A Reactive Performance Monitoring Framework

Katherine ChengLi

Directed By:
Prof. Liam Peyton

Thesis

Submitted to the
Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements for the degree of

Master of Computer Science

uOttawa

University of Ottawa

Ottawa, Ontario, Canada

June 2016

Abstract

With the ascendency of data and the rise of interest in analytics, organizations are becoming more interested in the use of data to make their business processes more intelligent and reactive. BI applications are a common way that organizations integrate analytics in their processes. However, it can be days, weeks or even months before a manual response is undertaken based on a human interpreting a report. Even when information technology supports automatic responses within an organization, it is often implemented in an ad hoc manner without following a systematic framework.

In this thesis, we present a reactive performance monitoring (RPM) framework which aims at automating the link from the analytical (how well is the operational achieving the strategic) to the operational (the particular process steps implemented within an organization that determine its behavior) aspects of businesses to bypass the strategic (the high level and long term goals an organization is trying to achieve) as needed and reduce the latency between knowledge and action. Our RPM framework is composed of an architecture, a methodology, and a rule environment which permits the redaction of rules possessing relevant conditions and actions. In addition, we present an OLAP rule engine which is demonstrated to be effective in our framework where events are streamed in, reacted upon in real-time, and stored in an OLAP database.

To develop and evaluate our framework, two case studies were undertaken. The first was done using IBM technologies implementing an application made to identify patients at high risk of cancer recurrence. The second was done using open source technologies. With this second implementation, we created an application that has the

goal of informing women from at risk populations of the different stages of pregnancy on a weekly basis.

Acknowledgements

Although the accomplishment of a thesis is a personal achievement, I would certainly not have been able to finish it without the help, support, and patience of my supervisor Prof. Liam Peyton. Thank you for your continued help and encouragement.

Thank you to my family for encouraging me throughout this process and for believing that I would be able to successfully accomplish this. A special thank you to Mathieu Jobin for his never ending support.

In addition, I would like to thank the organizations that helped fund my master's degree. This includes MITACS, NSERC, and OGS.

Finally, thank you to IBM and CENGN for providing me the opportunity to intern with them. In particular, thank you to Randy Giffen from IBM and my Extreme Blue team. It has been a great source of knowledge and experience.

Table of Contents

List of Figures

List of Tables

List of Acronyms

Acronym	Definition
AFMCP	Application Framework for Monitoring Care Processes
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
ASDI	Aircraft Situation Display to Industry
BAIT	Big data Analytics Infrastructure Testbed
BI	Business Intelligence
BPEL	Business Process Execution Language
BPM	Business Process Management
BPMN	Business Process Model and Notation
CEP	Complex Event Processing
CIWS	Corridor Integrated Weather System
CoSPA	Consolidated Storm Prediction for Aviation
CPMA	Care Process Monitoring Application
CPME	Care Process Monitoring Engine
DAO	Data Access Object
DSRM	Design Science Research Methodology
ET	Echo Top
ETL	Extract Transform Load
GUI	Graphical User Interface
HTML	Hypertext Markup Language

HRRR	High-Resolution Rapid Refresh
IBM ODM	IBM Operational Decision Manager
IBM PMQ	IBM Predictive Maintenance and Quality
IBM SPSS ADM	IBM SPSS Analytical Decision Management
IBM SPSS C&D	IBM SPSS Collaboration and Deployment Services
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
KPI	Key Performance Indicator
LDM	Local Data Manager
OLAP	Online Analytical Processing
REST	Representational State Transfer
RPM	Reactive Performance Monitoring
RTLS	Real-Time Location System
SME	State Monitoring Engine
SOAP	Simple Object Access Protocol
VIL	Vertically Integrated Liquid
XML	Extensible Markup Language

1 Introduction

1.1 Problem Statement

With the ascendency of data and the rise of interest in analytics (Brown, 2015), organizations are becoming more interested in the use of data to make their business processes more intelligent and reactive. Traditionally, organizations define operational business processes to achieve their strategic business goals (improve revenues, reduce costs, ensure customer satisfaction, improve quality of results), and then they collect data from those operational business processes into data warehouses where reports are created and the results are analyzed using Business Intelligence (BI) applications to see if the operational business processes are effective in achieving the strategic business goals. Unfortunately, the creation of such BI applications is complex.

Traditionally, BI applications follow Kimball's (Kimball, Ross, Thornthwaite, Mundy, & Becker, 2008) methodology: the requirements are initially collected; followed by the technical architecture and its plan; then design of the dimensional model; then physical and performance database planning; then design and development of the ETL system; finally the design and the development of the BI applications. However, there is little direct linkage between the analysis done in these applications, and appropriate reactions to adjust business processes in response. It can be days, weeks or even months before a manual response is undertaken based on a human interpreting a report. Even when information technology supports automatic responses within an organization, they

are often implemented in an ad hoc customized manner without following a systematic framework.

As Chamney described in his thesis (Chamney, 2014), the creation of BI applications and the accompanying backend is non-trivial. If we also add complex event processing (CEP), or business process management (BPM) to achieve some degree of automation, it can only be imagined that the creation of a seemingly simple application can quickly become complex without careful and controlled planning and execution.

The literature does offer examples of use cases where some sort of analytics framework has been implemented to achieve specific goals. For example, (Kelly, Craig, & Matthews, 2015) created an architecture to provide "analytics-as-a-service for algorithm developers to use when creating and testing new capabilities for air traffic management decision support." Although these different architectures provide a framework for building BI applications they often do not possess a rule environment to directly connect reports and decisions to actions based on incoming events and existing known variables.

This illustrates a lack of bridging between the operational (the detailed steps we follow as a business), and the analytical (how well is the business fulfilling the strategic) aspects of businesses. Typically, feedback is unidirectional with insufficient context. Moreover, feedback is given by people after the fact. This creates latency between having the data that informs us of what needs to be done and actually doing what needs to be done. The use of data for analytics is very disconnected from operational use so that

determining reports from operational data is done in an ad hoc manner, and determining operational actions from reports is also done in an ad hoc manner.

1.2 Thesis Motivation and Contributions

The main goal of this thesis is to provide a framework which will enable organizations to systematically design the relationships and reactions between the analytical and operational parts of business so that businesses are not simply limited to performance management of their operations to determine if strategic goals are achieved, but rather can define reactive performance monitoring so that automatic actions can be triggered operationally in real-time when needed as indicated by analytics.

Technology in the form of business process management exists to automate business processes (van der Aalst, 2013); and technology exists in the form of predictive modelling (zur Mühlen & Shapiro, 2010) to analyze data and predict when actions are needed; and technology exists for complex event processing that can stream data for processing and define rules to trigger actions when determined by a model (Luckham, 2012). It was our initial motivation to combine these technologies in a systematic framework.

Most rule engines used in complex event processing are based on simple object models that are independent of the rich multidimensional OLAP databases that are used in Business Intelligence applications (Abdelfattah, 2013). Another motivation for our work is that more appropriate rule engines and an architecture optimized for reactive performance monitoring could be created if OLAP Databases were more often integrated with complex event processing (Costa, Cecílio, Martins, & Furtado, 2011).

This thesis proposes a Reactive Performance Monitoring (RPM) framework that processes and analyzes events to determine appropriate re-actions across systems that are needed in a scenario where data is gathered, reported on, and acted upon or directed as needed. In particular, RPM is based on a rule environment which can trigger meaningful actions methods with condition functions based on analytics. RPM rests on a data warehousing infrastructure centered around OLAP databases. Ultimately, RPM attempts to close the gap between the strategic, analytical, and operational parts of a business by providing relevant interfaces and feedback mechanisms.

The main contributions of this thesis are the following:

1. A reactive performance monitoring framework (RPM) including:
    a. An architecture pattern which integrates various technologies and structures the event flow
    b. A methodology to build applications using said framework
    c. A rule environment using appropriate condition functions and action methods to respond to operational events based on data from an analytical data warehouse
2. An OLAP Rule Engine

While working on this thesis we were fortunate to be able to collaborate with two different organizations on two different internships. The first one is IBM (IBM Corporation, 2016c), the multinational technology company, where we were able to learn more about their technologies used in our case studies. The second internship was at CENGN (CENGN, 2016). CENGN is a network technologies accelerator and consortium

where we were able to participate in both their projects and underlying infrastructure. We also have one publication so far from this thesis:

ChengLi, Katherine. (2014, November) *Smarter Process Triggers: Predictive Analytics for Smarter Business Operations*. Poster session presented at CASCON 2014 24th Annual International Conference on Computer Science and Software Engineering "Smarter Systems of Interactions", Toronto, Canada.

1.3 Thesis Methodology and Organization

We followed the Design Science Research Methodology (DSRM). As described by (Hevner, March, Park, & Pam, 2004), DSRM is adapted to situations where the creation of an artifact is "intended to solve identified organizational problems". Figure 1 describes the entire process model of the DSRM proposed by (Peffers, Tuunanen, Rothenberger, & Chatterjee, 2007). This is the process we followed in the second iteration. For our first iteration, although still following DSRM, the process followed was a bit less structured.

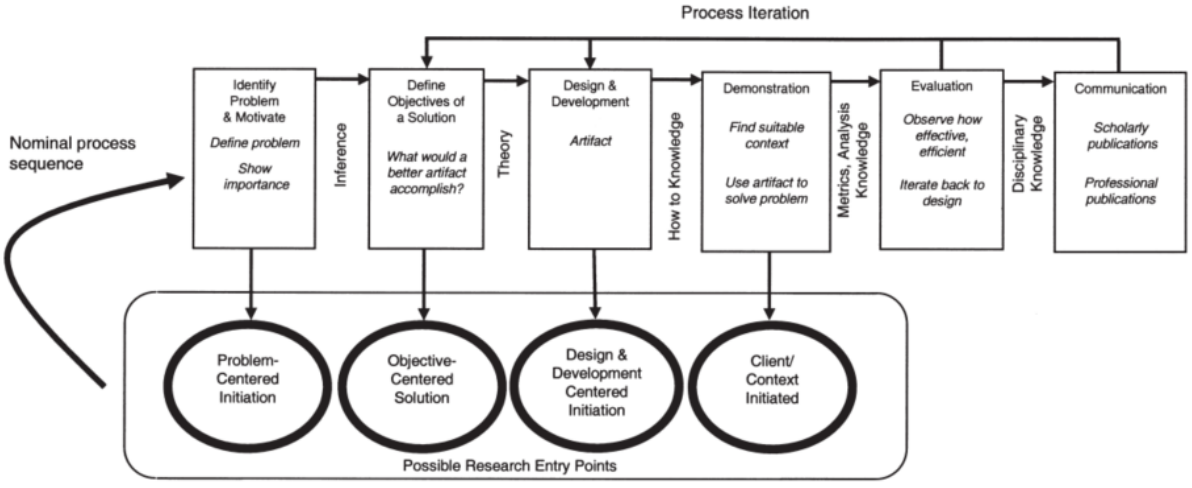

Figure 1: DSRM (Design Science Research Methodology) Process Model (Peffers et al., 2007)

1. Identifying the problem: A problem was identified with the integration of predictive analytics in business processes to bridge the gap between event-based analytics triggers and business processes. The problem was identified through the implementation of a proof of concept with IBM technologies at IBM and the comparison of their solution with current research group activities.
2. Defining the objectives of the solution: The goal of was to reduce the gap between business processes and real-time events.
3. Designing and developing the artifact: To reduce this gap, a framework integrating real-time Fact Events to trigger business processes was designed.
4. Demonstrating the artifact: In this first iteration, the initial problem was context initiated. Our demonstration consisted of a proof of concept made with IBM technologies.
5. Evaluating the artifact: We evaluated the framework comparing it with current practices and with the use of predictive analytics only.
6. Communication: A poster presentation was made called "Smarter Process Triggers: Predictive Analytics for Smarter Business Operations."

For our second iteration, these steps were followed:

1. Identifying the problem: DSRM as defined by (Peffers et al., 2007) starts with the identification of the problem and its definition through which the importance of the problem is shown. In our case, we re-identified the

problem. Businesses try to integrate analytics or big data in their day to day operations but with much difficulty. There is a considerable gap between the strategic and the operational when trying to integrate the results of analytics in real-time. We adapted our initial problem related to Fact Event based triggers for business processes to a broader analytical-operational bridging problem.

2. Defining the objectives of the solution: The second step of DSRM consists of defining the objectives of the solution and defining how the resulting artifact would yield better results. We want to bridge the gap between the strategic, the analytical, and the operational.
3. Designing and developing the artifact: We propose a framework which integrates several technologies in a structured way with defined interfaces and a methodology guiding the construction of the architecture and the integration of each part.
4. Demonstrating the artifact: To demonstrate our framework, we have two example uses of the framework. One uses IBM technologies, and the other uses open source technologies, namely QuickForms. These two examples helped refine the architecture and identify the existing gaps more precisely.
5. Evaluating the artifact: The framework was evaluated, validated, and reviewed by comparing it with literature examples as well as current practices, and the use of a data warehouse only. Our criteria are based on

the evaluation of the gaps between strategic, analytical, and operational parts of a business.

6. Communication: This thesis presents our final results.

As it is usually done in DSRM, our framework was actually re-evaluated multiple times in micro-iterations to be redefined and redesigned.

The thesis is organized as follows. Chapter 1 introduces the thesis. In Chapter 2, we present the underlying concepts needed to understand this thesis as well as the related works. In Chapter 3, we present the problem, our evaluation criteria, and our framework. In Chapter 4, two case studies and some rule examples are presented. In Chapter 5, our evaluation of the framework is given. Chapter 6 concludes our thesis.

2 Background

For a better understanding of this thesis, there are several concepts that should be known beforehand. This chapter is dedicated to those concepts as well as related works that have tried to solve or partially solve the problem stated in Chapter 1 Introduction. We have separated this chapter in 5 sections: Performance Monitoring, Business Process Management, Data Integration Technologies, On-Line Analytical Processing, and Related Works.

2.1 Performance Monitoring

To understand RPM, our reactive performance monitoring framework, it is necessary to understand performance monitoring. This section is dedicated to related concepts. We outline performance management, real-time analytics, predictive analytics, and the concept of an event.

2.1.1 Performance Management

As described by (Pollitt, 2013), performance management is composed of 6 main cyclical elements:

- activities (in business processes)
- measurements of particular aspects of activities (metrics)
- data
- criteria against which to compare the data,
- relevant information obtained from the data,

- and decision making pertaining to the activities (which leads to new activities)

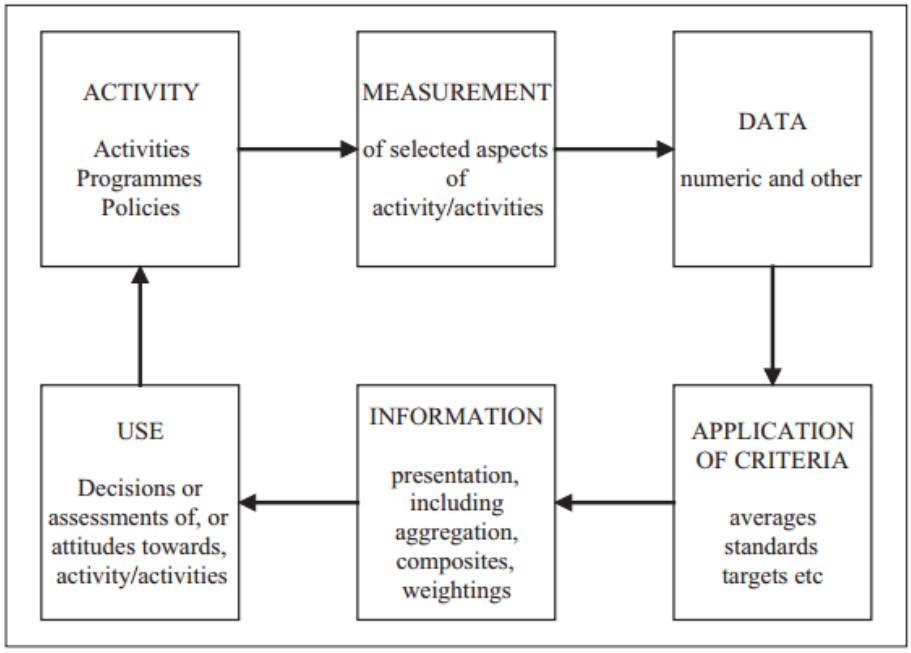

Figure 2: The basic elements of performance management (Pollitt, 2013)

In short, it is used to measure and communicate the success (or lack thereof) of business processes activities using metrics (Middleton, Peyton, Kuziemsky, & Eze, 2009).

2.1.2 Real-time Analytics

Real-time analytics consists of events, or simply data, being stored in real-time and displayed in real-time via a medium such as a dashboard, metrics, or alerts. The display then consists of current and historical data (Chieu & Zeng, 2008). Organizational performance can be measured using real-time analytics comparing fine-grained real-time performance metrics to actual objectives (Mouttham, Peyton, & Kuziemsky, 2011).

2.1.3 Predictive Analytics

"Predictive analytics uses statistical techniques, machine learning, and data mining to discover facts in order to make predictions about unknown future events" (Lechevalier, Narayanan, & Rachuri, 2014). As one of the advantages, this can allow the business to act ahead of time and diminish costs because actions have been undertaken before the system actually fails (Dash, Giffen, Ramakrishnarao, & Sreenivasan, 2013). Predictive analytics lets you predict and act ahead of failures while real-time analytics lets you know the current state of the system.

2.1.4 Event

"An *event* is an *object* that represents or records an activity that happens, or is thought of as happening" (Luckham, 2012). Events received by the RPM framework must have a structure that can be interpreted or transformed by another entity, namely the application management or a message broker.

2.2 Business Process Management

Once again, to further understand the RPM framework, a knowledge of business process management is necessary. This short section will explain what it is by covering what is a business process, a web service, and business process management itself.

2.2.1 Business Process

"A *business process* consists of a set of activities that are performed in coordination in an organizational and technical environment. These activities jointly realize a business goal. Each business process is enacted by a singled organization, but it

may interact with business processes performed by other organizations" (Weske, 2012). We are principally interested by **online** business processes. For example, online business processes using BPEL (Andrews et al., 2003) are easily interpreted because they follow a standard language and follow an executable format. Online business processes such as those using BPEL use web service interfaces exclusively to import and export information (Andrews et al., 2003).

2.2.2 Web Service

The W3C defines web services as "a software system designed to support interoperable, machine-to-machine interaction over a network" (W3C, 2004). Web services can easily be accessed through defined interfaces supporting SOAP and REST (Mumbaikar & Padiya, 2013). With their easy access and machine-to-machine interaction, Web services are meant to enable the communication of heterogeneous applications via a network.

