Avdelning för datavetenskap

Samiar Saldjoghi & Niclas Hanold

Utveckling av simulator för ärendehanteringssystem

Development of simulator for issue tracking system

Datavetenskap
C-uppsats 15hp

Datum/Termin:	2009-06-03
Handledare:	Simone Fischer-Hübner
Examinator:	Martin Blom
Löpnummer:	C2009:07

Karlstads universitet 651 88 Karlstad
Tfn 054-700 10 00 Fax 054-700 14 60
Information@kau.se www.kau.se

Utveckling av simulator för ärendehanteringssystem

Samiar Saldjoghi och Niclas Hanold

Denna rapport är skriven som en del av det arbete som krävs för att erhålla en kandidatexamen i datavetenskap. Allt material i denna rapport, vilket inte är mitt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

Godkänd, Date, Arkiv | Egenskaper | Eget

Handledare: Advisor, Arkiv | Egenskaper | Eget

Examinator: Examiner, Arkiv | Egenskaper | Eget

Sammanfattning

Denna rapport beskriver examensarbetet som vi gjorde åt Tieto i Karlstad. Tieto använder sig i nuläget av två ärendehanteringssystem, som kommunicerar via ett JMS- och XML-baserat gränssnitt. Målet med examensarbetet var att underlätta vid funktionstester genom att skapa ett program som kan simulera ett av ärendehanteringssystemen som används, vilket sparar tid då man behöver sätta upp en testmiljö.

Vår simulator består i huvudsak av tre delar:

- Grafiskt användargränssnitt
- Hantering av felrapporter
- Hantering av JMS-kö

Det grafiska användargränssnittet byggdes från grunden i Javas bibliotek Swing. Gränssnittet innehåller komponenter för att visa vilka felrapporter som hämtats, och en grafisk representation av en felrapport.

Vår applikation innehåller en modul för att hantera felrapporter och manipulering av dess data.

Simulatorn ansluts till en JMS-kö för att kunna hämta felrapporter från det andra ärendehanteringssystemet.

Development of simulator for issue tracking system

Abstract

This report describes the dissertation project we carried out for Tieto in Karlstad. At the moment Tieto uses two issue tracking systems, which communicates via a JMS and XML based interface. The aim of the thesis was to facilitate functional tests of the environment by creating a program that can simulate one of the issue tracking systems used. This saves time when a testing environment needs to be set up.

Our simulator consists mainly of three parts:

- Graphical user interface
- Trouble report handling
- Management of the JMS queue

We built the graphical user interface from the ground up using the Java Swing library. The interface contains components to display the trouble reports that have been collected, and a graphical representation of a trouble report.

Our application contains a module to manage trouble reports and the manipulation of its data. The simulator is connected to a JMS queue to get trouble reports from the other issue tracking system.

Innehållsförteckning

1 Inledning 1

2 Bakgrund 3

2.1 Ärendehanteringssystem 3

2.2 Trouble Reports (TRs) 3

2.3 Modification Handling (MH) 3

2.3.1 MH - Designunderhåll på Ericsson

2.3.1.1 CSR-TR Flödet

2.3.2 MH Databasen

2.3.3 Modification Handling Office (MHO)

2.3.4 MHWeb

2.3.4.1 GUI för MHWeb

2.3.5 CQ

2.3.5.1 CQ4E

2.3.5.2 GUI

3 Teknisk översikt 15

3.1 Enhetstester 15

3.2 JMS 17

3.3 REFLECTION 18

3.3.1 Kodexempel

3.4 XML 19

3.4.1 Syntax

3.5 ClearCase 21

3.5.1 Vy

3.6 Eclipse 21

3.6.1 Arkitektur

3.6.2 Rich Client Platform

3.7 Plugins 22

3.7.1 Jigloo

3.7.2 EclEmma

3.7.3 JUnit

3.7.4 Pulse

4 Uppdrag 27
4.1 Tieto 27
4.2 Problem 27
4.3 Syfte 28
4.4 Krav 29
4.4.1 Implementation
4.4.1.1 MHsim
4.4.1.2 CQsim

5 Implementation 31
5.1 Grafiskt användargränssnitt 31
5.2 TR-hantering 32
5.3 Meddelandehantering 34

6 User Guide 37
6.1 User Guide för CQSim 37

7 Resultat 40
7.1 Utvärdering 40
7.2 Testning 40

8 Slutsats 42
8.1 Nya kunskaper 42
8.1.1 Ärendehanteringssystem
8.1.2 JMS
8.2 Problem 42
8.3 Tidsåtgång 43
8.4 Framtida utveckling 43

9 Referenser 44

A Sammanfattning av förkortningar och begrepp 45

Figurförteckning

Figur 1: MH – Designunderhåll [7] 4
Figur 2: The MHDB, the global storage [7] 5
Figur 3: The MHO Concept [7] 6
Figur 4: MHWeb Startpage [8] 8
Figur 5: TrSearch [8] 9
Figur 6: Worklist [8] 9
Figur 7: AdminWeb [8] 10
Figur 8: ClearQuest [9] 11
Figur 9: CQ4E login [10] 12
Figur 10: CQ4E [10] 13
Figur 13: Eclipse [13] 22
Figur 14: Jigloo [14] 23
Figur 15: EclEmma [15] 24
Figur 17: Pulse [17] 26
Figur 18: Schematisk överblick av de nuvarande gränssnitten mellan MHWEB och CQ4E [19] 28
Figur 19: Schematisk överblick av systemen då simulatorn används. [19] 29

1 Inledning

Denna rapport beskriver examensarbetet som vi gjorde åt Tieto i Karlstad. På Tieto används idag två olika ärendehanteringssystem, MHWeb och CQ4E, vilka kommunicerar med varandra genom ett meddelandeorienterat middleware.

Vid utveckling av dessa ärendehanteringssystem krävs att de båda systemen körs för att man ska kunna utföra tester. Det skapar beroende av i vilken ordning de nya kraven implementeras i de olika systemen. Designers som arbetar med MHWeb är inte de samma som arbetar med CQ4E.

Målet med vårt examensarbete var att undvika dessa problem, genom att utveckla en simulator för CQ4E. Vi har inriktat oss på att simulera systemets hantering av felrapporter.

Vår simulator består i huvudsak av tre delar:

- Grafiskt användargränssnitt
- Hantering av felrapporter
- Hantering av JMS-kö

Inledningsvis i kapitel 2 ger vi en kort introduktion till ärendehanteringssystem. Vi går sedan in lite djupare på de två system som används på Tieto. I kapitel 3 beskriver vi tekniska aspekter kring det vi har arbetat med. Vidare i kapitel 4 berättar vi mer om uppdraget. Där berättar vi om Tieto och vilken roll ärendehanteringssystem har i företaget. Vi beskriver i kapitel 5 hur vi genomfört implementationen av vår applikation. I kapitel 6 hittar vi vår user guide som beskriver hur man använder simulatorn. Sedan redovisar vi projektets resultat i kapitel 7 och slutligen vår slutsats i kapitel 8.

2 Bakgrund

2.1 Ärendehanteringssystem

Ärendehanteringssystem är ett brett begrepp för system som stödjer ärenden i en organisation. I vårt exjobb har vi jobbat med Tietos ärendehanteringssystem, som innehåller funktionalitet för att hantera felrapporter, rättningar av fel och liknande.

2.2 Trouble Reports (TRs)

En Trouble Report är en rapportering av ett fel på en produkt, eller en produkts komponent. Felrapporten ska sedan analyseras och skickas till rätt instans för rättning. Under analysfasen kontrollerar man också att felet inte redan är rapporterat, eller om det är snarlikt ett tidigare fel som redan finns rättat. Man måste här också kontrollera att det som rapporterats verkligen är ett fel, och inte ett nytt krav från en användare som tycker att funktioner saknas. [7]

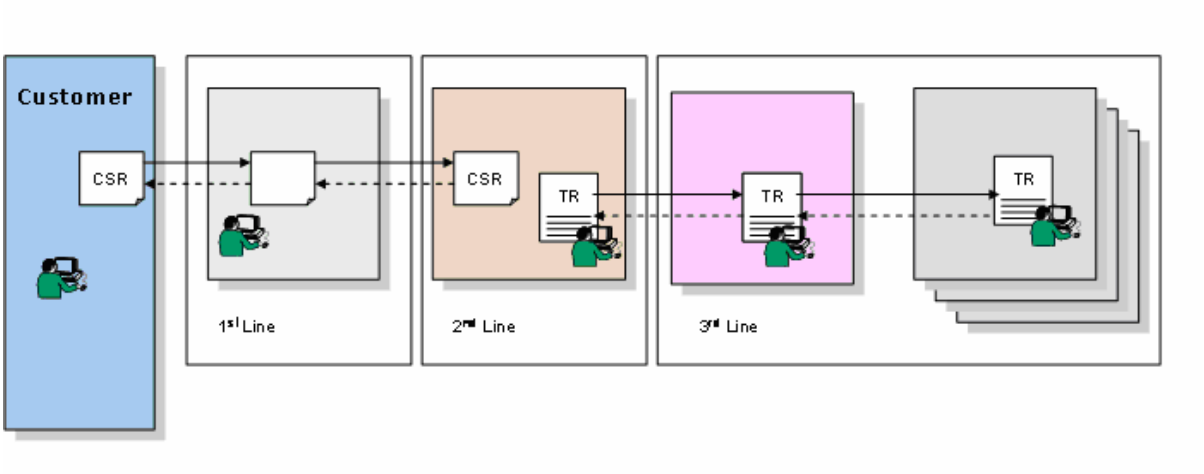
2.3 Modification Handling (MH)

Modification Handling är Tietos egenutvecklade ärendehanteringssystem som innehåller funktioner för att skapa felrapporter, söka på felrapporter och liknande. MH har ett webbaserat gränssnitt vid namn MHWeb, som är ett flitigt använt verktyg i organisationen.

Nedan följer en beskrivning av hur MH används i organisationen.

