a

Sudden Impact

Processorienterad programmering (1DT049) våren 2012. Slutrapport för grupp 1

Gruppdeltagare

Olof Björklund	880304-7136
Marcus Utter	890830-1974
Mark Tibblin	870519-7898
Luis Armando Mauricio	840911-0791

Innehållsförteckning

Inledning

Att föra samman två olika programmeringsspråk och få dem att fungera i balans är något som kan ha många fördelar. På detta sätt kan den rent räknemässiga kraften hos ett språk förenas med den grafiska tillgängligheten hos ett annat. Det är precis dessa två faktorer som tagits i akt när projektet Sudden Impact skapats. Projektet använder Erlang för samtidighet och Java för grafiskt interface och användarvänlighet.

Syftet med projektet är att lära gruppmedlemmarna mer om samtidighet och kraften av att förena två olika språk som Erlang och Java. Hur detta kan göras, vad för problem som kan uppkomma och möjligheterna som finns med en sådan sammanslagning.

Målsättningen med projektet har varit att få en fungerande prototyp av ett RPG-spel att fungera. Första målet är att som ensam spelare kunna spela mot datorstyrda spelare, andra målet att flera spelare skall kunna spela samtidigt på samma terminal, och slutmålet att fler spelare skall kunna spela tillsammans över ett öppet nätverk.

Det övergripande problemet med detta projekt är att få kommunikationen mellan Erlang och Java att fungera. Detta är den huvudsakliga nöten att knäcka för projektet, samt att få de två språken att fungera med samtidighet och smidighet. Utöver detta förväntas det bli en utmaning att med hjälp av *distrubuted Erlang* få programmet att kunna köras på flera terminaler samtidigt och på detta sätt göra det möjlighet att simulera scenarion för testning.

Avgränsningar som valts är att inte fokusera allt för mycket på nätverkskommunikation till en början. Programmets interna kommunikation måste fungera felfritt, inkl. tester, innan projektet går vidare och utökas.

Sudden Impact:

Bild 1 – Översikt

Sudden Impact är ett klassiskt RPG-spel i western-miljö. Du styr och kontrollerar din karaktär *Klintan* som måste besegra svurna fiender i ett nostalgiskt western-landskap som exemplifieras i *Bild 1*.

Du kontrollerar din karaktär i världen med tangentbordet. Kontroller och dess funktioner finns angivna i *Tabell 1*.

Tabell 1 - World-Kontroller

Knapp	**Funktion**
W	Gå uppåt
A	Gå vänster

S	Gå nedåt
D	Gå höger

När *Klintan* stöter samman med en annan spelare eller en datorstyrd fiende startas en strid. Striden sker i ett nytt fönster där de två kombattanterna slåss i tvekamp. Den som blir besegrad dör och försvinner ur spelet, segraren ges en form av belöning och får fortsätta spela. Kontrollerna vid en strid finns angivna i *Tabell 2*.

Tabell 2 - Battle-kontroller

Knapp	**Funktion**
Mellanslag	Skjuta

Bild 2 – Strid

Spelet är slut när samtliga fiender i världen är besegrade.
Vill du sluta spela stänger du helt enkelt ned fönstret med hjälp av krysset i högra hörnet.

Programmeringsspråk

För detta projekt har två språk valts: Erlang och Java.

