Microsoft Azure: Regelmotor

- Molnbaserad regelhantering och kommunikation

LUNDS UNIVERSITET
Campus Helsingborg

LTH Ingenjörshögskolan vid Campus Helsingborg
Institutionen för datavetenskap

Examensarbete:
Jimmy Wallberg
Carl-Henrik Laulaja

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

Tryckt i Sverige
Lunds universitet
Lund 2016

Sammanfattning

Examensarbetet utfördes mot Combitech AB i Helsingborg där man efterfrågade en regelmotor som skulle placeras i molnet. En del av projektet gick ut på att identifiera en funktionell lösning för detta och välja ut den regelmotor som ansågs vara bäst lämpad, utefter de krav som ställs på systemet. Flertalet kandidater analyseras och efter uteslutningsprocess kunde den bäst lämpade för projektet lyftas fram.

Utöver analys av existerande regelmotorer identifierades och skapades en passande lösning för lagring av behövlig information och användarinteraktion.

Denna rapport har fokus på analysprocessen och valen som gjorts angående form av kommunikation mellan de olika delarna i systemet samt vilken regelmotor som valts ut och varför.

Projektet fick ett lyckat resultat där en regelmotor placerades i molnet och samtliga involverade delar för kommunikation visade sig vara funktionabla. Regelmotorn som valdes var *NRules* som krävde vidare utveckling för att kunna placeras i molnet.

Utöver detta beskrivs hur resultatet blev och de komplikationer som uppstod med olika delar, varför just *NRules* var det bästa valet, vilka data-lagringsalternativ som fanns, vilka som valdes och varför dessa var bäst lämpade för önskad funktionalitet.

Molnet som examensarbetet riktades mot var Microsofts eget *Microsoft Azure* som är väl etablerat på marknaden och har ett utbud av tjänster där bland annat “webjobs” och “storage account” visade sig lämpliga för projektet.

Nyckelord: NRules, Regelmotor, Molnteknologi, Microsoft Azure, C# .NET

Abstract

This bachelor thesis and project were done at Combitech AB in Helsingborg where the company had shown a previous interest in a rule engine solution for placement in the cloud. Part of the project was to identify an existing rule engine that could be further developed and implemented into a larger solution also adhering to any requirements presented by the target cloud system. Several candidates were analyzed and after an exclusion and testing process, the best suited rule engine was chosen.

Beyond analyzing the existing rule engines, suitable solutions for storage and inter-app communication were identified and created along with a basic interface for user interaction.

This thesis focuses on the analysis process; which rule engine was chosen and why as well as the selections that were done concerning communication between the different parts of the system.

The result of the project was deemed successful as a rule engine was integrated and uploaded to the cloud, still showing positive results regarding its' data processing procedures and internal communication capabilities. The rule engine that was selected was *NRules*, which needed further development in several aspects before able to work as a cloud service.

This thesis further describes a number of other results met during the project as well as complications that were discovered while developing the various parts of the system and why *NRules* was deemed to be the most suitable rule engine solution.

The target cloud for the project was Microsoft's own *Microsoft Azure*, which is well established on the market today and has a wide selection of services, of which "webjobs" and "storage accounts" proved suitable for this project.

Keywords: NRules, Rule Engine, Cloud Technology, Microsoft Azure, C# .NET

Innehållsförteckning

1 Inledning

Antalet företag som inser fördelarna med molnbaserad mjukvara har på senare tid ökat då den bakomliggande teknologin nu nått en nivå som tillåter leverantörer att garantera en viss kvalitét på produkter och priser som ofta är lägre än kostnaden för en traditionell miljö där företagen själva hanterar all administration och distribution av hårdvara. I molnet kan mjukvara tillåtas en skalbar tillgång av resurser baserat på hur mycket den faktiskt kräver, vilket innebär att man kan erbjuda en flytande prismodell som blir mer attraktiv jämfört med inköp av servrar och löner till tekniker samt tillåter utvecklare ett friare spelrum när det kommer till programmering av just resurshantering.

På Combitech AB finns ett intresse för denna växande marknad och således har man efterfrågat två relaterade lösningar på produkter med inriktning på molnet och "*Internet of Things*". Dels är man intresserad av att utveckla mjukvara för sensorer som samlar information i hemmamiljö (som t.ex. temperaturer, luftfuktighet och dylikt i olika fastigheter) och dels molnbaserad mjukvara som kan fungera som ett filter som lyssnar på informationen från sensorerna och andra typer av indata och automatiskt utför uppgifter baserat på vad dess användare är intresserade av. Denna rapport handlar om den senare och arbetet med att implementera en regelmotor i *Microsofts Azure*-miljö.

1.1 Om Combitech AB

Combitech grundades 1992 som Combitech Software AB men bytte senare namn till Combitech Systems AB och blev uppköpta av Saab 2002. Under 2006 omorganiserades Saab-koncernen och man utförde en sammanslagning av Combitech Systems och AerotechTelub som var ett företag med inriktning mot försvarsteknik. Combitech blev således en specialinriktad del av Saab med fokus på teknik, säkerhet och miljö; "*Systems Engineering*", "*Security Solutions*" och "*Technical Information Solutions*", med ytterligare undergrenar inom varje avdelning.

Företaget ses allmänt som ett tekniskt konsultbolag och har kontor på flertalet platser i norden som riktar sig mot olika marknadssegment, bl.a. den civila marknaden inom industri och telekom men också mot försvarsmakten och olika myndigheter.
Under 2016 var fler än 1400 personer anställda av Combitech AB[1].

Kontakten inför detta examensarbete grundade sig i ett tidigare projektarbete på företaget som utfördes av en större grupp studenter från Lunds Tekniska Högskola i Helsingborg, på Combitechs lokala kontor i Helsingborg.

1.2 Syfte och mål

Med molnbaserad teknologi är huvudmålen att slippa de största problemen med att hantera sin egen serverhårdvara och de huvudbry som uppstår när produkten man skapat inte längre klarar av trycket av dess användare. Kanske var man inte alls beredd på genomslagskraften och servrar fick akut uppgraderas under en viktig lansering. Alternativt fick man inte alls den respons man hade förväntat sig och har istället fått betala i onödan för hårdvara och tjänster som aldrig kommer nyttjas till fullo.

Combitechs intresse för molnteknologin var i detta fall riktat mot smarta hemtjänster och automatisering av funktioner som kan underlätta arbete dels för bostadsföretag men även för enskilda hemanvändare. Man ville skapa ett användarvänligt gränssnitt och en tjänst som tillåter personer att skapa sina egna regler och handlingar, som då ska utföras när specifika kriterier uppfylls och viss data har samlats in från diverse sensorer. Man önskade en hög dynamik i systemet och att det tillåter enkel vidareutveckling.
Här följer exempel på scenarion som kan vara intressanta ur säljande perspektiv:

1. Fastighetsskötare skapar en regel: När temperaturen i bostad #1, #2 och #3 är över 25 °C, skicka en signal till värmepannan som sänker arbetslasten.
2. Bostadsägare skapar en regel: När min dörr är låst och rörelse detekteras i mitt hem, skicka SMS till min telefon.

Målet med detta examensarbete var att lyckas identifiera en lösning för datafiltrering som kan användas som grund i en kommersiell produkt för placering i molnet. Lösningen skulle integreras i en applikation som tillåter tolkning och följdaktion av indata baserat på specificerade filterregler som definieras på användarbasis. Utvecklingskoncept som användes skulle undersökas för att garantera god skalbarhet i slutprodukten. T.ex. ska systemet kunna användas för att filtrera och reagera på data från flertalet sensorer i ett bostadshus och låta de boende skapa och modifiera sina prenumerationer på denna data genom ett enkelt användargränssnitt.

1.3 Problemformulering

Baserat på en initial beskrivning av projektet tillhandhållen av företaget och ett informationsmöte med kontaktperson identifierades följande grundfrågor:

1) Identifiera en lösning för datafiltrering som kan fungera som bas i projektet.
   a) Vilka lösningar för datafiltrering (*rule engines*) är relevanta för implementation i molnet?
   b) Vilka fördelar och nackdelar har existerande lösningar och hur påverkar dessa vidare arbeten?
   c) Vid kostnadsfråga, hur påverkar detta intresset från Combitech AB?
2) Identifiera integrationsmöjligheter och krav hos *Microsoft Azure*.
   a) Vilka krav ställs det vid integrering med *Microsoft Azure*?
3) Implementera en applikationstjänst som integrerar lösning(1) och tillåter enkel modifikation av filtreringsregler med flera simultana användare.
   a) Vilken nivå bör komplexiteten kring modifikation av filtreringsregler hålla för att anses användbar av potentiell kund och ändå tillåta den dynamik som eftersöks av Combitech AB?
   b) Hur och var ska reglerna lagras?
4) Implementera ett simulationsprogram för testning och demonstration av applikationstjänsten(3).
   a) Hur ska tjänsten testas? Vilka faktorer blir relevanta att ta i beaktning?
   b) Vilka funktioner behöver demonstreras?
5) Implementera en tjänst för notifikation av användare baserat på av användaren valda filter.
   a) Vilka typer av notifikationer är av intresse för potentiella användare?
   b) Hur kombineras denna tjänst bäst med filtreringsapplikationen?
   c) Hur kan man kontrollera skalbarheten vid stort antal simultana användare?
6) Kombinera lösningar 1-5 i en demonstrerbar version som integreras med molntjänsten *Microsoft Azure*.
   a) Hur garanteras det att samtliga lösningar fungerar enligt de kriterier som ställs av *Microsoft Azure*?
   b) Vilka kriterier ställer Combitech AB på demonstrationens innehåll?

1.4 Avgränsningar

Då en stor del av arbetet som skulle utföras var relaterat till programmering ansågs det lämpligt att välja ett programmeringsspråk som inte låg alltför långt ifrån det som lärs ut på datateknikprogrammet av Lunds Tekniska Högskola, detta i synnerhet med tanke på tidsbegränsningar. Språket blev därför *C# .NET* då det liknar *Java* och dessutom är det språk som primärt används av den aktuella molntjänsten *Microsoft Azure*. Det bestämdes att detta språkkrav även skulle gälla för eventuell källkod till existerande regelmotor-lösning för att underlätta inlärning av koncept och undvika kompatibilitetsproblem.

Microsoft Visual Studio 2015 användes som primär utvecklingsmiljö då detta har en direkt officiell koppling till *Microsoft Azure* via företagets egna *SDK*[2], har inbyggda funktioner för att dela ett projekt mellan olika datorer och utvecklare samt finns att ladda ner gratis.

Från Combitechs sida fanns det en lista[3] med förslag på regelmotorer, till största del med öppen källkod, som kunde bli relevanta för projektet. Det ansågs lämpligt att förhålla sig till denna lista så länge den inte efter analys visade sig sakna lämpliga kandidater.
Detta innebär också att all information som hämtades och analyserades har sitt ursprung på diverse websidor.