2.2.3 Business Process Management (BPM)

Business Process Management is, as its name says, the management of business processes. From (Pourshahid et al., 2009), it "is the understanding and management of diverse and cross-organizational processes which link humans and automated system together." From (Pourshahid et al., 2009), it is usually done following 5 cyclical steps: Define, Model, Execute, Monitor, and Optimize. In the "Define" step, processes, business objectives, areas of improvement, and others are established. In the "Model" step, modeling of the processes to manage is done. In the "Execute" step, we implement execute the previously designed process models. In the "Monitor" step, the execution is

measured using previously carefully defined business metrics. In the last step, "Optimize", we try to improve the process models using the data gathered using BI methods. (van der Aalst, 2013) simplified these 5 cyclical steps in 3 cyclical phases as described in Figure 4. These 3 cyclical steps are: (re)design, run and adjust, and implement/configure. During the (re)design phase, model-based analysis can be done while during the run and adjust phase, data-based analysis can be done.

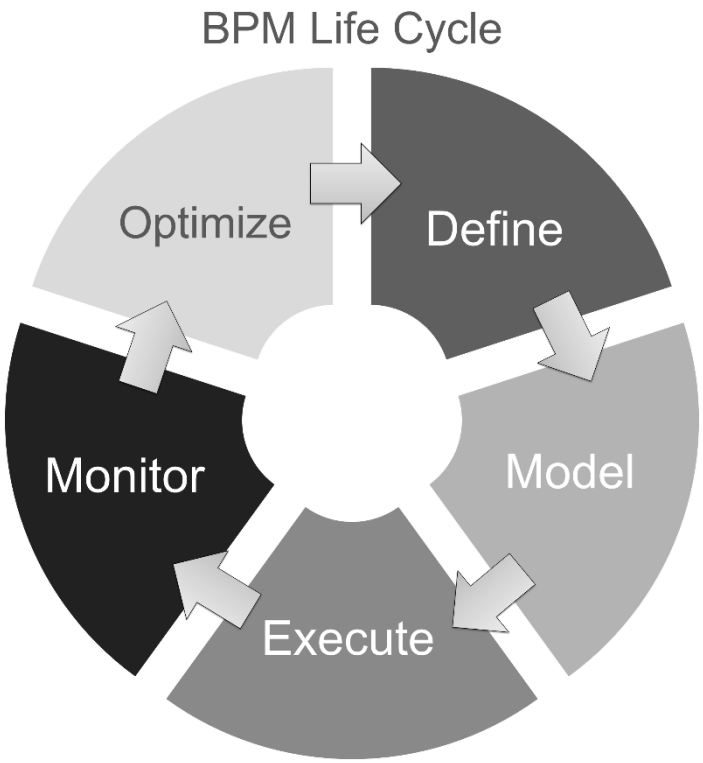

Figure 3: BPM Life Cycle as described by (Pourshahid et al., 2009)

IBM Business Process Manager (IBM Corporation, 2016d) is a tool that acts as a platform for business process management and which supports the importation and exportation using BPMN (Claus, 2011). It permits the creation of business processes and accompanying graphical user interfaces, including mobile. It also possesses tools that

allows the analysis of the processes in place. We have used IBM BPM in our IBM use case for the deployment of immediate actions such as business process triggers.

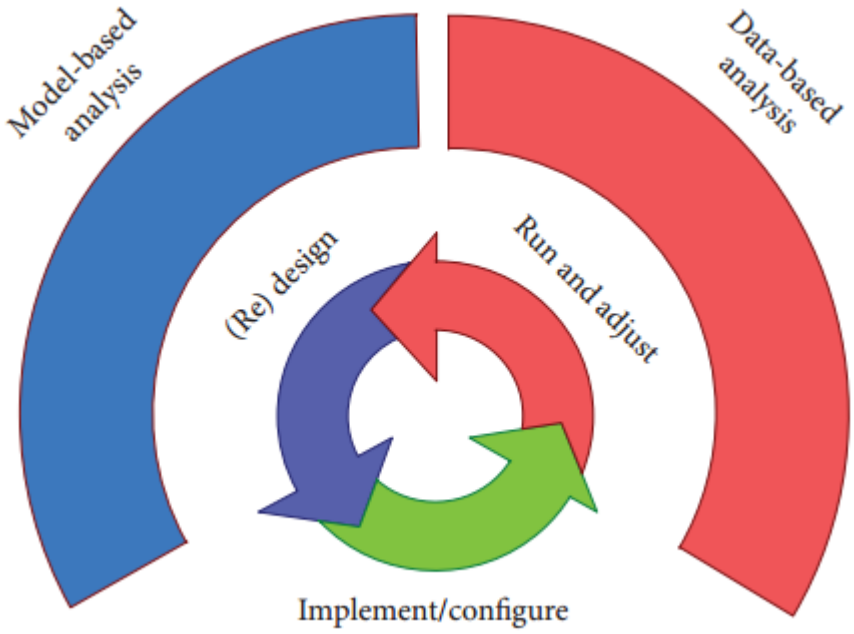

Figure 4: The BPM life cycle consisting of three phases: (1) (re)design, (2) implement/configure, and (3) run and adjust. (van der Aalst, 2013)

2.3 Event Processing Technologies

Our RPM framework depends on event processing technologies. In our framework, we are integrating performance monitoring with business process management in real-time. It would not be possible without event processing technologies. This section presents some of the main concepts such as what is a message broker, an event-driven architecture, complex event processing, and a notification.

2.3.1 Message Broker

In our IBM case study, a message broker was used to transfer events between the different components of our architecture. A message broker is an enterprise integration pattern that can receive different types of data streams, convert them to a desired format, and reroute it to the wanted destination (Hohpe & Woolf, 2003). The implementation used in the case study is the IBM Integration Bus (IBM Corporation, 2016e).

2.3.2 Event-Driven Architecture

In an event-driven architecture, a variety of events originating from different places are collected and processed (Middleton et al., 2009). It is an architecture pattern aimed at systems where there is an abundance of events that need to circulate between entities resulting in the production, storage, and use of events (perhaps to trigger notifications or create more events) (Niblett & Graham, 2005). Message brokers are often used in SOAs and event-driven architectures to transfer events from and to different sources and destinations. Event-driven architectures are often key in real-time and predictive analytics.

2.3.3 Complex Event Processing (CEP)

Complex event processing involves the use of an event-driven architecture. It is "a set of concepts and principles for processing events and methods of implementing those concepts" (Luckham, 2012) which leverages an event-driven architecture. There are several main concepts related to CEP. This includes but is not limited to: the immutability of events, adaptation, filtering, prioritization, computation on event data, event pattern detection, exception detection and handling, event pattern abstraction, event pattern-

triggered processes, and computable event hierarchies (Luckham, 2012). More specifically, it is used to infer high level business events, which are equivalent to complex events, in real-time (Ku, Zhu, Hu, & Lv, 2008). CEP uses relevant events to infer new ones that interest businesses for decision-making (A Baarah, Mouttham, & Peyton, 2011).

2.3.4 Notification

In the context of this thesis, a notification is an event which is used to alert an external party via the storage of a new fact which will be used by an application, or another service that can notify an external entity such as a service able to send an email (Huang & Gannon, 2006).

2.4 On-Line Analytical Processing (OLAP)

OLAP is the basis for analytics and fundamental to our RPM framework. This section will explain some of the associated key concepts such as what is a data access object, business intelligence, a data warehouse, a star schema, and OLAP itself.

2.4.1 Data Access Object

The Data Access Object (DAO) is a design pattern used to abstract the data sources from the applications, or other components (Sun Microsystems Inc., 2002). It is often used in service-oriented architectures and is considered “type-safe” by providing an API and helping achieve encapsulation (Mellqvist, 2006). This permits a complete abstraction of the data source with no reference to its implementation. It also highly decreases coupling with the data source (Sun Microsystems Inc., 2002). In one of our use

cases, we use QuickForms (Chamney, 2014) which implements a DAO to interact with our OLAP Databases.

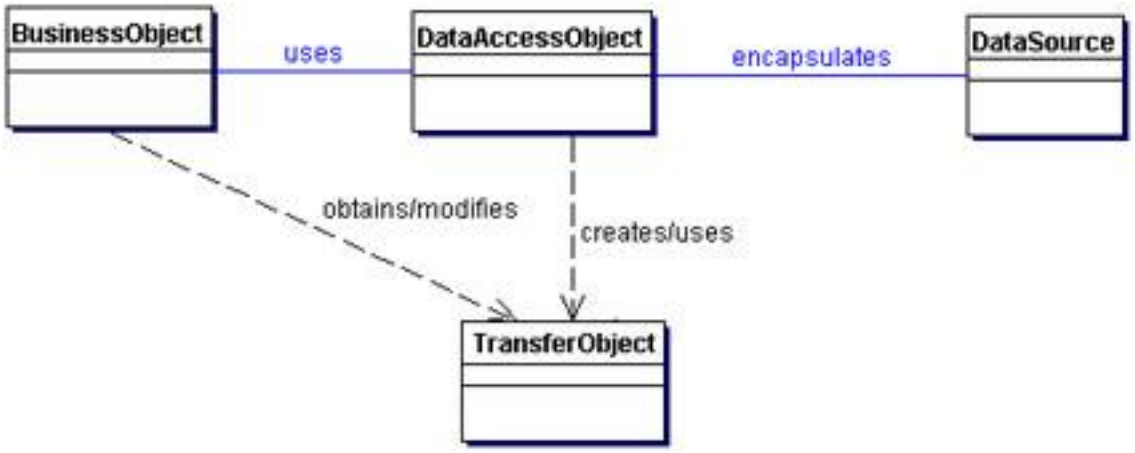

Figure 5: Data Access Object (Sun Microsystems Inc., 2002)

2.4.2 Business Intelligence (BI)

Business Intelligence (BI) refers to processing and display of data for analysis or reporting. It is usually based on data stored in a data warehouse (Kimball et al., 2008). BI Applications, refers specifically to applications providing analytics to users. BI Systems refer to the end-to-end processing of data from operational sources all the way through to supporting analysis of data in BI applications and the delivery of key performance indicators (KPIs) that measure how well strategic goals are being achieved (Exforsys Inc., 2008).

2.4.3 Data Warehouse

As described by (Kimball et al., 2008), a data warehouse is an organized collection of data used as a source of data to support business decision making. The entire process of organizing, collecting and managing such data is called *data warehousing*. Data warehouses must consider how the data is communicated to meet the need of the

business users. They are therefore optimized for reporting and providing a complete view of an organization and its activities. Data is organized in "fact" tables where metrics, measurements, or others are stored. Specific redundant characteristics or attributes of the facts are stored in "dimensional" tables (Kimball & Ross, 2013).

2.4.4 Star Schema

A star schema is defined as a "generic representation of a dimensional model in a relational database in which a fact table with a composite key is joined to a number of single level dimension tables, each with a single primary key" (Kimball et al., 2008). The dimensional model, consisting of denormalized operational tables to flatten the relationships, is optimized for query performance, making it ideal for reporting (Kimball et al., 2008). As a slightly more complicated version of the star schema, there is the snowflake schema. The snowflake schema consists of a star schema where the dimensions having "redundant many-to-one attributes are removed into separate dimension tables" (Ross, 2008). However, as (Ross, 2008) explains, snowflaking is generally discouraged due to reduced query performance.

2.4.5 Online Analytical Processing (OLAP)

"OLAP performs multidimensional analysis of business data and provides the capability for complex calculations, trend analysis, and sophisticated data modeling. It is the foundation for many kinds of business applications for Business Performance Management, Planning, Budgeting, Forecasting, Financial Reporting, Analysis, Simulation Models, Knowledge Discovery, and Data Warehouse Reporting. OLAP enables end-users to perform ad hoc analysis of data in multiple dimensions, thereby

providing the insight and understanding they need for better decision making" (PARIS Technologies Inc., 2015). When the dimensional model is stored in an OLAP database, as opposed to a relational database where we call it a star schema, it is called a cube (Kimball et al., 2008).

2.5 Rule Engine

Rule engines are engines which do data processing by applying a particular logic to it (Chisholm, 2004). Some rule engines specifically do data inference and also apply a wide variety of actions to the data that they process. In the case of business rule engines, the rules of the rule engine apply business rules that implement organizational policy on how data should be interpreted or processed. In this thesis we will be comparing our OLAP rule engine to several other rule engines.

2.5.1 State Monitoring Engine (SME)

A state monitoring engine is a particular type of rule engine designed specifically for performance monitoring (Baffoe, 2013). It integrates complex event processing with the storage of events in an OLAP database for reporting. Again, although, not used in our RPM framework, we compare it to the rule engine that we developed for RPM. The SME to which we will compare the RPM rule engine is a specific performance monitoring complex event processing engine created by Shirley Awusi Baffoe in the context of her master's thesis in Electronic Business Technologies at the University of Ottawa (Baffoe, 2013). One specific interesting capability of her SME is the generation of events by inference (Baffoe, Baarah, & Peyton, 2013).

2.5.2 IBM SPSS Analytical Decision Management (IBM SPSS ADM) and IBM SPSS Modeler

IBM SPSS Analytical Decision Management (IBM Corporation, 2016b), further referred to as IBM SPSS ADM, and IBM SPSS Modeler (IBM Corporation, 2016h) are both rule engines used in our IBM case study and default rule engines that come with the IBM SPSS (IBM Corporation, 2016h) suite. IBM SPSS ADM is a generic rule engine that is closely integrated in the IBM SPSS suite allowing the use of the predefined data sources in IBM SPSS Collaboration and Deployment that can both be used in IBM SPSS Modeler and IBM SPSS ADM. IBM SPSS ADM by itself cannot perform actions but it can infer events. It has basic comparison capabilities which will be described further in the IBM RPM case study.

IBM SPSS Modeler consists of a predictive analytics rule engine. It is not usually considered as being a rule engine but it does data inference where new (predicted) events are outputted following the insertion of events, the accumulation of others, and the application of a series of "rules". For this reason, IBM SPSS Modeler is considered as being a rule engine in the context of the RPM framework.

2.5.3 Drools and IBM Operational Decision Manager

Drools (Red Hat, 2016) is an interesting rule engine which requires more customization but offers a lot of functionality such as having its own rule language and enabling an easy integration with Java. Drools has not been used in our case studies but is offered as a comparison to the Rule Engines we did use. IBM Operational Decision Manager (IBM Corporation, 2016f), also known as IBM ODM, resembles Drools in terms of functionality. IBM ODM is not used in our use cases but is offered as

comparison like Drools. IBM ODM and Drools are both very powerful engines that can do a lot of different things but require a lot of customization and implementation for them to be used in specific use cases.

2.6 Related Works

2.6.1 Application Framework for Monitoring Care Processes (AFMCP)

Baarah created an interesting framework in the context of his thesis (Aladdin Baarah, 2013). His Application Framework for Monitoring Care Processes (AFMCP) addressed reactive performance monitoring in the context of care processes in clinics, hospitals, etc. His framework is based on the integration of a Care Process Monitoring Application (CPMA) with existing sources and new sources. For example, in Figure 6, the CPMA is used with online forms. In our context, the online forms would consist of the operational level and the CPMA would fit in the analytical level. The CPMA always contains a Care Process Monitoring Engine (CPME) which helps triage and infer facts to store in the data mart (star schema). His CPME is implemented using IBM ODM (previously known as IBM Websphere Business Events (IBM Corporation, 2016i)). The CPMA also contains a Performance Reporting Dashboard which allows BI capabilities in the context of the CPMA. When the CPMA is integrated in a hospital architecture, facts come in from various sources. Their framework usually makes use of a message broker in that case to be able to process the different facts.

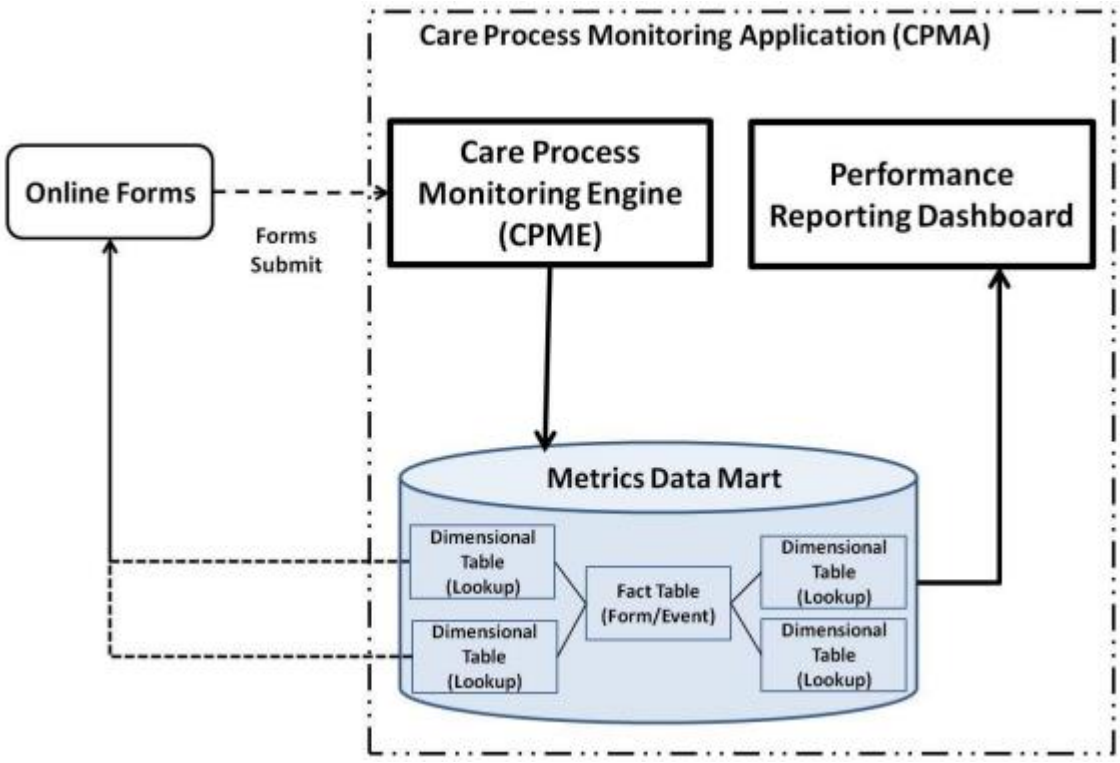

Figure 6: Example usage of AFMCP (Aladdin Baarah, 2013)

The AFCMP has significant similarities with our proposed RPM framework but does not directly trigger reactions such as notifying care providers when an alarm state occurred. The rule engine is specialized for their care processes based on a specific care process application model and their methodology is driven by that model. Specifically, their rule engine has been created to infer states and measure wait times.

2.6.2 Big data Analysis Infrastructure Testbed (BAIT)

There are many examples in the literature of frameworks that attempt to bridge the operational and the analytical. However, these frameworks are always very specific to their use cases, either aiming at maintenance (Lechevalier et al., 2014) or a particular problem. The main example that we will use to compare in this thesis is a framework called "Big data Analytics Infrastructure Testbed" (Kelly et al., 2015). The main

application of their framework was to air traffic management and had the architecture described in Figure 7.

