GNU Bayonne, a kapcsolatteremtő

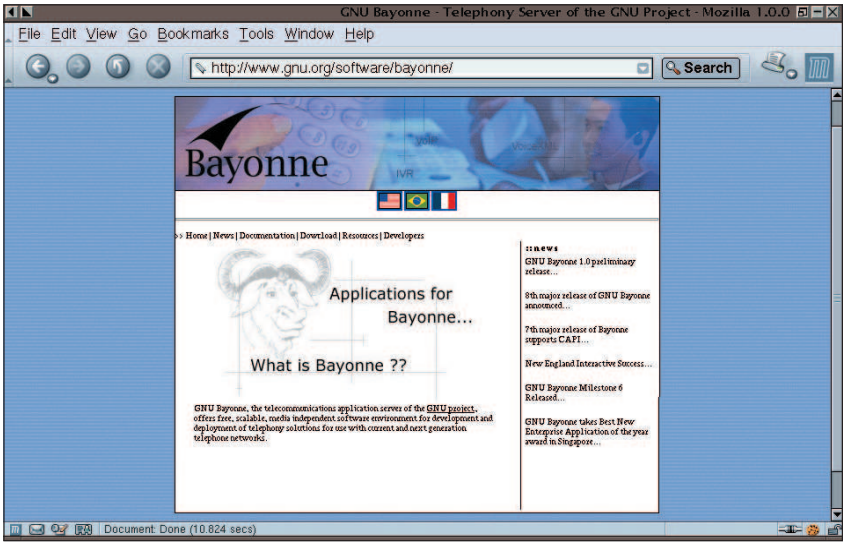
A Bayonne Projekt bármely gyártó kártyájával képes telefonos kapcsolatot létrehozni.