Arbetet utfördes huvudsakligen på Combitechs kontor i Helsingborg där det fanns utrymme för flera personer och tillgång till sensorhårdvara och datorer. Under utvecklingen användes dock privata laptops då detta blev mer effektivt med tanke på prestanda och att dedikerade utvecklingssystem kunde skapas på dessa.

2 Beskrivning av relaterad teknik

Detta stycke syftar till att ge en grundläggande beskrivning av element som varit av intresse för projektet; direkt relaterade koncept eller koncept som Combitech under projektets gång framfört en önskan om att vilja undersöka.

2.1 Regelmotorer

En regelmotor är ett system som baserat på förbestämda kriterier utför specifika handlingar. Kriterierna är baserade på viss indata som systemet hämtar eller får från någon eller flera källor och kan väljas eller bestämmas av användarna av systemet, oftast med simplifierade metoder som inte kräver någon tidigare erfarenhet av programmering. Exempelvis hade en affärskedja kunnat använda en regelmotor för att tillåta marknadsundersökare applicera rabatter på vissa produkter baserat på kundernas tidigare inköp, genom att mata in en text vars språk är mer likt svenska (dock i realistiska fall: engelska), t.ex.: "*FÖR $produkt[artikelnummer #1] GÄLLER: OM $kund HAR KÖPT $produkt[artikelnummer #2] APPLICERA RABATT 5%*", där versalerna symboliserar nyckelord som konverteras till lämpliga metoder inuti programmet.
Vilket språk eller struktur reglerna har bestäms av programmets skapare men generellt försöker man nå en hög dynamik och det finns ett antal färdiga standarder[4] tillgängliga på internet man kan följa, vissa så pass välkända att det dessutom finns kurser för folk som vill lära sig språken på professionell nivå.

Konceptet med regelmotorer kan användas på flera olika sätt och behöver inte nödvändigtvis vara någonting som agerar ifrån en server utan hade kunnat användas som ett lokalt program för att t.ex. konvertera mellan två (eller flera) olika filer eller filformat och inkludera och formatera information som användaren väljer genom att applicera sina egna regler. Vilka format detta är kan vara förutbestämt av regelmotorn alternativt att en implementation gjorts där variabla källor är tillåtna.

Projektet har inkluderat analys och jämförelse av flertalet regelmotorer utifrån dess preferenser, möjligheter till vidare utveckling och estimerad inlärningstid för grundläggande förståelse av regelmotorn. Ett antal av de

regelmotorer som undersökts är listade nedan, dessa som till någon grad varit intressanta för projektet och inte direkt visat sig vara olämpliga med avseende på krav och önskemål från Combitech.

Zapier
Utvecklad av *Zapier.com*. Detta är en online-baserad lösning för att skapa regler och följaktioner direkt i webbläsaren baserade på dataflöde mellan över 500 olika fördefinierade applikationer som t.ex. Mailklienter, forum och sociala medier[20].

Ubidots
Utvecklad av *Ubidots.com*. Detta är en online-baserad lösning där regler skapas direkt i webbläsaren men till skillnad från *Zapier* är denna specialiserad på inläsning av data från enheter så som *Arduino, Raspberry PI* och annan hårdvara som kan användas som minimala sensorer[22].

OpenHAB
Utvecklad av *Openhab UG*, baserad på *Java* och distribuerad som öppen källkod under *EPL - Eclipse Public License* på *GitHub*[19].

NRules
Utvecklad av Sergiy Nikolayev, baserad på *C#* och distribuerad som öppen källkod under *MIT*-licens på *GitHub* där även samtliga resurser och dokumentation för produkten finns tillgängliga[24]. *NRules* tillåter inläsning av regler skrivna med ett simplifierat *C#*-liknande format vid körning. Dessa regler kompileras och lagras i en sessions-variabel som sedan kan fyllas på med faktaobjekt var på ett anrop kan utföras för att bestämma vilka regler som matchar vid just det tillfället.

NxBRE
Utvecklad av David Dossot, baserad på *C#* och distribuerad som öppen källkod under *MIT*-licens på *GitHub*[26]. *NxBRE* tillåter inläsning av regler skrivna i *RuleML/XML* som kompileras vid körning.

SRE
Utvecklad av *Sierra Digital Solutions Corp*, baserad på C# och distribuerad som öppen källkod under *MIT*-licens på *SourceForge*[28]. Utvecklaren kallar *SRE* för en simplare och mindre prestandakrävande version av *NxBRE* och läser in regler i liknande *XML*-format.

BizTalk Services(Azure)
Utvecklad av *Microsoft corp*. *BizTalk Services* är inte en lösning som är tillgänglig för vidare utveckling utan är en tjänst som redan finns i *Microsoft Azure* och kan tecknas mot en månadskostnad som baseras på mängden data som hanteras[30]. Tjänsten har en del begränsningar då den endast kommunicerar med andra tjänster inom *Microsoft Azure* och då med vissa krav på meddelandeformaten.

Huginn
Utvecklad av Andrew Cantino och flertalet andra privatpersoner, baserad på *Ruby* och distribuerad som öppen källkod under *MIT*-licens på *GitHub*[31]. I motorn skapar man "agenter" som utför en uppgift baserat på olika plugins för applikationen, vilket i princip motsvarar vad som benämns regler i andra applikationer.

Drools
Utvecklad av *Red Hat Inc*, baserad på *Java* och distribuerad som öppen källkod under *Apache Software License 2.0* på *GitHub*[32]. *Drools* är en komplett, extensiv affärslösning där regelmotorn är en del av ett mycket större paket.

Code Effects
Utvecklad av *Code Effects Software* och distribuerad som bibliotek för *.NET* med olika licenser beroende på vald prismodell (t.ex. enterprise-version för motsvarande ca 33.000:-). Företaget själva påstår att produkten är den snabbaste *XML*-baserade regelmotorn i affärssammanhang[34].

2.2 Microsoft Azure

Azure är ett samlingsnamn för samtliga tjänster *Microsoft* erbjuder som har någon anknytning till deras molnteknologi; tjänster för online datalagring, datahantering, analystjänster och flera andra koncept som är intressanta inom utveckling. Totalt rör det sig om över 50 olika tjänster som man traditionellt fått köra på en egen server men nu kan få tillgång till via *Azure*, vilket även tillåter en standardiserad kommunikationsstruktur för informationsutbyte mellan de olika applikationerna[5].

2.2.1 Allmänt om molnteknologi

Moln eller molntjänster (engelska: “cloud computing”) är teknologi som flyttar olika tjänster och lagringsutrymmen från att ligga lokalt på en arbetsplats eller enskild miljö till att ligga i stora serverhallar som man hyr in sig på, för att slippa köpa in dyr programvara eller hårdvaru-utrustning själv, vilket i synnerhet kan vara lämpligt om man inte i förväg är medveten om vilken kapacitet eller beräkningsresurser ens tjänster kommer nyttja. I molnet kan detta skötas automatiskt genom att man delar resurser med alla andra som också hyr in sig hos företaget och när ens applikationer behöver extra kraft kan detta tilldelas temporärt. Skalbarheten blir beroende på arkitekturen i molnet som ofta består av hundra- eller tusentals datorer på olika fysiska platser i världen och kan således vara långt mycket mer kraftfullt än något man själv kan skapa på en lokal arbetsplats. Detta kan även minska riskerna för att data blir förstörd p.g.a. hårddisk-kraschar, brand eller andra liknande problem då företag som levererar molntjänster oftast placerar den data som användaren äger på flera olika serverdatorer, utifall en serverhall skulle ligga ner för underhåll alternativt råkar ut för andra problem som förhindrar funktionaliteten[6].

I den modell som många leverantörer av molntjänster använder sig av för marknadsföring brukar man lägga fokus på möjligheterna för företag och användare att expandera kapaciteten aktivt under tiden som de prenumererar på tjänsterna, utan att förändra eller uppgradera någon hårdvara vilket istället normalt sköts per automatik när en viss gräns är nådd för hur t.ex. en mängd data/dataflöde eller en andel beräkningskraft som nyttjats under en bestämd tidsperiod. Detta kopplas till olika betalningsmodeller och låter prenumeranter betala den minsta möjliga summan för att deras molntjänster ska fungera optimalt[7].

Nackdelarna med molnbaserade tjänster är säkerhetsaspekter, då man ger

leverantören full tillgång till det man lagrar beroende på hur användaravtalet lagts upp. Detta blir i synnerhet aktuellt om man t.ex. arbetar på något som inte än har blivit patenterat och ens konkurrenter lyckas köpa loss information från molnleverantören som ger för hög insikt i, eller avslöjar för mycket om ens arbete. Därav hamnar stor vikt på att faktiskt grundligt kontrollera det avtal man ingår och man måste själv avgöra om den aktuella molnleverantören går att förlita sig på när det kommer till övriga säkerhetsfaktorer; kryptering, datorintrång, o.s.v.

2.2.2 Stream Analytics

Stream Analytics är en av tjänsterna för data-analys som finns att tillgå i *Microsoft Azure* och går att nyttja direkt genom deras användargränssnitt eller kontrollera via anrop från andra tjänster som man kan programmera själv[8]. Tjänsten kom på tal då Combitech hade tankar om dess användning i detta projekt då dess beskrivning påminner om konceptet för regelmotorer. Tjänsten används för att hämta data från en eller flera källor, baserat på förfrågor skrivna ett simplifierat språk likt det som används i *SQL* (*Structured Query Language*[9]) och på kriterier man själv väljer, för att sedan sända informationen vidare till en annan tjänst inom *Microsoft Azure*. Det intressanta med just denna tjänst är att förfrågorna även går att konvertera till realtidsströmmar av data och således i teorin skulle gå att använda direkt som en regelmotor.

Tjänsten har möjlighet att lyssna på data ifrån *event-hubbar* och/eller hämta data ifrån *Blob Storage* (dessa beskrivs senare i kapitlet), analysera datan med hjälp av ett antal andra tjänster tillgängliga i molnet och sedan skapa ny data av diverse resulterande svar och leverera denna i ett *JSON*-liknande format till någon av följande molntjänster i *Azure*; *SQL server, blob storage, event hub, table storage, DocumentDB, Power BI*[8].

2.2.3 Event hubs

En *event-hub* är egentligen ingen enskild fristående del i *Microsoft Azure* utan finns inom andra tjänster där just kommunikationen är huvudfunktion. *Hub*:en tillåter användare att skapa lyssnare- och talare-relationer, där olika tjänster (lyssnare) prenumererar på information från *hub*:en som eventuellt kontinuerligt matas in av andra tjänster (talare)[10], som t.ex. hårdvarusensorer med mjukvara konfigurerad för att sända till en specifik *hub*.

