Design of Workflows for Context-aware Applications

By

Abu Zafar Abbasi

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

(April 2013)

Supervised By: Dr. Zubair Ahmed Shaikh
Author’s Declaration

It is hereby declared that the work presented in this thesis is my original work and does not contain any material that is already published by another person except where properly cited and referenced. The submitted text contains a description of my research work that I have carried out during my PhD studies and does not include any material that has been submitted to qualify for the award of any other degree or diploma in any university or institution.

Signature _____________________

Name_____________________________
Plagiarism Undertaking

It is hereby declared that the work presented in this thesis is not copied or plagiarized from any other source or existing literature, except where due references and citations are made.

Signature _____________________

Name_________________________
Abstract

Workflow technology is widely used for automation of business processes that supports their modeling, enactment, tracking, and analysis. Processes represented as workflows are composed of ordered and coordinated activities that integrate humans, resources, and applications. Workflow brings flexibility through explicit definition of processes enabling applications to be easily managed without disturbing the program code. Such unique characteristics of workflow technology enable it to be used in domains other than business also. In fact, workflow based programming brings a paradigm shift in the development of applications in more flexible, natural, and usable way.

Recent advancement in communication, electronics, sensors, and software architectures pose requirements of context-aware applications that are dynamic, flexible, and adaptable. These applications are not limited to emerging domains like pervasive and ubiquitous computing but also required by traditional business processes to survive in today’s competitive market. The design and maintenance of context-aware applications may greatly be simplified by using workflows to represent variability associated with these applications. However, rigidity in workflow representation and enactment necessitates modifications and extensions in workflow technology before it could be adopted in context-aware application development.

This thesis investigates key issues and challenges in adopting workflow technology for context-aware applications. Governed by the guiding principles of flexibility, adaptability, extensibility, reusability, autonomy, and usability; a generic framework for context-aware workflow design and management is presented. Making use of explicit modeling of context information, adaptation rule-set, and sub-workflows, a conventional workflow application may be transformed to become context-aware. A Context-Activity Architecture is presented that describes the integration and working of constituent components of context-aware workflow system. Most notably, an activity is represented as specially designed context-aware activity that adapts itself to a contextually suitable activity at runtime with the help of necessary functionality embedded within the activity itself. The proposed architecture is realized through the development of a context aware workflow designer tool (CAWD) using Windows Workflow Foundation classes. A couple of example scenarios and simulated testing elaborate and verify the intended adaptation operations and handling of evolutionary changes within a process that may be experienced by a context-aware workflow application.
Dedicated to

*My (late) Father*
Acknowledgements

During this arduous and challenging journey of research work, I am privileged to have the assistance of many people for which they have my endless gratitude and respect.

First of all, I would like to thank my supervisor Dr. Zubair Ahmed Shaikh for his support, inspiration and belief in me. He provided his full technical and moral support during the course of my PhD, for which I am extremely grateful.

I am highly indebted to Higher Education Commission (HEC), Pakistan, for providing me the opportunity, funds and resources for my PhD study and research work.

I would also like to thank faculty and staff members of Computer Science department, FAST-NU, Karachi Campus and fellow researchers at Center for Research in Ubiquitous Computing (CRUC), for promoting and maintaining a healthy academic and research atmosphere.

I would also like to acknowledge the efforts of undergraduate/graduate students for working with me in various projects. Special thanks to Mr. Muhammad Uzair Ahsan for providing support in programming.

I extend my gratitude to anonymous reviewers who reviewed my submitted papers in international/national conferences and journals. Moreover, I thank my foreign and local reviewers for sparing time to critically evaluate my thesis and provided me useful comments and suggestions.

It is impossible not to acknowledge my family who suffered the most of this academic expedition of mine. I am really grateful to my wife Shabina, kids Absar, Ayesha, and Ali for their understanding and support. I salute my parents, especially mother, who were my first teachers and now my benefactor with their prayers.
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE OF CONTENTS</td>
<td>vii</td>
</tr>
<tr>
<td>LIST OF PUBLICATIONS</td>
<td>x</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>xi</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>xiii</td>
</tr>
<tr>
<td>LIST OF ABBREVIATIONS</td>
<td>xiv</td>
</tr>
<tr>
<td><strong>CHAPTER 1: INTRODUCTION</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 MOTIVATION</td>
<td>3</td>
</tr>
<tr>
<td>1.2 RESEARCH PROBLEM STATEMENT</td>
<td>5</td>
</tr>
<tr>
<td>1.3 RESEARCH OBJECTIVES</td>
<td>5</td>
</tr>
<tr>
<td>1.4 RESEARCH METHODOLOGY</td>
<td>6</td>
</tr>
<tr>
<td>1.5 CONTRIBUTIONS</td>
<td>6</td>
</tr>
<tr>
<td>1.6 THESIS ORGANIZATION</td>
<td>7</td>
</tr>
<tr>
<td>1.7 SUMMARY</td>
<td>7</td>
</tr>
<tr>
<td><strong>CHAPTER 2: BACKGROUND</strong></td>
<td>8</td>
</tr>
<tr>
<td>2.1 INTRODUCTION</td>
<td>8</td>
</tr>
<tr>
<td>2.2 WORKFLOW TECHNOLOGY</td>
<td>8</td>
</tr>
<tr>
<td>2.2.1 Workflow Terminologies</td>
<td>11</td>
</tr>
<tr>
<td>2.2.2 Workflow Modeling, Instances and Languages</td>
<td>14</td>
</tr>
<tr>
<td>2.3 CONTEXT AND CONTEXT-AWARENESS</td>
<td>16</td>
</tr>
<tr>
<td>2.3.1 Preamble to Context</td>
<td>16</td>
</tr>
<tr>
<td>2.3.2 Context Definitions</td>
<td>17</td>
</tr>
<tr>
<td>2.3.3 General Classification of Context</td>
<td>18</td>
</tr>
<tr>
<td>2.3.4 Context Awareness</td>
<td>20</td>
</tr>
<tr>
<td>2.3.5 Context Modeling and Reasoning</td>
<td>21</td>
</tr>
<tr>
<td>2.3.6 Context Acquisition and Management</td>
<td>22</td>
</tr>
<tr>
<td>2.4 CONTEXT-AWARE WORKFLOWS</td>
<td>23</td>
</tr>
<tr>
<td>2.5 SUMMARY</td>
<td>25</td>
</tr>
<tr>
<td><strong>CHAPTER 3: SURVEY OF EXISTING CONTEXT-AWARE WORKFLOW SYSTEMS AND IDENTIFICATION OF ISSUES</strong></td>
<td>26</td>
</tr>
<tr>
<td>3.1 INTRODUCTION</td>
<td>26</td>
</tr>
<tr>
<td>3.2 CHARACTERISTICS OF CONTEXT-AWARE WORKFLOW APPLICATIONS</td>
<td>27</td>
</tr>
<tr>
<td>3.2.1 Flexibility in Workflow Representation</td>
<td>28</td>
</tr>
<tr>
<td>3.2.2 Modifications in Workflow Enactment</td>
<td>28</td>
</tr>
<tr>
<td>3.2.3 Context Information Representation</td>
<td>28</td>
</tr>
<tr>
<td>3.2.4 Context Integration in Workflow Representation</td>
<td>28</td>
</tr>
<tr>
<td>3.2.5 Support of Context Management Systems</td>
<td>29</td>
</tr>
<tr>
<td>3.2.6 Context-based Adaptation</td>
<td>29</td>
</tr>
<tr>
<td>3.2.7 Activity Binding Times</td>
<td>29</td>
</tr>
<tr>
<td>3.2.8 Rollback and Migration Strategies</td>
<td>30</td>
</tr>
<tr>
<td>3.2.9 Evolutionary Support features</td>
<td>30</td>
</tr>
</tbody>
</table>
List of Publications

**Journal Publications**


**Conference Publications**


List of Figures

Figure 1-1: Survey results for flexibility satisfaction/interest in workflow products ....... 5
Figure 2-1 Role of workflow automation ................................................................. 10
Figure 2-2: Workflow system characteristics ................................................................ 11
Figure 2-3: Relationship between workflow terms .......................................................... 13
Figure 2-4: Workflow application development process .................................................. 14
Figure 2-5: Process definition meta-model ...................................................................... 15
Figure 2-6: Using context model for reasoning ............................................................. 22
Figure 4-1: Framework for context-aware workflow design and management ............... 42
Figure 4-2: Workflow representation ............................................................................. 44
Figure 4-3: Context-aware activity .................................................................................. 45
Figure 4-4: Smart workflow definition meta-model ....................................................... 48
Figure 4-5: An illustration of workflow pool .................................................................. 49
Figure 4-6: Context meta-model ................................................................................... 52
Figure 4-7: Context-aware workflow enactment strategies ............................................ 53
Figure 4-8: A view of context management ................................................................... 54
Figure 5-1: Context-Activity architecture ................................................................. 57
Figure 5-2: Algorithm for dynamic finding of appropriate sub-workflows .................... 59
Figure 5-3: Spawn algorithm ....................................................................................... 59
Figure 5-4: The deletion operation ............................................................................... 60
Figure 5-5: The selection operation ............................................................................. 60
Figure 5-6: The addition operation ............................................................................... 61
Figure 5-7: A hypothetical context-aware workflow for simulated testing .................... 62
Figure 5-8: Selection of appropriate sub-workflow ...................................................... 64
Figure 5-9: Editing adaptation rule-set at runtime ....................................................... 66
Figure 5-10: Editing workflow pool at runtime ............................................................. 67
Figure 6-1: Workflow host ......................................................................................... 70
Figure 6-2: CAWD architecture .................................................................................. 71
Figure 6-3: Basic meta-model of context aware activity ............................................... 72
Figure 6-4: CAWD4 snapshot .................................................................................... 74
Figure 6-5: Major classes defined for implementation of SmartActivity .......................... 75
Figure 6-6: Realization of context aware activity architecture in CAWD ....................... 76
Figure 6-7: SmartActivity's ContextGroupIdArgs editor ............................................. 78
Figure 6-8: SmartActivity's ContextVariables editor ................................................... 79
Figure 6-9: SmartActivity's Rule-set editor .................................................................. 81
Figure 6-10: CAWD4 workflow pool editor ................................................................. 82
Figure 6-11: CAWD4 SmartActivity ............................................................................ 83
Figure 6-12: CAWD4 ContextRetrievalActivity ......................................................... 84
Figure 6-13: CAWD3.5 snapshot ............................................................................... 85
Figure 6-14: A local service context aware activity ...................................................... 87
Figure 6-15: Communication between host and workflows through local service ........ 88
Figure 7-1: An illustration of smart agriculture ............................................................. 91
Figure 7-2: An agriculture ontology-based context model ............................................ 93

xi
List of Tables

Table 2-1: General classification of context ................................................................. 19
Table 3-1: A comparison of context-aware workflow research .................................... 35
Table 4-1: Rule-sets example as (Condition, Action) tuple ........................................ 50
Table 7-1: Example of sensors used in smart agriculture ............................................ 92
Table 7-2: Example relationships for context derivation in smart agriculture ............. 95
Table 7-3: Context entities and adaptation rule-set for smart irrigation workflow ........ 99
Table 7-4: Context entities and adaptation rule-set for smart loan sanction workflow .. 107
Table 7-5: New adaptation rule-set according to change in loan sanction policy ....... 109
Table 7-6 Checklist of additional operations provided by developed context-aware Workflows ................................................................. 112
List of Abbreviations

AR  Adaptation Rule-set
BPEL  Business Process Execution Language
BPEL4WS  Business Process Execution Language for Web Services
BPMN  Business Process Modeling Notation
CA  Context-aware Activity
CAWD  Context-aware Workflow Designer
CAWE  Context-aware Workflow Execution
CAWM  Context-aware Workflow Management
CMI  Context Model Interface
CSR  Context Sensitive Region
DBMS  Database Management System
EAI  Enterprise Application Integration
ERP  Enterprise Resource Planning
GPS  Global Positioning System
HCI  Human Computer Interaction
IT  Information Technology
jPDL  Java Process definition Language
PA  Placeholder Activity
RDBMS  Relational Database Management System
RFID  Radio-frequency Identification
SPARQL  a recursive acronym for SPARQL Protocol and RDF Query Language
SQL  Structured Query Language
SWRL  Semantic Web Rule Language
UI  User Interface
uWDL  Ubiquitous Workflow Description Language
WF  (Microsoft Windows) Workflow Foundation
WFMC  Workflow Management Coalition
WFMS  Workflow Management System
WP  Workflow Pool
XAML  Extensible Application Markup Language
XML  Extensible Markup Language
XPDL  XML Process Definition Language
YAWL  Yet Another Workflow Language
Chapter 1

Introduction

With the advancement and rapid changes in mobile technology, wireless communication technology, sensor technology, software architectures, and network infrastructures, the demand of smart applications that are dynamic, flexible, and adaptable greatly evolves. Not only emerging domains like pervasive and mobile computing demands applications that are context-aware but traditional business and technical domains also pose additional requirements of context-awareness in their applications to optimize their business and technical goals. This leads to the popularity and evolution of context-aware applications that are designed and managed in a manner to adapt its behavior according to the current context information related to the user and its environment. The context may be treated as any relevant information (e.g., identification, location, network bandwidth, device used, user profile etc.) that characterizes the situation of a user is in and may be used for adaptation; whether sensed, non-sensed or derived. It is, therefore, recommended that applications should be designed and executed such that they must be aware and adapt to their changing contexts.

Applications developed for any system always comprised of processes that must be executed to achieve certain organizational or technical goals. Usually these processes are hard coded into applications (one may say that until recently this was the case), and are very difficult to evolve or change with process improvements and are not suitable for dynamic environments altogether. The solution lies in change of design and management style of applications making them flexible, adaptable and extensible. Leymann et al concluded with various studies in business domains that modeling processes as workflows, enterprises were able to modify or improve their processes more easily and effectively (Leymann and Roller 2000). Although workflow technology is more popular due to process automation it brings within an organization, the key to its success lies within the revolutionary concept of separating the process logic from application through explicit modeling of processes as workflows out of the application. This separation of concerns makes business applications more flexible and adaptable to
survive within a competitive and dynamic market. A workflow is considered as a means to model a process by representing a set of coordinated activities or services. These workflows are hosted in a business or technical application where a workflow engine executes them to achieve the desired goal. So, in effect they provide the flexibility of changing processes without disturbing the application logic. These properties of workflow make it a suitable and an ideal candidate to design processes which are destined to run in dynamic environments (Ranganathan and McFaddin 2004), (Sucha, Sea, and Maria 2008), (Abbasi, Shaikh, and Shaikh 2010).

However, in traditional workflow systems, the workflow definition itself is quite rigid and generally do not allow processes to evolve or change on the fly easily. In other words, such workflow definitions are useful if limited to processes that are typically characterized by predictability and repetitiveness. For example, an insurance workflow usually describes a predefined procedure which a clerk has to follow, or placement of a purchase order etc. In contrast, processes frequently require changes in modern business and emerging computing domains due to evolution in technical infrastructure, organizational policies, and runtime scenarios etc. As a result, it is necessary and unavoidable to enable workflows to consider context changes and adaptation in processes.

Now-a-days, more and more smart environments like smart home, factory, malls, hospitals etc. are being developed which ensure a high degree of availability of context information used by the IT infrastructure. These smart or ubiquitous computing environments realize the idea of Mark Weiser (1991) through seamless integration of technologies in user’s life by augmenting physical spaces with large number of sensors, devices, and services. However, smart applications require coordination of services for effective interaction. A workflow may be used to specify the potential execution order for a collection of services or reusable workflows as appropriate to the situation. There are several advantages of using workflows in smart space applications. They inherently model services or activities by allowing breaking of complex tasks into simpler subtasks. They allow flexibility in process definition without disturbing the individual service or the application. Moreover, they also improve scalability and reusability by allowing various sub-workflows to become part of a bigger workflow etc.

Although research on dynamic, flexible, and adaptable workflows have been around for some time to address the issues of workflow change management, research work is in progress for a new breed of workflows referred to as context-aware or smart workflows. Context-aware workflows, as the name implies, revolves around the relevant situation information i.e. the context to proceed execution within an instance of the workflow and on the fly adaptation as appropriate to that situation. Taking cue from existing research work, a context-aware workflow should provide dynamism to handle evolutionary changes, flexibility to cater for structural changes, and adaptability to provide a solution for ad-hoc or momentary changes. What really distinguishes the context-aware workflows from contemporary flexible and adaptable workflows is the introduction of a context layer that comprises of methods for explicit context representation, integration, and use.

In this thesis, the significance of using workflow technology in context-aware applications is highlighted so that users may benefit from advantages bring forward by such integration of technologies. Building workflow applications that are context-aware
dictate several design challenges to cater for highly dynamic environment and changing user requirements. Such challenges relate to collecting, modeling, storing, and using the contextual information; context-aware workflow modeling, context integration, and mechanisms for dynamic adaptation in workflow enactment. A framework for context-aware workflow design and management is presented in our thesis. It highlights the components of a workflow system that requires modifications and extensions in their traditional form for use in context-sensitive and dynamic situations. Although this framework presents components like context acquisition, storage, and management, it assumes them to be readily available to the workflow system. It is observed that the main challenge is to alter the process modeling concepts capturing the essence of context-awareness and adaptations without unnecessarily making the process modeling complex or less usable (Modafferi et al. 2005).

A process which is represented by a workflow model/schema in our case is, in fact, an abstract workflow model. The workflow is modeled such that simplest possible orchestrations of activities/services are done to realize process goals. The constituent activities of this model are specially designed context-aware activities which are in fact abstract activities having a place holder where a concrete contextually suitable activity is bounded at runtime. This approach makes the workflow model much simpler, flexible, adaptable, extensible, and reusable. To realize this design mechanism and execution of context-aware workflow models, a context-activity architecture is proposed in our thesis. This context-activity architecture describes the integration of the context model within the workflow system, use of relevant context, and dynamic selection and binding of suitable activity or workflow as the case may be. Apart from the workflow schema modification, explicit representation of context model is proposed again making the system more flexible and usable. This explicit representation and later integration of this context model in workflow system allows developers to model relevant information in any of the many context modeling techniques as suitable for the workflow system complexity and domain expert's expertise. As a deliverable, a standalone context aware workflow designer (CAWD) is developed that incorporates the context-activity architecture, allowing for integration of context models, verification of workflow models and a complete package of context-aware workflow that can be easily hosted in any application. A couple of examples elaborate handling of scenarios that may be experienced by a context-aware workflow application.

1.1 Motivation
The motivation of this research is the demand of context-aware applications that are on the rise in emerging domains like pervasive and mobile computing, and business as well. Moreover, efforts are underway to incorporate workflows in such context-aware system because of the various inherent properties of workflow technology suitable for providing flexibility in these applications. There exist a huge number of workflow developers, managers, and users present in the Information System industry today that directly or indirectly may take advantage of research in context-aware workflow design and management.

Based on experiences from practice, Reijers (2006) provides a brief discussion about the fact that workflow technology is less successful in delivering the intended flexibility it promised. Some critics argued that Workflow Management Systems (WFMS) make business process execution too rigid to allow any intervention from users to accommodate any breakdown or evolution. This rigidity, in the eyes of many observers,
is due to the use of formal workflow models, strict coupling between modeling and execution, and lack of support to handle dynamic and evolutionary changes (Sørensen et al. 2006). It is obvious that modifications in workflow technology are necessary to incorporate change management (Aalst and Jablonski 2000). However, it is felt along with a few of our contemporary researchers (Modafferi et al. 2005; Ardissono et al. 2006; Wieland et al. 2007; Fröhlich and Meier 2009) that by introducing context-aware infrastructure in the workflow technology, the use of workflows could be extended to other scientific and public domains than business only.

In order to gain firsthand knowledge of requirements and recommendations from the workflow user community in the context of our research, a survey is conducted locally involving developers and users from a variety of domains like banks, energy, production, hospital, telecom, software etc. The survey sample was moderate i.e. 30 but was carefully selected to include majority of areas mentioned above that have workflow users and developers. An assessment was made as to how much these users require flexibility, adaptability and context-awareness in their workflow systems. Interesting results were found. Some users do not appreciate the idea of an intelligent workflow system and preferred to remain loyal to the traditional rigid workflow management systems that precisely do what a model has initially described for the process. If any exception or change is needed to be incorporated in a running instance, either it is terminated to be started afresh or require manual interventions for further processing. But a large percentage of workflow managers liked the idea of context-aware, flexible, and adaptable workflow models that also have the ability of evolution and reuse.

This survey of workflow products is carried out with a small sample of 30 interviews of workflow users in business, production and development domain. Although context-aware workflows are more suitable for applications in pervasive and smart space environments, which is unfortunately lacked in our country. The survey, however, projects inclination and demand of workflow users for workflow products that are more flexible, adaptable, and context-aware. Figure 1-1 (a) shows the understanding of people about the degree of flexibility that their workflow products offer. Interestingly there is a mix of observations registered for same workflow product by different users and are categorized as flexible, inflexible or not known. These deviations or discrepancy in observations is a result of a mix of usage and expertise level in these products. In effect, modern workflow products provide a certain level of change management but not to the satisfaction of user or to the degree that they could be used in highly dynamic environments like pervasive and mobile computing. Moreover, Figure 1-1 (b) shows the percentage of workflow users interested in using workflow products which are more flexible in design and adaptable in execution.

Another motivation of using workflows is to improve flexibility and usability in applications for user's interaction with the environment such as helping visitors in public spaces like malls, museums, supermarkets and hospitals (Ranganathan and McFaddin 2004). Weiland et al advocates the use of context-aware workflows in smart factory to enhance productivity and reduce time and cost (Wieland, Nicklas, and Leymann 2008).
The research on context-aware workflows is still wide open as the contemporary research is still academic and usually on the pretext of a certain scenario or domain. In our research, a generic framework for developing context-aware workflow applications is presented that not only supports flexibility and adaptability but also process evolution with a high degree of usability.

1.2 Research Problem Statement

The problem statement of our research may be summarized as:

To address the issues of variability, dynamism, flexibility, and evolution associated with context-aware applications, an explicit process modeling and management technique (i.e. Workflow technology) could be effectively used with necessary extensions and modifications, to compensate for application’s dynamic requirements achieving desired goals along with improved usability and maintainability.

1.3 Research Objectives

The key research objectives are:

1. Study and review of context-aware workflow research and workflow change management to identify the issues involved in adopting workflow technology for context-aware applications.

2. Development of a framework for designing context-aware workflows by incorporating the required extensions and modifications in workflow systems.
1.4 Research Methodology
The research is carried out using the following methodology:

- Study of existing approaches in workflow management systems especially with respect to handling of dynamic and evolutionary situations.
- Study of Workflow Modeling and Enactment standards and practices
- Identify the issues and challenges in achieving research objectives by comparing existing research work and workflow products.
- Justify the need of a new framework for context-aware applications using workflows integrated with contextual adaptation infrastructure.
- Find the ways and means to integrate contextual information into the workflow definition for seamless execution.
- Present a generic framework for developing and managing context-aware workflows emphasizing the requirements for context-aware workflow design and enactment to remain within the scope of the thesis.
- Realization of the proposed framework to analyze its practicality and develop a context-aware workflow designer tool as a research deliverable.
- Develop simulated test applications and case studies
- Validate the working of extensions and modification made in workflows as presented through proposed framework

1.5 Contributions
The main contributions of this research are listed below:

1. Identification of key issues and challenges in developing context-aware workflows by highlighting the gray areas in contemporary research work.
3. Realization of proposed context-activity architecture in Windows Workflow foundation, addressing a large number of workflow users using Microsoft workflow products.
4. Provide a holistic approach for handling derived and sensed contexts suitable for handling both smart environments and evolutionary traditional processes.
5. A workflow designer tool with features that enable the design of context-aware workflows by allowing mechanisms for context based adaptations.
1.6 Thesis Organization
The thesis is organized as follows:

- **Chapter 2** provides a background of interrelated technologies essentially from their conceptual viewpoint that are necessary and relevant for the understanding of the rest of the thesis.