2.3.1 MH - Designunderhåll på Ericsson

2.3.1.1 CSR-TR Flödet

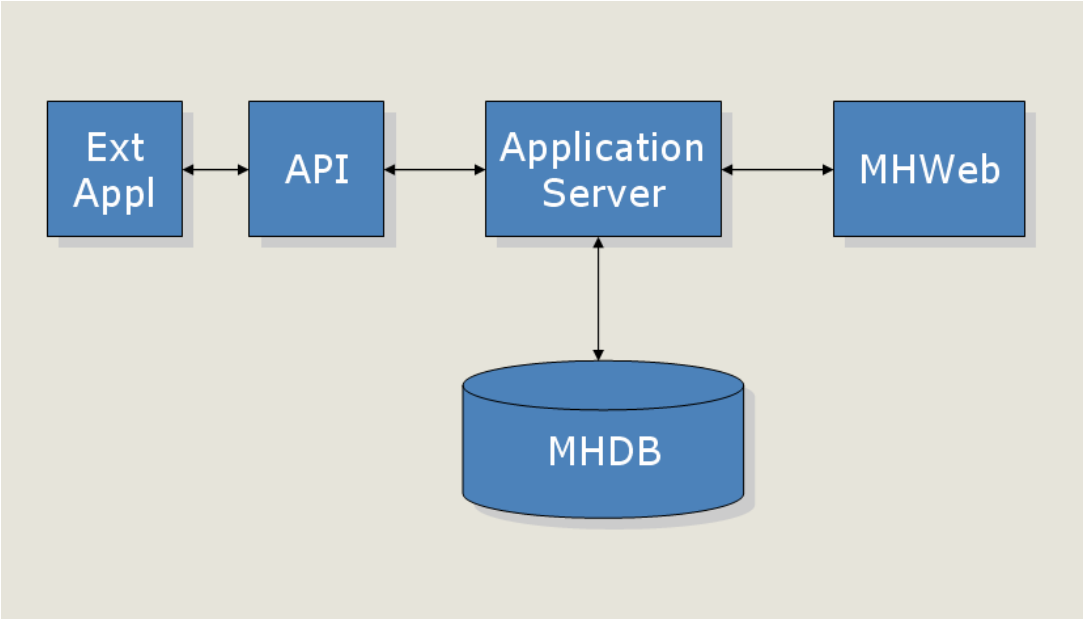


Figur 1: MH – Designunderhåll [7]

Beskrivning av ett exempel på flöde från kund till design.

1. Kunden registrerar en kundserviceförfrågan (CSR)
2. CSR tas emot av 1st line support. Om det är möjligt, så svaras den direkt. Problemet kan ha inträffat vid tidigare tillfälle och lösning kanske redan finns.
3. Om det inte går att hitta någon lösning till CSR, skickas den till 2nd line support.
4. Problemet analyseras. Om en lösning på problemet existerar, kan CSR svaras på.
5. Om ingen lösning kan hittas, registreras en TR och skickas till 3rd line support.
6. Om fortfarande ingen lösning kan hittas, skickas TR till det kontor som är ansvarig för denna typ av felrapporter.
7. En lösning på problemet skickas sedan till kunden.

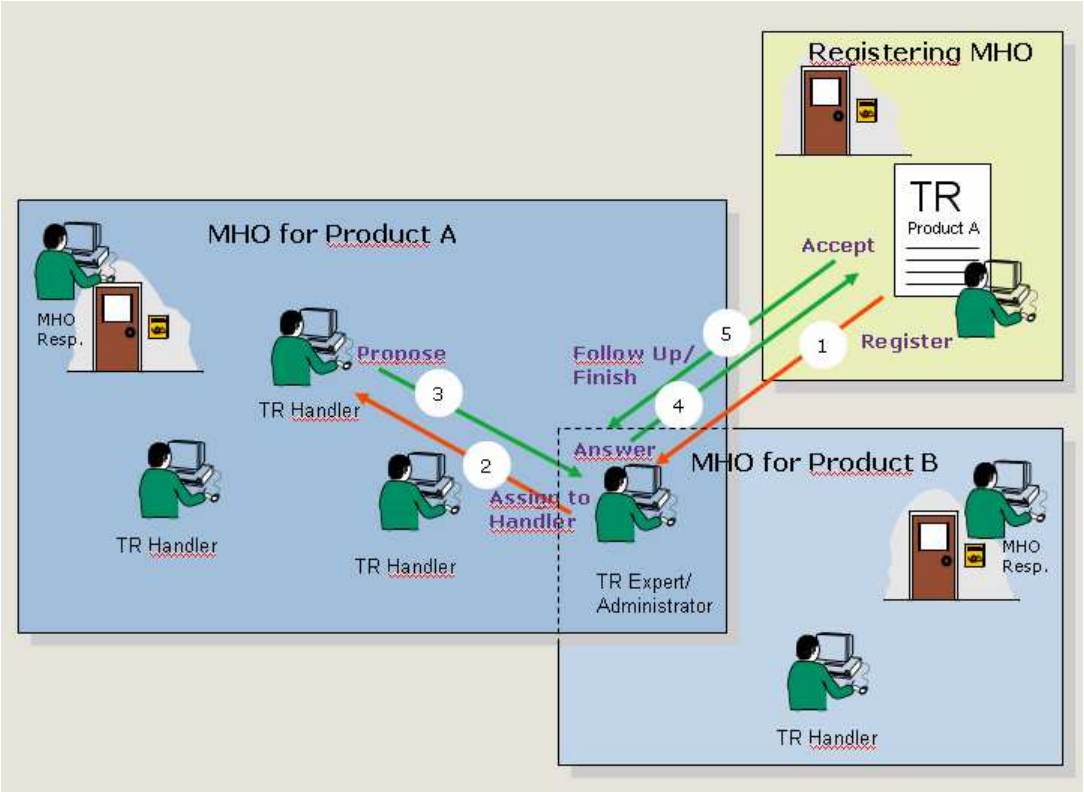
2.3.2 MH Databasen



Figur 2: The MHDB, the global storage [7]

MHDB är den globala databasen för TR och CR inom Ericsson. ClearQuest är ett exempel på externa system som använder sig av Application Server och MHDB.

2.3.3 Modification Handling Office (MHO)



Figur 3: The MHO Concept [7]

MHO är en arbetsgrupp som hanterar TR för en specifik produkt, område, projekt etc. En MHO kan tillhöra antingen CQ4E eller MHWeb i MHDB. En TR eller CR kan överföras mellan dessa MHO i ett specifikt TR eller CR flöde. En användare kan tillhöra en eller fler MHO och med olika rättigheter i varje MHO. Exempel: TR Handler i en MHO, TR Expert i den andra MHO.Man räknas som medlem i en MHO när man tilldelas någon av följande rättigheter: Handler, Expert eller Administrator. FIGUR 3 kan vara ett exempel på när TR för två produkter hanteras i två MHO. En teknisk samordnare (TR Expert/Administrator i båda MHO) tilldelar inkommande TR till en handler i aktuell MHO. Handler löser problemet och skickar ut lösningen. TR experten svarar sedan på TR. MHO Responsible – registrerar nya medlemmar tilldelar dem rättigheter etc i MHDB.

2.3.4 MHWeb

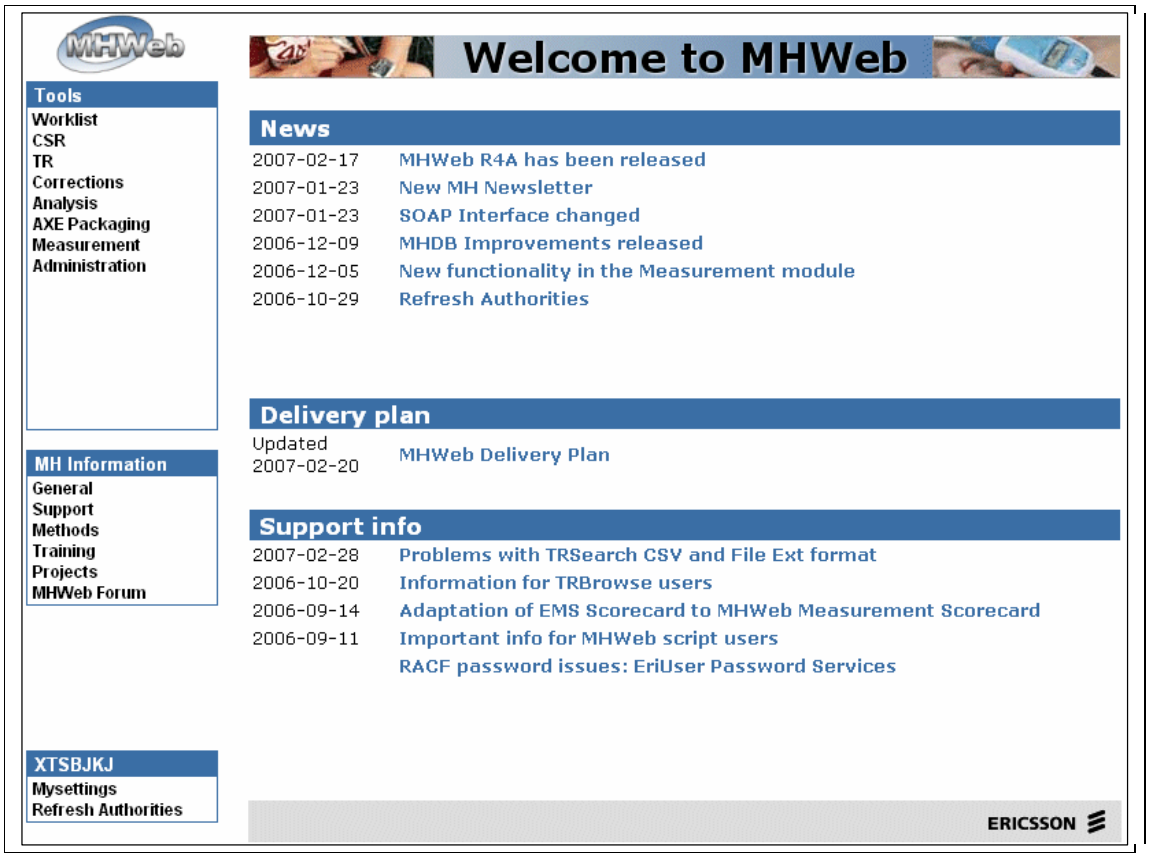
MHWeb är ett integrerat system på webben med komponenter som används i design-, designunderhålls- och supportorganisationer för underhålls- och kundsupportsändamål.

MHWeb erbjuder ett användarvänligt gränssnitt för hantering av TR, lösningar, analyser och mätningar. MHWeb erbjuder möjligheten att söka efter TR. Det erbjuder även olika möjligheter för analysering. MHWeb har även en TR-spårningskomponent för att förenkla spårningen av TR.

MHWeb innehåller även avancerade funktioner för statistik. Det finns också generell information, som riktlinjer och användarguider. [7]

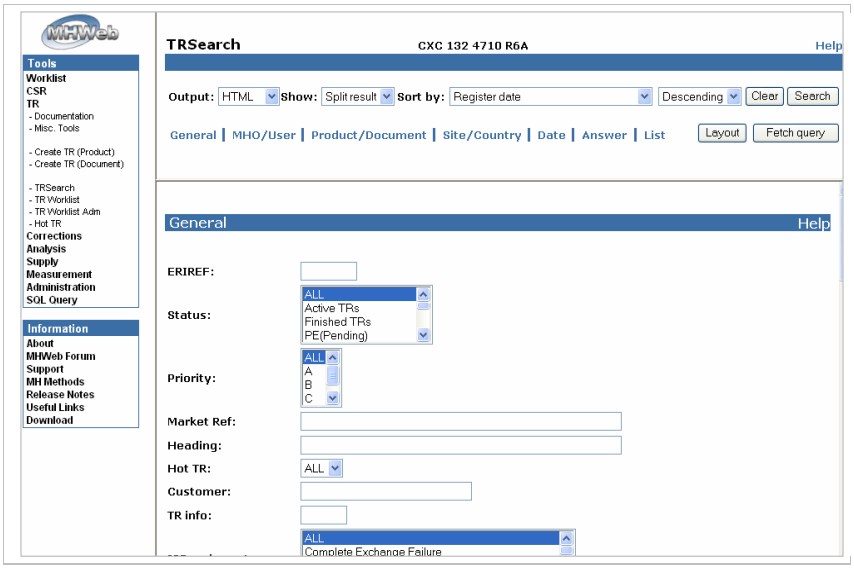
2.3.4.1 GUI för MHWeb

Se figur 4. Startsidan till MHWeb innehåller Nyheter och Leveransplan. Nyheter innehåller information om nya versioner och andra nyheter tillhörande MHWeb. Leveransplan är en beskrivning av kommande händelser. Ericsson-partners har begränsad åtkomst till MHWeb (partners har åtkomst till TR för vissa produkter).