I detta projekt är konceptet relevant då Combitech sedan tidigare bestämt att kommunikation skulle utföras över en förkonfigurerad *Azure IoT Hub* som har samma funktionalitet men även implementerar några extra kommunikationsprotokoll som är vanliga inom *Internet of Things* samt stödjer högre nivå av säkerhet genom autentisering av enskilda enheter[11].

2.2.4 Storage

Azure Storage är samlingsnamnet för den tjänst i molnet som ger tillgång till lagringsrelaterade funktioner och tjänster och finns i två olika varianter; vanlig *Storage*, som innehåller *containers (blobs), tables, queues* och *shares*, samt *Blob Storage*, som är avsedd för dedikerad lagring och således har något annorlunda egenskaper och högre kostnad jämfört med den tidigare[12].

I detta projekt är den vanliga typen av *Storage* relevant då Combitech framfört en önskan om dess användning p.g.a. den aktuella kostnadsmodellen för tjänsten.

Containers/Blobs

Containers eller *Blobs* inom *Storage* tillåter lagring av ostrukturerad data i molnet som kan tillgås av samtliga tjänster eller applikationer som har tillgång till den unika hemliga nyckeln för lagringsutrymmet. Tillgängligheten kan dock även konfigureras efter en offentlig modell där man specificerar vad som är tillåtet (för samtliga applikationer med tillgång till utrymmets *URL*); läsning, skrivning eller både och. Denna form av lagringsutrymme används ofta för sparning av filer som inte nödvändigtvis är textbaserade utan kan vara ljudfiler, bilder, backup-filer eller dylikt.

Tjänsten är uppbyggd på så sätt att en *Container* innehåller en eller flera *Blobs* som kan vara av olika typ och när man vill upprätta en kommunikation med lagringsutrymmet gör man det mot *Containern*, vilket innebär att man kan hantera flera Blobs åt gången. För tillfället finns det ingen begränsning för hur många *Blobs* det får finnas inuti en *Container*, dock finns det vissa krav på vad en *Blob* får innehålla beroende på vilken typ det är och vald prismodell för tjänsten. Beroende på typ kommer datan man lagrar struktureras på olika sätt och man får tillgång till specifika

operationer som endast är giltiga för vald typ och således kan man inte konvertera mellan de olika typerna när *Blob*:en väl är skapad[13].

De tre typer som finns tillgängliga är följande:

1) **Block Blobs**
   En *Block Blob* lagrar flertalet sektioner eller block där varje block har ett eget ID och är möjliga att ersätta om man laddar upp ett block med samma ID. Den maximala kapaciteten för ett enskilt block är 4MB men kan variera i storlek upp till denna gräns oberoende av de andra blocken. Totalt kan en *Block Blob* simultant innehålla 50 000 block, vilket gör det nödvändigt att, vid användning tillsammans med stora system, implementera en lämplig lösning för skalning för att inte överskrida den maximala gränsen, vilket skulle leda till ett system som inte fungerar[14].

2) **Append Blobs**
   En *Append Blob* är som namnet föreslår användbar när man alltid vill att den bifogade informationen ska hamna sist i den tidigare lagrade mängden data. Den tillåter inte att tjänster redigerar eller tar bort existerande data utan detta är endast möjligt genom *Azures* användargränssnitt. Storleksmässigt gäller samma begränsningar som för *Block Blobs*[14].

3) **Page Blobs**
   En *Page Blob* innehåller likt en bok flertalet sidor, varav varje sida har en specifik förbestämd storlek som det går skriva till och är speciellt anpassade för att hantera slumpmässiga läsningar och skrivningar. Den maximala storleken på en enskild *Page Blob* är 512 byte och den totala storleken för en samling *Page Blobs* som tillåts är 1TB[14]. Detta ger på ett ungefär 2,15*10^9st *Page Blobs* som maximal kapacitet.

Tables

I *table storage* ges möjligheten att lagra data i ett format som kan liknas med en mängd enheter där varje enhet har en eller flera mappningar mellan en nyckel och ett värde. Till skillnad från t.ex. *SQL* där varje enhet (i samma tabell) har samma nycklar kan enheter i *table storage* ha helt unika nycklar vilket ställer högre krav på att man vid bearbetning av data som

hämtats från databasen verkligen är medveten om vilka nycklar man är intresserad av och garanterar att dessa faktiskt existerar för det objekt man valt ut. Den maximala storleken på en enhet är 1MB som kan delas upp på upp till 252 nycklar (och värden)[15].

Queues
Tjänsten *Queues* (köer) ger tillgång till ett system som är avsett att användas för asynkron kommunikation mellan olika tjänster inom samma arbetsmiljö i molnet och dess funktionalitet påminner om den hos *Event Hubs*, d.v.s. en tjänst kan lägga ett meddelande på kön medan en annan tjänst lyssnar av den. Det finns ingen begräsning för antalet köer man kan lägga till under samma *Storage*-tjänst dock tillämpas en maximal storleksbegränsning på enskilda meddelanden som ligger på 64KB[16].

2.2.5 Power BI
Power BI används för att göra övergripande analyser av data som matas in i de olika tjänster man aktiverar under sitt *Azure*-konto, t.ex. mängd data som behandlas, var den används, om den skickas vidare och i så fall vart[17]. För projektet skulle den kunna användas för att se var i systemet en uppgradering behövs vid specifika tillfällen alternativt ifall regelmotorn behöver optimeras i en specifik del.

2.3 Sensorer

Sensorer i kontext av detta projekt refererar till en hårdvara med ett eller flera inbyggda mätverktyg och implicerar en viss typ av utgående dataflöde med specifikt format som för utvecklingens skull fördefinierades av Combitech. En enskild sensor kan innehålla flera värden baserade på t.ex. temperatur, luftfuktighet, rörelse-avkänning eller liknande och kan således användas som bas för automatiskt reglering av bl.a. uppvärmning genom utplacerande av flertal sensorer och beräkningar på deras data. Alternativt hade det varit möjligt att kontrollera mängden personer i en lokal genom placering av sensorer vid in- och utgångar. Hur datan väljs ut och skickas vidare bestäms av mjukvara på sensorn som i detta fall kommunicerar direkt med en *Azure IoT Hub* (se Kap. 2.2.3) genom *JSON*-serialiserade strängar vars format följer ett visst schema som tillåter att antaganden kan göras vid mottagning och bearbetning av datan.

Då utvecklingen av de fysiska sensorerna var del av ett annat parallellt examensarbete fanns det inledningsvis ingen fungerande hårdvara att utgå ifrån och således fick programvara för simulering av sensorer skapas. Dessa virtuella sensorer skulle förse miljön med slumpmässigt genererad data och skapades som simpla kommandobaserade program som kunde exekveras på *Windows*-baserade datorer och baserat på användarinteraktion skicka ut värden för temperatur och luftfuktighet enligt samma format som en fysisk sensor.

För att underlätta ytterligare för utvecklingen tilläts det förutsättas att varje sensor specificerade sitt unika ID-nummer i varje utsändning av data, något som i en realistisk miljö blir svårare att förutsätta då detta helt beror på sensorns mjukvara.

2.4 Kommunikationsprotokoll

Ett protokoll som varit av intresse för projektet är *AMQP*, då detta används av sensorer för att skicka data till *Azure Event Hubs* (se Kap. 2.2.3).

AMQP 1.0 är en internationell standard som är godkänd av *ISO* och *IEC* som *ISO/IEC 19464:2014* och har varit under utveckling som ett samarbete mellan flera stora teknikrelaterade företag och leverantörer sedan 2008[38].

Under projektet nyttjades *AMQP* av de virtuella sensorerna genom implementering av *Microsofts* färdiga bibliotek för protokollet och *Microsoft Azure SDK* och således krävdes ingen högre förståelse av denna standard. Detsamma gällde även övrig intern kommunikation i molnet som utförs automatiskt med implementering av korrekta bibliotek angivna av *Microsoft*.

2.5 C# .NET och Visual Studio 2015

Utvecklingen som har utförts under projektet har varit i programmeringsspråket *C# .NET*, detta då det är det primära språket för tjänster som skapas för exekvering i *Microsoft Azure*. Språket är likt *Java* men har vissa signifikanta skillnader, bl.a. hur realtidsprogrammering görs. Det är framtaget av *Microsoft* som ett högnivåspråk med extra funktionalitet jämfört med tidigare populära *C++*, som kan anses svårhanterat och saknar den automatiska minneshantering som *C# .NET* inkluderar.

En utvecklingsmiljö som är kompatibel med *C# .NET* är *Visual Studio 2015*, vilken använts under projektet då författarna i tidigare projekt bekantat sig med verktygen miljön innehåller. Dessa inkluderar verktyg för att förenkla utvecklingen mot C#, så som tolkningsfunktioner och kompilator för den senaste versionen av *C# .NET* i och med automatiska uppdateringar av utvecklingsmiljön. *Visual Studio 2015* erbjuder dessutom ett verktyg för att hantera de bibliotek som behöver inkluderas i projekt genom s.k. "*Nuget*"-paket, vilka kan hämtas och uppdateras direkt genom *Microsoft*.

2.6 Microsoft Azure SDK

Microsoft Azures SDK är ett bibliotek för att kunna implementera tjänster mot *Microsofts* molntjänst *Azure*. Detta inkluderar bl.a. kommunikationen mellan dess olika funktionaliteter som t.ex. *Azure Storage Account*, *Event Hub, IoT-hub* m.m. SDK:n ger även stöd för att ladda upp applikationer till motsvarande tjänst i molnet som ska nyttjas, där detta projekt utnyttjar *Microsoft Azure Webjob*. Den gör det även möjligt att ladda upp applikationer till samtliga beräkningstjänster som *Microsoft* levererar[2].

3 Metod och analys

I detta stycke beskrivs arbetsprocessen som följdes utmed projektet, den initiala planeringen, vilka krav som identifierades och motiveringar till avgörande val som gjordes samt en summering av initiala analysmoment angående regelmotorer.

3.1 Arbetsprocess

Inledningsvis fanns ett förslag från Combitech om att tillsammans med ytterligare en projektgrupp hjälpa till med drivandet av projektet genom gemensamma sprintar i en *Scrum*-baserad arbetsmodell[39], detta något som snabbt visade sig icke genomförbart då ledning från företaget blev väldigt spontan och oregelbunden samt att där fanns en stor skillnad på arbetsuppgifter mellan grupperna trots att båda var riktade mot samma ämne. Istället har stora delar av utvecklingsarbetet utförts med hjälp av "*exploratory programming*", där man skapar ett antal olika koncept för att hitta passande lösning med avseende på prestanda och resurser[40]. Detta är mycket fördelaktigt när utvecklaren inte är fullständigt insatt i alla involverade delar då man stegvis får undersöka varje byggsten men också svårare att planera omkring då det på förhand är svårt att avgöra hur lång tid det kommer ta att slutföra en viss uppgift. I kombination med detta användes parprogrammerings-metoden frekvent, där en person fysiskt skriver kod medan den andra agerar validator och granskar koden samtidigt som den skrivs, varefter man efter en stund byter rollerna för inte fastna i ett visst spår samt få en ökad kvalité på koden[41].