- **Chapter 3** provides a comparative survey of existing work in context-aware workflow design and management. It further discusses the issues in making workflows context-aware as derived from the related work and augmented by our understanding.

- **Chapter 4** initially discusses the principles upon which our proposed framework for context-aware design and management stands. The framework is presented and its components are described in detail.

- **Chapter 5** describes the underlying context-activity architecture which elaborates the integrated working of framework components. The operations for runtime adaptation and evolutionary changes are discussed in a formal way with the help of simulated testing.

- **Chapter 6** provides the implementation of proposed context-aware workflow architecture in the form of technical and development details of our context-aware workflow designer (CAWD) tool.

- **Chapter 7** discusses a couple of example scenarios showing the use and benefits of designing context-aware workflows through CAWD to handle both evolutionary and momentary changes. The example scenarios are followed by results and discussion of our research work.

- **Chapter 8** concludes the thesis and outlines the future directions to carry forward this research.

1.7 Summary
This chapter introduces the problem area, its background, and justification of the research. It further describes the motivation behind the research which is obtained through a survey of industry and existing research work. The key research objectives are defined and the adopted methodology of research is highlighted. In the end, the chapter outlines the structure of the remaining parts of the thesis.
Chapter 2

Background

2.1 Introduction
Before progressing towards the survey of existing work and our proposed research work in context-aware workflow design and management, it is better to understand the concepts and technical details of workflow technology and context-aware computing separately. In this chapter, after describing these technologies, it is elaborated that how these are merged together to develop context-aware applications for highly dynamic and context-sensitive environments.

2.2 Workflow Technology
Workflow is a core technology that is rigorously used by business organizations to support automation of their business processes. Workflow management systems (WFMS) provide automation through assistance in the specification, modeling, enactment and monitoring of structured work processes in dynamic and heterogeneous organizational and technical environments (Leymann and Roller 2000), (Georgakopoulos, Hornick, and Sheth 1995), (Aalst and Hee 2002). This automation of business processes brings improved efficiency, better process control and improved customer service in an organization (e-Workflow.org).

Although workflow technology is not new to the business community, the shape it now has and the facilities it offers have changed drastically from its earlier versions. Organizations and systems having structured and repetitive processes involving several tasks, persons and resources demanded the automation of such processes. This automation benefit organization through higher throughput, optimized results, minimized errors and reduced cost in terms of money and time (Baldauf, Dustdar, and Rosenberg 2007). Earlier forms of automation like document management, office automation, imaging processing etc. provide little what was desired, but nevertheless
laid the foundation of modern workflow management systems e.g. “Lotus Notes” was quite used in earlier days to achieve automation. The modern workflow systems now much effectively automate processes by defining, controlling, coordinating and monitoring the flow of information artifacts among different stakeholders of processes. Many organizations turned towards the workflow management systems to automate not only their business processes but also their human resource, finance and administrative processes, etc. In this continuum, many software vendors like IBM, HP, Oracle and Microsoft joined the bandwagon and produced standalone and integrated workflow products. In order to produce harmony and interoperability among various workflow products, Workflow Management Coalition (WfMC) in 1995 introduced a workflow reference model (Hollingsworth 1995) explaining the various components and interfaces of a typical workflow management system.

Strictly speaking in the context of business organizations, rapidly changing market conditions and increasing competition usually lead companies to review and investigate the efficiency of their process structures. The commitment for Total Quality Management leads to the adoption of various process-related management practices like Process Improvement, Business Process Innovation, and Business Process Redesign. Each of these approaches observed and emphasized the enabling role of information technology for restructuring of organizations. Consequently enterprises that engaged in these activities sought adequate information system support for management of their processes. Workflow management technology was the answer to cope with this exact problem i.e. the coordination of activities along a common process model without necessitating the automation of the activities themselves (Becker, zur Muehlen, and Gille 2002) (Leymann and Roller 2000).

If the trends in software and information systems development is traced back, it is observed that there is a gradual shift towards separation of concerns among various components (Aalst and Hee 2002). At very early stages, information system comprised applications that ran directly on operating systems providing a simple command line interface or custom-made graphical interface with its own databases and definitions. Then the need arises to separately store and manage data so that it remains permanently available outside the application and may be used by different applications; leading to the evolution of database management systems (DBMS). The trend then changed to extract the user interface from the application program resulting in ease of user interface design and also brings consistency in different user interfaces. At that time user interfaces were designed as per likeness or style of each designer, also taking up most of development time and ironically demands increase in learning for users. Since then the data management and user interfacing were largely disappeared from the applications themselves, and much of the software is devoted to procedures or case handling. In order to provide further ease and flexibility in information system design, processes are trending to be maintained and developed outside the application. This resulted in the evolution of workflow management systems (WfMS).

Roughly describing, a Workflow Management System (WfMS) is to a business process as a Database Management System (DBMS) to data. Just like a DBMS separates the data from the application and relieves developers of the need to manage physical aspects of data, a WfMS separates the process logic from the application and relieves developers of the need to manage the flow of data and control between interrelated activities and resources (see Figure 2-1). This separation or explicit representation of
processes was deemed important so that current business organizations may be able to cope with the ever changing processes, technologies, rules and regulations etc. The separation of concerns between process and application enables the process automation to be more flexible contrary to hard coding the process logic within the application. This allows application programs to act as independent computational units and greatly simplifies the task of enterprise application integration (EAI). It is due to these inherent properties of workflows i.e. explicit representation of processes outside the application and synchronized coordination of interrelated activities few other contemporary researchers have also chosen the workflow technology to build applications that are flexible and context-aware.

Figure 2-1 Role of workflow automation (Leymann and Roller 2000)

Almost all the workflow management systems, in the conformance of the workflow reference model proposed by WfMC, dissect them in two basic components: Build time functions and Runtime functions (see Figure 2-2). Build-time functions are concerned with the design and specification of processes i.e. workflow modeling. These functionalities enable analysts and administrators to define processes and activities, analyze and simulate them, and assign them to people. The Runtime functions are concerned with management of executing processes, resources and applications. It consists of the execution interface seen by end-users and the workflow engine assisting in coordinating and executing these processes and activities.

Although workflow technology is described comprehensively by WfMC, it has been interpreted in a variety of ways by vendors and developers. Ranging from its inference in large systems like ERP and EAI to hardcoded functionalities in application, people of varying views may be found and may treat the workflow systems based upon the complexity of processes and the infrastructure they hold. In essence, a workflow may be
categorized as applications that are built and executed as a coordinated set of interrelated activities, with an explicit definition of the executing process outside the application. This inherent property of workflow enables this technology to be used in domains other than business and now is emerging as a programming paradigm where a process is designed and programmed simultaneously such that the process definition remains outside the application core.

Figure 2-2: Workflow system characteristics (Hollingsworth 1995)

2.2.1 Workflow Terminologies
Although workflow technology has been used and researched for several years now, workflow concepts and terminologies are varyingly defined. However, a widely agreed upon glossary of workflow terms were proposed by WfMC. In later paragraphs, various workflows related terms are defined for consistent reading throughout the thesis, and illustrate their relationships as illustrated by WfMC in Figure 2-3.

- **Workflow**: Formally, a workflow is defined by WfMC 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* (Hollingsworth 1995).

  Alternatively, a workflow may be defined as a representation of ordered, interdependent activities, resources and information required to complete a business process (Mangan and Sadiq 2003).
• **Workflow Model (also referred as Process Definition):** It is the representation or specification of processes in a form suitable for automated manipulation and enactment by a workflow management system. This representation or definition contains the start and end conditions of a process, activities within a process, control and data flow among these activities, and references for allocated resources and participants.

Since the workflow model or the workflow schema is related to the design of workflows, this chapter later emphasized on workflow modeling, workflow languages and the formal approaches in detail.

• **Workflow Activity (also referred as Task):** It is

  A description of a piece of work that forms one logical step within a process (Hollingsworth 1995).

An activity is typically the smallest unit of work which is scheduled by a workflow engine during process enactment. An activity may be automated or manual that requires machine or human respectively for its execution. Note that a manual activity does not form the part of an automated workflow and include only to complete a process model.

• **Workflow Instance (also referred as Process Instance):** are the specific occurrence / execution of a process as defined by workflow model. Each instance of a workflow may run simultaneously as dictated by the workflow model with different data associated.

• **Workflow Management System:**

  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 (Hollingsworth 1995).

More simply a workflow management system (WfMS) is the software that provides the means for definition, execution, control and monitoring of workflows (Aalst and Hee 2002).

• **Work Items:**

  The representation of the work to be processed (by a workflow participant) in the context of an activity within a process instance (Hollingsworth 1995).

An activity during execution may generate one or more work items which constitute the task to be undertaken by the participant within that activity. In case of autonomous activities, the task is processed by an invoked application.
• **Invoked Applications:**

An invoked application is a workflow application that is invoked by the workflow management system to automate an activity, fully or in part, or to support a workflow participant in processing a work item (Hollingsworth 1995).

• **Workflow Engine:** also known as enactment service is defined as

A software service that provides the runtime execution environment for a process instance (Hollingsworth 1995).

• **Workflow Application:** It is the software program that interacts with a workflow engine, providing the necessary support and functionalities for particular activities. They may be client applications or invoked applications that request or provide service from or to the workflow enactment service.

• **Workflow Persistence:** refers to the act of storing workflow states and relevant data for long running workflows for later retrieval and execution.

![Figure 2-3: Relationship between workflow terms (WFMC 1999)](image-url)
2.2.2 Workflow Modeling, Instances and Languages
As envisaged from earlier discussion, workflow management primarily aims at modeling and controlling the execution of (business) application processes as per predefined process specification i.e. the workflow model or alternatively workflow schema. Conceptually, a workflow model may be treated as the formalization of workflow relevant aspects of a business or technical process. This formal and well-defined representation of processes enables them for automation and are expressed in workflow languages (Weske, Vossen, and Puhlmann 2005). Later in this section, more discussion about these languages is made, but first workflow models and workflow instances are discussed in detail by providing a conceptual and formal view of them. Please note that the discussion will be restricted to above and workflow management aspects like workflow enactment etc. would not be discussed as the background of these components would increase the scope of our discussion to manifolds.

Matthias Weske (2005) describes the Workflow Application Development Process in four steps as shown in Figure 2-4. It starts with the gathering of information relevant for the process under investigation. The techniques used during this step are usually informal and non-technical focusing on capturing the process details. With the help of available documentation, interviews and desired goals, an informal shape of a process is formed. The next phase involves process modeling where a (business) process model is specified using the gathered information. This process model serves as a basic document for information system experts and business analyst to understand the process mechanics in an easy-to-read notation e.g., Business Process Modeling Notation (BPMN), without diving into the technical details of automation overheads. In the next step, the business process model is transformed to a workflow model which enhances it with information needed for automation and controlled execution in a workflow management system. In this step formal workflow languages are used to model the workflows and hence the models are translated into the constructs of the workflow language used. Lastly, one or more instances of the verified workflow model are generated as per the requirement of workflow application.

![Figure 2-4: Workflow application development process (Weske, Vossen, and Puhlmann 2005)](image-url)
Formally, a workflow may be defined as,

\[ \text{A workflow } W \text{ is a directed graph } (T,D), \text{ where } T \text{ is the set of nodes, denotes the tasks } t_1, t_2, \ldots, t_n \text{ in } W, \text{ and } D, \text{ the set of edges, denote the inter-task dependencies } t_i \xrightarrow{x} t_j \text{ such that } t_i, t_j, \in T \text{ and } x \text{ is the type of dependency} \ (\text{Atluri and Chun 2003}). \]

The workflow models are representations of processes used by workflow management systems for controlling automated execution of workflows. Theses workflow models are specified by workflow languages which provide constructs for specifying a syntactically and semantically well-defined process using formalisms like graph theory or more recently petri-nets. The workflow languages used to model workflows follow a process definition meta-model, that comprises a set of modeling concepts to capture various aspects of workflows. Figure 2-5 shows a relationship between abstraction levels of workflows.

![Figure 2-5: Process definition meta-model](image)

As discussed above in the workflow development process, a workflow language enters into the picture of process automation when the desired process is conceived, verified, optimized and is ready for introduction into a workflow management system. Workflow languages are different from other general-purpose programming languages in a variety of ways. For example, they are highly domain specific, relaxed for computational completeness and more closely related to human-computer interaction. However, workflow languages are precisely defined in terms of syntax and semantics. Generally, low level constructs as in general-purpose languages are missing in workflow languages but they support constructs to integrate and invoke external applications and synchronize interactions. Some of the examples of workflow languages are XML Process Definition Language (XPDL) proposed by WfMC, Yet Another Workflow
Language (YAWL), Business Process Execution Language (BPEL) (OASIS 2007) etc. The XPDL may be described as an interchange format and usually only represents processes so that they may carry over the same process shape when used by another vendor’s product. YAWL evolves from academic research and focuses mainly on workflow patterns (Aalst et al. 2003) and formalism in petri-nets. The relatively new approach of service oriented architecture and its current form as web services influence the workflow technology and facilitated a major shift by composing workflows using inter-organizational services and applications. This leads to the rise of BPEL which deals with the orchestration of web services for specifying business processes. A new workflow based application development platform is introduced by Microsoft Windows as Windows Workflow Foundation (WF in short) that enables developers to model processes in its native language XAML.

Since BPEL lacks a graphic notation, a well agreed upon Business Process Modeling Notation (BPMN) is introduced that proposes language constructs that facilitate the modeling of workflows at a higher level. Even processes modeled in other graphical workflow languages, are now represented in BPMN to achieve consistency in understanding at higher level. Moreover, a number of BPMN constructs can be directly mapped to workflow languages especially BPEL.

Although developed for business applications, workflow technology has now crossed boundaries to be used in domains like grid and ubiquitous computing. In grid environments, workflows are used to orchestrate activities by coordinating distributed resources for performing scientific experiments (Fox and Gannon 2006), (Yu and Buyya 2005). Mobile and ubiquitous applications frequently use workflows for service orchestration and context-awareness (Ranganathan and McFaddin 2004), (Fröhlich and Meier 2009, Ardissono et al. 2006), (Wieland, Kaczmarczyk, and Nicklas 2008).

2.3 Context and Context-awareness

2.3.1 Preamble to Context

The word “context” is a very vague term widely used in a multitude of domains. It is one of those terms which are frequently used but rarely defined, and when defined, defined so multifariously. In plain English, context is defined by the Oxford Dictionary as

The circumstances that form the setting for an event, statement, or idea, and in terms of which it can be fully understood.

One may observe the richness of interaction and efficiency that the knowledge of context brings in our communications and actions, but the same is not always the case when interaction is made with computer systems simply because they either do not take context into consideration at all or do it inefficiently. In order to utilize context information in computer applications, several challenges have to be addressed like specification, acquisition, dissemination, integration and use of context information.

With the advances in wireless communication technologies, miniaturized electronics, sensors technology, web services and global software architectures etc., the availability of context information is very much increased now-a-days. Moreover, the widespread use of mobile technology creates the opportunity and necessitates the development of context-aware applications. These context-aware applications enable tasks to be
performed in an autonomous and flexible manner reducing the need of human interventions. Around two decades ago, Mark Weiser (1991) presented the idea of seamless integration of technologies in our lives which destined to revolutionize the way people interact with computer systems and effectively hiding them from the user. Its aim was to provide better services as required per user situation without even bothering the user of the multiplicity of devices and technologies it may have used without the knowledge of using them. In his own words,

The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.

Now this technology is recognized by different yet similar names like ubiquitous computing, pervasive computing, invisible computing or smart computing. Context-awareness is an inherent feature of all ubiquitous applications that imposes adaptations to the systems as per changes in context information and user demands.

Roughly speaking, there are two schools of thoughts in terms of specifying and modeling contexts. The first one recognizes context as the information acquired through sensors placed in a smart space only and potentially focuses on applications which are related to ubiquitous applications. While the other considers context as much complex entity that covers information not only coming from sensors but also from sources like policies, back office information or any information that can be compositely and logically inferred. Using the viewpoint of the first school of thought, context-aware application development is rather easier than the other one. However, in any case, it is needed to understand how to specify context information, generate context models, integrate and use context to support context-aware adaptations for applications.

2.3.2 Context Definitions
As stated earlier, context is defined in a variety of ways by researchers. Many of them defined context based upon their own perceptions and within the scope of the intended context-aware applications. In literature, many researchers trace the definitions of context as used over time (Sheng, Yu, and Dustdar 2010), (Baldauf, Dustdar, and Rosenberg 2007), (Salber, Dey, and Abowd 1999). Firstly, context is defined by Schilit and Theimer in 1994 (Schilit and Theimer 1994) as “location, identities of nearby people, objects and changes to those objects”. On similar lines Ryan et al. in 1997 (Ryan, Pascoe, and Morse 1997) referred to context as “the user’s location, environment, identity and time”. Schmidt et al. in 1999 (Schmidt, Biegl, and Gellersen 1999) opines that a context describes a situation and the environment a device or user is in and is identified by a unique name with relevant features and unique range of values. Chen and Kotz in 2000 (Chen and Kotz 2000) defined context as the set of environmental states and settings that determines an application’s behavior based on an event interesting to the user. However, A.K. Dey in 2000 provides a more generic definition of context and till date is the most referenced definition (Dey and Abowd 2000), which says that

Any information that can be used to characterize the situation of entities (i.e., whether a person, place or object) that is considered relevant to the interaction between a user and an application, including the user and the application themselves.
After Dey, there are various attempts to describe context but are limited to the specific research purpose or application domain. Henricksen (2003) is of view that context is an enabling mechanism for applications to perform tasks on behalf of users. Therefore, context should be related to tasks rather than the interaction between users and applications. Hence, they defined context as the set of circumstances surrounding a task that are potentially relevant to its completion. Gray et al. call context as spatio-temporal information concerning a user (Gray and Salber 2001). Modaferri et al (2005), Wie et al (2006) and Gloss (2007) relates context to any information needed to support personalized services such as user’s mobile device and its capabilities, and the networks used and their properties. Another viewpoint of context is presented by Wieland et al. as the information relevant to a system gathered by physical sensors and later used in generating higher level of abstractions (Wieland et al. 2007).

There are several other interpretations of context in literature which are not discussed here. While context may never be precisely or exhaustively defined; context model, which captures the relevant information structure is of more significance than the context itself. The applications that are context aware may not be harmed by this impreciseness and may utilize the context information as blue printed in a context model (again developed by a variety of people in a variety of ways) for use.

The characteristics of context may be summarized through inference from a massive set of interpretations as:

- Context influences applications and may affect their processing and presentation in general.
- Context revolves round user and may contain any real world information significant for adaptation as per user needs.
- Context acts as a filter and shields user and system from a plethora of information by filtering only the relevant information.
- Context provides a snapshot or instantaneous information relevant for the timely action for realizing optimized goals.

### 2.3.3 General Classification of Context

There are many ways context is classified in literature again depending upon the approach, objectives and requirements of application. Table 2-1 shows the various attempts made by different researchers to classify the context in general (Sheng, Yu, and Dustdar 2010), (Baldauf, Dustdar, and Rosenberg 2007).
<table>
<thead>
<tr>
<th>No.</th>
<th>Classified by</th>
<th>Context Categorization</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Prekop et al. (Prekop and Burnett 2003)</td>
<td>External</td>
<td>Categorization is based on different context dimensions. External context includes location, proximity, temperature, time, light etc., whereas internal context includes user events, emotional and physical state etc.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Internal</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Gray et al. (Gray and Salber 2001)</td>
<td>Sensed</td>
<td>Context is categorized as those which can be sensed by means of sensors or explicitly provided as user inputs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Explicit</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Goslar et al. (Goslar, Buchholz, and Schill 2003)</td>
<td>Hard</td>
<td>Context is categorized as information acquired through hardware devices e.g., sensors and also as software sensors using the application database</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Soft</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Wei et al. (Wei et al. 2006)</td>
<td>Static</td>
<td>Categorization is based on how frequently context is changed. Less frequently or once supplied information are treated as static while fast changing information is treated as dynamic context</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dynamic</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Khedr et al. (Khedr and Karmouch 2005)</td>
<td>Simple</td>
<td>Context may either be simple e.g., location or composite i.e., inferred through a combination of simpler contexts</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Composite</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Coppola et al. (Coppola et al. 2005)</td>
<td>Concrete</td>
<td>Khedr et al. view context as concrete i.e. primitive or abstract i.e. inferred from primitive. Coppola also classified these two context as public or private depending upon information access</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Abstract</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Hofer et al. (Hofer et al. 2002)</td>
<td>Physical</td>
<td>Context is categorized as those which can be accessed through physical devices e.g., hardware sensors or logical which are mostly specified by users or captured by monitoring user actions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Logical</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Razzaque et al. (Razzaque, Dobson, and Nixon 2005)</td>
<td>Primary</td>
<td>Context is categorized as those entities which are either frequently used (primary) and have more impact or secondary which are rarely used and have little impact on the context aware system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Secondary</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Karen (Henricksen 2003)</td>
<td>User Defined</td>
<td>Context is categorized as user-defined which is the information provided by user e.g., profile or sense which is the information acquire through sensors e.g., location or derived which is the information determined by combining multiple user-defined and sensed information</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sensed</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Derived</td>
<td></td>
</tr>
</tbody>
</table>
It may be observed that there are different yet similar approaches to categorize context in general. Everyone is of the opinion that the information related to user and its environment under consideration is of primary importance whether sensed or derived. Based upon the nature of context information, it may either be simple, composite, static or dynamic. Context is related to both space and time and their significance depend upon the nature of application in which they are used. Most of the context-aware application usually exploit sensed context like location, time etc. It is, however, noticed that much useful context-aware applications are those that uses both sensed and non-sensed information along with the derived information. The non-sensed information is usually supplied by user directly as inputs or indirectly as profiles or activity monitoring. The derived context makes use of sensed and non-sensed information and infers a higher level of information from them. For example, sensed location of a user from a GPS and his/her profile may be used to derive information about his/her action e.g., whether in a meeting at office or having a lunch at a restaurant.

Another way of viewing or classifying context was presented by Dey and Abowd that distinguished context within three entities: Places, People and Things (Dey and Abowd 2001). These entities are further described by various attributes which can be classified into four categories: identity, location, status and time.

It must be mentioned here that context information is highly heterogeneous in nature and its quality may gets compromised due to impreciseness in sensor values, failure of devices, and frequency of updates etc.

2.3.4 Context Awareness

A system becomes context-aware when it uses the context to adapt to the changed situation by providing relevant and useful services. Context-awareness has been in use as a technique for developing pervasive and mobile applications that are flexible, adaptable and capable of acting autonomously without the intervention of user. Example of such systems are intelligent navigation, traffic management, tour guides, and shopping guides etc.; smart spaces like smart office, smart hospital, and smart agriculture etc. To establish the understanding of context-awareness, a project of smart university that utilizes the RFID tags and profile data as context for execution is developed that highlighted issues relevant to context acquisition hardware (Aqeel-ur-Rehman, Abbasi, and Shaikh 2008).

The relatively lesser use of context-awareness in enterprise applications is due to the complexities in modeling and using context information which are not sensed but rather logically or computationally derived. But context-aware applications are not limited to pervasive and mobile applications only and may be found in industrial and business applications also.

