Ökad användarvänlighet av installationen till plattformen P90/E

Stefan Andersson

28 februari 2006
Examensarbete i datavetenskap, 10 poäng
Handledare vid institutionen för datavetenskap: Stefan Johansson
Examinator: Per Lindström

Umeå Universitet
Institutionen för datavetenskap
SE-901 87 UMEÅ
SWEDEN

Sammanfattning

Denna rapport går igenom hur Teligents installationsprocedur för P90/E systemet kan förbättras. Förslag på lösningar inom olika områden samt kravställiningar på systemet har tagits fram. En fördjupning i grafiskt användargränssnitt och hur det kan användas effektivare har gjorts. Ett verktyg som hjälper det existerande installationsprogarammet har implementerats. Detta vektyg använder man för att göra grovarbetet som ska göras före en installation.

Improved usability in the P90/E platform installation

Abstract

This Master's thesis describes how Teligent's installation for the P90/E platform can be improved. Solutions for improvements are proposed and the system's requirements are resolved. A studie has been done on how graphical user interfaces can be used efficient for this purpose. Finally, a tool that helps the existing installation system has been implemented. This tool is used to configure the installation before it is applied.

Innehåll

1 Introduktion **1**

2 Problembeskrivning **3**

2.1 Mål . . . 3

3 Befintligt system **5**

3.1 P90/E . . . 5

3.2 PI . . . 8

3.3 Utvärdering av PI . . . 8

4 Förslag till förbättringar **11**

4.1 Kravställning på komponenterna/systemet . . . 11

4.1.1 Programmeringsspråk . . . 12

4.1.2 Grafiskt användargränssnitt . . . 13

4.2 Distribution . . . 13

4.2.1 Supernod . . . 13

4.2.2 Klient/Server . . . 14

4.2.3 Verifiering . . . 14

4.2.4 Paketering . . . 14

4.2.5 Webbaserat gränssnitt . . . 15

5 Fördjupning **17**

5.1 Grafiskt användargränssnitt . . . 17

5.2 Visualisering av XML-filer . . . 19

5.3 Användbarhet . . . 20

6 Prestation **23**

6.1 Verktyg . . . 23

6.2 Arbetsutförande . . . 23

6.3 Metoder . . . 24

7 Resultat **25**
7.1 Funktionsbeskrivning . . . 25
7.2 Användargränssnitt . . . 29
7.3 Bruksanvisning . . . 32
7.3.1 Systemkrav . . . 32
7.3.2 Installation/Exekvering . . . 32
7.3.3 Menyer . . . 32
7.3.4 Innehåll . . . 32

8 Slutsats **35**
8.1 Begränsningar . . . 35
8.2 Framtida arbeten . . . 36

9 Tack **37**

Referenser **39**

A Komponenter **41**

Figurer

3.1 P90/E Kernel 5
3.2 Noder med P90/E kernel och komponenter. 7
3.3 Installationssteg 8
3.4 Flödesschema för PI. 9

4.1 Supernod 14
4.2 Klient/Server. 15
4.3 Verifiering av semantik. 15
4.4 Distributionsfil. 16

5.1 Monolitisk struktur. 18
5.2 Polylitisk struktur. 19

7.1 JTree komponent. 26
7.2 Klass diagram över XML-generatorn (del 1). 27
7.3 Klass diagram över XML-generatorn (del 2). 28
7.4 Installationsinformation och sökvägar. 29
7.5 Information om produkt. 30
7.6 Befintliga templates med namn. 30
7.7 Visar de komponenter som ingår i noden. 31
7.8 Konfiguration av en komponent. 31

Tabeller

6.1 Arbetsplanering. 23

7.1 Element som visas i JTree komponenten 32

A.1 Komponenter a-g 42
A.2 Komponenter g-p 43
A.3 Komponenter p-t 44
A.4 Komponenter t-z 45

Kapitel 1

Introduktion

Teligent är ett företag som producerar telefonitjänster för det mobila och fasta nätet. Företaget har drygt 300 anställda som är placerade i 11 olika länder. Huvudkontoret ligger i Nynäshamn och de andra svenska kontoren ligger i Stockholm (Globen), Linköping och Umeå. De tjänster som företaget erbjuder är i huvudsak mjukvarulösningar för telefonioperatörer med undantag av några mindre hårdvaruprojekt. Teligent har utvecklat en egen plattform (P90/E) som ligger som grund för alla de komponenter som utvecklas. Installationen av denna plattform är inte helt trivial eftersom man måste konfigurera alla komponenter innan man bygger systemet. Teligent är därför intresserade av ett installationssystem som kan förenkla leveransen.

I kapitel 2 beskrivs det grundläggande problemet av installationsproceduren. Det befintliga systemet beskrivs i kapitel 3 med en grundläggande information om P90/E och P90/E installer samt en utvärdering av P90/E installer. I kapitel 4 ges några förslag till förbättringar och vilka kravställningar som bör ställas på systemet. I fördjupningsdelen, kapitel 5, behandlas grafiska gränssnitt som relaterar till några av de delar som har anknytning till projektet. Kapitel 6 beskriver hur arbetet har lagts upp och vilka verktyg som har använts. Vidare har en applikation som förenklar arbetet med konfigurationen implementerats. Applikationen beskrivs ingående i kapitel 7. Rapporten avslutas med en slutsats av arbetet samt begränsningar och framtida arbeten i kapitel 8.

Kapitel 2

Problembeskrivning

Teligent utvecklar telefonitjänster där den egenutvecklade plattformen P90/E används som grund. P90/E består av ett antal olika komponenter som är sammankopplade i ett nätverk. Vid en leverans av ett system måste kunden konfigurera sitt system med de komponenter som man har för avsikt att använda. Denna process upplevs som svår och behovet av support är en belastning för Teligent. Teligent strävar mot målet att en installation av systemet ska kunna göras på egen hand av kunden.

P90/E är en plattform för telekommunikation och ligger som grund för de komponenter och tjänster som Teligent utvecklar. Systemet har med åren vuxit och antalet komponenter som ligger runt systemet har ökat. Detta har resulterat i en ganska komplex installationsprocedur som ska göras hos kunden när systemet levereras.

För att klara belastningen på systemet använder man flera sammankopplade datorer som kallas för noder. Komponenterna som systemet består av ligger fördelade på dessa. Det går dessutom att köra flera instanser av samma komponent utspridda på olika noder. Detta har inneburit att det har varit svårt för Teligent att utveckla en generell konfiguration för att bygga systemet. Det Teligent siktar mot är att få en installation av P90/E så trivial som möjligt. Det har under åren påbörjats ett antal försök att komma till bukt med detta problem. Vissa implementationer har varit framgångsrika medans andra där man kanske hade lagt ribban lite för högt har har varit mindre framgångsrika.

2.1 Mål

Målet med detta arbete är att analysera befintliga lösningar för konfigurering och installation, samt göra en kravställning på systemet och dess komponenter och implementera någon form av förbättring av installationssteget. Ett önskemål från Teligents sida är att konfigureringen ska ske på ett enkelt och konsekvent sätt över hela systemet. Kunden ska inte behöva en lång utbildning bara för att installera systemet. Som en avslutande del kommer en implementation att göras på en lämplig plats i systemet. Den ska sedan utvärderas och förhoppningsvis byggas ut i hela systemet.

Kapitel 3

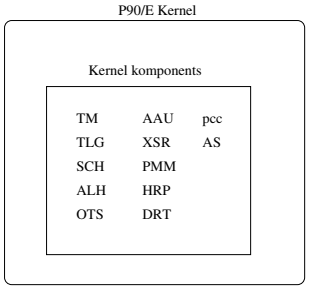
Befintligt system

Det system som används idag för att göra installationer kallas för PI (P90/E Installer). Detta program stöds av de flesta komponenter som Teligent levererar. Alla nya komponenter som utvecklas innehåller detta stöd och man kommer hålla fast vid detta så länge det inte finns en bättre lösning. Här ges en enkel beskrivning av plattformen P90/E och installationsporogrammet PI med en efterföljande utvärdering.

3.1 P90/E

Alla komponenter som utvecklas av Teligent använder plattformen P90/E. Denna plattform ligger ovanpå det ordinarie operativsystemet som oftast är någon form av UNIX eller Linuxsystem. Plattformen är ett lager som ligger mellan operativsystemet och komponenterna. Dess huvudsakliga arbetsuppgift är att se till att komponenterna hålls vid liv och de kan kommunicera med varandra. Kommunikation sker med hjälp av transaktioner som följer ett protokoll som heter Teligent P90 Transaction Format. I denna transaktion finns information som talar om för systemet vilken komponent som avses samt funktionella anrop. När en komponent har skickat iväg en transaktion svarar mottagaren med en ny transaktion som innehåller resultatet av det funktionella anropet.

P90/E:s kärna består av några komponenter som har olika ansvarsområden. Dessa komponenter har högre privilegium än vanliga applikationskomponenter. Figur 3.1 visar innehållet i kärnan (P90/E kernel).

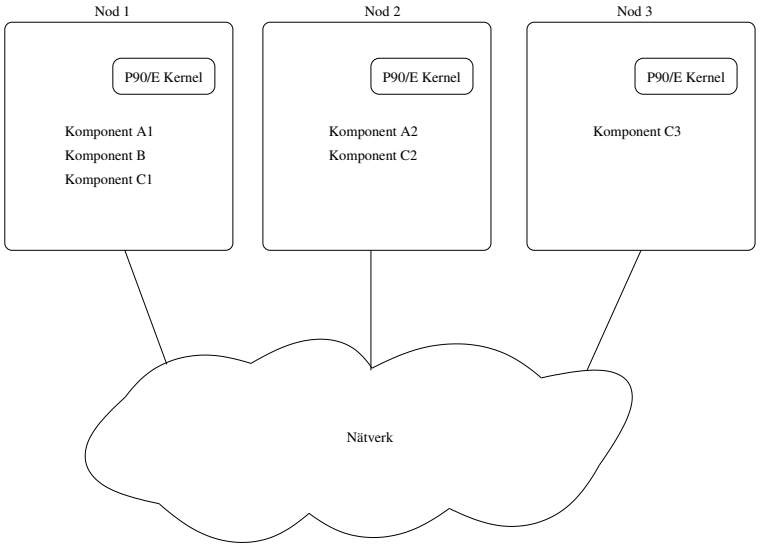


Figur 3.1: P90/E Kernel

En övergripande förklaring av kärnkomponenterna ges här (se även figur 3.1):

- **TM Transaction Manager:** Fungerar som en router för transaktionerna samt startar och stoppar komponenterna och ser till att dessa hålls vid liv.
- **TLG Transaction Log Manager:** Loggar transaktionerna i systemet så att eventuella fel kan spåras och reproduceras.
- **SCH Scheduler:** Hanterar jobb som har ett schema. Applikationer kan trigga dessa för att utföra sekvensiella saker. Innehåller köer, temporär lagring och mekanismer för transaktionsfördröjning.
- **ALH Alarm Handler:** Alla alarm och händelser i systemet passerar genom ALH innan det lagras i CAD(Current Alarm Database). Funktioner finns för att filterera dessa.
- **OTS Operator Terminal Server:** Hanterar operatöråtkomsten och allt vad detta innebär med säkerhet och åtkomst.
- **AAU Authentication and Authorization of Users:** Ansvarar för att lagra information om användare, grupper och rättigheter i en central databas. Denna komponent är oberoende av operativsystemet.
- **XSR Cross System Router:** Med denna komponent kan man koppla samman två eller flera TM:ar så att dessa delar på samma resurser.
- **PMM Performance Measurement Manager:** Samlar ihop statistik för varje komponent så att man kan använda ett verktyg för att ta reda på komponenternas prestanda.
- **HRP Host Resource Performance:** Mäter prestandan i den aktuella maskinen(noden).
- **DRT Data Router:** DRP underhåller systemets routing tabell. TM använder DRP för att hitta vägen till den bäst passande komponenten.
- **pcc P90 Component Controller:** pcc är ett interface mellan komponenterna och TM.
- **AS Application Selector:** AS är en distribuerad databas som fungerar som ett filter och en översättare till vissa transaktioner. Alla komponenter som hanterar extern hårdvara(Linje interface för telefon m.m.) måste översätta händelserna till transaktioner. Detta sker via AS databas.

