper, June 2003)
- itSMF Magyarország – Service Management Fórum
  http://www.itsmf.hu/portal/index.php