uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

u Ottawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Abel Tegegne

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.Sc. (Electronics Business Technologies)

GRADE / DEGREE

School of Electronics Business Technologies

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

A Model Driven Application Framework for Managed Processes

TITRE DE LA THÈSE / TITLE OF THESIS

Liam Peyton

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

Morad Benyoucef

Craig Kuzicmsky

Gary W. Slater

Le Doyen de la Faculté des etudes supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

A Model Driven Application Framework for Managed Processes

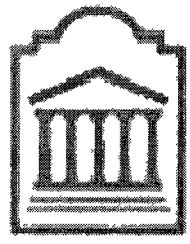
Abel A. Tegegne

Thesis submitted to the

Faculty of Graduate and Postdoctoral Studies

In partial fulfillment of the requirements

For the M. Sc. degree in E-business Technologies



University of Ottawa

Ottawa, Ontario, Canada

May 2010

Your file Votre référence
ISBN: 978-0-494-73753-8
Our file Notre référence
ISBN: 978-0-494-73753-8

Abstract

Organizations are frequently expected to improve operational efficiency through effective decision making. Effective decision making, among other things, requires a configurable business process application platform that can be used to execute, monitor and manage processes involving people, teams, departments, organizational units and their multiple applications. This kind of platform facilitates data collection for performance management and analysis for measuring quality of service delivery and business process optimization. It also facilitates collaboration between process participants, regulatory compliance, configurability to fit to changing requirements and accelerates decision making by sensing actionable events.

Healthcare is a significant area where such a platform can be of great use for monitoring patient care. Patient care monitoring applications are data intensive and require medical guidelines based collaboration between patients, individuals, care providers and organizational units. These applications should provide performance management information to measure the quality of healthcare service delivery processes. These types of applications must also be configurable to adapt to frequently changing types of business events and the process of collect event data.

This thesis proposes a model driven application framework that can be used to create configurable process oriented applications that can be customized to the current needs of an organization. The proposed framework also provides the ability to monitor, manage and report on processes and collected event data. A healthcare scenario is used to illustrate and validate our approach by building a patient monitoring application.

Acknowledgements

I would like to thank all the people who have helped and inspired me during my graduate study.

I especially want to thank my supervisor, Prof. Liam Peyton, for his guidance and help. Throughout the thesis writing, he provided encouragement and good teaching with a great deal of enthusiasm and effort to explain things from which I have learnt tremendously.

It is difficult to find words to express my gratitude to my wife who has been continuously encouraging, loving, caring and understanding and my son who has been constant source of joy. This thesis would have not been possible without their help and understanding.

My deepest gratitude also goes to my family for their unflagging love and support throughout my life. I am indebted to my father for his love, care and for the sacrifices he has made to pave the way for education. I am also forever grateful for the many cherishable moments with my mom. As always, I feel proud of my sister, brother and Enke who have given me their love and unequivocal support.

I would also like to thank all of my friends and my extended family for their support during my studies and for understanding my frequent absence in the course of my graduate study.

I would also like to thank Cyril Soga and Sailesh Patry for recommending me for this graduate studies.

Above all, I thank God for the unconditional love and guidance throughout my life, for providing me with this opportunity and granting me the capacity to complete this thesis.

Table of Contents

List of Figures

List of Tables

List of Acronyms

Acronym	Definition
AOM	Adaptive Object Model
BAM	Business Activity Monitoring
BPEL	Business Process Execution Language
BPM	Business Process Management
BPMI	Business Process Management Initiative
BPMN	Business Process Modeling Notation
BPXL	Business Process eXtension Layers
CEP	Complex Event Processing
CIM	Computation Independent Model
CRUD	Create, Read, Update, Delete
DAO	Data Access Object
DTO	Data Transfer Object
ECA	Event-Condition-Action
ESAS	Edmonton Symptom Assessment System
GUID	Globally Unique IDentifier
KPI	Key Performance Indicator
MDA	Model Driven Architecture
MPAF	Managed Process Application Framework
MVC	Model View Controller
ORM	Object Relational Mapping

PIM	Platform Independent Model
PSM	Platform Specific Model
SQL	Structured Query Language
URI	Uniform Resource Identifier
WfMC	Workflow Management Coalition
WFMS	Workflow Management Systems
XMI	Xml Metadata Interchange
XML	eXtensible Markup Language
XPATH	XML Path Expression Language
XSD	XML Schema Definition

Chapter 1. Introduction

1.1. Problem Statement

There is an increasing expectation on organizations to improve operational efficiency through effective decision making to apply incremental process improvements. Effective decision making requires the ability to monitor processes and provide metrics that can be used to evaluate processes according to the overall goals of the organization. To apply incremental process improvements, organizations must be capable of easily innovating new processes that will enable them to better achieve their goals. There is also a need for organizations to support constantly changing data and process requirements. Financial services, government and healthcare are among few of the areas where such requirements are prominent.

Organizations use multiple business process applications to enable interaction and collaboration between process participants. Exchange of information between these applications and other external sources is very limited. There is also no automated method of monitoring processes to detect and trigger alerts for events that need immediate attention. There is no mechanism in place to ensure process workflows are executed in a consistent manner because process participants have their own individual way of performing a task. There is also no easy way to measure the performance of processes and often it requires going through months of paper work to generate performance management reports.

Existing business process applications were built to fit requirements that were captured at some point in the past when the application was being developed. Through time, these applications become too inflexible to keep pace with the activities of the business they support [Forrester07]. In many cases, even a small change in functionality, such as a new data field, will require changing the application's source code, testing and deployment of the application. There are other cases where applications are designed to be flexible to incorporate a new data field or to embrace a new look and feel but are unable to accommodate a change in the flow of the business process they execute. Some applications also do a decent job of executing their business process and collecting the data required by these processes but do not report over the collected data to provide enough information to enable organizations to be agile by allowing them to monitor events, sense actionable events and then respond accordingly.

To support such requirements and solve the above problems, today's applications are expected to support the ability to collect data and to execute, monitor, manage and facilitate business process optimization in a flexible manner [Forrester08]. Therefore, current application frameworks must be designed with continuous data and process changes in mind to create configurable platforms that support business process agility, monitoring and management. This thesis describes a model driven application framework for building configurable process oriented and data intensive applications that execute a model into a running application and can be used to monitor events and manage processes. A palliative care severe pain management scenario is used to demonstrate our approach.

1.2. Thesis Motivation and Contributions

Current application development techniques focus on code centric application development approaches. Model driven approaches emphasize the transformation of a higher level of model into an application source code. These approaches provide generic application development tools and frameworks to build applications that are compiled from a manually crafted or tool generated source code. These approaches are not suitable for managed process applications because they do not support run time configurability. In addition, every time a managed process application is modified or created, application elements such as web forms, classes, services and database schemas have to be defined and compiled to create the managed process application. Therefore, to facilitate configurability and avoid source code recompilation for business level changes, we need a declarative approach where an application model that contains information about workflows, roles, entities, events, alerts and performance indicators is processed and interpreted at run time into a managed process application without a need for source code compilation.

The contributions of this thesis are:

1. Definition for managed process applications that identifies the key characteristics, services and components.
2. Set of evaluation criteria specific to managed process applications for comparing and evaluating model-driven application frameworks.

3. A model driven application framework for managed processes that enables run time configuration and monitoring of managed processes based on declarative model definitions.
4. Preconfigured services and service-oriented architecture for managed processes.

1.3. Thesis Methodology and Organization

The methodology we employed during our work is design-oriented research. Design oriented research "addresses research through the building and evaluation of artifacts designed to meet the identified business need" [Henver04]. "Purposeful artifacts are built to address heretofore unsolved problems. They are evaluated with respect to the utility provided in solving those problems"[Henver04]. The research work was evaluated using a scenario based case study and comparative analysis to illustrate the advantages of our proposed approach. In particular we took the following steps:

1. Identify and analyze problem.
2. Review literature and industry approaches to problem.
3. Identify the strengths and shortcomings of these approaches.
4. Develop an application framework for managed processes to address shortcomings and solve problem.
5. Evaluate framework and identify improvement areas.
6. Improve the managed process framework by incorporating simple event processing, alert trigger conditions and notifications.

7. Perform palliative care scenario based case study for a severe pain management patient monitoring application.
8. Re-evaluate and compare the proposed framework for managed processes with other approaches.
9. Draw conclusions and identify potential future work.

The thesis is organized as follows:

In Chapter 2, we set the context for the problem we are trying to solve by giving background information on business process management, business activity monitoring, workflow, healthcare applications, simple and complex event processing. We also provide some background on managed processes and model driven development.

In Chapter 3, we discuss the proposed framework for managed processes.

In Chapter 4, we introduce our scenario based case study drawn from patient monitoring application for palliative care severe pain management.

In Chapter 5, we evaluate our approach in comparison with a traditional code-centric two-layered web application development and development using AndroMDA, which is an MDA based extensible code generator framework for application development.

Finally, in Chapter 6 we summarize our contributions and discuss possible future extensions that could be built on this thesis.

Chapter 2. Background

2.1. Managed Processes

Traditional code centric application development focuses on application data and its operation. Application data includes database and object model while its operations include application logic in the form of functions, methods and APIs. Traditional code centric application development concentrates on architectural and technical implementation on how to better organize components, layers and services to realize an application. Knowledge about how an organization performs a task and the conditions that must be fulfilled to perform the task are sprinkled across many applications and different layers within a single application in the form of hard coded application logic. In the face of frequently changing business processes, keeping such applications up to date is an expensive and time consuming task because it requires updating hard-coded application logic and undergoing traditional application development cycles.[Schmidt06][Hailpern06][Weske07]

Another approach for application development is to put business processes that an organization has to perform, at the center of the application development process. "A business process is 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 single organization but it may interact with business processes performed by other organizations" [Weske07]. Business processes enable organizations to better understand who is responsible for doing what, what

happens next in a series of steps to perform a task, where are decision-making activities performed and what rules are used for these decisions.

Putting business processes at the center of the application development effort enables processes to be first defined and to let the process definition to be consumed by an application that interprets the process definition into business processes. This provides an additional layer of flexibility because it provides the ability to modify a process definition in a controlled manner and the consuming application will reflect these modifications accordingly. It also establishes an environment for configurable business processes that can easily be modified to address new requirements and enables organization to monitor and measure the performance of their business processes [van_der_Alast03].

Processes can be fully automated or they can be manual processes with very little automation. Automated processes have both business and technical aspects. Some processes do not need human involvement while others may need human intelligence and judgment[Weske07]. In the context of this research, a managed process is a process that is controlled by an automated system and contains a set of coordinated activities conducted by both people and systems to accomplish a specific organizational goal and is tracked and monitored to measure the efficiency and effectiveness of the process in meeting its goal[Smith03].

2.1.1 Workflow

The Workflow Management Coalition (WfMC) defines workflow as: "The automation of a business process, in whole or part, during which documents, information

or tasks are passed from one participant to another for action, according to a set of procedural rules." [Lawrence97]. A Workflow Management System (WFMS) is defined as: "A system that defines, creates, and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications." [Lawrence97]. WFMS can be used for automating and controlling business processes. WFMS separates application logic from process logic. This increases the visibility of processes within applications and enables one to build configurable process oriented applications.

2.1.2 Business Process Management

Business process management is defined as: "concepts, methods and techniques to support the design administration, configuration, enactment, and analysis of business processes" [Weske07].

The main objective of business process management is to iteratively model business process resulting in a formal representation of business process that enable execution, monitoring, evaluation and optimization of business processes (see Figure 1). "Business process model consists of a set of activity models and execution constraints between them. A business process instance represents a concrete case in the operational business of a company, consisting of activity instances. Each business process model acts as a blue print for a set of business process instances and each activity model acts as a blue print for a set of activity instances"[Weske07]. BPM provides the ability to specify and perform controlled changes to business processes rapidly. It also provides the ability

to measure effectiveness of changes to business processes and these measurements enable organizations to easily identify processes that need improvement.[Smith03]

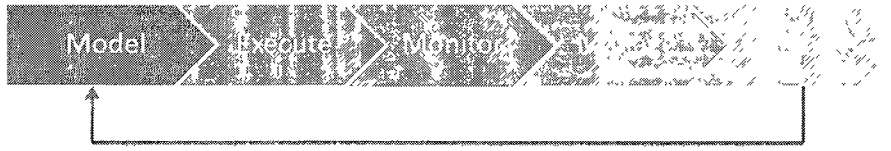


Figure 1 BPM Iteration

A business process management system is defined as: "a generic software system that is driven by explicit process representations to coordinate the enactment of business processes" [Weske07]. Business process management systems can be used to avoid programming to hard-code processes into tailor-made applications and instead components can be assembled to support process orientation, redesign and optimization. For example, WFMS can be used to integrate existing applications and support process change by modifying the workflow model and a rules engine, commonly provided with the WFMS, can be used to make an application's business logic transparent and easy to modify [Smith03].

2.1.3 BPEL

BPEL is an XML-based language that is used to model business processes. This model describes business processes as a "composition, orchestration and coordination" of a set of elementary web services [Jurič06]. A BPEL engine is used to execute the BPEL process definitions. Each BPEL process can in return be accessed as a web service by the BPEL engine. The BPEL definition is based on the W3C standards such as XML, XSD which is an XML Schema that is used to define the structure of an XML document,

XPATH that is used to retrieve XML elements from an XML document, WSDL that is used to describe web services.

2.2. Monitoring & Alerts

2.2.1 Business Process Monitoring

In the course of executing a business process, business process related data and data about the process itself are collected. This data is continuously monitored and inspected to provide visibility and control into business processes. Business process monitoring is an important mechanism for providing accurate information over the status of a business process instance [Weske07][Luckham04]. This information about the state of a business process instance is valuable and allows organizations to identify important business events as a result of changes in the state of a business process and respond to these business events promptly [Luckham04]

2.2.2 Event Processing

An event is an occurrence of some importance inside or outside of an application. More technically, an event can be defined as a record of something that happened in the course of executing a business process [Michelson06][Middleton09]. Event data is an important aspect of a business process because it is produced only when some type of a business activity is performed [Eze09]. These events are used to monitor the performance of a business process. Key performance indicators (KPIs) can be setup to be used as a reference for measuring the efficiency of a business process. They can also be used to generate alerts when the performance of a process strays away from the expected KPI values by more than a certain predefined threshold amount [Middleton09].

Event processing can be classified into two major groups. Simple event processing looks at a single event at a time without referring to any other events. Basic operations that are performed by simple event processing include event filtering, routing and transformation [Michelson06][CEP08]. On the other hand, complex event processing looks at a series of different types of events over a period of time in real-time which includes event filtering, correlating, aggregating and pattern matching [CEP08].

2.3. Enterprise Architecture

2.3.1 Enterprise Application architecture

Enterprise applications are systems that are deployed enterprise-wide to provide necessary information for users to effectively perform their daily tasks. Enterprise applications manage large amounts of persistent data. They manage many users and properly provide concurrent access to data. Data within enterprise applications has to be presented in different formats to accommodate various types of scenarios (data entry to reports) and accommodate different type of users (occasional to regular). Enterprise applications also need to integrate with many other enterprise applications [Fowler03][Trites09].

Application architecture can be defined as "the highest-level breakdown of a system into its parts; the decisions that are hard to change; there are multiple architectures in a system; what is architecturally significant can change over a system's lifetime; and, in the end, architecture boils down to whatever the important stuff is" [Fowler03]. More technically, software architecture is "the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the

relationships among them"[Bass03]. It also serves as high-level structural view of a system to provide a common understanding of the components of a system and their interaction [Fowler03][Microsoft09]

2.3.2 Service Oriented Architecture (SOA)

Service Oriented Architecture (SOA) is an application architecture that is composed of reusable services with well-defined interfaces. These services can be distributed, hosted and owned by different organizations across disparate domains of technology [Huhns05]. These services can be invoked in a defined sequence to realize business processes [Leymann02]. Interaction between services is message based.

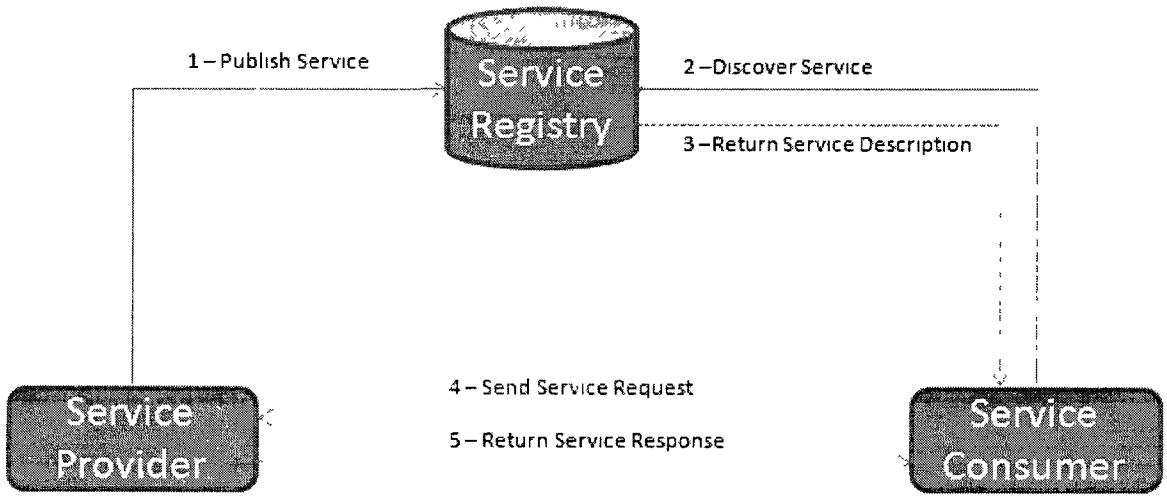


Figure 2 Service Oriented Architecture

The main building blocks of SOA are the service provider, the service registry, and the service consumer. The service provider publishes a description of the services they offer into the service registry. A service consumer looks up a service description with a desired functionality from the service registry. Once a service description is acquired from the service registry, the service consumer will then bind to the service and

send a request to the service provider according to the service contract defined in the service description. After receiving a request, the service provider will return a response to the service consumer.[Huhns05][Leymann02]

Web Service

A web service is defined by the W3C as "a software system designed to support interoperable, machine-to-machine interaction over a network" [WebService04]. Web services are automated software components with well defined interfaces that are can be accessed over a network such as the internet using a URI. Web services have become the natural choice for implementing service oriented architecture. Web services enable different applications running on different machines to exchange data and integrate with one another without requiring additional software other than the web server and operating system software that is used for hosting the web services. Applications that are built on web service technology can exchange information regardless of the language and platform they use [Curbera02].

SOAP – Simple Object Access Protocol

SOAP is an XML based communication protocol for exchanging messages. Web services use SOAP for exchanging messages between a service provider and a service consumer. SOAP is independent of the operating system, programming environment and application framework that is used by the service provider and consumer [Curbera02].

WSDL – Web Service Description Language

WSDL is an XML based specification for describing the public interface of a web service. This public interface includes information that consumer of a web service can

use to locate a web service and exchange messages. Typical information that are included in a WSDL document are the operations of the web service, the messaging protocols that are supported by the web service, XML data type and schema information for request and response messages, binding information about and the end point URI of the web service[Curbera02].