Varje nod har en P90/E Kernel och ett antal komponenter som ligger runt denna. Systemet kan konfigureras så att samma komponent kan ligga på flera noder och dela på arbetet. Figur 3.2 visar hur detta kan se ut. För att få en överblick på hur stort och flexibelt systemet kan byggas, listas de komponenterna som finns tillgängliga i Bilaga A.

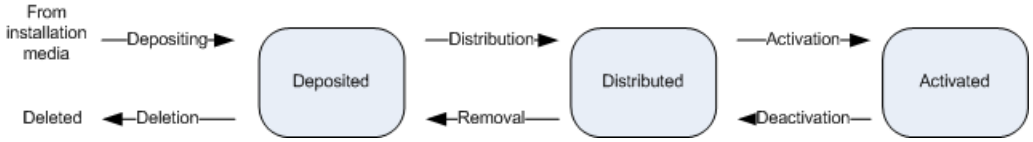


Figur 3.2: Noder med P90/E kernel och komponenter.

3.2 PI

PI (P90/E Installer) är det verktyget som idag används för att distribuera och aktivera komponenterna. Programmet är skrivit i bashskript och använder en XML-fil för att ställa in ändringar i konfigurationen. PI ligger på alla noder i systemet där en väljs som master. På den nod som man har valt som master lägger man in den senaste releasen av mjukvaran i form av en TPD (Teligent Deplyment Package).

När PI startas på masternoder väljer man den release man vill jobba med. Därefter kan man välja mellan att aktivera, deaktivera och distribuera den valda releasen. Vid aktivering och deaktivering körs aktiverings- respektive deaktiveringsskripten på alla komponenter som ingår i releasen. Dessa skript följer med komponenterna vid distributionen och ansvarar för att länka in binärerna på rätt plats och att köra igång och stoppa komponenterna. Distribueringen tar komponeterna som ligger i TPD och distribuerar dessa till systemets noder enligt den systemkonfiguration som finns i XML-filen. Figur 3.3 visar de steg som PI använder vid installation och avinstallation. Ett flödesschema för PI visas i figur 3.4. Här ser man bland annat att flödet beror av installationsfilen install.xml och användarens val i PI menyn.



Figur 3.3: Installationssteg

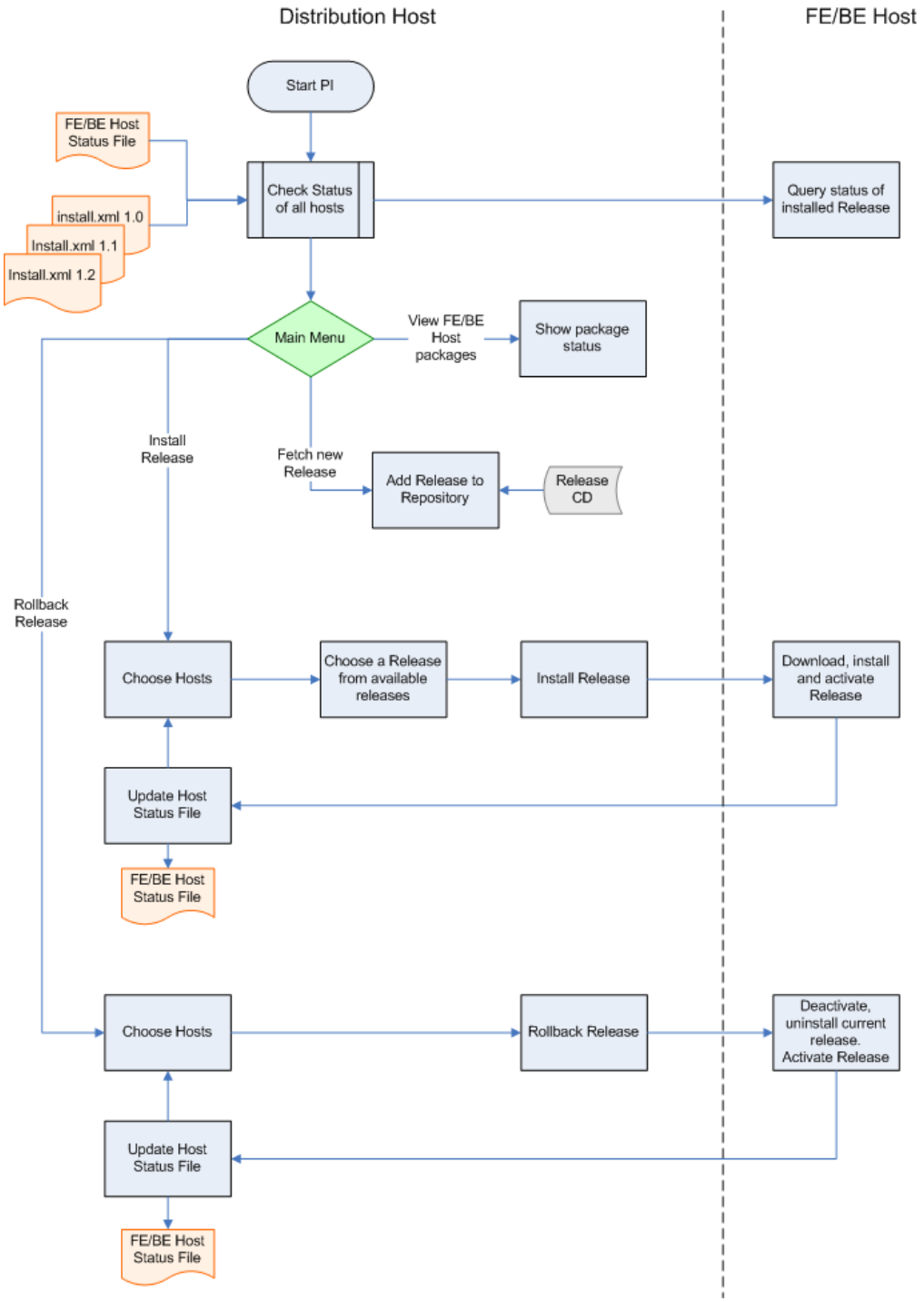
3.3 Utvärdering av PI

Det som uppleves besvärligt med PI är att man måste göra ändringar i en XML-fil för att mjukvaran ska distribueras på rätt sätt. Det kan dessutom vara så att samma information måste skrivas in flera gånger i olika XML-filer för att PI ska fungera på ett korrekt sätt. I de flesta fallen används en helt vanlig texteditor för att göra ändringarna direkt i XML-filen. Detta fungerar bra och går snabbt för en person som arbetar dagligen med detta, men om man sätter dit en ny person, exempelvis en konsult som ska utföra en uppgradering av en komponent, så blir det problem.

När man är klar med en installationsfil install.xml använder man ett skript för att bygga ett Teligent Deployment Package (som en rpm-fil) som innehåller de komponenter som ska finnas med i systemet. Teligent Deployment Package kan därefter distribueras till en nod i systemet där man sedan använder sig av PI för att distribuera komponenterna till de noder som har angetts i install.xml.

PI är ett bashskript och den naturliga användarmiljön är en kommandobaserad variant som körs i ett vanligt terminalfönster. Själva PI upplever jag som ett bra verktyg, enkel och robust och har alla funktioner som behövs för att göra en distribution. De problem som kan dyka upp är sällan relaterat till verktyget PI utan ligger i de filer som PI använder. De rpm-paketen som ska installeras måste använda en struktur som PI kan läsa och XML-filen som styr PI måste vara korrekt.

Har man gjort rätt i alla stegen som ska göras före exekveringen av PI så är installationen inget problem. Råkar man däremot att göra fel i XML-filen som till exempel ett syntaxfel eller eller missa en ändring som måste göras på flera ställen, så blir det



Figur 3.4: Flödesschema för PI.

stora problem som kan ta lång tid att rätta till. Av detta kan man dra slutsatsen att någonting måste göras för att förenkla de första stegen.

Kapitel 4

Förslag till förbättringar

Ett antal förslag till förbättringar av installationsprogrammet för P90/E kommer att presenteras nedan. Förslagen är på en ganska hög abstraktionsnivå och beskriver endast den grundläggande strukturen i systemet. För att dessa förslag ska fungera i praktiken måste vissa kravsällningar göras på den omgivande miljön. Kravsällningen på den omgivande miljön kommer också att beskrivas nedan.

4.1 Kravställning på komponenterna/systemet

För att kunna utveckla ett nytt installationssystem måste vissa krav ställas på komponenterna och systemet som programmet ska arbeta med. Ett installationsprogram bör vara plattformsoberoende och kunna köras både lokalt och mot fjärrsystem. Dess källkod ska vara skriven så att det är enkelt att utföra förändringar och lägga till nya funktioner. Det måste vara enkelt att från källkoden anropa/köra externa kommandon och läsa av resultatet från dessa. Installationsprogrammet måste kunna distribuera installationspaket till de berörda noderna och installera dessa.

Efter installationen måste en konfigurering av noden/noderna kunna göras med externa kommandon. Detta innefattar bland annat den procedur som stänger ner ett existerande system och länkar in de nya binärerna och kör igång dessa. Programmeringsspråken som kan användas för att uppfylla dessa krav bör i någon form vara objektorienterade med en enkel syntax. De språk som ligger närmast till hands är Java och PHP, men det finns egentligen ingenting som talar emot att använda sig av C++ för utveckla en komponent som sköter distributionen i installationsproceduren.

Kravställningen på själva installationspaketet är beroende på hur mycket som ska göras av installationsprogrammet. Installationspaketen bör innehålla en standardkonfiguration som med hjälp av installationsprogrammet kan ändras. För att installationsprogrammet ska få någon slags återkoppling om vad konfigurationsvariablerna ska innehålla måste man ange datatyperna på variablerna. Detta kan göras med en enkel XML-fil som anger alla variabler och dess datatyper. Denna fil ska följa med alla komponenter och med hjälp av denna går det att verifirera semantiken i installationsprogrammet. Den optimala lösningen är att låta installationsprogrammet sköta all konfiguration utan någon inblandning av små hjälpskript som körs vid aktiveringen. Detta underlättar de problem som kan uppstå när en förändring görs på en komponent som en annan komponent är beroende av.