Den löpande planeringen av projektet löstes med hjälp av framtagning av mindre delmål, uppföljning och testning inspirerad av *Kanban*[39] där nya uppgifter valdes ut varje påbörjad vecka baserat på vad som hunnits med veckan innan samt önskemål från Combitech som skjutits till under projektets gång. Initialt kunde de primära målen och grundkraven identifieras genom genomgång av projektets beskrivning tillsammans med företaget. Dessa utökades och modifierades senare under den analytiska fasen då bl.a. existerande regelmotorer och molnplattformen *Microsoft Azure* granskades och det blev tydligare vilka faktorer som faktiskt skulle komma att spela roll i projektet.

Då en lämplig grund med regelmotorfunktionalitet skulle identifieras ur en lista[3] (med några tillägg från Combitech) fick några relevanta aspekter för jämförelse bestämmas i samråd med företaget, bl.a. användarvänlighet (hur snabbt kunde man tolka de faktiska funktionsmöjligheterna), utvecklingspotential (om produkten överhuvudtaget hade öppen källkod och/eller licens som tillät vidare utveckling) samt möjlighet till support (hur aktiva eventuella kontaktforum för produkten var). Ytterligare var det av intresse att dessutom identifiera eventuella konkurrenter och vilken funktionalitet samt prismodell dessa kunde erbjuda. Underkapitel 3.3 innehåller en summering av denna analysfas.

Testning gjordes kontinuerligt under projektets gång genom skapandet av minimala testfunktioner för varje slutgiltig funktion i programmet samt simulering av multipla simultana användare och användningsfall både lokalt och på en distribuerad version i det faktiska molnet *Microsoft Azure*. Eftersom målet med den slutgiltiga produkten huvudsakligen var att bevisa ett koncept utfördes inga användarvänlighetstester av det web-interface som skapades för produkten. Programmeringsmässigt ansågs det lämpligt att följa konventionen för stil och kommentering i *C#* utgiven av *Microsoft*[18].

3.2 Krav

Utöver de frågor som ställdes under problemformuleringar (Kap 1.3) kunde följande konkreta krav bestämmas under projektets gång och användas för guidning under planeringsfaserna. Dessa togs fram dels genom återkommande diskussion med Combitech där önskemål på funktionalitet presenterades och dels efterhand som källmaterialet undersöktes, under den analytiska inledningsfasen.

1. Programmet ska implementera en regelmotor vars licens tillåter användning i kommersiella syften.

2. Programmet ska utvecklas i språket *C# .NET*.

3. Programmet ska utvecklas med framtida expansion i åtanke, d.v.s. klasser och metoder ska skrivas på så sätt att en person som inte tidigare arbetat med produkten snabbt ska få en överblick över programmets funktionalitet.

4. Programkoden ska följa de krav som ställs för utveckling mot *Microsoft Azure.*

5. Programmet ska kunna laddas upp i molnet *Microsoft Azure* som ett *Azure webjob.*

6. Programmet ska kunna laddas upp i molnet *Microsoft Azure* som en *Azure Web Application.*

7. Programmet ska ha funktionalitet för inläsning av regler från databas.

8. Databaser programmet använder sig av ska vara produkter inom *Microsoft Azure Storage*.

9. Programmet ska ha funktionalitet för ingående kommunikation från andra element i *Microsoft Azure*.

10. Ingående kommunikationsmöjligheter ska inkludera meddelanden från *Microsoft Azure IoT Hub* och *Microsoft Azure Event Hubs*.

11. Programmet ska ha funktionalitet för utgående kommunikation mot andra element i *Microsoft Azure*.

12. Utgående kommunikationsmöjligheter ska inkludera dynamiska meddelanden till *Microsoft Azure Queues* och *Microsoft Azure Event Hubs*.

13. Programmet får inte tillåta möjlighet till okontrollerade I/O-operationer som leder till ökad kostnad i *Microsoft Azure*.

14. Programmet ska skrivas med eftersträvad dynamik i användningen av inkommande och utgående kommunikationselement, d.v.s. dessa ska enkelt kunna bytas ut.

15. Programmet ska tillåta simultan exekvering och kompilering av regelförändringar, d.v.s. huvudfunktionaliteten för beräkningar på data ska inte avbrytas av ändringar i aktuella regler.

3.3 Analys av marknaden

Då molnbaserade produkter är ett koncept som för närvarande är i växande stadie kan man ofta förutsätta att andra aktörer redan jobbar på en produkt liknande den man själv har för avsikt att utveckla och således är det lämpligt att titta runt på marknaden innan arbete påbörjas. Detta gjordes i projektets första fas och i kombination med identifieringen av regelmotor för implementation genom att undersöka hemsidor relaterade till varje produkt i listan specificerad av Combitech[3] samt ett antal andra produkter som kunde hittas med enkla websökningar efter termer så som; "commercial rule engine", "open source rule engine", "business rule engine", m.m.

Ett par aktörer ansågs stå ut ur mängden med tjänster väldigt lika det som Combitech är intresserade av:

Zapier
Tjänsten kan betraktas som en konkurrent eftersom de enkelt hade kunnat utöka sitt applikationsbibliotek med möjlighet att lyssna på samma typer av dataflöden som är intressanta för detta projekt, därav kan deras prismodeller[21] bli relevanta som referensmaterial.

Ubidots
I tjänsten tillåts användare skapa händelser så som mail- och sms-notifieringar baserat på olika kombinationer av inkommande data men baserat på tillgängliga prismodeller verkar tjänsten huvudsakligen vara avsedd för enskilda personer snarare än t.ex. fastighetsägare. På sidan nämner man dock att man även erbjuder speciallösningar till företag eller för större användningsområden[23] och *Ubidots* kan därför betraktas som en direkt konkurrent till den produkt som skapats under projektet.

3.3.1 Analys av potentiella regelmotorer

Ett flertal regelmotorer analyserades med avseende på relevanta aspekter (så som programmeringsspråk, licens och eventuell prismodell) och de avgränsningar som ställts i kap. 1.4. Avgörande faktorer var även enkelhet (för vidareutveckling) med tanke på det förbestämda slutdatumet för projektet och graden av aktivitet bland andra utvecklare som på något sätt arbetade med samma motor (t.ex. aktivt utvecklarforum). Här följer en summerande lista på några av de produkter som granskades, inklusive de som valdes ut för vidare testning i projektet:

openHAB

Detta är en lösning för universell integrering av produkter för automatisering av hemmet som har färdiga verktyg för att skapa användargränssnitt och enkelt bygga ut systemet men saknar ursprungligt stöd för implementering som molntjänst utöver eventuell möjlighet att exekvera programmet på en virtuell maskin[19], detta medför att denna lösning inte är aktuell för detta projekt.

Konfigurering av mjukvaran i sig är inte något för den genomsnittliga datoranvändaren men den hade teoretiskt kunnat användas för att skapa lösningar likt den som eftersträvats i detta projekt. Dokumentationen är uppdaterad och utvecklarna arbetar fortfarande frekvent med produkten.

NRules

Detta bibliotek är inte anpassat för att köras i molnet och har heller inga funktioner för simultan sessionsexekvering och regelkompilering, däremot är funktionaliteten bland de enklare att förstå för någon som inte är insatt i regelmotorer eller avancerad användning av programmeringsspråket C#. Produkten är väldokumenterad och har ett aktivt utvecklarforum där utvecklaren själv frekvent svarar på frågor, vilket medförde att inlärningstiden upplevdes som realistisk för författarna.

Tolkning av informationen angående regelkompilering i en specifik del av dokumentationen[25] föreslår att det även är möjligt att skapa egna översättare för att konvertera mellan valfria objekt (t.ex. strängar) och fungerande regler och läsa in även detta format vid körning.

Källkoden för *NRules* fanns för senaste versionen av *C#* (*.NET 4.5.2*) och kan användas som bibliotek direkt i ett projekt för *Microsoft Azure Webjob*.

Den senaste versionen fanns även som *NuGet*-paket som kan användas om enklare uppdatering önskas för framtida utveckling, då detta eliminerar behovet av att ladda ner hela källkoden vid uppdatering av denna.

NxBRE
Biblioteket har ett stort utbud av funktioner och kräver viss tidigare erfarenhet med regelmotorer och dess logik vilket författarna saknade som medför en förlängd inlärningstid för *NxBRE*. Dokumentationen för *NxBRE* är ej konsistent och skaparens utvecklarforum var vid tidpunkten inaktivt sedan 2 år tillbaka[27], vilket medförde att inlärningstiden upplevdes för hög för att en produkt skulle kunna levereras på utsatt datum.

Ifall projektet hade varit planerat med mer tid avsatt för inlärning hade denna regelmotor kunnat vara aktuell då den har möjlighet att kompileras med *C# .NET* samt agera i molnet. Dess MIT-licens gör dessutom brukandet av regelmotorn någorlunda fritt.

SRE - Simple Rule Engine
Dokumentationen för motorn är mycket bristande då den inte uppdaterats i takt med att nya versioner av produkten släppts. Produkten i sig har inte uppdaterats på tre år och utvecklarforumet var senast aktivt 2009.

Trots åldern har *SRE* potential att vara en mycket effektiv motor i rätt sammanhang beroende på dess simplicitet. Den har dock begränsningar på operationer som finns tillgängliga vid skapandet av regler, nämligen endast standardoperation så som “lika med”, “inte”, “större/mindre än”, “och/eller” samt de fyra vanliga räknesätten[29].

Motorn kan med mindre antal uppdateringar i källkod kompileras för relevant version *.NET 4.5.2* och användas som bibliotek direkt i ett projekt för *Microsoft Azure Webjob* men har ingen uppenbar metod för konvertering mellan önskad och ursprunglig inläsningsmetod för regler p.g.a. *XML*-formatet. Begränsningarna för operation inom regler ligger begravd i avancerad kod som kräver högre erfarenhet av programmeringsspråket. Dessa faktorer bidrog till att motorn inte blev aktuell för projektet.

BizTalk Services (Azure)
Regelmotorn har väl uppdaterad dokumentation och har aktiva forum för diskussion kring utvecklingen och dess funktionalitet då det är en *Microsoft*-produkt som har ett stort antal användare. Det som gör att motorn inte är aktuell för projektet är kostnaden som är återkommande varje månad och ligger på 3500:- för den funktionalitet som krävdes av projektet och den innehar begränsningar i vilka tjänster den kan kommunicera med inom *Microsoft Azure*, med vissa krav på meddelandeformatet[30].