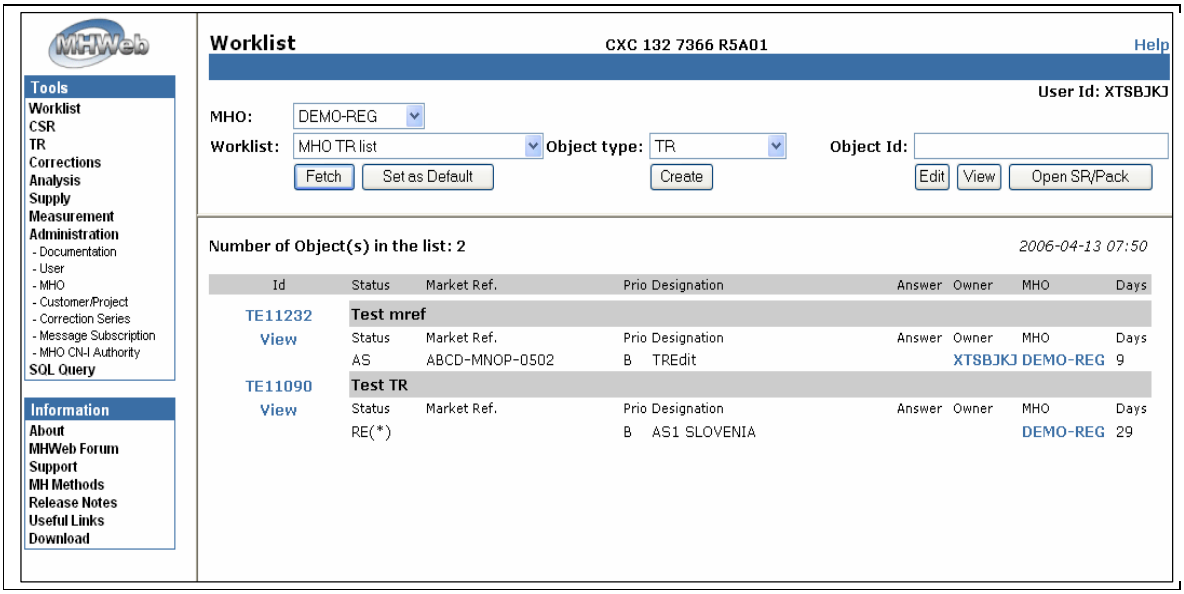
Figur 4: MHWeb Startpage [8]

Se figur 5. I TR Search får man möjligheten att söka efter trouble reports i MHDB. Sökkriterier kan sparas för senare bruk. Output kan sparas till en fil för fortsatt bearbetning. Output formaten FILE och FILE ext kan användas för att skapa en komma separerad fil. Filen kan då öppnas i MS Excel.



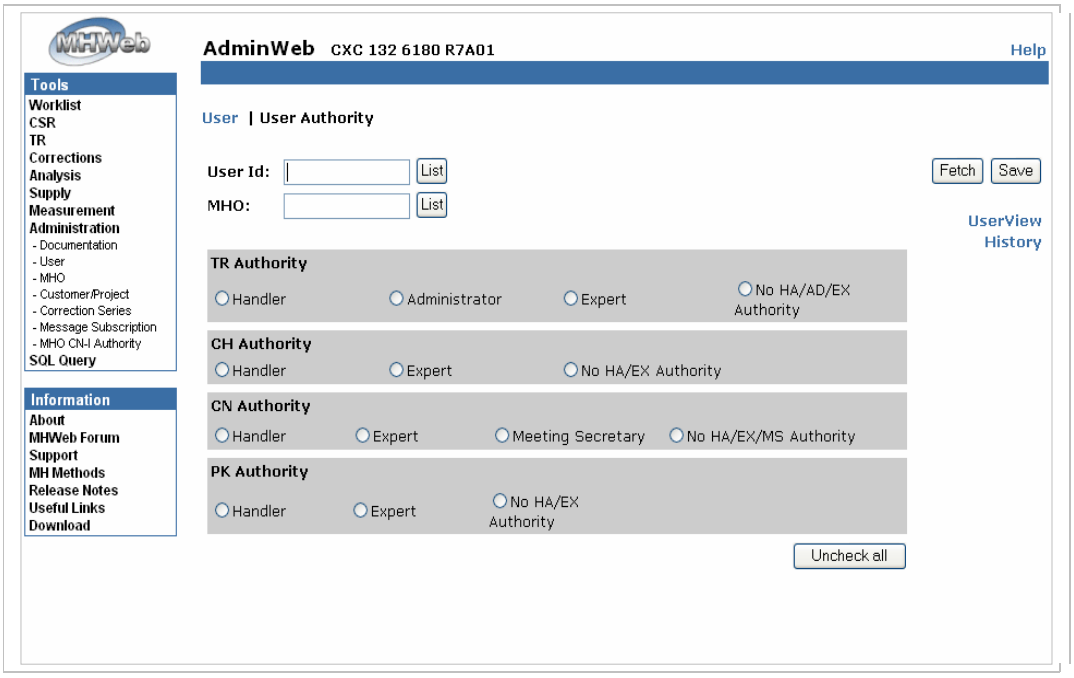
Figur 5: TrSearch [8]

Se figur 6. Worklist ger användaren överblick över TR i systemet. Own TR list visar TR som tillhör användaren. MHO TR list visar alla aktiva TR i användarens MHO.



Figur 6: Worklist [8]

Se figur 7. På AdminWeb sköts inställningar för rättigheter. Det utförs av personen som är ansvarig för MHO. Exempelvis kan ansvarig registrera ny användare, tilldela användare nya rättigheter, eller reducera användares rättigheter. Användare kan här skapa en prenumeration av TR, så att den inte missar när en ny TR publiceras.



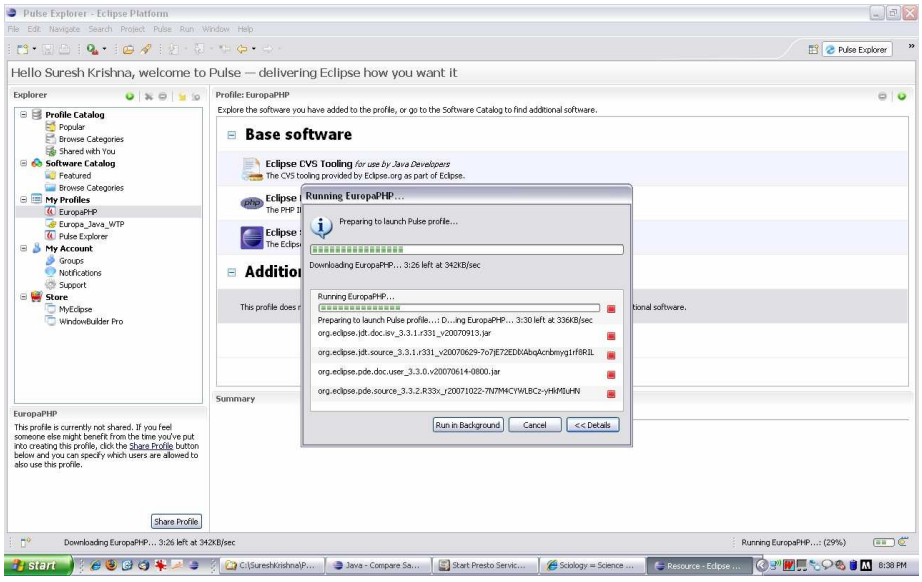
Figur 7: AdminWeb [8]

2.3.5 CQ

Rational ClearQuest (CQ) är ett verktyg för automatisering av arbetsflödet (särskilt felhanteringssystem) från Rational Software avdelningen på IBM. Verktyget kan anslutas till Microsoft Project.

ClearQuest var ursprungligen avsedd för att spåra fel (buggar) i projekt soom rör mjukvaruutveckling. Utöver förmågan att spåra buggar, tillåter ClearQuest organisationer goda organiseringsmöjligheter.

Det är utformat för att användas med andra Rational Software testverktyg, exempelvis Rational Performance Tester, Rational Functional Tester och Rational Manual Tester. ClearQuest erbjuder också en stabil miljö för kvalitetssäkring av programvara.



Figur 8: ClearQuest [9]

2.3.5.1 CQ4E

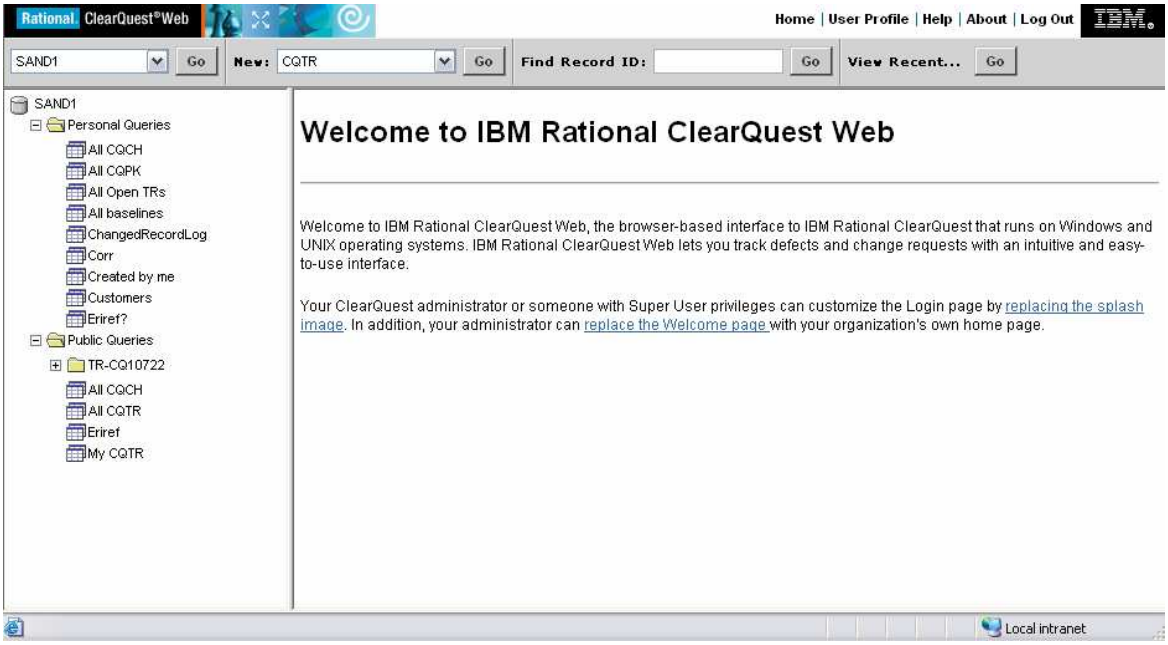
ClearQuest upptäcktes tidigt inte vara passande för Ericsson, då man har tydligt definierade processer som inte stöds i programvaran, varför man under en lång tid har vidareutvecklat verktyget med tusentals rader kod. Denna utbyggda version av ClearQuest, som påstås vara världens största, kallas CQ4E, ClearQuest for Ericsson. CQ4E finns i flera lokala instanser, till skillnad från MH som det endast finns en instans av.