I ett system där konfigurationen är interaktiv måste systemet ta hand om de situationer som uppkommer då en komponent är beroende av en annan. När en förändring i systemet utförs måste en konfiguration av de övriga komponenterna utföras i någonting som kan liknas vid en kedjereaktion. För att lösa detta måste en server i någon form köras på alla noder som ska behandla dessa förändringar. Klientdelen, eller om vi vill kalla det för installationsprogrammet, måste distribuera denna information till alla noder. Problemet med detta är hur noderna ska kunna informeras om vad som finns i systemet. Detta löser man enklast med broadcasting över UDP som informerar alla noder om förändringen. Kravet är då att alla noder måste ligga i samma närverk. En fördel med denna uppbyggnad är att systemet blir dynamiskt, det blir enkelt att flytta runt noderna/komponenterna i nätverket utan en massa inställningar.

Ett annat alternativ är att klienten måste informeras om alla noders IP-adresser varefter klienten frågar varje nod om dess konfiguration. Problemet med denna metod är att operatören av installationsprogrammet måste lägga till nodernas IP-adresser i klienten, men man slipper att vara bunden till ett nätverk. Den bästa lösningen är att implementera båda lösningarna. Systemet får då större utbyggnadsmöjligheter och noderna kan placeras i olika nätverk.

En viktig del i kravställningen är att komponenterna måste anpassas så att konfigurationen är konsekvent över hela systemet. Installationsprogrammet måste kunna göras generell så att det inte behöver skrivas om eller underhållas när en nyutvecklad komponent ska installeras. Använder man någon form av XML-filer för denna information så är det viktigt att hålla sig till de element och attribut som installationsprogrammet kan tolka.

Antalet konfigurationsmöjligheter på komponenterna är alldeles för många enligt mig själv. Att konfigurera komponenterna tar för mycket arbetskraft och därför bör man lägga ner lite mer tid till att hålla dessa på en låg nivå. Men här kommer frågor in om hur flexibel en komponent ska vara. Det jag tror Teligent tjänar mest på är att försöka sälja standardkomponenter med de saker som det finns efterfrågan på. Vill kunden ha någon annan funktionalitet kan det kosta lite extra.

4.1.1 Programmeringsspråk

Valet av programmeringsspråk är en viktig detalj om man har för avsikt att göra ett grafiskt användargränssnitt. Ett objektorienterat språk gör att abstraktionsnivån blir bättre och koden blir lättare att underhålla. I det här fallet bör man se till att man väljer ett språk där det är enkelt att anropa externa kommandon i systemet. Grafiskt användargrännsitt måste också vara enkelt att knyta till programmet. Prestandan på systemet är inte så viktigt eftersom det inte är någon större tidspress på installationen eller uppgraderingen. Om man tittar på de krav som ställdes upp för systemet så lutar valet åt att använda Java eller PHP. Java har ett bra stöd för GUI och det är enkelt att skicka information via ett nätverk, exekveringen av externa kommandon är också möjlig. PHP är oerhört kraftfullt när det gäller att köra externa kommandon och de bibliotek som finns att tillgå liknar de som används i C programmering. Informationsöverföringen arbetar på en ganska låg abstraktionsnivå, så det kräver lite extra arbete här. Det man i första hand tänker på när man pratar om PHP:s grafiska möjligheter är ett webbaserat gränssnitt, men PHP har även stöd för GTK (Gimp Tool Kit). GTK-PHP är ganska nytt och som med allting annat som är nytt så har det lite barnsjukdomar, därför bör man vänta ett tag innan man gör någon implementation som använder detta. C++ är ett alternativ om man använder det för att implementera serverdelen på noderna, men

klientdelen bör skrivas i något plattformsoberoende språk.

4.1.2 Grafiskt användargränssnitt

Användargränssitter bör vara anpassat så att det det ska vara enkelt att göra en ny distribution åt en ny kund. Det måste finnas en möjlighet att göra ändringar i komponenternas konfiguration. Problemet med detta är att antalet komponenter som installationsprogrammet ska klara av är relativt hög. Dessutom har varje komponent ett antal parametrar som är specifika för enbart denna komponent. En siffra som jag har hör men inte kan verifiera är att det finns ungefär 50000 olika parametrar. Om man sedan lägger till antalet permutationer som kan göras så blir detta problem ganska komplext. Ett användargränssnitt som kan hantera detta på ett intelligent sätt är svårt att implementera. Med intelligent menar jag hur programmet beter sig om en parameter ändras i en komponent som i sin tur får följder i en annan komponent. För att göra ett bra och ekonomiskt försvarbart GUI bör vissa delar av programmet byggas utan intelligens.

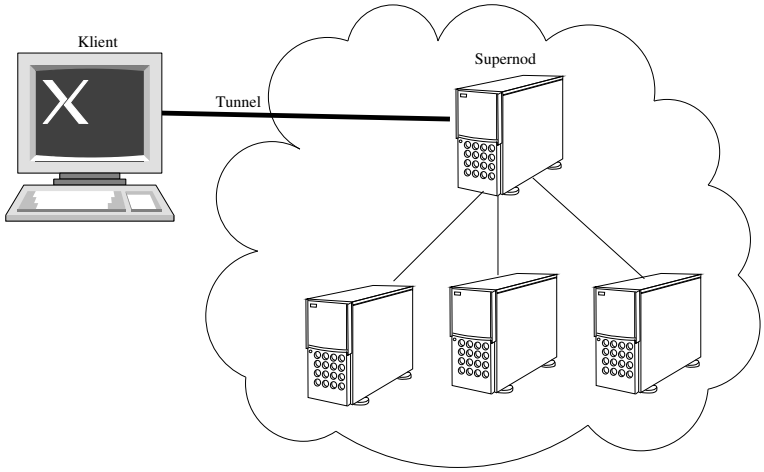
Det optimala vore förstås om man bara kunde välja komponenter från en lista och dra in dem i en graf som beskriver nätverkstopologin. Beroenden kan därefter visualiseras med hjälp av av pilar som går mellan noderna. Konfigureringen skulle kunna göras med hjälp av någon slags drop down meny där man väljer de inställningar som ska ändras. Ett problem med detta är att stora system kommer att kännas besvärliga att editera. Antag att vi har 50 noder och antalet komponenter på varje nod är i snitt 5 stycken. Den grafiska representationen kommer att bli rörig med alla pilar och referenser till de olika komponenterna. Man bör därför väga in för- och nackdelar med en allt för grafisk representation.

4.2 Distribution

Till de noder som ska installeras vill man distribuera en beskrivning på hur systemet ser ut och vilka paket som ska installeras på noderna. Det finns flera olika sätt att distribuera denna informationen, de flesta sätten kräver att man har både någon form av server samt klient på varje nod som sköter konfigureringen. I kapitel 4.2.1 och 4.2.2 beskrivs två olika metoder, supernod respektive klient/server. Topologin på själva nätverket har ingen betydelse för installationen utan kan varieras på olika sätt. Överföringsprotokollet bör använda sig av TCP för att säkerställa att informationen överförts på ett korrekt sätt. Distributionen av de paket som ska installeras görs enklast med verktyget cvs (Concurrent Versions System,[3]). Med hjälp av en systembeskrivning som talar om vilka paket och versioner som ska installeras kan serverdelen på noderna använda cvs för att installera rätt paket. Detta kräver dock att Teligent eller kunden har en cvs-server som är anpassad för detta ändamål.

4.2.1 Supernod

Supernod är den metod som PI använder för distribution. Den bygger på att man väljer ut en nod som master och de andra får agera som slavar. Den nod som man väljer som master distribuerar en fil som beskriver systemet till de andra noderna. Därefter kan olika installationskommandon distribueras till noderna, som till exempel INSTALL eller UPDATE. Figur 4.1 visar hur supernoden distribuerar informationen i det lokala nätverket.



Figur 4.1: Supernod

4.2.2 Klient/Server

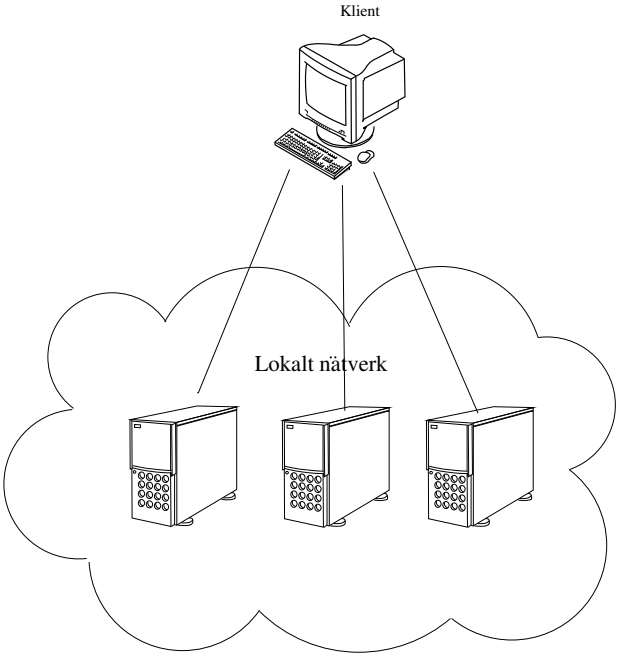
En klient/servermetod använder sig av en klient som är uppkopplad till alla noder i systemet. I detta fallet behöver man inte distribuera hela systembeskrivningen utan bara det som är specifikt för en viss nod. Om serverdelen på noderna berikas med en kontroll av semantiken på konfigurationsinformationen får man ett mer interaktivt system som är lättare att konfigurera. För att verifiera semantiken krävs det att komponenterna har datatypsinformation på alla sina konfigurationsvariabler. Denna funktionalitet går även att bygga in i ett Supernodnätverk, men det kräver ett mer sofistikerat protokoll där supernoden måste agera som en variant av en router. Detta betyder också att noderna måste checka ut de berörda paketen och installera dessa innan det går att kontrollera komponenternas konfiguration. Eftersom klienten ligger utanför det lokala nätverket ställer det krav på brandväggar och den säkerhetspolicy som kunden har. Figur 4.2 visar hur klienten kommunicerar med noderna som ligger i ett lokalt nätverk.

4.2.3 Verifiering

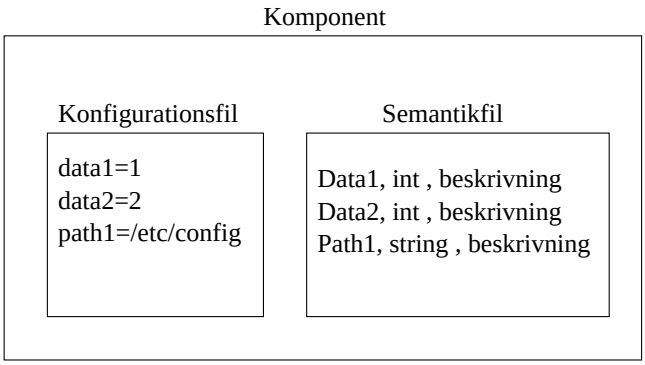
För att installationsprogrammet ska kunna verifiera semantiken i konfigurationen bör man använda sig av en fil som beskriver konfigurationsvariabler och datatyper. Denna information bör placeras i samma katalog som komponentens konfiguration finns, se figur 4.3. Komponenterna som ligger på en cvs-server måste installeras på noderna innan själva verifieringen kan göras. Detta ser jag som ett problem eftersom man skriver konfigurationsfilen innan systemet installeras. En lösning på detta är att man förser installationsprogrammet med en funktion som checkar ut de berörda paketen och packar upp verifieringsinformationen. Detta gör att man kan sitta vid en lokal dator och skriva konfigurationsfilen och verifiera denna.