Három évvel ezelőtt rá kellett jönnöm, hogy komoly hiány tapasztalható a szabad programok terén. Igaz ugyan, hogy a szabad programok elterjedtek, és a vállalkozói területen szinte minden rést be is töltöttek, de egyikük sem próbálta megoldani a távközlésben felmerült gondokat.
Pedig a kapcsolattartás nemcsak hogy minden üzletnek része, de egyben (gyakorta észre sem vett) része a felhasználói élménynek is. Ugyanakkor a nyilvános telefonos hálózaton keresztül működő telefonos szolgáltatások létrehozásához szükséges alkatrészek egyre szélesebb körben terjednek el a piaci PC gépek és operációs rendszerek körében, ideértve természetesen a GNU/Linuxot is.
Hogy szabad programmal célozhassuk meg ezt a piacrészt, néhányan úgy döntöttünk, egy keretrendszert hozunk létre, amelyben leírjuk, milyennek kellene lenniük ezeknek a szolgáltatásoknak; az asztali felhasználók és alkalmazásfejlesztők igényeitől kezdve egészen a legnagyobb távközlési cégek által támasztott elvárásaikig. A projekt később GNUCOMM néven vált ismertté, amikor hivatalosan is a GNU projekt munkacsoportjává vált.
Az egyik olyan terület, amit meg szerettünk volna határozni, a telefonos alkalmazáskiszolgáló volt. Egy ilyen kiszolgáló a könnyen telepíthető új telefonos alkalmazásszolgáltatások egyszerű létrehozását tenné lehetővé. Ezek olyan alkalmazások lennének, amelyeket kifejezetten arra fejlesztenek ki, hogy a kiszolgálót a hagyományos telefonvonalon, elérő hús-vér emberekkel tartsák a kapcsolatot, akik az alkalmazással hang, illetve a telefon nyomógombjainak segítségével érintkezhetnek.
Jellegzetesen ilyen alkalmazások a hangpostafiók-rendszerek vagy az előre fizetett (prepaid) hívási megoldások.
Minden ilyen rendszer meglehetősen összetett, valamint a PC-rendszer és a nyilvános telefonhálózat közötti kapcsolat megteremtése néha programozott rendszereket vagy különleges számítógépes telefonalkatrészeket igényel.
Ez lehet olyan eszköz amely analóg telefonvonalon képes beszélni a hívóval, de akár olyan berendezés is, amely többkapus hangvezérlést nyújt ISDN vagy T1 digitális hangáramkörön keresztül, amit a nagyobb vállalatok közvetlenül a helyi távközlési szolgáltató központjából kaphatnak meg.
Figyelembe véve, hogy az ilyen rendszerek a múltban általában igen költségesnek számítottak, soha nem voltak nyíltak, és gyakran nehezen programozhatónak bizonyultak, úgy gondoltam, ezeket a gondokat egyszerre fogom megoldani a jelenleg legtámogatottabb nyílt programrendszer, a GNU/Linux alá írt kiszolgálóprogram létrehozásával.
Amikor nekikezdtünk a projektnek, még igen kevés cég kínált GNU/Linux alatt is használható alkatrészeket, így azt használtuk, ami elérhető volt. Még manapság is minden telefonos kártya teljesen különböző, és többnyire saját API-t mellékelnek hozzá. Mivel sem az eszközök, sem az API-k nem szabványosítottak, a legtöbb ember, aki telefonos alkalmazásokat készít, csak egyetlen gyártó kártyacsoportjához fejleszt, és ezt kizárólag a gyártó által nyújtott API-n keresztül teszi. Ez a gyakorlat egyben azt jelenti, hogy a számítógépes távközlési üzletben minden gyártónak igen széles termékcsalád-kínálatot kell készítenie, mivel a felkínált termékskála réseit nem egykönnyen lehet más termékekkel helyettesíteni. Ezek a tényezők igencsak megnehezítik, hogy új telefonos kártyagyártó lépjen a színre, ugyanakkor megkönnyíti a korlátozott számban jelenlévő gyártók dolgát, hogy komolyabb változtatás nélkül megtarthassák piacukat.
Természetesen nem állíthatjuk, hogy egyáltalán nem történtek próbálkozások szabványosított API kialakítására. Végül is ott van a ECTF (European Community Telework/Telematics Forum). Lévén üzleti gyártók ipari konzorciuma, már elő kellett volna állniuk egy, bizottságokon keresztül vizsgált összetett szabványkészlettel, és olyan javaslatokkal, amelyek azt határoznák meg, hogyan fejlesszék és támogassák a gyártók a számítógépes telefonos megoldásokat.
Továbbá mindezt olyan módon kellene megtenniük, amely növeli a különleges tudás iránti igényt, emelve ezzel a számítógépi telefonos piacon jelenlevő tagjaikra nehezedő nyomás mértékét.
A másik népszerű szervezet az ITU (International Telecommunications Union) névre hallgat, amely leginkább arról ismert, hogy kinevezéseit gyakran nemzeti kormányzatok adják. Az USA-ban például a kinevezéseket az állami részleg adja, politikai alapokon, ahelyett, hogy a legjobb és legragyogóbb elmék közt válogatnának.
A mi célunk nemcsak az volt, hogy szabad programként telefonos kiszolgálót hozzunk létre, de egyúttal a telefonos alkalmazásszolgáltatásokat is olyan könnyen és egyszerűen kezelhetővé szerettük volna tenni, amilyen egy weblap készítése és karbantartása.
Ugyanakkor a telefonos meghajtókat és API-kat egy olyan szintig akartuk elvonatkoztatni, hogy ezek az alkalmazásszolgáltatások fejlesztésénél már egyáltalán ne számítsanak, és láthatatlanok legyenek. Ha ez sikerülne, az azt jelentené, hogy bárki tetszés szerint lecserélheti az eszközeit, ahelyett, hogy egyetlen gyártó által nyújtott termékekhez lenne kötve.