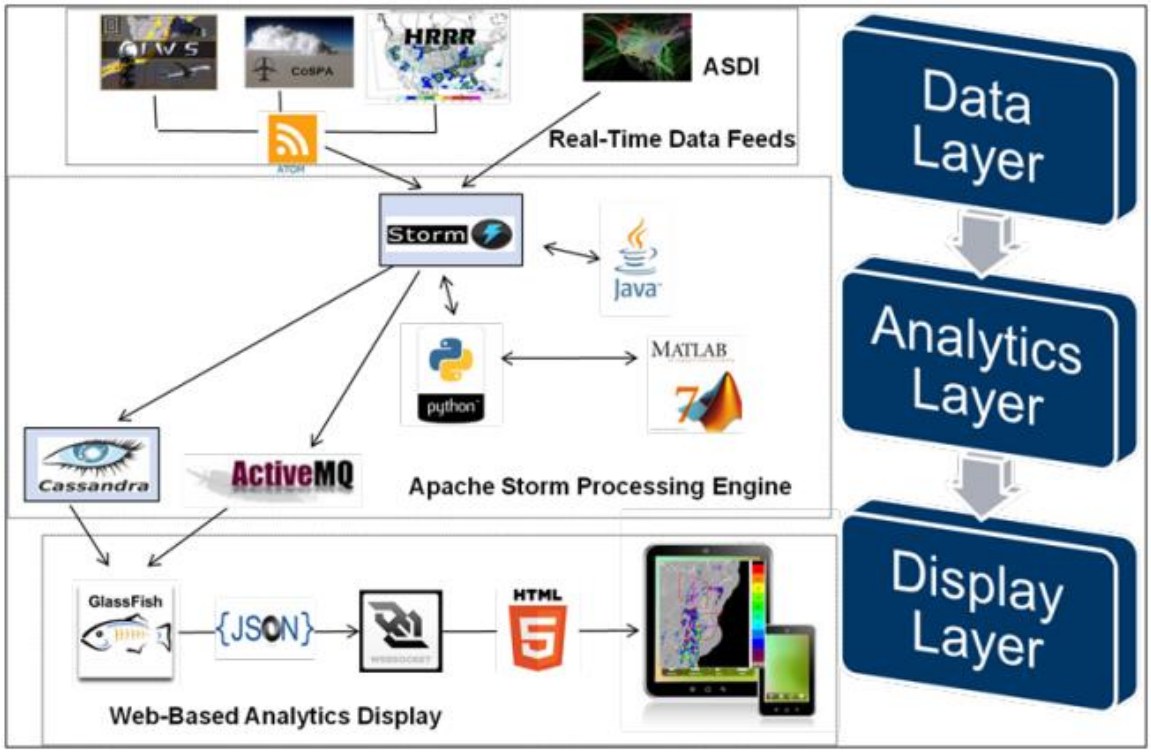

Figure 7: BAIT Architecture Layers (Kelly et al., 2015)

In Figure 7, we can see that their architecture consists of a "Data Layer", an "Analytics Layer", and the "Display Layer". In the "Data Layer", there is CIWS (Massachussetts Institute of Technology, n.d.) VIL and ET data transferred using Atom feeds (HOra & Gregorio, 2007), CoSPA (Pinto et al., 2010) VIL and ET data also transferred using Atom feeds, HRRR (Earth System Research Laboratory, 2016) VIL and ET data transferred using LDM transfer, and ASDI (Aircraft Situation Display to Industry) which transfers its data using the ASDI XML format. Every one of these tools have a different update schedule to send their data.

In the “Analytics Layer”, there is an Apache Storm Processing Engine (The Apache Software Foundation, 2015b) linked to a Cassandra (The Apache Software Foundation, 2015a) NoSQL (NoSQL, n.d.) database using Python (Python Software Foundation, 2016) scripts to wrap up and create an interface for MATLAB (The MathWorks, 2016) algorithms. MATLAB’s algorithms’ results need to be sent using JSON according to Apache Storm’s requirements. Apache Storm supports JSON-based protocol streams and JVM.

To transfer messages to the “Display Layer”, the processing engine uses a direct connection to the database, and a message broker, in this case, ActiveMQ (The Apache Software Foundation, 2016). The BAIT “Display Layer” is web-based and hosted on an application server. The application server, Glassfish (Oracle Corporation, 2015) in Figure 7, receives or retrieves data from the database or the message broker. Next, it sends the data to be displayed using the JSON format with the WebSocket protocol (Fette & Melkinov, 2011). The client is expected to display the data using HTML5 (W3C, 2014).

Similarly to our problem, the goal of this architecture was to provide feedback at the operational level from the analytical level in real-time. By providing “analytics-as-a-service”, they architected their infrastructure to be able to test various big data tools used in air traffic management.

Although they have managed to create real-time feedback from the analytical to the operational level, their approach is hard to generalize as it assembles tools specifically for their use case. In our context, the data layer is purely operational tools while the

analytics layer is purely analytical tools. The display layer is somewhat of an integration between the two because the different display tools can be used at one level or the other.

2.6.3 IBM Predictive Maintenance and Quality (IBM PMQ)

IBM PMQ (IBM Corporation, 2016g) is a collection of products creating a solution for the predictive maintenance of the assets of asset-intensive organizations. Several known software are contained in the IBM PMQ solution. Notably, there is the IBM SPSS (IBM Corporation, 2016h) suite including IBM SPSS Modeler, IBM Cognos Business Intelligence, and the message broker IBM Integration Bus. IBM SPSS is a product mainly focused on using predictive analytics. The IBM PMQ solution is structured to exploit IBM SPSS's capabilities to apply them to predictive maintenance. IBM PMQ is described in detail in chapter 4 as it is the starting point for our case studies.

3 A Reactive Performance Monitoring Framework

In this chapter, we present our proposed framework. We define and analyse the problem we wish to address in section 3.1 and 3.2. In 3.3, we identify the evaluation criteria that we will use to evaluate our research artifacts that address the problem, and in section 3.4 we present our Reactive Performance Monitoring Framework.

3.1 Problem Description

In any organization, there is a strategic level where the objectives and business goals that the organization wishes to achieve are determined by senior executives. As shown in Figure 8, these are translated into actions at the operational level where business processes are defined and human resources are staffed and information systems are implemented to carry out the operational business processes that are intended to achieve the strategic objectives and business goals of the organization. At the same time, the strategic is also translated into actions at the analytic level where managers and business analysts collect data and run reports to measure how well the operational processes are performing. These provide feedback to the strategic level, which might decide to change business processes, human resources or information systems in response. The data used by the analytical are typically gathered from operational processes as a log of operational events that have occurred. Often there is a reaction gap, where there is unnecessary latency, as the mapping between the strategic, the operational and the analytical is ad hoc at best (Barone et al., 2015).

Figure 8: Response Gap

Currently, decisions in businesses are made at the strategic level and the needed actions are passed to the operational and analytical levels. At the operational level, we have business processes being managed using business process management. They send events, or facts, to the analytical level so that the results can be analyzed and compared to the business' goals. The analytical level may use a systematic framework like balanced scorecards to organize reporting to the strategic level (2GC, 2015). Performance of operational business processes towards strategic business goals are measured in terms of metrics and key performance indicators. However, there is typically not a systematic framework for communication between the analytical level and the operational level. There is no centralized framework to automate feedback from the analytical to the operational, even for "easy" labelled scenarios, for example sending an email to the operational manager when the number of defects for a certain asset part reaches a particular number. At present, there might be feedback done from the analytical to the

operational level, but it is established in an ad hoc manner. Reactive performance monitoring consists of automating strategic decisions into rule-based actions when appropriate. Condition(s) in a rule allow us to flag certain characteristics which are thereupon used to trigger an action, such as a notification. Automating strategic decisions using rules is good for some cases where the outcome should be obvious. However, we have to be cautious to not automate complex decisions which necessitate strategy. A systematic model-based approach is needed to be able to more easily bridge the current gaps. Using a systematic model-based approach also has the potential of reducing the time lapse between identification and reporting from the analytical to actual changes and responses in the operational.

For example, consider an organization where maintenance needs to be done to keep big machinery, like cranes, in working order. At the operational level, each crane, has a maintenance schedule. There might even be different maintenance schedules for different parts of the crane.

At the analytical level, information is collected from the different cranes: date last inspected; which parts fail; after what interval of time the parts need to be replaced; and when the entire crane should be replaced. Once every quarter year, the analytical team puts together a report summarizing some of the data that interests stakeholders. Sometimes, they notice interesting information they use to put together a special report. This special report is then sent to the strategic level.

When the strategic level receives the report that states that crane use could be optimized if engine maintenance was done at the beginning of spring and the end of fall

instead of mid-winter, it might take a while until the strategic considers the information with a plan to change the schedule at the operation level. Subsequently, the plan needs to be implemented. If the operation level was not included in the discussion, the plan might need changes.

This process can be re-iterated multiple times for different scenarios. Obviously, the people involved in these actions can try to optimize their time use to make the process shorter. However, there is a limit to what can be accomplished with current practice. In an ideal world, the situation with respect to crane maintenance might be modeled effectively enough, that the analytical could be continuously assessing data from the cranes and providing feedback directly to the operational as to the optimal time for maintenance tasks using reactive performance monitoring.

3.2 Gap Analysis

As explained in 3.1, there is an existing gap between the strategic, the analytical, and the operational that we want to bridge. In particular, a direct interface between the analytical and the operational is missing. The current practices would highly benefit from a systematic model-based approach. A combination of several things would address the problem. First, we would need an architecture with components and interfaces to manage the interactions needed for continuous feedback between the operational and the analytical. Second, a methodology would be needed to systematically establish how to approach applications like crane maintenance. Finally, a flexible rule environment would be needed to define the business rules around continuous feedback within the architecture. Integrating the operational business processes with the analytical data

models would facilitate the communication of relevant events and inferred or predicted facts between the operational and the analytical.

As mentioned in Chapter 2.6, there is related work that has tried to address this problem. For example, The Application Framework for Monitoring Care Processes (AFMCP) had a specialized rule engine designed for care process monitoring, and a specific care process application model which drove the AFMCP's methodology. However, this framework is specialized for inferring states and measuring wait times only. In addition, it does not have actions as an output, it simply has dashboards that need to be actively consulted by the appropriate users. As a result, although this is a good framework for care process monitoring, it is too specialized and lacks resulting actions, such as notifications, that can actively link the operational and the analytical.

The Big data Analysis Infrastructure Testbed (BAIT) architecture is generalized for testing different tools in air traffic management. They succeeded in creating a useful big data infrastructure specifically made for air traffic management with mostly open source tools that provide constant monitoring of VIL, ET, and general flight data. The tools gathering the data (CIWS, CoSPA, HRRR, ASDI) are not necessarily open source, and MATLAB, which plays a key role when it comes to the algorithms available, is definitely not open source. The rest of the architecture uses open source tools. Although it might be adequate for this use case, this infrastructure did not offer actions other than a real-time dashboard. It would not be feasible to apply their infrastructure generally but it is a good example of an infrastructure attempting to solve the described strategic-

operational-analytical business gap problem for a specific domain (air traffic management).

IBM PMQ has managed to slightly reduce the gap between the operational and the analytical levels of businesses by using predictive models to create prioritized work lists. IBM PMQ is very specific to predictive maintenance and does not inherently offer actions other than ordering work orders which have to be manually consulted. It will be discussed in more detail in chapter 4, as part of our case study work.

3.3 Evaluation Criteria

Based on our analysis of the problem, our literature survey, and our experiences interacting with domain experts to define our case studies, we have identified the following list of criteria to evaluate any proposed solution or framework for our problem. We also identify specific criteria for the rule environment needed to bridge the gap between the operational and the analytical. We will use these criteria in chapter 5 to evaluate our proposed RPM framework.

3.3.1 Framework

- **Cost of hardware, software, and support:** This is the monetary cost of the hardware, software, and support needed for the framework. Cost is always an issue when considering a framework, tools, or methodology in an organization.
- **Developer Skillsets/Training:** This is an indication of how much do the developers need to know to be able to work with the framework. As reported by (Bandor, 2006), it is important to consider whether people can

work with the chosen tools, infrastructure, or framework. That is to consider if the people that will use the tool already have the necessary knowledge or require additional training.

- **Developer Effort:** This is an indication of the time and effort to implement reactive performance monitoring using the framework. Once again reported by (Bandor, 2006), it is important to consider the time and effort required to interface with a product and make use of its features.
- **Rule Support:** This is an indication of the framework's support of different rules and being able to change them. It implicitly indicates the presence of some sort of rule engine. To be able to integrate business rules and other analytics rules in a structured framework, a rule engine is necessary. Rules determine resulting actions and a component in the framework specifically made to write rules would highly facilitate the use of the framework.
- **Defined Architecture with APIs:** This criterion indicates the presence of APIs in the framework. As explained by (Tulach, 2008), APIs are important in complex frameworks or applications to enable an easier and better use of the components without worrying about the details by creating abstraction.
- **Predictive Modeling Integrated:** This is to indicate whether predictive modeling is directly integrated in the framework. A lot of the focus in literature and enterprise is on creating frameworks integrating predictive modeling, especially predictive maintenance. The BAIT (Kelly et al.,

2015) infrastructure, IBM PMQ (Dash et al., 2013), and others like the one presented by (Lechevalier et al., 2014) are examples using predictive modeling.

- **Latency between knowledge and action:** This is to indicate the time it takes between knowledge and action. This is key to make use of the information gathered and found as it is in business process management systems (zur Mühlen & Shapiro, 2010).
- **Supports OLAP databases:** Criterion to indicate whether the said framework possesses support for OLAP databases. Relevant facts to actions, such as the use of predictive modeling might come from different kinds of OLAP databases depending on the scenario. The support of multiple OLAP databases can become key in the generation of meaningful actions (Aladdin Baarah, 2013).
- **Methodology for reactive performance monitoring application development:** Criterion indicating whether a methodology to create performance monitoring applications exists. As it is with data warehousing applications (Kimball et al., 2008) or, for example, real-time monitoring applications (Aladdin Baarah, 2013), a methodology is needed to be able to accomplish the initially set goals.

3.3.2 Rule Environment

- **Dedicated Syntax:** This criterion indicates whether there is a set syntax to create the rules and actions. For the users to be able to write rules seamlessly, the rule syntax, or language, must have a certain regularity to

it. For example, Drools is often liked for its punctuation and versatility (Garzón, 2015). This criterion is mainly to compare the different types of approaches that the different rule engines have taken.

- **Open Source:** This criterion indicates whether the rule engine is open source or not. In its definition, open source software is freely distributable (Perens, 1999). This criterion generally affects costs, especially if there is a strong support developer community (MacCormack, Rusnak, & Baldwain, 2006). In the case of our evaluation, this also affected the **Extendible Operations** criterion.
- **Extendible Operations:** This criterion indicates whether it is possible to add different kinds of condition functions and action methods. Drools (Red Hat, 2016) and IBM ODM (IBM Corporation, 2016f) are great examples of extensibility where the implementation of different condition functions and action methods is almost necessary. This offers great flexibility in terms of application.
- **Directly Linked to Actions:** This criterion indicates whether it is possible to trigger an action natively from the rule engine or easily such as through a Web service from the rule engine. During my IBM internship, one of the goals was to start addressing this problem present in the underlying structure of IBM PMQ (IBM Corporation, 2016g) which is used for maintenance of assets management.
- **Scheduled Facts:** This criterion indicates the possibility of triggering actions based on a timing functionality in the rule engine. Schedulers can

be a real value added especially when considering business processes. This is why schedulers have been integrated with rule engines such as Drools (Red Hat, 2016).

- **OLAP Database Integrated:** This criterion indicates whether it is easily possible to link with and integrate any OLAP database. In particular, whether it is possible to articulate rules with conditions and actions based on the contents of OLAP databases. We realized that this was necessary for a well-integrated RPM framework when exploring the use of other rule engines while working on both the IBM RPM and the QuickForms RPM. The OLAP database is central to our context and not having access to it is awkward and difficult with most rule engines as they generally do not have support for this.
- **Predictive Inferencing:** The criterion indicates whether the rule engine possesses predictive analytics capabilities. Predictive Analytics has been proven to be very interesting for performance monitoring use cases, in particular in the domain of predictive maintenance (Dash et al., 2013).

3.4 Overview of the Reactive Performance Monitoring Framework

This section provides an overview of the RPM framework. It consists of:

- An architectural pattern that identifies the key component technologies and interfaces that need to be brought to bear and how they can be organized

- A methodology that defines the steps needed to be taken in selecting the relevant components for an application, defining models as appropriate, and iteratively identifying, defining, and reefing the events and rules used to link data analytics to actions
- A rule environment in which rules can be written, conditions used, and actions enacted

3.4.1 Architectural Pattern

As shown in Figure 9, the architectural pattern for our RPM framework is composed of 4 main components: **Operational Event Sources**, **Analytical Data Warehouse**, a **Rule Environment** that mediates between them, and **Actions** that can be invoked for continuous feedback. The first component, **Operational Event Sources**, represents any entity from which we could receive Fact Events. The second component is the **Rule Environment**, the core of our contribution. Third, the **Actions**, which has **Email**, **BPM**, **Web Service**, and **Scheduler** but could contain other possible actions. The last component is our **Analytical Data Warehouse**. In particular, the **OLAP Database** composes the heart of the **Analytical Data Warehouse**. In this section, the data flow will be described followed by a description of each component.

As we described in 3.1 using Figure 8, our architecture pattern aims at closing the gap between the operational and the analytical. The **Analytical Data Warehouse** contains all the data that the analytical uses to understand the operational. The **Operational Event Sources** provide data from the operational to the analytical, and the **Actions** provide feedback from the analytical to the operational. The **Rule Environment**

is really what distinguishes this architecture as a bridge for the analytical to give feedback to the operational. It is a specialized analytical engine for processing events and determining actions in real-time. This lets feedback skip the Strategic when it is not needed.

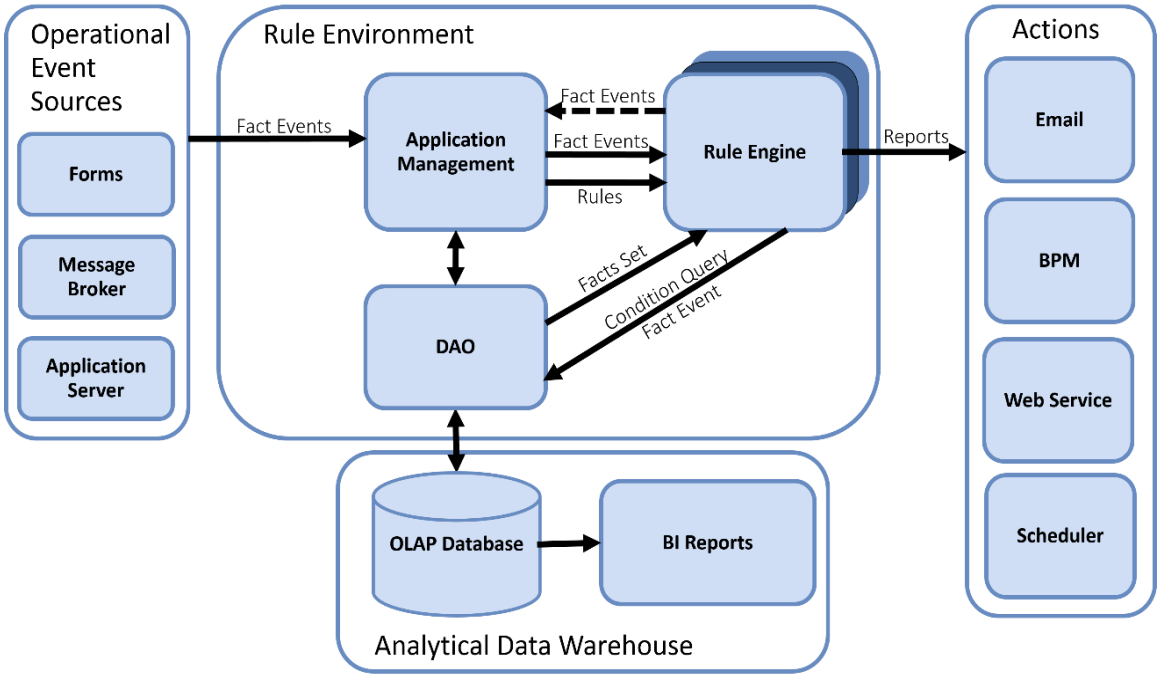

Figure 9: RPM Framework Architecture

In this architecture, **Fact Events** are received from **Operational Event Sources** and sent to **Application Management**. **Operational Event Sources** can be **Forms** filled by people, a **Message Broker** that forwards **Fact Events**, or an **Application Server** that sends **Fact Events** directly. Depending on the **Fact Event**, it will either be sent directly to the **DAO** to be stored, or to the **Rule Engine** if there are **Rules** corresponding to the **Fact Event**.