2.3.3 Object Relational Mapping (ORM)

In an object oriented application, object hierarchies are used to hold application data in memory. This application data may have to eventually be persisted into a relation database and can be retrieved from the database and loaded into memory when needed. Object hierarchies in memory can assume various structures while relational database table structure is usually fixed. That is object hierarchies cannot be directly mapped into database tables and this is usually referred to as the "impendence mismatch". ORMs solve this "impedance mismatch" to some level by using a mapping definition that is used to determine to which table(s) and column(s) an object and its properties are mapped. ORMs use mapping definition to generate and use the appropriate CRUD or query SQL statements for the underlying relational database. This simplifies the data access layer of an application since no SQL statements are required and are instead replaced with some sort of a mapping file that contains the class-to-table, property-to-column and other relevant mapping information[Keller97][Ambler97][Fowler03].

Object relational mapping definition is either created by an application developer or it can be generated on the fly at runtime. When the type of object hierarchy is not known at design time, as in the case of interpreted model driven applications, the object

relational mapping information can be generated after the object hierarchy is read from the model and identified at runtime. Hibernate (or its .NET equivalent NHibernate) is one of the well known ORMs and it is used in this research. NHibernate uses an XML format as a primary method of storing object relational mapping information [NHibernate10].

2.3.4 Adaptive Object Model (AOM)

AOM is one approach that is usually used by dynamic and adaptive systems to represent flexible object hierarchies and structures that can be altered at runtime. AOM "is a system that represents classes, attributes, relationships and behavior as *metadata*" [Yoder02]. A *metadata* [model] is used to represent the system behavior. Because the model is interpreted into a running application instance, any changes to the model will result in the modification of the behavior of the system [Yoder02][Yoder01B]. This type of modeling, AOM, is therefore a good candidate for a model driven method of representing in memory data and class/object relationships at runtime. Rendering, business rule (Strategy), and ORM based persistence methods can be used to incorporate additional flexibility in AOM frameworks [Yoder01A][Welicki08][Welicki09].

2.4. Model Driven Engineering (MDE)

Model driven engineering is a software development methodology where models are the core software artifacts that are produced during the software development process. The models can either be directly interpreted into a running application instance or they can be used to generate application source code for a particular programming language which will later be compiled into a software component or application.

Model Driven Architecture is OMG's approach for separating the specification of how a system operates from the details of how it is implemented on a particular technical platform [OMG10]. Particularly, MDA isolates the business and application logic of an application so that it can be defined and model independent of the target platform technology. It uses a higher level modeling language, such as UML, to specify model of an application's business functionality and behavior, independent of the target platform. These models can then be transformed into another set of models, application code and components that are specific to a particular platform [OMG10][Truyen06][Schmidt06].

A model is a visual or textual description of a system and the environment over which the system operates. MDA uses a model of a system to realize the operational system. MDA has three viewpoints. A viewpoint is defined as: "an abstraction technique for focusing on a particular set of concerns within a system while suppressing all irrelevant detail. A viewpoint can be represented via one or more models" [Truyen06]. The three viewpoints are:

1. *Computational Independent Model (CIM)* is a higher level and abstract model that represents the model of a system without any reference to computational details [Truyen06].
2. *Platform Independent Model (PIM)* is a model that describes the behavior and structure of a system regardless of the specific technical platform for which the system is targeted for [Truyen06].
3. *Platform Specific Model (PSM)* is a model that contains necessary detail about the particular target platform so that application developers can use

modeling tools to generate and build the operational system from the model [Truyen06].

In MDA, different modeling tools can be used to define the CIM and PIM. These modeling tools are used to transform the PIM into PSM for a particular target platform and/or programming language such as Java, .NET/C#, .NET/VB.NET etc... The modeling tools will eventually transform the PSM into source code for the selected target platform and programming language. The generated source code is compiled into a component or an executable application using a regular software development environment for the target platform (see Figure 3).

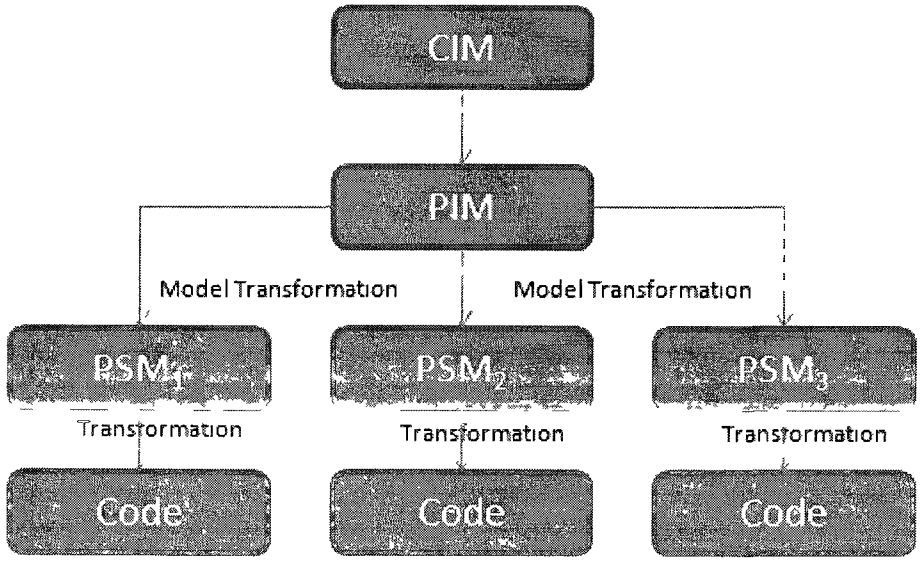


Figure 3 Model Transformation in MDA.

2.4.1 AndroMDA

AndroMDA is an extensible open source Model Driven Architecture (MDA) framework that generates source code from UML models. AndroMDA uses plug-in

components called cartridges to transform model elements into actual source code. Cartridges provide the ability to process model elements that are decorated with particular UML stereotypes (i.e. <<*Service*>>, <<*Entity*>>, <<*Enumeration*>>, etc.) to generate source code using templates that are defined within the cartridge (see Figure 4) [AndroMDA10]. The source code may be generated for any platform e.g. J2EE, .NET, Spring, Hibernate etc. AndroMDA and its cartridges can be extended to support source code generation for any programming language and any application architecture. AndroMDA is also capable of generating any type of text output, such as web pages, database scripts and configuration files that are required for the target platform [AndroMDA10].

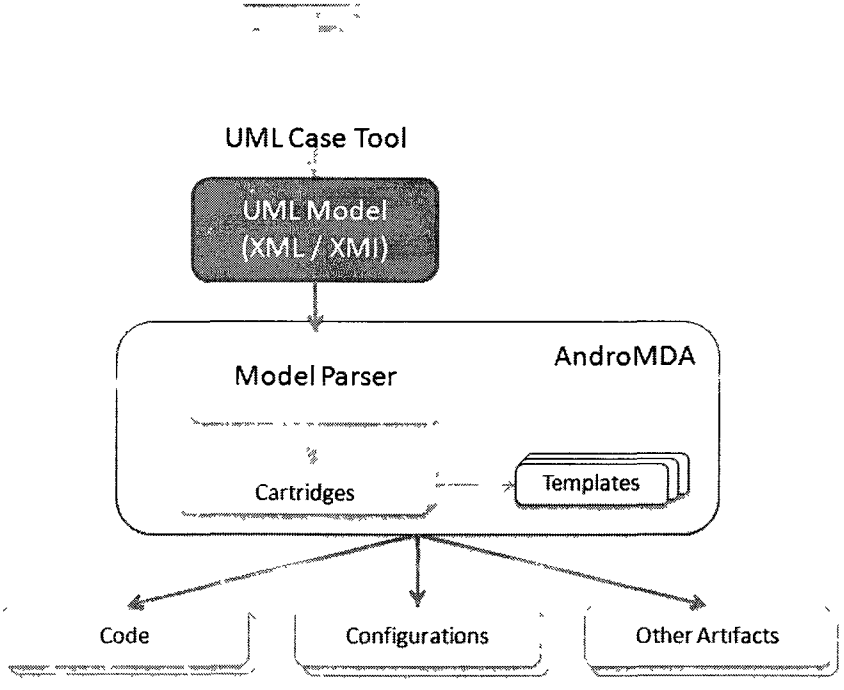


Figure 4 The AndroMDA Framework.

2.5. Multi-Dimensional Modeling

2.5.1 Dimension

Dimensions are information about things or objects, such as products, employees, equipment, patients and the like, that can exist without ever being involved in any type of a business event [Kimball02].

2.5.2 Facts or Events

Facts or events represent information that is captured as a result of a business activity. A fact data marks a business event involving dimensions. For example, a sale of a product would be a fact that records the employee who sold it, what product was sold, the customer that bought the product. In this case, the product, employee and customers are dimensions that were involved in the business event of the product sale. Facts also record other measurements of the business event. In the product sale example, such measurements can include, the quantity sold, price per item, total price, the date of the sale and other comment or description about the sale.

2.5.3 Multi-Dimensional Model

Star schema is a classic method of organizing database tables that are optimized for querying and analyzing data by reporting tools for the purpose of generating reports. Star schema is a multi-dimensional model that has a fact/event table that is related or joined to a number of dimensions. Star schema simplifies the data model because it uses smaller number of database tables. A star schema does not need multiple table joins to generate a result set for reporting. Using a star schema simplifies report generation when

using relational query tools and increases the speed of how fast results are generated for reporting [Kimball02].

2.6. Healthcare

2.6.1 Healthcare Information Systems

Healthcare information systems are systems that are used by patients and healthcare professionals in the process of care delivery. They are used to route clinical information that includes prescriptions, referrals and also serves as a repository of patients' medical, demographic and personal data.

One segment of healthcare information systems is patient care monitoring applications. Patient care monitoring applications are applications that are designed for the specific purpose of monitoring the healthcare delivery process to ensure the well being of patients and vulnerable people. Business process management systems can be employed to automate, manage and iteratively optimize patient care delivery processes. At the core of BPM is an engine that automates and manages process to monitor events as a result of executing business activities of a process. Such an engine can be used to build patient care monitoring applications that allow organization to detect and act to important events efficiently while executing processes in a consistent manner to improve patient safety and quality of patient care.

2.6.2 Palliative Healthcare

The 2002 World Health Organization (WHO)'s definition of palliative care is an "approach that improves the quality of life of patients and their families facing the

problem associated with life-threatening illness, through the prevention and relief of suffering by means of early identification and impeccable assessment and treatment of pain and other problems, physical, psychosocial and spiritual" [WHO10] . It is a "care that is provided to patients at end of life when curative therapies are not an option" and the focus of care is towards improving the quality of life of the patient such as providing pain relief [Liu08].

Palliative care requires interdisciplinary collaboration between different healthcare professionals such as doctors, nurses, case managers and the patient. This makes palliative care an ideal domain for studying managed processes because it can be used to facilitate the collaboration between teams, can be used to proactively monitor and trigger alert notification to promptly address situations that lower the patient's "quality of life" (such as severe pain) and measure the overall performance of the palliative care delivery process of teams.

Chapter 3. Model Driven Managed Process Application Framework

This chapter describes our proposed model driven managed process application framework (MPAF). In section 3.1 we present an overview of existing application development approaches highlighting the challenges one faces when developing managed process applications. In section 3.2 we will identify the criteria for a managed processes application framework. In subsequent sections we will present our proposed model driven managed process application framework (MPAF), including the models for different aspects of managed processes that it supports; the architecture and pre-defined services in the application runtime environment for our framework; as well as a discussion of software engineering issues and methodology relevant to the framework.

3.1. Overview

Many organizations today still use paper based processing to some extent. They are also continuously implementing application systems to automate some of their daily activities. These implementations are usually designed to fit requirements that were captured in the past and are not flexible enough to accommodate new business changes that surface when organizations grow and evolve. As organizations evolve, these application systems are expected to provide more functionality than just simply automating a manual task. Application systems are expected to support managed processes to facilitate data collection, monitoring, performance management, interaction with other external systems and collaboration between different stakeholders.

Current application systems emphasize the collection of data but do not facilitate the coordination of activities among stakeholders. Although application systems are used to manage documents and records, paper based documents are still used to pass information among teams and organizational units. Paper based communication is often slow and prone to errors. Paper based communication also makes it difficult to decrease the variation in taken sequence of steps taken to execute a process.

In the course of executing a business process a series of business events are produced. For example, in a typical healthcare scenario, the patient sets up an appointment, goes to the healthcare provider, registers for a visit, gets examined, receives a prescription for a treatment, undergoes the treatment and follows up on progress. All of the above activities are some form of business events. Many of these events are simple and do not need further inspection and processing. However, some business events may require further processing. For the healthcare example above, the drug prescription business event needs to be checked for any allergy, drug interactions or side effect warnings. Current application systems do not have the mechanism to monitory business events and alert organizations about any warnings that they must be aware of and assist them in their decision making. Detection of such events and alerting is currently done manually mostly by going through a pile of paper work and electronic records.

Currently applications do not also provide information on how well an organization is performing its task to achieve overall organizational goals. These applications are not capable of providing indication of which task or organizational unit

needs improvement to increase performance. Usually, organizations manually process records periodically to generate performance reports.

Organizations can also have frequently changing processes. For example, when a performance measurement provides an indication that a process needs improvement, the process has to be changed and optimized to improve its related performance measurement. This process redesign and optimization may be done many times iteratively to achieve desired performance levels. Organizations, their units and departments can differ on the data they collect and the process they follow to perform their tasks for the same type of task or service. Existing applications are not configurable enough to be easily modified to address changing business process requirements. Often, when a department or an organizational unit requires a new software application, it is developed from the ground up instead of configuring existing applications to fit to current needs.

Current application development methods follow traditional methods. The traditional method of application development is code-centric where code artifacts are developed to handle different aspects of an application. These code artifacts handle the presentation, data persistence, business logic and other aspects of an application. When using code-centric application development, application logic and business rules are not easily identified because they are spread over the application code. Processes are not transparent and standardized as they are not defined declaratively and are only identified by looking at the sequence of operation that users follow to achieve their work. Creating

new applications based on existing ones also requires considerable amount of effort with minimal reuse of the existing application.

To address the above challenges we need to describe the different aspects of an application to define a model based description of the application so that a predefined run time environment can take this application model as an input to translates it into a functional application instance that can be used by end users. This enables different aspects of an application, such as its business processes, to be configured dynamically by simply modifying the application model.

Our proposed approach uses a model driven application framework with preconfigured services that facilitates data collection, business process based collaboration, event monitoring, alert notifications and performance measurement in a configurable manner. Model driven development puts application model at the center of the application development process. The core concept behind model driven development is the use of an artifact to model an organization's business activities and to use this model to drive the organization's application system and evolve both the model and the application systems as new requirements surface when organization evolves.

Most standard MDA tools such as AndroMDA generate application code from a model. Generated code is highly customizable, can target any platform or programming language, is easier to debug and is usually perceived simpler. Our proposed approach, however, interprets the model definition at run time into executable components. Unlike code generation, model interpretation does not require going through the build-test-deploy application development lifecycle when a model is changed. When using model

interpretation, it is also possible for a model definition to be changed at runtime. To capture process, participants, event monitoring and performance management related information for a managed process application, our proposed approach uses model elements for workflows, roles, entities, events, alerts and performance indicator. It also uses a runtime environment that is used to execute these model elements into an application instance.

3.2. Framework Criteria

We identified the following criteria based on the literature review in Chapter 2 and the experience that we acquired in the course of studying the scenario for a palliative health care severe pain management patient care monitoring application in Chapter 4. The use of design oriented research methodology also allowed us to identify these criteria by iteratively looking at industry approaches to problems, the shortcomings and strength of these approaches to problems, and evaluation of our framework to identify improvement areas. These criteria will be used to evaluate model driven managed process application frameworks and to identify pros and cons of the different approaches for developing managed process applications.

3.2.1 Models

1. **Business Level Models**: Application model can be very generic and targeted for general purpose application development using relational database schema, object oriented representations, UML class and activity diagrams to capture the static and dynamic behaviors of an application. Or it can be very specific model that is targeted for a particular domain where concepts of the domain are well

defined and known before hand. The more a model is specific to a domain, the easier it becomes to develop an application and the model becomes more deterministic of how the resulting application will behave to model changes. Managed process application frameworks should consider the specificity of business level models to evaluate how well a framework can be used to represent workflows, roles, entities, events, alerts and performance indicators.

3.2.2 Application Runtime Environment

1. **Business Process Workflow**: An organization's daily tasks and activities involve the participation of various stakeholders with different roles where roles represent a participant's organizational position. Managed process application frameworks should have a runtime environment that is enabled with workflow execution to allow interaction of systems (automated activities such as web service invocation) and people (human activities such as manual event data collection) equivalently where transitions between activities can be delayed over a long running process. Organizations can use multiple applications to perform a business process where these applications can be legacy applications, home grown applications or line of business applications. Therefore, managed process runtime environments should also be designed to allow external applications to submit event data for an existing workflow or even start a new workflow.
2. **SOA**: SOA provides a common and standardized communication framework for services that are exposed by a service provider without exposing the

technical implementation of the service. This enables business processes to be composed of multiple services or a single service to be reused within multiple business processes. The internal implementation of a service can also be modified without directly affecting business processes that consume the service. Managed process runtime environments should make use of SOA to take advantage of service reusability, to be able to communicate with heterogeneous environments using web services, to allow externally located web service implementations to be modified behind the scene without affecting the managed process, enable flexible service composition within a managed process and expose web services to allow external applications to initiate managed processes and submit event data for managed processes. More specifically, managed process runtime environments should expose persistence, presentation, workflow, monitoring and security related pre-configured services.

3. **Configuration**: Business processes change frequently to address new requirements. Processes may change because of various reasons such as a new activity is introduced in a process, an activity is removed from a process, new services are introduced for use in a process, the type of data that is collected for a business event has changed or the trigger conditions for generating an alert notification need to be updated. To address frequent process changes, managed process run time environments must support the run time configuration of different aspects of a managed process application including changes to workflows, roles, entities, events, alerts and performance indicators.

4. **Alerting and Monitoring**: In the course of executing managed processes, event data is collected. These events should be processed in order to get a better insight into a business process and to take effective measures in response to important events in a timely manner. Managed process run time environments should support event monitoring and processing. Managed process run time environments must also be capable of logging additional information about events, analyzing events to identify important events and trigger alert for these events to notify process participants to proactively take effective actions in response to a particular event. Managed process runtime environments must also be capable of processing performance measurement and provide some form of visual feedback to display performance measurement values according to some pre-defined performance indicators.
5. **Data Models & Reporting**: Reporting is one of the key criteria of an application framework that is used to collect data. In addition, in a managed process application, performance management reports are important because they are continuously used for decision making and to assess the performance of a particular process to determine if the process requires optimization. Managed process runtime environments should facilitate flexible data model and performance indicator reporting by using pre-configured event based dimensional data models.