Kezdetben volt az ACS

Mivel a kiszolgálóban mindent alacsony szinten akartunk elvonatkoztatni, az első szükséges dolog egy C++ nyelven íródott osztálykészlet volt. Több okból is a C++-t szerettem volna használni.
Először is, a meghajtó csatolófelületek létrehozásához az osztálybeágyazások felhasználása tűnt természetesnek, figyelembe véve elvont természetüket.
Másodszor, úgy láttam, sokkal gyorsabban tudok hibamentes C++-kódot írni, mint hibamentes C-kódot. Tulajdonképpen ez lett az első nagyméretű C++-projektem.
Ugyancsak könnyű megmagyarázni, miért nem a már meglévő keretrendszereket használtuk. Tudtuk, hogy szükségünk lesz szálasításra, foglalatok (socket) támogatására és néhány egyéb elemre. Egyetlenegy létező keretrendszer sem tudta az összes szükséges dolgot biztosítani, kivéve néhányat, amelyek azonban jóval nagyobbak és összetettebbek voltak, mint amire

nekünk szükségünk volt. Például szerettünk volna egy telefonoskiszolgálóvázat. Ez idő tájt a legjobban hasznosítható keretrendszer a ACE (Adaptive Communication Environment) volt, amely általában néhány MB-tal növelte meg a futtatható könyvtár méretét. Mivel célunk az volt, hogy akár 8–12 MB memóriával rendelkező gépeken is futtatni tudjuk a programot, ez elfogadhatatlan hátránynak bizonyult.



A GNU Common C++ (eredetileg APE) azért jött létre, hogy egyszerűen értelmezhető és hordozható osztálymeghatározásokat nyújtson a szálak, a foglalatok (socket), a jelzők (semaphore), a kivételek és egyéb elemek részére. Az APE azóta megnőtt: mostanra már számos fejlesztés forrásává vált azon felül, hogy a GNU része.

Maguknak a szolgáltatásoknak a készítése közben ráébredtünk, hogy új módszerre van szükségünk a telefonos alkalmazások létrehozásához – mégpedig olyanra, amely a folyamatot egy átlagos rendszergazda számára is érthetővé teszi. Az egyszerűség kedvéért egy általános parancsnyelv használata mellett döntöttünk, amely később GNU ccScript néven vált ismertté. Parancsfájlok írásával és hangminták rögzítésével gyakorlatilag bárki anélkül hozzájárulhat a telefonos alkalmazásszolgáltatások létrehozásához, hogy különleges ismeretekre vagy átfogó rálátásra lenne szüksége; vagy olyan hihetetlenül összetett API-kra, mint amilyeneket a ECTF támogat. Mivel az alul elhelyezkedő telefonos vas láthatatlan, és csak elvont formában érhető el az alkalmazás parancsfájlnyelvéből, a kártyacsaládtól való függőség hátránya is megszűnt.

De milyen formát ölthet egy ilyen parancsnyelv? Sok parancsnyelv egyes értelmező példányhoz elkülönített végrehajtást tételez fel (szálanként vagy folyamatonként) minden ami sajnos használhatatlanná teszi őket a céljainkra. Sok nyelv feltételezi, hogy a kifejezés értelmezésének ideje nem határozható meg. Egy kifejezés ugyanis például önhívó (rekurzív) függvényeket vagy akár egész alprogramokat meghívhat.

Mi azonban – ahogy már korábban említettük – nem szerettünk volna minden egyes értelmező példányhoz egy-egy külön végrehajtási folyamatot rendelni, és azt sem akartuk, hogy az összes példány válaszoljon a telefonos vezérlő legelső eseményhívására, amint az állapota megváltozik, így aztán a már létező, általánosan használt megoldások (Tcl, Perl, Guile stb.,) egyike sem jöhetett közvetlenül szóba. Ehelyett első kiszolgálónkhoz inkább egy új, nem blokkosított, determinisztikus parancsnyelvet készítettünk.