When the **Fact Event** is sent to the **Rule Engine**, it is evaluated against its corresponding **Rules**. **Rules** need to be predefined so that the **Application Management**

can send them to the **Rule Engine** ahead of time. That way, the **Rule Engine** becomes ready to receive **Fact Events**. Depending on the **Rules** corresponding to the **Fact Event**, the **Rules** may use condition functions that require a **Condition Query** to test the current context when the **Fact Event** is received. The current context is defined by the **OLAP Database** that has stored all **Fact Events** currently received (according to a multi-dimensional model, as defined by OLAP). An **OLAP Database** consists of a database optimized for reporting, especially used by the analytical.

A **Condition Query** consists of a query made to an **OLAP Database** in order to obtain the relevant data needed to evaluate the business **Rule** for a **Fact Event**. In return the **DAO** will return facts as a **Facts Set**. The **Rule Environment** has appropriate functions defined in order to manipulate and test the information contained in the **Facts Sets** returned by the **Condition Query**. **Condition Queries** are essential to the well-functioning of the **Rule Engine**. They are used in rule conditions to extract relevant data from the **OLAP Database**. For example, in conditions necessitating contextual information such as the number of times a **Fact Event** of a particular type has been received in the last 5 hours, a **Condition Query** will be used to extract a **Facts Set** of the **Fact Events** of the particular type in the last 5 hours. These are explained in more detail in 3.4.3.

It is worth mentioning that there can be more than one **Rule Engine**, and other types of Rule Engines. For example, in the case of a **Rule Engine** possessing predictive analytics capabilities, this **Predictive Analytics Rule Engine** can predict events that will happen if actions are not taken. These predicted **Fact Events** can be sent back to the

Application Management in addition to operational events. A **Classic Rule Engine** with "IF <condition> THEN <action>" rules could then be used to translate either predicted or operational **Fact Events** in to appropriate actions. By default, there is at least one **Rule Engine** which should be an **OLAP Rule Engine** for it to be able to be integrated with the **OLAP Database** for logging, and inferring **Fact Events**, and for querying any aspect of the complete history of all **Fact Events** that might be relevant for defining context and determining **Actions**.

In the **Rule Engine**, after a **Fact Event** has been evaluated against one **Rule**, one or more resulting **Actions** may occur. An **Action** can consist of simply storing a single **Fact Event** or multiple **Fact Events** in different **OLAP databases** via the **DAO**. However, there are other interesting **Actions** that may occur. Generally, once a **Fact Event** has been evaluated positively against a **Rule** in the **Rule Engine**, an action method is executed using the **Fact Event**'s data (including **Fact Sets** returned by **Condition Queries**) and a **Report** is sent to the corresponding **Action** component.

We have identified four main possible **Action** components: **Email**, **BPM**, **Web Service**, and **Scheduler**. Furthermore, each of these components could have a role in the **Operational Data Sources** component, therefore creating a feedback loop. All **Actions** are meant to link the analytical with the operational. **Emails** consist of messages (**Reports**) sent to a number of recipients, on the operational side, to provide timely information that needs to be acted upon. **BPM** consists of either triggering a new business process, trigger an action in a business process, or simply sending useful information (**Reports**) to an already existing process, all on the operational side. A **Web Service**

simply consists of a generic service available to send information (**Reports**) to servers on the operational side. The **Scheduler** is to schedule any of the above **Actions** to take place in the future, as well as generating a scheduled **Fact Event** (in order to trigger rule processing at a scheduled time).

The **Operational Event Sources** are simply defined by their capacity to send **Fact Events**, from the operational to the analytical. They can be any entity that has that functionality. It could be from a **Form**, which enables humans to enter data defining or reporting a **Fact Event**. It could be from an enterprise **Message Broker** that receives and forwards events from any source within the organization using a publish-subscribe mechanism. Or it could be from an **Application Server** containing specific apps or web services that sends events directly to the **Rule Environment**.

Typically, a **Form** simply consists of a specific application made for manual input of events. The **Message Broker** could be implemented using technology like IBM's Integration Bus, BizTalk (Microsoft, 2016), ActiveMQ, etc. If we consider the strategic-operational-analytical problem, the different event generators correspond to the operational part of the business where we monitor different key metrics destined to the analytical part.

Another essential part of our framework consists of the data warehousing components. As per usual, the data warehousing components include **OLAP Databases**, and an engine allowing **BI Reports** to be made. All data used, processed, or generated by our RPM framework is persisted in **OLAP Databases**. In particular, this allows **OLAP**

Databases to report against history, making use of aggregations, contextual information, and trend analysis.

3.4.1.1 Critical Interfaces

The **Rule Environment** provides the critical interface to the operational and the analytical. The interface to the analytical is based strictly on a Data Access Object (DAO) design pattern (Sun Microsystems Inc., 2002). One component manages the interface to the analytical and can read and write to any database in the **Analytical Data Warehouse**.

The interface to the operational is more nuanced. The REST interface provided by the **Application Management** component to receive **Fact Events** is the mechanism for receiving inputs from the operational. The **Rule Engine** creating **Reports** to send to **Actions** is the mechanism for providing feedback to the operational.

3.4.2 Methodology

The RPM framework methodology is shown in Figure 10. It is a model-based approach in which business process models on the operational side are related to multi-dimensional **OLAP Databases** on the analytical side in terms of business **Rule Sets** that link **Fact Events** to **Actions** that deliver **Reports**. Three main roles are needed to complete an application using the RPM framework. First, there is the **Business Operations Architect**. Their role pertains to current operational business processes and how they can be integrated in the framework. The second role is the **Performance Analytics Analyst**. This role does not accomplish many tasks but is crucial to this framework. This role creates the OLAP Databases on which all applications are based.

The third and last role is the **RPM Applications Developer**. This role mostly develops what has been established by the previous roles.

In Figure 10, there are 13 main activities, each performed by one of the three different roles:

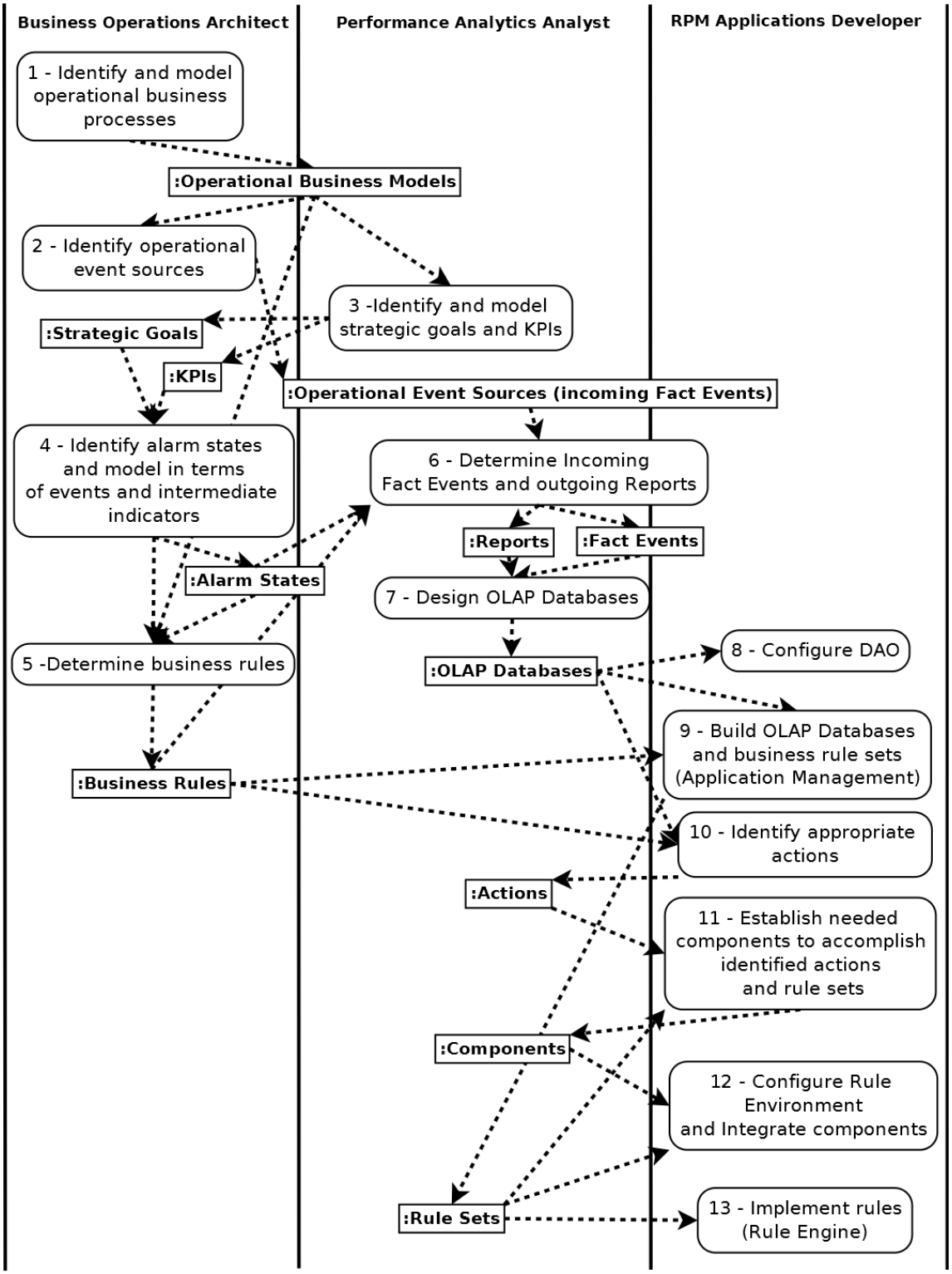

Figure 10: RPM Framework Methodology

1. **Identify and model operational business processes:** During this step, the **Business Operations Architect** creates operational models of the operational business processes that need to be monitored. This includes identifying the monitored objects, and any roles (people) that are involved in the process of monitoring or using those objects. Suppose one of our business interests was the maintenance of cranes in the domain of heavy machinery. We would have to identify specific operational business processes that interest us. For example, we could be interested in the maintenance of crane brakes. Once we have identified this specific operational business process, we model it. When we model it, this includes the identification of the things we want to monitor, in this case, the crane brakes, and any party affected in the process (for example, a mechanic, a management team, a technician, etc.) At the end of this step, the obtained completed artifacts are **Operational Business Models**.
2. **Identify operational event sources (incoming Fact Events):** Once we have the **Operational Business Models**, the **Business Operations Architect** can identify the **Operational Event Sources** that can provide **Fact Events** for monitoring the operational business processes. The **Operational Event Sources** are key to our architecture as they are the direct link of the Operational to the Analytical. In the crane brakes example, we would need to identify all brakes, the people in charge of maintenance, the people in charge of ordering parts, the crane users, and maybe even others. Each of these would communicate events through

Forms (e.g. for the people in charge of maintenance), **Message Broker** (e.g. for some of the sensors of the brakes), or **Application Server** (e.g. for the application used by the people ordering parts). In addition to identifying the actual sources, we would need to identify any information relevant to the crane brakes scenario that needs to be monitored about the brakes. This might include the level of use, an indication of unevenness, voltage, etc.

3. **Identify and model strategic goals and KPIs**: This step and "Identifying operational event sources" are steps that can be done in parallel since they both only depend on having modelled the operational business processes and they are performed by different roles. This step is performed by the **Performance Analytics Analyst**. When we have the models, the analyst can identify the relevant **strategic goals** the operational business process is intended to support. From the **strategic goals**, we can derive **KPIs** that it would be important to report and the **Fact Events** needed to generate the **Reports**. **KPIs** define the target range. When a **KPI** is outside of its target range, actions need to be taken. In the crane brakes example, we might like to improve the efficiency of the business process by doing preventative maintenance at the correct time instead of slowing down the operations by doing reactive repairs. For the **KPI**, and as an example, we would then aim to reduce the delays due to the brakes by 50%.
4. **Identify alarm states and model in terms of events and intermediate indicators**: This step done by the **Business Operations Architect** aims at

concretizing the business rules using the information that we have from **Fact Events** coming in to identify **Alarm States** for which we should generate **Reports**. For example, in the crane brakes example, we want to send an email to the people in charge of parts when the thickness of brake pad A attains a certain level. In this step, we need to identify where that fits in the **Operational Business Model**. This is also the step where we identify whether we need additional information that was not previously included. For example, we might want to add an **Alarm State** if the new brake pad has not been ordered (meaning that we need to identify when a part has been ordered) and it has been more than 3 days (meaning that we will need to keep timestamps). With this information, we have an **Alarm State** that can be identified with specific information that also needs to be tracked. With this **Alarm State**, we can create a report which will enable us to act upon it.

5. **Determine business rules**: Once **Alarm States** have been identified, the **Business Operations Architect** can actually create the **business rules** that will trigger the wanted **Actions** when the specific conditions are met.
6. **Determine Incoming Fact Events and outgoing Reports:** Using the identified **Operational Event Sources**, the **Alarm States**, and the **Business Rules**, the **Performance Analytics Analyst** can formalise the **Fact Events** and **Reports**, and triage which information should be kept. In our crane brakes example, we might want to keep crane information such as the model, year of fabrication, year of acquisition, as well as brake

information including model, location, and relevant metrics such as brake pad thickness, voltage, relevant timestamps, and any necessary information about the people in the process (emails, ids, etc.)

7. **Design OLAP Databases:** With the information established in step 6 as well as the **Business Rules**, the **Performance Analytics Analyst** can finally design the **OLAP Database(s)**. The **OLAP Database** is crucial for the **RPM Applications Developer** as well as represents the basis of the framework. In reality, there is already business reports that are being sent to the strategic to report on the operations. A combination of both the **Fact Events** and the **Reports** correspond to the dimensions that can be used to create the **OLAP Databases** for the performance monitoring in which we are interested. The dimensions combined with the previously identified **Fact Events** create the **OLAP Databases**.
8. **Configure DAO:** The first step of the **RPM Applications Developer** simply consists of creating the connection between the **OLAP Database**(s) in the DAO.
9. **Building OLAP Databases and business rule sets** (Application Management): Although the **OLAP Databases** have been designed, they still need to be implemented. The **RPM Applications Developer** accomplishes this during this step. When the **OLAP Databases** are implemented, it allows the rules to refer to them. With this integration, we can then prepare the business **Rule Sets** from the **Application**

Management point of view so that it can send the rules to the Rule Engine once they are implemented.

10. **Identify appropriate actions:** Even if the **Business Rules** have been determined by the **Business Operations Architect**, at this step, the **RPM Applications Developer** needs to properly identify what type of **Actions** will be needed. In the crane brakes example, we were interested in notifying the parts crew and perhaps the management group if a part has not been ordered within three days. We would then be interested in a notification action like using a service to send an **Email**. Another option might have been that we would send a trigger to a **BPM** system to start a process that would automatically assign the ordering brake pad part task to a specific person. This step is where we identify each action needed for each rule in each **Rule Set**.

11. **Establish needed components to accomplish identified actions and rule sets**: In this step, the **RPM Applications Developer** needs to identify the **Components** needed to accomplish the identified **Actions** and **Rule Sets**. Two main factors affect this choice: the action type, and the component availability. In our framework, we have identified four main external action types for reactive performance monitoring: **email**, **BPM**, **web service**, and **scheduler**. For example, for **email**, we need an email service. This email service would be identified as one of the needed components. For a **BPM** action, the best component might not be available. In that case, the action might be implemented differently and in

an ad hoc way, for example through an email sent to the actor triggering a business process.

12. **Configure Rule Environment and Integrate components:** Depending on the **components** chosen, some might come freely with the chosen tools but other **components** might require the addition of an interface to the service. For example, if we want to trigger a **BPM** system, this action might need to be added to the services supported by the **Rule Engine**. Configuration and integration become additional tasks of the **RPM Applications Developer**.
13. **Implement Rules:** Once all the details have been figured out, the **RPM Applications Developer** can simply implement the **Rules Sets** that have been predefined in the **Rule Engine** language.

3.4.3 Rule Environment

The Rule Environment is an important part of the RPM framework. It enables the storage of facts on which we can report on, and relevant actions. As briefly described in the architecture, the Rule Environment is composed of three main components: the Application Management, the Rule Engine(s), and the DAO.

Application Management is responsible for quick Fact Event processing as well as passing on Fact Events that require Rule processing, and sending out Rule sets information to the Rule Engines, available from the Rule Sets Configuration. It also ensures that many different applications could be running simultaneously. By quick Fact Event processing, we mean storing facts to the OLAP Database. The Rule Sets

Configuration contains the information necessary for the Rule Engine to recognize a Rule set, as well as the information necessary for a Fact Event to be evaluated against the Rule set(s) it is associated with.