Huginn
Det finns ingen färdig plugin som fyller de kommunikationskrav som eftersöks i projektet dock är det i teorin möjligt att implementera detta på egen hand samt köra applikationen på en virtuell maskin i molnet, detta är anledningen till varför den inte blir aktuell för detta projekt då tiden för att implementera detta inte var möjlig då det krävdes stora förändringar på regelmotorn för att denna funktionalitet skulle fungera på det sett som efterfrågats. Dokumentationen är väl uppdaterad och det finns ett chattforum där flera av utvecklarna är aktiva.

Drools
Regelmotorn är en del utav ett större paket som gör det svårare att plocka ut regelmotorn och analysera denna för att bestämma ifall den är passande eller inte för detta projekt. Den bryter även mot kravet om att den ska vara möjlig att utveckla vidare med programspråket *C# .NET* och därav inte aktuell för detta projekt.

Code Effects
Den huvudsakliga anledningen till varför denna regelmotor inte är aktuell i detta projekt är dess prismodell där en licens som är aktuell för detta projekt hade motsvarat ca 33.000:- i inköpspris. Utöver prismodellen är den användbar som referensmaterial och en regelmotor som går att använda för eventuella andra projekt med liknande krav och en större framtidsplanering där det finns en säkerhet till att projektet kommer ge en inkomst.

Övriga
Utöver ovan lista undersöktes en mängd andra produkter[3] där det genom enkel undersökning av dess hemsidor direkt kunde konstateras att de var irrelevanta för projektet på grund av licenser som inte fungerade kommersiellt alternativt var avsedda för andra syften än det som eftersöktes i projektet.

3.3.2 Licenser
Vid användande av mjukvara som utvecklats av någon annan går denna oftast under en specifik licens som valts av utvecklaren beroende på hur stor frihet denne vill ge till eventuella implementatörer av produkten. Vissa licenser tillåter totalt fri användning medan andra kräver att specifika krav följs så som t.ex. betalning av en licensavgift till utvecklaren, att produkten endast får användas i icke-kommersiella syften eller att den nya produkten måste distribueras med samma licens. Det kan även ställas krav på dokumentation där originalutvecklaren vill attribueras som delaktig.

Några vanliga licenser som stöttes på under projektet när det handlat om öppen källkod var *MIT* och *EPL*, där *MIT*-baserade produkter prioriterats då denna licens varit mest lämpad med avseende på Combitechs önskemål.

MIT - The MIT License
Denna licens betyder kort att det är tillåtet att bruka koden fritt och det är tillåtet att förändra den efter eget behov och även sälja mjukvaran under ett eget namn eller inuti en annan produkt som utnyttjar produkten som är licenserad med en *MIT* licens. Kravet licensen ställer på nyttjandet är att licensen ska följa med i mjukvaran och minst beröra den implementerade koden som går under licensen innan användandet.

Mjukvaran levereras “som den är” och det lämnas inget löfte om att den är funktionell eller stabil, licensen inkluderar även att skaparen/skaparna och ägaren av mjukvaran inte kan hållas ansvariga för eventuella fel som kan uppstå vid användandet av mjukvaran, utan detta är något nyttjarna av koden själva får ansvara för[35].

EPL - Eclipse Public License
Denna licens är till största del ganska fri men innehavaren av mjukvaran som används kan förändra licensen och modifiera den efter sin egen vilja och uppdatera licensen under tidens gång. Det är ingen fast licens. Det är

fritt att sälja och distribuera produkten som innehåller denna licens men inga garantier ges på att den fungerar[36].

Denna licens medför att varje licens måste kontrolleras innan användning för att undersöka speciella villkor som gäller den specifika mjukvaran så att produkten kan användas och distribueras på det avsedda användningsområdet. Detta kan inkludera försäljning, användning internt hos företag alternativt som att föra reklam för ett företag med en produkt som inte är tänkt att säljas.

Apache Software License 2.0
Denna licens är av typen fri att använda med restriktioner på hur man använder den. Det är fritt att modifiera mjukvaran och dess komponenter men endast ifall detta märks ut att det inte följer den mjukvara som var från ursprunget vid vidare distribuering. Vid vidare distribuering gäller även att denna licens måste följa med för de delar som tillhör produkten och ska vara i läsbart skick[37].

Skaparen av produkten äger minst 50% av den biten som tillhör produkten som använts, det måste även nämnas att det är denne produkt som används och får namnet på produkten får inte brukas i någon annan form än informativ för att denne finns i mjukvaran som distribueras. Patent får ej tas på en produkt som nyttjar denna mjukvara utan avtal med skaparen/skaparna i skriftlig form.

Med mjukvaran som distribueras med denna licens lämnas inga garantier på att den är funktionell eller kommer vara funktionell.

3.4 Källkritik

Då en majoritet av informationen som samlats under projektets gång har haft sitt ursprung på olika websidor har det varit av hög vikt att betrakta dessa med kritiskt öga. När man läser beskrivningar av en produkt skapade av samma personer som är ansvariga för produkten är det troligt att dessa inte ser framlyftandet av negativa aspekter som en prioritet. Dock har källgranskning i de flesta fall varit irrelevant då det handlat om att identifiera produktfunktionalitet och hämta guidande information direkt från officiell produktdokumentation och manualer i vilka det inte finns incitament för lögner då felaktig information skulle leda till en icke fungerande produkt.

Rapporten inkluderar referenser till ett fåtal inlägg i blogg-format där författarna har uppenbar teknisk bakgrund men också bara beskriver fakta som inte direkt kan betraktas som åsiktbaserad. Under projektets utvecklingsstadier har ett flertal öppna forum använts för information om funktioner i programmeringsspråket *C# .NET* i samband med användning i *Microsoft Azure*, i vilka fall det snabbt upptäckts om informationen varit felaktig då detta uppvisats direkt i tester av mjukvaran.

4 Resultat

I detta stycke beskrivs de huvudsakliga resultat som nåtts under projektets gång. Flera mindre program och komponenter för testning och simulering utvecklades men kommer inte att beskrivas här då de inte är del av slutprodukten.

Den slutgiltiga programvaran som levererades till Combitech var en fungerande lösning för implementering i *Microsoft Azure*, innehållande funktionalitet för mottagning av information från sensorer och utgående kommunikation via olika element inom *Azure*-plattformen. Regelmotorn *NRules* integrerades i lösningen och fungerande enligt följande scenario:

1. Regeldata (när något ska hända och vad som ska hända) läses in från en databas i sträng-format.
2. Inläst data konverteras till ett internt regel-format och kopplas samman med en följdaktion (vad som ska hända) som motsvarar en specifik funktion i programmet. Simplifierat exempel: Om "väder" = "regn" → "använd paraply", där "väder" är ett attribut man förväntar sig finnas i den sensor man är intresserad av, "regn" ett av dess möjliga värden och "använd paraply" en funktion i programmet som utför någonting.
3. Konverterade regler lagras i en lista och inväntar kompilering till en session (sessionsvariabel). Kompileringen och hanteringen av sessionen sköts av *NRules* och är ett sätt för att optimera senare matchning mellan inkommande värden och regelkriterier.
4. Programmet tar emot information från sensorer via en *Azure IoT Hub* (och/eller andra simultana källor) i *JSON*-format, gör ett försök att konvertera denna information till en specifik objekttyp och adderar sedan denna till sessionen.
5. Sessionen får signal om att tillgänglig information har uppdaterats och försöker exekvera alla tillgängliga regler. För de regler där kriterierna nu ger positivt resultat anropas dess följaktioner.
6. Varje följdaktion exekveras innan programmet återgår till väntande tillstånd. Dessa är t.ex. meddelanden till andra komponenter inom *Microsoft Azure* så som *Storage Account Blob* och *Storage Queue Triggers* som andra applikationer kan lyssna på och i sin tur kan hantera utskick av mail, SMS, mm, beroende på vad användaren är intresserad av.

Programvaran har inget inbyggt stöd för utgående kommunikation i andra former än de som specificeras av följande underkapitel (så som mail och SMS), detta då all programexekvering måste utföras i molnet där resurserna per applikation är begränsade och detta skulle påverka kapaciteten för mottagning av sensorinformation negativt.

Figuren (fig.1) nedan visar ett typiskt upplägg för ingående produkter, där variationer kan förekomma då flera delar stödjer operationer mot olika tjänster.

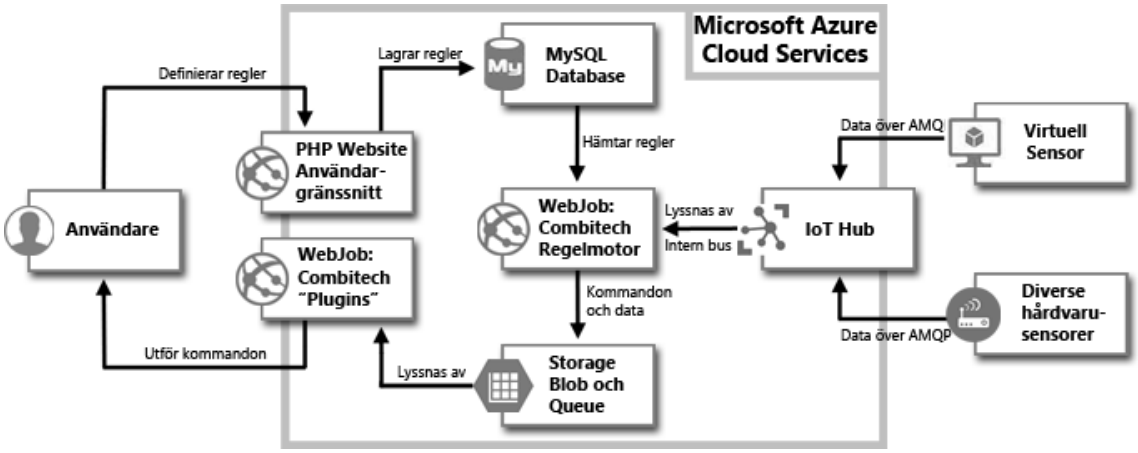

Fig.1: Ingående tjänster och upplägg

4.1 Val av regelmotor

Baserat på diskussion med Combitech, analysresultat och utvalda huvudkrav kunde listan av potentiella kandidater av regelmotorer för vidare testning begränsas till *Simple Rule Engine (SRE), NxBRE, NRules* och *Azure Stream Analytics*. Dessa hade en möjlighet till att vara aktuella för projektet huvudsakligen för att de hade möjlighet till att implementeras eller redan var implementerade mot *Microsoft Azure* och hade en rimlig prisplan. *SRE* blev exkluderad huvudsakligen p.g.a. dess dåliga dokumentation och dess simpla regelvillkor då regelvillkoren gärna skulle kunna utvecklas vidare eller redan ha ett stort urval av möjliga villkor. *NxBRE* blev inaktuell p.g.a. inlärningstiden som bedömdes för extensiv samt att dokumentationen var ej konsistent och föråldrad. *Azure Stream Analytics* hade kunnat vara väl lämpad för detta projekt om inte uppdateringshastigheten för tillägg av nya regler och modifikationer i

princip var oförutsägbar (denna kunde ta allt ifrån några sekunder till flertalet minuter).