3.2.3 Engineering

1. **Avoid Recompilation for Business Level Changes**: Model driven application frameworks use a model definition that is translated into an executable application. The two main model driven approaches for translating the model into an executable application are run time interpretation of the model and generating source code that can be compiled into an executable application. Managed application frameworks should increase productivity by minimizing the development effort and cost that is associated translating a model into an executable application to efficiently handle frequent business level changes.
2. **Coding Complexity**: Coding complexity is an engineering criterion that measures the complexity one has to face when developing an application. In traditional application development, the application developer has to write programming language based code to handle all aspects of an application, from presentation, business layer, data access layer and database layer. Application developers can also use model driven code generation approaches where a model is created and transformed into programming language based code which the application developer has to modify, maintain and compile to create the end result executable application. Managed process application frameworks should alleviate any coding complexity and must enable to the application developer to build new application by simply applying modifications to the application model.
3. **Model Abstraction level**: Model abstraction levels affect the development effort when developing managed process applications. Model abstraction level

can be low but generic as in the case of traditional development where database schemas and programming language (such as C#) based object models are used. UML models such as class and activity diagrams are used in many model driven approaches. UML models are have a medium level abstraction level and generic in a sense that they are used for modeling any time of application. Managed application frameworks should raise the abstraction level by using business level model elements to minimize the engineering effort when developing managed process applications.

4. **Tool support**: To some degree, efficiency of the application development process depends on the tools and frameworks that are used for development. Development tools can be complex, may require specialized tools, or setting up the development environment can be challenging. Some development tools can support different programming languages and in the case of model driven development, these tools can provide additional functionality such as model verification. As any other application framework, development tool support should be used to evaluate and assess managed process application frameworks.
5. **Learning Curve**: High learning curve can have an impact in the engineering effort when developing managed processes. Learning curve is related to the time it takes for application developers to be comfortable with the tools, components, modeling and development environment when developing an application. Managed process application frameworks should take learning curve criterion into consideration to evaluate and compare with other frameworks.

6. **Code Reuse for Similar Applications**: Managed process application frameworks should be reusable to create similar applications that are used to facilitate event data collection, workflow based collaboration, event monitoring, alert notifications and performance indicators. The engineering effort for creating the first application is relatively high because it is very likely that it will also involve creating the manage process application framework. However, creating subsequent class of applications should have lower engineering effort as many, if not all, assets and services of the managed application framework are reusable. This also helps in minimizing the engineering effort to maintaining applications. This criterion of "code reuse for similar applications" should be used to measure how well-suited managed process application frameworks are for creating applications that share a common and managed set of features.

3.3. Framework

Figure 5 is an illustration of a high level overview of our model driven managed process application framework. MPAF has model definition for workflows, roles, entities, events, alerts and indicators. This model definition is read by the runtime environment to execute an application that is used to create workflow instances and execute workflow activities according to the workflow steps defined in the model definition. Upon executing a workflow activity a request for collecting an event data may be sent to the runtime engine. Event data for a workflow can be submitted using interfaces of predefined workflow service. If the runtime environment gets a request, from its web user interface, to open a workflow that is waiting for an event data, the

runtime environment displays a web form that can be used to submit the event data for the workflow. That is, event data can also be submitted using the web form that is rendered for a particular workflow instance.

When an event data is submitted, the event data is sent to a monitoring service that will check the event for alert notification conditions. If the event satisfies the alert notification condition, an alert notification is created for the event. The monitoring service also checks all performance indicators that are related to the currently submitted event and if the performance indicator value is in a range of value that requires immediate attention, an alert will be created for the performance indicator. When an event is submitted, a persistence service is used to store the event data in a dimensional model based database using a persistence service. MPAF also collects run time data on workflow progress, alerts, events that can be used to generate reports for managing the process. In the next section we look at the detail of our model definition elements, the runtime environment for MPAF and the engineering considerations that must be taken into account when using MPAF..

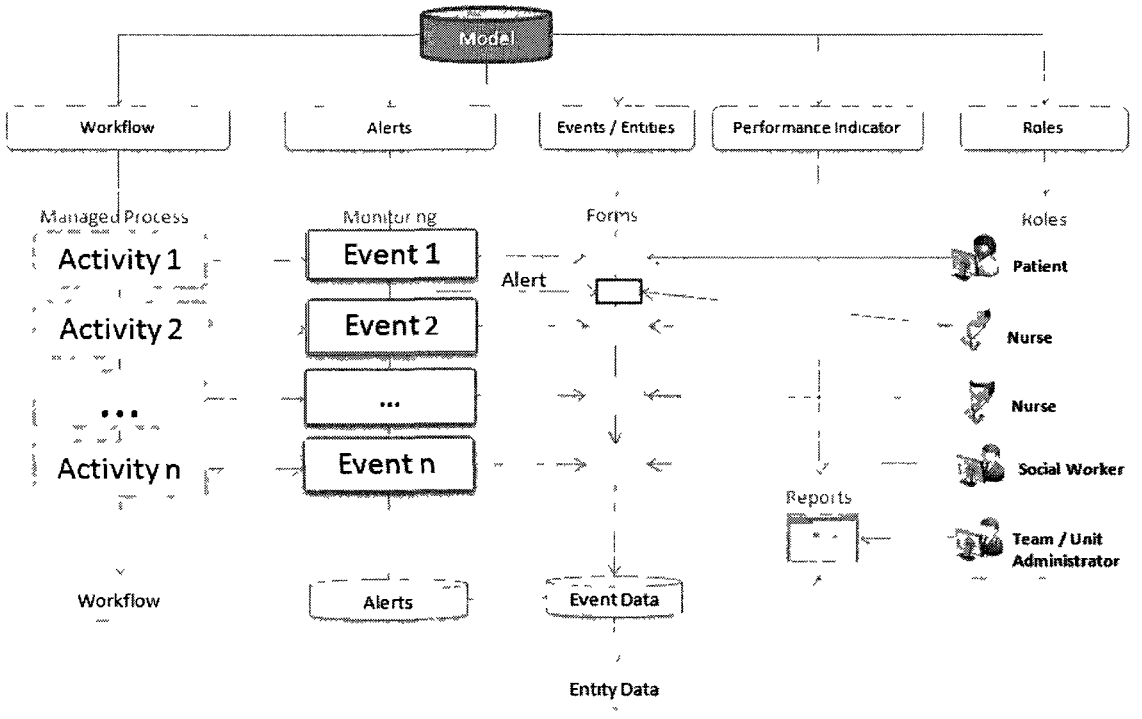


Figure 5 Model Driven Managed Process Application Framework

3.4. Models

The model holds information about the different elements of the application which is later read by a runtime environment and executes the model to render an application (see Appendix A for a model that is used in chapter 4 for the severe pain patient monitoring application). The model holds information about workflows, roles, entities, events, entities, alerts and performance indicators.

3.4.1 Workflows

A workflow definition holds a sequence of activity definitions that will later be used to execute a managed process at run time. A workflow can contain and orchestrate different types of activities. An activity can be a standard activity that is by default

provided as part of the workflow engine or it can be a custom activity that is custom created to implement a specific behavior that is not by default provided by the workflow engine. In MPAF we have created a custom workflow activity, that is called "*CollectEventDataActivity*", to represent event data collection activity. The workflow model allows custom activity definition such as "*CollectEventDataActivity*" to be listed in the sequence of activities that a workflow executes. A typical workflow model definition is as shown below.

```
<Workflow name="ESASWorkflow" description="ESAS">
  <CustomSequentialWorkflowActivity name="ESASWorkflow">
    < CollectEventDataActivity displayName="Submit ESAS"
name="submitESAS" eventTypeName="ESAS" roles="Nurse;Patient" />
    <CollectEventDataActivity displayName="Submit ESAS
Consultation" name="submitESASConsultation"
eventTypeName="ESAS_Consultation" roles="Nurse" />
  </CustomSequentialWorkflowActivity>
</Workflow>
```

In the workflow model definition shown above we have definitions for two sequential "*CollectEventDataActivity*" activities. The "*CollectEventDataActivity*" model element has the "*displayName*", "*eventTypeName*" and "*roles*" properties that should be configured in the model definition. The "*displayName*" is used to identify the workflow name. The "*eventTypeName*" is used to identify the event for which data will be collected when executing the "C*ollectEventDataActivity*" activity. The "*eventTypeName*" value is also used to identify the type of event for which web form with input elements that are used to collect information for the particular event type. The "*roles*" attributes defines the list of role names that are allowed to execute a particular activity in a workflow.

3.4.2 Roles

The MPAF model definition can have a list of role names that are used in the application. These role names are used in other model elements to identify the roles that have access rights to perform an action that is represented by the particular model element. For example, in the workflow model definition, a particular workflow activity can have a list of roles that are allowed to execute the activity in the "*roles*" attribute of the workflow activity model definition. The following an example of a roles model definition that was taken from the patient monitoring application scenario in Chapter 4.

```
<Roles>
  <Role name="Patient"/>
  <Role name="CaseManager"/>
  <Role name="Nurse"/>
  <Role name="TeamAdministrator"/>
</Roles>
```

3.4.3 Entities

Entities are constructs that are used to hold application data. They are used to describe business entities of an organization that can exist without ever being involved in a business activity. They commonly represent categorical and hierarchical information such as time and location. Entity data are also used for lookup or reference by an event data. An entity model definition can have one or more model definition for its properties. Each property of an entity has a name, data type, constraints (required, maximum length, size etc ...), data source (lookup, pick list or manually edited by user).

```
<EntityDefinitions>
  <EntityType name="Patient">
    <PropertyType name="DateCreated" type="System.DateTime"/>
    <PropertyType name="Patient ID" type="System.String" />
    <PropertyType name="OHIP Number" type="System.String" />
    <PropertyType name="First Name" type="System.String" />
```

```
      <PropertyType name="Last Name" type="System.String" />
      <PropertyType name="Gender" type="System.String"
isPickList="True" pickList=";Male;Female"/>
      <PropertyType name="DOB" type="System.DateTime" />
      <PropertyType name="Religion" type="System.String"
isPickList="True" pickList=";Christian;Hindu;Other" />
      <PropertyType name="Other - Religion" type="System.String" />
      <PropertyType name="Family Physician" type="System.String" />
      <PropertyType name="Referring Physician" type="System.String" />
      <PropertyType name="Emergency Contact" type="System.String"
isMultiLine="True"/>
    </EntityType>
  </EntityDefinitions>
```

The above XML fragment is a model definition for a "*Patient*" entity as specified in the "*name*" attribute of the "*EntityType*" element. The "*Patient*" entity has property definitions such as "*Patient ID*", "*First Name*", "*Last Name*", date of birth ("*DOB*") and many others. The data type is also specified using the "*type*" attribute. Typical values for the "type" attribute in the model definition include, "*System.DateTime*" for date or time valued properties and "*System.String*" for text valued properties. Each property of an entity has a name, data type, constraints (required, maximum length, size etc ...), data source (lookup, pick list or manually edited by user). For example, because the "*Gender*" property is a "pick list" (*"isPickList="True""*), it can only assume values that are described in the "*pickList*" attribute. That is the "Gender" property can only assume one of the following values "", "*Male*", "*Female*".

3.4.4 Events

Events are similar to entities in that both have definition for the properties that belong to them. However events are used to record actual facts or measures in a managed process and are the point of interest for data collection during the execution of a managed process. Event data are only recorded in the context of performing a workflow

activity within a managed process. The following is an event model definition that is used for assessing patient's pain level and other well being factors.

```
<EventType name="ESAS">
      <PropertyType name="WorkflowInstanceID" type="System.String"/>
      <PropertyType name="WorkflowActivityName" type="System.String"/>
      <PropertyType name="DateCreated" type="System.DateTime"/>
      <PropertyType name="DateSubmitted" type="System.DateTime"/>
      <PropertyType name="Patient ID" type="System.Int32"
isLookup="True" lookupName="Patient" lookupDisplayFormat="{Patient ID}
- {First Name} {Last Name}" calculated="true"
value="System.User.[Patient ID]"/>
      <PropertyType name="Pain" type="System.Int32" isPickList="True"
pickList="1;2;3;4;5;6;7;8;9;10" />
      <PropertyType name="Tiredness" type="System.Int32"
isPickList="True" pickList="1;2;3;4;5;6;7;8;9;10" />
      <PropertyType name="Nausea" type="System.Int32" isPickList="True"
pickList="1;2;3;4;5;6;7;8;9;10" />
      <PropertyType name="Depression" type="System.Int32"
isPickList="True" pickList="1;2;3;4;5;6;7;8;9;10" />
      <PropertyType name="Anxiety" type="System.Int32"
isPickList="True" pickList="1;2;3;4;5;6;7;8;9;10" />
      <PropertyType name="Drowsiness" type="System.Int32"
isPickList="True" pickList="1;2;3;4;5;6;7;8;9;10" />
      <PropertyType name="Appetite" type="System.Int32"
isPickList="True" pickList="1;2;3;4;5;6;7;8;9;10" />
      <PropertyType name="Well Being" type="System.Int32"
isPickList="True" pickList="1;2;3;4;5;6;7;8;9;10" />
      <PropertyType name="Shortness of Breath" type="System.Int32"
isPickList="True" pickList="1;2;3;4;5;6;7;8;9;10" />
      <PropertyType name="Other Problem Description"
type="System.String" isMultiLine="True"/>
      <PropertyType name="Other Problem" type="System.Int32"
isPickList="True" pickList="1;2;3;4;5;6;7;8;9;10" />
</EventType>
```

Event model definitions always contain property definition for "*WorkflowInstanceID*", "*WorkflowActivityName*", "*DateCreated*" and "*DateSubmitted*". These properties are used to capture information about the id of the workflow instance that is used to capture the event data, the name of the workflow activity that is used to capture the event data, the date the event data is created and submitted. Event data can also make a reference to an entity data. In the above event model definition, the "*Patient ID*" property is used for making a reference to a "*Patient*" entity.

3.4.5 Alerts

An Alert model definition holds information about the name of the alert ("*name*" attribute), the event entity type name ("*eventTypeName*" attribute) that will be checked for an alert trigger condition and the alert trigger condition ("*triggerCondition*" attribute) for the event that is related to the alert. This alert model is mainly used by the monitoring service of to check if events match an alert condition. The following is an example alert model definition that is used to create severe pain alerts for "*ESAS*" assessments with a pain score level of eight or more on a scale of ten. Note that the alert trigger condition is described in terms of properties of the event that is related to the alert (*triggerCondition="ESAS.[Pain]>=8"*).

```
<AlertNotifications>
   <AlertNotification name="Severe Pain Alert" eventTypeName="ESAS"
triggerCondition="ESAS.[Pain]>=8" triggerWorkflow="" roles="">
   </AlertNotification>
</AlertNotifications>
```

3.4.6 Performance Indicators

Performance indicator model holds information about the name of the performance indicator ("*name*" attribute), the formula that will be used to calculate its value ("*value*" attribute), the criteria that will be used to filter the set of events that are used to calculate the performance indicator value ("*filter*" attribute). This formula defined in the "*value*" attribute is described in terms of the properties of the events that are related to the performance indicator ("*eventTypeNames*" attribute) and the query capabilities of the runtime environment's persistence service. For example, to calculate a performance indicator that measures the average duration between two event data submissions in hours, a formula similar to "*avg(hour(Event_1.DateSubmitted)* –

hour(Event_2.DateSubmitted)) can be used. The "*eventTypeNames*" attribute is also used to identify the event types that are related to the performance indicator. Performance indicator model also defines color coded range of values for the performance indicator. The ranges that are color coded as "*green*" or "*yellow*" are assumed to be acceptable ranges. However, if the performance indicator value falls in the range of values that are coded as "*orange*" or "*red*" an alert notification will be created for the performance indicator. Performance indicators are checked every time an event is submitted to the monitoring service of the runtime environment.

```
  <PerformanceIndicators>
    <PerformanceIndicator name="Daily Average Severe Pain Management
Response In Hours" eventTypeNames="ESAS; ESAS_Consultation"
value="avg(hour(ESAS_Consultation.DateSubmitted) -
hour(ESAS.DateSubmitted))" filter="ESAS.DateSubmitted between
current_date() and current_date() + 1" >
      <IndicatorRange min="0" max="2" status="green"/>
      <IndicatorRange min="2" max="4" status="yellow"/>
      <IndicatorRange min="4" max="6" status="orange"/>
      <IndicatorRange min="6" max="" status="red"/>
    </PerformanceIndicator>
  </PerformanceIndicators>
```

3.5. Application Runtime Environment

The application runtime environment is the runtime environment in which managed process applications execute based on a model definition. The application runtime has the following elements: application engine, services, event based dimensional database and model manager (see Figure 6).

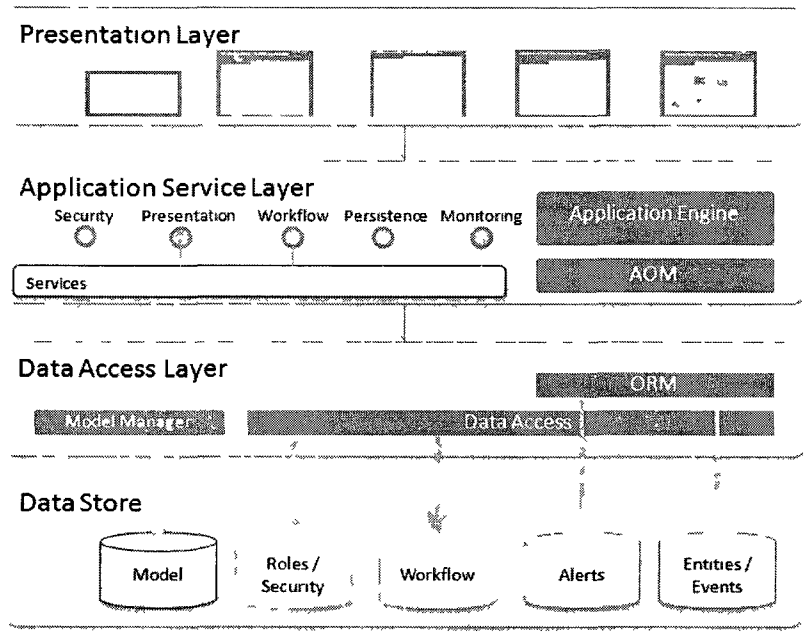


Figure 6 Managed Process Application Run Time Environment

3.5.1 Application Engine

The application engine is part of the application runtime environment that manages the in memory AOM data for workflows, roles, entities, events and alerts.

The application engine also coordinates calls between services including the presentation and persistence services. For example, when an event is submitted using the workflow service, the application engine makes sure that this event persisted using the persistence server and sent for monitoring via the monitoring service. Services

Workflow Service

The workflow service is responsible for creating a new workflow instance, submitting event data for a workflow activity, retrieving information about the list of active workflow instances and retrieving information about a particular workflow instance. The workflow service is the interface for interacting with the workflow engine.

The workflow engine manages the transition between workflow activities of managed processes.