Parancsnyelvünk több szempontból is egyedülállónak volt nevezhető. Először is lépésenként végrehajtott volt és nem blokkosított. Az utasítások vagy azonnal végrehajtódtak és visszatértek, vagy a végrehajtó segítségével végrehajtásukat egy későbbi időpontra időzítették. Így lehetővé vált, hogy egyetlen szál több értelmezőpéldányt hívjon meg és kezeljen. A telefonos kiszolgáló akár egy idejű hívások százait kell kezelje egy nagy terheltségű távközlési eszközön. Mivel nekünk nincs szükségünk a kiszolgálón futó hagyományos szálak százaira, szerény CPU-terhelést kapunk. A másik dolog, amiben parancsnyelvünk egyedülálló, a memóriába töltött parancsfájlok kezelése. Hogy a parancsfájlok töltése során fellépő késlekedést vagy akadályoztatást megelőzzük, a memóriában az összes parancsfájl egy virtuális gép (Virtual Machine azaz VM) memóriaszerkezetébe töltődik be, és itt értelmeződik. Ha meg akarjuk változtatni a parancsfájlt, egy teljesen új VM-példány jön létre, amely az új parancsfájlt tartalmazza. A jelenleg futó hívások a régi VM-ben futnak tovább, az új hívók számára azonban már az új VM lesz elérhető. Amikor az utolsó régi hívás is lezárult, a teljes régi VM törlődik. Így még akkor is elérhetjük a százszázalékos rendelkezésre állást, ha a szolgáltatások időközben módosulnak.

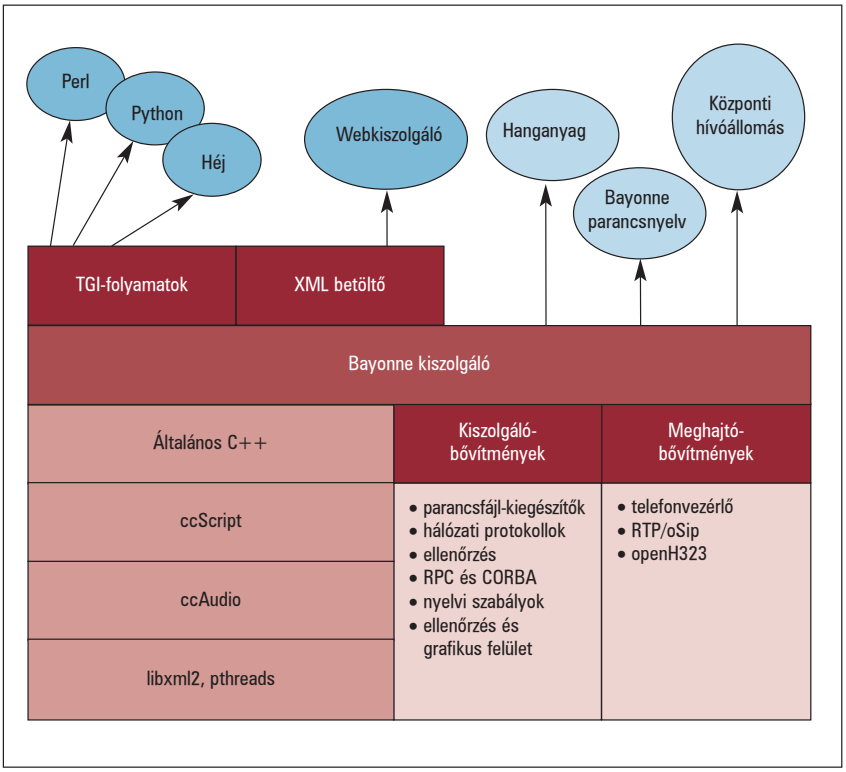
Végül, mivel C++ parancsnyelv-rendszert készítettünk, a parancsértelmező közvetlen osztálykiegészítését is lehetővé tettük, hogy a későbbiekben új parancsképességekkel lehessen bővíteni. Így bárki készíthet a nyelvből egy az adott alkalmazáshoz, vagy (ha szükséges) az adott telefonos vezérlőhöz levezetett nyelvváltozatot, egyszerűen csak a hagyományos C++-osztálykiterjesztést kell felhasználnia. A kiegészítés az eredeti nyelvből levezethető.