Formally, context-awareness is defined by many but the most referenced definition of context-awareness is presented by A. K. Dey as:

A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user’s task (Dey 2001).
The inherent complexity and challenges of context-aware applications require the support of adequate software engineering methods. The maintainability and evolvability of context-aware systems demand that the functionalities of such systems should not be entwined with the specification and evaluation of context information which is often subject to change over time. Therefore, context-aware applications should support good context modeling explicit to applications, seamless integration and context reasoning techniques. Also context-aware applications need to address the key issues of context acquisition, management and storage.

2.3.5 Context Modeling and Reasoning
Fundamental to every context-aware system is a context model. Context modeling is a means for categorizing, naming, storing, retrieving, reasoning with, and binding of contextual information to tasks, thus making the system context-aware (Mitra and David 2008). Alternatively, context modeling is the specification of all entities and relations between these entities which are needed to describe the context as a whole e.g., information on location, time, user and its planned activity, and computational entities. A good context information modeling formalism reduces the complexity of context-aware applications and improves their maintainability and evolvability (Bettini et al. 2010).

Context reasoning is the derivation of high level context suitable for a process from low-level context. As illustrated in Figure 2-6, a context model is used to retrieve a relevant high-level context from lower level contexts. The context model represents a university and the detailed structure and relationships among different concepts of universities. For example, in a university classes are conducted at particular times, university staff and students have RFID cards etc. The relationships among students, teachers, courses and buildings are maintained in the context model. At a particular instant, one may get the low-level context information of ID, time and position through various sensors. The values acquired through this low-level context are transformed to a relevant context of position within the university, the time of a particular lecture and the identification of card bearer. As the card bearer is “Abu Zafar” and the time shows lecture three is being conducted, a higher level context is formed as “Abu Zafar is taking class with Dr. Zubair” because the context model shows that lecture number 3 is HCI whose instructor is Dr. Zubair and Abu Zafar is his teaching assistant.

Developing flexible and useable context models that cover a wide range of possible contexts is a challenging task. Strang et al. classified context modeling approaches based on scheme of data structures used to exchange contextual information as: Key-Value models, Markup scheme models, Graphical models, Object oriented models, Logic based models, Ontology based models (Strang and Papien 2004). Based upon the complexity of the application domain and expertise of the developer, simpler models like key-value or detailed and rich models like ontology could be used. However, Bettini proposed the use of relational database and hybrid models to enhance flexibility and reuse(Bettini et al. 2010).
2.3.6 Context Acquisition and Management

The context data acquisition also plays a very important role in the design of context-aware systems as it predefines the architectural style of the system at least to some extent (Baldauf, Dustdar, and Rosenberg 2007). Now-a-days, everybody is inconspicuously exposed to a bundle of information thanks to multiple information services, mobile devices, communication infrastructure and miniaturized sensors. In case of a smart space e.g. smart house, smart factory or smart hospital etc., one may encounter multiple types of devices and sensors that sense the user and environment context making the system adapt appropriately to the situation. These sensors may be noticeable or even so small or ubiquitous that are termed as “dust” (Satyanaraynan 2003). The maturity of sensor networks eases the gathering of context data that is distributed over a large space. In (Aqeel-ur-Rehman et al. 2011), it is illustrated that how sensor and actuator networks are employed in context data acquisition for the development of context-aware applications in agriculture.

A context toolkit approach proposed by Dey et al. provides a programming support for context data acquisition by defining three abstract components namely widgets, interpreters and aggregators (Dey and Abowd 2001). The major component of the toolkit i.e. widgets provides wrappers for sensors. Widgets hide low-level detail of sensing and ease application development due to their reusability. Furthermore, interpreters and aggregators raise the level of abstraction of context information to a more usable form other than raw data and combines different context information to provide relevant single information respectively. This tightly coupled approach though efficient lacks profiled information support, handling of uncertain context information and robustness to component failures.

In a more general way, Chen presented three different approaches on how to acquire and manage contextual information (Chen 2004).

1. Direct Sensor Access: This context data acquisition technique applies to those applications that are built on systems that have sensors locally built-in. These
applications may gather desired information directly from these sensors by either push or pull mechanism and do not require any additional layer of acquiring and preprocessing sensor data. This is a very tightly coupled approach and is used in rare cases. An example may be an application running on a Smartphone sensing location information through a built-in GPS receiver.

2. **Middleware Infrastructure**: The middleware based approach hides low-level sensing details to applications. In comparison to direct sensor access this technique introduces flexibility and extensibility as the client code needs not to be changed for any sensing hardware change as it gets handled at middleware layer.

3. **Context Server**: In this approach data collected from distributed sensors and other sources are stored in a server enabling the shared use of context information for multiple clients.

A context management system should not only be capable of context data acquisition by providing interfaces for sensors but may also perform some preprocessing, aggregation and fusion before storing them in a database.

### 2.4 Context-aware Workflows

In previous two sections of this chapter, discussions are made on workflow technology and context-awareness essentially from their conceptual viewpoint. Context-aware application requirements are on the rise due to advanced communication, electronics and software infrastructure now available. These context-aware applications usually centered round situation information about user demands, systems and environment and require autonomous adaptations by providing the relevant functionalities or services as appropriate to the situation. Workflow technology provides facilities to compose services to achieve a desired goal and that so also by specifying the process logic explicit to the application. This makes workflow technology an ideal candidate to represent processes that demands the requirements of context awareness.

Although processes modeled as workflows, through its separation of process from application, provide flexibility to the systems, it remains rigid once a workflow model is defined and executed. The limitation of the contemporary workflow technology prevents it from adequately supporting evolutionary and on-the-fly changes demanded for fluctuating business and environmental situations. In the few cases where the inherent nature, complexity and constraints of some processes that involves policy decisions, multiple user involvement, and of course huge sum of money, processes are modeled as workflows that should remain rigid and follow a predefined flow without any alteration. However, such rigidity of workflows is treated as a serious impediment in the application of this technology in various domains. Processes always have the tendency to change over time, whether permanently or temporarily. So, workflow products are developed with the limited capabilities of providing long term structural changes or ad-hoc changes on the running instances.

The rigidity of business processes is one property that determines the suitability of workflow technology to achieve goals in a particular domain (Sadiq 1999). The degree of rigidity may be measured in terms of how much the process definition is likely to change involving change in activities, control flow, resources and policies. Earlier workflow solutions did not support dynamically changing workflow processes and were only suitable for rigid process support. In such cases, the workflow changes were
handled through bypassing the workflow management system which undermined the automation promise. Some recent versions of workflow management systems offer support for dynamism in processes but only to handle a particular instance. This evolution or change in process may be due to the result of business process reengineering, changing demands, change in rules and regulations, opting for technological advancements etc. Several approaches are proposed in research to handle change in workflows like methods for exception handling, flexible composition of workflows, ad hoc workflows, and workflow migration to newly defined process models. However, the emergence of enabling technologies like wireless communication, sensor network, high speed internet, semantic web, and service oriented architecture etc., enable workflow technology to be applied in diverse areas like pervasive and mobile computing. This diversity has varied challenges to cater changing processes.

Workflow change management is widely researched over last a decade (Sadiq 1999) (Aalst and Jablonski 2000) (Schonenberg et al. 2008) under the names of dynamic workflows, flexible workflows and adaptive workflows. However, until recently this change management or dynamic behavior of workflows was focused only to business processes taking care of business process improvements, process innovation, changing policies, exception handling etc. The advancement of technologies, however, extends the range of workflow technology to be adopted in more dynamic and changing domains. These domains e.g., pervasive computing, mobile computing and internet-based commerce are highly dynamic and context-sensitive. Based on the earlier research work in workflow change management, which do not consider context information as a major ingredient in the design and implementation of workflows, extensions and modifications may be made to integrate and use context information to develop workflows suitable for applications that are highly dynamic, flexible and context-aware.

One may argue that a context-aware workflow may be designed easily without any modifications and extensions in existing workflow models and frameworks. By modeling all possible scenarios based on relevant context information in a workflow model through the use of available workflow patterns (Aalst et al. 2003) like multiple choice, parallelism etc., an intelligent workflow model could be developed that could choose the most suitable execution path based on context information. However, this simple technique has severe limitations and inflexibility that would be a major hurdle in use for context-sensitive scenarios. First of all, it is impossible to design a workflow model that exhausts all possible situations. Second, to incorporate any change or addition in the workflow model it must be re-designed that makes a severe impact on running workflow instances. Third, such kind of design makes a workflow model complex and hard to read or modify. So, it is necessary to design a context-aware workflow such that it becomes flexible, adaptable, evolvable and autonomous.

Context must be treated as first class citizen in context-aware workflows that not only drives but influence the design and management of application and workflow design. To allow context-awareness, the way an application is designed and developed needs to be modified and the integration of context be seen as a key design issue and workflows/applications are in fact need to be designed from the start so that they become inherently context aware. The context information is recommended to be modeled explicit to the workflow specification as it makes it flexible to evolve with time. This context model may be shared among workflow definition, workflow enactment and context management services. Context acquired through sensors,
software sources or profiles get updated autonomously by transforming the low level context data into a higher level abstraction.

The most challenging design issue for context-aware workflows is the smart workflow specification that allows for context integration, context-based adaptation and flexibility for evolutionary changes. These modified workflow specifications should be compatible to workflow enactment services otherwise they also need to be modified as in the case of some research (Ardissono et al. 2012), (Wieland, Kaczmarczyk, and Nicklas 2008), (Choi et al. 2007).

To understand the reasons for adaptation in context-aware workflows, they may be categorized as customization, compensation and optimization (Erradi, Maheshwari, and Tosic 2006). A workflow may be adapted to autonomously customize through accommodating changes in context or user requirement such as need or replacement of a service, change in user environment etc. Similarly a workflow may be adapted to compensate for a failed resource or service by rolling back its execution or restarting a part of workflow to cancel out the effect of faults and failures. A workflow may also be adapted to improve the performance of workflow execution times, resolve contingencies and increased throughput.

In the next chapter, it is discussed that there are several ways in which context-aware workflows are interpreted, designed and managed. However, whatever may the case, a context-aware workflow is defined in our own words as,

\[
\text{A workflow that fulfills the additional requirements of flexibility, adaptability and autonomy by reacting to context changes, which are relevant information for the system, so that dynamic replacement, addition and deletion of activities may be done at runtime for efficient achievement of workflow goals without explicit user intervention} \quad (\text{Abbasi, Shaikh, and Ahsan 2012}).
\]

2.5 Summary

In this chapter, the academic and research background is described which was necessary for better understanding of the thesis. In fact, the conceptual viewpoint of related technologies is covered and discussion is made on only those aspects which are relevant to thesis research work. The usefulness of workflow is discussed in process modeling, its evolution, and limitations. Context and context-awareness is described emphasizing the requirement of shared context model to be used in various components of context-aware system. It is also debated that workflow characteristics of task composition and explicit process definition are very suitable to develop context-aware applications with few modifications and extensions. In the end, a formal definition of context-aware workflow is produced as per our understanding.
Chapter 3

Survey of Existing Context-aware Workflow Systems and Identification of Issues

3.1 Introduction
Computing is becoming ubiquitous and pervasive day by day thanks to the seamless integration of advanced technologies and services in our lives. Similarly business applications are now being tested in more dynamic and competitive environment. Context-awareness is now being demanded from applications to sense and adapt to situation i.e. the context in both traditional and pervasive domains. An efficient way of developing these context-aware applications is the utilization of workflow technology with these applications to model their varying dimensions. However, traditional workflow technology needs to be modified and extended so that they make use of context information within the workflow model for dynamic adaptation. In this chapter, a survey of related work is done in the area of context-aware workflow design and management that result in the identification of key issues and challenges in developing context-aware workflow applications.

Historically, the change management in workflows is well researched (Aalst and Jablonski 2000), (Casati et al. 1998), (Eder and Liebhart 1998), (Sadiq, Sadiq, and Orlowska 2001), (Golani and Gal 2006). Sadiq et al. classified the change manifestation in workflows as dynamic, adaptive and flexible (Sadiq, Sadiq, and Orlowska 2001). Dynamism is the ability of workflow processes to change when business processes evolve. This evolution of workflow i.e. the change in process definition may be a result of process reengineering, incorporation of new policies and actors, or change in infrastructure; and could be drastic enough to completely change or re-orchestrate a process. Such change in workflows poses the biggest challenge of handling active
workflow instances, which were initiated on old workflow model and need to comply with the new version. Many researchers have focused on this problem (Sadiq 1999), (Aalst and Jablonski 2000) and proposes the solution of completely flushing old instances of workflows or migrating the old instances to that level where they may corroborate with the new model. *Adaptability* is the ability of workflow processes to react to exceptional circumstances or demands, usually affecting one or few running workflow instances. Unless these demands or exceptions are captured within the workflow model, the workarounds to handle these exceptions from outside the system may result in severe consequences to process goals. *Flexibility* is the ability of workflow processes in which they are defined as partial or loosely specified models and concretely composed at runtime. The main challenge is to allow for such abstractness and mechanisms for late concrete specifications at build time and runtime also. W.M.P. van der Aalst et al., on the other hand distinguishes workflows that respond to changes as *ad-hoc* (momentary) or *evolutionary* (Aalst and Jablonski 2000). Ad-hoc changes in workflows results from the exceptions, emergencies, or case to case base situations and effect only few cases of workflow instances, whereas evolutionary changes in workflows are due to process improvements and are of structural nature affecting the process definition represented as workflow model. Along similar lines, A. Marconi et al. categorizes adaptation to change in workflow models as *short term* and *long-term* (Marconi and Pistore 2009). Short-term adaptation allows for reaction to changes in running flows, whereas long-term adaptation refers to those changes in flows that ultimately alter the structure of workflows upon which new generation of workflows are instantiated.

The research in workflow change management, under above mentioned classification, although provide support for dynamism, flexibility and adaptability in workflows, they lack integrated components necessary for making workflows to be used in context-aware applications. Essentially, the earlier research work revolves around the change management in traditional business processes where the context does not change frequently as in case of context-aware environments, so they generally took care of structural changes in the model required to compensate for evolutionary or long-term changes. However, context-aware workflows dictate modifications in workflow components along with the addition of components required for context management and adaptation to cope with the fast changing contexts. The introduction of context infrastructure within the workflow framework is what distinguishes context-aware workflows from other flexible and dynamic workflows.

The study of related work in context-aware workflow systems highlights the different approaches used by researchers for making context-aware workflow systems and revolves round the concepts addressed in the next section.

### 3.2 Characteristics of Context-aware Workflow Applications

In contemporary research literature, the key characteristics of context-aware workflow applications are mentioned but are not comprehensively defined (Sucha, Sea, and Maria 2008), (Sørensen et al. 2006). However, through literature review and study of workflow products, following key characteristics of context-aware workflows have been identified:
3.2.1 Flexibility in Workflow Representation
As illustrated in Figure 2-5, a process is represented as workflow model, specified in workflow language and is understandable by the workflow engine for execution of its instances. Any business or application logic builds with workflows is not well-suited in current form for use in context-aware applications. There are two approaches adopted by researchers to build desired flexibility and context-awareness in workflows: one is the necessary modifications and extensions in existing workflow language making it suitable for use in context-aware applications, second is the design and development of a new context-aware workflow language altogether. C-BPEL (Gherida and Mezni 2006), context4BPEL (Wieland et al. 2007), extensions in JPDL (Ardissono et al. 2012) etc. are examples of the first approach while uWDL (Han, Cho, and Choi 2005) etc. are examples of the second approach. However, it must be noted that the above are research prototypes and usually intended for use in specific scope.

Notable approaches to bring flexibility in workflow languages for context-aware adaptations are: representation of activities in abstract form that are replaced with concrete activities in workflow instances, representation of activities in hierarchical form such that activity within the scope is selected at runtime for execution, or even specifying all possible adaptations within the workflow definition making the approach least flexible.

3.2.2 Modifications in Workflow Enactment
Modifications and extensions in existing workflow representations pose requirements for workflow transformation suitable for execution in the respective workflow engine or a wrapper on the enactment service to cater for context relevant parts and making the workflow execution transparent. On the other hand, new workflow languages require new enactment services that understand the syntax and semantics of new workflow language and able to execute them keeping the basic principles of workflow enactment intact.

3.2.3 Context Information Representation
As the driving force behind every methodology of context-aware workflows is the context itself, it is an important characteristic that how context is modeled to capture the user, system and environment in a form suitable for accessing, using and extending. The context information modeling is highly dependent upon complexity of given system and may be modeled using standard methodologies as discussed in section 2.3.5.

3.2.4 Context Integration in Workflow Representation
Each context-aware activity within a workflow model depends upon single or multiple atomic or composite context entities. These entities of interest within the activity may form an intrinsic part of the activity. It makes context fetch and update easy and forms basis for the selection of concrete activities by evaluating any rule-set based on these entities only. On the other hand the associated context model with the workflow holds the necessary information about the relationships between context entities and is used with adaptation rules for selection of appropriate concrete activities. A context-aware workflow designer tool should be capable enough to provide the context-relevance space at design time and allows for binding of interested context entities within a context-aware activity with support of adaptation.
3.2.5 Support of Context Management Systems

Context-aware applications do not work without a context management system. A context management system provides support for accessing, using, storing, and acquiring context data. Depending upon the application scope and domain, these context management systems may be localized, distributed, or server-based. Context-aware applications such as mobile applications usually have this system locally at the device where the application is running. For example, sensed entities like location (through GPS receiver), network quality, and signal strength provide data in the same device so context management may be done locally. However, for applications that require back-office information, environment, and real-world data for adaptation, uses distributed or server-based context management. For example, in case of smart workflows, Wieland et al. uses NEXUS context management system which is a part of a big augmented world model (Wieland et al. 2007).

The challenge in context-aware workflow applications for integration with the context management system is that the necessary interaction and integration should be provided in the workflows rather than the application. The context management system uses the shared common context model and provides necessary abstractions to them for storing after acquiring raw data from sensors and other sources. Depending on the methodologies, data structures, and functionalities of context model, a context management system provides mechanisms for context query and management.

3.2.6 Context-based Adaptation

A flexible composition of workflow activities allows for adaptation at two levels i.e. within workflow definition and running instances. Any context-aware workflow that models a process is subject to change and evolves permanently or temporarily. Permanent changes demand the changes to be reflected in the workflow definition and require that new and if possible old instances of same process follow the composition as modeled in the changed workflow. In contrast, a temporary change is any context-driven adaptation that is reflected and influenced only in running workflow instances.

The general operations of adaptations may be categorized as selection of an appropriate activity or sub-workflows, insertion/addition of a new activity and deletion of an activity within the workflow. Workflow definition level adaptation is difficult as it requires handling of running workflow instances. Any change at workflow definition level may result in inconsistencies or even crash. Any mechanism that allows for adaptation operations dynamically and does not affect the basic / abstract workflow definition would be useful and effective for such adaptations. Runtime workflow may be incorporated by use of sensed context and predefined adaptation rules to provide the operations for adaptation.

3.2.7 Activity Binding Times

As rightfully identified by Sucha et al, a concrete activity or service must be assigned or bound to abstract activity either before or during runtime (Sucha, Sea, and Maria 2008). In the first case, all abstract context-aware activities need to become concrete before the instance is executed. Such binding of activities may be done using a workflow design tool that should be capable of fetching context updates and automatically or manually provide a mechanism for selection of appropriate concrete activities, or the activities get assigned autonomously before execution by the workflow host application itself.
However, this approach is not effective as it may fail if a service become unavailable and also any context change during execution is not handled making the workflow less reliable.

Runtime binding of concrete activities into context-aware abstract activities may be achieved through late binding. The late binding approach bounds a concrete activity or service to each abstract activity as the execution proceed to each task in the running instance. This approach requires the implicit or explicit use of adaptation rules and activity / service directory.

### 3.2.8 Rollback and Migration Strategies

Some context-aware workflow applications require frequent roll back and migration in workflows. These are advanced features and are not supported by many existing systems. In order to handle a failure or context-change, the workflow instance needs to go back in order of execution. This implies rolling back of workflow activities to a suitable level. In a closely similar case, when a process definition is modified while the instance was still running, the instance needs to be migrated to the new definition with all its internal and external data intact. Since this is a difficult operation, many approaches choose to terminate the running instance and start afresh as per new model to handle such situations.

### 3.2.9 Evolutionary Support features

To increase longevity and reusability, context-aware workflows should also support evolutionary changes. Although, strictly speaking, evolutionary changes do not fall in the case of context-aware adaptation contrary to ad-hoc changes. But the context-aware workflow design and maintenance tool should provide editing, addition and deletion of rule-sets and alternate sub-workflow collection in a way so as not to disturb the actual workflow model.

### 3.3 Existing Work in Context-aware Workflow Systems

The significant existing work related to our research area is critically analyzed and presented in following sub-sections.

#### 3.3.1 Context Aware Workflow Execution Framework

(L. Ardissono et al.)

L. Ardissono et al. present Context Aware Workflow Execution (CAWE) framework for the management of context-aware applications by coordinating web services (Ardissono et al. 2006, Ardissono et al. 2012). They use the strategy of representing workflows with one or more abstract activities. Each abstract activity is associated with a group of predefined concrete implementation of alternative activities organized hierarchically. This hierarchical representation of alternate workflows though provide an efficient mechanism of selection, reduces reusability of alternate workflows that may be required for other tasks. The selection is done at execution time which is determined by the declarative specification of context conditions. To properly enact the context-aware workflows, they introduced a wrapper on workflow engine as a context-aware workflow manager, which is responsible for selecting an appropriate concrete activity when it encountered an abstract one. The approach is not much flexible and be applied to systems where the situation is context-sensitive but otherwise rigid, like the case of hospital workflows.
In their work, they do not address issues of evolutionary changes and consequently the handling of running instances. Their approach only deals with replacement of abstract activities with concrete implementations but more difficult adaptation operations like addition and deletion are not addressed. The context model they use for their framework is not standard and also not suitable for highly dynamic environments.

Although they implemented their framework using extensions in jPDL (java Process Definition Language), it is not supported by standard enactment services and require an additional component for enactment wrapping, thus compromising the flexibility. Moreover, no discussion of a workflow design tool is made that addresses the integration of framework components.

3.3.2 Smart Workflows for Production (M. Wieland et al.)
M. Wieland et al. targeted the workflow application in smart factory environment where it may be used for automation of technical processes (Wieland et al. 2007), (Wieland, Kaczmarczyk, and Nicklas 2008), (Wieland and Leymann 2010). The context information (e.g., machines, tools, workers) is acquired automatically from sensors and changes in process flow as per physical context are handled pervasively. They couple their workflow system with a context provisioning platform NEXUS (Nicklas et al. 2001) which is responsible for managing context. The Nexus platform is an example of a distributed context platform and provides a unified world-scale context model composed of numerous context servers. All possible adaptations for execution paths are specified in workflow definition, and the appropriate one is selected at runtime based on context. They implement a context integration layer between workflow definition and context provisioning platform to bridge the gap between low level provisioning of context and concepts needed in smart workflows. This layer, however, effective; make the system pretty rigid and inflexible as each time the process changes, the workflow specifications need to be changed along with their high level queries.

3.3.3 Workflows for Location and Context-aware e-Health Application (N. Frohlich et al.)
N. Frohlich et al. proposed a Location and Context-aware e-Health Infrastructure (LoCa) that aims to provide context and location-aware e-Health applications for monitoring physiological data and activity status of patients in a digital home environment (Frührlich and Meier 2009). Although their work is in conceptual stage and yet to be materialized, they proposed solutions for context-aware adaptation in applications where process logic is implemented as workflows. Unlike few other context-aware workflow systems, they intend to provide adaptation for runtime context changes and process evolution both. They intend to use late composition adaptation technique for flexible workflow composition, where only process fragments are specified at build and at run-time workflows are composed out of these available process fragments available. Although the technique is very flexible and effective, but since no implementation is made they do not address the related issues of data and control consistency, workflow verification and validation, roll back and migration.