2.3.5.2 GUI

Först får man logga in på CQ4E med de användaruppgifter som man har tilldelats.

Figur 9: CQ4E login [10]

Se figur 10. Därefter kommer man till startsidan. Här visas den valda databasen. Personliga och publika databasfrågor måste expanderas för att man ska kunna se innehåll. Publika frågor är tillgängliga för alla användare i samma databas.



Figur 10: CQ4E [10]

3 Teknisk översikt

3.1 Enhetstester

Mjukvarutestning är en empirisk, teknisk utredning för att förse intressenter med information om en mjukvaruprodukts kvalité. [3]

Enhetstestning är testning av den minsta testbara enheten i ett system. I ett objektorienterat system kan en metod, tillhörande en specifik klass, ses som en enhet. Enhetstestning kan ske genom att gå igenom koden med en debugger, men moderna enhetstester sker oftast med hjälp av ett testramverk. Dessa testramverk finns tillgängliga för de allra flesta stora utvecklingsspråk, och kan ofta integreras i en utvecklingsmiljö.

För att koncentrera testerna på en särskild enhet måste denna enhet ibland isoleras från resten av systemet. Detta är för att varje testad enhet ska vara så oberoende av andra enheter som möjligt. Ett vanligt misstag utvecklare gör är att man testar enheter som kommunicerar med en databas, vilket diskvalificerar testet som ett enhetstest. För att åstadkomma denna isolering kan man använda falska objekt som imiterar en viss del av systemet, exempelvis en databasanslutning.

Målet med enhetstestning är att isolera ett systems alla beståndsdelar och visa att var och en av dessa fungerar som specificerat. [4]

Enhetstesterna kan ses som en körbar specifikation över systemet. Allting som beskrivs i testfallen ska uppfyllas av systemet, och vid en körning av specifikationen valideras detta. Denna typ av ”levande” dokumentation har en större förmåga att följa systemet när nya funktioner introduceras, till skillnad från traditionell skriven dokumentation som utvecklare glömmer bort att uppdatera och därmed halkar efter. [5]

Enhetstester hjälper utvecklaren att refaktorera koden, dvs. bearbeta den så att den bättre följer kodkonventioner och/eller praxis. Utvecklaren kan då omedelbart efter det att han modifierat koden, köra igång testerna för att se att allting fortfarande fungerar.

Testdriven utveckling är en utvecklingsmetod där utvecklaren först skriver testfallet, och sedan skriver koden som testfallet testar. Då testfallet är skrivet och först körs, kommer detta test att misslyckas. Målet för utvecklaren är sedan att skriva minsta möjliga kod för att testet ska lyckas. När testet har lyckats bör utvecklaren refaktorera koden, innan han börjar om processen med att skriva ett nytt testfall. [6]

Testdriven utveckling är en av hörnstenarna i Extreme Programming, en utvecklingsmetodologi skapad av Kent Beck under ett av hans projekt i slutet av 1990-talet.

För att implementera ett enhetstest i ramverket xUnit, ett av de vanligaste ramverken för enhetstestning, skapar man en ny klass. Denna klass ska ofta, beroende på vilket språk man använder sig av, märkas som en testklass. Testklassen fyller man sedan med sina testfall. Ett testfall kan skrivas på följande vis i java:

```
public void testAdd()
{
            Money m12CHF = new Money(12, "CHF");
            Money m14CHF = new Money(14, "CHF");
            Money expected = new Money(26, "CHF");
            Money result = m12CHF.add(m14CHF);
            assertTrue(expected.equals(result));
}
```

Testfallet testar Moneyobjektets addmetod. Först skapas ett Moneyobjekt med värdet 12, sedan ett med värdet 14. På tredje raden skapas ett Moneyobjekt ”expected” med värdet 26. På nästa rad skapas en objektreferens ”result” som sätts till resultatet av summan av värdet för det första och andra moneyobjektet. På den femte och sista raden jämförs expected och result, och om de utvärderas till samma värde sägs testet vara med lycklig utgång.

Om testat lyckas kan man konstatera att addmetoden i klassen Money är korrekt, åtminstone för dessa inmatade värden.

3.2 JMS

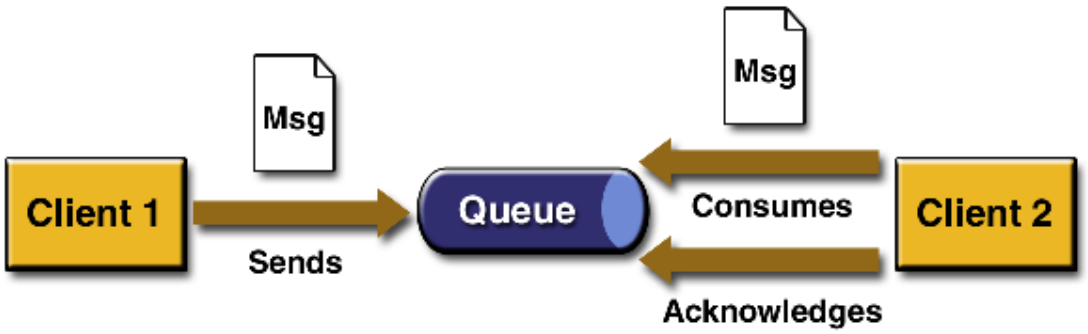
Java Message Service (JMS) är en standard för överföring av meddelanden som tillåter komponenter baserade på J2EE att skapa, skicka, ta emot och läsa meddelanden. JMS möjliggör distribuerad kommunikation som är löst kopplad, pålitlig och asynkron. **Fel! Hittar inte referenskälla.** JMS kan klassificeras som ett Meddelandeorienterat Middleware (MOM).

Meddelanden som skickas genom JMS, ofta genom att lägga meddelanden på en JMS-kö, är inte meddelanden riktade till människor, utan skickas från en applikation (eller komponent) till en annan applikation (eller komponent). Meddelanden skickas peer-to-peer, vilket betyder att alla inblandade parter ses som klienter. [2]

En JMS-implementation måste innehålla dessa tre delar:

- En Provider - det system som implementerar JMS.
- JMS-klienter - de javakomponenter som skickar och tar emot meddelanden.
- Meddelanden – varje applikation definierar en mängd meddelanden som används för att utbyta data mellan komponenter

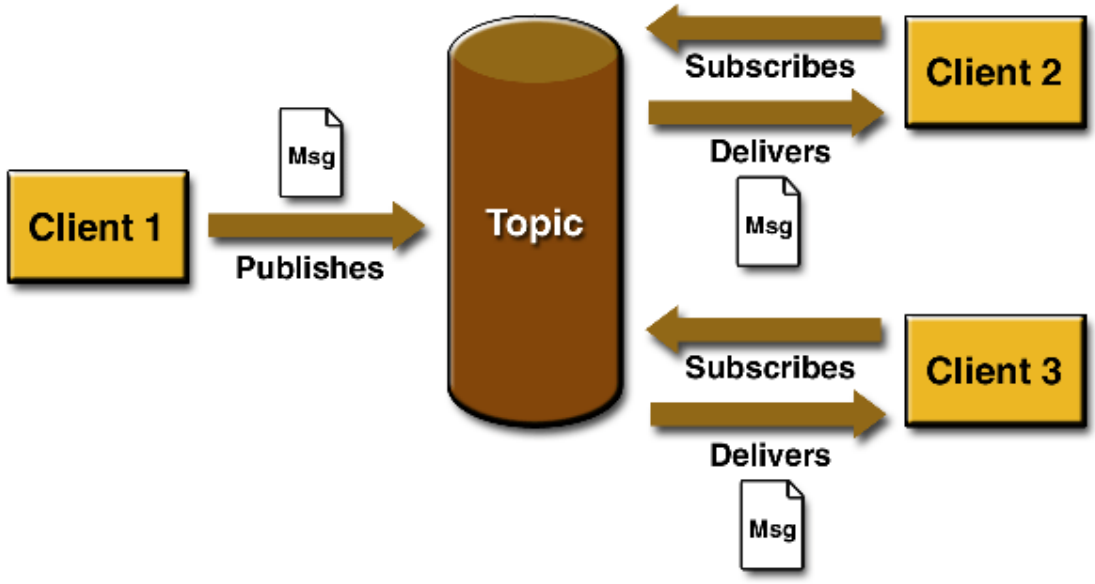
En JMS-applikation kan använda sig av ett av två olika sätt att skicka meddelanden på. Det första sättet, som vi använt oss av i vår applikation, är point-to-point-modellen. Modellen bygger på att en sändare skickar meddelanden till en specifik kö och en mottagare läser meddelanden från samma kö. Sändaren känner till meddelandets slutmål och skickar därför meddelandet direkt till mottagarens kö. Då mottagaren läser meddelanden från kön sägs det att denne konsumerar meddelandet, vilket innebär att meddelandet försvinner från kön i samma ögonblick som det hämtas.



Figur 11: Point-to-point [11]

Den andra modellen som kan användas är den som kallas ”publish and subscribe”. dvs. ”topic”

dvs. ett ämne. Prenumeranter tillåts registrera intresse för att ta emot meddelanden tillhörande ett specifikt meddelandeämne. Denna modell är lösare kopplad än först nämnda modell, då sändare och mottagare inte behöver känna till varandra.



Figur 12: Publish and subscribe [11]

3.3 REFLECTION

Reflection är en process där ett program kan observera och modifiera sin egen struktur. Det är en form av metaprogrammering.

Reflection kan användas till att observera och modifiera ett programs exekvering vid runtime. Ett programmeringsspråk som stödjer reflection möjliggör för ett antal funktioner vid runtime som annars vore omöjligt att uppnå i lågnivåspråk. [20]

Några exempel på dessa funktioner:

- Upptäcka och modifiera källkod
- Konvertera en sträng, till en referens av en klass eller metod som matchar strängen.
- Utvärdera en sträng som om den vore ett koduttryck

3.3.1 Kodexempel

// Utan reflection

```
Foo foo = new Foo();
foo.hello();

// Med reflection
Class cls = Class.forName("Foo");
Object foo = cls.newInstance();
Method method = cls.getMethod("hello", null);
method.invoke(foo, null);
```

Båda kodexempel skapar en instans av klassen Foo och kallar dennas hello() metod. Skillnaden är att I första kodexemplet är namnen till klassen och metoderna hård kodade; det är inte möjligt att använda en klass med annat namn. I andra kodexemplet kan namnen på klassen och metoden enkelt ändras vid runtime. Det negativa med andra exemplet är att den är svårare att läsa och ingår inte i kompilatorns syntax kontroll.