4.2.4 Paketering

Filerna som ska distribueras kan paketeras som en rpm-fil alternativt som ett tar-arkiv. Eftersom paketeringen endast syftar till att komprimera och skapa ett enkelt sätt att föra över filerna via ett medium, har konfigurationen ingen betydelse här. Konfigurationen



Figur 4.2: Klient/Server.

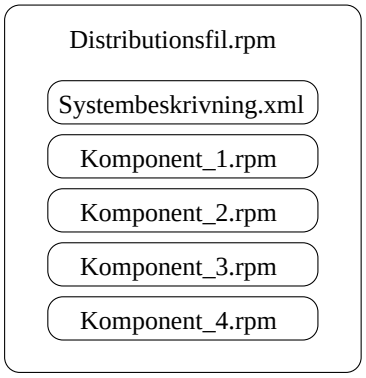


Figur 4.3: Verifiering av semantik.

av komponenterna finns redan inbakade i komponenternas rpm-paket (alt. tar-arkiv) och den medföljande systembeskrivningen, se figur 4.4. Serverdelen på varje nod har till uppgift att installera dessa paket och byta ut standardkonfigurationen mot den som är definierad i systembeskrivningen.

4.2.5 Webbaserat gränssnitt

För att göra det enkelt att ändra konfigurationen skulle det vara en bra idé att göra ett webbaserat gränssnitt mot noderna. Detta kräver dock att man implementerar en enkel webserver eller använder någon befintlig. Denna service bygger på att man redan har ett fungerande system och vill göra en liten ändring på ett snabbt och enkelt sätt.



Figur 4.4: Distributionsfil.

Man är fortfarande beroende av någon slags fil som beskriver hela systemet vid en ny installation.

Kapitel 5

Fördjupning

5.1 Grafiskt användargränssnitt

Tanken med ett grafiskt användargränssnitt är att abstraktionsnivån ska bli högre och att det ska kännas mer naturligt att arbeta. Grafiska användargränssnitt har funnits med ända sedan tidigt 80-tal och de flesta komponenter som används i ett fönsterbaserat gränssnitt har inte fått några större eller betydelsefulla förändringar. De förändringar som har tillkommit är i huvudsak kosmetiska där komponenterna har försetts med 3D känsla.

Vi känner alla till fönster, menyer och dialoger vid det här laget och vi har fått lära oss att arbeta med de. Men varför ser komponenterna ut som de gör? Detta beror till stor del på de inmatningsmöjligheter vi har att tillgå, det vill säga tangentbord och mus. Jag kan även tänka mig att man från början designade komponenterna efter metaforen ”skrivbord”, för att de som var vana att använda penna, papper och skrivmaskin skulle känna igen sig.

Om vi bara använder tangentbord och mus för inmatning av data så fungerar dessa komponenter bra. Jag misstänker att de kommer att finnas kvar ganska långt in i framtiden även om vi byter ut inmatningsmöjligheterna. Det som är viktigt är att man bygger in funktionalitet som gör att användaren har möjlighet att avancera i kunskap om programmet. Detta gör man till exempel när snabbtangenter beskrivs i menyer och popup fönster, användaren utsätts då för inlärning varje gång funktionen ska utföras. I en artikel av E. Dubuis [6] beskrivs tre saker som måste uppfyllas för att ett grafiskt användargränssnitt ska betraktas som bra:

- **Igenkänning:** Hur snabbt kan meningen med ett grafiskt element bestämmas?
- **Minne:** Funktionaliteten av det grafiska elementet.
- **Diskriminering:** Gruppera komponenter.

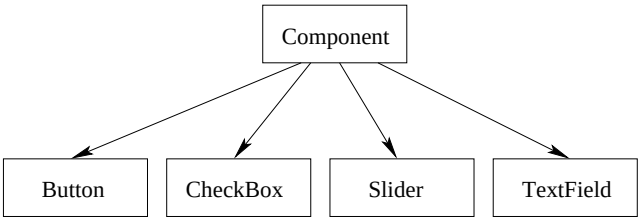
På samma sätt har han tagit fram tre regler man bör tänka på när man gör sin layout av komponenterna.

- **Balans:** Komponenten bör vara anpassad så att ögat dras till mitten av den. Ramar runt komponenterna är ett bra trick för att få denna effekt.
- **Rutsystem:** Liknande komponenter bör ligga på samma rad eller kolumn för det ska bli estetiskt tilltalande.
- **Proportionalitet:** Liknande komponenter bör ha samma storlek.

Om dessa punkter betraktas ordentligt har man goda chanser till ett väl fungerande gränssnitt.

En viktig sak som man bör tänka på när man gör en grafisk applikation är att använda standard komponenter så mycket som det går. Användaren upplever det enklare att lära sig programmet om det finns grafiska objekt som han/hon har använt sig av tidigare. Det är därför viktigt att applikationen implementeras så att det grafiska objektet motsvarar den förväntade funktionaliteten.

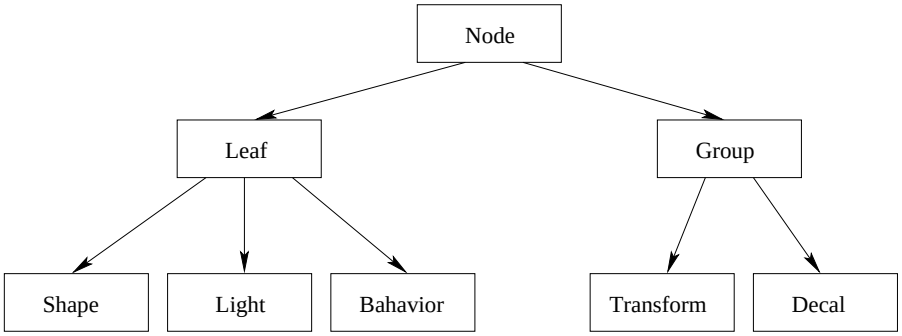
De komponenter som finns i Microsoft's MFC, .NET samt Sun's AWT och Swing är designade så att den största delen av implementationen ligger ganska högt i objekthierarkin. I artikeln *Toolkit Design for Interactive Structured Graphics* [2] beskriver man dessa som konkreta klasser som är starkt bundna till verkligheten. Detta gör att det är relativt enkelt att använda dessa komponenter i applikationen. Att designa egna komponenter blir däremot lite besvärligare eftersom programmeraren måste implementera funktionalitet i de ärvda klasserna. Artikeln beskriver denna uppbyggnad som monolitisk. Motsatsen till detta kallas för polylitisk och ett exempel på en sådan uppbygnad kan vara Java 3D som har kameror, transformationer och grafiska objekt som man lägger till en scen. Funktionaliteten, som i det här fallet är en transformering läggs till som en klass i scenen och detta gör att denna kan bytas ut i realtid. Funktionaliteten i monolitiska system måste däremot göras vid kompileringen, figur 5.1 och 5.2 visualiserar skillnaden. Man har jämfört de två olika uppbyggnaderna och kommit fram till att prestandamässigt så är polylitisk uppbyggnad mycket effektivare än den monolitiska som vi är vana vid. Kanske finns det någon baktanke med att designa ett grafiskt toolkit på ett



Figur 5.1: Monolitisk struktur.

monolitisk vis. Det första man tänker på är att det blir väldigt enkelt att använda dess komponenter. Applikationen behöver inte mycket kod för att visualisera dessa element och detta gör att visuella programmeringsmiljöer går smidigt att använda. Hur skulle programmen se ut om det var enklare att skapa egna komponenter än att använda de som följer med i toolkitet? Jag misstänker att detta skulle ställa till med stora besvär för användarna.

Komponenterna som följer med i dessa toolkits bör användas i största möjliga mån och man bör hålla sig till denna design. I de fall där det inte finns någon komponent



Figur 5.2: Polylitisk struktur.

som passar in bör man se till att den egentillverkade komponenten får samma utseende och karaktärsdrag som de andra så att applikationen blir konsekvent.

Look&Feel är en annan sak som bör utnyttjas om man inte vill att applikationen ska se ut som vanligt. Här kan man ändra lite på färg och form medans funktionaliteten ligger fast förankrad vid applikationen. Detta är ett bra sätt att få lite variation utan att röra till det för användaren. Man bör även ge användaren en möjlighet att ändra mellan olika teman och standardtemat bör alltid finnas med som val.

Om applikationen ska behandla mycket information måste designen vara uppbyggd så att informationen grupperas. Användaren måste känna att informationen tillhör gruppen på ett naturligt sätt. Uppdelningen av informationen kan göras på flera olika sätt med de standardkomponenterna som följer med i dessa toolkit. Som ett exempel kan informationen delas upp med dialoger, expanderbara träd eller under olika flikar.

Färgsättningen av komponenterna bör också följa de standardvärden som finns med i komponenterna. I boken Interaction Design [11] har man intervjuat några användare och experter och kommit fram till att man ska ta det försiktigt med användandet av färg och animationer, eftersom det kan upplevas som störande. Det finns fall då gränssnittet vill att användaren ger större uppmärksamhet och då kan det vara befogat att använda någon färg eller animation. Att välja färger är en svår konst men det finns en regel som är bunden till ögats fysiska egenskaper. Använd aldrig färger som ligger långt ifrån varandra i färgspektrat. Till exemepl blått och rött ligger på var sin sida om spektrat och skriver man med röd text på en blå bakgrund ser texten suddig ut.

5.2 Visualisering av XML-filer

I takt med att XML har fått en större betydelse för att representera data efterfrågas också verktyg för att redigera dessa i grafiska användargränssnitt. Det finns idag en rad olika verktyg för detta ändamål där man kan redigera XML-filens element och attribut. Problemet med dessa är att de följer XML-filens arkitektur som en trädbaserad model. Vill man editera de attribut som ligger längst ut i trädet är man tvingad att editera varje gren för sig. Om trädet är stort är det besvärligt att orientera sig i strukturen.

För att utnyttja de grafiska komponenter som följer med i Java Swing eller i andra toolkit måste man skriva ett program som är anpassat för XML-filen. Problemet dyker upp då ny funktionalitet läggs till i XML-filen, detta medför att applikationen måste skrivas om för att stödja ändringarna.

En lösningen på denna omständiga procedur kan vara UIML (User Interface Markup Language)[1] som använder element och attribut i XML-filen för att bygga upp ett grafiskt gränssnitt. UIML har inget större stöd för att behandla data utan detta lämpar sig bäst för att mata in data i filen. I [1] beskrivs UIML som ett språk för att göra grafiska interface till olika plattformar. Där anser också jag att UIML:s största användningsområde finns, men det finns även ett annant användningsområde för detta språk. Om man blandar ihop XML och UIML i samma fil så har man både data och användarinterface i denna fil. Blir det förändringar i XML strukturen är det enkelt att korrigera gränssnittet som ligger i samma fil. Ett UIML tillägg till installationfilen (install.xml) för PI skulle vara ett alternativ till den applikation som har utvecklats i detta projekt. Naturligtvis så är man fortfarande bunden till en applikation som kan läsa av UIML informationen och presentera detta samt att editera datat. Fördelen är att applikationen inte behöver något underhåll vid framtida förändringar. Lay P. mfl. beskriver i [9] hur man kan använda XSLT för att utföra samma sak. Det som skiljer denna metod från den tidigare är egentligen endast syntaxen på de element som beskriver gränssnittet.

5.3 Användbarhet

Användbarhet är en av de viktigaste delarna i ett användargränssnitt. Ett gränssnitt med en bra användbarhet har dessa kriterier[12]:

- Teknologin ska vara transparent.
- Optimal åtkomst till funktionalitet
- Fokusera på ett arbete inom ett bestämt område.