3.3.4 Dynamic and Flexible worklets in YAWL (M. Adams et al.)
This is one approach where both flexibility in process definition and dynamism for adaptation in changing scenarios are provided in tandem. They implemented a service known as Worklets that provide an extensible repertoire of selection of sub-service
A workflow may contain a task that is marked for worklet substitution and selected at runtime from an available set of worklets using a ripple down rule mechanism. Additionally, an exception-handling service uses the same repertoire and rule-set framework to provide targeted and multi-functional exception-handling processes, which may be dynamically invoked at the task, case or specification level, depending on the context of the work instance and the type of exception that has occurred.

Worklet has been deployed as a discrete service for a well-known, open-source workflow environment YAWL.

The worklet research is promising and may be used for context-aware workflow applications. However, it mainly deals with the process flexibility and exception handling and does not discuss the critical issues of context representation, context integration and use necessary to completely describe a context-aware workflow system.

### 3.3.5 Workflows for Dynamic User Demands (J. Choi et al.)

J. Choi et al. propose the approach for modification of workflow at runtime by applying changes of user’s service demand or situation information into running instance of a workflow without the need of stopping it (Choi et al. 2007). Services are modeled as sub-workflows which can be dynamically inserted into the main workflow if the context conditions are fulfilled. This approach modifies workflow definition but it is reflected only in the running instance. To enable this context-aware adaptation, they introduce a *Ubiquitous Workflow Description Language* (uWDL) (Han, Cho, and Choi 2005) that uses contexts into a well-formed workflow scenario as transition conditions for workflow service. However, a non-standard new context-aware workflow execution manager is required to enact processes modeled in uWDL. An example of use of uWDL for context-aware workflow application is found as u-agriculture (Cho, Moon, and Yoe 2010).

### 3.3.6 Context-sensitive Regions in Workflows (S. Modaferri et al.)

S. Modaferri et al. suggested existing process modeling languages to allow modeling context sensitive regions (CSR) which are part of a business process that may have different behaviors depending on context (Modafferi et al. 2005). CSR includes a description of how to react to a context change defined in a context change pattern. Context change patterns are means to identify the contextual situations (in particular context change situations) that may have an impact on the behavior of a business process. But these changes in workflow constructs translated to seamless execution through workflow engine are not specified. They limit their focus on context information that is relevant to provide personalized services to users based on their environment and needs, including user location, current user device, and network bandwidth only. An important characteristic of their approach is the introduction of “migration points” in their CSR. These migration points allow rolling back of executing tasks to allow switching between alternate predefined implementation. However, the drawback is the predefined positioning of migration points and may lead to handle data consistency before rolling back.
3.3.7 Workflows for Intelligent Navigation (F. Tang et al.)
F. Tang et al. propose a context-aware workflow management algorithm (CAWM) which can dynamically adjust workflow execution behaviors based on current context information (Tang et al. 2010). They used petri-net formalism to show the correctness of dynamism in workflows. The particular focus of their interest is in intelligent navigation systems in pervasive environment where they choose context entities like network status, device type and location information only in their context model. In effect, they address the issues of high mobility of users in dynamic and heterogeneous environments. Their model does not describe process flexibility or build-time adaptations and may only be useful for catering network and device information for adaptation. However, their approach may augment a more flexible method of context-aware workflow system by supporting mobility operations.

3.3.8 Workflows for Knowledge Intensive Tasks (Mitra et al.)
Mitra et al. dynamically integrate knowledge and workflow processes by providing proper support for the real-time handling of both the current context of a process and its execution path (Mitra and David 2008). They advocate that the progression of knowledge, which they consider as context, may produce optimized results. They proposed management of context through issues, context variables and their relevance space. This approach treats context as the knowledge gained through workflow progresses, hence their context model is quite different from others.

This is an interesting approach as it considers context quite differently than used by other researchers. It considers a unique aspect of a process that describes that context may change within a process task and the use of this context change which is more subjective than the sensed context entities should be considered in the execution of later tasks.

3.4 Comparison of Existing Context-aware Workflow Systems
The common characteristics of context-aware workflow systems which are identified earlier in section 3.2 are observed partially in all the surveyed systems as shown in Table 3-1. This indicates that the research in this area is still emerging. Müller et al. (Müller, Greiner, and Rahm 2004) classifies adaptation mechanism for workflows as reactive and predictive to cater for failures in workflow tasks, however, Sucha et al. (Sucha, Sea, and Maria 2008) opines that the same may be used for classifying context-aware workflow adaptations also. In case of reactive adaptation, sensed or updated context triggers the need for adaptation for the task that is about to be executed. Whereas, predictive adaptation adapts workflow parts affected by a logical failure or context change in advance and the adaptation operations are reflected in the workflow downstream.

It is observed that existing context-aware systems all employ the reactive adaptation strategy, which is in fact easy to implement and suitable for fast changing environments like pervasive and ubiquitous computing. Only N. Frohlich et al. suggested the use of late composition that is used to compose a new workflow based on any context change but didn’t discuss its implementation and issues. Almost all approaches provide flexibility in workflow representation to devise a strategy for context-aware adaptation. The degree of flexibility is, however, varied in different approaches. For example,
Ardissono et al., M. Wieland et al., S. Modaferri et al. uses techniques that provide flexibility for supporting alternate course of actions for context information but does not support evolutionary changes i.e. extensions after designing the initial workflow, that limits their flexibility and use in highly changing environment. Generally no particular standard context modeling technique is used by any approach, rather there are customized context models and in few cases holds very less information, for example, in approaches provided by S. Modaferri et al. and F.Tang et al. Context management system is suggested in all approaches whether locally or as a context server. For example, M. Wieland et al. uses NEXUS context management system that is a part of a bigger project to provide context data gathered from various distributed sources. For seamless execution of modified flexible workflow representations, the majority of approaches use a customized wrapping service around the standard engine that transforms the representation syntax into a form understandable by a standard workflow engine. The only approach that does not require any transformation for enactment is presented by M. Adams et al. where the necessary adaptive changes are made within a special service i.e. worklets which works seamlessly with the YAWL workflow engine. Context-aware adaptation dictates runtime adaptation to alternative activities decided by context information and is implemented as late binding i.e. binding of a concrete activity to abstract activity at runtime, or late selection i.e. selection of an alternate activity replacing the inappropriate one, or service discovery. Few approaches like M. Wieland et al. allows for build-time activity binding also. A limited form of rollback and migration feature is provided only by S. Modaferri et al. while the rest of the approaches considers context to progress ahead in the flow only.

The comparison of approaches used by various researchers is made by thoroughly reviewing their available literature only and few of the points mentioned as “not discussed” are the ones where no information is found from their available published work.
<table>
<thead>
<tr>
<th>Researcher</th>
<th>Flexibility in Workflow Representation</th>
<th>Context Representation</th>
<th>Context Integration</th>
<th>Support for Context Management</th>
<th>Workflow Enactment</th>
<th>Adaptation Techniques</th>
<th>Activity Binding Time</th>
<th>Roll back/ Migration</th>
<th>Evolutionary support feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>L. Ardissono et al.</td>
<td>Abstract workflows with hierarchical implementations. Proposed modifications in jPDL</td>
<td>Explicitly defined customized context model</td>
<td>Context data used in rules for adaptation</td>
<td>Suggested</td>
<td>Use Wrapper for enactment service</td>
<td>Reactive</td>
<td>Late binding of activities at runtime</td>
<td>Not supported</td>
<td>Not supported</td>
</tr>
<tr>
<td>M. Wieland et al.</td>
<td>Composed of all possible adaptations. Proposed modifications in BPEL as context4BPEL</td>
<td>Use predefined explicitly defined context models of Augmented World project</td>
<td>Context queries for each activity</td>
<td>Use NEXUS context provisioning platform from Augmented World Project</td>
<td>Use Wrapper for enactment service</td>
<td>Reactive</td>
<td>Late binding of activities but before execution</td>
<td>Not Supported</td>
<td>Not supported</td>
</tr>
<tr>
<td>N. Frohlich et al.</td>
<td>Remodeling suggested</td>
<td>Explicitly defined customized context model</td>
<td>Context query for complete workflow</td>
<td>Supported Locally</td>
<td>Not discussed</td>
<td>Suggested</td>
<td>Predictive but not implemented</td>
<td>Late composition is suggested but not implemented</td>
<td>Not discussed</td>
</tr>
<tr>
<td>M. Adams et al</td>
<td>Flexible activities as worklets. Implemented in YAWL</td>
<td>Not discussed</td>
<td>Contained within activity</td>
<td>Not discussed</td>
<td>Use standard enactment service</td>
<td>Reactive</td>
<td>Replacement at runtime</td>
<td>Partial through exception handing</td>
<td>Supported</td>
</tr>
<tr>
<td>S. Modaferri et al.</td>
<td>Suggested CSR in BPEL processes</td>
<td>Implicitly defined inflexible context model</td>
<td>Context data use as preconditions</td>
<td>Supported Locally</td>
<td>Use Transformation before enactment</td>
<td>Reactive</td>
<td>Late selection during runtime</td>
<td>Supported predefined migration points</td>
<td>Not supported</td>
</tr>
<tr>
<td>J. Choi et al.</td>
<td>Activities composed of Web services in uWDL</td>
<td>Explicit defined context model as RDF schema</td>
<td>Context query for each activity</td>
<td>Suggested</td>
<td>Use customized enactment service</td>
<td>Reactive</td>
<td>Service insertion during runtime</td>
<td>Not supported</td>
<td>Not supported</td>
</tr>
<tr>
<td>F. Tang et al.</td>
<td>Limited flexibility for handling mobility of users</td>
<td>Implicitly defined context model to support mobility</td>
<td>Predefined context variables in workflow</td>
<td>Uses service discovery</td>
<td>Use Wrapper for enactment service</td>
<td>Reactive</td>
<td>Service discovery at runtime</td>
<td>Not supported</td>
<td>Not supported</td>
</tr>
<tr>
<td>Mitra et al.</td>
<td>Limited flexibility as external contexts are not catered</td>
<td>Integrated context variables</td>
<td>creates a dynamic context relevance space</td>
<td>Supported Locally</td>
<td>Use customized enactment service</td>
<td>Reactive/Predictive</td>
<td>Sharing of knowledge data gained during runtime</td>
<td>Not supported</td>
<td>Not supported</td>
</tr>
</tbody>
</table>
Through this comparative analysis of various approaches, the areas where more research are required highlighted and few of them are addressed in the thesis which comes under the scope of the research. These are:

- Need for bridging the gap between flexibility and adaptability in context-aware workflows. To broaden the scope of context-aware workflows, they need to address evolutionary changes with momentary changes as well.
- Need for transparency in modifications and extensions in workflow representation for their seamless execution by standard workflow engines.
- Need for addressing implementations of approaches in workflow languages other than BPEL and YAWL.
- Need for a workflow design tool that provides integrated support for modified context-aware workflow modeling and its associated components as standalone software increasing the usability for designing context-aware workflows.
- Need for more research in predictive adaptation strategy involving rollbacks.
- Need to address the issues of extensibility and reusability of workflows.

3.5 Issues and Challenges in Context-aware Workflow Management Systems

Context-aware applications are hard to develop and most of the times do not cover the range of context types and richness of context-aware features in the developed applications. Furthermore, the incorporation of variability of these applications as workflows, although bring numerous benefits, results in more issues and challenges to be handled in the form of modifications and extensions in workflow technology. As observed in the previous section, existing context-aware workflow applications are limited to scope of specific domain and usually consider fewer context information. For example, many consider location-aware services as context-aware services rejecting the other relevant information on the course. Similarly, many other context-aware applications augment the location information with network quality and device type to cater for mobile applications.

There are several issues and challenges in the design and development of context-aware workflow applications mainly under the umbrella of:

- Difficulty of use of context information.
- Modification and extensions in workflow system.
- Integration of context in workflows for adaptation.

The key issues and challenges are listed below:

1. The selection of appropriate context information that is useful for connecting with the situations in the real world. The impreciseness, incorrectness and
ambiguities of context entities and their relationships often leads to difficulties in development of context-aware applications. Often a domain expert is required to identify the context entities and their influence in the given system. For example, an IT expert cannot model context information regarding the agriculture domain.

2. The modeling of context information with correct methodology as demanded by the complexity of the given application. The various modeling approaches represent, store and access context data in varied ways. For example, a relational database could be used to model an average level of complexity while ontology and other hybrid approaches are useful for more complex applications. In the former case, a simple SQL query could be used to access and update context data while later demands the use of more complex queries like SPARQL or SWRL.

3. The heterogeneity of devices and sources used to acquire context data like sensors, software and network.

4. The provision of low level interface to communicate with sensors / network and aggregation and abstraction of sensed data in the form suitable for context management system.

5. The design of new language constructs to develop flexibility in workflow models necessary for mechanisms of adaptation.

6. The integration of context information that is usable for reasoning and adaptive enactment.

7. The need for specification of rule-sets and related components necessary to make workflows able to adapt to situation at runtime.

8. The provision of alternate execution paths or sub-workflows with mechanisms to dynamically bind the appropriate one as reasoned by the help of context information and rule-sets.

9. The flexibility to improve context-aware workflow definition while there are running instances present in the system. The non-handling of these instances may cause severe inconsistencies and needs to be addressed gracefully. This issue needs to be handled in a dynamic way so that the context-aware application is not affected by this change.

10. The flexibility to add, delete or edit rule-sets as the process evolves or improves. Provisions should be made to make these new rule-sets available to new and running instances uniformly.

11. The interaction of context-aware workflows with the host application in an asynchronous manner so as to allow application to work independently; while the workflows related to processes of application work in the background.
12. The essential challenge of usability of the entire context-aware workflow design, development and execution process. Since the context-aware workflow applications require interaction with various additional components, their integration and use should not hamper the user experience.

This thesis only addresses the issues that are related to design and adaptive enactment of context-aware workflows, assuming that context information is properly managed for use.

3.6 Summary
In this chapter, a survey is presented from existing research on context-aware workflow design and management. The common characteristics of a context-aware workflow management system have been identified and discussed, and a comparative analysis of existing work based on the discussion of these characteristics has been done. It is observed that context-aware applications are being built with the help of context-aware workflows, but require extensions and modifications in workflow technology and support for context management to broaden its scope of use. It is further observed that context-aware workflows are still under-researched and the range of applications that could be developed with context integration is enormous. Most of the research presented carry forward an individualistic perception of context-awareness and is therefore limited to a specific scope. The issues that are identified in this chapter serve as the basis of our research in context-aware workflow design and management.
4.1 Introduction
The previous chapters highlighted the significance of utilizing context information in applications that are primarily designed for dynamic and evolving environments or situations. It was also noted that the use of context is not only fruitful for applications in dynamic domains like pervasive and ubiquitous computing but the traditional business and technical applications may also get benefits if designed in a manner to use derived contexts from its various data sources. This fact is already established that workflows are a useful and natural way of programming applications as it enables the composition of tasks in a coordinated fashion to achieve a process goal. The use of workflows in applications itself brings flexibility by allowing process description to be modeled outside the application. However, it was observed that workflows though very successful in business applications, failed to deliver the same in frequently changing environments. The rigidity of a workflow model and lack of features in workflow products to adapt to changing environments poses challenges for its use in context-sensitive environments.

However, with the appropriate extensions and modifications in existing workflow models and architectures, workflows could be used to design context-aware applications with great success. It is essential to understand that along with the extensions and modifications, the way these applications and workflows are designed, needs to be addressed also. Just like the case when object-oriented programming and event-driven programming surfaced, they changed the way the programs were developed. To further elaborate, take the case of event driven programming which advocates that the
developer focus should remain on events. Most of the time event driven application do nothing but to wait for an event and through trapping that event fires the appropriate event handler, which is contrary to the case of functional programming. It is, therefore, important that one may treat context information as first class citizen and design applications or workflows that revolves around the context by providing necessary infrastructure (Abbasi and Shaikh 2009).

In an attempt to make workflows context-aware so that they could be used in applications in context-sensitive domains, a framework is proposed that describes the requirements, modifications and extensions for designing context-aware workflow applications. The proposed framework augments necessary components to make it appropriate for the design and management of context-aware workflow systems. More importantly it highlights the modeling of context entities and smart workflows, context integration and context-based adaptation techniques. Although the framework discusses the role of context acquisition and management system, it assumes that context is readily available through a context management system e.g. through NEXUS context management platform in case of smart workflows (Wieland et al. 2007) (Nicklas et al. 2001). Further research in context management is not included in the scope of this thesis discussion.

4.2 Principles Governing the Framework

Before describing the proposed context-aware workflow design and management framework, it is important to define the foundation principles upon which the framework stands. It is tried that these principles are reflected in the proposed framework and contains necessary ingredients and mechanisms to display these characteristics.

4.2.1 Flexibility

As discussed earlier, a process hardcoded within an application is totally inflexible and recurrently requires changes in application code to cater for uncertain and certain process improvements. Although workflow models bring flexibility through explicit specification of processes, they once defined are difficult to modify as they require redesign and issues of handling active instances. Furthermore it is impossible to model a workflow at design time that is complete and not liable to any change in the future. Primarily, flexibility is necessitated by process evolution and dictates the modifications in the workflow specification so that they may be loosely or partially defined such that full-blown or concrete workflow model are made at runtime. In order to bring flexibility in workflow models, necessary modifications are required in conventional workflow languages and enactment so that the recurrent changes in workflow model may easily be done at both model and instance level.

4.2.2 Adaptability

Adaptability in workflow models is required to cater for ad-hoc changes that are relevant or unique to a workflow instance, resulting in runtime changes in the workflow model effective for such instances only. The need for adaptation may vary from occasionally changing context to frequently changing context, based upon the domain in which the workflow application is being used. Generally, traditional business processes require occasional adaptations but the processes in pervasive systems require frequent adaptation upon context change. This adaptation may be the replacement of an activity or service with a proper one, addition of a new activity or a sequence of activities, or
altogether the deletion of the activity. In order to introduce this capability of runtime adaptation in workflows, integration mechanisms are required in workflow modeling and enactment with the introduction of extensions like adaptation rule-sets and late service binding etc.

4.2.3 Evolvability
Evolution is a property associated with all progressing things. Processes, whether business or technical, tend to evolve due to a variety of reasons. The business process improvements, change in technologies, change in policies and organizational hierarchy etc. lead to process evolution and need to be taken care of seamlessly in order to avoid hindrances in ongoing process instances. Again, it is emphasized that processes that are built flexible may provide easier mechanisms for evolvability but special mechanisms are required in workflow design tools to introduce new processes, services, and rules into the system with little cost.

4.2.4 Extensibility
When a process model evolves or improves, often it results in extensions in the workflow model. The traditional workflow models do not allow for any extensions without stopping and altering the workflow application and running instances. However, if the flexibility is achieved in a manner that allows extensibility in context-aware workflow components for addition and deletion of sub-workflows, extensions in context models and rule-sets; workflow systems may achieve the desired extensibility.

4.2.5 Usability
It is critical to understand that any modification and extension in workflow system for making it context-aware should not compromise the usability, and ideally should be transparent from users and enactment services. The consequent tool that gets developed for the design of context-aware workflows should provide features for above mentioned principles and are easy to learn and use.

4.2.6 Autonomy
Context-aware workflows should eliminate or reduce the manual intervention as is normally the case of conventional rigid workflows where any deviation from the formal workflow model may stagnate or halt the system.

4.2.7 Reusability
It is a very important feature of context-aware workflows. For the design of context-aware workflows, often the alternative courses of actions i.e. sub-workflows are developed. The scheme of integration should be such that these workflows could be used for other workflow applications and vice-versa. It greatly reduces the design time and allows processes to evolve over time seamlessly.

4.3 Framework Components
The proposed framework describes the extensions and modifications required to transform a conventional workflow system to a context-aware one. These extensions and modifications are realized as various components shown in Figure 4-1.
The design and integration of these components are carried out such that the governing principles described previously are reflected in the framework. The framework may be implemented through use of any existing workflow language and enactment service and

Figure 4-1: Framework for context-aware workflow design and management

The design and integration of these components are carried out such that the governing principles described previously are reflected in the framework. The framework may be implemented through use of any existing workflow language and enactment service and
does not force to use any particular product or system. To implement the presented framework, workflow foundation classes have been used by us as described in the next chapter.

The next section describes the framework components in detail. However, the key features are:

- Flexible representation of workflow processes by introducing the idea of abstract activities that are bound to appropriate concrete activities based on context information at runtime.
- Introducing reusable workflow activities in a workflow pool.
- Explicit representation of context information with integration in workflow models.
- Explicit representation of adaptation rules for increased flexibility and extensibility.

The application logics of the intended system are developed as context-aware workflows. The varied functionalities are available in the form of services (web service) or sub-workflows and workflows are executed by dynamic adaptation of these activities/services at runtime.

The gray blocks in Figure 4-1 represent extensions and modifications in conventional workflow systems while the blocks with dotted lines are optional to be constituted as a part of context-aware application.

4.3.1 Smart Workflow Representation

The foundation stone of the proposed framework to support context-awareness in workflows for dynamic adaptations lies within the workflow representation. It is of significant importance how a workflow model is defined, how context-related issues are handled and more importantly what are the techniques adopted to dynamically select, add or delete activities as per the integrated context-based decisions. Such capabilities of workflow model will impose a direct impact on the quality of support and degree of flexibility and adaptability for providing context-based workflow executions. In the proposed framework, such a workflow model which incorporates context representation, integration, and flexibility for adaptation, is termed as a smart workflow model.

In general context-aware workflow representation requires additional language constructs to model alternative courses of action, provision of context integrations, and mechanisms for adaptation policies. Further to these requirements are the flexibility related changes in representation that could allow both process level and instance level adaptation. It must be noted that these modifications in workflow representation would affect the execution of a workflow instance by a workflow engine as it may encounter language constructs that it do not understand. So, it must be tried that these modifications must remain transparent to the workflow engine otherwise workflow transformation and validation requirements may arise for seamless execution of instances of the workflow model.

In the light of guiding principles, a workflow is modeled as simple as possible by explicitly defining the alternative courses of context-dependent actions. To clarify the idea, consider Figure 4-2 that shows the illustration of how workflows may be
represented. Normally a workflow has at least a single activity (a), or a composition of activities (b) grouped together in a wide variety of patterns (Aalst et al. 2003). An activity in workflows may be atomic or comprise sub-workflows. To handle a situation of multiple choices of activity executions (normally presented as OR-split/join (c)), a workflow representation incorporates multiple activities. Although this pattern comes in handy, it makes the workflow model less readable and extensible, especially for large and rapid changing processes. In contrast, the proposed approach represents a workflow process in abstract form with augmentation of a pool of sub-workflows (d) that is used in binding the appropriate activity at runtime. The abstract activity, which is termed as context-aware activity, is loaded with necessary functionalities for runtime selection and adaptation of appropriate activities/services for execution. Figure 4-2 (e) represents a workflow using context-aware activities as composite design pattern. Context-aware activity is further elaborated in next sub-section.

(e) Workflow (d) represented as composite design pattern

Figure 4-2: Workflow representation
4.3.1.1 Context-Aware Activity
A smart workflow is generally composed of one or more context-aware activities along with optional normal (conventional) activities. A context-aware activity is a special kind of abstract activity that has the capabilities of dynamic and late binding of concrete activities with the help of extensions for context-based decision making and adaptation. This approach is closely similar to one used by Ardissono et al but has varied implementation of associated concrete activities (Ardissono et al. 2006). A context-aware activity is illustrated in Figure 4-3.

A context-aware activity is composed of following functionalities:

I/O Arguments: Just like a normal workflow activity that may require input and output arguments to carry necessary values to execute task associated with the activity, a context-aware activity may also contain these arguments. These arguments, however, are used by the concrete activity bounded at runtime with the placeholder in context-aware activity.