As shown in Figure 7, The "*StartWorkflow*" operation invokes the workflow engine to create a new workflow instance. The workflow engine executes the workflow activities step by step as defined in the workflow model. When a "*CollectEventDataActivity*" is executed the workflow instance is set to be idle and waits for an event data to be submitted. When the workflow service receives a call to "*SubmitEvent*" the particular workflow instance identified by its "*workflowInstanceID*" is resumed and its waiting "*CollectEventDataActivity*" is notified to receive an event and the workflow instance continues processing. Other operations of the workflow service also interact with the workflow engine to collect "*WorkflowInstanceDescription*" for workflow instances. The "*WorkflowInstanceDescription*" holds information such as the workflow instance ID, the workflow name and status.

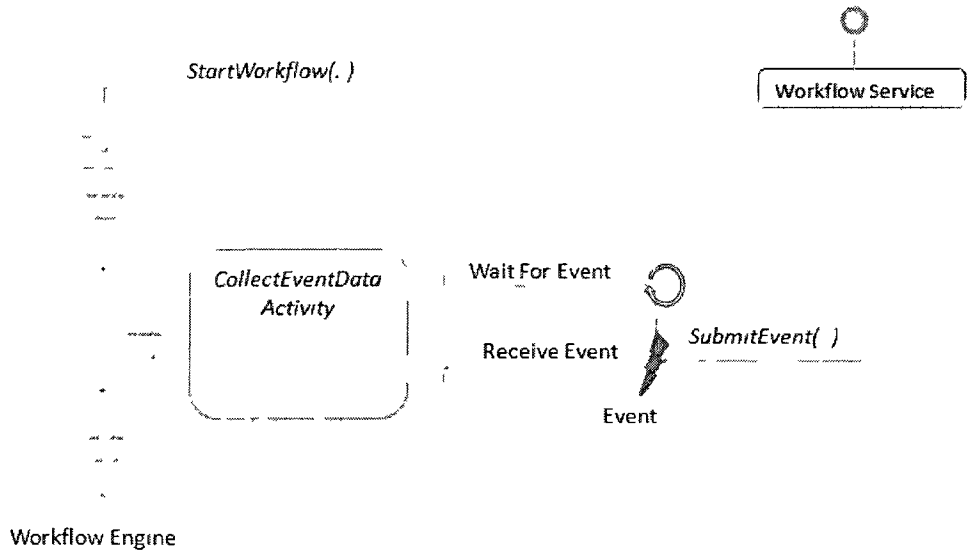


Figure 7 Interaction between the workflow service and workflow engine

Key operations of the workflow service are:

1. *WorkflowInstanceDescription StartWorkflow(String workflowName)*
2. *WorkflowInstanceDescription GetWorkflow(Guid workflowInstanceID)*
3. *SubmitEvent(Guid workflowInstanceID, Event eventData)*
4. *List<WorkflowInstanceDescription> GetAllWorkflows()*

Monitoring Service

The monitoring service is used for inspecting event data to create an alert for the event, if the event matches an alert trigger condition. The monitoring service is also responsible for creating alert notification when performance indicator value has fallen in a "*red*" or "*orange*" indicator value range and the alert will be available to all roles that are defined in the alert notification or performance indicator model definition. When an event data is submitted using the workflow service, the workflow service sends the submitted event to the monitoring and persistence services. The monitoring service has a single operation, called "*MonitorEvent(Event eventData)*", that is used for submitting events to be checked for alerts and performance indicator value ranges.

Persistence Service

Events and entities need to be retrieved and persisted during an execution of a managed process. The persistence service is responsible for retrieving and storing all types of events and entities. The persistence service uses event and entity model definition to perform persistence related actions. The persistence service uses an ORM, to persist and retrieve entities in a SQL relational database. At run time, the persistence service, transforms event and entity model definitions into ORM mapping configurations

and uses them to process the transformation of entity model and data into an SQL statement which will update or insert or retrieve information from the relational database. The persistence service also makes use of a flexible event based dimensional database schema that can accommodate changes in an entity model definition to add or remove new properties in the event or entity.

The persistence service has operations for retrieving and saving/updating events and entities. It also has operations for retrieving events for a workflow or an activity. It also provides search operations where example events and entities with pre-populated property values that are provided in the call to the search operation are used to create the search criteria. The persistence service provides a "*QueryScalar*" operation that is used to retrieve scalar valued queries for a given a query string. This operation is used to retrieve performance indicator values given a query string that resolves into a single numerical value for the performance indicator.

Key operations of the persistence service are:

1. *Save(Entity entity)*
2. *Save(Event eventData)*
3. *Entity GetEntity(int entityID)*
4. *Event GetEvent(int eventID)*
5. *List<Event> GetWorkflowEvents(Guid workflowInstanceID)*
6. *Event GetWorkflowActivityEvent(Guid workflowInstanceID, workflowActivityName)*
7. *List<Entity> SearchByExample(Entity entity)*

8. *List<Event> SearchByExample(Event eventData)*
9. *int QueryScalar(String queryString)*
10. *List<Alert> GetAlerts()*

Presentation Service

The presentation service is responsible for generating html markup that will be used by the application engine to render web forms that users interact with. The presentation service generates html markup for event and entity input, for viewing event and entity data. The presentation service also provides a short description for event and entity data that is used for the purpose of presenting quick preview of the content of an event or entity data. The presentation service also updates an entity or event given a collection of input web form elements into which a user has entered data.

Key operations of the presentation service are:

1. *String GetEntityInputMarkup(Entity entity)*
2. *String GetEntityViewMarkup(Entity entity)*
3. *String GetEntityDescription(Entity entity)*
4. *String UpdateEntityFromInput(Entity entity, FormCollection inputFormCollection)*
5. *String GetEventInputMarkup(Event currentEvent)*
6. *String GetEventViewMarkup(Entity currentEvent)*
7. *String GetEventDescription(Entity currentEvent)*
8. *String UpdateEventFromInput(Entity currentEvent, FormCollection inputFormCollection)*

Security Service

The security service, among other things, is used for validating user credentials, issuing an authentication token, validating an authentication token and checking user's role membership. The "*Authenticate*" operation returns an "*authenticationToken*" upon successfully validating and authenticating the "*username*" and "*password*" that is provided. The "*authenticationToken*" is sent as part of the SOAP header when invoking web service operations that require the caller user to be authenticated.

Key operations of the security service are:

1. *String Authenticate(String username, String password)*
2. *Boolean IsValidAuthenticationToken(String authenticationToken)*
3. *Boolean IsUserInRole(String role)*

3.5.2 Event Based Dimensional Database

Another element of the application runtime environment is the event based dimensional database. Events and entities are stored in separate tables that are organized using a "*star*" schema. Event data can make a reference to an entity data but not vice versa (see Figure 8). Both the event and entity tables are capable of handling new event and entity models without the need to modify the event or entity table schema. The approach we have taken to handle this flexibility is to have separate tables for storing events and entities and design both the event and entity tables to have several fields for each possible data type that they can potentially handle. That is, several fields for each

model data type (short text, medium text, long text, integer, and date time) are created on both event and entity table.

At run time, the event and entity model definition are inspected by the ORM to map properties of the entity to corresponding fields of the event or entity table. The persistence service uses this mapping information to appropriately store and retrieve event and entity information from the dimensional database using the ORM.

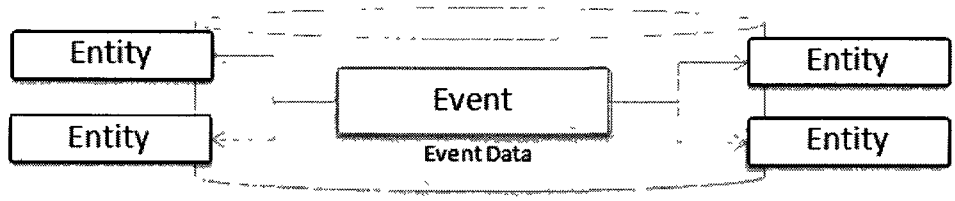


Figure 8 Event, Entities and their relationship.

3.6. Engineering

MPAF uses business level (higher level) models that simplify the development of managed process applications. Application development using MPAF does not require any coding and instead the business level model elements are configured to create an application. This can potentially enable domain experts that do not have technical software engineering expertise to create and define model definitions for managed process. Because the runtime environment directly executes a model definition into a running application instance without the need for recompilation, domain experts and application developers can make a model change and have it reflected in the application immediately.

Model definitions can be reused to create similar applications that are run using the same runtime environment. This facilitates reuse of both the runtime environment and

the model definition when creating similar type of managed process applications. Editing model definitions for MPAF does not require specialized modeling tools and can be achieved using simple text based XML editors. Although defining the model definitions for a managed process are simple, creating and maintaining the runtime environment requires software engineering and application development expertise. Maintenance of the runtime environment must be done carefully to avoid any alterations of the runtime environment that will negatively affect existing model definitions.

3.6.1 Model Driven Application Development Methodology

Application development using MPAF starts with identifying the performance indicators that will determine the type of workflows, entities, events and alerts that must be used to collect data and monitor events that are used measure how well an organization is performing to meet its current organizational goals.

The following are steps that are taken when implementing applications using MPAF.

1. Identify performance indicators based on an organization's current strategies and objectives.
2. Identify process workflows.
3. Identify the events that need to be collected based on the performance indicators identified in step 1 and the process workflows identified in step 2.

   For each event also identify

a. Entities (such as lookup data) that must be pre-populated.

b. Identify the conditions for generating alerts.

4. Identify and assign roles and permissions for the processes identified in step 3.
5. Create an application model based on the elements that are identified in steps 1-4.

Chapter 4. Case Study

In order to develop, refine and evaluate our framework, we applied it to a palliative care scenario in which a severe pain patient monitoring application was built and it was used for evaluation and comparison with existing approaches. We used a palliative care scenario to evaluate our framework because palliative care patient monitoring for severe pain management is a good example of a managed process application that has well defined workflow, events, entities, alerts and performance indicators where the patient submits pain assessments, nurse provides consultation to assessments and severe pain alerts, case worker monitors the care delivery process and team/unit administrator sets up performance indicators to measure overall quality of the care delivery process.

In section 4.1 we will provide a definition for palliative care and point out some of the issues in palliative care in the context of patient care monitoring for severe pain management. In section 4.2 we will provide descriptions of the palliative care scenario. In approaching the scenario we looked at three different implementations in order to compare our framework with both current practices (PAL-IS, implementation based on two-layered architecture) and a UML model based approach using AndroMDA.

4.1. Palliative Care

In palliative healthcare, the patient care is provided by a team of healthcare professionals from different disciplines and different organizations. The care delivery team can include doctors, nurses, case managers, social workers and other healthcare

professionals. To provide quality care, communication and coordination among these healthcare professionals is necessary.

As outlined in the WHO definition of palliative care (see section 2.6.2), pain management is a key component of palliative care. Pain management is used to provide relief to a patient from pain and other distressing symptoms. In the process of pain management, pain descriptions will be collected periodically from the patient and these descriptions will be assessed to decide the course of action that needs to be taken to provide a relief to the patient. Pain is subjective and a pain description that is provided by the patient who is experiencing the pain is usually used for assessment by a healthcare professional. Pain descriptions can also be categorized for management and identifying severity. For example, severe pain is described as a pain level of 8 or more points on a 10 point rating scale. It is reported that severe pain that 20% to 40% of severe pain cases are not properly managed and as a result there is a need to new approaches for severe pain management[Kuziemsky08]. Our palliative care scenario revolves around severe pain management by capturing periodical pain assessments from patients, triggering alert notification when a pain is severe and generating related performance indicators reports.

4.2. Patient Care Monitoring Application

As shown in Figure 9, in our severe pain management patient monitoring application, there are four participants: the patient, nurse, case manager and the team/unit administrator. The patient submits pain assessments few times daily. Pain assessment can be submitted either using a web form that the patient has access to or the patient can call the palliative care nurse and communicate the pain assessment over the phone. After the

pain assessment is recorded, the nurse will provide consultation over the pain assessment. Based on the nurse's consultation some action may be taken such as a recommendation for reviewing the patient's prescriptions can be sent to the patient's doctor. Upon receiving the nurse's recommendation, the doctor may issue a new prescription, to alleviate the patient's pain to ultimately improve the patient's quality life. The case manger will also follow up with the patient regularly to ensure that the patient is receiving quality care. The team/unit manager's responsibility is to make sure that standard of quality of care is adequate. [Liu08]

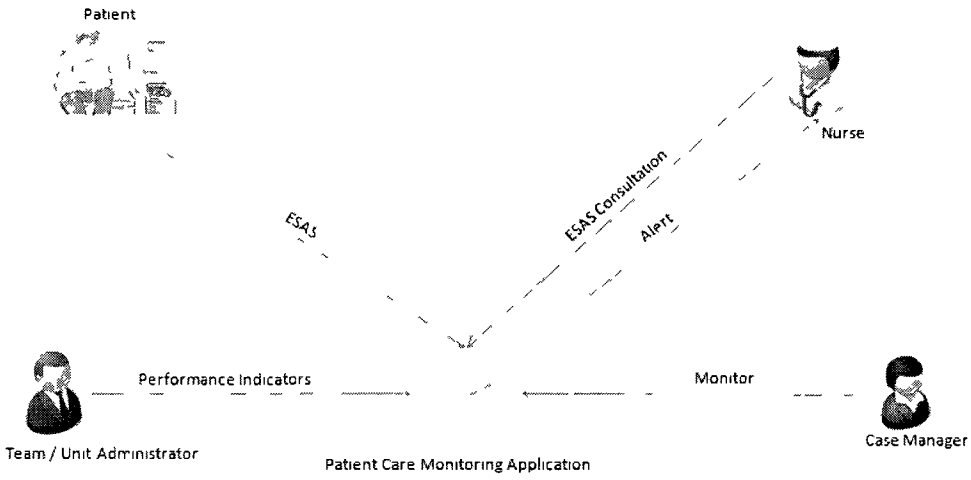


Figure 9 Patient Care Monitoring Application

Palliative care units can set their standard for handling severe pain management. These standards are usually based on healthcare guidelines. For example, a severe pain assessment (8,9,10 on a 10 point scale) must be handled as soon as possible and palliative care units can set the threshold for the maximum number of hours a severe pain assessment can stay without being consulted to measure the timely delivery of palliative healthcare services.[Liu08]

To adhere to such standards, nurses should be notified when a severe pain assessment is submitted by a patient so that they can consult the patient in a timely manner. Team administrators should also set performance indicator to measure how well the pain management processes adheres to the targeted performance standards. Team administrator can use the average number of hours it takes for a severe pain assessment to be consulted to measure the performance of the team in handling patient's severe pain assessment submissions. Performance indicators can be measured, daily, weekly and monthly.

Based on the above description of the palliative care patient monitoring application, the key requirements are:

1. Allow patients and/or nurses to submit pain assessment.
2. If a pain assessment has a pain score that is greater than or equal to eight on a scale of zero to ten then notify a nurse.
3. Allow nurses to record their consultation for a pain assessment.
4. Allow team/unit administrator to set performance indicators and view reports.

4.3. PAL-IS

PAL-IS is an existing patient care monitoring application, built for the palliative care consulting team in Ottawa. PAL-IS has been developed using the traditional two-layered application architecture using the Microsoft ASP.NET framework. A layer refers to a separate aspect of an application that focuses on a particular concern of an application (see Figure 10). For instance, the presentation layer of an application is

separate from its data access layer that is used to access databases. The higher layers use services from lower layers but not vice versa.

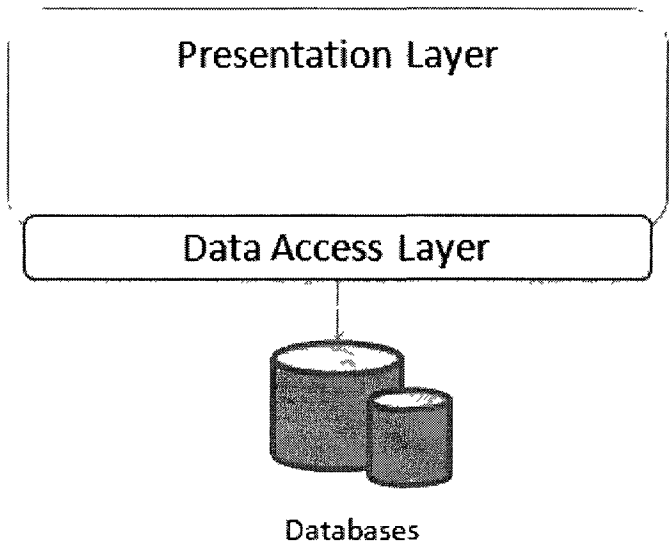


Figure 10 Two-layered Application Architecture

In a two-layered architecture, application logic and data access are embedded within the presentation layer. It is very common to use a two layered architecture when programming in ASP.NET mainly because of the built-in functions such as drag-and-drop and user interface control data binding that facilitates rapid application development. Common user interface functionalities such as sorting, paging, editing, updating, adding, and deleting are provided in ASP.NET controls. It is also easier to get an overview of the overall source code structure because all aspects of the application including its presentation, business logic and data access are located in the same area.

The two-layered architecture has some disadvantages when it comes to handling frequent changes. Frequent change in business rules or changes to the database schema

such as the addition of a new field or table, or even the type of data store requires going through all the application code to make sure that the new required changes are handled appropriately. There is often code duplication and it is difficult to isolate a single component for modification as the business rules and data access logic is spread all over the presentation layer elements. From an end user's perspective it does not make a difference that a two-layered architecture was used instead of a multi-layer one. However, from a technical point of view, the two-layered architecture is not the ideal choice when developing enterprise applications where frequent changes and continuous maintenance are inevitable.

4.3.1 Models

PAL-IS is developed based on code-centric application development using the Microsoft Visual Studio .NET integrated development environment. PAL-IS does not have any declarative external application models. Database schema and code-level object model are implicit models in the application code and database that are specific to the Microsoft .NET framework and SQL Server database.

4.3.2 Application Runtime Environment

Business Process Workflow

The current implementation of PAL-IS does not use a workflow engine for implementing business process workflows. Instead, business processes are simply implemented as hard-coded web form navigational flow where users of PAL-IS can navigate from one page to another to fill out forms with no restriction of the steps that can be followed during navigation.

SOA

PAL-IS does not make use of SOA. One of the advantages of SOA is the possibility of integration with external systems by exposing functionality in the form of a service. PAL-IS is not integrated with any external application and does not expose services nor does it consume external services. Users of the system have to manually re-enter the data on an external application to have information from PAL-IS web portal available in the external application. If integration with an external system is desired, new web services can be developed or external service invocations can be coded. This requires the application code to incorporate these new changes and then it has to be built and re-deployed.

Another possible method of integration, without the need to modify the application code, is to use the underlying database as the point of integration. To use the database layer for integration, external applications have to be granted access to the database to read and write data into the tables that are used for integration. These tables can have "triggers" that are activated when a record is inserted, updated or deleted. These triggers can be implemented to perform any necessary database action to manage records that are used by the PAL-IS web portal.

This method of integration has some disadvantages. The PAL-IS web portal database has to be made directly accessible to external applications, exposing the application to potential security and privacy risks. It will be difficult to make schema changes as there is no abstraction layer between the database and consumers of the database. If the schema of the table that is used for integration is changed, other external applications have to be updated accordingly to correctly use the new schema.