NRules var den regelmotor som blev utvald för att den gick under *MIT*-licensen, vilket medförde att den gick att bruka utan att några vidare kostnader måste betalas till skaparen. Regelmotorn hade dessutom det mest öppna interfacet där jämförelsesatserna för vilka regler som kan hanteras har möjlighet att bli placerade utanför regelmotorns källkod, vilket tillåter hantering av villkoren på friare sätt jämfört med övriga undersökta motorer och därmed enklare vidare utveckling och utökning. Utöver detta var den väldokumenterad och bedömdes ha lägst inlärningstid, vilket förenklade inlärningsprocessen för författarna.

4.2 Huvudprodukt och molnimplementation

Grunden till huvudprogrammet skapades med hjälp av *Microsoft Azure SDK*[2] och *Visual Studio 2015*, genom användning av de medföljande guiderna för *Azure webjob* och *Azure application* vilka underlättar uppladdning till molnet då specifik filstruktur appliceras automatiskt. Detta utökades sedan med lämpliga klasser och interface för de olika typerna av I/O-operationer som önskades av Combitech (kap. 3.2) där varje ny funktion testades både enskilt och tillsammans med övriga komponenter under flera olika användarscenarion för att garantera stabilitet i produkten. Vid påträffande av felkällor modifierades berörda funktioner efter hand för att motverka samlande av problem på hög.

Vid testning av funktionerna i *NRules* visade det sig att den resurs- och tidskrävande faktorn var kompilering av regler till sessionsvariabel, medan exekvering av regler mot fakta visade sig relativt snabb. Detta blir problematiskt då sessionsvariabeln är det objekt man använder för att utföra återkommande operationer inom motorn för att avgöra när en regel blir positiv och ska utföras av programmet. Således vill man inte blockera variabeln under tiden kompileringen utförs. Testet som utfördes var en simulering av 10.000 användare med varsin unik regel samt en gemensam regel, vilken tvingades till att aktivera var tionde sekund genom att mata motorn med fakta som motsvarade vad regeln förväntade sig.

Implementationen av *NRules* lyckades men för att uppnå önskad funktionalitet i huvudprogrammet fick flera tillägg göras, såsom bl.a. funktioner för att läsa in strängbaserad regeldata från en databas och omvandla denna till regler som kunde användas i *NRules* samt hantering av flera programtrådar för att kunna kompilera modifierade regler samtidigt som existerande regler fortsatte exekveras. Samtliga tillägg som gjordes testades i programmet på molnet och visade positiva resultat.

Flertalet tester utfördes mot programmet när det väl var uppladdat i *Microsoft Azure* med hjälp av både hårdvarusensorer och virtuella sensorer och programmet visade på lyckad mottagning från samtliga källor. Regler kunde läsas in från en databas, exekveras i motorn mot information tillhandhållen av sensorerna, varefter följdaktioner utfördes i enlighet med de kriterier som bestämts i reglerna.

Följande figur (fig.2) ger en simplifierad bild av huvudfunktionerna i applikationen som implementerades för exekvering i molnet.

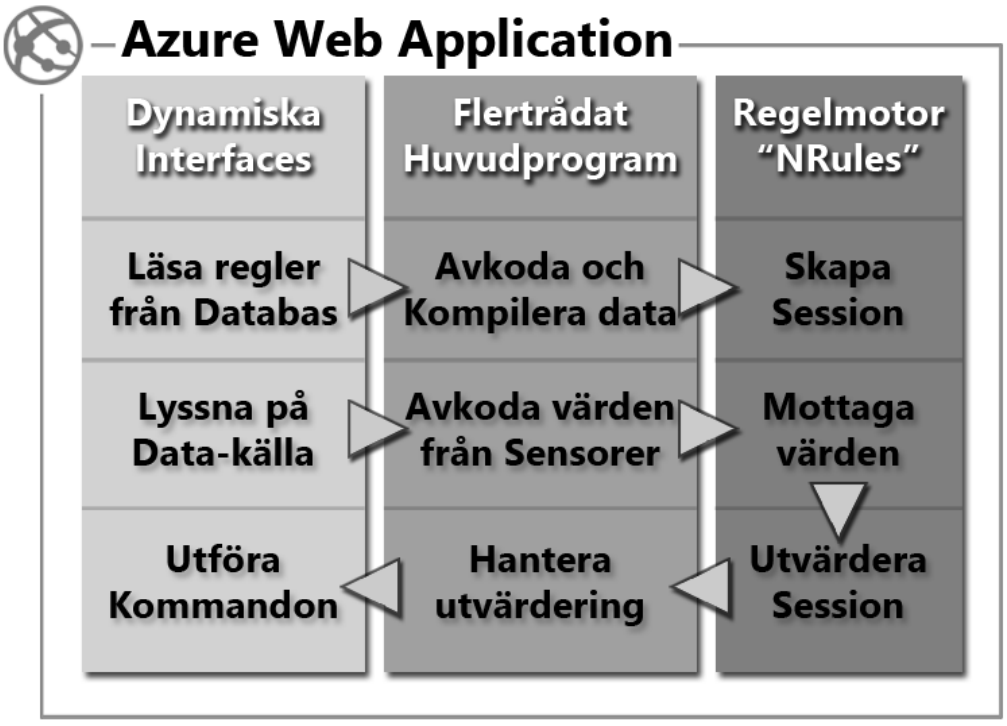

Fig.2: Applikationen som implementerades

4.3 Användargränssnitt

Användargränssnittet som skapades är baserat på *PHP* och kommunicerar med en *MySQL*-databas där samtliga regler och händelser lagras för inläsning av regelmotorn. *MySQL* valdes då detta är en tjänst som redan finns tillgänglig i *Microsoft Azure* och lagringen utförs för att göra det möjligt till förändring av regler utan eventuellt störande kommunikation direkt mot regelmotorn. Det tillåter skapandet av kopplingar mellan användare och regler för presentation i användargränssnittet samtidigt som det tillåter regelmotorn att endast hämta den information som krävs för att bygga regeluppsättningar (där t.ex. användarnamn kan vara överflödigt) och således minskar mängden data som behöver hämtas.

Användargränssnittet är en hemsida där användaren har möjlighet till sitt eget konto som denne loggar in med för att kunna lägga till regler och händelser alternativt ta bort dessa. Funktionalitet för att kunna överskåda samtliga regler och dess händelser finns även implementerat.

Syftet med hemsidan är endast att visa på en möjlig typ av webbaserat användargränssnitt som kan kommunicera med regelmotorn för tillägg och modifikationer av regler och följdaktioner.

I den levererade versionen saknas stöd för att initiera regeluppdateringar mot regelmotorn direkt från hemsidan, detta för att buggar uppdagats mellan konverteringen från *PHP* till *C#,* där meddelanden som skickas över *Azure Storage Queue* inte avserialiseras på korrekt sätt. Dessa problem är återkommande i flera fall och går inte åtgärda då felet ligger i den låsta programvaran som *Microsoft* levererar.

Följande två bilder visar sidorna för att lista samtliga skapade regler samt skapa regler och specificera vilken följdaktion som utförs och dess kriterier och tillhörande användare.

Rules

UserID	RuleID	Conditions	Actions
21	251	Type == FEED;Fields ReedState == true;Fields PirState == true	IoTSender{Raspberry{ "DeviceId": "RaspberryPi", "Sensor": "PIR", "CommandType": "1" }}
21	261	Type == FEED;Fields ReedState == false;Fields PirState == false	IoTSender{Raspberry{ "DeviceId": "RaspberryPi", "Sensor": "PIR", "CommandType": "0" }}
21	271	Type == FEED;Fields ReedState == true;Fields PirState == false	IoTSender{Raspberry{ "DeviceId": "RaspberryPi", "Sensor": "PIR", "CommandType": "2" }}
21	281	Type == FEED;Fields ReedState == false;Fields PirState == true	IoTSender{Raspberry{ "DeviceId": "RaspberryPi", "Sensor": "PIR", "CommandType": "3" }}
21	282		

Fig.3: Lista av de skapade regeluppsättningarna

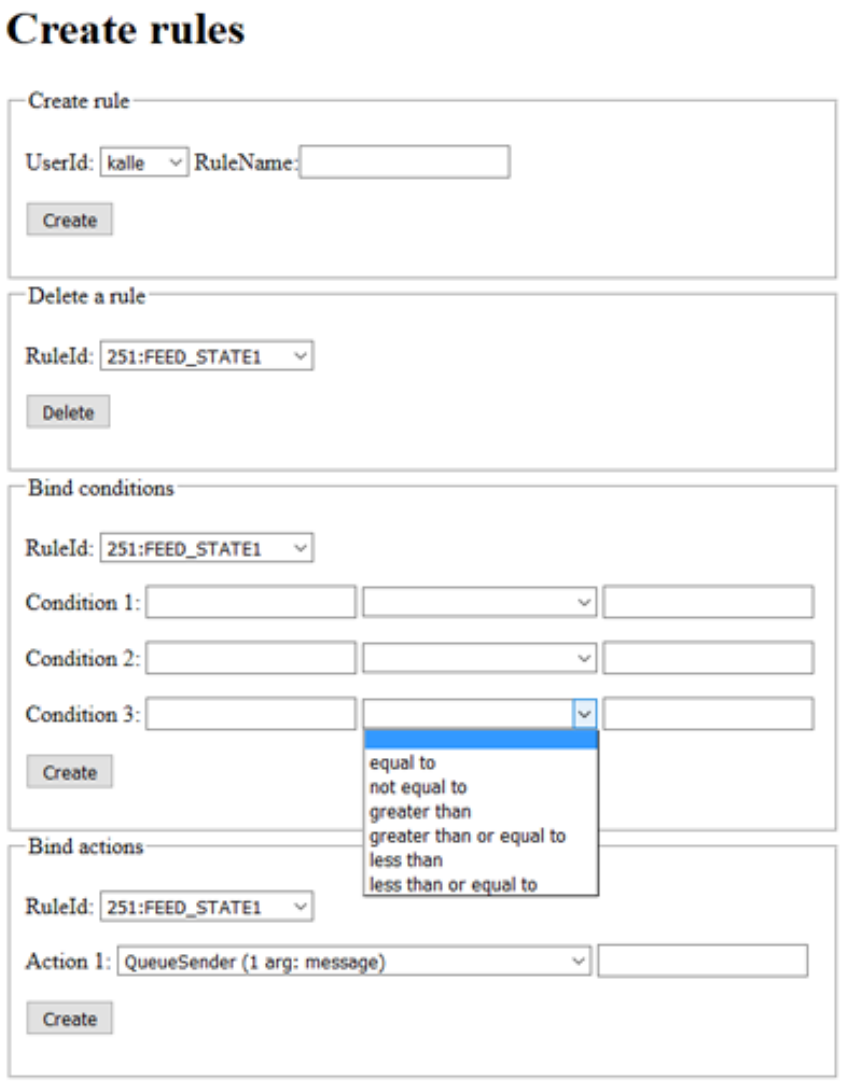

Fig.4: Användargränssnittet för att skapa regeluppsättningar

4.4 Kravuppfyllnad

Programmet implementerar regelmotorn *NRules* som går under *MIT*-licensen och både motorn och själva programmet är skrivna helt i *C# .NET* och byggda som ett *Microsoft Azure Webjob* och fungerar även som *Microsoft Azure Application*(krav 1, 2, 4, 5 och 6). Programmets stöd för inläsning av regler, mottagning av sensordata i *JSON*-format och utmatning av data är skapat med hjälp av olika interfaces vilket innebär att dessa delar enklare kan utvecklas eller bytas ut internt (krav 14) och en majoritet av funktionerna i programmet har medföljande kommentering som beskriver funktionens syfte (krav 3).