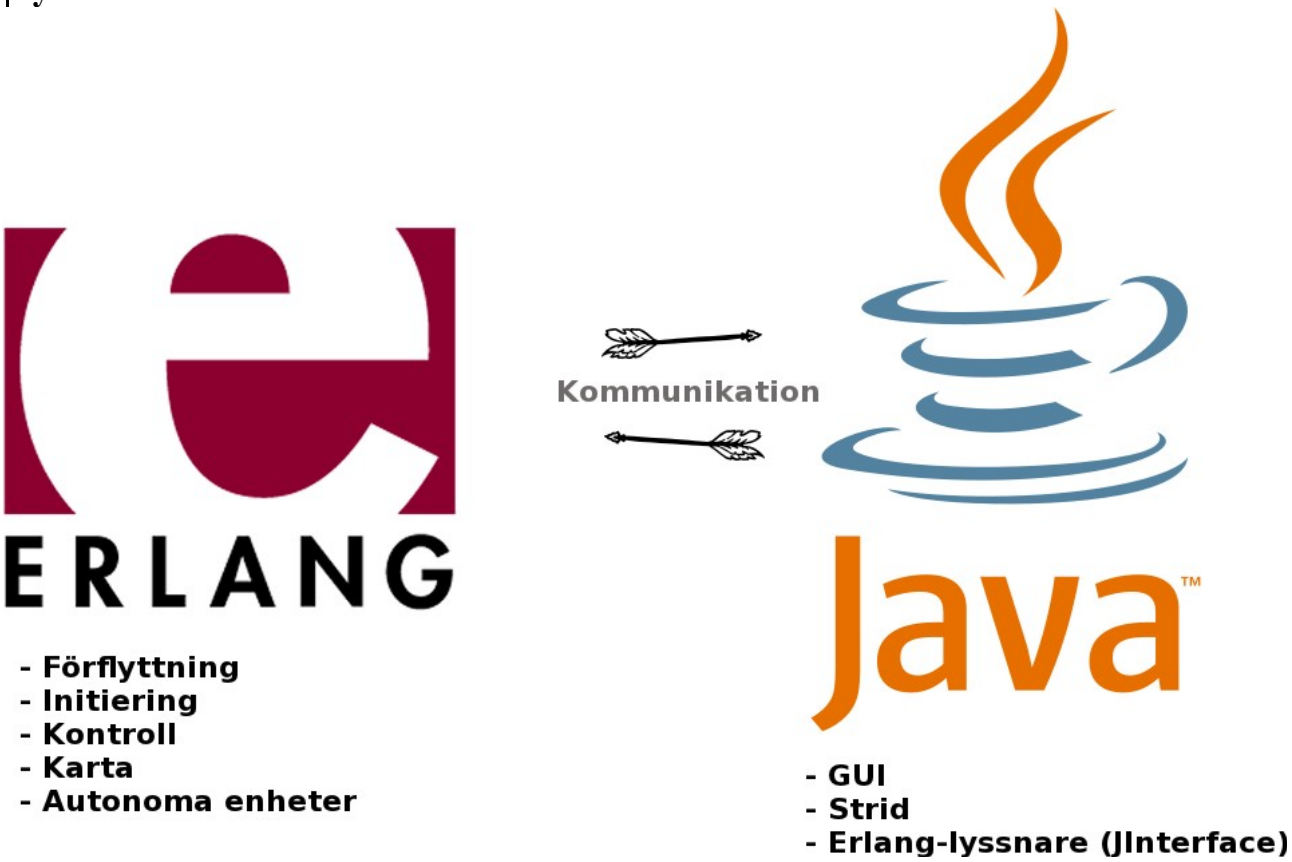
Erlang har valts dels på grund av sin snabbhet gällande samtidighet och *message passing* dels i utbildningssyfte / av intresse för språket. Eftersom Erlang är ett funktionellt språk lämpade de sig väldigt bra för spelets logik och rent räknemässiga funktioner.

Java valdes på grund av dess kompatibilitet med grafisk programmering / GUI samt att det medföljer ett verktyg för att upprätta kommunikation mellan Java och Erlang, kallat JInterface.

Den största nackdelen med Erlang är att ingen av gruppmedlemmarna har någon direkt erfarenhet av språket, varför det kommer ta mycket tid att ta reda på exakt hur saker och ting kan göras. Framförallt för att kommunikationen med Java skall fungera bra.

Nackdelen med Java är att det egentligen inte lämpar sig speciellt effektivt när det kommer till nätverkskommunikation. Förutsatt att spelet skulle skalas upp och implementeras på ”nätverksnivå” med hundratals spelare skulle med största sannolikhet flaskhalsar uppenbara sig i Java-koden.