![Figure 4-3: Context-aware activity](image)

Context Model Interface: Context Model interface provides:

- Reference to explicitly defined context model.
- List of context entities related to the scope of activity.
- Context fetch and update mechanism from supported context management system at the start and end of activity respectively.
**Reference to workflow Pool:** The context-aware activity contains reference to an explicitly defined workflow pool that contains a collection of alternate sub-workflows to cater the high degree of variability associated with context-aware workflow applications. This workflow pool is available at build time also to construct a concrete workflow since inception. However, its significance lies in the use for runtime adaptation as per context situation.

**Reference to Adaptation Rule-set:** The context-aware activity contains reference to an explicitly defined adaptation rule-set that contains a collection of rules that is used for dynamic selection of appropriate sub-workflow for execution within the scope of context-aware activity.

**Placeholder Activity:** The placeholder activity may be treated as a container that holds the concrete sub-workflow selected appropriately. The placeholder activity does the following:

- Evaluates the rule-set for the context variables in activity scope and selects the concrete sub-workflow suitable for adaptation based on context information.
- Spawns the selected sub-workflow within the placeholder activity.

### 4.3.1.2 Smart workflow Definition Meta-Model

The WfMC process definition meta-model (WfMC 1999) is used as a standard for any workflow definition language. It, however, is not suitable in its correct form for addressing context-awareness in workflows. Therefore, the WfMC process definition meta-model is extended as shown in Figure 4-4 to incorporate the modifications and extensions made in the form of context-aware activity and associated components (shown as grayed blocks) within the overall scope of the process definition. The context-aware activity makes use of explicitly defined context model, adaptation rule-set and workflow pool for design time and runtime adaptation.

Formally, context-aware workflow represented in Figure 4-4 can be described as a directed graph

\[ W = (V, E) \]  \hspace{1cm} (4.1)

Where \( V \) is the set of activities (nodes) and \( E \) is the set of transitions (edges) such that

\[ E \in \{(u, v) \mid u, v \in V, u \neq v\} \]  \hspace{1cm} (4.2)

In case of our proposed smart workflow \( W \), the set of nodes i.e. activities \( V \) is represented as

\[ V = V_N \cup V_S \]  \hspace{1cm} (4.3)

Where \( V_N = \) Set of Normal (non context-aware) activities

\( V_S = \) Set of smart (context-aware) activities
For each $v_i \in V_S = (CV, WP, AR, PA)$

$CV \subseteq C =$ list of context variables \{c_1, c_2, \ldots, c_m\} in scope of $v_i$

and $C =$ global set of context variables as described in context model

$WP =$ reference to a Workflow pool \{a_0, a_1, a_2, \ldots, a_n, a_{n+1}, \ldots\} of concrete activities such that

Each $a_i (i \neq 0) \in WP$ characterized by a (ID, Priority, implementation)
and $a_0 =$ NULL

AR = reference to adaptation rule-set

PA = a placeholder activity replaced at runtime by $a_i \in WP$. 
Figure 4.4: Smart workflow definition meta-model
4.3.2 Workflow Pool
A workflow Pool (or sub-workflow pool) is a collection of reusable workflows. As discussed earlier, to simplify the workflow representation, the alternative courses of action are not modeled through inflexible OR split/Join or Multi-choice workflow pattern, rather the alternate activities or sub-workflows are explicitly defined outside the workflow model and collected in a repository which is the workflow pool. This Workflow Pool is an open-bounded set of sub-workflows which consists at least one sub-workflow which is a NULL workflow that in effect represents a “do nothing” workflow. This approach is flexible and extensible as it allows for adding/deleting and editing of alternative sub-workflows without disturbing the actual workflow model. It also allows for the reusability of already developed sub-workflows in other processes modeled as smart workflows.

Within a workflow pool as shown in Figure 4-5, reference to each alternate sub-workflow which is represented as a sub-workflow is stored with an ID and priority. The priority of each workflow is initially same, but a workflow designer may change these priorities during or before runtime to allow complex adaptation operations like addition/insertion.

4.3.3 Adaptation Rule-set Repository
The list of context entities, which were associated with context-aware activity through the context model interface at build time, dictates the selection and binding of appropriate concrete sub-workflow for adaptation operations. For such adaptations, a mechanism is proposed that involves the definition of a set of rules where each rule may
evaluate multiple context entities and result in selection of sub-workflow from the workflow pool.

Rules for context adaptation are defined in the form of groups called rule-sets. A context-aware activity evaluates one complete rule-set to select zero or more workflows from the workflow repository. The collection of these rule-sets defined for each context-aware activity in a workflow is termed as Adaptation Rule-set Repository.

A rule is defined as \((\text{Condition}, \text{Action})\) tuple, where a \text{Condition} is a logical relation between different context entities resulting in Boolean value. \text{Action} is the selection of a workflow from the workflow repository if the corresponding \text{Condition} evaluates to \text{true}.

Table 4-1 shows an example rule-set for a hypothetical context-aware activity with C1, C2 and C3 as relevant context variables of type Boolean, Integer and String respectively.

<table>
<thead>
<tr>
<th>S.No.</th>
<th>Example Adaptation Rule-set</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Condition: C1 = True &amp;&amp; C2 = 1</td>
</tr>
<tr>
<td></td>
<td>Action: Select Workflow1</td>
</tr>
<tr>
<td>2</td>
<td>Condition: C1 = True &amp;&amp; C2 = 2 &amp;&amp; C3 = text1</td>
</tr>
<tr>
<td></td>
<td>Action: Select Workflow2</td>
</tr>
<tr>
<td>3</td>
<td>Condition: C1 = False &amp;&amp; (C2 = 3</td>
</tr>
<tr>
<td></td>
<td>Action: Select Workflow3</td>
</tr>
</tbody>
</table>

It is advised that all relevant context entities should be associated with a context-aware activity through the context model interface at build time. However, rules could be made to use all or fewer of these context entities to satisfy the selection of an alternate sub-workflow. The context values for entities not considered are treated as don’t care. But this approach enables the addition of new rules for a context-aware activity to consider these don’t care entities in future. Moreover, rules in a rule-set need to be mutually exclusive until and otherwise deliberately made to perform the addition operation in the workflow model where the order of execution of multiple selected sub-workflows is governed by priority associated with these workflows.
4.3.4 Context Model

Much of the earlier research towards context-awareness revolves around hard-coding the semantics and adaptation logics into the underlying system implementation. This technique, however, works but is too rigid to use in domains with fast evolving contexts. Therefore, a need for the explicit definition of context is required which remains external to the application implementation and is shared by constituent components in a context-aware workflow application. As described in section 2.4, a true context-aware application considers context information as a first class citizen and the application execution revolves round the context information making it essential to devise mechanisms for context integration and use.

In a context-aware workflow application, context information is integrated with workflows rather than the application itself. This context or change in context information serves as a precondition or an entry-condition to select, add or delete a workflow activity for execution which is most suitable for the given situation.

In order to effectively and consistently manipulate data entities, attributes and context, one need to organize, represent, and describe them in a model, commonly referred as context model. This model is shared and may serve as the basis of common understanding of data entities, attributes and context between constituent subsystems of a context-aware system. Context modeling poses an additional requirement of a domain expert to model the context information especially if the system is complex. However, this additional constraint is presumed as a positive thing as it involves right persons to describe the environment and users who are going to use the system and the system design is supposed to be more usable and close to reality.

As part of framework, a context meta-model have been developed that covers the essence in which a context model for a particular application domain is modeled. As discussed earlier, various context modeling techniques are available ranging from simple value matching to composite semantic based relations. Based upon the proposed context meta-model, a context model may be developed from any technique that is best suitable as per the complexity of applications and expertise of developers. Moreover, it is proposed that the selection of context model and its integration in workflows are flexible. At design time, the workflow designer tool should enable to associate any particular kind of workflow model and may create a desired context relevance space as dictated by the selected context model technology. For example, in case of selection of a relational model, one may use the context model as a relational database model with tables and relations. This context model may be associated to the workflow just like a database is attached to an application.

Figure 4-6 shows a context meta-model in Entity-Relation notation used in our framework. This meta-model signifies that any context entity may be viewed as a Person, Place or Thing as suggested by A. K. Dey (Dey and Abowd 2000). Moreover, a composite context entity resulted from a relation between atomic context entities may be used to represent a context entity with a higher level of abstraction. Context entity values are usually provided manually, through sensing devices or any other software component like profile database or back-office information. The type of context provider information is useful for context management systems to sample, process and
store context data. Also a context model may reference another context model or even gets derived from one. This hierarchical or extensible context modeling approach is dependent upon the context modeling technique used to model context information. For example, ontology-based context modeling may allow extensions and relation within the context model.

Figure 4-6: Context meta-model
4.3.5 Context-aware Workflow Enactment

Every workflow is instantiated and executed by a workflow enactment service that takes the explicitly defined workflow model and executes it. When a workflow model modified along the lines described in section 4.3.1 is enacted by standard workflow enactment service, inconsistencies would arise that need to be tackled for smooth execution of workflow instances.

Figure 4-7 (a) shows the ideal way of execution of a workflow instance which shows that any addition or modification made in workflow languages are transparent to the enactment service and it enacts the context-aware workflow definition similar to a conventional one. Figure 4-7 (b) shows that a workflow model needs to eliminate the inconsistencies caused by using constructs for adding context variables and rules. In this case a transformation and validation are made so that workflow enactment service could work transparently. Figure 4-7 (c) shows the feature of using a context management wrapper on workflow enactment service, so that context-dependent activities are sorted earlier and passed to workflow engine in manageable form. Lastly, one may develop a new workflow enactment service that can understand the modified workflow specification and be easily integrated with other context related components.

However, the scheme presented by the proposed framework does not require any transformation or enactment service wrapper or a new enactment service. The integration of context entities, the adaptation strategy, and concrete binding of activities are made in a manner that transparently serves the purpose with the help of standard workflow enactment service.

![Diagram of Context-aware workflow enactment strategies](image)

**Figure 4-7: Context-aware workflow enactment strategies**
The context-based adaptation mechanism used by the enactment services have strong relation with the flexibility provided in workflow representation. In the framework, the use of reactive adaptation strategy is proposed. When each context-aware activity is being executed, it is effectively replaced by an alternate sub-workflow whose selection and binding was delayed to adjust the context information till last time. These selections and blending adaptation may also result in more complex operations of deletion and addition of activities. Finally, it must be noted that the context-aware activity should be loaded with embedded functionality of context fetch and update mechanism by directly accessing the context management system.

4.3.6 Context Management
Context management support is an important and essential component of any context-aware application. It provides preprocessing, storage and access to context information as acquired through context providers i.e. sensors and software (shown in Figure 4-8).

![Figure 4-8: A view of context management](image)

In Figure 4-1, context management component is shown in dotted block which signifies that it may or may not be an integrated part of context-aware application. However, in the framework, it is proposed as a separate entity so that it becomes more manageable and used by various applications at the same time.

An illustration of a context management system with interaction with context space and context-aware workflow application is shown in Figure 4-8. The context-aware workflow and context management system share the same context model. This enforces the context management system to acquire, process and store data in a form accessible through queries made by workflow instances. The dotted line in Figure 4-8 from context space to context-aware application signifies that occasionally some systems are made where direct inference for context values is made by application bypassing the context-management system.
Since the scope of this thesis is limited to design of context-aware workflow applications, it is assumed that context information as modeled through a context model is available for access in updated form always.

### 4.3.7 Context Space

Context space is a term associated with the sources used within the system boundary to acquire relevant information necessary to define the static and dynamic characteristics of the system for which the context-aware workflow application is being developed. In chapter 2, discussion is made in detail about the concept of context entities, their classification and sources. In general, within a context space, one may have sensing devices, information databases and manual updates. If the application is developed for a highly dynamic environment like pervasive smart environments, users mostly rely on sensed data coupled with profile information to acquire situation information for adaptation. While some workflow processes inquire only information from software sources to cater for evolutionary changes or selection of the appropriate execution path. A context management system described above is the main interface which acts as a bridge between context space and application.

As part of research continuum, a wireless sensor network for smart agriculture have been developed that constitutes the major portion of context space for smart agriculture applications (Aqeel-ur-Rehman et al. 2011). It contains smart sensor nodes that collect data about soil moisture, humidity, temperature, light etc. Smart agriculture is further explained in chapter 7 as an example scenario.

However, within the scope of thesis research, context space is considered as available with a context management system that is used in design and development of context-aware workflow applications.

### 4.4 Summary

In this chapter, a generic framework for the design and management of context-aware workflows is presented. The framework proposes major changes in workflow representation that enable the existing workflow languages to be used in the development of context-aware workflow applications. Central to modifications made in workflow representation is the concept of context-aware activity that is, in fact, an abstract activity at design time but holds the necessary functionality to adapt to contextually appropriate concrete activity at runtime which is stored as a collection of alternate sub-workflows in a pool. The selection is based on integrated context entities within the context-aware activity and an evaluation is made with the help of explicitly defined adaptation rule-sets. A context meta-model is presented so that a context model could be developed with the help of available context modeling approaches. Context management and context space are described for their significance and use in the overall prospect of context-aware workflow application execution.
Chapter 5

Context-Activity Architecture

5.1 Introduction
The previous chapter described the framework components separately. The integration and working of these components to provide context-awareness in workflows is realized by Context-Activity architecture (Abbasi, Shaikh, and Ahsan 2012). Since context and activity have strong bonding and context information greatly influences the evaluation and selection of appropriate concrete activity for dynamic binding, the name of architecture is proposed as context-activity architecture which is discussed in detail in this chapter. A formalism of the operations of adaptation is described along with the underlying algorithms. These operations of adaptation during runtime and the handling of evolutionary changes are tested and verified through simulated testing.

5.2 Working of Context-Activity Architecture
Figure 5-1 (a) shows the integrated working and role of framework components in the selection and binding of a concrete activity at runtime and figure 5-1 (b) further explains the same in the form of sequence diagram elaborating the temporal relationships.

1) First the Fetch Context part of the Context Model Interface queries the associated context model for the updated values of relevant context entities.

2) The Context Model provides updated context values to the Context-aware activity.

3) An Adaptation Rule-set which was defined for this context-aware activity and attached to Rule-set Repository is evaluated through spawning a rule-set evaluator making use of the updated context values.
Figure 5-1: Context-Activity architecture
(4) Rule-set evaluation results in the selection of one or more sub-workflows from the Workflow Pool.

(5) Selected workflow(s) are then bound dynamically to the Placeholder Activity for execution.

(6) After the execution of selected workflow(s), the Update Context part of Context Model Interface may pass the new values of relevant context entities back to the context model, if required.

The same procedure is adopted autonomously for all context-aware activities associated with the workflow until the very workflow is terminated. It must be noted that the explicit integration of adaptation rule-set repository and workflow pool enables the addition, deletion and editing of rules and alternate sub-workflows making the context-aware workflow application extremely flexible and extensible.

5.3 Operations of Runtime Adaptations
The proposed framework has presented an architecture where any evolutionary or runtime (i.e. ad hoc) change or adaptation is carried out without altering the structural shape of the workflow. This approach enables handling of running instances of that workflow without difficulty. Any runtime change or adaptation is realized through finding, binding and execution of concrete activities/sub-workflows at runtime that is best-suited for that particular abstract context-aware activity. The two algorithms FIND and SPAWN presented in Figure 5-2 and Figure 5-3 respectively show the techniques of finding and spawning of concrete activities/sub-workflows from workflow pool. The evolutionary changes can be made in the workflow through extensions and modifications in rule-sets and workflow pool while the workflow instances are still running, which adapted to the new modifications at runtime.

In algorithms, straight forward functionalities are not further expanded. For example fetch_context() and update_context() are used to get and submit the updated context values before and after the execution of selected concrete sub-workflow, execute() executes the sub-workflow within the placeholder, evaluate() evaluates the rule-set based on context entities of interest and selects appropriate sub-workflow from workflow pool, and sort() orders the sub-workflows if selected more than one.

Assume

\[ W = \{w_1, w_2, ..., w_k\} \] ; Set of context-aware activities in workflow W

\[ A = \{a_0, a_1, ..., a_n\}, n \geq k \] ; Workflow Pool associated with W and a_0 is NULL workflow

\[ C = \{c_1, c_2, ..., c_m\} \] ; Set of Context entities

\[ R = \{r_1, r_2, ..., r_p\} \] \( \forall r_i \in R = (Condition, Action) \) ; Adaptation rule-set

\[ \varphi \subseteq A = \text{Set of Selected Activities for } w_i \]

Also, \( \forall w_i \in W \exists C_i \subseteq C \mid |C_i| \neq 0, \forall a_j \in A \exists (ID, Priority) \)
\[ \varphi = \text{FIND} (W, A, C, R) \quad ; \text{For each } w_i \in W \]

BEGIN

\text{SET BOOL } B = \text{FALSE}

\[ C_i = \text{fetch\_context} (C) \]

\text{FOR } j = 1 \text{ TO } p \]

\[ B = \text{evaluate} (C_i, \eta_j) \]

\text{IF } (B \text{ == TRUE}) \text{ THEN}

\[ \varphi_j = a_q | a_q \in A, a_q \neq a_0 \]

\text{ELSE}

\[ \varphi_j = \{ \} \]

\text{ENDIF}

\text{NEXT } j

\[ \varphi = \varphi_1 \cup \varphi_2 \cup \ldots \cup \varphi_p \]

\text{SPAWN } (\varphi)

\text{update\_context} (C)

END

Figure 5-2: Algorithm for dynamic finding of appropriate sub-workflows

\begin{verbatim}
SPAWN (A, \varphi)
BEGIN

INTEGER \( x = 0 \)

\text{IF } |\varphi| = 0 \text{ THEN}

\text{execute} (a_0) \quad ; \text{the deletion case}

\text{ELSEIF } |\varphi| = 1 \text{ THEN}

\text{execute} (\varphi) \quad ; \text{the selection case}

\text{ELSEIF}

\text{sort} (\varphi)

\( x = |\varphi| \)

\text{FOR } i = 1 \text{ TO } x

\text{execute} (\varphi) \quad ; \text{the addition case}

\text{NEXT } i

\text{ENDIF}

END
\end{verbatim}

Figure 5-3: Spawn algorithm
The finding of relevant activities/sub-workflows based upon the context-driven conditions may result in the selection, deletion and addition adaptation operation.

### 5.3.1 The DELETION Operation

The deletion operation signifies that although there is an activity present in workflow for doing a particular task, the context-based conditions determine that it is appropriate not to perform the task. The FIND algorithm in case of deletion results in $\varphi \subseteq A = \emptyset$ i.e. a null set of sub-workflows from the workflow pool. The SPAWN algorithm upon checking that the selected activities set is null, place $a_0$ in the Placeholder for execution. In other way, if

$$|\varphi| = 0$$

the operation is deletion (illustrated in Figure 5-4).

![Figure 5-4: The deletion operation](image)

### 5.3.2 The SELECTION Operation

The selection operation signifies that context-based conditions determine the selection of an appropriate concrete activity/ sub-workflow from workflow pool. The FIND algorithm in case of selection results in $\varphi \subseteq A = A_j \neq 0$ i.e. any single workflow other than NULL workflow from the workflow pool. The SPAWN algorithm upon checking that the selected activities set is not null and equal to one, place $A_j$ i.e. the selected sub-workflow in the Placeholder for execution. In other way, if

$$|\varphi| = 1 \text{ } \varphi \neq \{a_0\}$$

the operation is selection (illustrated in Figure 5-5).

![Figure 5-5: The selection operation](image)
5.3.3 The ADDITION Operation
A context-aware activity can add / insert activities to the workflow by defining adaptation rule-set in such a way that more than one workflow are selected from evaluation of rule-sets. In our approach, an addition operation signifies that context-based conditions determine the selection of more than one appropriate concrete activities/sub-workflows from workflow pool. The FIND algorithm generates a set of selected activities whose order of execution is determined by the SPAWN algorithm. The Priority attribute associated with each sub-workflow in the pool helps in deciding the order. The selected sub-workflows executed sequentially in a descending order of priority and executed in parallel for same priorities. In other way, if

\[ 1 < |\varphi| \leq n \]  \hfill (5.3)

the operation is addition (illustrated in Figure 5-6).

![Figure 5-6: The addition operation](image)

The significance of addition operation arises when one wanted to insert an activity before or after a context-aware activity due to some process improvements and not wanting to sacrifice the reusability of existing sub-workflows in the pool. By setting the priority attribute of a sub-workflow with respect to another sub-workflow, one may dictate the execution order of these activities.

5.4 Handling Evolutionary Changes
In general, a context-aware workflow application is characterized by ad-hoc changes in structure and behavior of workflow instances to cater for context information and changes. However, process evolution, change in polices etc. demands long-term changes in workflow process model. The proposed framework and architecture provides compensation for both ad-hoc and evolutionary changes. After the modifications in workflow model, new workflow instances follow the new definition and ideally speaking it should be reflected in running instances also.

The workflow representation proposed in the framework is quite smart and do not require redesign of workflow model to incorporate evolutionary changes. Since explicit representation and association of adaptation rule-sets and sub-workflows for alternate actions are employed, one may edit or append the necessary changes with the help of
our workflow designer tool while there are instances already running. Since the workflow instances carry only the references to the workflow pool and adaptation rule-sets, any change in them remains transparent for the instances. This approach works seamlessly for new workflow instances for the modified workflow model. However, there is a limitation of rolling back of running instances and they only adhere to the new workflow definition from the point forward where they were executing when the change is made.

The evolutionary changes in workflow model are made through.

1. Addition, deletion and editing of adaptation rule-sets.
2. Addition, deletion and editing of sub-workflows in workflow pool.

The next section describes the handling of evolutionary changes with the help of a hypothetical example.

5.5 Simulated Testing of Context-Activity Architecture

For the simulation and testing of the context-activity architecture, a test application named *Workflow Execution and Tracking App* was developed. This test application provides us a user interface to execute and track multiple workflows at the same time. The tracking messages from each workflow are displayed in a separate tab. The application also allows us to forcefully change the values of different context variables, attached rule-set and the workflow pool.

Based on the types of features tested, testing procedure can be divided in two categories

1. Testing runtime adaptation operations.
2. Testing evolutionary change adaptation.

In order to demonstrate and test both types of features, let us consider a hypothetical example of a simple workflow consisting of two context-aware activities $W_1$ and $W_2$ with related rule-set $R_1$ and $R_2$ and set of relevant context variables $C_1$ and $C_2$ respectively as depicted in Figure 5-7.

![Figure 5-7: A hypothetical context-aware workflow for simulated testing](image)

Where, $C_1, C_2 \in C$ (C is the associated context model), and $R_1, R_2 \in R$ (R is the associated rule-set repository)
The context model C integrated with our example workflow contains only the following context variables:

1. $c_1$ of type \textit{Boolean}.
2. $c_2$ of type \textit{String}.
3. $c_3$ of type \textit{Integer}.

The set of relevant context variables is defined as $C_1 = \{c_1, c_2\}$ and $C_2 = \{c_2, c_3\}$

The workflow pool associated with our example workflow is a set $A$ containing workflows from $A_1$ to $A_{10}$.

The attached rule-set repository contains the following rule-sets.