Also, allowing direct access to database tables does not provide a clear specification of an API contract for communication between the external application and the PAL-IS portal. Although some level of restriction can be imposed using database security constraints and application logic in relational database triggers, external applications that have access to these tables can update or insert records arbitrarily as there is no direct way of enforcing a contract that consumers of the database have to adhere to when modifying records in these tables.

Configuration

PAL-IS does not have configuration elements that can be used to change the behaviour of the application. Any change has to be done using application development tools to modify source code and database schema.

Alerting and Monitoring

The current implementation of PAL-IS is a direct conversion of the paper based processing into web based processing where paper based data is simply converted into electronic format. Alerts are manually created and the nurses use the alert form to create new alerts after they have manually identified a situation that should be classified as an alert condition. Although storing records in electronic format makes it easier for further processing and reporting, users have to still rely on their personal judgement and what they remember as an alert condition that is outlined in medical guidelines

In the current PAL-IS web portal implementation there is no automated event monitoring and processing capability. Users of the system, particularly, the nurses manually determine what event needs attention and decide what action must be taken next. Manual

event processing is prone to error. It can also lead to delayed processing of events as humans can only process limited amount of events at time while event processing systems can process several thousand events per second. In addition, complex event processing is difficult to process manually and when processed manually, complex events are identified only after an adverse event has occurred and may require one to go through a pile of paper work to retrace steps of activities that lead to a complex event.

Data Model and Reporting

PAL-IS uses a manually crafted database model. The database is directly accessed from PAL-IS using hard-coded data access application logic. PAL-IS does not offer a direct way for processing performance measurement. Currently, an audit of months of paper work is performed periodically to measure performance and check the quality of the care delivery process. Some ad-hoc reports can also generated to provide basic information and performance measurement reports.

4.3.3 Engineering

The two layered architecture is very common in ASP.NET application development using the Microsoft Visual Studio IDE. No new framework, architecture or method of code organization has to be learnt when using two layered architecture and this makes the learning curve relatively low. Although ASP.NET includes some advanced technologies, it is not mandatory to use these technologies and all aspects of application logic can be embedded within the ASP.NET presentation layer. In addition, there are many features such as visual web form designer, drag-and-drop, web form user interface control customization, control data-binding, standard navigation web form controls, standard user

membership management web form controls, user membership services, internationalization services, personalization services and user interface theme support that enable rapid application development using ASP.NET.

Maintenance in a two-layered architecture can become cumbersome because application logic is placed in the front end of the application spread across multiple ASP.NET web forms. It is also difficult to isolate a single component for maintenance increasing the potential risk of degrading the quality of an application. For example, if an application data sources changes from MS SQL Server database to an XML database or vice versa, it is necessary to update every ASP.NET web form that accesses a data source to be able to load data into the application. The same difficulty will occur if a database field name is changed because all involved ASP.NET web form pages have to be altered. In addition, after any application source code change has been applied, the application has to go through the usual steps of recompilation, functional testing and application deployment stages. It is also difficult to reuse application code in PAL-IS. To create an application with similar functionality to PAL-IS, the whole PAL-IS application code and database schema has to be copied over and modified to apply customizations for the new application.

4.4. AndroMDA

A possible improvement over the hard-coded two-tier approach of PAL-IS, is to use a classic, generic model-driven framework. We took one of the popular frameworks, AndroMDA, that use MDA approach and re-implemented the severe pain management patient care monitoring application.

AndroMDA is an extensible MDA based framework that generates application source code from annotated UML models. AndroMDA can generate application code for any type of platform and architecture. Two of the major platforms that are supported out of the box by AndroMDA are Java and Microsoft .NET. The default architecture that is generated by AndroMDA for the Microsoft .NET platform uses multi-layered approach and SOA [AndroMDA10]. For our scenario comparison, we will be using an AndroMDA generated application for the .NET platform as shown in Figure 11[AndroMDA10].

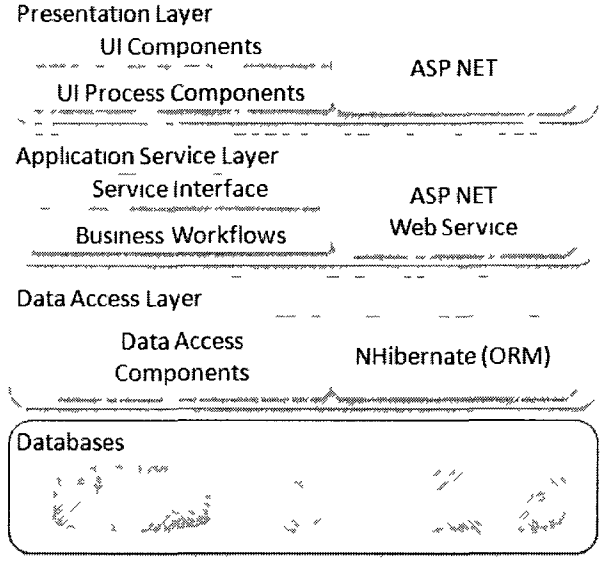


Figure 11 Architecture of an AndroMDA generated application for the Microsoft .NET framework.

The presentation layer consists of components such as web pages that are needed to interact with the end user. Currently, there is an experimental ASP.NET cartridge that can be used to generate ASP.NET pages (ASPX)[AndroMDA10].

The business layer encapsulates the business logic of an application. The business layer generated by AndroMDA has service interfaces that are used to hide the internal complexity of an application's business logic. These services can be directly accessed

internally in the generated application source code. Web services can also be implemented, on the top of the service interfaces, so that the services can be externalized and run independent of the main application.

AndroMDA uses the NHibernate object-relational mapping tool for its data access related functionalities. Data access objects (DAOs) are generated from AndroMDA entity classes that are defined in the UML model. These data access objects use the NHibernate API to store objects as database records and to convert database records to objects. NHibernate supports many types of data store such as Microsoft SQL Server, MySql, Microsoft Access and several others.

Both AndroMDA entities (DAOs) and Data transfer objects (DTOs) can be used to pass data between the data access layer and the business layer. However, only DTOs are used to transfer data between the business layer and the presentation layer. Although this approach requires more classes (DTOs) to be generated or created, it helps to keep all business logic within the business layer and no AndroMDA entities that contain business logic related information will be exposed to the presentation layer. It also prevents the use of AndroMDA entity classes in the presentation layer to handle a business logic which helps to avoid the splitting of the business logic between the presentation layer and business layer. Containing all business logic in the business layer also makes the business layer ready to be used by another type of presentation layer or an external application. It also makes the data transfer between the business layer and presentation layer lightweight because it avoids the serialization of the whole AndroMDA entity when only partial information such as entity identifier is required by the presentation layer [AndroMDA10].

The following steps were used to implement the palliative care severe pain management patient care monitoring application using AndroMDA.

1. **Select and/or define the architecture for the application**: SOA is one of the default application architectures supported by AndroMDA and it is the architecture that we have selected for our scenario. If we had selected an architecture that is not supported out of the box by AndroMDA, it would have required us to either customize or create new cartridges that are used to generate application code to match our architecture.
2. **Create the platform independent model (PIM)**: PIM models are defined using UML and are stored in an XMI format which will later be processed by AndroMDA. For our scenario, we used a community edition of MagicDraw. MagicDraw is a CASE tool that is recommended for use with AndroMDA and is used for creating UML models. Using MagicDraw we designed the UML use case, activity and class diagrams (see Figure 12, Figure 13 and Figure 14). As seen in Figure 12, we have defined class diagrams for "*Patient*", "*Prescription*", "*ESAS*", "*ESASConsultation*", "*AlertNotification*" and "*PerformanceReport*". We have also defined class diagrams for "*PatientService*", "*NotificationService*" and "*ReportService*". These classes are used to retrieve, store and apply business logic on the classes that hold information. In Figure 12 we have excluded other classes such as security related classes that are not directly involved in the realization of the palliative care severe pain management application. However, for the sake of completeness, we have included the "*User*" class that belongs to the security component of the application.
3. **Apply appropriate UML stereotype decoration**: For AndroMDA to be able to determine which cartridge to use for a particular class, the classes in the class diagram

have to be decorated with a particular stereotype. Particularly we used three stereotypes namely "<<*Service*>>", "<<*Entity*>>" and "<<*Enumeration*>>" (see Figure 12). AndroMDA generates code that implement Data Access Objects (DAO) and Data Transfer Objects (DTO) design patterns. A Data Access Objects (DAO) pattern is used for communication with the underlying database and Data Transfer Objects (DTO) are used to pass data across application layers. A DTO class can be defined using the "<<*ValueObject*>>" stereotype. A class with the stereotype "<<*Service*>>" becomes a service class that can easily be exposed as a web service using ASP.NET XML Web services or Windows Communication Foundation. A class that is decorated with the "<<*Entity*>>" stereotype is generated as a set of classes that includes an entity base class, DAO base class and implementation classes.

4. **Generate the code for a particular platform**: AndroMDA generates two separate categories of source codes. The first category is the section of code that will be regenerated from scratch every time we tell AndroMDA to generate code. Source code that is generated in this category should not be modified by the application developer because it will be overwritten the next time AndroMDA is commanded to generate code. The other category is generated only once and it is safe for modification by the application developer.
5. **Customize the generated code to implement business logic**: AndroMDA only generates the skeleton source code with implementation for well known and repetitive tasks, such as DAOs, DTOs and service interfaces. For ASP.NET, there is an experimental AndroMDA cartridge that is used to generate ASP.NET web forms (ASPX) while for JAVA it generates STRUTS based user interface. Once the code is

generated, the application developer can customize the generated application to implement and hardcode business logic and workflows.

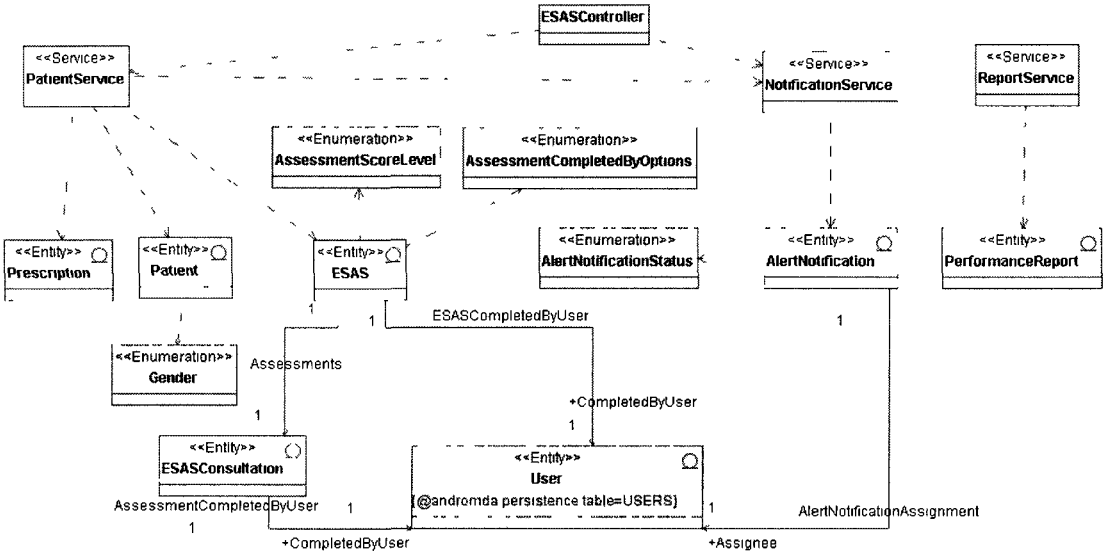


Figure 12 UML Class Diagram the Severe Pain Management Patient Monitoring Application

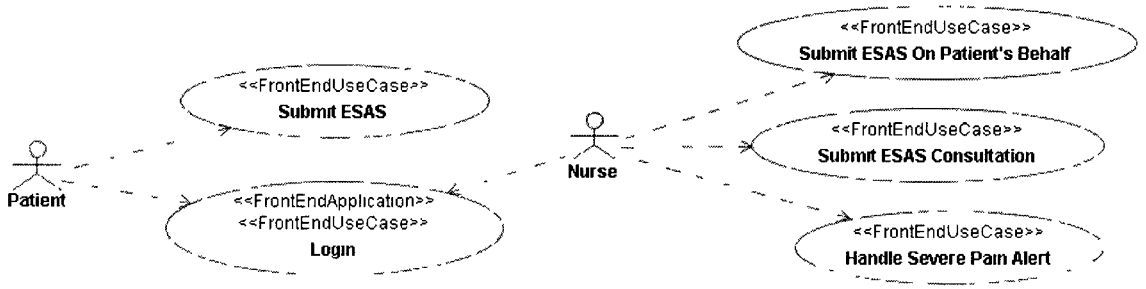


Figure 13 Partial Use Case Diagram for the Palliative Care Severe Pain Management Patient Monitoring Application.

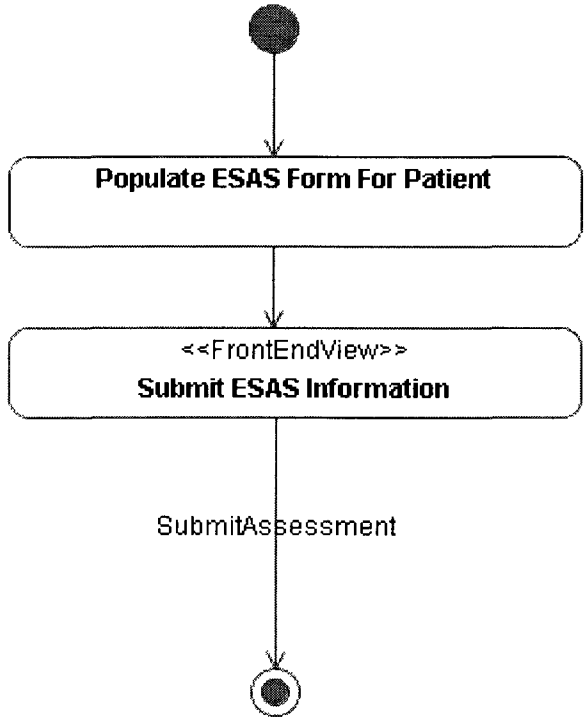


Figure 14 UML Activity Diagram for the "Submit ESAS" Use Case

The current version of AndroMDA (version 3.3) provides a stable and extensible code generator framework for rapid application development. AndroMDA generates code directly from a PIM that is decorated with stereotypes and tagged values. The PSM is not available for modification before code generation. The PSM model is generated in memory during code generation and is used by AndroMDA and its cartridges. However, stereotype annotations can be used to provide hint to AndroMDA on how to generate the desired PSM model during code generation.

Using AndroMDA enables application developers to capture their business logic and navigational flow in a UML model to later generate a programming language (such as C#) source code. After any change to the model the application code has to be regenerated and compiled again.

4.4.1 Models

AndroMDA enables model driven application development by processing and generating code from a UML based model that is stored in an XMI format. AndroMDA uses use case and diagram models to generate presentation layer code. Class diagrams are used for modeling services, web services, value objects (DAOs), entities (DAOs) and their corresponding NHibernate mapping for data access. Managed application models (workflows, roles, entities, events, alerts and performance indicators) have to be translated into their corresponding UML model. For example, workflows can be modeled as activity diagrams, entities can be modeled as a class diagram that has the "*<<Entity>>*" stereotype. Roles, events alerts and performance indicators have to be modeled using UML diagrams and specialized logic for these models has to be hard-coded.

Use Case and Activity Diagrams

The presentation model in AndroMDA is represented by UML use cases and activity diagrams. To model the application front-end use cases must be defined for each unique set of functionality that the application provides. For example, "*Login*", "*Submit ESAS*" and "*Submit ESAS Consultation*" are some of the use cases for our severe pain management patient monitoring application as seen in Figure 13. To provide additional hint to AndroMDA, use cases can be decorated with the "*<<FrontEndUseCase>>*" and "*<<FrontEndApplication>>*" stereotypes. There can only be one use case that has the "*<<FrontEndApplication>>*" and this use case will be considered as the application's entry point. Use cases that require user interaction are labelled with the "*<<FrontEndUseCase>>*".

Activity diagrams are used to model application workflow behaviour (see Figure 14). An activity diagram must be linked with one use case. It uses the states and transitions between states in the course of a sequence of steps taken by the user in the context of acting on a single use case. The initial state denotes the starting point of the use case the activity diagram is linked with. The action states are used to represent both client and server side states of an ASP.NET web form. Client side action states are tagged with the "<<*FrontEndView*>>" stereotype and will result in the generation of an ASP.NET ASPX page with buttons, hyperlinks, menu, breadcrumbs and other user interface elements. Server side states are represented with an outgoing transition as shown in Figure 14. Transitions represent the application's processing logic and will result in the generation of an ASP.NET code-behind with form fields derived from the parameters defined on the transitions.

Class Diagrams

Managed process entity models can be mapped to class diagrams that are tagged with the "<<*Entity*>>" stereotype. NHibernate mapping XML file is generated for class diagram that has the "<<*Entity*>>" stereotype. This mapping describes the table and field mapping of an entity to relational database tables. The NHibernate package contains the SchemaExport (hbm2dll) tool that can be used to generate database schema for a particular database technology from the mapping XML file. The generated schema includes tables, primary and foreign key constraints. Services are represented using class diagram models that are tagged with the "<<*Service*>>" stereotype. To exposes services as a web service, service classes must also be tagged with the "<<*WebService*>>" stereotype.

AndroMDA does not have specific model for alerts and event monitoring. In our AndroMDA implementation, alerts are simply modeled as UML classes and the alert notification is a custom code that was implemented after the application code was generated using AndroMDA. This custom code was implemented at the service level inside the "*PatientService*" to check for the severity of the pain score level for the currently submitted ESAS data. AndroMDA does not have an explicit model for reports and performance indicators. Reports and performance indicators must be modeled just as any other use case, activity and class diagram to generate the corresponding web forms, services, entities that are required for the report. Ad-hoc reporting tools can also be used to generate reports by directly accessing the data store.

4.4.2 Application Runtime Environment

Business Process Workflow

AndroMDA does not provide a workflow model or workflow runtime engine out of the box. Workflows are modeled using activity diagrams that are used for generating source code generate navigational workflow. Once the navigational workflow is generated, it is hard coded and compiled. To modify the workflow, the activity diagram for the workflow has to be changed and the application has to be recompiled and deployed.

SOA

AndroMDA supports generating application code that is based on SOA. Web services can be exposed for external application consumption or web service invocation can be hard coded to access externally located services. Web services enable a vender neutral, platform and independent integration between multiple applications.

Configuration

AndroMDA supports configuration via its model definition. To change the behaviour of an application, its AndroMDA UML diagrams have to be modified to reflect the new behaviour and source code for the whole application will be generated.