Bár a kiszolgáló parancsnyelv teljes telefonos szolgáltatások létrehozását is támogatja, nem általános célú programozási nyelvnek terveztük, és külső programkönyvtárakkal sem lehet egybeépíteni, ahogyan a hagyományos nyelveket. A nem blokkosítottság előfeltétele, hogy minden a kiszolgálóhoz készülő kiegészítés magas szinten testreszabott kell legyen. Ehelyett mi egy általános célú parancsfájlkészítő módszert szerettünk volna, ami adatbázisokkal és más rendszererőforrásokkal is képes kapcsolatot tartani. Ezért ACS (Adjunct Communication Server) Projektünk létrehozásakor lényegében ugyanazt a modellt választottuk, mint amit a webkiszolgálók esetében alkalmaznak.

Kiszolgálónk TGI-modellje nagyon hasonló a webkiszolgálók CGI-rendszerének működéséhez. A TGI külön folyamatként indul el, majd az adatokat környezeti változókon keresztül juttatja a telefonhívóhoz. Azért használunk környezeti változókat parancssori paraméterátadás helyett, mert így megelőzhetjük az adatok utáni kémkedést (hiszen azok lényeges dolgokat is tartalmazhatnak, például hitelkártyaszámokat), amelyek egyébként egy egyszerű `ps` paranccsal kiolvashatóak lennének.

A TGI-folyamat szabványos kimeneten keresztül kapcsolódik a kiszolgálóhoz, ahol a TGI-alkalmazás által készített

kimenetet kiszolgáló parancsok meghívására használjuk fel. Ezek a parancsok különböző feladatokat láthatnak el: visszatérő értékeket állíthatnak be, amelyeket például egy adatbázis-keresés eredményeképpen kapunk, vagy új folyamatokat hívhatnak meg, ilyen lehet például a kifelé menő tárcsázás. Ahelyett, hogy minden egyes egyidejű hívásnak



külön átjárót biztosítanánk, minden TGI-átjáróhoz egy-egy folyamatkészletet tartunk fenn, így azokat korlátozott erőforrásoknak tekinthetjük. Feltételezzük, hogy az átjáró végrehajtási ideje a teljes hívási időnek csak kis százalékát jelenti, így a gyors TGI-indítás érdekében hatékony megoldást jelent egy kis számú folyamatkészlet állandó készenlétben tartása. Ez a módszer segít elkerülni a torlódást, például ha az összes hívó éppen egyszerre érkezik a TGI-hez.
Ezekkel az alapvető eszközökkel végül lehetővé vált a hangválaszokat adó (voice response) alkalmazások elkészítése. Amikor a rendszer végre működni kezdett, első telefonos kiszolgálónk már üzleti környezetben futott az Open Source Telecom és más cégek gépein.
Ez a széles körű felhasználás más vállalkozási-gazdasági tényezők mellett részben annak volt köszönhető, hogy annyira könnyen lehet új alkalmazásszolgáltatásokat készíteni, illetve telefonos alkalmazásokat a kiszolgáló alá beépíteni. Mint már korábban megjegyeztük, az egyetlen követelmény az, hogy a rendszergazda kiszolgálóoldali parancsfájlokat legyen képes készíteni, hangfelvételeket tudjon lejátszani és rögzíteni, illetve legyen valamennyi gyakorlata valamilyen általános eszköz (például Perl) használatában.
Kiszolgálónk egy jellemző alkalmazása valahogy úgy nézhet ki, miként azt az *ábrán* láthatjuk (elérhető az ➲ www.linuxvilag.hu/Bayonne címen), ez nevezetesen a ***playrec*** parancsfájl.
A parancsfájl bemutatja a jelenlegi parancsnyelv különböző jellegzetességeit, az érvényességi köröket, az eseménykezelést, amelyek (nevesített parancsfájl-hivatkozások alatt futva) létrehozzák az interaktív telefonos alkalmazás feldolgozásához szükséges logikai láncot. A 2. listában (ami a ➲ www.linuxvilag.hu/Bayonne címen érhető el), a Perl felhasználására mutatunk be egy példát a ***TGI.pm*** modul és a ***tgigetdbval.pl*** Perl parancsfájl segítségével.