1. \textbf{R}_1:

   \begin{itemize}
   \item \textbf{Condition}: $c_1 = \text{true} \land c_2 = \text{text1}$
   \textbf{Action}: Select Workflow $A_1$
   \item \textbf{Condition}: $c_1 = \text{false} \land c_2 = \text{text1}$
   \textbf{Action}: Select Workflow $A_2$
   \item \textbf{Condition}: $c_1 = \text{false} \land c_2 = \text{text2}$
   \textbf{Action}: Select Workflow $A_3$
   \item \textbf{Condition}: $c_1 = \text{true} \land c_2 = \text{text2}$
   \textbf{Action}: Select Workflow $A_4$
   \end{itemize}

2. \textbf{R}_2:

   \begin{itemize}
   \item \textbf{Condition}: $c_2 = \text{text1} \land c_3 = \text{1}$
   \textbf{Action}: Select Workflow $A_6$
   \item \textbf{Condition}: $c_2 = \text{text1} \land c_3 = \text{2}$
   \textbf{Action}: Select Workflow $A_7$
   \item \textbf{Condition}: $c_2 = \text{text2} \land c_3 = \text{1}$
   \textbf{Action}: Selected Workflow $A_8$
   \item \textbf{Condition}: $c_2 = \text{text2} \land c_3 = \text{2}$
   \textbf{Action}: Select Workflow $A_9$
   \end{itemize}
5.5.1 Testing Runtime Adaptation Operations

Runtime adaptations are the changes made in workflow instances at runtime. The changes include selection of a sub-workflow, deletion of an activity or addition/insertion of an activity. As discussed in the previous section that finding appropriate sub-workflow(s) described all the three selection, deletion and addition operations, only the selection operation at runtime is tested showing the context-aware workflow behavior at runtime based on forceful context changes.

The ability to select a sub-workflow for the example workflow by running two instances of the workflow simultaneously with different context conditions is tested. To distinguish between the two instances, the display IDs of the workflow are set as Workflow1 and Workflow2. For Workflow1, the values of the context variables were $c_1 = \text{true}$, $c_2 = \text{text1}$ and $c_3 = 1$; whereas for Workflow2, the values were $c_1 = \text{false}$, $c_2 = \text{text2}$ and $c_3 = 2$.

Below are the screen shots of the tracking messages for both the cases. The screen shots in Figure 5-8 clearly show that each context-aware activity adapted to the values of the context variables as per the associated rule-set.

![Screen shots](image)

(a) (b)

**Figure 5-8: Selection of appropriate sub-workflow**
Looking at the rule-sets defined above, one can see that in case of context conditions specified for *Workflow1* rule no. 1 of each rule-set should be fired while in case of *Workflow2* rule no. 3 of R₁ and rule no. 4 of R₂ should be fired.

Figure 5-8 (a) shows that context-aware activities W₁ and W₂ in the workflow instance *Workflow1* correctly select the sub-workflows A₁ and A₆ respectively according to the rules 1 of each R₁ and R₂ while Figure 5-8 (b) shows that context-aware activities in *Workflow2* select the sub-workflows A₃ and A₉ as expected.

Deletion and addition of an activity are performed in a way similar to the selection of a sub-workflow and are discussed in detail in chapter 7.

**5.5.2 Testing Evolutionary Change Adaptation**

**5.5.2.1 Editing Adaptation Rule-set at Runtime**

To test the ability of editing the associated adaptation rule-set during the execution of a workflow, the example workflow is first executed and then after Smart activity W₁, it is paused. During the time when the workflow was paused, the second rule in the adaptation rule-set R₂ is edited so that the rule-set R₂ becomes,

```
Condition: C₂ = text1 && C₃ = 1
Action: Select Workflow A₆

Condition: C₂ = text3 && C₃ = 3
Action: Select Workflow = A₅

Condition: C₂ = text2 && C₃ = 1
Action: Select Workflow = A₈

Condition: C₂ = text2 && C₃ = 2
Action: Select Workflow = A₉
```

The values of the context variables were then set as C₂ = text3 and C₃ = 3 so that only the newly edited rule would fire. Below is the result of this test (Figure 5-9). It is clear that the workflow successfully adapted to the change in the associated adaptation rule-set.
5.5.2.2 Editing Workflow Pool at Runtime

Process evolution requires that a Workflow pool attached to a context-aware activity must also be editable at runtime. To test this ability, similar approach that is took while testing for rule-set editing during runtime is used, i.e., the example workflow is executed and then after Smart activity W₁, it is paused. During the period when the workflow was paused, a new workflow A₁₁ is added to the workflow pool and also the second rule of the rule-set R₂ is changed as follows,

\[
\text{Condition: } C_2 = \text{text3} \land C_3 = 3 \\
\text{Action: Select Workflow} = A_{11}
\]

The context condition was kept unchanged so that only second rule of the rule-set R₂ fired again. The screenshot below (Figure 5-10) shows that the example workflow successfully adapted to the change in workflow pool and the rule-set at the same time.
5.6 Summary
The integration and working of context-aware workflow components is described through context-activity architecture with the detailed description of the feature set it provides for context-awareness and process evolution. A simulated testing is carried out with a specially developed workflow tracking application where changes were forcefully made in context information at runtime and the adaptation operations on running workflow instances are tested. The runtime adaptations and evolutionary changes are tested successfully with the help of this application as evident from application snap shots.
Chapter 6

Context-aware Workflow Designer (CAWD)

6.1 Introduction
Context aware workflow designer (CAWD) is a software tool that is developed to realize and implement the concepts presented in our proposed framework and architecture (Abbasi et al. 2010). It is a standalone software that facilitates the design of context-aware workflows by providing the necessary functionalities and integration of components in a single package. It effectively provides support for both build time and runtime reactive adaptation by selecting, adding or deleting appropriate activities as inferred through contextual information. CAWD is developed using Microsoft Windows Workflow Foundation (WF) classes and may also be used to design and verify traditional workflows. CAWD extends the WF workflow designer classes by introducing context-aware activities that have the additional functionality of context integration and adaptation. Context-aware activities are introduced in such a way that no transformation rules are required to execute context-aware workflows through standard workflow engine.

As discussed in section 2.2.2, workflow modeling languages like XPDL (Shapiro 2008), BPEL4WS (OASIS 2007) and YAWL (Aalst et al. 2003) are widely used for modeling workflows. But the introduction of Workflow Foundation by Microsoft has attracted a wide audience for workflow modeling that had already been using Microsoft development platforms and commercial workflow products like Sharepoint and Biztalk. Hence, it was decided to work on Workflow foundation and develop a specialized workflow designer customized for modeling context aware workflows fulfilling the additional requirements in their design. In addition to providing an access to a large audience, WF3.5 provided some powerful features like Dynamic Update and Local
Services (Bukovics 2008) that was thought would be helpful in implementing context-aware workflows. CAWD is the first implementation as any context-aware workflow designer tool using Windows Foundation classes.

The initial version of CAWD (Abbasi et al. 2010) was built on WF version 3.5 (WF3.5) shipped with the .Net Framework 3. When the .Net Framework 4.0 was released, Microsoft decided to completely revamp the Windows Workflow Foundation and in the process added many useful features while dropping many others. The new version of WF, which is the latest till now, was termed WF4. Since WF4 is completely a new framework; CAWD had to be redesigned. The new version is called CAWD4 while the previous version is renamed as CAWD3.5. Although the basic architecture was same there are some additional features that are made in CAWD4 which will be described later in this chapter.

In this chapter, first a brief description of WF and its terminologies will be presented. Next, the motivation behind the development of CAWD will be explained followed by the architecture and implementation details of CAWD.

6.2 Introduction to Windows Workflow Foundation (WF)

Windows Workflow Foundation comprises programming model, engine, and tools for quickly building workflow-enabled applications on Windows. It consists of a namespace, an in-process workflow engine, and designers for Microsoft Visual Studio. Windows Workflow Foundation is a framework which enables users to create system or human workflows in their applications written for Windows Vista, Windows XP, and the Windows Server 2003 family. Advantages of WF include ability to run activities in parallel relieving the developer of the complexities involved in programming parallel applications, automatic tracking of a workflow execution, ability to create re-usable custom activities and making a process more visible and understandable.

Some of the terms described in WF documentation are as follows:

Workflows
Workflow is a way of describing a process by defining ordered execution and dependent relationships between pieces of short or long running work. A workflow is composed as activities in a model from start to finish executed by people or system functions. In WF4, steps of the workflow are defined as a hierarchy of activities, therefore the topmost activity in the hierarchy can be said to define the workflow itself.

Activities
Activities are the elemental unit of a workflow. When all the activities in a given flow path are finished running, the workflow instance execution is completed. An activity can perform a single action, such as writing a value to a database, or it can be a composite activity and consist of a set of activities. Activities have two types of behavior: runtime and design time. The runtime behavior specifies the actions upon execution. The design time behavior controls the appearance of the activity and its interaction while being displayed within the designer.

Workflow Run-time Engine
Every workflow model is instantiated and executed by an in-process runtime engine that is commonly referred to as the workflow runtime engine. There can be several
workflow runtime engines within an application domain, and each instance of the runtime engine can support multiple workflow instances running concurrently.

A workflow instance is executed inside any Windows process including console applications, forms-based applications, windows services, ASP.NET web sites, or web services. A workflow can easily communicate with its host application as it is hosted within it. Figure 6-1 shows how workflows, activities, and the workflow runtime engine are all hosted in process with a host application.

![Diagram](image)

**Figure 6-1: Workflow host**

### 6.3 Motivation for CAWD

WF4 provides way of re-hosting a workflow designer outside of Microsoft Visual Studio as a standalone application. Although the suggested context-aware activities can be used with the workflow designer provided with Microsoft Visual Studio, it was deemed useful to develop own workflow designer primarily for the following reasons:

- To enable developers to design context-aware workflows without requiring them to have Microsoft Visual Studio.
- To enable advance developers to view and directly edit the xaml representation of the workflow using the *Xaml Editor Pane*.
• To facilitate the design of context-aware workflows by allowing the developer to operate on the integrated context model and the workflow pool within the workflow designer.

• To enable handling of evolutionary changes by adding, deleting and editing rules-sets and workflow pool without disturbing actual workflow model.

• To increase the usability.

6.4 CAWD Architecture
The basic architecture of CAWD is depicted in Figure 6-2 below. A re-hosted workflow designer provides the basic functionality of designing workflows visually by simply dragging the activities from activity toolbox, placing these on the designer surface and setting their properties appropriately.

![Figure 6-2: CAWD architecture](image)

Context-aware activities that have been developed are the most important ingredient of CAWD and are discussed in much detail later. These activities are actually the workflow extensions that our proposed framework for context-aware workflows defines for a smart workflow representation. Context-aware activities are capable of being integrated at design time with a context model, a pool of related workflows and a set of rules for workflow selection. Then at runtime, a context aware activity can smartly make use of the integrated context model, workflow pool and set of rules to achieve a desired task.
Figure 6-3 below presents an expanded view of smart workflow meta-model presented in Figure 4-4 focusing context aware activity.

**Figure 6-3: Basic meta-model of context aware activity**

A context aware activity must enable the workflow developer to integrate a *Context Model* with the activity, fetch relevant context variables through a *Context Model Interface (CMI)* and evaluate adaptation rule-set in the *Rule Repository* to select a workflow at runtime from its list of related workflows i.e. *Workflow Pool (WP)*.

It has been assumed that a *Context Model* is already developed and the mechanism to acquire the context information either through hardware sensors or software agents is in place. So, all one need is to access this *Context Model* and inquire it for the values of the relevant context variables. A *Context Model* may be represented in any of the standard forms such as relational database or ontology etc.

*Workflow validation* validates a workflow syntactically and generates errors in case a workflow is defined incorrectly.

Besides visual design of workflows, CAWD gives advanced users the ability to manually edit the *Workflow markup* i.e. *Xaml* in case of WF. *Workflow markup* is the final output of the workflow designer and is subsequently used by the host process in conjunction with a context model/manager and a set of related workflows and web services.

Now the details of various components of the CAWD architecture (depicted in Figure 6-2) will be discussed as implemented in CAWD4 and CAWD3.5.
6.5 CAWD4 Implementation Details

It is reiterated here that the thesis presented a generic framework for design and management of context-aware workflows whose implementation could be achieved in any standard workflow language and system. For example, *Yet Another Workflow Language* (YAWL) and *Java Process Definition Language* (jPDL) could be modified along the same lines to be used in java based platforms. BPEL4WS is another popular workflow language which provides the orchestration of web services aptly supported by Linux platform. Whatever platform/workflow language one use for context-aware workflow representation/ execution, modifications and extensions are required as suggested in our proposed framework. In some cases, these modifications would be easier to implement while in others it may not. Say, in case of BPEL, the context-awareness may be provided by designing smart/abstract web services that dynamically bind to concrete web services from a pool of web services as per the result of a context query.

However, it was preferred to work with the windows Workflow Foundation to implement the proposed architecture for representing and managing workflows for context-aware applications. The motivation for this choice was stated earlier in this chapter.

Now, the implementation of various components of CAWD will be discussed in detail.

6.5.1 Re-hosted Workflow Designer

The ability to re-host a workflow designer in third party software offers a number of benefits to the users,

1. An application re-hosting a workflow designer can be compiled as a redistributable package to workflow designers so that they do not need entire Microsoft Visual Studio just for the purpose of designing workflows.  
2. An application can augment the workflow design experience by re-hosting it and integrating it with specialized features specific to a particular domain.

Acknowledging the benefits of re-hosting the workflow designer, the *WorkflowDesigner* class is used to host the fundamental functionalities of the workflow designer in CAWD. In this re-hosting two properties are important:

- **View**: It returns a UI element that allows the user to view and edit the workflow visually. This UI element is commonly referred to as the *workflow canvas*.

- **PropertyInspectorView**: Gets the property inspector, which is commonly known as the *property grid*.

To view and edit the workflow markup file in the *Xaml editor pane*, one more *WorkflowDesigner* property was required,

- **Text**: *Gets or sets the XAML string representation of the workflow*.

*ToolBoxControl* class implements the *Activity Toolbox*. This class provides support for representing the toolbox controls when re-hosting the Windows Workflow Designer. The *ToolBoxControl* contains a collection of toolbox categories that hold various
categories of tools that can be accessed with the `Categories` property. It is attached with the `WorkflowDesigner` instance used for designer re-hosting using the `AssociatedDesigner` property.

Figure 6-4 shows a screen shot of CAWD depicting Xaml Editor Pane and Context Relevance Space integrated with the standard designer.

![CAWD4 snapshot](image)

### 6.5.2 Context Model / Rule-set Repository / Workflow Pool Integration

As per our proposed framework, a context-aware activity needs access to *Context Model, Rule-set Repository* and *Workflow Pool*. Since every context-aware activity requires these external models, they need to be integrated into some kind of root activity. In WF4.0, any activity can be the root activity of the workflow. But for workflows which are composed of more than one activity, there are only two possible root activities; a `Sequence` or a `Parallel`. Since `Sequence` is most commonly used, a context-aware root activity called `SmartSequence` activity similar to WF4.0 standard `Sequence` activity is defined that executes its child activities according to the defined order. However, one difference between `SmartSequence` and `Sequence` activities is that the former has three additional properties named `Context Model`, `Rule-set Repository` and `Workflow Pool` which defines paths for the `Context Model`, the `Rule-set Repository`
file and the *Workflow Pool* file and integrates these to this *SmartSequence*. All other context-aware activities are required to be placed within a *SmartSequence* giving them automatic access to the integrated models.

Another *SmartSequence* property named *ContextModelType* enables us to make available only those context-aware activities which are designed for the selected context model type. A context model can be of any standard type, such as ontology or relational database.

*SmartSequence* activities can be nested within other *SmartSequence* activities but any context aware activity with more than one *SmartSequence* activities in its parent chain has access to only one set of models i.e. the ones which are bound to the *SmartSequence* activity closest to it in its parent chain.

### 6.5.3 Workflow Validation

For workflow validation, WF4.0 provides a *ValidationService* class and an *IValidationErrorService* interface to this class. *ValidationService* provides the functionality required to validate a workflow activity designer while it is being edited and to provide the results of the validation process and *IValidationErrorService* defines ways to access a specified list of activity validation errors.

### 6.5.4 Context-Aware Activity (CA)

The CA is implemented in such a way that the WF runtime engine needs no transformation rules. CA developed for CAWD4 is termed as *SmartActivity*. A class diagram of the major classes used in SmartActivity is shown in Figure 6-5.

*Sequence* and *NativeActivity* are the classes already provided in WF4.0 while other classes in Figure 6-5 are user defined. *SmartSequence* is derived from the *Sequence* class and is extended to include attributes like *Context Model*, *Rule-set Repository* and *Workflow Pool*.

---

![Figure 6-5: Major classes defined for implementation of SmartActivity](image-url)
As described in section 6.5.2, a *SmartSequence* acts as a parent to the actual context-aware activity, i.e. *SmartActivity* and the *ContextRetrievalActivity*, a helper activity in a context-aware workflow, derived from *NativeActivity*. *Workflow* is a class that represents a workflow in the workflow pool where each *Workflow* object contains a Workflow Name, Workflow Path and Priority as its attributes.

Figure 6-6 is the CAWD realization of the context-aware activity proposed in Figure 4-3 earlier. Now, the implementation details of each component of *SmartActivity* as shown in Figure 6-6 will be discussed.

**Context Model Interface (CMI)**

Since a *SmartActivity* always resides inside a *SmartSequence* activity and therefore is automatically integrated to a context model, a CMI is needed to query this attached context model.

CMI comprises of following two components:
1. *ContextVariables*.
2. *Context Fetch / Update Service*.

**ContextVariables**

*ContextVariables* of a *SmartActivity* is a set of variables defined in the attached context model and which are relevant to the decision making process of that particular *SmartActivity*. A dialog based property editor is being provided to Add or Delete Context variables from a *SmartActivity*. The *SmartActivity*’s context fetch service queries the context model at runtime and automatically updates values of all the *ContextVariables*.

**Figure 6-6: Realization of context aware activity architecture in CAWD**
These updated values are then used by *Adaptation Rule-set* to select a workflow from the Workflow Pool. Since the implementation of *ContextVariables* property editor as well as the context fetch service depends upon the format of the attached context model, for each type of context model representation, i.e. ontology or relational database, a separate *SmartActivity* is developed.

The set of *ContextVariables* is implemented as a *Dictionary* collection of a user-defined data type *ContextVars*. The sole purpose of using a *Dictionary* type is to be able to access this collection within the rule-set using *String* indices rather than the cumbersome integer indices. *ContextVars* contain all the information required to retrieve a variable from the context model into the *SmartActivity*.

For the purpose of this discussion, it is assumed that the context model is a relational database and in that case, *ContextVars* include information such as *variable name, variable type, database path, database table name* etc. If the attached context model is a relational database, the property editor for *ContextVariables* is essentially a SQL query builder. To retrieve a variable from a database, it is needed that primary key value(s) within the SQL query should be supplied. Since these primary keys effectively act as the IDs to access a group of context variables related to a particular entity and are passed to a *SmartActivity* as arguments, they are named as *ContextGroupIDArgs*.

To further explain how to pass the *ContextGroupIDArgs* to a *SmartActivity*, first let us understand a little about the WF4.0 data model. The WF4.0 data model is composed of three concepts: *variables, arguments, and expressions*. *Variables* represent the storage of data and *arguments* represent the flow of data into and out of an activity. *Arguments* are bound (assigned a value) using *expressions* that may reference *variables*.

Apart from *variables* and *arguments*, an activity can contain *properties* (e.g., *ContextVariables* and *Rule-set* are properties of *SmartActivity*) that can be used to pass some data to that activity.

The main issues regarding *ContextGroupIDArgs* are:

1. In order to update the values of *ContextVariables* at runtime, one need to access these *ContextGroupIDArgs* within the *Execute* method (this method is called by workflow runtime to execute an activity) of the *SmartActivity*.

2. One needs to be able to assign values to these *ContextGroupIDArgs* at runtime.

3. Since one cannot predict the number and data types of *ContextGroupIDArgs* that a *SmartActivity* might need, they cannot be defined inside the definition of *SmartActivity*.

None of the three approaches, i.e. using *variables, arguments* or *properties* is enough to address all of the issues discussed above.

*Variables* are storage locations for data that are declared at design time as part of the definition of a workflow. But the problem with a *variable* is that it can only be accessed in an *expression* at design time and are unavailable within the code of the *Execute method* of an activity even if the *variable* is defined within the scope of the activity. This restriction makes *variable* unsuitable to be used as a context group id.
This leaves us with \textit{properties} and \textit{arguments} since both can be accessed within an activity’s \textit{Execute} method and both can be assigned values at runtime using \textit{expressions}. But unlike \textit{variables}, one cannot define \textit{properties} or \textit{arguments} of an activity at workflow design time; instead these are defined within an activity’s code and thus are unable to address the issue of unknown number or data types of context group ids.

Considering all the above limitations, a workaround was sought and fortunately WF4 provides some features that enabled us to achieve our goal. Keeping in mind that a \textit{property} of an activity can be edited at workflow design time, following WF4 features were used:

- \textbf{DynamicArgumentDialog}: a \textit{ContextGroupIDArgs} property of type \texttt{Dictionary<string,argument>} is defined and the property editor used \textit{DynamicArgumentDialog} provided in WF4 to edit the elements of this dictionary at design time. As shown in Figure 6-7, a \textit{DynamicArgumentDialog} enables us to define any number of arguments and define their Name, Direction, Type and a default Value using \textit{expressions}. This solves the problem of unknown number or the data type of Context Group IDs.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure6-7.png}
\caption{SmartActivity’s \textit{ContextGroupIDArgs} editor}
\end{figure}

- \textbf{CacheMetaData Method}: To be able to access the \textit{arguments} defined within the \textit{ContextGroupIDArgs} at runtime, the workflow runtime was explicitly asked to make them available. This is where \textit{CacheMetaData} method comes into play. \textit{CacheMetaData} creates and validates a description of the activity’s \textit{arguments}, \textit{variables}, child activities, and activity delegates. Since \textit{ContextGroupIDArgs} are defined as a \textit{property} and not as an \textit{argument}, the default implementation of \textit{CacheMetaData} is not enough. To describe the \textit{arguments} defined within \textit{ContextGroupIDArgs} to workflow runtime, the \textit{CacheMetaData} was overrode and included following code snippet,

\begin{verbatim}
protected override void CacheMetadata(NativeActivityMetadata metadata)
{
    //define a RuntimeArgument corresponding to each Argument within
    //PrimaryKeyArgs, bind them together, and the add the //RuntimeArgument
\end{verbatim}
// to the activity metadata

foreach (string argumentKey in this.PrimaryKeyArgs.Keys)
{
    Argument argument = this.PrimaryKeyArgs[argumentKey];
    RuntimeArgument runtimeArgument = new RuntimeArgument(argumentKey, argument.ArgumentType, argument.Direction);
    metadata.Bind(argument, runtimeArgument);
    metadata.AddArgument(runtimeArgument);
}

As shown in Figure 6-8, ids defined in ContextGroupIDArgs are made available via a combo box in the ContextVariable property editor and can be used in the WHERE clause of the generated SQL query.

![Figure 6-8: SmartActivity's ContextVariables editor](image)

The property editor also lists all the current values of the database field that is defined as the primary key in the selected database table.
**Context Fetch / Update Service**
A service is written within the SmartActivity that fetches the context variables at runtime and also updates the variables after the execution of SmartActivity. Since this service depends upon the type of the context model attached, a separate SmartActivity with customized service code is required for integrating with different type of context models.

**Pool Workflow Arguments (I/O Arguments)**
The same strategy that was adopted to pass ContextGroupIDArgs to the SmartActivity at runtime was also used to pass values to arguments in WP workflows. SmartActivity defines a PoolWorkflowArgs property of type Dictionary<string,argument>. Upon each addition of a workflow in WP, new arguments are automatically added in this dictionary object. DynamicArgumentDialog, therefore, in this case is not needed to add or remove the arguments from PoolWorkflowArgs. However, DynamicArgumentDialog was used as PoolWorkflowArgs property editor for the sole purpose of binding workflow arguments using the expression editor provided in it. Validation errors are raised to prevent users from accidentally adding or removing an argument from the PoolWorkflowArgs collection.