Det som jag anser viktigast är att den funktionaliteten som används mest bör vara lätt att komma åt. Naturligtvis bör man kunna ställa in alla olika parametrar från gränssnittet, men de funktioner som används minst kan stoppas undan i någon konfigurationsmeny eller något liknande. Användbarheten beror till stor del på hur effektivt gränssnittet är. Det ska finnas möjlighet att endast använda tangentbordet för den erfarne användaren. För att optimera användbarheten går det att göra en empirisk undersökning med testpersoner. Ivory och Hearst beskriver i [8] hur man kan gå till väga för att automatisera detta arbete. Programmet som ska optimeras kan förses med en funktionalitet som loggar alla händelser och mäter tiden mellan dessa. Tangentbordstryckningar, musklick, musrörelser, händelser från operativsystemet och händelser från det grafiska toolkitet, är några av de parametrar som används. Dessa händelser kategoriseras efter betydelse för programmet. Med detta datat kan man senare göra en automatisk analys och avgöra vad det är i gränssnittet som upplevs svårt eller tar för lång tid. I artikeln använder man en avancerad mjukvara och hårdvara för att logga alla händelser men för att ta fram en användbar data för kommersiellt bruk tycker jag att det räcker att tidsstämpla de händelser som det grafiska toolkitet levererar. Med händelserna från toolkitet går det att analysera:

- **Inlärning**: Tiden som krävs för inlärning.
- **Återkommande sekvenser**: Bör rationaliseras bort och läggas till som en funktion som programmet utför.
- **Effektivitet**: Hur snabbt användare löser ett problem.

– **Användbarhet**: Ett totalt mått på användbarhet. Ett värde som går att jämföra mot andra gränssnitt.

Detta anser jag ger en kostnadseffektiv lösning för att effektivisera av programmet. Denna metod går även att använda i ett skarpt system för att optimera programmet för slutanvändaren.

Kapitel 6

Prestation

Eftersom jag har tagit på mig ett extra arbete hos Teligent vid sidan om examensarbetet har det varit svårt att sätta upp någon ordentlig tidsplan. Det jag har gjort var att sätta upp några punkter med en uppskattad arbetstid för varje delmoment, se tabell 6.1.

2 veckor	Inläsning av Literatur och specifikationer
1 vecka	Studier av existerande kod.
1 vecka	Design av lösning
4 veckor	Implementation
1 vecka	Fördjupning
2 veckor	Rapportskrivning

Tabell 6.1: Arbetsplanering.

6.1 Verktyg

Programmeringsspråket för den valda implementeringen är Java och för att snabba på den grafiska delen så användes NetBeans för att implementera det grafiska gränssnittet. Till rapportskrivandet har Latex och BibTex använts samt en mall som Jonas Birmé och Ola Ågren har utformat. För att kunna arbeta på olika ställen har Subversion[5] använts för versionshantering. Övriga delar såsom datorutrusting och intern dokumentation har Teligent stått för.

6.2 Arbetsutförande

Den första fasen i arbetet bestod i att samla på mig så mycket information som möjligt. Här har den interna dokumentationen varit till stor hjälp. Jag har läst in mig på hur P90/E systemet fungerar på ett övergripande plan samt hur det existerande installationsprogrammet PI fungerar. Här har jag även haft nytta av att jag arbetar med ett projekt vid sidan av examensarbetet.

PI testkördes och dess kod granskades för att få en en grundläggnde förståelse hur det fungerar. Under denna fas började en idé att ta form om hur redigerandet av XML-filen skulle kunna förenklas. Den grundade sig på att jag själv tyckte att det var invecklat att skriva en konfigurationsfil.

Jag började inrikta mig på att förstå och analysera själva innehållet i alla delar av XML-filen. En enkel implementation som läser in och tolkar en skarp version av en XML-fil lade grunden till XML-generatorn (se kapitel 7).

För att enkelt kunna behandla en XML-fil i Java var jag tvungen att läsa på lite om Document Object Model (DOM)[7] och XPath[4]. Jag såg möjligheterna till ett enkelt användargränssnitt och började implementera en grundläggande klass som gjorde om XML-filen till ett DOM objekt. Därefter implementerades det grafiska gränssnittet runt detta.

Rapportskrivandet har skett parallellt med utvecklingen av XML-generatorn och anpassats till detta.

Det projektarbete som utfördes vid sidan av examensarbetet har periodvis varit intensiv och därför tagit all fokus från examensarbetet. Detta har medfört att det har gått åt lite extra tid till att återuppta arbetet. Fördelen är att arbetet har känts omväxlande.

6.3 Metoder

För att analysera och ta fram förslag på lösningar har brainstorming och personligt tycke varit den största källan till detta arbete. Samtal med personal som använder PI i sitt dagliga arbete har också varit en bra källa för information. För att ta reda på hur PI fungerar har jag tittat på olika konfigurationsfiler och stegat igenom källkoden så mycket som möjligt. En del av informationen fanns att tillgå på den interna projektsidan för PI.

För implementationsdelen har jag använt mig av en färdig och verifierad installationsfil som utgångspunkt. Denna har analyserats för att ta fram lämpliga element som de grafiska elementen skulle kunna bindas till. För att göra XML hanteringen så bra och effektiv som möjligt har jag använt mig av XPath[4] och DOM[7]. Med hjälp av dessa gjorde jag ett program som tolkar XML-filen och visar den i textuell form. XPath har använts för att göra sökningar efter element och attribut i DOM objektet. Det grafiska användargränssnittet lades därefter till på den textuella trädstruktur som hade tagits fram. Därefter började arbetet med att lägga till information från det grafiska gränssnittet till XML-filen via ett objekt som är baserat på DOM.

Kapitel 7

Resultat

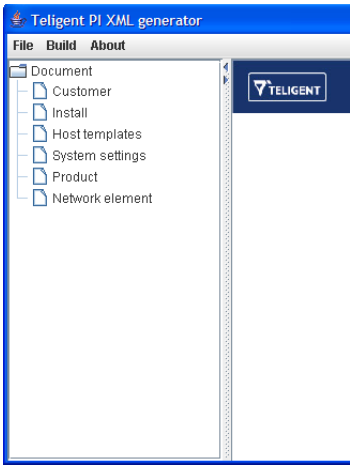
Den implementation jag har valt att göra är ett grafiskt användargränssnitt till XML-filen som PI använder. PI är det nuvarande installationsprogrammet som Teligent använder för sitt system och alla dess komponenter. XML-filen som PI använder sig av beskriver hela systemet och dess konfiguration. Den huvudsakliga informationen i XML-filen är vilka paket som ska installeras på noderna och vilka komponenter som ska köras. Komponenterna som ligger i releasepaketen har en standardkonfiguration. När någonting måste ändras i denna konfiguration så inför man dessa ändringar i XML-filen som PI använder sig av. XML-generatorn är skriven i Java för att behålla plattformsoberoendet och för att Java har ett bra stöd för XML hantering. De grafiska komponenterna som används i XML-generatorn är vanliga standardkomponenter från Swing biblioteket som följer med en Java installation. När XML-generatorn startar använder den en template-fil som sätter upp ett minimalt XML träd som behövs för att det ska fungera. Användaren kan därefter lägga till och konfigurera systemet utifrån det. För att göra sökningar och traversera XML-filen används XPath [4] tillsammans med DOM [7] för göra XML-dokumentet

7.1 Funktionsbeskrivning

XML-generatorn är skrivet i java och använder vanliga standardkomponenter i Swing-biblioteket för den grafiska delen. En template-fil används för att bygga upp ett XML skelett när XML-generatorn startas. Denna fil parsas igenom för att bygga ett DOM objekt för att minska behovet att konvertera XML data till olika format. DOM modellen används sedan för att göra sökningar med XPath frågor. XPath frågor har ungefär samma funktionalitet som vanliga SQL frågor men är mer anpassat för trädstrukturer. Detta passar bra eftersom XML-filen har en trädstruktur samtidigt som syntaxen för XPath är ganska enkel. Avancerade förfrågningar på DOMobjektet går att göra med enkla XPath frågor och detta miskar behovet att söka igenom trädstrukturen.

Den JTree komponent som visas längst till vänster i XML-generatorn använder en abstrakt trädklass som gör att vissa delar av XML-filen visas, se figur 7.1. De element som visas där är en stomme som måste följa med i alla genererade XML-filer. XML-generatorn använder dessa element för att lokalisera var i dokumentet som en förändring ska ske.

Undermenyerna är till den största delen uppbyggda av JTable med en abstrakt TableModel som visar alla attribut i elementen som man har för avsikt att visa. På några

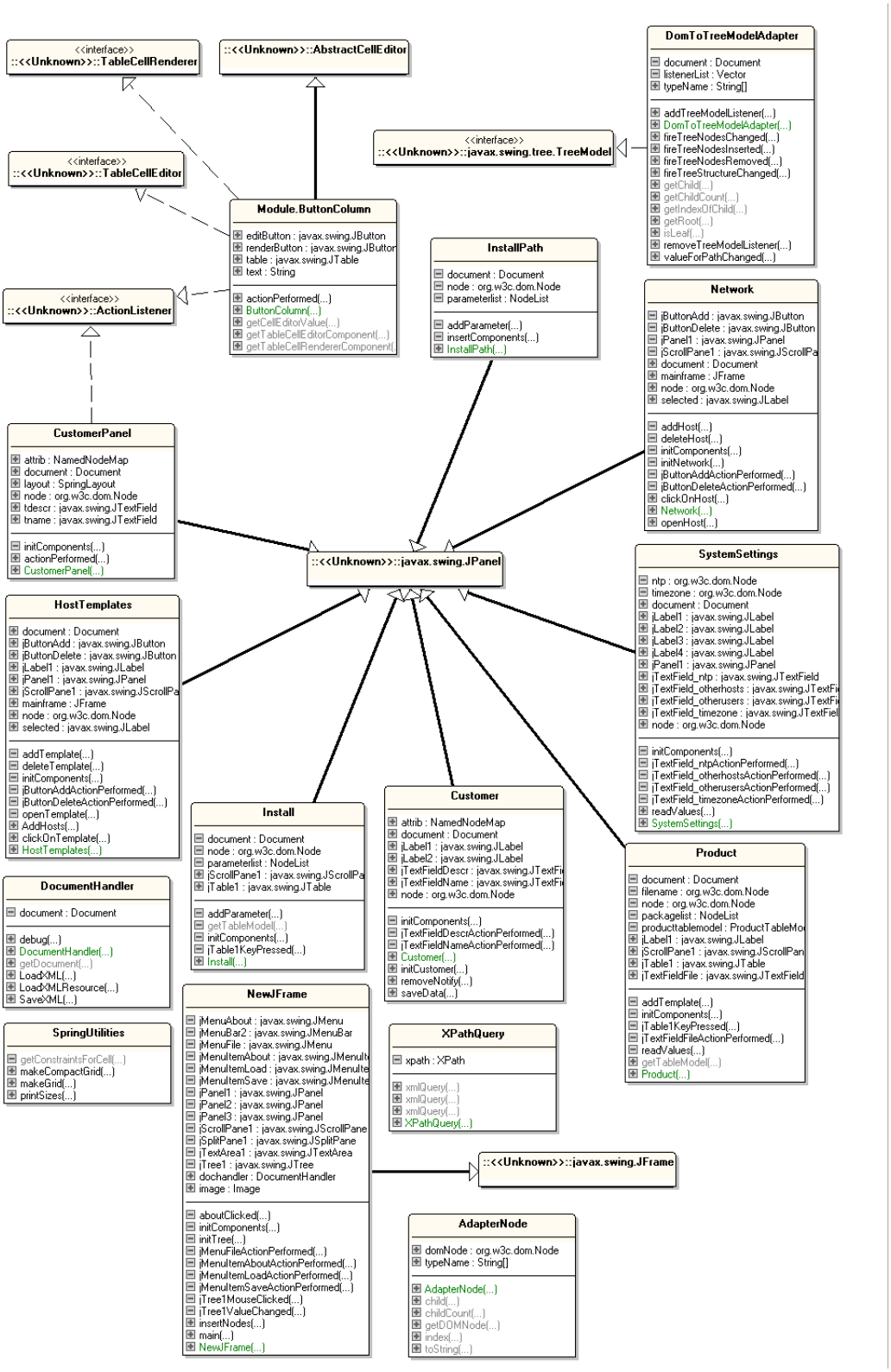


Figur 7.1: JTree komponent.