Hogyan lett az ACS-ből GNU Bayonne?

Mint korábban már említettük, ezeket a célokat az első ilyen telefonos alkalmazáskiszolgálóval már nagyjából két évvel ezelőtt elértük, ezt Adjunct Communication Servernek, röviden ACS-nek neveztük. Sajnos az ACS névgondokkal küszködött, és számos olyan levelet kaptam különböző emberektől, akik arra mutattak rá, hogy az ACS nevet már több másik projekt is használja, például az Al's Circuit Simulator. Ez természetesen baj volt.
Ugyanakkor az ACS szerkezetének korlátai is kezdtek megmutatkozni. Először is azon az elképzelésen alapult, hogy a kiszolgálót közvetlenül a telefonos kártyához kell kapcsolni, valahogy úgy, ahogyan az XFree86 3 köti az X-kiszolgálót az adott videokártya-családhoz. Ez azt jelenti, hogy minden egyes kártyacsaládhoz külön kiszolgálót kell fordítani, ezáltal sok kód szükségtelenül többszörözötten fog szerepelni.
Úgy döntöttem, hogy a teljes kiszolgálómagot az alapoktól kezdve újraírom, és ezt néhány héten belül be is tudtam fejezni. Az első dolog, amit megtervezte tem, a bővítmények (plugin) támogatása volt, egy kicsit más elképzelés alapján, ahogyan azt a legtöbb ember korábban csinálta.
A bővítmény általában egy kis objektumfájl, amelyet egy ismert szimbólum vagy szerkezet alapján dinamikusan töltünk be, és azután a betöltött fájl átvizsgálásával könnyedén megtalálhatjuk. Így a modult az `dlopen` utasítással bárki megnyithatja, majd a `dlsym` segítségével megkeresheti az adott szimbólumot, hogy aztán a modulon belül függvényeket hívhasson meg.
Én egy másik megoldást alkalmaztam: az új kiszolgálót úgy alakítottam ki, hogy a saját szimbólumait exportálja.
A kiszolgáló egy létrehozókkal (constructor) ellátott alaposztályhalmazzal rendelkezik, amely a rendszerleíró szerkezetet készíti fel a használatra.
A betölthető modulokat C++ nyelven, származtatott osztályként lehet megírni, melyek alaposztálya a kiszolgálóban van megadva, és ezek levezetett osztályainak statikus objektumokat tehet elérhetővé. Amikor a modult az `dlopen`-nel betöltjük, a statikus objektumok létrehozói (constructor) önműködően meghívódnak, az alaposztályból a kiszolgálólenyomatra (server image) mutató hivatkozások pedig önműködően feloldásra kerülnek (resolved).
A kiszolgálólenyomatban tárolt alaposztály a létrehozó által önműködően meghívódik és bejegyzi a modulobjektumot. Így aztán az `dlopen` nemcsak, hogy betölti a modult, de egyben a használatra való felkészítést is elvégzi – mindezt egyetlen műveletben.
Továbbá néhány dolog, ami korábban az ACS része volt, most külön csomagba került. Ekkor vált a GNU ccScript és a