Systemarkitektur



I projektet används en arkitektur som bygger på kommunikation mellan Java och Erlang med hjälp av ett Java-bibliotek kallat Jinterface[1]. Utvecklat av Ericsson tillåter detta bibliotek att kommunikation sker direkt mellan Erlang och Java. Den huvudsakliga kommunikationen sker genom att Erlang skickar tupplar med meddelande till Java, vilka uppfattas och förändrar GUI.

Samtidighet (concurrency)

I projektet används *message passing* i samband med *processer* för att skapa samtidighet / concurrency. T.ex. är varje individuell spelare en unik process som har sitt eget PID (process identification number) vilket kan användas för att kommunicera med just den processen. På samma sätt har Main-processen sitt eget PID vilket bland annat används för att kommunicera med GUI.

Message passing i Erlang underlättar kommunikationen med GUI avsevärt då meddelanden kan ”bangas” mellan Java och Erlang oberoende. Om något händer i Java som direkt skall meddelas till Erlang kan vi enkelt skicka ett meddelande som snappas upp av en väntande process i Erlang.

De delar av systemet som använder sig av message passing är framförallt Erlang-delen. Implementationen är gjord så att kommunikationen mellan Java och Erlang huvudsakligen sker från Erlang till Java, alltså att Erlang skickar meddelande till Java som reagerar på dessa och målar upp förändringar på GUI. Till viss del sker kommunikationen även åt andra hållet, t.ex. då spelets stängs ned eller när en spelare flyttar på sig.

Delar av systemet som inte utnyttjar sig av samtidighet är t.ex. poäng-registrering vilket sker oberoende av annan kommunikation. Även när en strid startas sker de som en separat process vilken inte kommunicerar med de övriga systemet via concurrency.

Ett behov av synkronisering uppkommer då en strid skall startas. Efter att två spelar stött ihop och skall börja en strid kan inte en tredje spelare tillåtas stöta ihop med någon av dessa. Detta löser vi med hjälp av ett tillstånd *freeze* som avgör om en spelare är i strid (freeze) eller inte i strid (unfreeze).

[1]

Vidare används synkronisering för spelarnas positioner i världen vilket representeras av ett ständigt flöde mellan Java och Erlang.

Fördelar med den valda implementationen är att kommunikationen mellan processer sker väldigt snabbt tack vare message passing. För övrigt underlättar den avsevärt för testning då scenarion kan specificeras och testas för trovärdighet. Nackdelen med att använda två olika språk är att biblioteket *Jinterface* krävs för att kommunikationen skall fungera. Om ett fel uppstår i Java kommer detta att påverka vad som händer i Erlang och vice versa.

Ett alternativ till den valda implementationen hade kunnat vara att skriva hela programmet i Java eller Erlang. Detta skulle skulle minska risken för fel men projektet skulle förlora lite av sitt syfte, att lära mer om kommunikation mellan två olika språk. Skulle implementation gjorts i t.ex. enbart Java skulle trådar komma att användas istället för processer och message passing, men eftersom endast en tråd används i Java finns ingen risk för deadlock, vilket är väldigt positivt!

Algoritmer och datastrukturer

Hela spel-världen representeras som ett koordinatsystem. När en karaktär förflyttas skickas nya X- och Y-koordinater till Erlang vilka registreras i en hash-tabell (ett Dictionary[2]). Denna tabell används senare för att identifiera om en viss koordinat är upptagen eller inte och på så vis avgörs om en förflyttning kan genomföras eller ej.

I Erlang har en process övergripande betydelse, *mainloop*. Denna process registrerar knapp-tryckningar vilka identifierats i Java och skickats till Erlang. Denna process har ansvar för att förflytta spelare till nya koordinater, registrera nya spelare samt avregistrera de som gått ett illa öde till mötes.