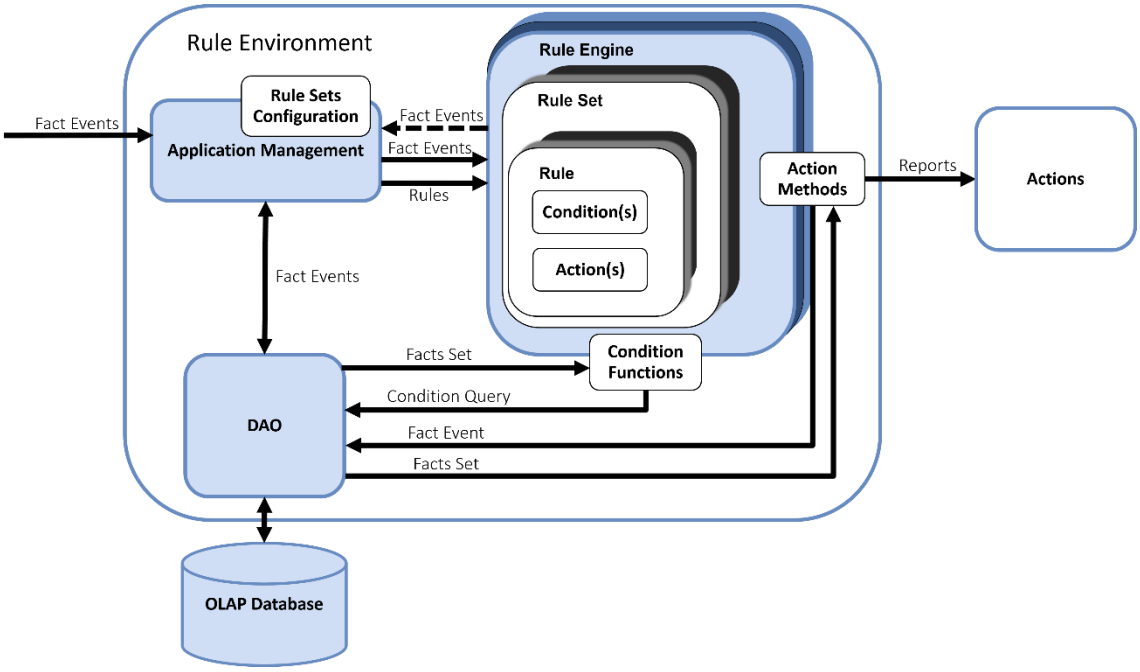

Figure 11: Rule Environment

In the Rule Environment, there can be more than one Rule Engine. In the context of this thesis, three main types of Rule Engines are identified. The first one is the Classic Rule Engine. The Classic Rule Engine follows the "IF <condition> THEN <action>" pattern and has no specific support for OLAP. The second type is the Predictive Analytics Engine. This type has predictive analytics capabilities and does not generally follow the classic "IF <condition> THEN <action>" pattern. A Fact Event goes into the Predictive Analytics Rule Engine and another Fact Event comes out after having passed through a predictive analytics model. The third type is the OLAP Rule Engine. This Rule Engine is like a classic Rule Engine but possesses OLAP support. A Rule Engine's role is to react

upon the incoming Fact Events and make them go through their respective Rule Sets. For a Fact Event to go through another Rule Engine, it must be modified by the first and passed back to the Application Management.

3.4.4 OLAP Rule Engine

To fully link the Operational with the Analytical for reactive performance monitoring, we define an OLAP Rule Engine.

Figure 12 shows the OLAP Rule Engine object model. A Rule Engine can contain multiple Rule Sets. A Fact Event can be evaluated against multiple Rule Sets from the same Rule Engine. A Rule Set corresponds to an OLAP Database. A Rule Set is composed of a series of Rules.

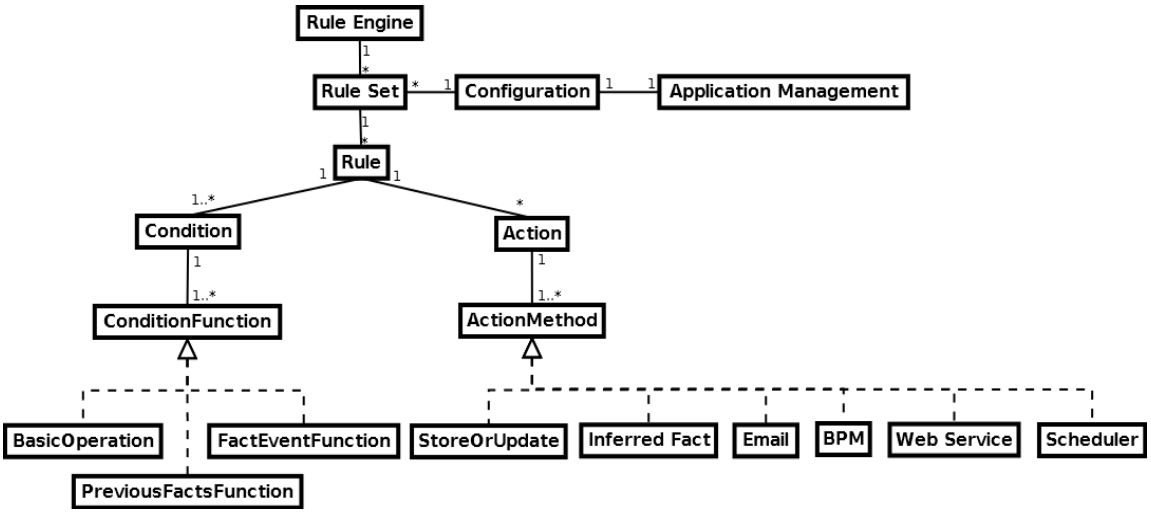

Figure 12: OLAP Rule Engine Object Model

Each Rule is composed of one or more conditions which can be expressed with plain basic operations or using available condition functions. Each Rule also has a series

of actions that are executed against the Fact Event if the condition has been evaluated as true.

There are three main types of condition functions: Basic Operations, Previous Facts Functions, and Fact Event Functions. Basic Operations simply consist of using basic operators that can be applied to integer, real, characters, and booleans. Previous Facts Functions are only available in OLAP Rule Engines because it requires an OLAP Database connection and permits us to access previously stored facts to trigger an action. It makes use of historical facts and aggregations. This is particularly used if we need contextual information. For example, if we wanted to send an email to the managers only if the brake pad has been evaluated three times as being too thin, we would need to get the count of previous facts where the brake pad has been evaluated as being too thin. This is done by using a Previous Facts Function. Actually, the Previous Facts Function would do a Condition Query asking for the previous facts for the particular brake pad. In return we would get a Facts Set in which we would count the number of relevant entries. The third type of Condition Function is to compare Fact Event fields. Fact Event Functions allow us to extract data from the Fact Event by field name and compare strings. We can also simply verify that a field is present using a Fact Event Function.

On the actions side, we have categorized them using 4 main types. The first type is “StoreOrUpdate” and is necessary with an OLAP Rule Engine. This is a type that is always present in business analytics applications since we always need to store or update new data. For that reason, it is not really an external action, and it is done directly via the Rule Environment to the OLAP database. We called the second type “Inferred Fact”. This

type is one of the main reasons why an OLAP Rule Engine is necessary. Like the "StoreOrUpdate" action, "InferredFact" is done directly from the Rule Environment to the OLAP database and therefore only available in an OLAP Rule Engine. Inferred Fact consists of facts that are determined via a series of factors. These factors can be a series of Fact Events, the presence of a certain entity combined with an incoming Fact Event, contextual data combined with a Fact Event, etc. The inferred fact highly depends on the condition function, especially previous fact functions for contextual information. However, previous fact information functions are not only used in condition functions. For example, for the BPM action, previous fact information can be useful to the process triggered. BPM has been identified as one of the main action types because being able to trigger BPM processes directly translates on linking the analytical to the operational.

Another type of action is Emails. It is easy to see that emails are incredibly useful in RPM scenarios since it is a type of actions that can directly be transferred to operations. With the Web Services action, technically any type of action that can be called upon using a service could be accepted in the RPM framework. It is a generic action type. We simply have categorized the actions that seemed to be the most useful in an RPM framework. As a last action, we have the Scheduler. The Scheduler is simply an action that allows us to call other actions at another time.

For the OLAP Rule Engine, each Fact Event must contain information about the Rule Set, or OLAP Database, to which they correspond. This is in order for the DAO to be able to identify the correct OLAP Database associated with the Fact Event to store. In our case, the role of the DAO is to do the translation between the Database and the Rule

Environment. As described in the diagram, the DAO receives and sends back Fact Events to the Application Management. The DAO possesses methods which can be called by other entities to store and update facts, as well as request a facts set with, or without, a Condition Query. These simple methods allow all interactions with the OLAP Database.

However, the OLAP Rule Engine does not have free access to the DAO as interactions are done through action methods or condition functions. It only requests DAO functions through condition functions and action methods. The Rule Engine has access to condition functions and action methods against which the Fact Events are evaluated if they are associated with a particular Rule Set.

4 Case Studies

4.1 Overview

This chapter presents our case studies and some rule examples. First we present an existing IBM technology which inspired the RPM framework. Second, our first iteration of the framework is presented, followed by the third. Lastly, two rules are presented in the last two sections showcasing the strengths and weaknesses of the different rule engines used in these case studies, with a particular focus on the OLAP Rule Engine we presented in chapter 3.

In 5.2, we present IBM PMQ, an already existing framework we worked with at IBM. It is while working with IBM PMQ that we established the need for an RPM framework. IBM PMQ was created to do the predictive maintenance of assets and is a great tool for it. While using both a Predictive Analytics Rule Engine and a Classic Rule Engine, it is able to predict asset failure and can provide a worklist with different levels of urgency. But in spite of being of being a great tool, IBM PMQ still does not quite fit the RPM framework. We think that IBM PMQ is a missed opportunity because predictive maintenance of assets is only a subset of predictive maintenance which itself is only a small subset of RPM applications. Furthermore, IBM PMQ failed at closing the gap between the operational and the analytical by not providing direct support for invoking actions. Work Orders can be created but do not consist of actions as they require active consultation by people to go and look at the list of Work Orders to see what needs to be

done. There is no timely and automatic delivery of the Work Orders to the right recipients.

For these reasons, in 4.3, we undertook our IBM RPM case study based on our proposed RPM framework. The IBM RPM case study implemented a successful RPM framework prototype using an IBM technology stack. We extended the IBM technology stack that was used for IBM PMQ, to include an IBM BPM adaptor in support of directly invoked action to trigger business processes. As well, we adjusted our RPM methodology, explained in 3.4.2, to customize and generalize the OLAP database from IBM PM, not just asset management. Yet, this implementation still had a few caveats. Amongst those, the generalization of the OLAP Database and the creation of the BPM adapter were complex, and the combination of Predictive Analytics Rule Engine and Classic Rule Engine was awkward to use. IBM PMQ experts, an expert database administrator, and at least one developer would be needed to use the framework in other settings.

This provided motivation to create our own RPM framework using open source technologies. QuickForms RPM is presented in 4.4. Using technologies that were free and much lighter, we managed to create an RPM framework that had all the needed components. This included the creation of an OLAP Rule Engine that was much more appropriate for RPM applications and that uses reports for both the conditions and actions of the rules. In addition, we identified that the main type of actions needed for RPM is notifications, so we provided both an Email and a Scheduler Action component. Web services were provided as a general all-purpose Action component. While BPM was not

one of the included actions, the presence of an API easily accepting Web services sufficiently demonstrate that it would be possible to add BPM.

In the last sections of this chapter, 4.5 and 4.6, we show two rule examples that were used in the IBM RPM and QuickForms RPM case studies to highlight the differences between the Predictive Analytics, Classic, and OLAP Rule Engines.

4.2 IBM PMQ

4.2.1 Architecture

As briefly described in 2.6.3, IBM Predictive Maintenance and Quality (IBM PMQ) is an architecture designed for predictive maintenance of assets in asset-intensive organizations. It has managed to slightly reduce the gap, but not close it, between the operational and the analytical level by using predictive models to create prioritized work lists.

Figure 13 shows a simplified architecture of PMQ. Device Events associated with the assets are continuously streamed into the Real-Time Event Processing implemented with the IBM Integration Bus message broker. The message broker then sends events both to be stored in the Analytics Data Store (OLAP Database) implemented by IBM DB2 and to the Predictive Analytics (Predictive Analytics Rule Engine in our context) implemented by IBM SPSS Modeler. After it receives the device event, IBM SPSS Modeler processes the device event through the prediction model, which will output a result depending on the model type. An example would be an asset failure risk value.

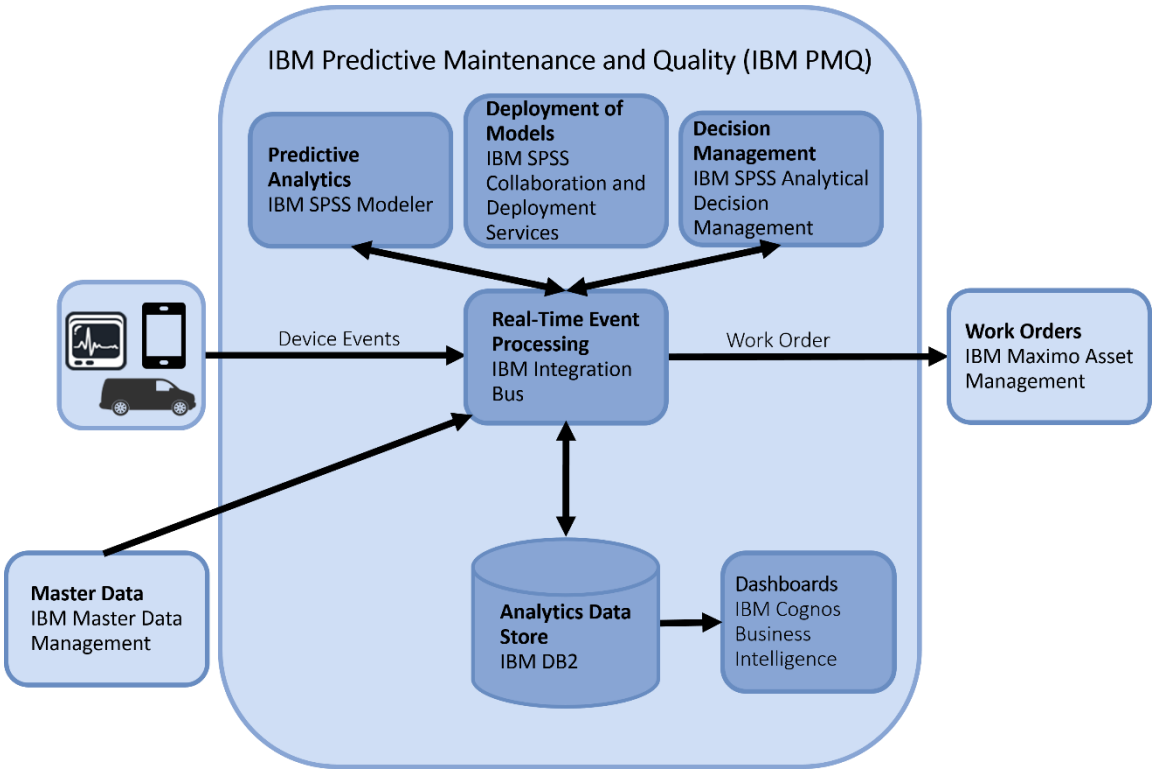

Figure 13: Simplified architecture view of IBM PMQ

Once this risk is outputted, it is stored to the Analytics Data Store and passed on to the Decision Management implemented by IBM SPSS ADM. The Decision Management, which is simply a Classic Rule Engine, outputs a value depending on the rule associated with the risk. The value outputted is subsequently stored in the OLAP Database and passed on to the Work Orders component if it exists and if that message flow was defined. The Work Orders, is a component that can be added to work with IBM PMQ and is implemented by IBM Maximo Asset Management. It simply consists of an intricate asset management worklist. Furthermore, it is interesting to note that in reality, whenever a message goes either to IBM SPSS Modeler or to IBM SPSS Analytical Decision Management, it goes through the "Deployment of Models" IBM SPSS Collaboration and Deployment Services (IBM SPSS C&D).

As we can see in Figure 13, there is also the Dashboards and Master Data components which have not been discussed. The Dashboards component simply corresponds to BI Reports that are implemented with IBM Cognos BI. The Master Data component is something that is part of the overall Analytic Data Warehouse in our RPM framework. Implemented by IBM Master Data Management, it is simply IBM PMQ's way of managing the fact tables and dimensions in the OLAP database.

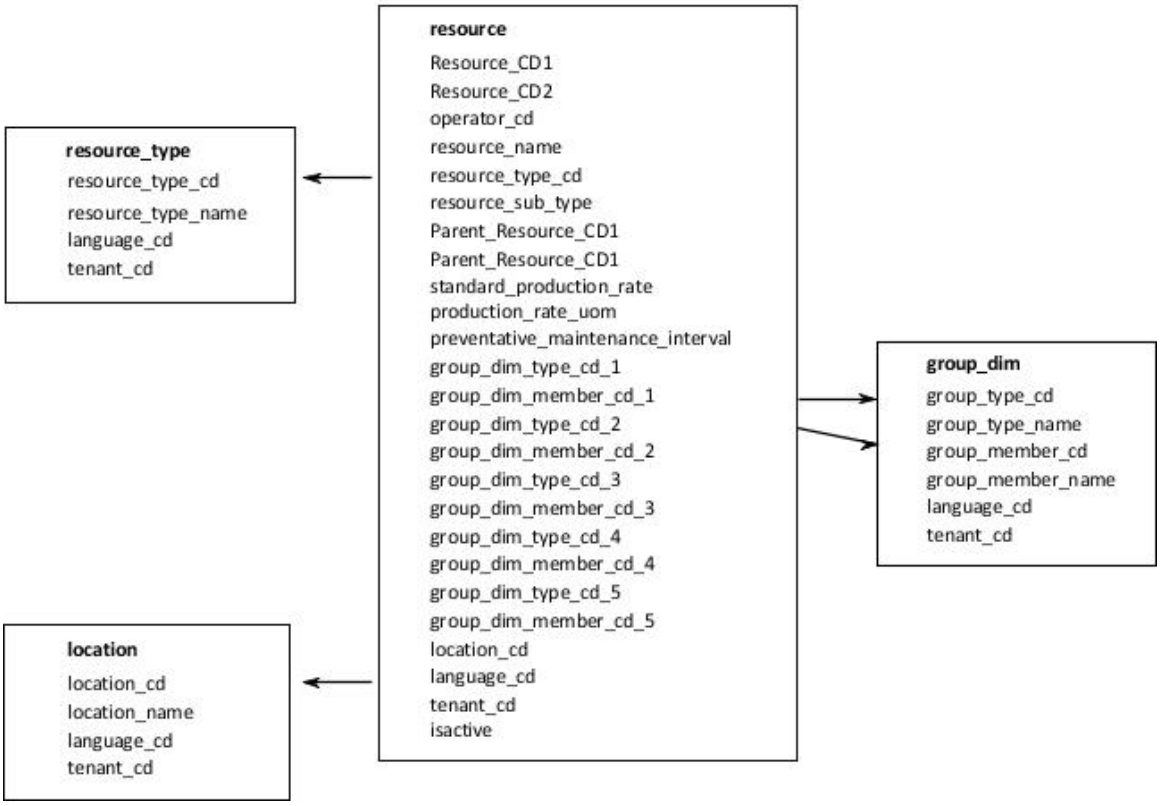

Figure 14: IBM PMQ Master Data Model (IBM Corporation, 2016a)

As shown in Figure 14, it is fairly complicated to manage the fact tables and dimensions to use IBM PMQ. IBM has divided the task in two categories: one they call "Master Data", and the other is called "Event Data". "Master Data" consists of the monitored assets. It is optionally managed by "IBM Master Data Management" or through a flat file following a specific model adapted for predictive maintenance that can

be extended. We have only experienced the second method. “Event Data” consists of the events that go through the architecture. Each of them needs to be defined. Like the “Master Data”, “Event Data” possesses a flat file API which must be in accord with the “Master Data”.