Alerting and Monitoring

Alert notifications in the AndroMDA generated applications are hardcoded. Alert notifications can be modeled in activity diagrams as a decision branch where the decision condition is used to identify the action and if the condition is satisfied an alert notification is created otherwise the normal application logic will proceed (decision branches are not supported by the ASP.NET cartridge). Modeling alert conditions in activity diagrams allows alert notification processing logic to be generated by AndroMDA. However, this approach makes alert notification to be tied to application logic processing at the presentation layer and the same code has to be repeated if it is desired to detect alert notification on other presentation components or application layers. The other approach is to hardcode the processing logic to detect alert notification processing at the service layer. This enables identification of conditions that require alert notifications to be consistent and available to all layers that consume the service layer.

The AndroMDA generated application does not have an automated event monitoring and processing mechanism. In this application, events are merely entities managed by the application. Entity processing logic are either manually crafted or generated by using activity diagram models. Therefore, any simple event processing logic must be hardcoded in the application using standard "if-then-else" statements. Implementing complex event

processing logic using standard programming language constructs is inconvenient, difficult to maintain and inflexible to frequently changing complex event identification mechanisms.

Data Model & Reports

AndroMDA, out of the box, does not offer a direct way of modeling performance indicators. However, business activity monitoring and performance management tools can be integrated to directly access the database or consume operations that are available at the service layer. Because the AndroMDA generated application uses services, it is easier to realize processes as an orchestration of services. This make it relatively easier to react to performance management reports and refine and optimize processes than the PAL-IS application.

4.4.3 Engineering

AndroMDA has a steep learning curve. Configuring the modeling and code generation environment is time consuming and can become quite challenging. Although UML based visual modeling using MagicDraw appears to facilitate the application development effort, the application developer has to understand how the modeling tool works and how AndroMDA interprets the model. The application developer has to make sure to use the correct UML stereotype keeping in mind how it will be interpreted by AndroMDA during code generation. Particularly, modeling user interaction and interaction between ASP.NET pages using activity diagram was not intuitive. The code generated by AndroMDA is not directly deployable and has to be modified to include business logic and other application logic that cannot be generated from a model.

AndroMDA generates several classes and files even when a simple model that contains very few services, entities and use cases is used.

The maintenance effort in AndroMDA involves either a model change or a manual code change. Even a small change, such as a new field on an entity requires the whole application code to be generated and recompiled again. Model changes, such as removal of a field from an entity, can easily result in an application code that cannot be successfully compiled. Overall, maintenance of a simple model change and manual code change in the portion of the code that is allowed for modification by the application developer is manageable whereas maintenance of complex model change can easily become a daunting task and might require applying model changes in gradual steps instead of applying all changes once. UML Models can be copied over and modified to create similar types of application.

4.5. Model Driven Managed Process Application Framework

The last implementation of the palliative care severe pain management patient monitoring application is based on our proposed model driven managed process application framework. This approach uses interpretation of model elements at run time to build elements and components that makeup the application that is described by the model. When using this approach there is no code generation and instead a runtime environment is used to execute the models into a running application instance.

In our proposed approach there is no direct code centric application development. Only model changes are available to implement application specific behaviours and functionalities. This limits the scope of variation and configuration there can be between

different applications that use the same features available in the run time environment. Our approach uses a specific model that is targeted for creating managed process oriented applications that are used to collect data, monitor and report over the collected data.

To implement an application using the proposed framework we have to follow the steps as outlined in section 3.6.1.

1. Identify and model performance indicators based on the health organization's current strategies objectives.

   One of the goals of the palliative care is to improve severe pain management. The average number of hours taken for a severe pain assessment (ESAS) before providing a consultation by a nurse can be used to measure the performance of severe pain management in the palliative care patient monitoring application. We will call this performance indicator the average "severe pain management" (SPM) in hours.

   The daily average SPM performance indicator is as shown in Appendix A. The performance indicator's value will have the average hour it took for severe pain assessements to be looked after by a nurse. We also specified the range of values for the performance indicator that are acceptable to be marked as "green" and those values that indicate that the severe pain management needs improvement will be marked by "yellow", "orange" and "red" status colors. Performance indicators can be viewed by the team or unit adminstrators.

2. Identify process workflows.

We took a very simple two step workflow as an example. An ESAS form is filled out by a patient or by a nurse after receiving the assessment information over the phone from the patient. After an ESAS form is completed it is followed by a consultation which will be performed by a nurse. This is modeled with two consecutive "*CollectEventDataActivity*" activities that are used to capture event information for ESAS and ESAS consultation. Each "*CollectEventDataActivity*" is assigned a display name, the type of entity that will be captured by the activity and who can act on the activity. The workflow model for the patient monitoring application is as shown in Appendix A.

3. Identify the events that need to be collected and alert notifications that should be triggered based on the performance indicators identified in step 1 and the process workflows identified in step 2.

   Performance indicators and processes determine the type of information that an application needs to collect. In our patient monitoring application, the performance indicator and the workflow use the same event information. Both the performance indicator in step 1 and the workflow in step 2 use ESAS and ESAS consultation information. This information is modeled with two event entities "*ESAS*" and "*ESAS_Consultation*". The "*ESAS*" assessment values are represented with a "*pick list*" that allows values in the range of 1 to 10. The "*ESAS_Consultation*" entity has a property for registering consultation and recommendation information. Patient information is managed by the application without the need for a separate process to create patient information and for this reason it is modeled as an entity. The "*Patient*"

entity and "*ESAS*" and "*ESAS_Consulation*" events can be viewed by the "*CaseManager*" role. The event and entity model for the patient monitoring application are as shown in Appendix A.

Both "*ESAS*" and "*ESAS_Consultation*" events require a reference to patient entity and this reference is modeled by the "*Patient ID*" property which holds the internal numeric identifier of the entity. The type of entity referenced by the "*Patient ID*" property is indentified by the "*lookupName*" and it is the "*Patient*" entity in both "*ESAS*" and "*ESAS_Consultation*" events. In the "*ESAS*" event the "*Patient ID*" is determined from the currently logged in user if the current user has a "*Patient*" role. If the currently logged in user has a "*Nurse*" role, the "*Patient ID*" will be a lookup that the nurse user can select from existing list of patients. In the "*ESAS_Consultation*" event, the "*Patient*" entity reference is derived from the "*ESAS*" event data that was used to create the current "*ESAS_Consultation*". After event entities are defined, alert notifications for events can be defined. In our scenario, severe pain assessments are identified with a trigger condition for an "*ESAS*" entity with a "*Pain*" score level of eight or more (see the alert notification model in Appendix A).

4. Identify and assign roles and permissions for the processes identified in steps 1 to 3.

   From the above steps, the "*Nurse*", "*Patient*", "*TeamAdministrator*" and "*CaseManager*" roles are determined. These roles will be described in the model and are used for making sure that valid roles are used in the model. For example, in the application model shown in Appendix A, the "ESAS" submission workflow activity

has it's roles attribute set to "*Nurse;Patient*" that signifies that either a "*Nurse*" or "*Patient*" role can perform this activity. They are also used to check that these roles are already registered with the application's security module and are available for use by the administrator during user management.

Other information such as the application's title and description can also be updated in the application model. The title and description will be used by the application engine when rendering the application (see the "*ApplicationInfo*" section in the application model that is shown in Appendix A).

4.5.1 Models

Our proposed approach uses a very domain specific type of model to capture workflows, roles, entities, events, alert notifications and performance indicators. The model is represented using an XML format.

Workflow

The process workflow model is represented by a workflow that contains a sequence of activities that are used to collect event information. An activity model definition for capturing event data can be defined using the "*CollectEventDataActivity*" workflow activity (see Appendix A). In our implementation, we have defined two "*CollectEventDataActivity*" in our workflow for capturing "*ESAS*" and "*ESAS_Consultation*" events.

Entity

In our implementation we only have the "*Patient*" entity that holds patient related information. The "*Patient*" entity has several properties such as "*First Name*", "*Last Name*", "DOB", "*OHIP Number*" etc

Event

The "*ESAS*" and "*ESAS_Consulation*" are two events that we have defined in our model as shown in Appendix A. These Events, in addition to the event data, they also hold information about the workflow instance in which the event is created, the date it is created and submitted and the activity name that was used to capture the event data ("*WorkflowInstanceID*", "*WorkflowActivityName*", "*DateCreated*", "*DateSubmitted*" see Appendix A) .

Alert

In the severe pain management patient monitoring application, the alert notification is defined for the "*ESAS*" event as shown in the alert notification model definition in Appendix A). When an "*ESAS*" entity submitted, if it has a pain score that is greater than or equal to eight, an alert notification will be created. This alert trigger condition is defined in the alert model as by "*eventTypeName*" and "*triggerCondition*" attributes (*eventTypeName="ESAS" triggerCondition="ESAS.[Pain]>=8").*

Performance Indicator

The model definition for our implementation includes performance indicators. We have created a performance indicator for the "Severe Pain Management" (SPM) time in hours to measure how well the care deliver process for severe pain is being performed. The

model definition includes average SPM for the current day, week and month. The daily SPM performance indicator is modeled using the "*value*" and "*filter*" attributes as "*value="avg(hour(ESAS_Consultation.DateSubmitted) – hour(ESAS.DateSubmitted))" filter="ESAS.DateSubmitted between current_date() and current_date() + 1"*". The "*value*" attribute specifies the formula that is used to calculate the performance indicator value. The "*filter*" attribute is used to select the events that will be used in the performance indicator calculation and we have used the "*ESAS.DateSubmitted*" value to filter events that were submitted on the current date.

4.5.2 Application Runtime Environment

Business Process Workflow

The application runtime environment is implemented over the Microsoft .NET framework. The presentation service uses ASP.NET MVC framework to provide web forms that users can interact with. The workflow service uses the Microsoft Workflow Foundation WFMS to implement the workflow processes. The workflow model definition is read at runtime by the runtime environment is used to create workflow instances at runtime. The presentation service provides web forms for viewing active ongoing process workflows (tasks), viewing alert notifications, viewing and submitting event data for an active process workflow and viewing reports (see Figure 15).

Figure 15 Patient Monitoring Application

Users with a "*Nurse*" role use the "*Tasks*" web form to create new tasks, view existing tasks and view alert notifications. To submit an "*ESAS_Consultation*" event, the "*Nurse*" opens one of the ongoing workflows to be provided with a form for collecting data for the "*ESAS_Consultation*" event. "*Case Managers*" can look at the list of events that were created for a particular "*Patient*". Event data that is collected in the course of executing several "*ESAS*" workflows is used to generate performance indicator reports.

SOA

Our MPAF based implementation uses the pre-configured services that are provided with MPAF. These services are the presentation, workflow, security, monitoring and persistence services. External service invocation is enabled via the "*InvokeWebService*" custom workflow activity and the "*SubmitEvent*" and "*StartProcess*" workflow service

operations. The "*InvokeWebService*" custom workflow activity is used to make a call to an external web service and is not used in our implementation. This custom activity takes an entity or event as an input to compose a web service request XML using an XSL transformation. It also uses an XSL transformation to convert a web service response into an entity. Using this custom activity it is possible to perform exchange information between applications by just modifying the application model. However, to use this web service based communication, external applications need to expose web services. The "*SubmitEvent*" and "*StartProcess*" web service operations are used to acquire event data or start a new process from an external application. External applications can use these operations to participate in the workflow process and submit event data.

Configuration

Configuration in our implementation is provided through the application model. Workflows, roles, events, entities, alerts and performance indicators can be configured using the application model. Additional application related information such as the application title and description that are used by the runtime environment can also configured using the application model.

Alerting and Monitoring

When a patient submits an "*ESAS*" event data, it is inspected by the event monitoring service to check if the pain score satisfies the condition for classifying it as a severe pain. If the "*ESAS*" is classified as an assessment with a severe pain, an alert notification will be created and will be visible for all users that have the "*Nurse*" role as specified in the model definition. A nurse can open an alert notification and access the process workflow

or event that triggered the alert notification to provide timely "*ESAS_Consultation*" for the "*ESAS*" with a severe pain score.

We have a very simple event processing in our application in the form of an alert notification condition and process workflow rules. This event processing follows a simple format of "Event-Condition-Action" where a condition is checked for alert notification after an event is submitted and the next action is determined by whether the condition is met or not. Complex event processing is not directly possible. However, basic pattern recognition of events can be implemented by using custom workflow activities that are capable of looking at past event data to determine if there is a pattern match.

Data Model and Reports

Alert notifications are used to monitor events in the application. The application generates alert notifications when events that affect the overall performance measurement occur. In our particular case, "*ESAS*" submissions with severe pain score affect the overall performance measurement of how well severe pain management is being performed our process. Alert notifications and performance indicators are setup in the application model for "*ESAS*" submissions. The persistence service queries the data store to retrieve performance indicator values based on the model definition of the performance indicator and display the performance indicator values in a report as shown in Figure 16.

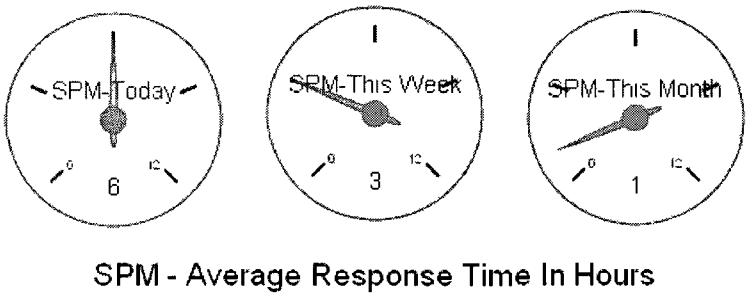


Figure 16 Performance Indicator Reports

The persistence service uses Microsoft SQL Server based flexible data model as shown in Figure 17. Events and entities are stored in separate database tables. Event entities can have a reference to entities but not vice versa. The type of data that is stored on a given record in an event or entity table is mainly determined by the properties that an event or entity has. Because event entities are captured while executing a workflow, additional information such as the id of the workflow instance that was used to create the event entity, the workflow activity that was used to capture the event entity data, the date the event data is created and the date the event data was submitted for processing ("*WorkflowInstanceID*", "*WorkflowActivityName*", "*DateCreated*" and "*DateSubmitted*") are stored with the event entity.

In our MPAF based implementation, we have created several columns for each data type; namely short text, medium text, long text, integer and date time for both event and event database tables. At run time, the event and entity model definition is inspected and NHibernate mapping file is generated for the event or entity. The persistence service uses this generated mapping file to store and retrieve entity information from the database using NHibernate. Each event or entity property is mapped to a particular column in the respective database table based on its data type (see Figure 17).

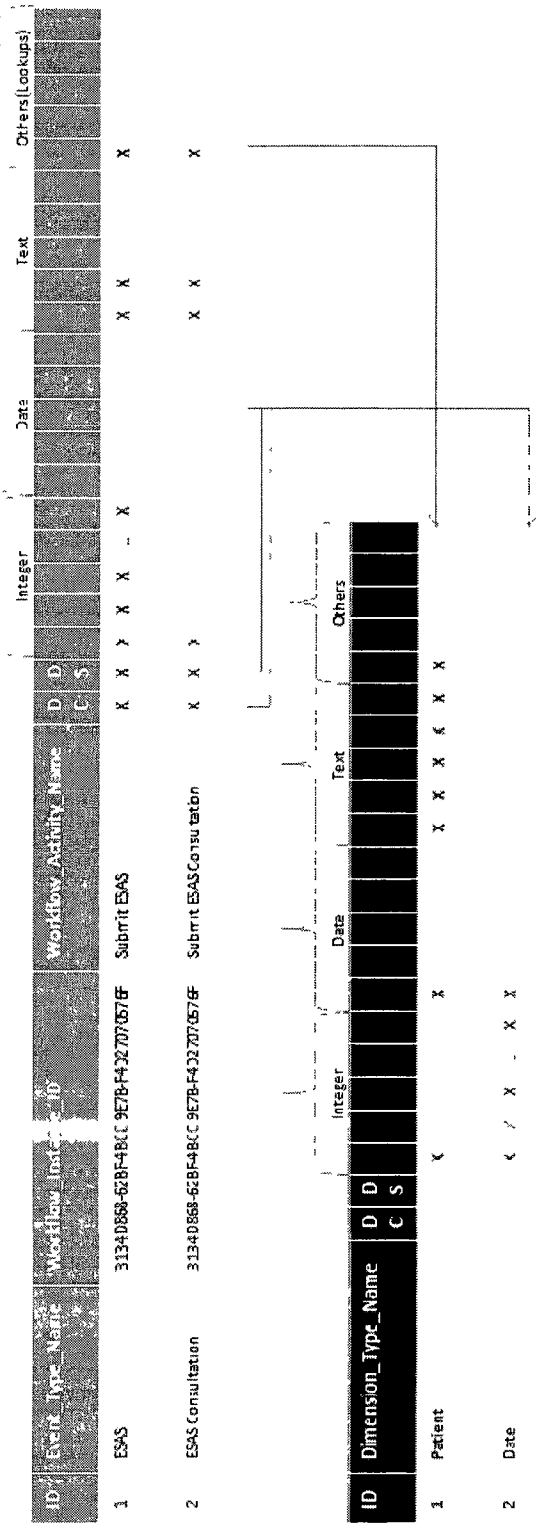


Figure 17 Event and Entity Database Tables

4.5.3 Engineering

For our approach, most of the engineering effort is spent on building the runtime environment. Building the application engine is complex task and involves the use of different components such as a workflow engine, NHibernate ORM and ASP.NET MVC framework. The runtime representation of entities using AOM and the mapping of entities to a single generic database schema may become complex. However, this upfront cost in engineering effort pays off when building applications. An application is merely an XML file that contains business level model elements and there is no programming language based application code that needs to be managed. Modeling can also be done using any regular text editor without the need for specialized model editors. The same application engine can be used for different applications by just using a different application model for the different applications.

Maintenance can be in the form of runtime environment maintenance or application model maintenance. Maintaining the runtime environment is done by modifying application code and using standard application development environments such as Microsoft Visual Studio .NET. The likelihood of introducing issues when maintaining the application engine is very high and as a result the application engine has to be verified and tested using different application models. The application engine has to be built and deployed after it has been maintained. Maintaining a model is relatively simpler. After a model definition is updated the old application model has to be replaced with the new one. Although editing a model definition is a trivial task, modifying the model definition should be done carefully to avoid introducing backward compatibility issues.

Chapter 5. Evaluation

In this chapter, we will evaluate our proposed framework (MPAF) based on our literature survey and scenario based case study, and compare it with a hard-coded two layered implementation (PAL-IS) and a generic model-driven approached (AndroMDA). We use the criteria that we identified in chapter 3.2 as the basis for the evaluation. These criteria are not intended to be complete or exhaustive and they simply represent the criteria the seemed to be most important based on our findings and various discussions with other stakeholders that were involved in the PAL-IS implementation. We will use these criteria to compare how well each approach addresses Managed Process applications in terms of declarative models, application run-time environment and development effort.

5.1. Models

Models			
Criteria	**PAL-IS**	**AndroMDA**	**MPAF**
Business Level Models workflow, roles, entities, events, alerts, indicators	No. Hand-crafted DB schema + implicit object model	Possible, mixed with code level elements (UML activity diagram + object model)	Yes

Table 1 Models Comparison

Table 1 compares MPAF, AndroMDA and PAL-IS with respect to models. In MPAF, we have used business level models specific to the requirements of Managed Process applications. Workflow, role, entity, event, alert and performance indicator

model definitions are used to directly represent elements of a managed process. Using business level models lowers application development complexity and allows domain experts to easily involve in the application development without the need for domain experts to understand the technical or code level details of the managed application framework.