Funktionen *start* i Erlang hjälper programmet att kompilera korrekt genom att skapa nya spelare (datorstyrda som mänskliga) samt bestämma ett korrekt *HostName* vilket måste användas för att kunna starta kommunikationen mellan Erlang och Java med hjälp av make-file.

```
{ok,Host} = inet:gethostname(),
  HostFull = string:concat("sigui@",Host),
```

Kod-exempel 1 - *HostName*

Denna kod-snutt används sedan i make-filen för att kompilera Erlang med korrekt host-name för nätverkskommunikation.

```
start: all
        (cd ebin && erl -sname "player" -eval 'main:start()')
```

Kod-exempel 2 – *Make start*

Algoritmen för en mänsklig respektive datorstyrd spelare är i grund och botten densamma, med undantaget att en datorstyrd spelare rör sig på måfå runt i världen. En knapptryckning registreras i Java med hjälp av en lyssnare som skickar tillbaka en tuppel innehållandes {move, ”tangent som tryckts”} till Erlang. Denna registreras och spelarens position uppdateras. T.ex. skulle en registrering av att knappen ”w” tryckts tolkas på följande sätt:

```
{move,w} ->
        MainPID ! {walk, self(), ["w"]},
           humanloop(MainPID, GUIPID);
```

Kod-exempel 3 – move

Där MainPID är PID till main-processen och GUIPID är PID till Java. När spelaren erhållit sina nya koordinater skickas dessa tillbaka till Java:

```
{newposition, {CoordinateX,CoordinateY}} ->
            GUIPID ! {move,human,self(),self(),CoordinateX,CoordinateY},
```

Kod-exempel 4 – newposition

Erlang och JAVA kommunikationen sköts med hjälp av klassen “erlController” som har JInterface biblioteket som stöd. Klassen har en *listener* som under hela sin körning lyssnar på allt Erlang väljer att skicka till JAVA som bland annat innefattar anrop som “Battle”, “inBattle” och andra logiska funktioner som behövs för att få spelet att fungera. Självaste

[2] http://www.erlang.org/doc/man/dict.html – Fungerade senast 05/22/12

dataflödet mellan språken sker via objektet kallade “mbox” asynkront. Meddelanden plockas genom noder från Erlang (processer) som identifieras genom deras PID (Process Identification Number).

Strukturen på meddelandet vid messagepassingen I JAVA-koden liknar den I Erlang-koden (Kod-exempel 5). Meddelandet från mbox delas upp i olika datastrukturer. Atomerna utnyttjar vi för att identifiera vilken typ av karaktär eller event som skall manipuleras under spelets gång.
De dubbla PIDsen används för att hantera när två karaktärer hamnar i strid dvs mellan spelare och bot. “x” och “y” använder vi för att styra objekten på spelplan.

```
msg = (OtpErlangTuple)object;
command = (OtpErlangAtom)msg.elementAt(0);
player = (OtpErlangAtom)msg.elementAt(1);
player1PID = (OtpErlangPid)msg.elementAt(2);
player2PID = (OtpErlangPid)msg.elementAt(3);
x = (OtpErlangLong)msg.elementAt(4);
y = (OtpErlangLong)msg.elementAt(5);
```

Kod-exempel 5 – ErlController

För att skicka tillbaka händelser som sker i JAVA har ErlController följande metoder:

close()

Dödar alla processer i Erlang.

Move(String direction)

Meddelar Erlang att flytta spelaren i önskad riktning.

Kill(OtpErlangPid loser, OtpErlangPid winner)

Terminerar processen loser och meddelar vem som är vinnaren

Förslag på förbättringar:

- Lägga in Paneln “battle” i main Frame
- Ordna upp Start GUI
- End game GUI
- In-game events
- Spelar-Chat
- Score count
- Battle-statistik
- Bot level/ AI
- Skapa en större värld genom att skapa flera planer av same storlek som nuvarande Jpanel med olika typer av miljöer
- Skapa andra typer av botar med olika beteendemönster
- Storyline för att få en bättre in-game känsla
- Multiplayer gaming genom användning av något liknande som peer to peer
- Mer typer av object för vilda västern känsla

Allt grafiskt in-game skall se mycket mer smidigare ut med annan typ av implementation men det som är aktuellt är ett gränssnitt som funkar för vårat spel för att demonstrera hur det eventuellt kan komma att se ut framöver. Allt skall ske I en och samma "frame" med ett start GUI där önskan av karaktär, val av spelnivå, online-gaming, grafik inställningar och story-line inställingar.