When both “Master Data” and “Event Data” has been defined, in this architecture, every message flow between every component needs to be defined. This is also very complicated as it is done both through the message broker directly and a component called the “Orchestration”. The “Orchestration” necessitates the definition of Master Data and Event Data before being created. It consists of an implementation of an XML model that can possibly be created through a Web application (IBM Corporation, 2015). Figure 15 consists of an example of the configuration file that needs to be written for the “Orchestration” that works with a unique event coming in. The “<event_orchestration_mapping>” identifies the event’s business process and the “<orchestration>” (message flow), that it needs to follow. The “<orchestration>” defines things like which event(s) are expected in, from which component, where it needs to be sent, and which event(s) need to be stored in the process. It is common to have multiple “<orchestration>”, especially if more than one Fact Event is tracked.

```
<event_orchestration_mapping>
   <event_orchestration_key_cd>inspection</event_orchestration_key_cd>
   <orchestration_cd>pmq.inspection</orchestration_cd>
</event_orchestration_mapping>

<orchestration>
  <orchestration_cd>pmq.inspection</orchestration_cd>
    <step>
   <adapter_class>com.ibm.analytics.foundation.adapter.profile.ProfileAdapter</adapter_class>
   <adapter_configuration xsi:type="ns3:profile_adapter_configuration">
        <observation_profile_update>
           <observation_selector table_cd="EVENT_OBSERVATION">
              <observation_field_value>
                <field_name>MEASUREMENT_TYPE_CD</field_name>
                  <value>INSPECT</value>
              </observation_field_value>
           </observation_selector>

           <profile_update_action>
              <profile_row_selector>
                 <shared_selector_cd>PRODUCT_KPI</shared_selector_cd>
              </profile_row_selector>
                <shared_calculation_invocation_group_cd>PRODUCT_KPI_INSPECT_COUNT
                </shared_calculation_invocation_group_cd>
           </profile_update_action>
        </observation_profile_update>

        <observation_profile_update>
           <observation_selector table_cd="EVENT_OBSERVATION">
              <observation_field_value>
                   <field_name>MEASUREMENT_TYPE_CD</field_name>
                   <value>FAIL</value>
              </observation_field_value>
           </observation_selector>
           <profile_update_action>
              <profile_row_selector>
                 <shared_selector_cd>PRODUCT_KPI</shared_selector_cd>
              </profile_row_selector>
              <shared_calculation_invocation_group_cd>
PRODUCT_KPI_FAIL_COUNT</shared_calculation_invocation_group_cd>
           </profile_update_action>
       </observation_profile_update>
      </adapter_configuration>
    </step>
</orchestration>
```

Figure 15: Example of a simple Orchestration file

4.2.2 Results

IBM PMQ is a great solution for RPM applications but also a missed opportunity due to it only be applicable to a subset of them, namely predictive maintenance of assets. In addition, it does not quite accomplish what we are trying to do with the RPM framework, which is to bridge the operational and the analytical. It possesses some of the components we are interested in for the RPM framework, including two rule engines (a

Predictive Analytics Rule Engine and a Classic Rule Engine), an OLAP Database, and a component with BI capabilities but misses the capability of integrating Actions.

4.3 IBM RPM

The main objective of our IBM RPM case study was to be able to handle any RPM application, not just asset management, and to incorporate direct support for actions. In this case study, we built a simple RPM application related to the detection of patients with a higher risk of cancer recurrence where we had the role of RPM Application Developer. More specifically, we used rules to automatically start the cancer recurrence hospital process for all patients whose risk of recurrence was higher than 60%. For example, if Amelia was 54, female, overweight, and had repeatedly very high blood pressure, the risk of cancer recurrence prediction model would output a risk of 62% which would trigger a rule to automatically start the cancer recurrence business process for preventative treatment. In the context of performance monitoring, the goal of this application is for patients to have a timelier doctor visit when there is a high risk of cancer recurrence thus bridging the analytical to the operational by reacting to events coming in and triggering the follow up preventive cancer process proactively.

4.3.1 Architecture

To create the IBM RPM framework, we adapted the IBM PMQ architecture. In our RPM architecture, three operational sources of events are possible: Forms, Message Broker, and Application Server. Having adapted this architecture from the IBM PMQ, it is easy to see that our source of events for the IBM RPM is the Message Broker implemented from IBM Integration Bus. An Android mobile application to monitor blood

pressure was made to receive data from a blood pressure Bluetooth device and a Bluetooth scale. The mobile application then sent Fact Events to the Message Broker. The Message Broker then sent Fact Events to Application Management component using a REST service.

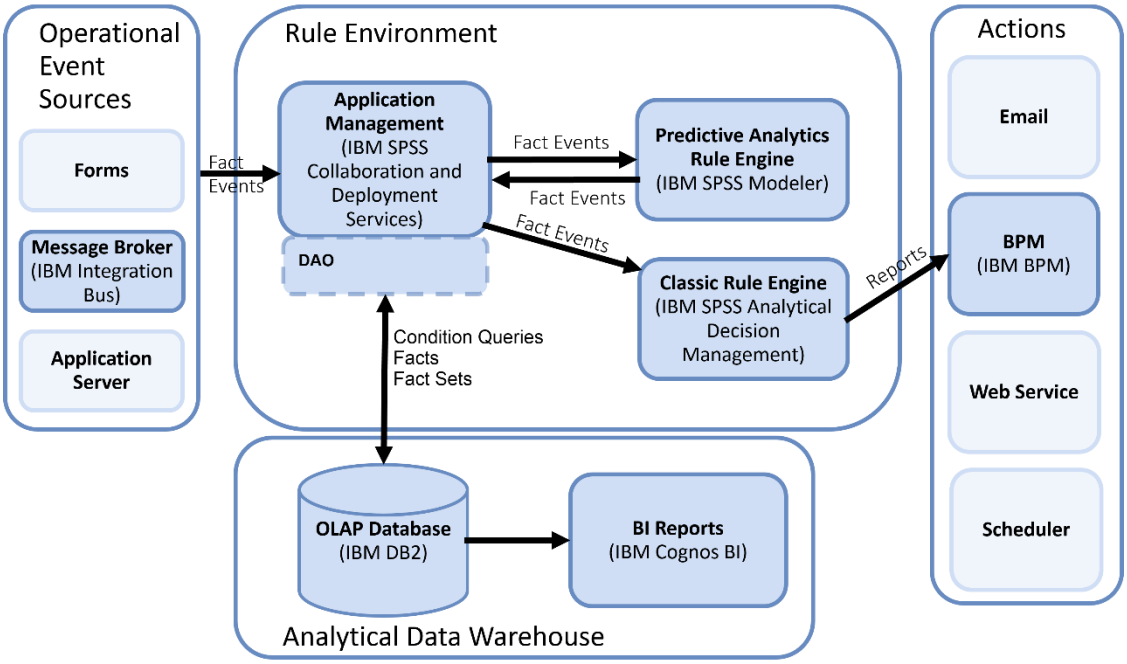

Figure 16: IBM RPM Architecture

The Application Management component in the architecture, as shown in Figure 16, is implemented by IBM SPSS C&D and supported by IBM Integration Bus. It is the main component interacting with the OLAP Database through a built-in DAO. The OLAP Databases were in IBM DB2 supporting BI Reports using the IBM Cognos BI tool, as it is in IBM PMQ.

Furthermore, as briefly mentioned previously, the IBM PMQ possesses two Rule Engines: a Predictive Analytics Rule Engine and a Classic Rule Engine implemented by IBM SPSS Modeler and IBM SPSS ADM respectively. When IBM SPSS Modeler

receives a Fact Event, it is evaluated against a predictive model, which generates new Fact Events (the risk of cancer). The result of this model is then sent to the Classic Rule Engine by forwarding the resulting event through Application Management. The Classic Rule Engine (IBM SPSS ADM) then determines whether a trigger should be sent to IBM BPM, our BPM Action, depending on our predetermined rules.

The BPM adapter was implemented while interning at IBM in order to incorporate rule-processing for triggering external actions. Consequently, we had one available Action to consolidate the IBM RPM framework which was strictly to launch the appropriate business process to respond using IBM BPM. In this case, automatically triggering a follow up preventive cancer process which would automatically populate follow up tasks in the Nurse and Physician inboxes.

4.3.2 Methodology

With this IBM RPM architecture, we created an RPM application that was meant to identify patients at high risk of cancer recurrence. Our RPM methodology was adapted to customizing and extending the IBM PMQ solution. We filled the role of RPM Applications Developer. Other members of the team filled the Business Operations Architect and Performance Analytics Analyst roles. This means that steps 1 through 7 of our RPM methodology, from Figure 10 in chapter 3.4.2, were either preconceived or done by others. This includes the creation of Operational Business Models; and the identification of Strategic Goals, KPIs, Operational Event sources, Alarm States; and the determination of Fact Events, and Business Rules; and the design of OLAP Databases

and identification of the need for a Predictive Analytics Rule Engine for some of the Business Rules.

In IBM PMQ, the OLAP Database is not touched directly. All modifications to generalize it so that it fitted a context other than predictive maintenance of assets had to be done through a configuration file for “Master Data” and had to be aligned with the predefined IBM PMQ “Master Data” Model as seen in Figure 14. In addition, all message flows are defined in the “Orchestration” or the IBM Integration Bus directly, including the ones going to the OLAP Database.

Therefore, in steps 8 and 9, the OLAP Database, Business Rules Sets, and DAO configuration were modified to fit our case study involving patients and healthcare providers by changing the “Master Data” and the “Orchestration” configuration files. For step 10 and 11, the actions needed were triggered in the IBM BPM process to alert the healthcare provider. Step 12 consisted of creating a BPM adapter to communicate with IBM BPM using the available REST APIs but that needs to be used by means of the “Orchestration”.

Step 13 consisted of implementing the rules. In the case of the Predictive Analytics Rule Engine, the predictive model had to be implemented. In the case of the Classic Rule Engine, specific rules had to be written to trigger BPM. In addition, the “Orchestration” had to be modified to properly direct the Fact Events to the right components.

4.3.3 Rule Environment

From a Rule Environment point of view, the one in IBM RPM is somewhat complex. IBM PMQ is more focused on message flows than on the rules that need to be applied to the Fact Events. Three main things had to be taken into account: the "Orchestration", the Predictive Analytics Rule Engine, and the Classic Rule Engine.

For the "Orchestration", this means that all of the Fact Events paths had to be predefined granularly. For example, we needed to explicitly specify that the Predictive Analytics Rule Engine expected blood pressure and weight fact events and that the output was a health risk Fact Event. Thankfully, since the IBM SPSS suite is tightly linked, it did not require an entry to specify the output of the IBM SPSS Modeler into the IBM SPSS ADM. In another entry, we had to specify that the possible output of the Classic Rule Engine was to be sent to the IBM BPM adapter. Nevertheless, it can quickly become very complicated with multiple Fact Events from different sources. In our case, the weight and blood pressure data was combined in one Fact Event.

For the Predictive Analytics Rule Engine, the predictive model had to be implemented. This requires an IBM SPSS Modeler expert that understands predictive analytics and the inner workings of the tool. For the Classic Rule Engine, it is not as complicated but still requires someone that knows how to specify the Fact Event that comes from the IBM SPSS Modeler as an input to the rules in IBM SPSS ADM and the relationship to BPM processes.

4.3.4 Results

Using IBM PMQ, we managed to implement an IBM RPM framework by generalizing the underlying OLAP Database and by adding a BPM Action. Further customizations of the configurations might be required for other scenarios, including in the IBM Integration Bus for added message flows. Adding an Action was not ideal as it required the development of an additional adapter in the underlying IBM PMQ "Orchestration". In short, the usage of the IBM RPM framework is possible and might be suited for big organizations that require heavy duty software to execute their operations but maybe not for smaller organizations. Generalizing the IBM PMQ framework for different contexts is complex due to many premade configurations that need to be modified. The addition of Actions is also complex and so is the definition of message flows in the underlying architecture of IBM PMQ through the "Orchestration". This further complicates the redaction of rules since the RPM Application Developer must be an expert of the IBM PMQ architecture, its jargon (as seen in the "Master Data" model in Figure 14 and the "Orchestration" file in Figure 15), the OLAP Database, the prediction model, the Predictive Analytics Rule Engine, and the Classic Rule Engine. Nonetheless, the rules we implemented in the framework did help us achieve reactive performance monitoring by proactively reacting to events coming in by scheduling timelier doctor visits.

4.4 QuickForms RPM

The complexity and cost of the IBM RPM framework motivated us to investigate prototyping the RPM framework using open source technologies with the components expected by the RPM framework. The QuickForms RPM case study was implemented

using an open source software stack called QuickForms developed at the University of Ottawa (QuickForms (uOttawa), 2015a). QuickForms is an open source application framework to develop mobile applications for monitoring and reporting (QuickForms (uOttawa), 2015b). This framework has been developed to facilitate the development of BI applications and integrates them with an application-specific OLAP database. Figure 17 shows the architecture of the QuickForms framework. QuickForms has three main layers: the app database, the independent middleware, and the app user interface. In this case study, the App database is the OLAP Database from our architectural pattern in 3.4.1, and we have used the independent middleware (QuickForms Service) as a basis for our Rule Environment. The App User Interface is an Event Source where HTML forms can be used to create events.

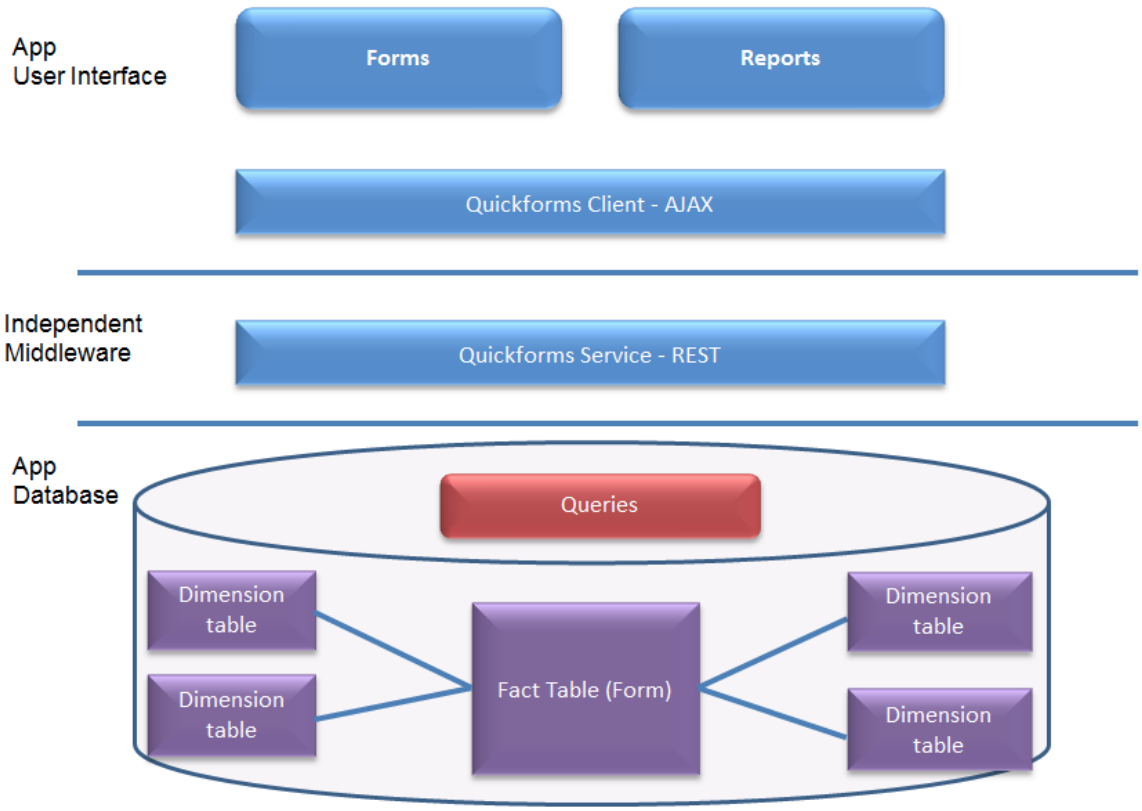

Figure 17: QuickForms Application Framework (QuickForms (uOttawa), 2015a)

In this case study, we built a very simple RPM application named "Celebrate Creation" where we had the role of RPM Application Developer. The goal of this application was to inform women from at risk populations of the progress of their pregnancy and for them to view the information in a timely manner. To do so, we wanted to have a rule that would enable us to send a weekly email reminder to view content associated with the patient's current pregnancy week using a link to the content. For example, if Amelia's due date is October 18th, then, on August 16th, send Amelia an email reminder with a link to week 31 content. In the context of reactive performance monitoring, the rule demonstrated allows us to be proactive in the business process by directly trying to improve the analytics (how often is the content viewed, how many people click the content links, etc.) by directly affecting the operational. Although, this is a very simple RPM application, we show in 4.5, that this particular rule is not easily supported by the Predictive Analytics or Classic Rule Engines.

4.4.1 Architecture

The architecture for our case study is shown in Figure 18, based on the architectural pattern from section 3.4.1. The faded out boxes (Message Broker, Application Server, BI Reports, BPM, and Web Service) were not needed for the Celebration Creation application but are supported by the architecture. The Dashed Black box shows the existing QuickForms server, which we extended to forward Fact Events to our own open source OLAP Rule Engine that we built ourselves, as well as Email service, and Scheduler that we developed during this case study. Existing Rule Engines, Email services, and Scheduler were investigated, but in the end it was deemed optimal to build our own. We deemed it better to build our own OLAP Rule Engine

because it has access to the OLAP Database and that allows us to use Reports in both conditions and actions of Rule, as it will be demonstrated in chapters 4.5 and 4.6.