AndroMDA uses generic UML based models. Use case, activity and class diagrams are used to model presentation, navigation flow, entities, data access objects, data transfer objects and services of an application. To develop managed process applications, managed process elements such as workflows, roles, events, entities, alerts and performance indicators have to be mapped to their equivalent UML representation and a code will have to be generated based on the UML representation. UML models, however, are not simple enough to easily be understood by domain experts to involve them in the application development process. Because AndroMDA uses UML based models, it can be used to develop any type of application and is not in any way limited to the development of managed process applications. In addition, AndroMDA requires the appropriate stereotypes and tags to be applied on the UML model elements so that the model will be translated by AndroMDA to generate a correct and valid source code. This additional requirement of AndroMDA imposes additional complexity even for seasoned developer that is already accustomed with UML modeling.

PAL-IS, on the other hand, has no business level models but has very limited database schema and implicit object model. As a result, all of the application source code in PAL-IS is manually crafted and is not generated or interpreted from a model.

5.2. Application Runtime Environment

Table 2 is the summary of comparisons related to the application runtime environment between the three approaches. PAL-IS is a totally customized runtime environment, whereas AndroMDA and MPAF have pre-configured environments based on their model-driven approach. PAL-IS and AndroMDA allow application or system level configuration while MPAF allows business/domain level configuration.

Application Run Time Environment			
Criteria	**PAL-IS**	**AndroMDA**	**MPAF**
Business Process Workflow	No. Hard-coded workflow.	No. Hard-coded workflow generated based on activity diagram.	Yes
SOA	Not really. Hard coded services.	Generated from a Model. Hard-code invocation	Yes. Pre-configured.
Configuration	Possible, but ad hoc.	Possible, but ad hoc.	Yes
Alerting and Monitoring	Possible, but hard-coded.	Possible, but hard coded.	Yes. Pre-configured SOA driven by Event & Alert models.
Data Models & Reporting	Possible, hard-coded to hand-crafted database schema.	Possible, hard-coded to generated database schema.	Yes. Pre-configured event-based dimensional models.

Table 2 Application Framework Comparison Criteria

5.2.1 Business Process Workflow

Support for business processes is a key criterion for any process oriented application. MPAF uses a workflow management system to manage the flow of activities in a business process based on a declarative workflow model. This makes business

process execution transparent with support for inspection, visualization, and tracking execution status. Support for dynamically changing the flow of business process instances at runtime is also provided. There is also support for long running processes that can take hours, days, weeks or longer to execute. Finally human or computer interaction is flexibly integrated using event data collection activities that wait for event data to be submitted by either a human user or an external application that sends event data via a web service.

AndroMDA uses activity diagrams to model process flows. AndroMDA uses the activity diagrams to represent navigational flow instead of modeling the actual business processes. Navigational flow can be used to model a single task that is performed from beginning to end in a short period of time. However, it is extremely difficult to model a full long running business processes using only activity diagrams that represent navigation flow of an application. It is also challenging to identify long running processes in AndroMDA's activity diagrams as they span multiple navigation flow activity diagrams. The sequence of steps in a process is not enforced as users can leave the navigation flow at any time and as result it is not possible to have process conformance to make sure that different users will go through the same set of steps to execute the same type of process.

PAL-IS does not have an automated support for business processes. It simply provides forms that users can use for data collection. If there is any processes model, it can only be in the form of a guideline that instructs users the manual steps that they have to follow to interact with the forms of the application. Also, PAL-IS does not support

long running business processes. As in the case of the AndroMDA based implementation, in PAL-IS, it is not possible to easily identify business processes that are currently running.

5.2.2 Service Oriented Architecture (SOA)

SOA facilitates reusability and when web services are used to realize an SOA framework, it also facilitates loose coupling and standards based communication between components and applications. MPAF has preconfigured services to handle presentation, persistence, event monitoring, and workflow execution and alert monitoring aspects of an application. External service invocations are defined in the application model within a workflow definition in the form of a web service invocation activity. Managed process execution is also provided as a service via the "*StartProcess*" web service.

AndroMDA, services are not preconfigured and instead they are generated from class diagram models that are decorated with the <<*Service*>> or <<*WebService*>> stereotype. External service invocations are hardcoded while designing and implementing the application. PAL-IS does not use SOA based approaches however web services can be defined manually and external service invocations can also be hardcoded easily.

In an environment where multiple applications are utilized in a day-to-day activity, such as patient care monitoring, application integration between these applications is important. Web services are suitable for integration between applications because they provide a standard based communication mechanism between services and are loosely coupled in a manner their internal implementation can be modified without the consumers of the service. PAL-IS does not have services that can be used to facilitate

integration between PAL-IS and an external application. However, database level integration can be use by creating separate database table that are used to transfer data between PAL-IS and the external application.

MPAF is enabled with web services that can be used to submit event data and to invoke or start a new managed process. MPAF also has a custom workflow activity that is used to invoke web services that reside externally. The web services that MPAF invokes are easy to identify because they are described in the model definition using the custom workflow activity that is used for invoking web services. AndroMDA also uses web services. However, unlike MPAF web service based interactions have to be hard coded and the application code has to be inspected to identify the web services an application uses.

5.2.3 Configuration

MPAF supports configurability via modification of the application model. Workflows, roles, entities, events, alerts and performance indicators are configured using the application model definition. This enables changes in the model definition to be reflected at runtime when the application engine executes the model in a running managed process application instance. In AndroMDA and PAL-IS code is the central artifact that drives the application and configuration is only possible but using ad hoc development tools that are used to modify and compile application source code.

5.2.4 Alerting and Monitoring

Alerts and event monitoring are supported in MPAF at the model level. Event data is collected in the course of executing a business process and any event that matches the trigger condition for an alert will result in an alert notification being created. Event monitoring and alerts in PAL-IS and AndroMDA are possible but are hard coded and have to be implemented using manually crafted code. In the existing implementation of PAL-IS, alerts are created manually when an event situation that matches an alert condition is manually detected.

5.2.5 Data Model & Reporting

To support multi-dimensional reporting, MPAF uses pre-configured flexible "*star schema*" based database tables and an ORM based persistence service that is used to store, retrieve and query entity and event data that are persisted according to the configuration in the application's model definition. All entity records are stored in a single table and all event records are stored in a separate event table. This enables MPAF to perform simple joins between entities and event tables to generate multi-dimensional report. However, one data field in the event table can also hold different data for different events. This type of database model has an overhead on ad hoc reporting tools because ad hoc reporting tools have to process and use the application model definition to identify records for particular event or entity type and locate the columns that are used for the event or entity type.

In AndroMDA, the database schema is generated based on the entity class diagram UML model. The database schema has to be regenerated ever time a new entity

is added or removed from the model. The database table names and field (column) names directly map to the entity class and its properties. This simplifies the use of ad hoc reporting tools over the generated tables. In situations where there is an existing legacy database, both MPAF and AndroMDA cannot easily be modified to make use of the legacy database. In PAL-IS the database schema is manually created by the application developer and ad hoc reporting tools can be used for report generation.

MPAF allows explicit declaration of performance indicators in the model. The numeric value of the performance indicator is displayed in a color coded graphical presentation to indicate whether the performance indicator value is in a normal operation range or if it is in a range that requires attention. In AndroMDA, performance indicators are calculated using hardcoded formulas and queries while the existing implementation of PAL-IS does not have direct support for performance indicators but ad hoc reporting tools can be used to generate reports with performance indicator values.

5.3. Engineering Effort

Table 3 is a summary of the engineering effort comparison between the three approaches

Engineering Effort			
Criteria	**PAL-IS**	**AndroMDA**	**MPAF**
Avoid recompile for business changes	No, Always need to code.	No. Always recompile, sometime code.	Yes, no code for workflow, roles, entities, events, alerts, indicators
Coding Complexity	High, manually coded	High, generate from models & manual coding of behavior	Low, configure engine with models

Model Abstraction level	C# Objects (Low) DB Schema (Low)	Full UML (Medium)	Business Elements (High)
Tool Support	Visual Studio	AndroMDA framework + UML Case tool (MagicDraw)	XML Editor to edit XML files.
Learning Curve	Programming Language	Programming Language, UML AndroMDA Framework	XML Business Model for workflow, roles, entities, events, alerts, indicators
Code reuse for similar applications	Low. Ad hoc	Medium. Model-structured code	High Engine based. Reuse Models.

Table 3 Engineering Effort Comparison between PAL-IS, AndroMDA and MPAF

5.3.1 Avoid Recompilation for Business Level Changes

In PAL-IS there is no explicit conversion of a model into an executable application. Instead, application code that includes C# classes and database schema is manually crafted and compiled using Visual Studio.NET. As a result, for any change that will affect the application, especially business level changes, the application source code has to be modified and recompiled again to create the executable application.

Even though AndroMDA uses UML models, any change in the model requires the whole application source code to be generated and recompiled again. Business level changes can require either a model change (with a complete regeneration of the code which must be recompiled) and/or direct modification of the portion of the source code that is generated by AndroMDA.

MPAF avoids the need to recompile the whole application when business level changes that are supported by the application model are applied. Changes in the sequence of activities in a business process workflow, roles, entities, events, alerts and performance

indicators can be applied directly on the model without the need to compile, build and deploy the application. Application models can also be easily reused to create other similar type of applications.

5.3.2 Model Abstraction Level

AndroMDA uses generic UML models that apply to any type of application. As a result, the model abstraction level is quite complex or high. AndroMDA uses stereotypes to categorize and identify UML models for code generation. It also requires UML Activity diagram models to be organized in a specific manner so that they will correctly be transformed into a code. This additional detail for the model and the generic nature of UML put AndroMDA's model complexity at a high level.

On the other hand, MPAF has a medium level of model complexity as there are only very few model elements (workflow, roles, entities, events, alerts, performance indicators) that need to be configured. MPAF models are particularly designed and targeted for managed process applications and this gives it a high model specificity value.

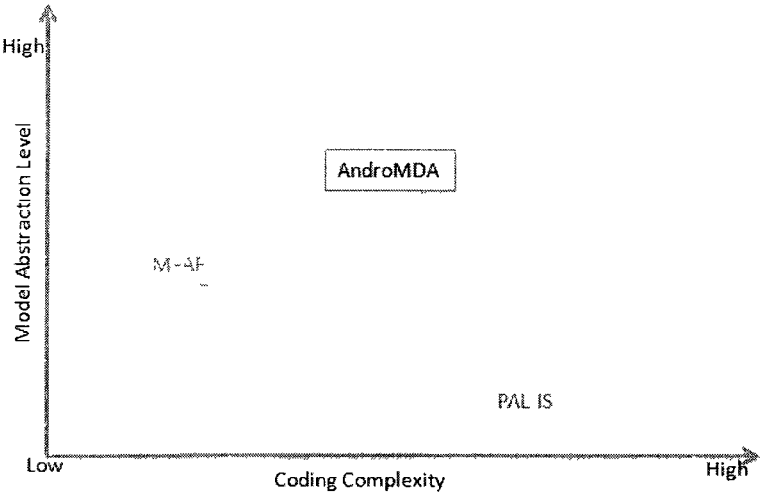


Figure 18 Application Model Comparison between PAL-IS, AndroMDA and MPAF

As shown in Figure 18, PAL-IS has a very low model complexity and very low abstraction level because it does not use models at all but the development complexity for PAL-IS is of course high.

5.3.3 Coding Complexity

The modeling complexity for AndroMDA is very high because of the initial complex development environment setup (Maven build tool, core AndroMDA libraries and AndroidVS which is a Visual Studio add-in for AndroMDA) and details of using a UML modeling tool (such as MagicDraw). In addition, the correct set of stereotypes, tagged values, data types, relationship between diagrams and other UML diagram configurations that are specific to AndroMDA have to be learnt. However, coding complexity is reduced because the code is generated from the models. The code that is generated by AndroMDA follows a particular type of structure where there are C# classes that are generated only once and are intended to be modified by the application developer. There are also classes that are generated every time AndroMDA code generation is executed and are not intended for modification because they will be overwritten every time AndroMDA is commanded to generate code. As a result this and the code structure that is chosen by the default ASP.NET AndroMDA cartridges, it generates several number of classes.

To build the existing PAL-IS application Visual Studio .NET and Microsoft SQL server are used. Setting up these tools is a straightforward task. The Microsoft .NET framework provides standard class libraries and controls for data access, data binding,

navigation, security and many other important that assist the development of data driven web applications. However, the detail of how these controls and several hundreds of other .NET framework classes has to be well understood by the application developer to build web applications. That is the coding complexity of PAL-IS is higher than AndroMDA based development.

The coding complexity when using MPAF is lower than AndroMDA. Actually, the application model can be simply edited using any text editor. The application model is defined using XML. When editing the model definition, one has to make sure that the modified model definition is well formed XML document and that content of the XML document is valid according to the XML schema that is expected by MPAF.

5.3.4 Tool Support

In PAL-IS Microsoft Visual Studio is used for development. The .NET framework including ASP.NET is used as the web development framework. Microsoft SQL Server is used as the database backend. These tools are highly supported by the vendor and the plethora of virtual communities. This makes it easy to resolve development related issues as in many cases either the issues are already resolved by the vendor or the development community has a work around for many of these issues. These tools are also very integrated with each other which help simplify the development and coding activity.

In AndroMDA, a case tool has to be used to create the UML models. We used MagicDraw 9.5 community edition for creating UML models. The Java runtime is required for development because AndroMDA's core components are Java based.

Maven, the Java based build tool that is used by AndroMDA is also required for building projects. Microsoft Visual Studio .NET is used for .NET framework based development using AndroMDA. AndroMDA provides a Visual Studio add-in called "Android/VS" that is used to build projects and generate code based on a model from within Visual Studio. The .NET/ASP.NET cartridges have to also be setup and configured correctly for AndroMDA development over the .NET framework. NHibernate is used for mapping objects to databases and Microsoft SQL Server is used as the database backend. Configuring AndroMDA(version 3.3 at the time of this writing) is a very time consuming task. Although there is limited documentation around AndroMDA and the open source user community for AndroMDA is not very active to provide information to configuration and development related issues in a timely manner.

To develop managed process applications, such as the patient monitoring application, using MPAF, a simple text editor can be used to create the XML application model definition. Even better, an XML editor can be used to modify and create the application model which would assist in making sure that XML model definition is well formed and confirms to the schema that is expected by MPAF. Although editing XML content using a text editor is manageable when the XML data small, it becomes very difficult to manage when the XML data becomes larger. A visual model edition tool can also greatly assist in the development of model definitions for managed process application. Model editing tools is one area that our approach did not address and is an area where further work is needed.

5.3.5 Learning Curve

PAL-IS has a high learning curve. Application developers have to learn the development environment and the large set of classes that are available in the .NET Framework. AndroMDA, in addition to Visual Studio .NET and the .NET framework, requires one to become versed in a UML diagramming tool (such as MagicDraw), UML diagram stereotypes, tagged values, relationship between diagrams and other configurations. This puts the learning curve for AndroMDA above PAL-IS. Building applications using MPAF requires specific knowledge about the relevant business model elements (workflows, roles, entities, events, alerts, indicators) for a managed process application. According to our implementation, it is very easy and straight forward to configure the attributes of the model elements and the relationships between these model elements once these attributes and relationships are well understood. Based on the above discussions MPAF's learning curve can be rated as easy for someone familiar with business-level models for managed processes, or average for a general developer.

5.3.6 Code Reuse for Similar Applications

Reusing the existing PAL-IS application code to build similar applications is not easy because the actual source code might have to be copied over and modified to create the new application. Whereas for MDA, both UML models and the generated source code can be re-used between similar applications but in both cases the new application has to be compiled after the UML model or the application code is reused in the new application.

However, for MPAF, the initial cost for developing the framework itself is high (see Figure 19). But once after the MPAF framework is built, the MPAF framework is highly reusable for creating similar applications. This is because, creating a new application is simply done by creating a model definition for the new application. It would be interesting to consider whether a generic model-driven framework like AndroMDA might be useful for creating our framework.

Overall, PAL-IS has low reusability level for building similar applications. AndroMDA has a better level because some portion of UML model can be reused when building similar applications. MPAF has the highest level of reuse because the framework is reused every time we create a new managed process application model definition. In a way this make MPAF a software product line framework that is used for creating a set of similar applications from a common set of services and an application engine using a common technique for developing these applications.

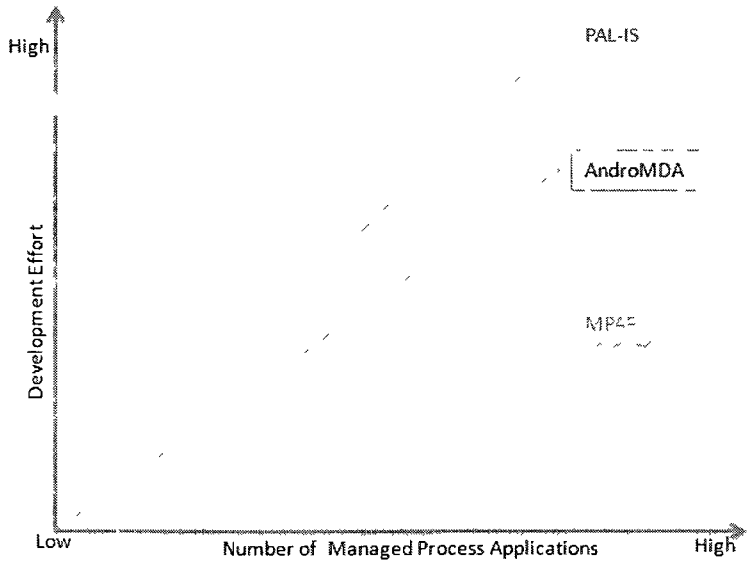


Figure 19 Engineering Effort Comparison between PAL-IS, AndroMDA and MPAF

Chapter 6. Conclusions

6.1. Summary of Contributions

In section 1.2, we outlined the key contributions of this research. Here we summarize and discuss these contributions based on our scenario and evaluation.

Contribution 1: Definition for managed process applications that identifies the key characteristics, services and components.

In this thesis we have provided a definition for managed process applications as an application that manages a process that is controlled by an automated system and contains a set of coordinated activities conducted by both people and systems to accomplish a specific organizational goal and is tracked and monitored for performance management purposes.

A managed process application facilitates the participation roles that perform their task collaboratively to achieve some organizational goal. A managed process application enables data collection to obtain operational data for business activities and their related business events. A managed process application also supports monitoring of event data that is captured while performing business activities. These event data are tracked and analyzed to trigger alert notifications for actionable events. Managed process applications also allow performance indicators to measure how well an organization's business activities are being executed. To support frequent changes in business process, event and entity data structure, alerts and performance indicators, managed process applications must be configurable. In addition to the above characteristics of a managed

process application, our thesis also identified the services and components that are required for a managed process application. The key services include presentation, workflow, event monitoring and persistence services. Our thesis also identified the use of an event-based dimensional model for supporting flexible data storage and reporting.