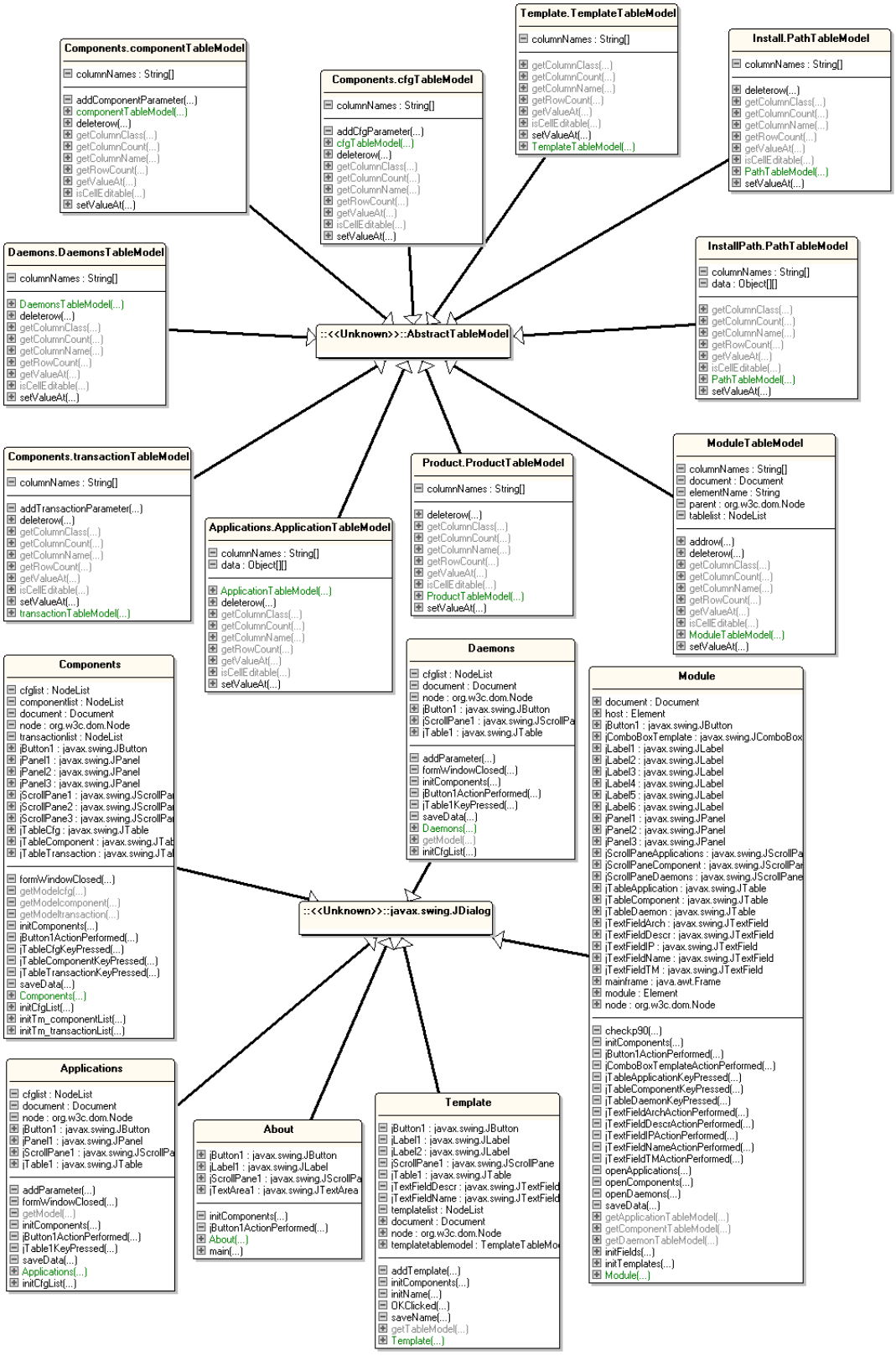
ställen använder XML-generatorn JTextField för att visa de element som endast har ett fåtal attribut.

Vid ett val av en undermeny görs en sökning med XPath så att noden som beskriver undermenyn skickas med som attribut till undermenyn. På liknade sätt görs detta i undermenyn då ett annat element har för avsikt att editeras. XML-generatorn bryter sönder trädstrukturen när nya objekt skapas. Detta betyder att XPath sökningarna blir effektivare eftersom den utgår från den nod som valdes av användaren.

För att lägga till information i DOM objektet måste XML-generatorn se till att en korrekt sökväg finns i trädstrukturen. Om den inte finns lägger den in vanliga element med de nyckelvärden som erfordras. Därefter kan element och attribut skapas och läggas in på rätt ställe i noden. Figur 7.2 och 7.3 visar klassdiagramen över implementationen.



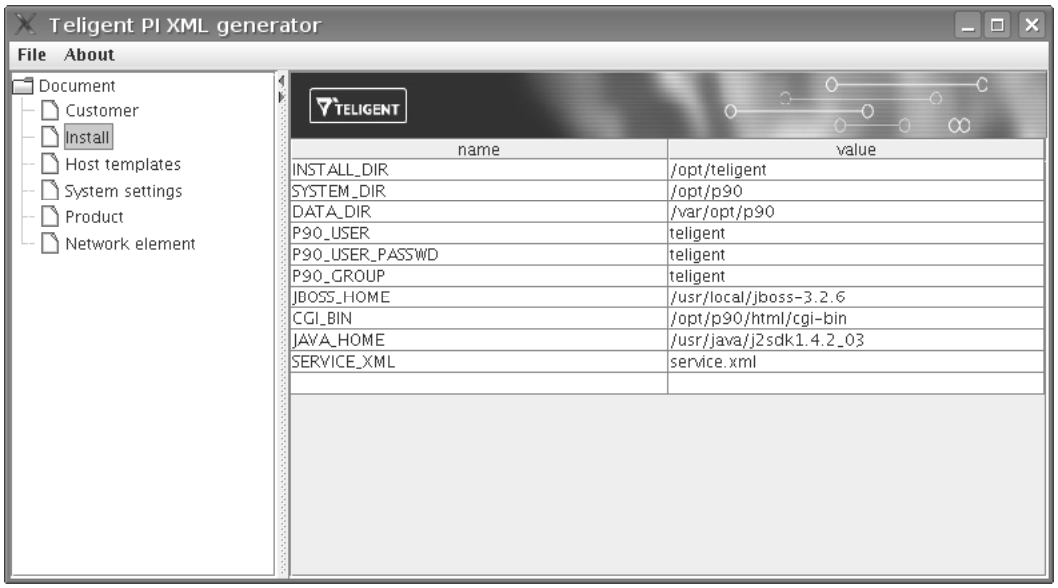
Figur 7.2: Klass diagram över XML-generatorn (del 1).



Figur 7.3: Klass diagram över XML-generatorn (del 2).

7.2 Användargränssnitt

Det grafiska gränssnittet använder vanliga Swing-komponenter för att visualisera data. Figur 7.4 visar hur det ser ut när användaren startar XML-generatorn. Till vänster

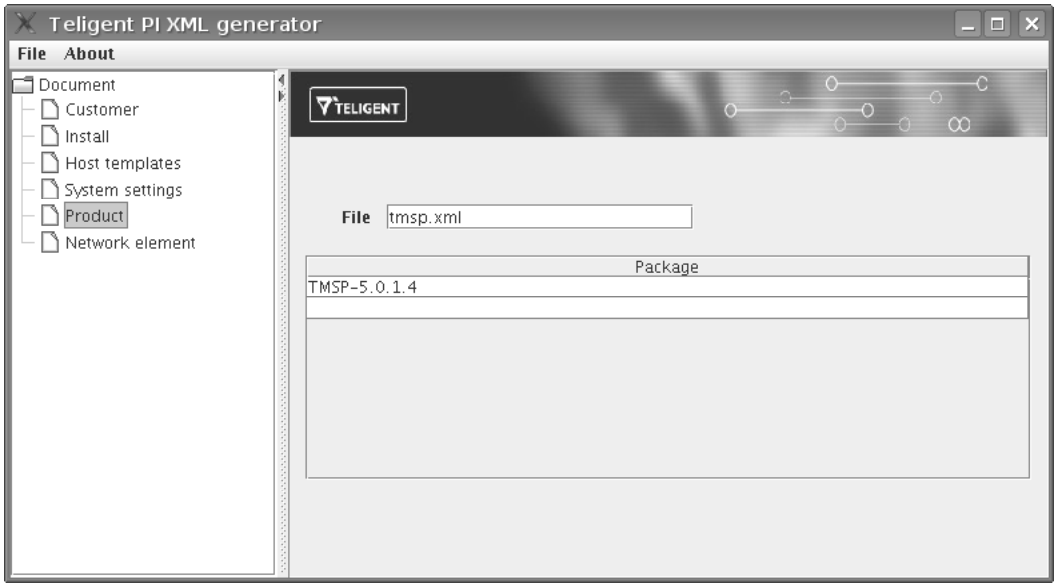


Figur 7.4: Installationsinformation och sökvägar.