GNU ccAudio külön osztálykönyvtárrá, minthogy ezek jelképezték az ACS-ben korábban is megtalálható igen hasznos parancsnyelvmotort és a hangfeldolgozó szolgáltatásokat. Különösen az iránt érdeklődtünk, hogyan tudnánk a parancsnyelvet más kiszolgálókban használni, amely a GNUCOMM része lehetne.
A GNU ccAudio bizonyítottan hasznos, általános célú hangfeldolgozó könyvtár. Felhasználható egyszeres és duplafrekvenciás hangok előkészítésére, amelyeket később a memóriából lejátszhatunk, továbbá több bemeneti fájlból csomagolt, állandó hosszúságú keretekből (fixed-length frames) álló hangfelvételt képes összeállítani (a végükön szünettel), ahogyan azt a legtöbb DSP (digital signal processor; digitális jelfeldolgozó) bemenete megköveteli. Ez a képesség kiemeli a többi hangfeldolgozó könyvtár sorából, amelyek ilyen mutatványokra rendszerint nem képesek. A legjobb az lenne, ha a GNU ccAudió-t sikerülne teljes értékű, általános célú hangfeldolgozó keretrendszerré fejleszteni, amely egyben kiszolgálóalapú DSP-szerű feldolgozásra is alkalmas.
Megvolt tehát az új kiszolgálónk, mindössze a neve hiányzott. Mivel valami egyedi és mások által valószínűleg még nem használt nevet szerettünk volna, úgy döntöttünk, nem használunk újabb rövidítést. Helyette, mivel a kiszolgáló tulajdonképpen híd a számítógép és a telefonos világ közt, logikusan egy hídmetaforát választottunk. De melyik hídra essen a választásunk?
Kézenfekvő lenne a Brooklyn híd. Csakhogy ez meglehetősen gyakran használt név, és negatív mellékzöngéi vannak, így nem tűnt jó választásnak. Ugyanígy a Golden Gate név is meglehetősen felkapott, és többnyire az IBM Java kezdeményezésével kapcsolják össze. Szóba került a Tacoma Narrows is mint lehetőség, de figyelembe véve, hogy önmegsemmisítéséről volt híres, úgy gondoltuk, ezt inkább ejtjük, meghagyjuk a washingtoni kereskedelmi gyártóknak. Létezik viszont egy híd nem messze innen, New Jersey-től, a Bayonne. Valószínűleg nem nagyon hallott róla senki, tehát a neve is kevéssé használt.

A ma és a holnap

2002 nyara a GNU Bayonne 1.0-s változatának megjelenését jelzi. Jelenleg a GNU Bayonne nemcsak, hogy része a GNU Projektnek, de csomagját több GNU/Linux terjesztés alaprészeként is megtalálhatjuk, így a GNU/Debian és Mandrake-terjesztésekben is. Minthogy a célunk a telefonos alkalmazásszolgáltatások elérhetőségének minél teljesebb körben történő elterjesztése a szabad program fejlesztői közt, ez igen kellemes előrelépést jelent.
A GNU Bayonne-t máris széles körben használják szerte a világon. A felhasználók köre az orosz üzleti távközlési szolgáltatóktól az USA állami és szövetségi kormányzati ügynökségeiig terjed, és számos olyan vállalkozást találhatunk köztük, amelyek különleges hangszolgáltatásokat biztosítani tudó egybekötött webhelyeket vagy hangüzenet-szolgáltatáshoz hasonló vállalkozási alkalmazásokat kerestek.
A GNU Bayonne nem önmagában létezik, hanem egy sokkal nagyobb metaprojekt, a GNUCOMM része.
A GNUCOMM célja a jelenlegi és a következő nemzedékbeli telefonos hálózatok részére telefonos szolgáltatás nyújtása, szabadon felhasználható (free license) program segítségével. Ezeket a szolgáltatásokat a következőképpen határozhatjuk meg