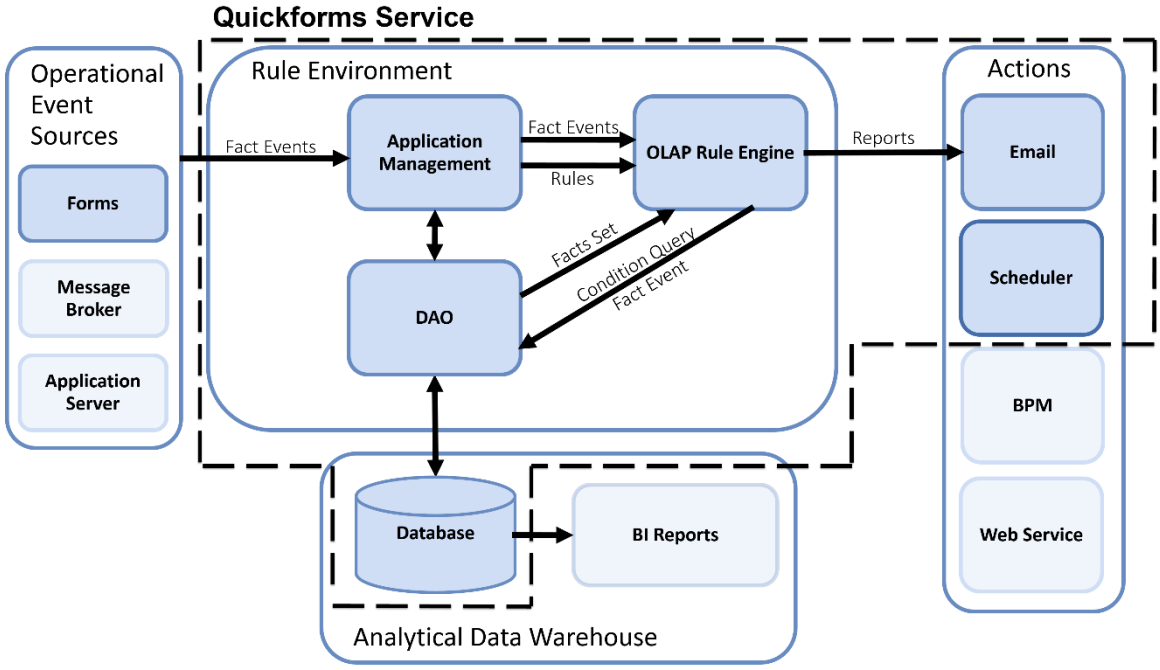

Figure 18: QuickForms RPM Architecture

Direct links to Operational Event Sources, the BI Reports, Web Services, and the BPM were not implemented as part of this case study, but are supported by the QuickForms Service API. For the Operational Event Sources, we used a Form which contained the patient's subscription information including pregnancy due date, email, etc. This Form sends Fact Events to the Application Management. The rest of the architecture was part of the QuickForms Service. The OLAP Rule Engine was developed for this case study, as well as the Scheduler. The Email service was modified to be adapted to the architecture. The Scheduler was programmed to schedule the Email service based on the Fact Events received.

Our DAO is a component which simply interacts with the Database to store facts, update facts, and retrieve Facts Sets. It receives Fact Events and Condition Queries to return the Facts Sets. Compared to the DAO, the Rule Engine has a more significant role in the resulting actions of the framework. The Rule Engine receives all Fact Events that can potentially be affected by a Rule. When a Fact Event is affected by a Rule, the evaluation of the Rule determines whether the Fact Event should be changed, stored, or should trigger an external action.

4.4.2 Methodology

For this case study, all the steps of our RPM methodology accomplished by the Business Operations Architect and the Performance Analytics Analyst had been previously done by others. In addition, the business application "Celebrate Creation" to inform pregnant women from at risk populations of the different stages of their pregnancy containing the Form already existed. We simply had to fill the role of the RPM Applications Developer and implement the needed rules.

In short, our role consisted of identifying the appropriate actions which were notifications that were implemented using Email and scheduled Emails (using a Scheduler), as in step 11 of our RPM methodology. For step 12, we opted to implement the Email service in Java, QuickForms implementation language as well as using Quartz (Terracotta, 2016) for the Scheduler component which was integrated in the QuickForms Service. For step 13, we proceeded to implement the rules that would subscribe a person and trigger the Scheduler.

4.4.3 Rule-Based Environment

Our QuickForms Rule Environment closely follows the defined Rule Environment in the RPM framework. We have an Application Management component which needs to configure the Rule Sets and send them to the Rule Engine. There is also an OLAP Rule Engine that executes the rules against the Fact Events and has access to various action methods and condition functions, sometimes via external services.

Our current OLAP Rule Engine assumes that a Fact Event corresponds to only one Rule Set. When it receives a Fact Event, the Rule Set is identified with the application from which the Fact Event comes from. Subsequently, it is evaluated against all the Rules contained in the Rule Set, in serial.

Using QuickForms, we have a tight link between the data model and the values available to the Rule creator. When creating a QuickForms application, knowledge of the business data model is crucial to create the application. Otherwise, it is impossible to specify the correct fields. For the creation of the Rules, we took a similar approach which assumes that the RPM Applications Developer has complete knowledge of the OLAP Database and the possible Fact Events. In QuickForms RPM, it would be possible to add functionality that could make a list of suggestions for the fields needed to create Rules due to the OLAP Database Integration.

Currently, the RPM Applications Developer needs to have a basic knowledge of Java to be able to use the necessary syntax to write the Rules. However, a library of functions used to write rules have been created which simplifies the task. They use

readable function names which describe their usage. It is important to note that the Rule organization is still not appropriate for an unexperienced Java user to write the Rules.

Conditions are generally written according to events coming in. QuickForms sends its events in a JSON format. Keeping that in mind, predefined functions, part of the "Fact Event functions", have been created to deal with the content of the JSON structure. In particular, two functions are particularly useful. One is "getEventFieldValue" which accepts a string type field name and returns its value. We can then compare this value using "equals" (in the case of a string). The other is "checkEventFieldPresence" to simply determine whether a field is contained in the event or not. With these simple two functions, we can make a lot of different comparisons with enumeration values or convert the JSON field to a numeric value which we can compare with Java's operators.

In terms of the framework, action methods correspond to predefined functions that can be referred to from the Rule class associated with the QuickForms application. For example, a function called "sendEmailFromGmail" has been created part of a class called "NotificationService" providing the Email Action. This function can be used as long as the service has been instantiated and its required parameters have been provided. BPM and Web Services have not been added to this framework but the QuickForms Service API would easily support it.

For our case study, once the star-schemas, business rules, needed facts, and others had been established, as an RPM Applications Developer, we created the Rules needed to trigger the needed Actions. If a Rule is evaluated as true, the Actions following are enacted. In the "Celebrate Creation" business application, we used a StoreOrUpdate

Action, an Email Action, and a Scheduler Action. The first to record the subscription of the patient, the second to notify the patient of their subscription success, and the third and fourth to schedule weekly emails to inform the patient of their pregnancy stages.

4.4.4 Results

Using open source technologies, we have created a much lighter RPM framework than the one using IBM technologies, namely IBM PMQ. In addition, it is a much closer implementation of the intended RPM framework. In the process, we have also created an OLAP Rule Engine which had access to the OLAP Database. This allowed us to create Rules that could use Reports both in the conditions and for the Actions. For the available Actions, we had Email, a Scheduler, and an API supporting Web services which inherently supports BPM. We used an incredibly simple business application to create the rules but as we will see in the examples of 4.5 and 4.6, these rules are not achievable with either a Predictive Analytics Rule Engine or a Classic Rule Engine

4.5 Rule Example 1

In this first example, we will demonstrate an important rule example that could be implemented using the OLAP Rule Engine. In the "Celebrate Creation" application, the goal was to proactively inform women from at risk populations of the progress of their pregnancy. To do so, we wanted to have a rule that would enable us to send a weekly email reminder to view content associated with the patient's current pregnancy week using a link to the content. For example, if Amelia's due date is October 18th, then, on August 16th, send Amelia an email reminder with a link to week 31 content.

Using our RPM Framework with an OLAP Rule Engine, this is not too difficult to achieve since we have a Scheduler:

1. Every Day at 3am:
   SendEmailReminderEvent
2. If SendEmailReminderEvent and WeeklyDueDateReport:
   SendEmailsAction (WeeklyDueDateReport)

The "Every Day at 3am" condition is doable since we have a Scheduler and an OLAP Rule Engine which is linked to actions. The "SendEmailReminderEvent" is simply creating a Fact Event which triggers the second rule. The "If SendEmailReminderEvent and WeeklyDueDateReport" condition checks if the Fact Event being processed is "SendEmailReminderEvent", and if the "WeeklyDueDateReport" returns a Facts Set containing at least one Fact. "WeeklyDueDateReport" consists of a condition query made to the OLAP Database which returns a Report having the following fields: Name, Email, Due Date, and Week. If the rule condition passes, the action "SendEmailsAction" is enacted. This action simply consists of sending an Email for every Fact (patient) in the report "WeeklyDueDateReport". We can simply use an Email template with which we can fill in the fields with the content of the Report previously obtained. This is the same as sending a Report with a template to fill. For example:

"Hello [Name],
Since your due date is [Due Date], here is your weekly content associated with week [Week] of your pregnancy:
http://www.example.com/week[Week]!
Yours truly,
The Celebrate Creation Team"

When we consider reactive performance monitoring, this rule allows us to proactively improve analytics and directly affect the operational. It would not be possible to implement these using a Classic Rule Engine because a Classic Rule Engine does not have access to Actions or the OLAP Database and would therefore not (typically) have access to a Scheduler, and could certainly not be able to retrieve the information required for the report.

Table 1: WeeklyDueDateReport example

Name	Email	Due Date	Week
Amelia	amelia@example.com	2016-10-18	31
Sofia	sofia@example.com	2016-09-20	35
Karman	karman@example.com	2016-12-27	21

On the other hand, the Predictive Analytics Rule Engine is simply not adapted for this task. It does not have rules formulated the same way. Under the premise that the Predictive Analytics Rule Engine works by receiving a Fact Event, processes it with the prediction model (a black box), and outputs a Fact Event according to the predictive model, it would have to be trained with data to know when to send each Email. Presumably, this is possible, if a few weeks' worth of emails had already been sent manually, but this seems unnecessarily awkward.

4.6 Rule Example 2

For our second rule example, we will demonstrate rules that were implemented for the IBM RPM case study using the Predictive Analytics Rule Engine combined with the Classic Rule Engine and analyze how well our OLAP Rule engine could support

them. Our application with the IBM RPM case study was related to the detection of patients with a higher risk of cancer recurrence to improve the timeliness of doctor visits. More specifically, we needed rules to start the cancer recurrence hospital process for all patients whose risk of recurrence was higher than 60%. For example, if Olivia was 54, female, overweight, and a repeatedly very high blood pressure, the risk of cancer recurrence prediction model would output a risk of 62%. Then, the cancer recurrence business process would be kicked off for Olivia. The following summarizes the sequence and rules:

1. BloodPressureEvent in -> Prediction Model -> out CancerRecurrenceRisk
2. If CancerRecurrenceRiskEvent > 60%
   StartCancerRecurrenceBusinessProcess

Where Rule 1 would be implemented by the Predictive Analytics Rule Engine, and Rule 2 would be implemented by the Classic Rule Engine.

Alternatively, we could use an OLAP Rule Engine. Presumably, there exists data that was used to train the Predictive Rules Engine. Presumably, once the predictive model was finalized, one could characterize and document the patterns used by the predictive model. For example, patients whose sex is female and age is over 50 and whose height in cm / weight in lbs is < 1 have a risk of cancer that should be treated proactively. Such characterizations, could be formulated as a FemaleAtRiskReport. Then, we could have a rule similar to the rule in 4.5:

1. Every Day at 3am:
   SendFemaleReportReminderEvent
2. If SendFemaleReportReminderEvent and FemaleAtRiskReport:
   SendEmailsAction (FemaleAtRiskReport)

Clearly, this would not be as optimal or as dynamic as the IBM RPM rule. However, for many applications (including this health care example), it would be a) sufficient and b) more easily understood, maintained and updated as new medical research was done.

Table 2: FemaleAtRiskReport example

First Name	Id Number	Sex	Age	Height(cm)to Weight(lbs) Ratio	Healthcare Provider Email
Olivia	12345678	Female	54	0.97	amanda@hospital.com
Regina	87654321	Female	72	0.93	amanda@hospital.com
Melody	12367845	Female	59	0.94	marc@hospital.com

In the cases where this was not sufficient, the OLAP rule engine would still be relevant as a replacement for the Classic Rule Engine. It would complement a Predictive Analytics Rule Engine better. It would have built in capabilities to trigger actions directly, and it would have more flexibility in scheduling both processing of events and triggering of actions, with better OLAP Database integration.

5 Evaluation

In this chapter, two different comparisons are being presented using the criteria described in 3.3. In 5.1, our two implementations of the RPM framework are compared with the related work described in section 2.6: AFMCP, BAIT, and IBM PMQ. In 5.2, a comparison of the RPM OLAP Rule Engine is made with IBM ODM, SME, IBM SPSS ADM, IBM SPSS Modeler, and Drools. The chapter concludes with our limitations and assumptions in 5.3.

5.1 Comparison of Frameworks

In Table 3: Comparison of Frameworks, we compare the different frameworks presented in this thesis. From left to right, we have AFMCP, BAIT, IBM PMQ, IBM RPM, and QuickForms RPM. The first criterion to compare is the cost of software, hardware, and support. AFMCP has an expensive cost due to the different pieces of IBM software used (BPM, ODM, Integration Bus, and Cognos BI), as well as RTLS from Ekahau(Ekahau Inc., n.d.). BAIT possesses mostly open source components but also depends on proprietary components such as MATLAB. IBM PMQ is an IBM solution which is proprietary and expensive. This product is aimed at bigger organizations. IBM RPM has the same issues as the IBM PMQ and is therefore expensive. The QuickForms RPM framework uses open source technologies only and is free.

Table 3: Comparison of Frameworks

Criteria	**AFMCP**	**BAIT**	**IBM PMQ**	**IBM RPM**	**QuickForms RPM**
Cost of sw/hw/support	Expensive	Mostly Open source	Expensive	Expensive	Open source
Developer Skillsets/Training	Many, specific	Simple, specific	Many, specific	Many, general	Simple, general
RPM Developer Effort	~2 weeks	~1 week	1 week	2 weeks	1 week
Rule Support	Yes	No	Yes	Yes	Yes
Defined Architecture with APIs	Yes	Yes	Yes	Yes	Yes
Predictive Modeling Integrated	No	Yes	Yes	Yes	No
Latency between knowledge and action	Ad hoc	Manual	Manual	Real-time	Real-time
Supports any OLAP Database	Domain specific	No	Domain specific	Yes, but limited mapping	Yes
Methodology for reactive performance monitoring application development	Domain specific	No	No	Yes	Yes

The second criterion consists of the needed developer skillsets and training. AFMCP is composed of multiple proprietary pieces of software that require extensive skillsets. In addition, this framework is not general, but requires skills specific to care monitoring processes. In the case of BAIT, the developer skillsets required are much simpler due to the components they have chosen to be part of their architecture. However, BAIT, much like AFMCP, requires very specific skillsets in air traffic management. As a result, we evaluated BAIT as requiring simple skillsets and specific training. For IBM PMQ, much like the AFMCP, many skillsets are required to use it. In addition, it is once again specific to a domain, in this case, predictive maintenance. IBM RPM also requires many skillsets for the different components. Although the framework is general it still requires knowledge of the specific mapping technique used in order to customize it for general applications. We have evaluated it as requiring many skillsets and general training. The QuickForms RPM requires much simpler skillsets but is still general.

The third criterion is the RPM Developer effort. This criterion is a rough estimate of the time and effort required to implement an application of similar complexity to the cancer prediction application used in section 4.2. The values we list here is very subjective but based on our experience with that application, our experience working with IBM PMQ and QuickForms RPM, our conversations with those who implemented AFMCP, and our reading on BAIT. This criterion is highly dependent of the skillsets required, but we assume the developer would have all skillsets needed. IBM PMQ has an estimate of 1 week due to its high specificity which, given the knowledge and training, almost simply requires plug and play once the framework is up and running to build a working application. The IBM RPM requires much more customization due to its

generality and many skillsets are required to build an application. Because of this, our estimate is of 2 weeks given the experts. The QuickForms RPM has an estimate of 1 week. It is still a general framework but requires much simpler implementations than the IBM RPM once the framework is up and running. We did not get to try the AFMCP or the BAIT so these two are pure estimates based on what we know from the technologies involved and their literature. In the case of AFMCP, although the framework is highly specific, it possesses many different components that need to be configured to be used. This is why we gave an estimate of ~2 weeks. BAIT on the other side has both a specific context and little to do once the framework is in place due to its high specificity on the tools side. For this reason, the estimate is ~1 week.

On the Rule support side, all frameworks except the BAIT possessed some kind of Rule Engine. All frameworks possessed defined architectures with APIs as opposed to if different components had simply been integrated in an ad hoc fashion as it is sometimes ad hoc solutions.

For the sixth criterion which consisted of predictive modeling being integrated, AFCMP did not have that integration, but the other frameworks did. For the QuickForms RPM, the integration is not direct and has not been implemented but it would easily be doable through the API the QuickForms API offers.

For the latency between knowledge and action, we have different degrees at which each of the frameworks fill that criteria. For AFMCP, it has been evaluated as being ad hoc. This is because the reports generated need to be consulted by the right people but there is no control done by the framework to help manage that. For BAIT,

there is a real-time aspect with the graphs generated directly to the operators due to how they have structured their architecture. However, the operators need to determine what to do with the data that is presented to them. For IBM PMQ, the feedback given is real-time but the data is not necessarily given to the concerned people in a real-time fashion as they need to manually consult the work lists containing the work orders manually. The RPM framework using either of the technologies has been designed to be able to deliver the information in real-time with selected specific and useful actions such as emails, or data fed via the database to a notification application.

The eighth criterion was OLAP Database support. The BAIT architecture does not support OLAP Databases as it uses a NoSQL database. The AFMCP and the IBM PMQ frameworks support domain specific OLAP Databases. The AFMCP supports care process models while the IBM PMQ has a premade model which is specifically for maintenance scenarios. Both of the RPM frameworks support OLAP Databases. However, the one using IBM technologies requires extensive work to be changed and adapted to any kind of scenario as a modification of the IBM PMQ part of the RPM framework is needed.

Our last criterion was to determine if there is a methodology for reactive performance monitoring application development. AFMCP had its own methodology but it is highly domain specific for healthcare and is not actually reactive. The BAIT did not possess a particular methodology as they focused on the functionalities and the architecture. IBM PMQ also did not have a methodology for reactive performance monitoring application development. The reason is that their solution is based on

premade templates that developers simply have to fill in to connect things together. As for the RPM framework, we have a methodology which has been defined in this thesis.

5.2 Comparison of Rule Engines

In Table 4: Comparison of Rule Engines, we compared 6 different Rule Engines with 6 criteria identified in section 3.3.2. From left to right, we have IBM ODM, SME, IBM SPSS ADM, IBM SPSS Modeler, Drools, and the RPM OLAP Rule Engine. IBM ODM, IBM SPSS ADM, and Drools are all Classic Rule Engines. IBM SPSS Modeler, is a Predictive Analytics Rule Engine, while both SME and the RPM OLAP Rule Engine are OLAP Rule Engines.

For the dedicated syntax criterion, both IBM ODM and Drools have specific languages associated to write the rules. They use specialized languages for rule writing which makes it easier for the users to write specific rules and also understand them. In the case of the IBM SPSS ADM, they use a GUI which is easy to write simple rules with. The only thing is that their GUI and rules are quite limited because they only support a limited number of operations. On the other hand, the IBM SPSS Modeler also has a GUI which facilitates rule writing, but is specific to predictive analytics applications. In the case of our developed RPM OLAP Rule Engine, and also the SME, there is no specialized rule language, nor is there a GUI associated with it. This highly reduces the usability. But SME and the RPM OLAP Rule Engine have a useful library of condition and action functions that makes it straightforward to write rules. The lack of such a library made it awkward to write rules with the other rule engines.