Den levererade versionen har stöd för det av Combitech specificerade *JSON*-schemat för sensorer som kan läsas via *Azure IoT Hub*, *Azure Storage Queue* eller *Azure Event Hub* och inläsning från *MySQL*-databas som placerats i *Microsoft Azure* (krav 7, 8, 9 och 10). Vid exekvering av regler kan följdaktioner uträttas mot *Azure Storage Queues*, *Azure IoT Hubs* och *Azure Storage Blobs* (krav 11 och 12). Vid kompilering eller omkompilering av regler görs detta av en separat tråd som inte stör huvudfunktionaliteten i programmet (krav 15).

Det är inte möjligt att genom manipulering/”*code injection*” få programmet att utföra kommandon som annars endast skulle vara tillgängliga för högre nivå av användare så som administratörer (krav 13). Vid inläsning av data som normalt skulle avbryta programmet eller få det att krascha ignoreras detta istället.

5 Slutsats

En stor del av tiden som spenderades på detta projekt gick till att läsa dokumentation kring olika regelmotorer och prova dessa i simulerad miljö, var vid även förväntat enkla moment så som kompilering och exekvering av programmen snarare kunde visa sig problematiska då flera av motorerna förutsätter viss förkunskap inom "*business rule logic*". Programmeringskoden för de flesta lösningarna håller komplext nivå och skulle kräva att avsevärd tid avsatts för inlärning, något som hade blivit problematiskt då det handlade om ett helt nytt programmeringsspråk i kombination med en deadline.

Anledningen till varför *NRules* blev basen för detta projekt (problemformulering 1) var att den var kostnadsfri och fri att använda i kommersiella syften då den går under *MIT*-licensen[24], vilket var av intresse för Combitech. Det gick dessutom att genom testning avgöra att den hade möjlighet till de modifikationer som skulle krävas för att uppnå kraven för *Microsoft Azure* och en stor del av projektet. Tester visade också att motorn var kapabel att hantera ett antal regler motsvarande den mängd användare som av Combitech ansågs vara lämplig för systemet.

Nackdelar med *NRules* var att den inte hade ett förimplementerat stöd för att initiera nya regler utan avbrott däremot var det möjligt att lägga till detta stöd med hjälp av flertrådad uppdelning av programlogiken. Mjukvaran var heller inte avsedd för placering i molnet men var möjlig att modifiera för att införa detta stöd och lägga till funktioner för att hämta in data relaterad till sensorer, regler och följdaktioner, vilka kunde implementeras på ett sätt som tillåter enkel utbyggnad av systemet.

Kraven som ställs vid integrering med *Microsoft Azure* (problemformulering 2) uppfylldes delvis automatiskt genom användandet av *Microsoft Azure SDK* då denna tillgodoser korrekt filstruktur och hjälpmedel för uppladdning till molnet. Faktorer som dock måste beaktas innefattar bl.a. det faktum att prestandan för tjänsten i form av *Microsoft Azure Webjob* beror på prismodellen; i den version som är gratis garanteras inte att programmet tillåts exekvera kontinuerligt utan kan avbrytas, var på det måste startas manuellt via användargränssnittet för *Microsoft Azure*.

Efter dialog med Combitech bestämdes det att ett fullt fungerande och användarvänligt gränssnitt inte var nödvändigt (problemformulering 4, 6)

utan kunde skapas endast för att påvisa möjligheten till denna typ av funktionalitet. Detta ledde till att information angående användare och deras regler lagrades som textsträngar i en *MySQL*-databas (problemformulering 3) och endast minimalt med tid spenderades på utvecklingen av användargränssnittet som skulle kommunicera med denna databas. Gränssnittet tillåter tillägg och borttagning av regler men dessa måste skrivas in med ett specifikt format för att accepteras av programmet och är således inte användarvänligt eller avsett som del i en slutprodukt.

Eftersom inget komplett användargränssnitt utvecklades blev det också mindre relevant att lägga tid på inbyggda notifikationstjänster (problemformulering 5), i synnerhet när testning visade att detta kunde störa huvudfunktionaliteten i programmet och hellre borde hanteras av en separat applikation, eventuellt ett annat *Microsoft Azure Webjob* som agerar inom samma konto. Detta var anledningen till att regelmotorn istället utökades med en specifik följdaktion avsedd för detta syfte, d.v.s. en funktion som kan väljas när en regel skapas, som kommunicerar med ett specifikt element i *Microsoft Azure* och sedan kan läsas av en separat applikation för att simulera t.ex. mail eller SMS-utskick.

I projektet eftersöktes möjligheten att jämföra typer av okänd datatyp med varandra, något som i praktiken blir problematiskt då programmet vid något tillfälle i koden ändå måste få veta vilken typ av jämförelse som ska utföras då resultatet skiljer sig mellan t.ex. strängar och nummer. Detta fick hanteras genom att låta programmet alltid försöka konvertera den okända typen till ett numeriskt värde, vilket i längden kan bli ett problem då detta måste utföras för varje värde och varje regel när ny sensorinformation tas emot. För att komma undan detta problem bör man definiera datatyperna i det *JSON*-schema som används och se till att sensorer också följer korrekt format. Anledningen till varför man inte kan jämföra strängar mot varandra på samma sätt som nummer är att funktionen för strängar utförs genom att dels jämföra tecken för tecken och där se ifall något är större eller mindre och dels jämföra antalet tecken i strängen. Detta betyder att ett värde som t.ex. 5 kan bli större än värdet 70 i en strängjämförelse medan det i en numerisk jämförelse givetvis är mindre.

För att utföra tester på mjukvaran och dess funktioner krävdes vissa hjälpapplikationer för att läsa av *Azure Storage Queues*, *Azure Storage Blobs* och *Azure IoT Hubs* samt kunna skicka data till dessa

(problemformulering 4, 6). De tester som behövde utföras var att kontrollera att regelmotorn kunde ta in data från sensorer och detta utfördes genom att sända in både felaktig och korrekt data till den *Azure IoT Hub* som regelmotorn lyssnade på. För att kontrollera att den faktiskt tog emot datan kunde man t.ex. göra en enkel regel som reagerade på all form av inkommande data och var associerad med en följdaktion som sände ut ett nytt meddelande som kunde läsas av användaren. Således kunde både inkommande och utgående kommunikation påvisas som funktionell.

De följdaktioner som skapats i programmet ger tillgång till flera olika typer av utmatning och kan väljas genom användargränssnittet, bl.a. finns en aktion som är avsedd att användas som en sorts dynamisk kommunikation mot separata applikationer, genom kopplingar som kan göras i användargränssnittet och inte kräver att huvudprogrammet kompileras om (problemformulering 3). Detta är möjligt för att man i användargränssnittet definierar målet i form av olika typer av köer inom *Microsoft Azure* som andra applikationer kan lyssna av och fungerar således lite som ett plugin-system, vilket bidrar till den eftersökta dynamiken i programmet. Bland de tillgängliga följdaktionerna finns även en funktion för att kommunicera mot en enskild enhet som är uppkopplad mot samma *Azure IoT Hub* som programmet lyssnar på, förutsatt att man har tillgång till denna enhets ID. Denna funktionalitet efterfrågades av Combitech för att t.ex. kunna signalera till en enhet när en viss temperatur är uppnådd ska uppvärmning avslutas.

Under projektets gång har buggar i *Microsofts* mjukvara upptäckts där bl.a. kommunikation mellan *PHP* och *C#* via *Azure Storage Queue* och användning av de förbyggda funktionerna för *Queue triggers* ger upphov till systemkrascher då *C#*-applikationen misslyckas med avserialiseringen av meddelanden. Eftersom detta handlar om ett fel i *Microsoft Azure SDK* finns det inte möjlighet att utföra en *try-catch* på detta vilket medför att en krasch blir oundviklig ifall data med oväntat format hamnar i kön. För att undvika detta problem tills *Microsoft* åtgärdat felet är det möjligt att utveckla en egen lyssnare på de köer som finns och där utföra en *try-catch* för att fånga upp undantagsfallet som genereras.

6 Framtida utvecklingsmöjligheter

För att underlätta vidare utveckling har flera delar i programmet skapats med hjälp av interfaces, vilket innebär att dessa delar kan bytas ut mot nya genom att implementera lämpligt interface och ersätta ett fåtal rader kod i huvuddelen av programmet. Detta kan dessutom göras ytterligare dynamiskt genom att tillåta förändringar mellan aktiva interfaces genom ett användargränssnitt. Relevanta delfunktioner där detta appliceras; uppkoppling mot databas, avlyssning från molnet, utgående kommunikation mot molnet.

Ytterligare tillägg kan göras i form av utökande av tillgängliga följdaktioner för utförande i samband med regler. Detta kan göras antingen som en plugin mot systemet (d.v.s. separata applikationer) eller helt nya klasser som implementerar interfacet. Funktionalitet för att skicka till valfritt utrymme finns redan ifall man har skapat lagringsutrymme på molnet, då räcker det att länka till dessa i användargränssnittet.
Vid hanteringen av inkommande sensorinformation kan ändringar göras för att tillåta flera olika typer av serialisering eller olika *JSON*-scheman, hellre än bara ett specifikt *JSON*-schema. Detta är delvis implementerat då den objekttyp som datan i nuläget behandlas som också implementerar ett interface som är tänkt att användas för alla former av inkommande data och redan refereras till av de funktioner som utför operationer på objekten. Dock är det inte känt hur begränsad man blir av den inre funktionaliteten i den regelmotor som användes när olika datatyper adderas till samma session.