Contribution 2: Set of evaluation criteria specific to managed process applications for comparing and evaluation model-driven application frameworks.

In section 3.2 we identified criteria for evaluating managed process application frameworks. These criteria are also discussed in detail in Chapter 5. The two areas that are covered by these criteria are the application framework and the engineering effort considerations that must be taken into account when using such a framework. Application framework criteria include support for business level models (workflow, roles, entities, events, alerts, and indicators), support for business process workflow, SOA, configurability, alerting and monitoring and data models and reporting. Engineering effort criteria include avoiding recompilation for business level changes, coding complexity, model abstraction level, tool support and code reuse for similar applications. These criteria can be used to evaluate a framework and compare it with other frameworks. These criteria can also be used for evaluating if a managed process application based approach is good fit for developing applications for a particular business problem at hand.

Contribution 3: A model driven application framework for managed processes that enables run time configuration and monitoring of managed processes based on declarative model definitions.

Current application development techniques and model driven approaches, such as OMG's MDA, are code centric and advocate code generation from a model. These development approaches are generic in a way that they can be used for developing of any type of application regardless of the business domain. In this paper we propose an application framework that focuses on interpreting and executing domain specific models that can be used for building configurable managed process applications.

The framework focuses on two key aspects of an application: the static *model* definition that describes elements and behavior of the application at design time and the *application engine* that interprets the model into a running application instance at run time. The model describes workflow, role, entity, event, alert and indicator definitions that will be consumed by the application engine. The application engine consists of presentation, workflow, persistence and monitoring services that are used to render the application behavior based on its model definition.

Contribution 4: Preconfigured services and service oriented architecture for managed processes.

Our implementation of MPAF delivers preconfigured presentation, workflow, event monitoring and persistence services. These services are available out of the box when developing applications using MPAF. This facilitates rapid application development and encourages application developers to make use of services and take advantage of service oriented architecture.

6.2. Thesis Limitations

Event processing in our framework is mainly based on the alert notification trigger conditions that are defined for an event. This simple form of event processing is very basic and can be enhanced to support identification of advanced types of events.

In theory, our framework should be applicable to any type of managed process application that has the characteristics we have highlighted. This would include applications in other areas such as insurance and government, as well as other applications in healthcare. However, at this time we have a prototyped framework that was used to build a proof of concept managed process application for a palliative care severe pain patient monitoring scenario. More comprehensive case studies should be performed to further validate our approach and to indentify an exhaustive list of criteria that can be used for evaluation. Using our framework in other domains may also indicate additional requirements and components that our framework should address.

The model definition that we developed in our scenario is relatively simple in that it can be edited by a simple text editor. However, the model for real scenarios will much larger in scope and will have several model elements which can become difficult to manage and may require a visual modeling tool. We have attempted building a visual modeling tool with very limited functionality and it has helped us to understand that it is difficult to build one and that even a modeling tool with a limited functionality can become handy.

6.3. Future Work

6.3.1 Complex Event Processing

Capturing business process events only, does not provide enough information about an organization's business activity. These events need to be further processed to provide additional insight over an organization's operation. One possible future enhancement of the proposed framework is to incorporate complex event processing. Event processing can range from simple to complex event processing. Results of an event processing can be used to generate reports, initiate new business processes, and detect opportunities and risks. It can also be used to provide key business intelligence that can be used to optimize an organization's business operation.

6.3.2 Visual Model Editing Tool

XML is usually said to be human readable because of its visual structuring. However, when an XML text becomes large it becomes tedious and unreadable. Visual representation is better alternative. Our framework simply uses an XML editor and a visual workflow editor that was provided by the Microsoft Workflow foundation. This makes it difficult to comfortably manage frequent model modifications. If a new WFMS is used, users or developers who are responsible for editing the model have to learn the new WFMS visual notation. Therefore, visual modeling tool and visual representation of a managed process using standardized notations such as BPMN is an area that can be further studied.

References

[Ambler97] Ambler, S. W. (1997). *Mapping Objects To Relational Databases.* Retrieved 04 10, 2010, from AmbySoft: www.AmbySoft.com/mappingObjects.pdf

[AndroMDA10] *AndroMDA.* (n.d.). Retrieved 03 06, 2010, from AndroMDA: http://www.andromda.org

[Bass03] Bass, L., Clements, P., & Kazman, R. (2003). *Software Architecture in Practice 2nd ed.* Addison-Wesley.

[Bui03] *Building Flexible, Large-Scale Application Architecture.* (2003). Retrieved May 20, 2009, from CCPACE website: http://www.ccpace.com/Resources/documents/application_arch.pdf

[Clements01] Clements, P., & Northrop, L. (2001). *Software Product Lines: Practices and Patterns.* Addison-Wesley.

[Curbera02] Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., & Weerawarana, S. (2002). Unraveling the Web Services Web. *IEEE* , vol. 6, no. 2, pp. 86–93.

[CEP08] *Empowering the business to sense and respond: Delivering Business Event Processing with IBM WebSphere Business Events.* (2008). Retrieved 05 01, 2010, from IBM: ftp://ftp.software.ibm.com/software/integration/wbe/5565_Empowering-the-Business-US-white-paper.pdf

[Forrester08] (2008). *Enabling Dynamic Business Processes With BPM And SOA.* Forrester.

[W3C] *Extensible Markup Language (XML).* (n.d.). Retrieved May 20, 2009, from World Wide Web Consortium(W3C): http://www.w3.org/XML/

[Eze09] Eze, B., Kuziemsky, C., Peyton, L., Middleton, G., & Mouttham, A. (2010). Policy-based Data Integration for e-Health Monitoring Processes in a B2B Environment: Experiences from Canada. *Journal of Theoretical and Applied Electronic Commerce Research* , 56-70.

[Fontura99] Fountura, M. (1999). *A Systematic Approach For Framework Development.* Rio de Janerio: Ph.D Dissertation, Computer Science Department, Pontifcal Catholic Iniversity of Rio de Janerio.

[Fowler03] Fowler, M. (2003). *Patterns of Enterprise Application Architecture.* Addison Wesley.

[Frankel03] Frankel, D. S. (2003). *Model Driven Architecture: Applying MDA to Enterprise Computing.* Wiley Publishing.

[Gam95] Gamma, E., Helm, R., Johnson, R., & Vlissides, J. M. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley .

[Hailpern06] Hailpern, B., & Tarr, P. (2006). Model-driven development: the good, the bad, and the ugly. *IBM Systems Journal* , 451-461.

[Hess09] Hess, R. (2009). The Missing Link to Success : Using a Business Process Management System to Automate and. *Journal of Healthcare Information Management* , vol. 23, no. 1.

[Henver04] Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design Science in Information Systems Research. *MIS Quarterly* , 75-105.

[NHibernate10] *Hibernate-Relational Persistence for Java and .Net.* (n.d.). Retrieved 04 10, 2010, from Hibernate: http://www.hibernate.org/

[Huhns05] Huhns, M. N., & Singh, M. P. (2005). Service-Oriented Computing: Key Concepts and Principles. *EEE Internet Computing* , vol. 9, no. 1, pp. 75-81.

[Jurič06] Jurič, M. B., Mathew, B., & Sarang, P. (2006). *Business Process Execution Language for Web Services.*

[Keller97] Keller, W. (1997). Mapping Objects to Tables - A Pattern Language. *Proc. Of European Conference on Pattern Languages of Programming Conference (EuroPLOP)''97.*

[Kimball02] Kimball, R. (2002). *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling, 2nd Edition.* John Wiley & Sons.

[Kuziemsky08] Kuziemsky, C. E., Weber-Jahnke, J. H., Lau, F., & al., e. (2008). An Interdisciplinary Computer-based Information Tool for Palliative Severe Pain Management. *Journal of the American Medical Informatics Association* , 374-382.

[Lawrence97] Lawrence, P. (1997). *Workflow Handbook 1997, Workflow Management Coalition.* New York: John Wilen and Sons.

[Leymann02] Leymann, F., Roller, D., & Schmidt, M.-T. (2002). Web services and business process management. *IBM Systems Journal* , vol 41. no 2, p.198-211.

[Liu06] Liu, H. (2006). *Model Based Enterprise Processes in a Service Oriented Architecture.* Ottawa.

[Liu08] Liu, X., Peyton, L., & Kuziemsky, C. (2008). A Requirement Engineering Framework for Electronic Data Sharing of Health Care Data Between Organizations. In *E-Technologies: Innovation in an Open World* (pp. 279-289). Springer Berlin Heidelberg.

[Luckham04] Luckham, D. (2004). *The Beginnings of IT Insight:Business Activity Monitoring.* Retrieved 05 01, 2010, from Complex Events: http://www.complexevents.com/media/articles/cep-article-three.pdf

[Michelson06] Michelson, B. M. (2006, February 2). *Event-Driven Architecture Overview Event-Driven SOA Is Just Part of the EDA Story.* Retrieved January 20, 2010, from Object Management Group: www.omg.org/soa/Uploaded%20Docs/EDA/bda2-2-06cc.pdf

[Microsoft09] *Microsoft Applicaton Architecture Guide 2nd Edition (Patterns & Practices)* . (2009). Microsoft Corporation.

[Middleton09] Middleton, G. (2009). *A Framework for Continuous Compliance Monitoring of B2B Processes. Msc Thesis.* Ottawa: University of Ottawa, Canada. .

[Forrester07] Moore, C., & Rymer, J. R. (2007). *The Dynamic Business Applications Imperative.* Forrester Research.

[OASIS09] OASIS. (n.d.). *BPEL.* Retrieved June 2009, from OASIS: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel

[OAS09] OASIS. (n.d.). *BPEL.* Retrieved June 2009, from OASIS: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel

[OASIS06] OASIS. (2006, August 2). *Reference Model for Service Oriented Architecture 1.0.* Retrieved June 13, 2009, from OASIS: http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf

[Ref06] OASIS. (2006, August 2). *Reference Model for Service Oriented Architecture 1.0.* Retrieved June 13, 2009, from OASIS: http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf

[OMG10] *OMG Model Driven Architecture.* (n.d.). Retrieved 03 06, 2010, from OMG - Object Management Group: http://www.omg.org/mda/

[Ross04] Ross et al., B. G. (2004). The Canadian adverse events study: the incidence of adverse events among hospital patients in (2004). *Canadian Medical Association Journal* , 1678--1686.

[Schmidt06] Schmidt, D. C. (2006). Model Driven Engineering. *IEEE Computer* .

[Smith03] Smith, H., & Peter, F. (2003). *Business Process Managment:The Third Wave.* Meghan-Kiffer Press.

[Stahl06] Stahl, T., & Voelter, M. (2006). *Model-Driven Software Development: Technology, Engineering, Management.* Wiley Publishing.

[AdverseEvent04] *The Canadian Adverse Events Study and Medication Safety.* (2004, 07). Retrieved 04 10, 2010, from Institute for Safe Medication Practices Canada : www.ismp-canada.org/download/hnews/HNews0407.pdf

[Thomas04] Thomas, D. (2004). The Elusive Search for Business Frameworks. *Journal of Object Technology, vol. 3, no. 1* , 7-13.

[Trites09] Trites, G., & Boritz, J. E. (2009). eBusiness: A Canadian Perspective for a Networked World - 3rd Edition. Toronto, Ontario, Canada.

[Truyen06] Truyen, F. (2006, January). *The Fast Guilde to Model Driven Architecture - The Basics of Model Driven Architecture.* Retrieved March 06, 2010, from OMG-Object Management Group: http://www.omg.org/mda/mda_files/Cephas_MDA_Fast_Guide.pdf

[van_der_Alast03] van der Alast, W. M., Hofstede, A. H., & Weske, M. (2003). Business Process Management: A Survey. In *Business Process Management* (p. 1019). Springer Berlin / Heidelberg.

[WebService04] *Web Services Architecture.* (2004, February 11). Retrieved May 20, 2009, from World Wide Web Consortium(W3C): http://www.w3.org/TR/ws-arch/

[Welicki08] Welicki, L., Yoder, J. W., & Wirfs-brock, R. (2008). A Pattern Language for Adaptive Object Models: Part I- Rendering Patterns.

[Welicki09] Welicki, L., Yoder, J. W., Wirfs-brock, R., & Johnson, R. E. (n.d.). *Towards a Pattern Language for Adaptive Object Models.* Retrieved May 20, 2009, from Adaptive Object Model: www.adaptiveobjectmodel.com/Posters/AOM_Patterns_Map1.ppt

[Weske07] Weske, M. (2007). *Business Process Management: Concepts, Languages, Architectures.* Berlin Heidelberg: Springer-Verlag.

[WHO10] WHO. (n.d.). *Palliative Care Defintion.* Retrieved from WHO: http://www.who.int/cancer/palliative/definition/en/

[Yod00] Yoder, J. W., & Razavi, R. (2000). Metadata and Adaptive Object Models. *ECOOP'2000 Workshop; Lecture Notes in Computer Science* .

[Yoder02] Yoder, J., & Johnson, R. (2002). *The Adaptive Object-Model Architectural Style.*

[Yoder01A] Yoder, J., Balaguer, F., & Johnson, R. (2001). Adaptive Object-Models for Implementing Business Rules. *Third Workshop on Best-practices for Business Rules Design and Implementation.*

[Yoder01B] Yoder, J., Balaguer, F., & Johnson, R. (2001). Architecture and design of adaptive object-models. *OOPSLA* (pp. 50-60). New York: ACM.

Appendix

Appendix A

```
<AppModel>

  <ApplicationInfo>
    <Title>Palliative Care Patient Care Monitoring Application</Title>
    <Description>This is a sample patient care monitoring application
developed as a proof of concept at the University of
Ottawa.</Description>
  </ApplicationInfo>

  <Roles>
    <Role name="Patient"/>
    <Role name="CaseManager"/>
    <Role name="Nurse"/>
    <Role name="TeamAdministrator"/>
  </Roles>

  <EntityDefinitions>
    <EntityType name="Patient">
      <PropertyType name="DateCreated" type="System.DateTime"/>
      <PropertyType name="Patient ID" type="System.String" />
      <PropertyType name="OHIP Number" type="System.String" />
      <PropertyType name="First Name" type="System.String" />
      <PropertyType name="Last Name" type="System.String" />
      <PropertyType name="Gender" type="System.String"
isPickList="True" pickList=";Male;Female"/>
      <PropertyType name="DOB" type="System.DateTime" />
      <PropertyType name="Religion" type="System.String"
isPickList="True" pickList=";Christian;Hindu;Other" />
      <PropertyType name="Other - Religion" type="System.String" />
      <PropertyType name="Family Physician" type="System.String" />
      <PropertyType name="Referring Physician" type="System.String" />
      <PropertyType name="Emergency Contact" type="System.String"
isMultiLine="True"/>
    </EntityType>
  </EntityDefinitions>

  <EventDefinitions>
    <EventType name="ESAS">
      <PropertyType name="WorkflowInstanceID" type="System.String"/>
      <PropertyType name="WorkflowActivityName" type="System.String"/>
      <PropertyType name="DateCreated" type="System.DateTime"/>
      <PropertyType name="DateSubmitted" type="System.DateTime"/>
      <PropertyType name="Patient ID" type="System.Int32"
isLookup="True" lookupName="Patient" lookupDisplayFormat="{Patient ID}
- {First Name} {Last Name}" readonly="true" calculated="true"
value="System.User.[Patient ID]"/>
      <PropertyType name="Pain" type="System.Int32" isPickList="True"
pickList="1;2;3;4;5;6;7;8;9;10" />
```

```
      <PropertyType name="Tiredness" type="System.Int32"
isPickList="True" pickList="1;2;3;4;5;6;7;8;9;10" />
      <PropertyType name="Nausea" type="System.Int32" isPickList="True"
pickList="1;2;3;4;5;6;7;8;9;10" />
      <PropertyType name="Depression" type="System.Int32"
isPickList="True" pickList="1;2;3;4;5;6;7;8;9;10" />
      <PropertyType name="Anxiety" type="System.Int32"
isPickList="True" pickList="1;2;3;4;5;6;7;8;9;10" />
      <PropertyType name="Drowsiness" type="System.Int32"
isPickList="True" pickList="1;2;3;4;5;6;7;8;9;10" />
      <PropertyType name="Appetite" type="System.Int32"
isPickList="True" pickList="1;2;3;4;5;6;7;8;9;10" />
      <PropertyType name="Well Being" type="System.Int32"
isPickList="True" pickList="1;2;3;4;5;6;7;8;9;10" />
      <PropertyType name="Shortness of Breath" type="System.Int32"
isPickList="True" pickList="1;2;3;4;5;6;7;8;9;10" />
      <PropertyType name="Other Problem Description"
type="System.String" isMultiLine="True"/>
      <PropertyType name="Other Problem" type="System.Int32"
isPickList="True" pickList="1;2;3;4;5;6;7;8;9;10" />
    </EventType>

    <Event name="ESAS_Consultation">
      <PropertyType name="WorkflowInstanceID" type="System.String"/>
      <PropertyType name="WorkflowActivityName" type="System.String"/>
      <PropertyType name="DateCreated" type="System.DateTime"/>
      <PropertyType name="DateSubmitted" type="System.DateTime"/>
      <PropertyType name="Patient ID" type="System.Int32"
isLookup="True" lookupName="Patient" lookupDisplayFormat="{Patient ID}
- {First Name} {Last Name}" readonly="true" calculated="true"
value="ESAS.[Patient ID]" />
      <PropertyType name="Assessment Note" type="System.String"
isMultiLine="True"/>
    </EventType>

  </EventDefinitions>

  <WorkflowDefinitions>

    <Workflow name="ESASWorkflow" description="ESAS">

      <CustomSequentialWorkflowActivity name="ESASWorkflow">

        < CollectEventDataActivity displayName="Submit ESAS"
name="submitESAS" eventTypeName="ESAS" roles="Nurse;Patient" />

        <CollectEventDataActivity displayName="Submit ESAS
Consultation" name="submitESASConsultation"
eventTypeName="ESAS_Consultation" roles="Nurse" />

      </CustomSequentialWorkflowActivity>

    </Workflow>

  </WorkflowDefinitions>

  <PerformanceIndicators>
```

```
    <PerformanceIndicator name="Daily Average Severe Pain Management
Response In Hours" value="avg(hour(ESAS_Consultation.DateSubmitted) -
hour(ESAS.DateSubmitted))" filter="ESAS.DateSubmitted between
current_date() and current_date() + 1" roles="TeamAdministrator" >
      <IndicatorRange min="0" max="2" status="green"/>
      <IndicatorRange min="2" max="4" status="yellow"/>
      <IndicatorRange min="4" max="6" status="orange"/>
      <IndicatorRange min="6" max="" status="red"/>
    </PerformanceIndicator>

  </PerformanceIndicators>

  <AlertNotifications>

    <AlertNotification name="Severe Pain Alert" eventTypeName="ESAS"
triggerCondition="ESAS [Pain]>=8" triggerWorkflow="" roles="Nurse">
      </AlertNotification>

  </AlertNotifications>

</AppModel>
```