-Multiplayer gaming

Multiplayer gamingen skall ske med en client/server uppkoppling där val av DM (Deathmatch) eller Co-op kan ske.Servern skall göras i erlang där vi har en uppsättning av att sköta all logik ska skötas i erlang. Detta skall I vår mening helt enkelt göra att vi helt och hållet måste se över en ny system arkitektur för spelet än den vi har när man spelar "single game", dvs i princip skall all logik förbli i erlang för att underlätta synkroniseringen. Det skall även ingå kommunikationsväg genom en chat funktion.

Först och främst finns ett flertal ideer om självaste "battle". Den nuvrande utgår från "one-shot, one-kill" kill princip med alltid träff.
Battle sker genom att motståndarna har en "healthbar" eller antal träffar som livslängd under striden. Självaste skjutandet skall utgå från att man siktar in sig mot ett mål så bra som möjligt för att träffa fienden. Efter stridens slut så ska en statistik poppa upp med värden som "Accuracy", "Hits", "Kill-style" m.m. Dessutom sätts mer statistik för fortsatt "lvlande" och äventyrande för slut eventet I spelet.

"Wild West booze fight" som innefattar en all out strid för alla som är på plats för skjuta ner varann. Man ska kunna röra sig runt I antingen öken miljö eller stad.

Olika typer av "minispel" för olika typer av uppdrag relaterat till vilda västern t.ex. lassokastning

Storylinen kommer relatera till alla Clint Eastwood filmer där vi lägger till en massa sköna repliker som Clintan dragit under sin skådespelar karriär. Men den vill vi inte avslöja helt för den är helt enkelt epic.

En mer utvecklad AI upplevelse på botarna för att få en interaktion med spelaren. Ett exempel är att en bot hamnar i "frenzy mode" och börjar jaga en.

Reflektion

Det största mest spännande har varit att jobba i två språk i ett och samma projekt. Först och främst se hur pass enkelt det var med messagepassing mellan språken och utnyttja båda språkens fördelar.

Spelet hade kunnat göras snyggare, mer avancerat och hade gått snabbare att utveckla om enbart Java använts. Det var ett klart val ur utbildningssynpunkt att inkludera Erlang. Att använda ett programmeringsspråk gör ett projekt avsevärt mycket lättare eftersom du inte behöver gräva dig in i hur du skall få två olika språk att fungera parallellt, dessutom underlättar det även för förändringar och tester eftersom du inte behöver testa med avseende på ett annat språks funktionalitet.

Hade vi haft chansen att göra om projektet skulle tester och dokumentation skrivits mer löpande. Angående tekniska aspekter hade det varit väldigt givande med någon form av mer stabilt / lättlärt repository, kanske något förutbestämt, som vi kunnat använda. Varför inte en ge en kort tutorial inom kursen för detta då det faktiskt är något obligatoriskt för samtliga grupper att använda.

Installation och fortsatt utveckling

Av Java så används den senaste versionen och av Erlang används version R15B. JavaDoc och Edoc används för att generera dokumentationen av koden, samt för testningen så använder vi oss av JUnit och EUnit för Java respektive Erlang.

Källkoden finns tillgänglig på Github. För att ladda ner den så är det bara att köra denna kommando: “git clone https://[KONTONAMN]@github.com/isqb/POP.git”, där [KONTONAMN] ska bytas ut till det användarnamn som Ni har på Github.

Som det ser ut så finns all kod under src katalogen, där inne finns det en till katalog, Graphics, som innehåller alla bilder som används. Katalogen doc innehåller rapporten för denna projekt. Katalogen ebin finns för att innehålla alla beam filer som uppstår när Erlang koden kompileras

För att sedan starta upp programmet gå in i katalogen där make-filen för programmet finns och skriv in ”make start” på kommandotolken. Det som kommer hända då är att den öppnar först en jar fil som innehåller den förkompilerade java delen av programmet vilket öppnar ett fönster samt väntar på att få information från Erlang, och sedan kompileras och körs Erlang koden vilket startar med att skapa en del botar samt en spelbar karaktär och säger till Java vart dem ska vara någonstans på fönstret.