3.4 XML

XML (Extensible Markup Language) är ett universellt och anpassningsbart märkspråk. Det är klassificerat som ett utbyggbart språk, eftersom det ger användaren möjlighet att definiera element. Syftet med XML är att stödja informationssystem att dela strukturerad data, särskilt via Internet.

XML har en mängd verktyg som hjälper utvecklare att skapa webbsidor, men kan komma till nytta i många andra användningsområden. XML, i kombination med andra standarder, gör det möjligt att definiera innehållet i ett dokument separat från sin formatering, vilket gör det enkelt att återanvända innehållet i andra program. Dessutom ger XML ett grundläggande syntax som kan användas för att dela information mellan olika datorer, olika program och olika organisationer utan att behöva passera genom många olika omvandlingar. [21]

Ett XML-dokument har två nivåer av korrekthet:

- Well-formed:. Ett välformat dokument överensstämmer med XML-syntax regler, t.ex. om en start-tagg (<>) visas utan motsvarande slut-tagg (</>), är det inte giltig. Ett dokument som inte är välformat, räknas inte som XML.
- Valid: Ett giltigt dokument som dessutom överensstämmer med semantiska regler, antingen användar definierade eller i ett XML-schema. Om ett dokument innehåller en undefined element, så är det inte giltigt, men räknas som XML.

3.4.1 Syntax

Syntax för ett element:

<name attribute="value">content</name>

XML är dessutom skiftlägeskänsligt.

Man kan beskriva olika information genom att kombinera flera element i en hierarki.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<bilsamling>
   <bil ursprung="USA">
      <modell>Focus</modell>
      <tillverkare>Ford</tillverkare>
   </bil>
   <bil ursprung="Sverige">
      <modell>V70</modell>
      <tillverkare>Volvo</tillverkare>
  </bil>
</bilsamling>
```

Deklaration

Första raden i exemplet ovan är en XML-deklaration.

Element

Element är grunden i ett XML-dokument. Ett element består vanligtvis av en start- och en sluttagg, och kan ha olika attribut och innehåll. Så här kan ett komplett element se ut:

<modell>V70</modell>

Attribut

Förutom innehållet i ett element kan ett element även ha attribut. Ett attribut läggs till i starttaggen och har ett namn och ett värde.Så här kan användandet av attribut se ut:

<bil ursprung="USA">

Här har elementet bil fått attributet ursprung, med värdet USA.

Nästling

Element kan innehålla andra element:

```
<bil ursprung="USA">
    <modell>Focus</modell>
    <tillverkare>Ford</tillverkare>
</bil>
```

3.5 ClearCase

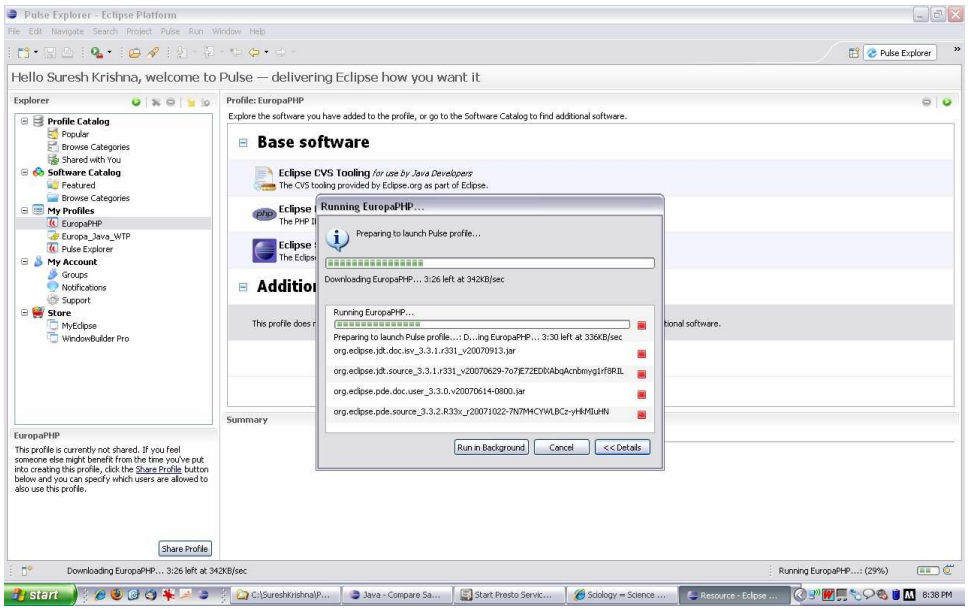
Rational ClearCase är ett program för revision (t.ex. configuration management, SCM) av källkod och mjukvaruutvecklings tillgångar. Den är utvecklad av Rational Software avdelningen på IBM. ClearCase utgör grunden för versionskontroll åt många stora och medelstora företag och kan hantera projekt med hundratals eller tusentals utvecklare.[22]

3.5.1 Vy

Objekt under versionskontroll i ClearCase sparas i arkiv som kallas VOBs (Versioned Object Base). En egenskap hos ClearCase är ett proprietärt nätverksuppbyggd filsystem (MVFS: MultiVersion File System), som kan användas för att montera VOBs som ett virtuellt filsystem genom en dynamisk vy. Den dynamiska vyn gör det möjligt att mappa till en Mjukvaru konfiguration.

3.6 Eclipse

Eclipse är en flerspråkig mjukvaruutvecklings plattform . Den är skriven främst i Java och används för att utveckla applikationer i detta språk. Eclipse tillåter olika plug-ins och möjligheten att tillämpa andra programeringsspråk.



Figur 13: Eclipse [13]

3.6.1 Arkitektur

Eclipse använder plugins för att tillhandahålla funktionalitet ovanpå (och inklusive) runtime systemet, i motsats till vissa andra applikationer där funktionaliteten vanligtvis är hårt kodade. Runtime systemet för Eclipse är baserad på Equinox, en implementation som uppfyller OSGi-standard.

3.6.2 Rich Client Platform

Eclipse tillhandahåller Eclipse Rich Client Platform (RCP) för att utveckla vanliga applikationer. Eclipse Rich Client Platform består av följande komponenter:

- Equinox OSGi – vanlig ramverk för bundling
- Core platform – startar igång Eclipse, kör plugins
- Standard Widget Toolkit (SWT) – en portable “Widget toolkit”
- JFace – fil buffert, text hanterare
- Eclipse Workbench – vyer, redigerare, perspektiv

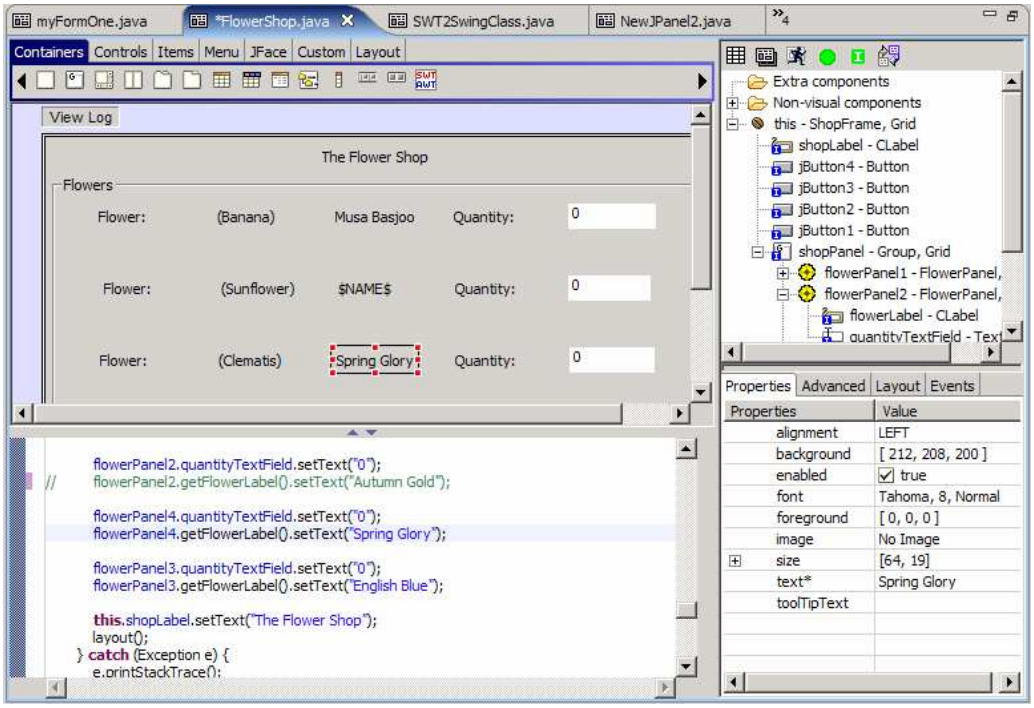
3.7 Plugins

Vi har använt oss av en del olika plugins som varit nyttiga I utvecklingen av projektet.

3.7.1 Jigloo

CloudGardens Jigloo GUI Builder är en plugin för Eclipse Java IDE och WebSphere Studio, som gör att du kan skapa och hantera både Swing och SWT GUI klasser.

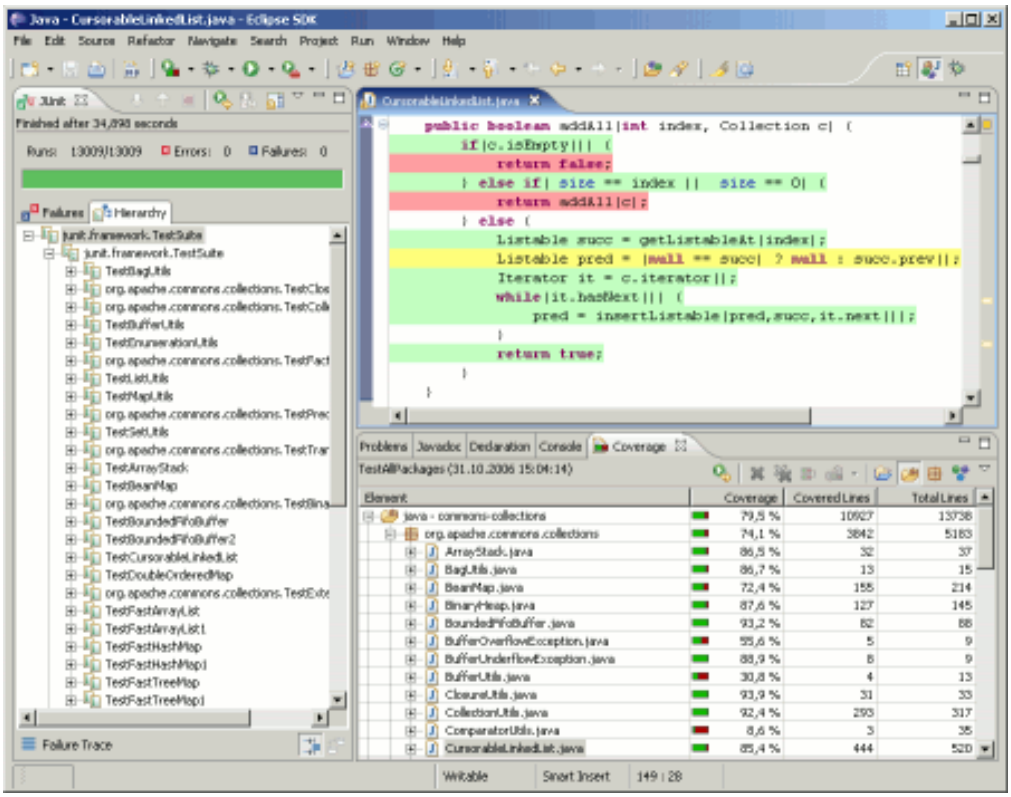
Jigloo skapar och tar hand om koden för alla delarna av Swing och SWT GUIn, samt kod för att hantera händelser. GUIn visas samtidigt som byggs upp.



Figur 14: Jigloo [14]

3.7.2 EclEmma

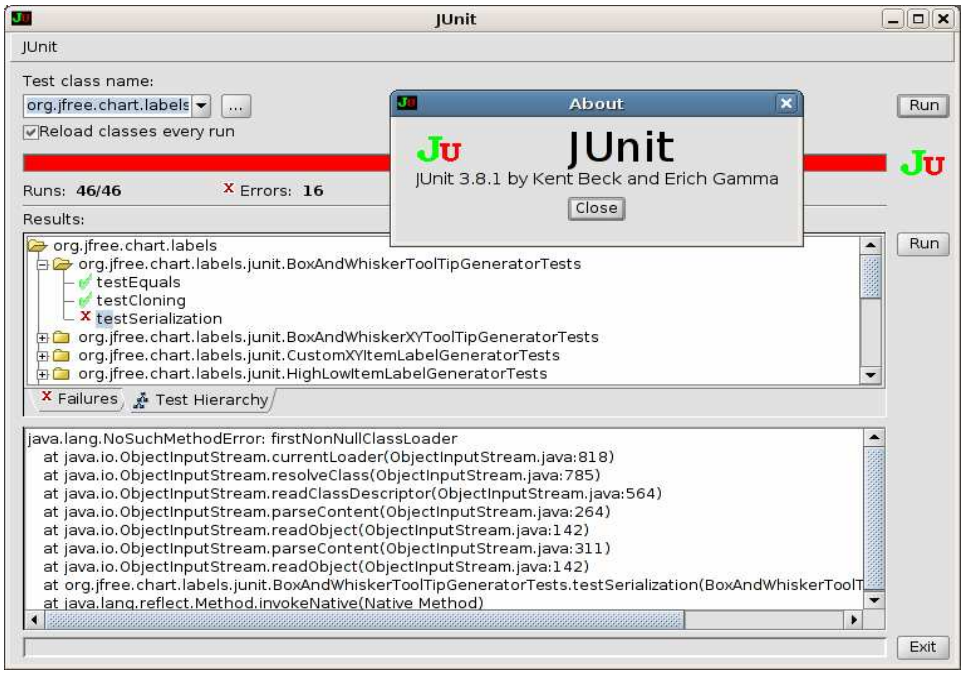
EclEmma är ett verktyg för kod täckning(code coverage) som mäter hur mycket av kodinnehållet som körs, vid igångkörningen av en applikation. Vanligtvis används det för att mäta hur mycket täckning en unit testing får, men det är inte begränsat till bara tester. Man kan lika gärna använda det för att ta reda på hur mycket kod som inte används I en GUI applikation.



Figur 15: EclEmma [15]

3.7.3 JUnit

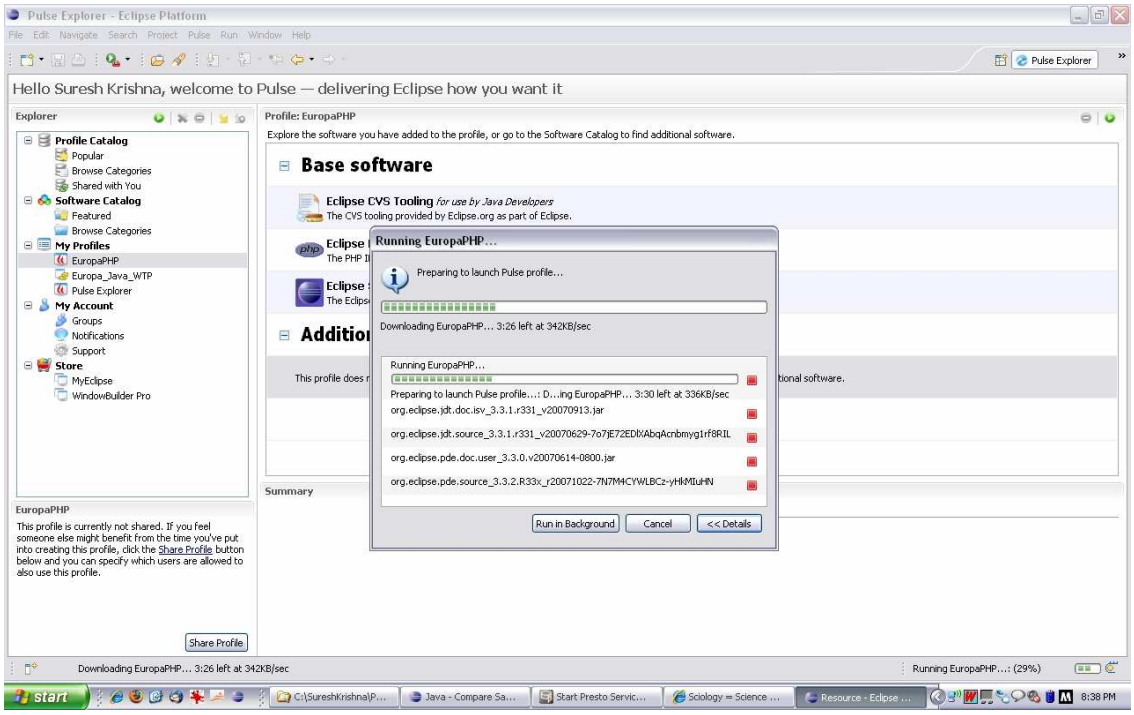
JUnit är ett simpelt ramverk för att skriva repeterbara tester. Det är en instans av xUnit arkitektur för enhetstestning ramverk.



Figur 16: JUnit [16]

3.7.4 Pulse

Pulse är en produkt som ger Eclipse användare en effektiv lösning för att hitta, installera och underhålla flera Eclipse programvaru profiler (inställningar). Det är gratis för privatpersoner.



Figur 17: Pulse [17]

4 Uppdrag

Kapitlet inleds med en kort bakgrund om företaget Tieto som vi gör vårt examensarbete hos. Därefter följer syftet med examensarbetet. Slutligen redogörs de problem och de mål som Tieto vill att vi ska uppnå under vårat examensarbete.

4.1 Tieto

Tieto är ett IT- företag som erbjuder tjänster som IT-, Forskning och utveckling- och konsulttjänster. Tieto har idag cirka 16000 anställda och är ett av de ledande IT-tjänsteföretagen i norra Europa och världsledande inom vissa segment.[18]

Tieto förvaltar och vidareutvecklar ärendehanteringssystem, som bland annat behandlar felrapporter, åt Ericsson. Då Ericsson är ett oerhört internationellt företag med kontor världen över ställs höga krav på deras ärendehantering. Mycket av komplexiteten ligger i att ta emot felrapporter från användaren av systemet, och sedan forsla den vidare till rätt kontor i världen, rätt avdelning på det kontoret, och slutligen till rätt person på avdelningen.

Tietos stora flaggskepp sedan många år tillbaka har varit produkten MH (Modification Handling), ett stort, helt egenutvecklat ärendehanteringssystem. Produkten används fortfarande och är basen i organisationen, men nyligen har man i Ericsson beslutat sig för att man i fortsättningen i så stor mån som möjligt använda sig av standardsystem. Detta beslut resulterade i att man började använda IBM-produkten ClearQuest.

Den 1 december 2008 bytte företaget namn från TietoEnator till Tieto.

4.2 Problem

Tools and Operational Excellence-sektionen på Tieto jobbar med ärendehanteringssystem. I nuläget finns två stora ärendehanteringssystem, Modification Handling och ClearQuest, som kommunicerar med varandra via ett JMS- och XML-baserat interface.

Exjobbet handlar om att utveckla en simulator som simulerar det ClearQuest-baserade systemet så att man kan köra funktionstester från det J2EE-baserade systemet utan att behöva sätta upp ClearQuest, vilket är en väldigt tidsödande aktivitet.

Om tid blir över ska även en Modification Handling-simulator utvecklas.

Applikationen skall utecklas i Java, skall kunna läsa och skriva från och till JMS-kö och skall kunna parsa XML för att visa den i en websida.

Utveckligsmiljö är Eclipse, Java 1.4 eller 1.6, Windows. Applikationer skall köras på Windows, Linux, Solaris.

4.3 Syfte

Vid utveckling av MHWEB och CQ4E krävs idag att de båda systemen körs, se Figur 11, för att man ska kunna utföra tester. Det skapar beroende av i vilken ordning de nya kraven implementeras i de olika systemen. Designers som arbetar med MHWEB är inte de samma som arbetar med CQ4E. För att undvika detta beroende under utvecklingsfasen krävs att man utvecklar en simulator.

Figur 18: Schematisk överblick av de nuvarande gränssnitten mellan MHWEB och CQ4E [19]

Syftet med examensarbetet är att utveckla en simulator som kan simulera de två systemen och därigenom göra det möjligt att testa ett system i taget. Se Figur 12.

Figur 19: Schematisk överblick av systemen då simulatorn används. [19]

4.4 Krav

Följande krav har vi fått av Tieto. [19]

Då simulatorn startas ska simulatorn koppla upp sig mot JMS-kön. Simulatorn börjar därefter polla kön och hämta alla meddelanden den hittar. Detta pollningsintervall ska vara konfigurerbart.

Därefter ska användargränssnittet startas upp och tillgängliga meddelanden skall visas i en lista. Möjlighet att visa mottagna eller skickade meddelanden ska finnas. Meddelande innehållande objekttyper som inte stöds ska kastas.

En ”Empty queue”-knapp ska finnas tillgänglig för att tömma kön. En ”Empty list”-knapp ska finnas tillgänglig för att kunna tömma listan. När användaren väljer ett meddelande visas detta upp som en lista av redigerbara fält. En ”Send”-knapp ska finnas tillgäng i användargränssnittet för att kunna skicka ett modifierat meddelande tillbaka till MH.

Skickade meddelanden skall sparas i ett bibliotek ”sent”. När ett meddelande skickas räknas meddelandets versionsnummer upp.

1. Simulatorn skall implementeras i Java, eftersom applikationen ska kunna köras på flera olika plattformar. Java är det vanligaste språket på vårt Tietokontor vilket gör att koden vi skriver blir lättare tillgänglig för dem i framtiden.