Det finns även utvecklingsmöjligheter kring vilka jämförelse-operationer som ska vara tillåtna då detta i den levererade versionen är begränsat till ett fåtal basoperationer för testsyften. Detta är definierat som en funktion i det objekt som används för inkommande data och är således enkelt att utöka. Exempelvis kan det vara lämpligt att implementera hantering av tidpunkter då sensorer senast uppdaterades och tillåta jämförelser med dessa för att kunna skapa regler där man är intresserad av hur länge en temperatursensor har hållit en specifik temperatur. Större utökningar kan även ge möjlighet att tolka hur stor skillnad det har varit i temperaturen över en viss tidsperiod.

En annan potentiellt intressant förbättring hade varit att tillåta användare koppla in och definiera sina egna sensorer genom att använda en

autentiseringsapplikation/server som kommunicerar mellan regelmotorn och de sensorer som kopplas in och delger dessa med relevant information för att automatiskt skapa en uppkoppling mot regelmotorn via en *Azure Event Hub*. För att detta ska fungera korrekt inuti motorn är det dock viktigt att varje sensor verkligen tilldelas ett unikt ID eftersom det måste gå att skilja på källorna när information uppdateras i motorn.

7 Terminologi

Term	Betydelse
Azure Web Application	Specifik tjänst inom Microsoft Azure som programmeringsmässigt påminner om ett traditionellt kompilerat program.
Bibliotek	Programkod som inte exekverar på egen hand utan är avsedd att användas som en del i ett större program, där man importerar de funktioner man är intresserad av från biblioteket.
Exekvering	När ett program utför de uppgifter som det är skapat för.
Följdaktioner	Term som inom detta projekt refererar till specifika funktioner i form av unika klasser som kan utföras till följd av att vissa villkor uppfylls.
Interface	Ett skal som definierar hur en klass ska byggas. Används t.ex. där man vill skapa funktioner som kan hantera flera olika typer av objekt (som samtliga implementerar samma interface).
IoT (Internet Of Things)	Term som används för att beskriva delar av internet där automatiserade tjänster rör sig, t.ex olika typer av hemelektronik som utför någon funktion med hjälp av internet.
JSON	JavaScript Object Notation. En standard för hur en mängd data ska formateras. Används för att garantera att en mottagare hanterar datan på det sätt som sändaren avsett.

Kompilering	När programkod optimeras till en eller flera filer som kan exekveras av en dator. I detta projekt används termen även för att beskriva det scenario där regelobjekt skapas inför användning i regelmotorn.
Sprintar	Period med bestämd tidsbegränsning där ingående arbetsmoment har bestämts i förväg.
SQL (MySQL)	Structured Query Language - Ett språk som används för att begära specifik data ifrån en databas.
try-catch	Funktion inom objektorienterad programmering som används för att fånga upp undantagsfall som normalt skulle få det aktuella programmet att avbryta sin exekvering.
Virtuell maskin	Programvara som simulerar en hel dator eller operativsystem.
XML	Extensible Markup Language. Standard för hur en mängd data ska formateras.

8 Referenser

[1] Combitech AB, 2016, *Företagsfakta*. Tillgänglig på: http://www.combitech.se/Om-Combitech/Foretagsfakta/ [15 Maj 2016]

[2] Tom Dykstra, 8 Jan 2016, *What is the Azure .NET SDK*. Tillgänglig på: https://azure.microsoft.com/en-us/documentation/articles/dotnet-sdk/ [13 Mar 2016]

[3] Adam Temple, 25 Feb 2016, *Newest automation/IOT software list*. Tillgänglig på: https://nebrios.com/blog/software-list-iot-automationworkflow-rule-engines [20 Feb 2016]

[4] Geoffrey Wiserman, 19 Jun 2006, *Real-World Rule Engines*. Tillgänglig på: http://www.infoq.com/articles/Rule-Engines [23 Feb 2016]

[5] Microsoft, 2016, *Microsoft Azure: Cloud computing Platform & Services*. Tillgänglig på: https://azure.microsoft.com/ [23 Feb 2016]

[6] Eric Griffith, 3 Maj 2016, *What is cloud computing?* Tillgänglig på: http://www.pcmag.com/article2/0,2817,2372163,00.asp [9 Maj 2016]

[7] Microsoft, 2016, *How Azure pricing works*. Tillgänglig på: https://azure.microsoft.com/en-us/pricing/ [25 Feb 2016]

[8] Jeff Strokes, 5 Mar 2016, *What is Stream Analytics?* Tillgänglig på: https://azure.microsoft.com/en-us/documentation/articles/stream-analytics-introduction/ [20 Mar 2016]

[9] Microsoft, 29 Sep 2015, *Stream Analytics Query Language reference*. Tillgänglig på: https://msdn.microsoft.com/library/azure/dn834998.aspx [20 Mar 2016]

[10] Seth Manheim, 12 Apr 2016, *What is Azure Event Hubs?* Tillgänglig på: https://azure.microsoft.com/en-us/documentation/articles/event-hubs-what-is-event-hubs/ [22 Apr 2016]

[11] Microsoft, 2016, *Azure IoT Hub*. Tillgänglig på: https://azure.microsoft.com/en-us/services/iot-hub/ [22 Apr 2016]

[12] Tamra Myers, 24 Feb 2016, *Introduction to Microsoft Azure Storage.* Tillgänglig på: https://azure.microsoft.com/en-us/documentation/articles/storage-introduction/ [03 Maj 2016]

[13] Tamra Myers, 25 Apr 2016, *Get started with Azure Blob storage using .NET*. Tillgänglig på: https://azure.microsoft.com/en-us/documentation/articles/storage-dotnet-how-to-use-blobs/ [25 Apr 2016]

[14] Microsoft, 4 Aug 2015, *Understanding Block Blobs, Append Blobs, and Page Blobs*. Tillgänglig på:https://msdn.microsoft.com/library/azure/ee691964.aspx [05 Apr 2016]

[15] Tamra Myers, 29 Apr 2016, *Get started with Azure Table Storage using .NET*. Tillgänglig på: https://azure.microsoft.com/en-us/documentation/articles/storage-dotnet-how-to-use-tables/ [3 Maj 2016]

[16] Seth Manheim, 18 Nov 2015, *Azure Queues and Service Bus queues - compared and contrasted*. Tillgänglig på: https://azure.microsoft.com/en-us/documentation/articles/service-bus-azure-and-service-bus-queues-compared-contrasted/ [05 Apr 2016]

[17] Microsoft, 2015, *Azure and Power BI.* Tillgänglig på: https://powerbi.microsoft.com/en-us/documentation/powerbi-azure-and-power-bi/ [25 Apr 2016]

[18] Microsoft, 2015, *C# Coding Conventions (C# Programming Guide)*. Tillgänglig på: https://msdn.microsoft.com/en-us/library/ff926074.aspx [1 Mar 2016]

[19] openHAB, 2016, *Features - Introduction*. Tillgänglig på: http://www.openhab.org/features/introduction.html [14 Mar 2016]

[20] Zapier, 2016, *Developer Platform - Zapier*. Tillgänglig på: https://zapier.com/developer/documentation/v2/ [18 Mar 2016]

[21] Zapier, 2016, *Pricing Plans - Zapier*. Tillgänglig på: https://zapier.com/app/pricing [18 Mar 2016]

[22] Ubidots, 2014, *Data Capturing*. Tillgänglig på: http://ubidots.com/features-data-capturing.html [13 Mar 2016]

[23] Ubidots, 2014, *Our cloud services plans*. Tillgänglig på: http://ubidots.com/pricing.html [17 Maj 2016]

[24] Sergiy Nikolayev, 2016, *Getting Started*. Tillgänglig på: https://github.com/NRules/NRules/wiki/Getting-Started [23 Feb 2016]

[25] Sergiy Nikolayev, 2016, *Creating Rules at Runtime*. Tillgänglig på: https://github.com/NRules/NRules/wiki/Rule-Builder [25 Feb 2016]

[26] David Dossot, 2015, *NxBRE, lightweight Business Rule Engine*. Tillgänglig på: https://github.com/ddossot/NxBRE [24 Feb 2016]

[27] SourceForge, 2016, *NxBRE Discussion Forum*. Tillgänglig på: https://sourceforge.net/p/nxbre/discussion/ [24 Feb 2016]

[28] Sierra Digital Solutions Corp, 11 Apr 2013, *Simple Rule Engine*. Tillgänglig på: https://sourceforge.net/projects/sdsre/ [25 Feb 2016]

[29] Sierra Digital Solutions Corp, i.d, *SRE - Simple Rule Engine*. Tillgänglig på: http://simpleruleengine.tripod.com/ [25 Feb 2016]

[30] Microsoft, 27 Nov 2015, BizTalk Services. Tillgänglig på: https://msdn.microsoft.com/library/hh689864.aspx [29 Feb 2016]

[31] Andrew Cantino, 25 Feb 2016, *huginn: Build agents that monitor and act on your behalf. Your agents are standing by!* Tillgänglig på: https://github.com/cantino/huginn [29 Feb 2016]

[32] Red Hat Inc, 2016, *Drools*. Tillgänglig på: https://github.com/droolsjbpm/drools [3 Mar 2016]

[33] Red Hat Inc, 2016, *Drools - Business Rules Management System (Java, Open source)*. Tillgänglig på: http://www.drools.org/ [3 Mar 2016]

[34] Code Effects Software, 2016, *Code Effects: ASP .NET and MVC Business Rule Engine*. Tillgänglig på: http://www.codeeffects.com [3 Mar 2016]

[35] Open Source Initiative, 2016, *The MIT License (MIT)*. Tillgänglig på: https://opensource.org/licenses/MIT [4 Apr 2016]

[36] Eclipse, 2016, *Eclipse Public License - Version 1.0*. Tillgänglig på: https://www.eclipse.org/legal/epl-v10.html [4 Apr 2016]

[37] The Apache Software Foundation, 2004. *Apache License Version 2.0.* Tillgänglig på: http://www.apache.org/licenses/LICENSE-2.0 [18 Maj 2016]

[38] Seth Manheim, 10 Maj 2016, *Service Bus AMQP overview*. Tillgänglig på: https://azure.microsoft.com/en-us/documentation/articles/service-bus-amqp-overview/ [1 Jun 2016]

[39] Henrik Kniberg, Mattias Skarin, 2010. *Kanban and Scrum - making the most of both (Enterprise Software Development)*. Edition. Lulu.com

[40] Techopedia, 2016, *What is the exploratory model?* Tillgänglig på: https://www.techopedia.com/definition/16383/exploratory-model [14 Jun 2016]

[41] Don Wells, 2016. *Pair Programming*. Tillgänglig på: http://www.extremeprogramming.org/rules/pair.html [14 Jun 2016]