**Adaptation Rule-set Reference**
As discussed in the proposed framework, rules for runtime adaptation are defined as a group called rule-set. To maintain the properties of flexibility, extensibility, evolvability and reusability, rule-sets are defined explicitly in rule-set repository and only a reference of this rule-set is passed to the SmartActivity at design time so the rule-set is available for editing at design time and for evaluation at runtime.

C# provides a Ruleset class which, in turn, is a collection of objects of class Rule. WF3.5 introduced these classes along with other support classes to validate and execute a RuleSet. A comprehensive rule-set editor was also introduced in WF3.5. This powerful feature of WF is used to let a user define rules in if-then-else format and then these rules are evaluated at runtime to invoke workflows from the WP. Figure 6-9 is a snapshot of the rule-set editor provide in WF.
Workflow Pool (WP) Reference
The WP, like adaptation rule-set, is defined explicit to the workflow definition in order to provide the same properties of flexibility, extensibility and evolvability. A reference of this WP provided to the SmartActivity enables editing of the WP at design time and makes it accessible at runtime.

The SmartActivity’s WP is implemented as a list of objects of a user defined class named “workflows”. The properties defined in the workflows class are

1. A string containing Workflow Name.
2. A string containing Workflow Path.
3. A floating point number to indicate the relative priority of the workflows.

A workflow pool editor (shown in Figure 6-10) is provided in CAWD to help the developer edit the WP during workflow design. When a new workflow is added to the WP, it requires a name and path of the workflow. The priority of every new workflow added has a default value of NULL. A developer can define priority of a selected workflow relative to another workflow in the pool by simply specifying if the selected workflow has either higher or lower priority than the other. A priority number is
automatically assigned to both the workflows depending upon the relative priorities of remaining workflows in the pool.

![Workflow Collection Editor](image)

**Figure 6-10: CAWD4 workflow pool editor**

**Placeholder Activity (PA)**

The Placeholder Activity component of *SmartActivity* is responsible for invocation of the sub-workflows selected as a result of the associated rule-set evaluation. If the rule-set evaluation selects no workflow at all, the PA automatically selects the NULL workflow from the workflow pool which is basically a *Do Nothing* workflow. Moreover, if two or more workflows are selected then the PA invokes each workflow according to their relative priority. In case the priorities of two or more workflows are same, the workflows are executed in parallel otherwise sequential.

WF4 classes of *WorkflowInvoker* and *WorkflowApplication* make workflow invocation a simple task that does not need any special mechanism to be placed in PA.

A snapshot of the *SmartActivity*’s designer interface is shown in Figure 6-11.
6.5.5 Adding Constraints
WF4 allows adding validation constraints to an activity. These constraints may range from a restriction on minimum length of activity’s display name to the activity’s relationship with other activities in the workflow. WF4 provides the following three helper activities that allow the developer to validate relationships between activities.

*GetParentChain:* Provides the collection of all activities that belong to the parent of the current node.

*GetChildSubtree:* Provides the collection of all activities that belong to the sub-tree of the current node, excluding the current node.

*GetWorkflowTree:* Provides the collection of all activities in the same tree as the current node.

Currently one constraint to the `SmartActivity` has been added. This constraint named `Parent Constraint` imposes a restriction that a `SmartActivity` must have at least one `SmartSequence` activity in its parent chain. This is necessary because the context model, the adaptation rule-set repository, and the workflow pool have been integrated to the `SmartSequence` activity; and a `SmartActivity` can access these only through a parent `SmartSequence` activity.

Similarly other constraints may easily be added like `Predecessor Constraint` or `Successor Constraint` and let the designer of context aware workflow choose at design time which constraints to activate and also which activity type should be the mandatory predecessor or successor of a particular `SmartActivity`.

6.5.6 ContextRetrievalActivity
Our experience with working of context-aware workflows tells us that often times one needs to access the context model just for the sake of retrieving some context information and does not necessarily use this information for a workflow selection. Hence, another activity in CAWD4 is developed that is called `ContextRetrievalActivity`. Like `SmartActivity`, this activity has two properties that are needed for context retrieval i.e. `RequiredContextVariables` and `ContextGroupIDArgs`. In addition to these properties an `out argument` of type `SortedDictionary<string, object>` is defined that makes the
values of the required context variables available in the form of \((\text{variable\_name}, \text{value})\) pair. Figure 6-12 shows the designer interface of a \textit{ContextRetrievalActivity}.

![ContextRetrievalActivity](image)

\textbf{Figure 6-12: CAWD4 ContextRetrievalActivity}

### 6.6 CAWD 3.5 Implementation Details

Prior to CAWD4, an earlier version as CAWD3.5 was developed using WF3.5. It was discontinued because with the advent of WF4.0, the Workflow Foundation got completely revamped and WF3.5 became obsolete.

The basic architecture of CAWD3.5 and CAWD4 is similar and, only those parts of CAWD3.5 will be described whose implementation was remarkably different from that of CAWD4.

#### 6.6.1 Re-hosted Workflow Designer

There are a number of classes and services in WF3.5 which a developer can use to build a custom re-hosted design environment.

\textit{WorkflowView} class provides a design surface which renders a visual representation of a process flow. It offers a rich set of user interface functionality needed by the activity designers for rendering and for responding to various Windows-generated events.

\textit{DesignSurface} class implements what the user perceives as a designer. \textit{DesignSurface} is the user interface the user manipulates to change design-time features.

\textit{WorkflowDesignerLoader} class is used to load the designer with the contents of a workflow markup file and save any changes to the state of the designer back to the markup file.

Services like \textit{MenuCommandService}, \textit{IEventBindingService} and \textit{IToolBoxService} are used to implement the Workflow menu commands, bind events defined in activities to event handlers and to provide a toolbox of activities respectively.

Figure 6-13 shows the re-hosted workflow designer in CAWD3.5 where a standard workflow designer is augmented with \textit{the Xaml Editor Pane}. 

84
Workflow validation is a process where a developed workflow model is verified for structural integrity of the process, and is evaluated for correctness and completeness in a syntactic manner. Workflow languages use proven workflow validation techniques that are well documented in the research literature. These techniques normally use a particular formalism like petri-nets or graphs to validate a workflow design. Since established validation methods are already implemented in most workflow languages, focus is made on reusing these methods rather than devising a new one.

In Workflow Foundation (WF), compiling a workflow automatically validates a workflow structure and return errors in case a workflow is incomplete or ill defined. Therefore, in CAWD3.5, Workflow validation makes use of the WorkflowCompiler, WorkflowCompilerParameters and WorkflowCompilerResults classes available in the .Net framework.

The WorkflowCompiler class represents a compiler for the workflows that are developed for .Net framework. These workflows may be developed using C#, Visual Basic code or XAML markup. The Compile member method of a WorkflowCompiler object compiles...
the workflows and takes two input parameters; a String array containing a list of workflows that are to be compiled and a WorkflowCompilerParameters object.

The WorkflowCompilerParameters is a class that represents the parameters that are required by the WorkflowCompiler during the compilation process. There are a number of compiler parameters that can be set using WorkflowCompilerParameters member properties but the most significant are as follows,

- ReferencedAssemblies: A StringCollection containing the names of reference assemblies that are referenced by source workflow to compile. For example, if a workflow uses custom activities defined in a dll file then that dll should be added in this collection
- LibraryPaths: Another StringCollection that contains the names of directories under which compiler looks for referenced assemblies
- OutputAssembly: A String telling the compiler the name of the output assembly

The returned argument of the WorkflowCompiler’s Compile method is an object of class WorkflowCompilerResults representing the results of workflow compilation. The most important results that are returned are as follows,

- CompiledAssembly: An Assembly object representing the compiled assembly
- Errors: A CompilerErrorCollection object indicating the errors and warnings resulting from compilation, if any
- Output: A StringCollection representing the output messages of the compiler

6.6.3 Context-Aware Activity (CA)

The major difference between the CAWD3.5’s and CAWD4’s CA is that of Placeholder Activity while the basic architecture remains same.

Placeholder Activity (PA)

For the purpose of PA, three choices were identified that are possible in WF3.5, each with its own pros and cons, these are

1. Use Invoke Workflow Activity.
2. Use empty Activity and then use dynamic workflow update feature.
3. Use Local Services to communicate with the host and Spawn new workflows.

InvokeWorkflow Activity

Since WP is a set of xoml only workflows, the first idea is to invoke a workflow using InvokeWorkflow Activity already available in WF3.5. A CA queries the context model integrated in host CWA and tells the Invoke Workflow Activity in PA which workflow to run. The advantage of using this approach is its simplicity; once an activity is selected from the pool it is compiled at runtime and then submitted to the InvokeWorkflow Activity to run. As discussed next, InvokeWorkflow activity approach can also be handled by the dynamic workflow update, so it was chosen not to implement this technique.
**Dynamic Update**

WF3.5 provides a powerful feature of dynamic workflow update, i.e. a workflow can update itself at runtime and can add any number of activities and in any sequence. A CA that uses dynamic workflow update uses an empty Placeholder Activity. After one or more workflows are selected from the WP, the PA is dynamically populated with workflows at runtime. The changes thus made are then passed to the root CWA because only root workflow activity can apply changes to the workflow. A method is implemented in CWA for the purpose of updating the workflow with the changes passed on by the CA. This approach is more powerful than the *InvokeWorkflow* Activity approach, since it can add multiple activities in varying order at runtime. Furthermore, this approach can be used to imitate the *InvokeWorkflow* approach by adding *InvokeWorkflow* activity to PA at runtime and then passing the selected activity to it. Through this method, activities will run synchronously and an asynchronous activity execution can be achieved by using *InvokeWorkflow* activity as described above. Just one shortcoming of using dynamic workflow update has been that if a synchronously invoked activity halts for some reason, the parent workflow will wait indefinitely for it to finish and therefore halts itself.

**Local Services**

A third approach that uses local services can solve this shortcoming although it is more expensive in terms of implementation and coding. Local service is a mechanism used for communication between host and running workflow. This mechanism is used to ask the host to run a workflow. The PA in this case consists of a number of activities as shown Figure 6-14 below.

![Figure 6-14: A local service context aware activity](image-url)
The local service that was defined provides an interface to both the executing workflow and the host for communication. The service consists of four methods and four corresponding events that these methods generate. Figure 6-15 shows the methods and corresponding events defined in the local service. The first activity in PA is the \textit{CallExternalMethod} Activity. This activity is used to call the \textit{SpawnWorkflow} defined method in the local service. The host receives the event generated by \textit{SpawnWorkflow} method and spawns the required workflow.

![Diagram showing communication between host and workflows through local service](image)

\textbf{Figure 6-15: Communication between host and workflows through local service}

The next activity in PA is a \textit{Listen Activity} which listens for the three events, i.e. \textit{SpawnedWFCompleted}, \textit{SpawnedWFTerminated} and \textit{SpawnedWFSuspended} from the host. The Listen Activity is composed of four branches; three of them to handle the events from host plus the fourth one to handle the Timeout event. It is possible for the parent to specify a timeout so that it does not have to wait indefinitely for a workflow to complete; this solves the problem of indefinite waiting faced when dynamic workflow update approach is used.

The disadvantage of this approach is that it puts an additional burden on the host to spawn new workflows and then report the status back to the parent.

\section*{6.7 Summary}

In this chapter, implementation details of our proposed framework and architecture are presented in the form of a tool i.e. CAWD. Windows Workflow Foundation classes for this implementation are used. Windows is a popular platform for workflow users and developers, so the implementation which is available in the form of a standalone complete package could be used for the development of context-aware applications by a
huge community of developers. CAWD lets you design a context-aware workflow by providing necessary support for workflow definition, context integration and adaptation in a very usable way. An important characteristic of CAWD is the support for process evolution, which is essentially not considered a part of context-awareness in workflows but actually is considered very important by workflow developers.
Chapter 7

Example Scenarios, Results and Discussion

7.1 Introduction

In this chapter, two highly diverse classes of example scenarios are introduced to understand the effectiveness of the proposed context-aware workflow applications design and management approach. The examples discussed below show the use of context-aware workflow applications in both high dynamic smart environments and less dynamic but evolvable traditional processes. Context-aware applications like e-Health, intelligent navigation etc. are frequently referred in literature and some existing approaches are suitable for these applications only. This thesis presents a different example of Smart Agriculture which is very useful in improving farming and its related concepts. In our research center, Center for Research in Ubiquitous Computing (CRUC), work is being carried out on a similar project which uses the context-aware workflows embedded within the smart applications.

7.2 Smart Agriculture

Optimal and profitable use of available land and water are of prime importance for the survival of ever increasing human population. Moreover, agriculture is traditionally a labor intensive work and requires a lot of manpower. With the advancement in technology, one can now automate the agricultural processes in such a way that they require lesser man power as well as make optimal use of land and water resources.

Workflows have been quite successful for service automation in various environments and are suitable for automating agriculture processes too. In order to optimize the agricultural processes, one need to take certain context information into account like soil moisture content, crop type and crop growth stage etc. This requirement of context
awareness makes the “Context-aware Workflows” best possible option for automation and optimization of agricultural processes.

Let us consider a scenario depicted in Figure 7-1. An agricultural land where a farmer cultivates different types of crops is populated with sensors that continuously gather various types of context information. The sensors include temperature sensors, humidity sensors, water level sensors, soil moisture sensors etc. These sensors are connected wirelessly to a context management server located in a central control room. The context management server maintains a context model and continuously updates the context entities with the latest information from sensors. A study of smart sensors commercially available is made by us. Extracted from the Table 1 of our research paper (Aqeel-ur-Rehman et al. 2011), a simplified Table 7-1 shows few example smart sensors that provide more than single context entity value. Apart from the sensed information, certain other information is also added in the context model either manually or through some software agent. This manual or derived information includes the crop type, crop age, farmer profile etc.

Figure 7-1: An illustration of smart agriculture
Table 7-1: Example of sensors used in smart agriculture

<table>
<thead>
<tr>
<th>Type of Sensor</th>
<th>Context Entities Sensed</th>
<th>Example Sensor:</th>
<th>Reference:</th>
</tr>
</thead>
</table>
| Soil          | • Temperature  
                • Moisture  
                • Rain/Water Flow  
                • Water Level  
                • Salinity       | Hydra Probe II Soil Sensor | www.stevenswater.com |
| Leaf/Plant    | • Photosynthesis  
                • Moisture  
                • Hydrogen  
                • Temperature | CI-340 Hand-Held Photosynthesis | www.solfranc.com |
| Weather       | • Temperature  
                • Humidity  
                • Wind Speed  
                • Wind Direction | CM-100 Compact Weather Sensor | www.stevenswater.com |

There are actuators like water sprinklers or alarms spread out in the agricultural zones. These actuators are also controlled remotely by various services accessible to the workflows running in the control room.

In the central control room, context management server ensures the accessibility of context model, both to a context-aware workflow developer designing workflows in CAWD as well as to the host running a context-aware workflow.
7.2.1 Smart Agriculture Context Model
The smart agriculture context model could be developed using RDBMS or other approaches as discussed in chapter 2. However, an ontology for smart agriculture is introduced here (Aqeel-ur-Rehman and Shaikh 2011). Ontology is a tool that is widely used for context modeling. Ontology not only enables us to define concepts related to a particular domain but allows definition of relationships between those concepts as well. This ability to relate different concepts in its definition makes ontology an ideal tool for modeling contexts.

Figure 7-2 illustrates an ontology-based context model for smart agriculture.

Although, the context data acquisition, dissemination and management are not within the scope of the thesis, a possible sensor network architecture is illustrated to capture the ontological context model in Figure 7-3.
The ONTAgr ontology scenario presented in Figure 7-2 (Aqeel-ur-Rehman and Shaikh 2011) is divided into two main parts: system ontology and domain ontology. Domain ontology is then further divided into two parts named Core and Services.

The System ontology contains definitions of concepts that are exclusive to a smart agriculture system. These include the definition of sensors, actuators and communication medium among other concepts that are used in such a smart system.

Core part of domain ontology defines concepts that are commonly related to agriculture, like Crop, Soil, Pests, Field and Stresses.

Figure 7-3: A possible sensor network architecture for smart agriculture
The other part of domain ontology is again related to smart agriculture and defines services that a smart system calls in order to perform different tasks. These services include irrigation service, pests control and fertilization.

Context entities can be divided into three categories namely Sensed, User supplied and Derived. The ONTAgri ontology contains all three types.

**Sensed Context**
As the name suggests, this type of context information is acquired directly through sensors. Examples in our scenario include soil moisture, temperature and humidity.

**User Supplied Context**
There is some context information that is provided by a user to the system. These may include crop type, sowing date etc.

**Derived Context**
The advantage of modeling context as ontology is that the relations can be defined between different sensed or user supplied context entities resulting in derived context information. There may be derived context entities like average soil moisture, average relative humidity and average temperature that provide much more meaningful information than just values of current soil moisture, humidity or temperature. Derived Context entities that are at a higher level of abstraction and involves more complex relations than just averaging may also be defined. Table 7-2 shows some such examples of derived context entities and rules or formulae for their derivation.

For our example scenario, above context model would need to be extended and some new attributes like CropType, CropAge and CropGrowthStage should be added to the context model.

### Table 7-2: Example relationships for context derivation in smart agriculture

<table>
<thead>
<tr>
<th>Context Entity</th>
<th>Rule / Formula for Derived Context Entity</th>
</tr>
</thead>
<tbody>
<tr>
<td>CropAge</td>
<td>Current Date – Sowing Date</td>
</tr>
<tr>
<td>SoilMoistureLevel</td>
<td>IF AvgSoilMoisture &lt; 30% THEN SoilMoistureLevel (\leftarrow) BelowThreshold</td>
</tr>
<tr>
<td></td>
<td>ELSE IF AvgSoilMoisture &lt; 50% THEN SoilMoistureLevel (\leftarrow) Normal</td>
</tr>
<tr>
<td></td>
<td>ELSE IF AvgSoilMoisture &lt; 70 THEN SoilMoistureLevel (\leftarrow) High</td>
</tr>
<tr>
<td></td>
<td>ELSE IF AvgSoilMoisture &gt; 70 THEN SoilMoistureLevel (\leftarrow) Flooded</td>
</tr>
<tr>
<td>CropGrowthStage</td>
<td>IF CropType = Wheat AND CropAge &lt; 0 THEN CropGrowthAge (\leftarrow) BeforePloughing</td>
</tr>
<tr>
<td></td>
<td>IF CropType = Wheat AND CropAge &gt; 30 days AND CropAge &lt; 45 days THEN CropGrowthAge (\leftarrow) Flowering</td>
</tr>
<tr>
<td></td>
<td>IF CropType = Wheat AND CropAge &gt; 45 days AND CropAge &lt; 60 days THEN CropGrowthAge (\leftarrow) GrainDevelopment</td>
</tr>
</tbody>
</table>

...
7.2.2 Workflow Representation
There may be a number of workflows defined for a smart agriculture space like Irrigation workflow, Pests control workflow and Fertilization workflow. A traditional workflow representation of each of these may be developed as depicted in the following Figures.7-4 and 7-5

The Figure 7-4 is a part of traditional irrigation workflow. Since every crop type has different irrigation requirements for different stages of their life, a workflow should be able to handle each crop type separately.

It is evident that, using multiple choice constructs, a traditional workflow too can cater for the context information like presence of soil moisture level or current crop stage etc. but the workflow starts becoming complicated as more context information is incorporated in it and eventually becomes almost incomprehensible.

Moreover, it is not humanly possible to define every possible context scenario in a workflow at design time. There will always be a chance that a new context scenario will arise in the future resulting in change of workflow.

Other agricultural processes like fertilization may similarly be defined as traditional workflows.
Figure 7-4: A Traditional irrigation workflow
Figure 7-5: A Traditional fertilization workflow

Figure 7-6 is the context-aware workflow representation of the irrigation workflow presented above along with the attached workflow pool. The relevant context entities and the rule-set for selection of appropriate workflow are given in Table 7-3.

Figure 7-6: A context-aware irrigation workflow
Table 7-3: Context entities and adaptation rule-set for smart irrigation workflow

<table>
<thead>
<tr>
<th>S.No</th>
<th>Irrigate Activity Adaptation Rule-set</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td><strong>Condition:</strong> CropName = Rice &amp;&amp; SoilMoistureLevel = Normal &amp;&amp; Rain = False</td>
</tr>
<tr>
<td></td>
<td><strong>Action:</strong> Select Moderate Irrigation service</td>
</tr>
<tr>
<td>2.</td>
<td><strong>Condition:</strong> CropName = Rice &amp;&amp; SoilMoistureLevel = BelowThreshold &amp;&amp; Rain = False</td>
</tr>
<tr>
<td></td>
<td><strong>Action:</strong> Select Heavy Irrigation service</td>
</tr>
<tr>
<td>3.</td>
<td><strong>Condition:</strong> CropName = Rice &amp;&amp; SoilMoistureLevel = BelowThreshold &amp;&amp; Rain = True</td>
</tr>
<tr>
<td></td>
<td><strong>Action:</strong> Select Light Irrigation service</td>
</tr>
<tr>
<td>4.</td>
<td><strong>Condition:</strong> NOT(CropName = Rice) &amp;&amp; SoilMoistureLevel = Flooded</td>
</tr>
<tr>
<td></td>
<td><strong>Action:</strong> Select Drain service</td>
</tr>
</tbody>
</table>
| 5.   |  **Condition:** CropName = Wheat && SoilMoistureLevel = Normal &&  
|      | (CropGrowthStage = BeforePloughing || CropGrowthStage = Flowering || CropGrowthStage = GrainDevelopment) && Rain = False |
|      | **Action:** Select Light Irrigation service                                                        |
| 6.   |  **Condition:** CropName = Wheat && SoilMoistureLevel = Normal && NOT(CropGrowthStage = BeforePloughing || CropGrowthStage = Flowering || CropGrowthStage = GrainDevelopment) && Rain = True |
|      | **Action:** Select Drain service                                                                  |
| 7.   |  **Condition:** CropName = Wheat && SoilMoistureLevel = BelowThreshold &&  
|      | (CropGrowthStage = BeforePloughing || CropGrowthStage = Flowering || CropGrowthStage = GrainDevelopment) && Rain = False |
|      | **Action:** Select Moderate Irrigation service                                                     |
| 8.   |  **Condition:** CropName = Wheat && SoilMoistureLevel = BelowThreshold && NOT(CropGrowthStage = BeforePloughing || CropGrowthStage = Flowering || CropGrowthStage = GrainDevelopment) && Rain = False |
|      | **Action:** Select Light Irrigation service                                                        |
| 9.   |  **Condition:** CropName = Beans && SoilMoistureLevel = Normal &&  
|      | (CropGrowthStage = Flowering || CropGrowthStage = PodDevelopment) && Rain = False               |
|      | **Action:** Select Light Irrigation service                                                        |
7.2.3 Workflow Execution
Both traditional and context-aware workflows were executed simultaneously on different machines in order to examine the effect of different context conditions on each of them.

7.2.3.1 Selection of an Activity
A traditional workflow selects an appropriate activity by taking a certain path of execution based on the context information. The paths and the conditions for selection of a path are defined within the workflow. For example, consider a scenario with following values of relevant context entities,
- Crop type is Wheat.
- Soil moisture level is Flooded.

In our example traditional workflow, there is a path defined for such context values which leads to the selection of Drain service.
A context-aware workflow selects an appropriate activity or workflow from the workflow pool based on the rules defined on relevant context entities. Therefore, in case of the context values given above, rule no. 4 in the associated rule-set of our example context-aware workflow would select the Drain workflow from the workflow pool which is the same as in case of traditional workflow.

Now consider another context condition where context entities have following values,
- Crop type is Beans.
- Crop growth stage is VegetationGrowth.
- Soil moisture level is Normal.
- It is raining.