2. Simulatorn skall kunna konfigureras med en fil, för att t.ex. kunna byta JMS-kö utan att kompilera om programmet.

3. Data mellan CQ-simulatorn och MHWeb skickas som XML med hjälp av JMS-kö.

5. Kön skall pollas med ett konfigurerbart intervall.

6. Tillgängliga meddelanden skall visas grafiskt.

7. Det skall vara möjligt att välja vilket av de tillgängliga meddelanden som ska visas.

8. Utvalt meddelande skall presenteras grafiskt.

9. Meddelande i simulatorn skall presenteras i samma ordning som MHWeb gör, för att lättare kunna jämföra data mellan de två systemen.

10. Alla presenterade meddelande fält skall vara redigerbara.

11. Det skall vara möjligt att tömma listan av tillgängliga meddelanden.

12. Det skall finnas möjlighet att rensa köerna.

13. Det skall finnas möjlighet att skicka tillbaka ett meddelande med modifierad data till MHWeb/CQ4E.

14. Simulatorn skall ha stöd för Trouble Reports (TRs).

15. Simulatorn skall ha stöd för Corrections (CH).

16. Den grafiska presentationen sker med hjälp av HTML, Swing eller SWT.

17. Mottagna meddelanden verifieras med hjälp av XSLT stylesheet för respektive objekt och version.

18. Meddelande innehållande objekttyper som inte stöds kastas.

19. Unittester för den producerade koden skall finnas.

4.4.1 Implementation

4.4.1.1 MHsim

När CQ4E testas vill man inte behöva sätta upp IAS-applikationen som hör ihop med MHWeb. Därför är förslaget att ISJava-applikationen i CQ4E ändras så att det är konfigurerbart om JMS kö eller XML filer i filsystemet skall användas.

ISJava innehåller två applikationer, Send och Receive. Det skall vara möjligt att konfigurera dessa för att skicka/ta emot meddelande till/från JMS kö eller till/från XML filer i filsystemet.

4.4.1.2 CQsim

Ingen påverkan på befintliga applikationer behövs utan det räcker med att koppla CQsim mot befintlig JMS kö.

5 Implementation

5.1 Grafiskt användargränssnitt

Implementationsfasen inledde vi med att börja konstruktionen av vårt grafiska användargränssnitt. Vi hade enligt kravspecifikationen tre olika teknologier att välja bland, nämligen Swing, SWT och web. Vi uteslöt först att utveckla användargränssnittet med webteknologier, eftersom vi anser att en webapplikation innebär stora brister i användbarhet, visuell framtoning och ofta vad gäller svarstider.

Därmed stod vårt val mellan Swing och SWT. Swing är Suns egna GUI-verktyg, vilket vi antog skulle innebära en större användarbas, vilket i sin tur skulle innebära fler kodexempel och mer dokumentation att tillgå på Internet. Detta kan vara till stor hjälp vid eventuella problem. Vi hade dessutom haft kontakt med Swing förut, till skillnad från SWT. Valet föll därför på Swing.

När vi fick kravspecifikationen analyserade vi den och insåg att vårt användargränssnitt i stort sett har tre komponenter;

1. En lista bestående av inkomna TR
2. En lista bestående av redigerbara fält för vald TR
3. Knappar för att operera på TR och TR-listan, t.ex. ”Skicka TR”, ”Rensa Listan”.

Dessa de tre komponenter valde vi att disponera på följande vis:

Listan för inkomna TR behöver mycket utrymme på den vertikala ledden och därför gav vi den ett utrymme som löper från fönstrets topp och nästan ända ner till dess nedre kant. Vi tänkte oss att varje objekt i listan ska visas med namn och eventuellt datum vilket gör att den också kräver stort utrymme i horisontell ledd.

En felrapports fält visas i komponent 2 och även här behövdes mycket plats i vertikal ledd. För att utöka den vertikala kapaciteten tänkte vi här också att komponent 2 skall utrustas med en rullista.

Vad gäller placeringen av komponenten som skulle innehålla knapparna, ansåg vi att vi inte behövde precis hela nedre delen av fönstret. Därför valde vi att placera knapparna i vänstra nedre delen av fönstret, då vi inte ville kompromissa med utrymmet för komponent 2.

För att dela upp vår GUI i tre delar, skapade vi tre JPanels, där varje JPanel motsvarar punkterna ovan. De tre panelerna placerades sedan ut i vårt programfönster, på ett sätt som motsvarar figuren nedan. Vi valde det här upplägget dels för att det ska vara så användarvänligt som möjligt, dels för att det är förhållandevis simpelt att implementera.

Nästa steg var att skapa listan som visar vilka TR som lästs in. Vi valde att använda oss av en JList tillsammans med en händelselyssnare. Denna lyssnare lyssnar efter att användaren väljer något i listan, och avfyrar då en händelse.

5.2 TR-hantering

Då händelsen sker, läses vald TR in och visas upp i panel 2. I början av projektet läste vi här in TR direkt från filsystemet, men byttes senare ut mot läsning av en JMS-kö som specificerats som krav.

En TR representeras grafiskt med hjälp av en lista innehållande fält. Varje fält består av en titel, eller fältets namn, och ett redigerbart textfält, där fältets värde visas. Här stötte vi på problem vid implementationen, då vi inte visste vilka fält som skulle visas och i vilken ordning.

Enligt kravspecifikationen skulle vår simulator visa upp fälten för en vald TR i samma ordning som de visas på MHWeb. Detta visade sig vara problematiskt för oss, eftersom fältens titel i MHWeb inte alltid matchade fältens namn i TR-objektet. Detta gjorde att vi beslöt oss för att göra det så simpelt som möjligt för en användare att lägga till nya fält, ta bort fält eller modifiera ordningen på fälten.

Denna funktion implementerade vi genom att låta användaren ange i vilken ordning den vill presentera fälten. Detta anges i en textfil som vi kallar GuiOrder.txt. Användaren kan här också lägga till rubriker som ska visas i fönstret, för att i bättre mån kunna simulera utseendet i MHWeb.

- För att lägga till fältet ”heading” kan användaren skriva detsamma i textfilen, på en i övrigt tom rad.

- För att lägga till en huvudrubrik, exempelvis med texten ”Description” skrivs detta i GuiOrder som ”@Description”.

- För att lägga till en underrubrik skrivs ”$General”.

Vi har programmerat inläsningen på så vis att den första bokstaven i ordet bestämmer dess funktion. Exempelvis symboliserar ett inledande @-tecken huvudrubriker och bindestreck symboliserar en rad som inte läses in av programmet. En sådan rad kan användas för att lägga kommentarer.

När fältordningen är inläst ska vår simulator rita ut fälten och fältens värden på skärmen. För att hämta värdet för varje fält används en funktion som heter getValue(). Metoden tar två argument, TR-objektet samt namnet på fältet man vill hämta värdet från.

Värdet hämtas genom en metaprogrammeringsteknik som kallas reflection. Tekniken möjliggör att objekt kan genom introspektion upptäcka vilka metoder den tillhandahåller. Det är på det här sättet man kan skicka in ett fält tillsammans med TR-objektet och som utdata få värdet från inskickat fält. För att åstadkomma detta letar objektet efter den metod som motsvarar fältets namn. Vår kod är skriven så att fält alltid anges i gemena bokstäver, medan TR-objektets metoder innehåller både gemener och versaler. Detta har vi löst genom att implementera två metoder, lookupGetMethod samt lookupSetMethod. Metoderna används för att leta upp korrekt get- respektive setmetod som motsvarar det fält som skickas som inparameter. I metoden konstrueras sedan en sträng, methodName, som består av strängen ”set” ihop med fältets namn. Eftersom metoden kan innehålla versaler konverteras den först till gemener. De två strängarna kan nu jämföras och om två är identiska returneras metodobjektet.