1. olyan szolgáltatások, amelyek az asztali felhasználókkal tartják a kapcsolatot, ilyenek például a telefonszámok tárcsázására képes címtárak és a telefonprogram-alkalmazások (soft phone applications);
2. olyan szolgáltatások, amelyek telefonkapcsolatokat valósítanak meg, ilyen például a IPSwitch GNU softswitch projekt és a GNU oSIP proxykiszolgáló;
3. olyan szolgáltatások, amelyek átjáróként működnek a jelenlegi és a következő nemzedékbeli telefonos hálózatok közt, ilyen a troll és a tűzfallal védett telefonos hálózatok proxykiszolgálói, például az Ogre;
4. valós idejű adatbázis-adatforgalmazó rendszerek, mint például a preViking Infotel és a BayonneDB;
5. hangalapú alkalmazásszolgáltatások, mint amilyeneket a GNU Bayonne-n keresztül is megvalósíthatunk.

Még a GNU Bayonne 1.0 befejezése előtt, 2001 végefelé már elkezdődött a GNU Bayonne örökösének fejlesztése is. Ez az örökös számos olyan szerkezeti választást megpróbál leegyszerűsíteni, amelyek a fejlesztések kezdeti szakaszában kerültek a projektbe, remélve, hogy az új megoldások által a GNU Bayonne-t könnyebb lesz majd átdolgozni és egyszerűsíteni. A terv kiválasztása és a kezdeti tervezés nagy része 2001 végén két nap alatt ment végbe, mialatt a londoni találkozón összejöttem a preViking telefonos kiszolgálót tervező emberekkel. Néhány ilyen változtatás indokolta a preViking projekt közvetlen bevonását a GNU Bayonne fejlesztésébe.
Az egyik legnagyobb kihívást a jelenlegi GNU Bayonne kiszolgáló fejlesztésében a telefonos kártyák moduljainak elkészítése jelenti. Ezek gyakran minden egyes meghajtóhoz komoly fejlesztést követelnek meg, és a kód is sokszor szerepel többszörözötten. A GNU Bayonne 2 ezt a kihívást úgy oldja meg, hogy a vezérléshez használt állapotgépet (state machine) a kiszolgálómagba helyezi át, majd C++-osztálymeghatározásokon keresztül teljes mértékben elfedi. Ezáltal a meghajtók egyszerűbbé válnak, ugyanakkor lehetővé teszi számunkra, hogy egyetlen kódalapból többféle kiszolgálót hozzunk létre.
A GNU Bayonne 2 másik újdonsága a távközlési cégeknek készített linuxos megoldások sokkal közvetlenebb támogatása. Azaz elődjével szemben ez az új megoldás folyamatosan élő kiszolgáló esetén is képes a szolgáltatásból kapukat kivenni és visszahelyezni, ami lehetővé teszi, hogy a kártyákat menet közben telepítsük (hot-plug) vagy cseréljük (hot-swap). A távközlési osztályú kiszolgálókon a rendszermag figyelmeztetést küld a változás tényéről, az alkalmazásszolgáltatások pedig figyelhetik és válaszolhatnak ezekre az eseményekre.
A GNU Bayonne 2-t úgy terveztük, hogy vegye figyelembe ezt a „értesítő” elvet az általa irányított erőforrások kezelése során.
Végül a GNU Bayonne 2 a kezdetektől úgy készül, hogy több módon is kihasználja az XML nyújtotta előnyöket. Beállításnyelvként saját XML-nyelvjárást használ, webszolgáltatásként is működik, amely egyszerre képes az éppen futó GNU Bayonne pillanatnyi állapotát XML formában megadni, illetve az XMLRPC-t támogatni. Ez illeszkedik ahhoz az elképzelésünkhöz, amely szerint a telefonos kiszolgálókat webkiszolgálókkal építhetjük egybe, s amely jól mutatja, hogyan képzeljük el a projekt további útját.

Linux Journal 2002. augusztus, 100. szám

David Sugar
több mint 20 éve fejleszt szabad programokat. A GNU Project számos csomagjának elsődleges szerzője, köztük a GNU Bayonne-é is. Ő az Open Source Telecom alapítója és a DotGNU-t kormányzó bizottság elnöke.