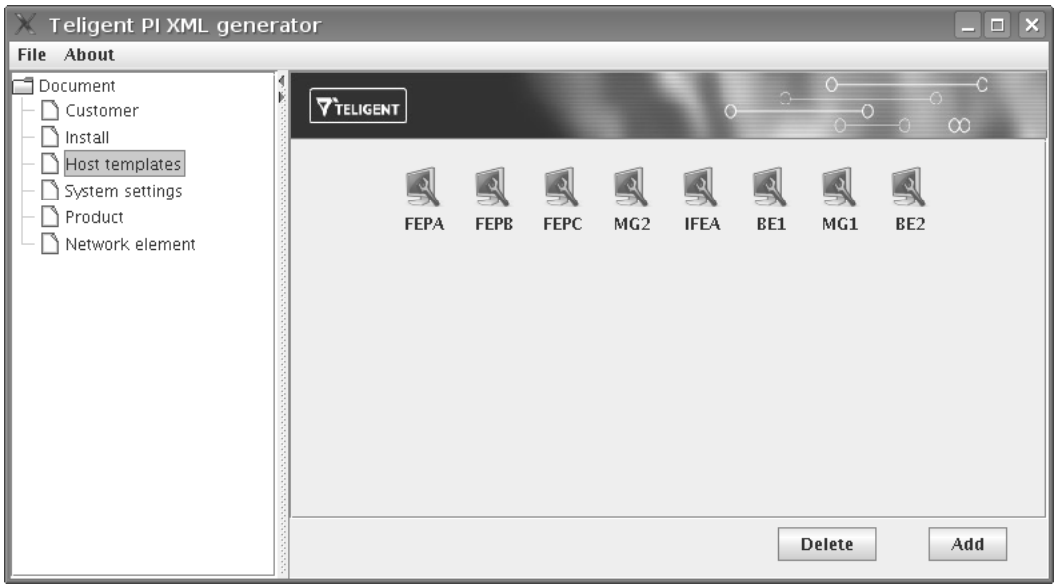
ligger alltid en JTree komponent som man använder för att snabbt traversera igenom XML-filen. Endast fasta element visas i denna komponent och det går inte att ändra eller lägga till någon information. Den högra delen används för att visa innehållet i elementen som finns i den vänstra delen. Vissa delar av trädet är ganska djupt och därför har jag använt dialogrutor för att det inte ska kännas rörigt. För att ladda och spara ett dokument används en vanlig meny som i sin tur öppnar en dialogruta där man får välja filnamn och sökväg. Figur 7.4 visar installationsinformationen där man kan ställa in miljövariabler, sökvägar och egna definitioner.

Om man önskar kan man använda en produktinformationsfil som är beskriven i PI:s användarmanual. Figur 7.5 visar hur denna information ska fyllas i. Denna information går även att lägga in i de templates som man önskar att använda, se figur 7.6.

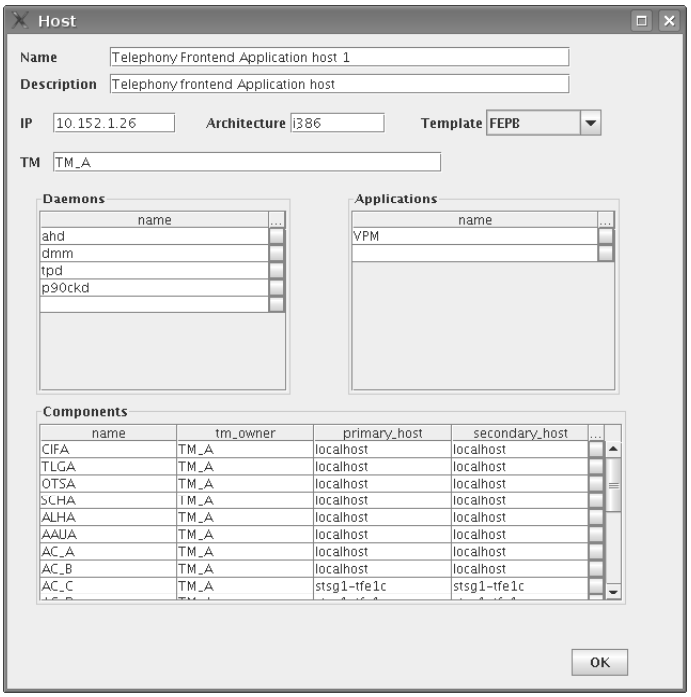
Figur 7.7 visar de komponenter som inngår i noden och gränssnittet för att lägga till och ta bort dessa. För att konfigurera dessa komponenter klickar man på knappen som ligger längst till höger på varje rad. Ett nytt fönster för konfiguration öppnas, se figur 7.8. För att konfigurera komponenterna hänvisar jag till manualen för dessa, där alla konfigurationsmöjligheter beskrivs.



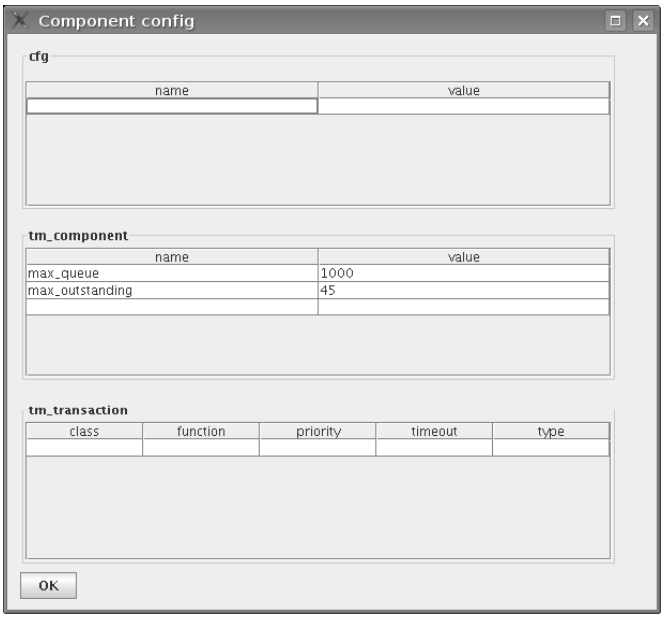
Figur 7.5: Information om produkt.



Figur 7.6: Befintliga templates med namn.



Figur 7.7: Visar de komponenter som ingår i noden.



Figur 7.8: Konfiguration av en komponent.

7.3 Bruksanvisning

7.3.1 Systemkrav

XML-generatorn kräver Java Runtime Environment version 1.5 eller nyare.

7.3.2 Installation/Exekvering

Hela programmet är paketerat i en körbar jar-fil som innehåller alla filer och paket som behövs för att köra XML-generatorn. Spara XML-generatorn på godtycklig plats i systemet. I Windowsmiljö startas XML-generatorn enklast med att dubbelklicka på jar-filen. Använder man Linux eller något annat system använder man kommandot: **java -jar filnamn**.
När XML-generatorn startas laddas automatiskt en templatefil som innehåller de viktigaste och de nödvändiga elementen i XML-filen.

7.3.3 Menyer

För att ladda och spara XML-filerna använder man menyn *File*. XML-generatorn har ett filter som gör att endast .xml-filer visas i alternativet Load. Detta går att ändra med drop down listan för filtyperna. Med dialogerna som visas vid *Load* och *Save* kan man välja filnamn och bläddra i kataloger för att ta fram den sökväg som erfordras.
Menyn *About* visar information om programmet och tillverkaren.

7.3.4 Innehåll

När XML-generatorn har startat visas en trädbaserad komponent längst till vänster. Denna del använder man för att välja vilken information man har för avsikt att redigera. Tabell 7.1 visar de alternativ som man kan välja för att fylla i informationen. För de

Customer	Information om kund.
Install	Miljövariabler och sökvägar.
Host Templates	Information om vilka paket som ska installeras i varje nod.
System settings	Inställning av tidsserver och konfiguration av noderna.
Product	Produktinformation.
Network elements	Information och konfiguration av noderna i systemet.

Tabell 7.1: Element som visas i JTree komponenten

värden och variabler som kan ändras i konfigurationsdelen hänvisar jag till dokumentationen för komponenten. Eftersom det inte finns någon semantisk kontroll på detta så är det viktigt att variabler och värden stavas rätt.

Network elements är den delen där konfigurationen av komponenterna utförs. Här ställer man in namnet, beskrivningen, IP adressen och arkitekturen. En template som har definierats i *Host templates* kan väljas och namnet på nodens TM (Transaction manager) kan väljas. Därefter kan demoner, applikationer och komponenter läggas till. Konfigurationsinställningarna för dessa lägger man till genom att trycka på knappen som ligger längst till höger i varje rad.

En komponent som betecknas LI har en speciell konfiguration för hårdvaran som den använder. Dessa inställningar gör man genom att trycka på knappen *Telephony* i

komponent konfigurationen. Här kan man ställa in vad det är för hårdvara man kör på, olika protokoll, timeslots med mera. För en utförligare beskrivning hänvisar jag till dokumentationen för LI komponenten.

Kapitel 8

Slutsats

Den implementation som gjordes för att enkelt generera en installationsfil till PI visade sig inte bli så användarvänlig som jag hade önskat. Detta beror på att det endast går att kontrollera syntaxen och inte själva semantiken för konfigurationen. XML-generatorn har ingen möjlighet att kontrollera varibelnamnet och datatypen, detta ligger på användarens ansvar. Att använda detta program utan att veta hur installationsfilen och komponenterna är uppbyggd är relativt svårt. Problemet med att semantiken för konfigurationen av komponenterna är okänd gör att användaren måste ha stor kännedom om komponenterna. Det som är positivt är att abstraktionsnivån blir något högre och man slipper tänka på syntaxen och kan istället koncentrera sig på innehållet.

En fråga som jag har ställt till mig själv är: Kommer detta att användas? Och svaret på detta blir: Det vet jag inte. Det flesta som använder PI dagligen kommer troligtvis fortsätta att ändra direkt i XML-filen för att det är det snabbaste sättet. Om man önskar att göra en förändring i den befintliga installationen så kommer detta verktyg att förenkla arbetet.

Den slutsats som som man kan dra av kravställningarna och förslagen är att man bör hålla fast vid den teknik som PI använder sig av. PI är ett bra verktyg och vill man förbättra det bör man endast lägga på ett grafiskt användargränssnitt. För att erhålla en enklare installationsmiljö måste man se till att konfigurationen av komponenterna kan verifieras när man skriver systembeskrivningen. Det är egentligen systembeskrivningen som PI använder som är det största problemet och det är här man måste lägga ner energi för att få till en enkel och bra lösning. Antalet konfigurationsmöjligheter är en annan sak som bör ses över och man bör lägga ner lite arbete för att ta fram några färdiga lösningar för att minimera konfigurationen.

8.1 Begränsningar

Eftersom XML-generatorn är beroende av vissa element i XML-filen så är man bunden att använda denna struktur på filen. PI använder till en viss del samma bindningar och då utgör denna begränsning inte så stora problem. XML-generatorn har ingen funktion för att kontrollera att en fil med rätt struktur laddas in. Programmet kommer att kasta en exception till standard ut när den påträffar ett allvarligt fel. Vissa komponenter använder speciella element för att styra sin konfiguration och det finns med största sannolikhet någon komponent där det behövs ett tillägg i XML-generatorn för att den ska kunna generera XML-koden för denna komponent. Denna information går att edi-

tera för hand och vill man sedan använda XML-generatorn går detta bra eftersom den är designad att inte förstöra befintliga element. XML-generatorn är endast testad under Java Runtime Environment 1.5, men eftersom den inte använder någon av de nya funktionerna så bör det fungera med en äldre version.

8.2 Framtida arbeten

För att göra ett installationsprogram som kan verifiera semantiken i komponentkonfigurationen måste man se till att alla komponenter har en fil som beskriver variabelnamnen och datatyperna för komponenten. XML-generatorn som implementerades i detta arbete kan därefter byggas ut så att den använder dessa filer. XML-generatorn bör testas mot alla komponenterna och och utökas i de fall där det finns undantag i konfigurationen, eller alternativt utforma komponenterna så att de arbetar på ett enhetligt och konsekvent sätt.

XML-generatorn kan utökas så att den genererar en distributionsfil med hjälp av ett befintligt skript. Det som behöver implementeras är en funktionalitet som sparar det redigerade DOM objektet till en XML-fil, därefter anropas det externa skriptet med XML-filen som parameter.

PI kan utökas så att den använder semantiken i komponenterna samt ett extra element som är en templatekonfiguration. Templatekonfigurationen är sedan tänkt att användas som attribut i själva konfigurationen. Detta gör att om några komponenter har en gemensam konfiguration så behöver man bara ändra på ett ställe.