```
public Method lookupSetMethod(Class c, String field)
{
            String methodName = "set" + field;
            for(Method method : c.getMethods())
            {
                        if(method.getName().toLowerCase().equals(methodName))
                        {
                                    return method;
                        }
            }
            return null;
}
```

Detta metodobjekt används sedan för att anropa metoden som objektet representerar. Metoden returnerar ett värde, precis som vid ett normalt anrop till en getmetod. Värdet returneras sedan från metoden getValue.

```
Method m = lookupGetMethod(trInfoClass, field);
Object value = m.invoke(trInfo);
return value.toString();
```

5.3 Meddelandehantering

ClearQuest och MH kommunicerar genom att skicka meddelanden till varandra via Java Message Service (JMS). Ett meddelande består av en header, som specificerar information om meddelandet, samt i vårt fall en TR. TR är, då den skickas som meddelande mellan CQ och MH, serialiserad till en XML-fil.

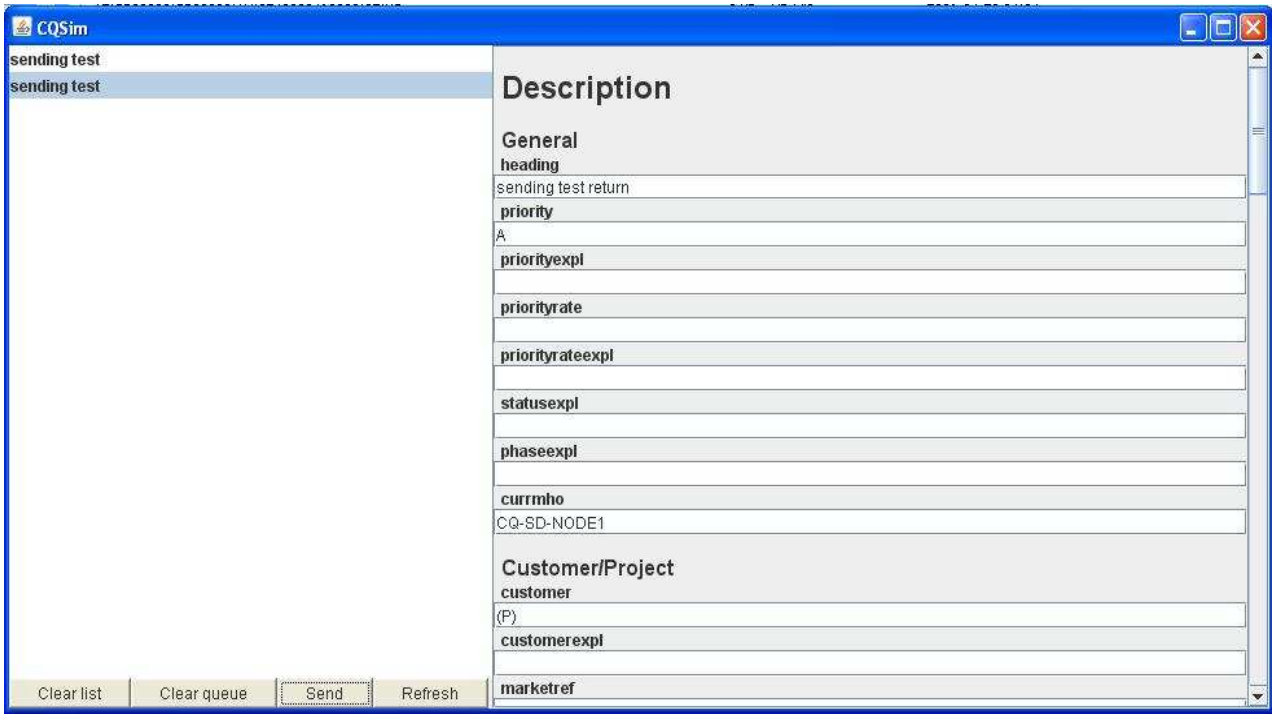
Vi började med att implementera hämtning av meddelanden från MH:s utkö. TR hamnar på kön som ett meddelande då användaren väljer att skicka en TR från MH till CQ. För att vår simulator ska kunna hämta meddelanden från rätt kö måste den veta JMS-serverns adress. Köns namn måste också vara känt, eftersom det kan finnas flera köer på en server. Vi valde att lägga dessa uppgifter i en separat propertiesfil, så att användaren själv kan specificera vilken JMS-kö som ska användas, utan att användaren behöver skriva om källkoden och kompilera om.

För att ansluta sig till kön använder vi oss av Javas JMS-API. Här finns många färdiga klasser vi använt oss av, till exempel klassen QueueReceiver, som innehåller en metod för att hämta nästa meddelande på kön. API:et definierar också en klass TextMessage, som representerar ett textmeddelande som kan skickas över kön. Denna klass används vid kommunikation mellan CQ och MH. Dess metod getText returnerar texten som ska levereras. I vårt fall är texten ett TR-objekt serialiserat till en XML-sträng.

Då vi skickar meddelanden tillbaka till MH, initierar vi först ett objekt av klassen QueueSender. Vårt nästa steg är att skapa meddelandet som ska skickas, med rätt värden i headern. Till exempel: för att vår TR ska skickas tillbaka till MH och uppdatera befintlig TR, krävs att egenskapen Action sätts till FULL_UPDATE. När meddelandet är skickat från simulatorn till MH har den uppdaterats med de eventuella ändringarna som gjort sedan den skickades från MH till simulatorn.

6 User Guide

Utöver programmeringen av simulatorn fick vi i uppdrag att skriva en användarguide för vår simulator. Guider av den här typen skrivs så att användaren ska kunna ta de första stegen in i användandet av programmet, och är ingen heltäckande funktionalitetsbeskrivning. Den är skriven på engelska, då Tieto är ett väldigt internationellt företag.



6.1 User Guide för CQSim

Figur 20: CQSim

CQSim is simulating ClearQuest by

- Receiving TRs routed from MHWeb
- Displaying chosen fields of the TR
- Allowing the user to edit the contents of those fields
- Allowing the user to send the TR back to MH

On startup, CQSim will receive all messages on the JMS queue specified in the properties file.
If TRs are added during runtime, the user may press the refresh button to receive TRs recently added to the queue.

The TRs are displayed in a list on the left hand side of the window. When selected, the TR will be displayed on the right hand side. The values may now be edited by the user.

The user may change which fields are visible by manipulating the GuiOrder file. Fields are written as they appear in the TRInfo class, but with lower case characters only.

For example: to add an editable field in the GUI for the TRInfo method “getCountry()”, this should be written as “country” on a new row in the GuiOrder file.

To add headings, put @ before the heading name for a big heading (Description in the picture) or $ for a small heading (General and Customer/Project in the picture). Lines starting with – are not read by the program (may be used as a comment row).

To send the TR back to MH simply press the “Send” button.

To clear the list of available TRs press “Clear list”.

To clear the queue, press “Clear queue”.

7 Resultat

Rapporten behandlar simulator projektet som vi gjorde åt Tools and Operational Excellence-sektionen på Tieto i Karlstad.

7.1 Utvärdering

Kravet vi fick var att skapa CQ simulatorn skulle hämta Trouble Reports från Ericssons MH system. Om vi fick tid över, skulle vi även skapa en MH simulator som hämtar Trouble Reports från CQ. Då det var så mycket med att skapa CQsim , fick vi inte tid över till att skapa MHSim.

Något som vi inte lyckades få fullständigt är några utav fälten i simulatorn. Vi skulle hitta matchande fält från MHWeb och en XML fil. Problemet var att vissa av fälten kunde ha helt olika namn, vilket gjorde att vi var tvungna att testa oss fram på en del av dom. Andra gick inte att testa sig fram då vi inte hade möjligheten att använda oss av dom fälten i MHWeb.

I kraven står specificerat att simulatorn skall ha stöd för Corrections, vilket vi inte hade tid att implementera. Vår applikation är skriven på ett sånt sätt att Corrections kan implementeras, med minimal ändring av koden. Förutom ändring av koden behövs också enhetstester för Corrections, vilket kan göra att det ändå kan dra ut lite på tiden.

I kraven specificeras också att mottagna XML-meddelanden skall valideras med hjälp av en XSL-transformering. Vi implementerade en metod för att transformera en XML-fil med hjälp av given XSL-fil, men utdatat blev inte som väntat. Vår handledare på Tieto misstänkte att vi hade fel XSL-fil. Vi lade sedan detta krav åt sidan för att koncentrera oss mer på JMS-hanteringen.

7.2 Testning

För att testa och validera systemet har vi kontinuerligt använt oss av enhetstestning. Vi försökte hela tiden under utvecklingsfasen att skriva åtminstone ett testfall för varje metod, och i vissa fall skrev vi fler testfall för en metod för att verkligen försäkra oss om att metoden

gör vad den var designad att göra. Vi har antagit att systemet som helhet fungerar om alla dess delar fungerar.

8 Slutsats

Vi fick en ganska stor uppgift som berörde en hel de områden som var absolut nya för oss. Det tog ett tag innan vi kom igång med arbetet ordentligt, då vi fick lära oss det mesta på egen hand och genom att testa oss framåt. Vi blev tvungna att göra om mycket, för att en del saker som vi jobbat med inte skulle fungera längre fram projektet utan att vi kunde veta om det.

8.1 Nya kunskaper

Vi har fått många nya lärdomar då vi jobbat med flera saker som vi inte tidigare har jobbat med.

8.1.1 Ärendehanteringssystem

När vi började med projektet hade vi ingen aning om vad ett ärendehanteringssystem var för något. Nu har vi både en hel del kunskaper inom området och även hur man använder sig utav det.

8.1.2 JMS

Innan projektet hade vi aldrig jobbat med meddelande orienterad middleware. Nu har vi god kunskap om hur man upprättar en meddelandekö och hur man skickar meddelanden över den.

8.2 Problem

Vi har stött på några problem under projektets gång. De har varit både programmeringsrelaterade och administrativa. Bland de programmeringsrelaterade problemen kan nämnas åtkomst av vissa attribut i TR objektet med hjälp av reflection. Då attributen är uppbyggda liknande en trädstruktur och det var ganska komplicerat att få reflection metoderna att komma åt alla de olika nivåerna.

Bland de administrativa problemen kan vi nämna att med att MHWeb har legat nere vid några tillfällen, vilket gjort att vi inte haft möjlighet testa simulatorn, då det inte går att skapa några TR.

8.3 Tidsåtgång

I början av projektet fick vi lägga tid på att sätta oss in i och få god förståelse för hur ärendehanteringssystem fungerar. Detta gjorde att vi inte kunde sätta igång med implementationen så tidigt som vi hade hoppats. Därefter kunde vi fokusera mer på programmeringen.

8.4 Framtida utveckling

Vi har skapat ett program som mycket väl går att bygga ut på. Tanken från början var att bygga en god grund för fortsatt utveckling, vilket vi har uppnått. Följande är exempel på några funktionaliteter som kan tänkas implementeras.

Först och främst är det MH-simulatorn som kan vara nyttig att ha. Med hjälp av källkoden som vi har skrivit, anser vi att det skulle gå snabbt att skapa en MH-simulator.

En annan funktion som kan tänkas implementeras är en översättningsfunktion för en felrapports fält. I nuläget skrivs fälten ut i användargränssnittet med samma namn som i XML-filen. Detta gör att fältet i många fall är svårtolkat för användaren, om den inte är djupt insatt i den underliggande strukturen. Denna funktion finns i MHWeb och skulle förmodligen kunna implementeras med hjälp av redan existerande kod.

9 Referenser

[1] http://java.sun.com/products/jms/, 2009-05-25

[2] Sun Microsystems, Java Message Service Specification Final Release 1.1 http://java.sun.com/products/jms/docs.html, 2009-05-25

[3] Exploratory Testing, Cem Kaner, Florida Institute of Technology, *Quality Assurance Institute Worldwide Annual Software Testing Conference*, Orlando, FL, November 2006

[4] Automated Defect Prevention: Best Practices in Software Management, Kolawa, Adam; Huizinga, Dorota (2007).

[5] xUnit Patterns http://xunitpatterns.com/Goals%20of%20Test%20Automation.html, 2009-05-26

[6] Unit Tests, http://www.extremeprogramming.org/rules/unittests.html, 2009-05-25

[7] MH Introduction, Internt dokument, Tieto, 2009-05-05

[8] MHWeb, Intranät, Tieto, 2009-05-05

[9] ClearQuest Perspective, http://www.ibm.com/developerworks/rational/library/04/r-3089/Intro_ClearQuest_Perspective.jpg, 2009-05-25

[10] ClearQuest, Intranät, Tieto

[11] Basic JMS API Concepts, http://java.sun.com/products/jms/tutorial/1_3_1-fcs/doc/basics.html, 2009-05-25

[12] http://loic-mathieu.developpez.com/conception/article/clearcase/images/clearcase_05.jpg, 2009-05-25

[13] http://theopensourcery.com/images/javajspEclipse.gif, 2009-05-25

[14] http://www.cloudgarden.com/jigloo/images/screenshot5.PNG, 2009-05-26

[15] http://wiki.callistaenterprise.se/download/attachments/7438680/smallscreen.gif, 2009-05-27

[16] http://www.object-refinery.com/classpath/plastic-junit.png, 2009-05-25

[17] http://sureshkrishna.files.wordpress.com/2007/11/php_download.jpg, 2009-05-25

[18] http://www.tieto.com, 2009-05-25

[19] Kravspecifikation för CQSim, Tieto Karlstad

[20] http://java.sun.com/docs/books/tutorial/reflect/index.html, 2009-05-25

[21] http://www.w3.org/TR/2008/REC-xml-20081126/, 2009-05-25

[22] http://www-01.ibm.com/software/awdtools/clearcase/, 2009-05-25

A Sammanfattning av förkortningar och begrepp

JMS – Java Message Service

XML – Extended Markup Language

XSL – Extended Stylesheet Language

XSLT – Extended Stylesheet Language Transformation

TR – Trouble Report

CQ – ClearQuest

CQ4E – ClearQuest for Ericsson

MH – Modification Handling

MHO – Modification Handling Office

CSR – Customer Service Request

SAP – Systems, Applications and Products

CRM – Customer relationship management

CSR – Customer Service Request

CR – Change Request

J2EE – Java 2 Enterprise Edition

API – Application Programming Interface