Table 4: Comparison of Rule Engines

Criteria	**IBM Operational Decision Management (used in AFMCP)**	**State Monitoring Engine (SME)**	**IBM SPSS ADM**	**IBM SPSS Modeler**	**Drools**	**RPM OLAP Rule Engine**
Dedicated Syntax	Yes	Structured API	GUI inter-face	GUI Interface	Yes	Structured API
Open source	No	No	No	No	Yes	Yes
Extendible operations	Yes	No	No	No	Yes	Yes
Directly linked to actions	No	No	No	No	Extendible	Yes
Scheduled Facts	No	No	No	No	Yes	Yes
OLAP Database Integrated	No	Inferred facts but no conditions	No	Yes	No	Yes
Predictive Inferencing	No	No	No	Yes	No	No

Our second criterion for the Rule Engine was the open source aspect. Both of the IBM Rule Engines were not open source but Drools and the RPM OLAP Rule Engine were. The third criterion, extendible operations, is partially affected by the second one. In the case of both the IBM SPSS ADM and IBM SPSS Modeler, we are not able to extend their operations. In the case of the IBM ODM, its basis is in creating all kinds of rules. As a result, it was already extendible. Drools is a similar open source solution but has even more leeway so it is also extendible. The SME is a collection of different components and

is not extendible due to its structure. Since the rules are still developed in Java in the case of the RPM OLAP Rule Engine, the operations are extendible.

The fourth criterion indicates whether the Rule Engine is able to directly link actions as part of rule evaluation. Classic Rule Engines and Predictive Analytics Rule Engines do not have this capability. SME does not have this ability either. Even if Classic Rule Engines follow the "IF <condition> THEN <action>" pattern, the "action" is not an actual action. It simply represents an action that needs to be accomplished, or a current state. However, by developing the case studies, we realized that a direct link to Actions was an important feature. Only the RPM OLAP Rule Engine is truly directly linked to actions. However, we did notice that partly due to its extensibility for operations, it would not be difficult to create an "OLAP Drools" and directly link it to actions.

The fifth criterion is the ability to schedule facts. SME is not designed to schedule events but could be able to do so by receiving scheduled external events. The IBM ODM is not designed to be able to schedule facts although it might be possible by extending its operations. Both the IBM SPSS ADM and the IBM SPSS Modeler do not have this capability at all and would have to depend on an external action or other operational event source to schedule something. Drools has integrated scheduling capabilities as does the RPM OLAP Rule Engine,

The last criterion consists of a core asset for the framework, the integration of OLAP Databases. This is where the SME is interesting because it is highly integrated with an OLAP Database. However, the entire SME depends on one object model linked

to the OLAP Database. This is not realistic when there is multiple applications with different OLAP Databases but it could otherwise be sufficient when considering this criterion. For the IBM ODM and Drools, the OLAP Database is not natively linked. However, it might be possible, especially for Drools, to better integrate it by extending both of the Rule Engines' capabilities. IBM SPSS ADM is highly integrated in the IBM SPSS suite which gives access to the OLAP Databases. When it comes to RPM, the RPM OLAP Rule Engine has been designed with the integration to the OLAP Database in mind, generalizing SME's capabilities. By consequence, it easily fills the criterion with it being extendible to multiple applications and multiple OLAP Databases.

5.3 Limitations and Assumptions

There are several limitations and assumptions that we can highlight in this master's thesis. First and foremost, based on design science research methodology, our framework was not proven. We demonstrated the potential of the RPM framework using two different applications with the IBM technologies and open source technologies which is sufficient to justify further research and more comprehensive trials. We would need additional studies to prove that the framework works for other domains of reactive performance monitoring applications such as project management, device management, monitoring systems, environmental surveillance, etc. In addition, these demonstrations were made in laboratory by myself and were not deployed for actual business use. More studies would be needed to see whether other people can use the framework and not just myself.

With the creation of this framework, we are making an important assumption. We are assuming that OLAP databases with multidimensional models are adequate to represent the context in which automated reactions between the analytical and operational. However, we think that this is a reasonable assumption as it is already well established in literature that OLAP databases are sufficient for analyzing many business applications, especially organizational applications. In particular, even healthcare processes can be analyzed using OLAP Databases as described by the AFMCP. However, we acknowledge that there may be phenomena for which the RPM framework and OLAP modeling may not be appropriate.

Another assumption we are making is that we assume the availability of a stream of events. The stream of events allows us to incrementally store them and process them through the Rule Engine as needed and allows us to be reactive.

6 Conclusions and Future work

6.1 Conclusions

This thesis aimed to close the gap between the analytical, and the operational aspects of businesses. We have managed to do demonstrate the viability of doing so by creating the RPM framework. We managed to demonstrate our RPM framework using two different simple applications. For one, we used IBM technologies to recreate the RPM framework and a prototype application meant to detect "At Risk" patients before they develop a disease. For the other, we developed the RPM framework using open source technologies such as QuickForms and an application to inform pregnant women from at risk populations. The RPM framework we created is a reactive framework for performance monitoring that streams events in an OLAP database instead of processing them in bulk using ETL. Included in the framework, we have created an OLAP Rule Engine which is necessary for our applications. This necessity is due to how the analytical side models business processes using OLAP Databases. To effectively stream events in, an OLAP Rule Engine which could parse events and create meaningful events for our model was created.

In short, our first main contributions is the RPM framework which is composed of an architecture, a methodology, and a Rule Environment permitting appropriate rules composed of conditions and actions. Our second main contribution is an OLAP Rule Engine which has been manifested as being essential in the RPM framework.

However, this framework was only exercised by myself in the context of my master's degree. To prove the efficacy of the RPM framework, we would have to test it with different people using it in different settings. In addition, we only used this framework with two applications in laboratory while assuming that OLAP Databases were sufficient to represent business operations. In conclusion, although we presented an interesting framework, there is still much work to do to prove its efficiency and correctness.

6.2 Future Work

Many instances of future work are possible to continue this thesis' work. For example, although we believe that using OLAP Databases is sufficient to represent operations, it would not be unreasonable to study the extent at which our framework is applicable to other processes and how other phenomena applications are not appropriate to our framework. We also would like to have other developers use our rule engine.

In addition, our RPM framework QuickForms Rule Engine's usability is highly lacking when comparing it to the other Rule Engines mentioned in this thesis. There is great improvement possible in this area, especially in terms of GUI support for building rules and a declarative rule language that would eliminate the need for writing rules in Java. We would like to revisit the Drools rule engine and see if it can be extended to provide integrated OLAP database support and integrated action support for report-based notification.

References

2GC. (2015). Balanced Scorecard. Retrieved from https://2gc.eu/services/balanced-scorecard

Abdelfattah, M. (2013). A Comparison of Several Performance Dashboards Architectures. *Intelligent Information Management*, *5*(2), 35–41. Retrieved from http://www.scirp.org/journal/CTA.aspx?paperID=29589

Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., … Weerawarana, S. (2003, May). Business Process Execution Language. Retrieved from http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf

Baarah, A. (2013). *An Application Framework for Monitoring Care Processes*. (Doctoral Dissertation). University of Ottawa, Ottawa, Ontario, Canada.

Baarah, A., Mouttham, A., & Peyton, L. (2011). Improving Cardiac Patient flow based on Complex Event Processing, 1–6.

Baffoe, S. A. (2013). *A Generic BI Application for Real-Time Monitoring of Care Processes*. (Master's Thesis). University of Ottawa, Ottawa, Ontario, Canada.

Baffoe, S. A., Baarah, A., & Peyton, L. (2013). Inferring state for real-time monitoring of care processes (pp. 57–63). San Francisco, USA,: IEEE.

Bandor, M. S. (2006, September). Quantitative Methods for Software Selection and Evaluation. Carnegie Mellon University. Retrieved from http://www.sei.cmu.edu/reports/06tn026.pdf

Barone, D., Peyton, Li., Rizzolo, F., Amyot, D., Mylopoulos, J., & Badreddin, O. (2015). Model-Based Management of Strategic Initiatives. *Journal on Data Semantics*, *4*(3), 149–165.

Brown, B. (2015, September). 13 more big data & analytics companies to watch. Retrieved from http://www.networkworld.com/article/2976305/big-data-business-intelligence/13-more-big-data-and-analytics-companies-to-watch.html

CENGN. (2016). CENGN. Retrieved from http://www.cengn.ca/

Chamney, A. (2014, September). *A BI Applications Framework for Integrating Mobile Forms Data*. (Master's Thesis). University of Ottawa, Ottawa, Ontario, Canada.

ChengLi, K. (2014). Smarter Process Triggers: Predictive Analytics for Smarter Business Operations. Toronto, Canada: CASCON 2014 24th Annual International Conference on Computer Science and Software Engineering "Smarter Systems of Interactions."

Chieu, T. C., & Zeng, L. (2008). Real-time performance monitoring for an enterprise information management system (pp. 429–434). IEEE Computer Society.

Chisholm, M. (2004). What Are Business Rules and Business Rules Engines? In *How to build business rules engine extending application functionality through metadata engineering* (pp. 1–7). Elsevier Inc.

Claus, T. J. (2011, December). BPM Voices: What is BPMN 2.0 and why does it matter? Retrieved from http://www.ibm.com/developerworks/websphere/bpmjournal/1112_col_jensen/1112_jensen.html

Costa, J., Cecílio, J., Martins, P., & Furtado, P. (2011). Blending OLAP Processing with Real-Time Data Streams. In *Database Systems for Advanced Applications* (pp. 446–449). Springer Berlin Heidelberg.

Dash, D., Giffen, R., Ramakrishnarao, J., & Sreenivasan, L. (2013). *IBM Predictive Maintenance and Quality Technical Overview*. IBM Redbooks. Retrieved from http://books.google.ca/books?id=HDyrAgAAQBAJ

Earth System Research Laboratory. (2016, February). NOAA Earth System Research Laboratory. Retrieved from http://ruc.noaa.gov/hrrr/

Ekahau Inc. (n.d.). RFID-over-Wi-Fi Tracking Systems, RTLS and WLAN Site Survey. *Ekahau Inc.* Retrieved from http://www.ekahau.com/

Exforsys Inc. (2008, May). Business Intelligence Key Performance Indicators. Retrieved from http://www.exforsys.com/tutorials/business-intelligence/business-intelligence-key-performance-indicators.html

Fette, I., & Melkinov, A. (2011). *The WebSocket Protocol* (RFC 6455). Internet Engineering Task Force. Retrieved from https://tools.ietf.org/html/rfc6455

Garzón, M. (2015). *The Rule Language*. *Reverse Engineering Object-Oriented Systems into Umple: An Incremental and Rule-Based Approach*. (Doctoral Dissertation). University of Ottawa, Ottawa, Ontario, Canada.

Hevner, A. R., March, S. T., Park, J., & Pam, S. (2004). Design Science in Information Systems Research.

Hohpe, G., & Woolf, B. (2003, October). Message Broker. Retrieved from http://www.enterpriseintegrationpatterns.com/patterns/messaging/MessageBroker.html

HOra, B. de, & Gregorio, J. (2007). *The Atom Publishing Protocol*. Retrieved from https://tools.ietf.org/html/rfc5023

Huang, Y., & Gannon, D. (2006). A comparative study of Web services-based event

notification specifications (pp. 8–14). Columbus, OH,: IEEE.

IBM Corporation. (2015). IBM ® Analytics Solutions Foundation Developer Guide. Retrieved from http://www-01.ibm.com/support/docview.wss?uid=swg27045826&aid=1

IBM Corporation. (2016a). api_eg1. Retrieved April 28, 2016, from http://www.ibm.com/support/knowledgecenter/SSTNNL_2.5.1/com.ibm.swg.ba.cognos.pmq_solution_guide.2.5.1.doc/api_eg1.jpg

IBM Corporation. (2016b). Documentation - IBM® Analytical Decision Management 8. Retrieved from http://www-01.ibm.com/support/docview.wss?uid=swg27038937#en

IBM Corporation. (2016c). IBM - Canada. Retrieved from http://www.ibm.com/ca-en/

IBM Corporation. (2016d). IBM Business Process Manager. Retrieved from http://www-03.ibm.com/software/products/en/business-process-manager-family

IBM Corporation. (2016e). IBM Integration Bus. Retrieved March 7, 2016, from http://www-03.ibm.com/software/products/en/ibm-integration-bus

IBM Corporation. (2016f). IBM Operational Decision Manager. Retrieved from http://www-03.ibm.com/software/products/en/odm

IBM Corporation. (2016g). Products. Retrieved from https://www-01.ibm.com/software/analytics/solutions/operational-analytics/predictive-maintenance/products.html

IBM Corporation. (2016h). SPSS software. Retrieved from http://www-01.ibm.com/software/analytics/spss/

IBM Corporation. (2016i). WebSphere Business Events. Retrieved from http://www-01.ibm.com/software/integration/wbe/

Kelly, C., Craig, K., & Matthews, M. (2015). Real-Time Predictive Analytics to Estimate Air Traffic Flow Rates (pp. N1–1 – N1–12). Herdon, VA,: IEEE.

Kimball, R., & Ross, M. (2013). *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling* (3rd ed.). John Wiley & Sons.

Kimball, R., Ross, M., Thornthwaite, W., Mundy, J., & Becker, B. (2008). *The Data Warehouse Lifecycle Toolkit* (2nd ed.). Indianapolis, Indiana USA: Wiley Publishing Inc.

Ku, T., Zhu, Y. L., Hu, K. Y., & Lv, C. X. (2008). A Novel Pattern for Complex Event Processing in RFID Applications. In *Enterprise Interoperability III* (pp. 595–607). London,: Springer. Retrieved from http://dx.doi.org/10.1007/978-1-84800-221-

0_47

Lechevalier, D., Narayanan, A., & Rachuri, S. (2014). Towards a Domain-specific Framework for Predictive Analytics in Manufacturing (pp. 987–995). Washington, DC,: IEEE.

Luckham, D. C. (2012). First Concepts in Event Processing. In *Event processing for business: organizing the real time strategy enterprise* (pp. 1–26,49–76). Hoboken, N.J.,: Wiley.

MacCormack, A., Rusnak, J., & Baldwain, C. Y. (2006). Exploring the structure of complex software designs: an empirical study of open source and proprietary code. *Management Science*, *52*(7), 1015–1030.

Massachussetts Institute of Technology. (n.d.). FAA WEATHER SYSTEMS: Corridor Integrated Weather System (CIWS). Retrieved from https://www.ll.mit.edu/mission/aviation/faawxsystems/ciws.html

Mellqvist, P. (2006, May). Don't repeat the DAO! Retrieved from http://www.ibm.com/developerworks/java/library/j-genericdao/index.html

Microsoft. (2016). Microsoft BizTalk integration platform. Retrieved from http://www.microsoft.com/en-ca/server-cloud/products/biztalk/overview.aspx

Middleton, G., Peyton, L., Kuziemsky, C., & Eze, B. (2009). A framework for continuous compliance monitoring of eHealth Processes, 152–160. Retrieved from http://search.proquest.com/docview/502545069?accountid=14701

Mouttham, A., Peyton, L., & Kuziemsky, C. (2011). Leveraging Performance Analytics to Improve Integration of Care (pp. 56–62). Waikiki, Honolulu,: ACM.

Mumbaikar, S., & Padiya, P. (2013). Web Services Based On SOAP and REST Principles. *International Journal of Scientific and Research Publications*, 668–671.

Niblett, P., & Graham, S. (2005). Events and service-oriented architecture: The OASIS Web Services Notification Specifications. *IBM Systems Journal*, *44*(4), 869–886.

NoSQL. (n.d.). NOSQL Databases. Retrieved from http://nosql-database.org/

Oracle Corporation. (2015, October). GlassFish - World's first Java EE 7 Application Server. Retrieved from https://glassfish.java.net/

PARIS Technologies Inc. (2015, December). What is the definition of OLAP? Retrieved from http://olap.com/olap-definition/

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, 24:3, 45–77.

Perens, B. (1999). The Open Source Definition. In *Open sources: voices from the open source revolution* (pp. 171–188). O'Reilly Media Inc.

Pinto, J., Dupree, W., Weygandt, S., Wolfson, M., Benjamin, S., & Steiner, M. (2010). ADVANCES IN THE CONSOLIDATED STORM PREDICTION FOR AVIATION (CoSPA). In *Conference on Aviation, Range and Aerospace Meteorology* (pp. 18–21). Atlanta, GA,: American Meteorological Society.

Pollitt, C. (2013). The logics of performance management. *Evaluation*, *19*(4), 346–363.

Pourshahid, A., Amyot, D. L., Peyton, L., Ghanayati, S. P., Chen, P., Weiss, M., & Forster, A. J. (2009). Business Process Management with the User Requirements Notation. *Electronic Commerce Reserach*, *9*(4), 269–316.

Python Software Foundation. (2016). Welcome to Python.org. Retrieved from https://www.python.org/

QuickForms (uOttawa). (2015a, April). uoForms/quickforms3/README.md. Retrieved from https://github.com/uoForms/quickforms3/blob/master/README.md

QuickForms (uOttawa). (2015b, July). QuickForms Wiki. Retrieved from https://github.com/uoForms/quickforms3/wiki

Red Hat. (2016). Drools - Drools - Business Rules Management System (Java™, Open Source). Retrieved from http://www.drools.org/

Ross, M. (2008, September). Design Tip #105 Snowflakes, Outriggers, and Bridges. Retrieved from http://www.kimballgroup.com/2008/09/design-tip-105-snowflakes-outriggers-and-bridges/

Sun Microsystems Inc. (2002). Core J2EE Patterns - Data Access Object. Sun Microsystems Press and Prentice Hall. Retrieved from http://www.oracle.com/technetwork/java/dataaccessobject-138824.html

Terracotta, I. (2016). Quartz Enterprize Job Scheduler. Retrieved from http://www.quartz-scheduler.org/

The Apache Software Foundation. (2015a). Apache Cassandra. Retrieved April 17, 2016, from http://cassandra.apache.org/

The Apache Software Foundation. (2015b). Apache Storm. Retrieved from http://storm.apache.org/

The Apache Software Foundation. (2016, March). Index. Retrieved from http://activemq.apache.org/

The MathWorks, I. (2016). MATLAB. Retrieved from http://www.mathworks.com/products/matlab/?s_tid=hp_fp_ml

Tulach, J. (2008). *Practical api design: confessions of a java framework architect*. Apress. Retrieved from http://common.books24x7.com.proxy.bib.uottawa.ca/toc.aspx?bookid=29020

van der Aalst, W. M. P. (2013). Business Process Management: A Comprehensive Survey. *ISRN Software Engineering*, *2013*, 1–37.

W3C. (2004, February). Web Services Architecture. Retrieved from http://www.w3.org/TR/ws-arch/

W3C. (2014, October). HTML5. Retrieved from https://www.w3.org/TR/html5/

Weske, M. (2012). *Business Process Management: Concepts, Languages, Architectures* (Second). Springer-Verlag.

zur Mühlen, M., & Shapiro, R. (2010). Business Process Analytics. In J. vom Brocke & M. Rosemann (Eds.), *Handbook on Business Process Management 2* (pp. 137–157). Berlin Heidelberg,: Springer-Verlag.