Kapitel 9

Tack

Jag vill tacka alla som har bidragit till att detta arbete har kunnat genomföras. Personalen vid Teligents kontor i Umeå och de som har varit inblandade i utvecklingen av PI har gett mig mycket stöd. Stefan Johansson, min interna handledare, har varit till stor hjälp för själva rapportskrivandet. Denna rapport är skriven i LaTeX[10] med en mall som Jonas Birmé och Ola Ågren har utformat. Mallen var väl anpassad för detta arbete och sparade mycket tid.

Referenser

[1] M. Ali and M. Abrams. Simplifying Construction of MultiPlatform User Interfaces Using XML. In *Proceedings of the User Interface Markup Language Conference, Paris, France*, March 2001.

[2] B. Bederson, J. Grosjean, and J. Meyer. Toolkit design for interactive structured graphics. Computer Science Department, University of Maryland, College Park, MD., Tech Report HCIL2003 -01.

[3] P. Cederqvist. Version Management with CVS. http://ximbiot.com/cvs/manual, 2006-02-26.

[4] J. Clark and S. DeRose. Xml path language (xpath) version 1.0. http://www.w3.org/TR/xpath, 2006-02-26.

[5] B. Collins-Sussman, B. Fitzpatrick, and C. Pilato. Version Control with Subversion . http://svnbook.red-bean.com/index.en.html, 2006-02-26.

[6] E. Dubuis. Graphical user interfaces: Mess them up! In *Winter School of Computer Graphics 1996*, 1996.

[7] A. Hors, P. Hégaret, L. Wood, G. Nicol, J. Robie, M. Champion, and S. Byrne. Document Object Model (DOM) Level 3 Core Specification. http://www.w3.org/TR/DOM-Level-3-Core, 2006-02-26.

[8] M. Ivory and M. Hearst. The state of the art in automating usability evaluation of user interfaces. *ACM Comput. Surv.*, 33(4):470–516, 2001.

[9] P. Lay and S. Lüttringhaus-Kappel. Transforming XML schemas into Java Swing GUIs. In P. Dadam and M. Reichert, editors, *GI Jahrestagung (1), INFORMATIK 2004 - Informatik verbindet, Band 1, Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 20. September - 24. September 2004 in Ulm*, volume P-50 of *LNI*, pages 271–276. GI, 2004. ISBN 3-88579-379-2.

[10] T. Oetiker, H. Partl, I. Hyna, and E. Schlegl. The Not So Short Introduction to LATEX2. http://ctan.tug.org/tex-archive/info/lshort/english/lshort.pdf, 2006-02-26.

[11] J. Preece, Y. Rogers, and H. Sharp. *Interaction design, Beyond human-computer interaction* . J. Wiley, 2002. ISBN 0-471-49278-7.

[12] C. Wilding. Practical gui screen design: making it usable. In *CHI '98: CHI 98 conference summary on Human factors in computing systems*, pages 125–126, New York, NY, USA, 1998. ACM Press. ISBN 1-58113-028-7.

Bilaga A

Komponenter

A2S	Alarm to SNMP
AAU	Authentication and Authorisation of Users
ACC	Accounting. Not used in new projects.
ACT	Holds accounts (part of RTC)
AC_	AC_ (Application Controller)
ADA	Access screening Database Accesss
ADJ	Adjudication Process Mass calling
agentxlib	A C++ library implementing the AgentX protocol
ahd	Alarm Handler Database daemon
ALC	Application License Component
ALH	Alarm Handler Platform
ALI	Mobile
APM	PMM component and pmlib functionality to applications.
Appl_Builder	A graphical tool
ASM	Application Selection Matrix
ATC	Telia UMIT Authentication component
AVD	Application Variable Database
AVR	reserved Accelerated Variable Receiver
BCH	Broadcast Call Handler
BWL	reserved Black and White list
CAB	Works as a SOAP client towards Tele2 CABS server via XML/HTTP
CAI	reserved Call Attendant Interface
CAM	reserved Call Attendant Manager
CCM	ConCatenated Message component Mobile
CDA	Data access component for CName Mass calling
CDB	Data SQL Access (Oracle, MySQL, TimesTen or ODBC)
CHG	Handles charging sessions (part of RTC) Platform
CIF	The general media format conversion solution for media types.
CRA	Call Record Archiver
CRD	Call Record Dispatcher
CRR	Call Record Receiver
CRT	Call Record Translator
CTM	E-CTM (Ericsson Cellular Text Modem)
DCC	reserved Diameter Credit Control component
DDI	DS Data Interface
DFR	Distributed File Receiver
dmm	Data Mover Manager daemon
DMS	not supported Dial up over modem - SMS distribution
DRT	Data Router
DR_	Mobile
DTA	Data access component.
DTC	Data Type Converter
ELI	reserved Enhanced Line Interface
EMC	Email Composer
EMG	Email Get
EMI	SMS distributor, SMPP client over TCP/IP
EMP	reserved Eagle MSU Parser, triggerless ISUP
EMS	Email Send
FCC	File conversion component.
GAP	Intelligent Network Signalling interface
GMC	unknown Get Mail Component
GMS	Implements the SMS-G-MSC function in an SMSC

Tabell A.1: Komponenter a-g

GPR	reserved GPRS Manager (Charging Gateway)
gtilib	reserved Gordano/Teligent mail interface library
GTP	GPRS Tunnelling Protocol component
HRP	Host resource performance
HTC	HyperText Component
HTS	HTTP server and CGI
IAL	Isup Abstraction Layer.
IAP	Interface (to) Adjudication Processor
IAS	Information Access Server
ICG	IPS CDR Generator
IDA	Indirect Access / Access Screen Data Access (used in Centrica)
IDI	IAS Data Interface
IMB	unknown Hardware Monitoring via IMB
INF	INFormation gateway
INS	Intelligent Network Signalling interface
IPS	SIP proxy component
ISC	unknown Fixed
ISU	ISUP signalling component
JAWS	Java Work Station (Replaced by Workstation)
LC_	Locator for SMS routing and alert subscribtion.
LDC	LDAP client
LDI	LQS Data Interface
LIM	LI Modem
LI_	Line interface
M7M	MM7 component interacting with an MMSC.
M7S	MM7 component interacting with a service provider.
MAC	reserved Mail access component
MAP	The MAP component implements most of the GSM MAP protocol
MCA	Missed Called Alert component
MD3	not supported Main Data component.
MDI	Interface to handle database (TELE2)
MDS	Main data server.
MER	Message Retrieval
MGR	unknown The TFM Manager component
MM7	MM7 (MMS over SOAP) interface supporting SOAP and TPTF
MQS	MQ Series Interface
MRE	Main Rating Engine
MS_	Mailbox server Limited support for bugfixes etc
MUX	Multiplexer (Depreciated, use multiple wait in TSL instead)
NOS	Node Operational State component
OCC	OSA/Parlay Call Control
OC_	Java Object Container
ODA	reserved Mobile Office Data Access
ODB	ODBC component. Enables connection to Oracle/MySQL/TT databases.
ODI	OAS Data Interface
OSM	OpenVoice Service Matrix
OTS	Operator Terminal Server
OWS	deprecated Operator Workstation. (Replaced by Workstation)
p90bash	deprecated Bash interfaces (Replaced by p90cmd)
p90cmd	Command line tools for sending and receiving TPTF through the OTS.
p90ctlg	Transaction Log Presentation
p90e_dk_ss7	Intel (Datakinetics) stacks, firmware and drivers
p90e_isched	Intelligent Scheduler package
p90e_java_slee	Java Service Logic Creation Environment package
p90e_kernel	The required base package for any P90 installation

Tabell A.2: Komponenter g-p

p90e_prereq	Prerequisites for the p90e_kernel package
p90e_rtc	Real Time Charging package
p90e_sdk	P90/E component C/C++ SDK package
p90e_slee	Service Logic Creation Environment package
p90e_ss7_codecs	SS7 codecs C runtime library package
p90e_ss7_codecs++	SS7 codecs C++ library package
p90e_tsl_sdk	TSL compiler and helper script package
p90perl	deprecated Perl interface
PCC	P90 Component Controller
PIF	P90 Interface
PMM	Performance measurement management
PPH	Mobile
PPS	Prepaid component (Tesion)
PS_	Process supervisor
QR2	Quick rating engine 2
QRE	deprecated Quick Rating Engine (part of RTC)
QSI	Query Service Interface component.
RAC	deprecated RAC
RAM	Rating Manager (Charging Gateway)
RAP	Radius Authentication Protocol component
rc-P90	deprecated P90 prepare
RCS	Rate and Charge SOAP client.
RGC	Residential Gateway Controller, MGCP Call Agent component
RIP	reserved RUD (API) Interface Process
RMI	Remote Method Invocater
ROI	Rose Over ISDN
RST	Runtime Statistics
RSU	unknown RSU
RTD	deprecated In-memory Real Time database access.
RTS	Real Time Summariser
SCC	SCheduler Client. Transaction interface to Scheduler Manager.
SCH	Scheduler
SCR	SCCP Relay Component.
SC_	Subscriber cache (Charging Gateway)
SEC	Security component. Encryption/decryption and checksum calculation.
SFM	Signalling Front-End Manager.
SMC	reserved System Management Console
SMD	Short Message Data interface component
SMM	SMS Manager, manage and authenticate sms connections via db
SNA	SNMP Agent. SNMP interface.
snmp	SNMP daemon extension
SOP	Rate and Charge SOAP server. Used within Telecel MPP
SPA	SPA (Siemens Prepaid system Accessor).
SPM	SMPP Mass distributor, SMS Broadcast, filebased provisioning
SP_	SMS distributor, SMPP server over TCP/IP
SRF	Specialised Resource Function.
SSM	SS7 Manager
STS	reserved Swan To SMS
TAP	SMS distribution over TAP 2000
Tariff_Builder	Replacement for MRE.
TCP	reserved Generic TCP/IP Interface Component.
TDA	Tapping/Monitoring Data Access
TEM	reserved The Teligent Element Manager
TIA	reserved Triggerless Access Screening Component
TKC	unknown Turkcell component.

Tabell A.3: Komponenter p-t

TLG	Transaction Logger
TLU	unknown Used for filebased provisioning and CDRs TLU
TM_	Transaction Manager
TNK	Works as interface to Tele2 billing system TankaKontant.
tpd	Teligent P90 Daemon
TRT	Mobile
TSC	MGCP Trunking/Signalling gateway Controller
UCP	SMS distributor, UCP client over TCP/IP
UCS	reserved SMS distributor, UCP server over TCP/IP
UCX	not supported SMS distributor, UCP over X.25
UDA	reserved UM Data Access
UIM	deprecated User interaction manager. Adaptiware
VCC	Virtual Call Center Component
VCH C7	VPN Call Handler (C7)
VDA	VPN Data Access
VDC	Voice (and) Data Capture
VFP	VPN File Based Provisioning Component for Mobifon
VPM	Voice Prompt Manager
VPN	unknown VPN
VQM	reserved Virtual Queue Manager
VQR	reserved Virtual Queueing Room
VRD	Voice and Data Remote Download
Workstation	The Teligent P90/E system management tool.
XFG	deprecated Extended Configuration
XSR	Cross System Router
ZIA	ZT5087 cPCI chassis alarm monitor component.

Tabell A.4: Komponenter t-z