The path defined in the non-context aware workflow for the above context condition will decide to do nothing since it does not take into account the information about weather. However, since the context-aware workflow does check the weather condition
before selecting a workflow, it will select the Drain workflow as per rule no. 10, instead of NULL as selected by non-context-aware workflow which is incorrect.

7.2.3.2 Deletion of an Activity
A traditional workflow cannot delete an activity at runtime. It is possible that a path defined for a particular context condition does not contain any activity, but if it does contain an activity, that activity will execute in any case. Unlike a traditional workflow, a context-aware workflow can delete an activity based on some context information. Take following context conditions for example,

- Crop Type is Wheat.
- Crop Growth Stage is Flowering.
- Soil Moisture Level is BelowThreshold.
- It is raining.

In these context conditions, the traditional workflow will always select Light Irrigation workflow. Whereas due to the fact that it is raining, the context-aware workflow will select no workflow since none of the conditions specified in the rule-set will hold true. This will consequently execute the NULL workflow which effectively means deletion of an activity which is the desired result.

7.2.3.3 Addition / Insertion of an Activity
During the execution of a workflow, there may be a scenario where it is desirable to add or insert another activity into the workflow in response to certain context conditions. For example, in the fertilize workflow it may be decided to add another activity that places order for more fertilizer based on the current quantity of fertilizer available.

In a traditional workflow, there is no provision for adding or inserting an activity into the workflow at runtime. However, in case of context-aware workflows, the ability to define rule-set in a way that it selects more than one workflow and the fact that a context-aware activity allows editing of rule repository as well as workflow pool during execution enables us to add or insert an activity during workflow execution.

To add an OrderFertilizer1 activity, one have to add a workflow named OrderFertilizer1 in workflow pool and set its priority so that it is run prior to the UseFertilizer1 activity. Then add a rule to the rule-set as follows,

```
Condition: ServiceSelected = UseFertilizer1 && Fertilizer1AvailableQuantity < fertilizer_threshold
Action: Add OrderFertilizer1 service
```

7.2.3.4 Process Evolution
Let’s assume that the farmer now realizes that in case of wheat in the flowering stage, if soil moisture is below the threshold, a more proper method of irrigation is misting rather than normal irrigation and for that he has to make changes in the running workflows.
Implementing change in a traditional workflow is very costly for two reasons,

1. Because of the complexity, it is very difficult and time consuming even to find the exact place where a change is required let alone to ensure that such a change is reliable and does not affect any other part of the workflow.
2. Changes in a traditional workflow cannot be made during workflow execution and therefore every running instance needs to be halted before implementing the change.

In case of context-aware workflow, the farmer can make the changes very easily and without having to stop the running instances. To call the Misting service for a flowering wheat crop, he needs to add the new service to the workflow pool called Misting. Then a rule is added to the adaptation rule-set is as follows,

```
Condition: CropType = Wheat && SoilMoistureLevel = BelowThreshold &&
            CropGrowthStage = Flowering

Action: Select Misting service
```

Rule 8 in the rule-set table also need to be changed so that it does not include flowering stage in its condition clause.

All the new context-aware workflows as well as every running instance of the context-aware workflow adapts to the change in the irrigate activity seamlessly and efficiently.

### 7.2.4 Benefits of Context Aware Approach

The above example thoroughly demonstrates the benefits of a context aware approach of workflow design. These benefits can be classified into two groups,

1. Benefits in workflow representation
2. Benefits in workflow execution

#### 7.1.4.1 Benefits in Workflow Representation

As described earlier, a non-context aware workflow is able to handle contextual information using multiple choice constructs but the workflow design becomes more and more complex as the amount of context information increases making it almost incomprehensible in the end.

A context-aware workflow, on the other hand, incorporates the contextual information very neatly in the form of human readable rule-sets without affecting the workflow’s visual representation at all.

#### 7.1.4.2 Benefits in Workflow Execution

Once a workflow is in execution, our proposed context-aware framework offers more advantages as compared to the traditional workflow.
The explicit definition of rule-set file and workflow pool allows us to change the rules and edit the workflow pool anytime during the execution of a workflow without the need of restarting them while in a non-context aware workflow any change is impossible without stopping the running workflows.

These added features in context-aware workflows provide us the much desired properties of flexibility, extensibility, runtime adaptability, reusability, usability and process evolution.

**7.3 Loan Sanction Process**

In order to show the variety of application areas that can benefit from context workflows, consider an example that is entirely different from the smart agriculture example discussed above. Apart from pervasive environments like smart agriculture farms where sensors are the major source of context information, there are other scenarios where most of the context information is derived or manually supplied. In a pervasive environment, the change in context is usually very frequent while in other case the change is less frequent in comparison.

A loan sanction process is an example of such application where most of the context information is derived or manually supplied. Suppose the government decides to offer loans to needy young individuals in order to help them set up a business of their own. This will make these young people self-reliant as well as a useful citizen with valuable contributions to the country’s economy. Workflows are commonly used for processes like these since they inherently involve well defined steps or activities.

A typical loan sanctioning workflow may include the following steps,

- Collect applications.
- Check applicant’s eligibility.
- Issue notification of approval or rejection.
- Payment/Repayment.

Check applicant’s eligibility activity may include following sub-activities.

- Check if applicant exceeds age limit.
- If applicant is single, check if the applicant’s income exceeds limit.
- If applicant is married, add applicant’s and spouse’s total income.
- Check if total income exceeds limit.

**7.3.1 A Simple Loan Sanction Context Model**

The context model in this case is relatively simple and is modeled as a relational database as shown in Figure 7-7.
7.3.2 Workflow Representation

Figure 7-8 shows a traditional loan sanction workflow. To avoid complexity, it is assumed that all the steps to be atomic activities except when checking applicant’s eligibility for loan. But despite of this simplifying assumption, the workflow is still quite complicated.
Figure 7-8: A traditional loan sanction workflow
Now compare the above traditional workflow with the context-aware workflow shown in Figure 7-9. It is obvious that the context-aware workflow is much simple and easy to understand. Part of the workflow pool is also shown below.

![Workflow Diagram]

**Figure 7-9: A context aware loan sanction workflow**

In Figure 7-9, the possible sub-workflows in the workflow pool that a particular context-aware activity may select are highlighted. If an activity is required by more than one context-aware activity it can easily be reused in both. For example, an activity named *Send a Copy to Record section* may be required by both the *Collect Application* activity as well as by *Issue Notification* activity as depicted in the Figure.

Relevant context entities and the adaptation rule-set for the check eligibility activity are shown in Table 7-4.
Table 7-4: Context entities and adaptation rule-set for smart loan sanction workflow

<table>
<thead>
<tr>
<th>S.No</th>
<th>Check Eligibility Activity Adaptation Rule-set</th>
</tr>
</thead>
</table>
| 1.   | **Condition:** Age > AgeLimit && Approved = false  
      | **Action:** Select DeclareNotEligible          |
| 2.   | **Condition:** Age < AgeLimit && MaritalStatus = Single && Income < IncomeLimit && Assets < AssetLimit  
      | **Action:** Select DeclareEligible             |
| 3.   | **Condition:** Age < AgeLimit && MaritalStatus = Single && (Income > IncomeLimit || Asset > AssetLimit) && Approved = false  
      | **Action:** Select DeclareNotEligible          |
| 4.   | **Condition:** Age < AgeLimit && MaritalStatus = Married && (Income + SpouseIncome) < IncomeLimit && (Assets + SpouseAssets < AssetLimit)  
      | **Action:** Select DeclareEligible             |
| 5.   | **Condition:** Age < AgeLimit && MaritalStatus = Married && ((Income + SpouseIncome) > IncomeLimit || (Assets + SpouseAssets) > AssetLimit) && Approved = false  
      | **Action:** Select DeclareNotEligible          |

Check Eligibility Relevant Context Entities:

- Age
- Income
- Assets
- Marital Status
- SpouseIncome
- Spouse Assets
- IncomeLimit
- Approved
- AgeLimit
7.3.3 Workflow Execution

The difference between selection and deletion of an activity in each type of workflow is same as discussed in the smart agriculture example. Since processes like this are usually very well defined and planned before hand, a traditional workflow is less likely to encounter an unknown context scenario and resulted in stalling. However, these kinds of processes are likely to evolve with time and needs the compensation in the workflow model without disturbing the actual model and running instances.

7.3.3.1 Selection of an Activity

Suppose the loan policy sets the eligibility age to 35 years, total income limit of 50,000 per month and total asset’s value limit to be 5,000,000. Now if a person has following contextual information:

- Age is 30 years
- Monthly income is 25,000
- Marital status is Married
- Spouse Income is zero
- Assets value is 3,000,000
- Spouse assets value is 3,000,000
- Loan Approved is false

The traditional workflow of our example (figure 7-8) will traverse a path that will eventually end at DeclareNotEligible activity because the total asset value after adding the spouse’s assets will exceed the allowable asset limit of 5,000,000.

The defined context-aware workflow (figure 7-9) will select the same workflow, i.e. DeclareNotEligible on the basis of rule no. 5 defined in the rule-set (table 7-4).

This shows that both traditional and context-aware workflows will eventually end up with the same desired result. But the selection in inflexible in traditional workflow and requires modifications in workflow model for any new change; whereas by changing rules and adding more sub-workflows in workflow pool, the outcome of selection activity may be changed in case of context-aware workflows.

7.3.3.2 Deletion of an Activity

A context-aware workflow can make use of other context information during runtime to decide if an activity needs to be executed at all or not; a feature that is not possible in traditional workflows.

Consider a scenario where a context entity named Approved indicates whether a person is already approved by any higher authority and need not be checked for eligibility. So if Approved is true, the context-aware workflow will select NULL workflow since no rule in the defined rule-set will hold true. The traditional workflow will always select DeclareNotEligible workflow until and unless it is changed to incorporate the Approved context entity.

7.3.3.3 Insertion of an Activity

Suppose at a later stage, management decides to inform the director about every person who is declared as eligible. There is no way to add a new activity in a non-context
aware workflow without stopping the currently running workflows and redesigning the workflow again.

In a context-aware workflow, a workflow can be added without affecting the running workflows. In the above a workflow named *InformDirector* can be added to the attached workflow pool and a rule can be added to add this workflow to the currently selected workflow. The new rule will be as follows,

```
Condition: WorkflowSelected = DeclareEligible
Action: Add InformDirector
```

### 7.3.3.4 Process Evolution
The real advantage of context-aware workflows in cases like this is when the process evolves. Since number of running workflows is usually large in such scenarios, therefore it is very costly and inconvenient to halt all the running workflows to implement a simple policy change.

Consider the case when government changes its policy and drops the requirement of including spouse’s income in the calculation of total income.

**Traditional Workflow**
In a traditional workflow, to implement such change in policy decisions means halting or terminating all running instances of the workflow and then make the change. Since number of running instances in cases like these may be very large, halting or terminating all of them just for making a simple change is very costly in terms of both time and money. Moreover, even after making such a costly change, one can never be sure that another such change will not occur in future.

**Context-Aware Workflow**
In case of context-aware workflow, this change in policy is easily achieved by changing the associated adaptation rule-set as shown in Table 7-5.

<table>
<thead>
<tr>
<th>S.No</th>
<th>Check Eligibility Activity Adaptation Rule-set</th>
</tr>
</thead>
</table>
| 1.   |  **Condition:** Age > AgeLimit  
      | **Action:** Select NotEligible              |
| 2.   |  **Condition:** Age < AgeLimit && Income < IncomeLimit  
      | **Action:** Select DeclareEligible         |
| 3.   |  **Condition:** Age < AgeLimit && Income > IncomeLimit  
      | **Action:** Select NotEligible             |
Since adaptation rule-sets are explicitly and independently defined, such a change does not require halting or terminating the running instances of the workflow which saves a lot of valuable time and money.

### 7.4 Discussion on Results

The focus of research presented in this thesis was the design of workflows for context-aware applications. Context-aware applications require a great degree of flexibility in their design so that they may be able to adapt to the dynamic and changing situation of the environment, system and user. By hard-coding these variability demanded by context-aware applications within the application limits its flexibility and makes the system difficult to develop and extend. Workflow technology on the other hand have proved its merits in the business application domain by its unique characteristics of explicit process representation and executing process goals by coordinating ordered set of activities that integrates human, applications and heterogeneous system resources under well defined policies.

It was observed and concluded in the thesis that workflow technology could be used for providing the immediate flexibility that a context-aware system demands by representing the various application logics as workflow processes explicit to the application. This characteristic makes context-aware applications very manageable as application variability is managed outside the application thus providing flexibility and extensibility. However, the adaptability that a context-aware application demands requires that these process logics must be flexible, dynamic and adaptable during and before runtime. This adaptability requirement serves as a serious impediment in using workflows for context-aware applications as traditional workflow models are very rigid and cannot be modified without halting the process execution and in general modified manually. The need is that this adaptation should be autonomous and without halting the execution for context-aware applications. So the need arises to develop workflow models that are flexible and adaptable enough to satisfy the requirements of context-aware applications. For this purpose various approaches were researched over the years, which are discussed in chapter 2 and chapter 3 in detail. However, these approaches are still far from maturity and some gray areas are needed to be addressed before effectively adopting the workflow technology for the design and development of context-aware applications.

The framework proposed in this thesis discusses in detail the complete scope of workflow-embedded context-aware applications with the desired extensions and modifications required to make an effective context-aware workflow application. However, our focus remained in the design of context-aware workflows such that their enactment poses no difficulties for any standard workflow engine. A context-aware activity is proposed as an additional workflow construct to model an abstract activity at build time and provide mechanisms within the activity to dynamically bind this abstract activity with a predefined concrete activity selected based on latest context-based decisions. For this purpose a workflow system is appended to have components like context model, workflow pool and adaptation rule-sets whose working and integration was discussed in detail in chapter 4 and chapter 5.
The research presented in this thesis was directed to devise a framework or methodology that addresses the adoption of workflows in context-aware applications. As indicated through simulated testing in chapter 5 and discussion of example scenarios in this chapter, it is proved that the desired flexibility and adaptability have been achieved with other additional benefits like reusability, extensibility and usability enhancements.

The usability and ease of developing workflows for context-aware applications is increased by the development of context-aware workflow designer tool CAWD that provides all the features and integration necessary for designing a context-aware workflow. Earlier works along the lines of extending workflow language constructs were primarily focused on BPEL and YAWL which are open source.

In our survey of workflow products being used, it was found that a large community is using Microsoft workflow products. So to address this neglected community, it was decided to develop the design tool CAWD using Windows Workflow Foundation classes. The extensions and build time and runtime feature sets that is proposed in our approach are forthright which are difficult to handle within the environment of Microsoft Visual Studio. Hence, the standalone CAWD environment was developed that not only provides an integrated environment for design but also validates and executes the developed workflows from within its environment.

In contrary to almost all previous approaches of context-aware workflow applications where a specialized workflow engine or a wrapper around standard workflow engine were used to enact the workflow models, modifications and extensions are made transparent to the engine in our approach thus enabling the continuous use of already available products.

The flexibility in our proposed approach is manifolds as it uses all the extensions explicit to the workflow model and context-aware application. The flexibility to add, delete or edit sub-workflows in workflow pool and rule-sets in adaptation rule-set repository has been proved in chapter 5 and chapter 7. The running instances need not to be flushed or terminated to be adapted to the new changes and they keep on executing based on new definitions. However, the ability to rollback is not provided. The running instances only adapt to new definitions from the point forward where they were executing at the time of change. The new workflow instances work fine with the new modifications. The aspect of rollback has not been covered in any contemporary research.

All the adaptation operations during or before runtime i.e. selection, deletion and addition are provided and tested with applications as discussed in chapter 5 and chapter 7.

Very few context-aware workflow systems allow the handling of ad-hoc changes and evolutionary changes side by side in an approach. The proposed approach, however, successfully demonstrated that this gap can be bridged and the workflows developed with our designer could be used in a wider scope to cater fast context-changing dynamic applications and less-frequently changed evolving applications as well.
A couple of test smart applications having context-aware workflows designed with CAWD have been developed, as discussed in chapter 5 and chapter 7, and results are discussed in detail with simulated data. It has been observed that context-aware applications designed as context-aware workflows provide autonomous adaptation for contextual changes by exploiting addition, deletion, and selection of appropriate sub-workflows. Moreover, process evolution is proved to be handled as changes in rule-sets and workflow pool during design and runtime.

At the end of section 2.4, a definition of context-aware workflow is presented that highlighted the additional requirements for a conventional workflow to become context-aware. As discussed in this section, these requirements along with additional benefits of reusability and maintainability are provided in workflows through our approach. For further clarity, a checklist of operations provided by our approach to make workflows useful for context-aware applications is tabulated below.

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Operation</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Selection of Relevant Activity within a workflow at runtime</td>
<td>✓</td>
</tr>
<tr>
<td>2</td>
<td>Deletion of Unnecessary Activity within a workflow at runtime</td>
<td>✓</td>
</tr>
<tr>
<td>3</td>
<td>Addition of one or more extra activities to fulfill the process goal at runtime</td>
<td>✓</td>
</tr>
<tr>
<td>4</td>
<td>Editing along with addition and deletion of adaptation rules at build time and runtime</td>
<td>✓</td>
</tr>
<tr>
<td>5</td>
<td>Addition and deletion of sub-workflows in workflow pool at build time and runtime</td>
<td>✓</td>
</tr>
</tbody>
</table>

There are, however, limitations in our work and further work is required to make it more effective. These limitations and possible suggestions are addressed in the next chapter.

### 7.5 Summary
This chapter presented a couple of scenarios to show the effectiveness and working of our proposed approach for context-aware workflow design and management. These two
examples are the cases of frequently changing contexts environment and occasionally changing context processes. In the last section of this chapter, a short discussion is made on the effectiveness of context-aware workflows and results of our thesis were highlighted. It is proved that the additional requirements of workflows to become context-aware, i.e. dynamic adaptation operations and handling of evolutionary changes, have been fulfilled and are verified through simulated testing and case studies presented in the thesis.
Chapter 8

Conclusion and Future Work

8.1 Introduction
This chapter concludes the thesis by presenting major contributions and highlighting the future work in the context of our research.

8.2 Contributions
The design and development of context-aware applications are difficult because of inherent dynamic and flexible nature of these applications. Since workflows provide separation of concerns between process and application logic, it is proposed that the variability of application domain and necessary flexibility and adaptability should be handled within workflows and these workflows then used in applications requiring context-awareness. This approach makes the extensions and maintenance of context-aware applications easy. However, the adoption of workflow technology in context-aware applications presents several challenges. These challenges and issues are investigated in this thesis and a framework is presented for handling issues of design and management of context-aware workflows.

The contribution of the research work may be listed as:

- A local survey of workflow product users and developers was conducted to determine their inclination towards a more flexible and adaptable workflow solution. It was observed that current workflow products provide little support for flexibility and adaptability and are not suited for context-aware workflow developments in their current form.

- A literature survey of related technologies and existing work in context-aware workflows was done that enabled us to identify the key issues and challenges in context-aware workflow adaptations.
• A generic framework for context-aware workflow design and management is presented. It discusses the modifications in traditional workflow systems along with the necessary extensions of components required for context-awareness. The framework is designed in such a way that it supports both evolutionary and ad-hoc changes in workflow models and instances.

• The framework is governed by the principles of flexibility, adaptability, extensibility, autonomy, reusability and usability. These principles are realized through explicit representation of context, adaptation rule-set and workflow pool. Context-Activity architecture is presented that describes the interaction and working of modified and additional components. As a major modification in workflow representation, specially designed context-aware activities are introduced that are equipped with necessary functionalities for context-aware adaptation before and during execution.

• The proposed framework concepts and context-activity architecture are realized in the form of a design tool which is named as Context-aware Workflow Designer (CAWD). The implementation is made using Windows Workflow Foundation classes that enable the use of this designer tool for developing context-aware applications in .Net enabled windows platform.

• Two diverse example scenarios are discussed in detail that elaborate the design and execution procedures for context-aware applications and show the feature set provided with our approach for needed flexibility and adaptability in workflows to be used within context-aware applications.

8.3 Future Directions
The research has great potential and more work can be carried out in the following future directions:

• The approach presented in the thesis is reactive and results in adaptation operations of addition, deletion and selection of activities. However, a predictive approach of handling context changes that may result in re-organization and re-orchestration of workflow could be adopted for more flexibility.

• The design issue of roll back of activities to previous states could be handled to enhance the domain of context-aware workflow use.

• The proposed method uses the context entities as provided by the context model assuming that complex context management is done by the Context Management System using methods like relational queries in RDBMS or ontological relations in ontology based context models. However, such management of complex or fused context could be done within context-aware activity for much effective use.
• Keeping the same architecture with slight modifications, an auto-compose workflow model could be designed by enhancing functionality in CAWD.

• Advanced features necessary for the use of context-aware workflows in organizations and field projects like provenance, security and authentication for change needs to be addressed.

• Interesting and emerging software applications other than context-aware applications like e-governance and social computing could be developed with our approach for more effectiveness.

• Existing scientific workflow development tools like Taverna workflow management system (http://www.taverna.org.uk) may also utilize the concept of context-awareness in workflows for representing variability in applications like bioinformatics, chemistry and astronomy.

8.4 Summary
This chapter described the contributions of submitted research towards the intended and desired objectives. The research has the potential of further enhancements, and few of the issues and directions are highlighted for future research work in similar area.
References


Appendix A

Using CAWD to Develop a Context-aware Process Model

Context-aware workflow designer (CAWD) is a very easy to use tool for designing context-aware as well as traditional workflows. It provides the necessary interfaces for the extensions made in workflow systems to make them context-aware. Moreover it provides special activities termed as context-aware activities that are required to make a context-aware workflow.

Following is a step-by-step walkthrough of how to use this tool to develop a context-aware workflow.

1. Select a *SmartSequenceActivity* and drop it on workflow canvas as shown in figure A-1. This activity acts as a container where a context-aware workflow is developed. In figure below, a red circled exclamation mark is present in this *SmartSequenceActivity* in workflow canvas. This shows that the activity is not properly defined.
The `SmartSequenceActivity` requires properties like (i) Context Model (ii) Rule Repository (iii) Workflow Pool as shown in figure A-2. The context model property requires a path to a Context Model developed for the context-aware application and which the `SmartSequenceActivity` binds to all the activities resides within the container of this smart activity.

---

**Figure A-1: Adding a smartsequenceactivity**

![Figure A-1: Adding a smartsequenceactivity](image)

**Figure A-2: Properties of a smartsequenceactivity**

![Figure A-2: Properties of a smartsequenceactivity](image)
(3) Figure A-3 shows how to select Rule Repository Workflow Pool path as required in figure A-2. You may select an existing file or create a new one.

![Figure A-3: Selection of Workflow Pool /Rule-set repository](image)

(4) When the context model is successfully attached, a context-relevance space shows the details of context entities for perusal as shown in figure A-4. Similarly a workflow pool is made visible when a successful reference is made after attaching a workflow pool file as shown in figure A-5. It may be noted that initially we have just “nullWorkflow” attached to a new workflow pool. You may add/remove more sub-workflows as shown later.

![Figure A-4: Context relevance space](image)
(5) Sub-workflows in workflow pool may be added and removed with the help of workflow editor as shown in figure A-6. When a workflow is added, we may add an existing workflow or made a new one using another instance of CAWD. We may also set the sub-workflow priorities from here, which are useful in certain runtime adaptation operations.
(6) Now is the time to add activities within the SmartSequenceActivity container. A SmartActivity is a context-aware activity that holds the necessary mechanisms for runtime adaptation as shown in figure A-7. It requires information like:

(i) **Context Group ID Args** refers to the outside parameters required for this activity from host application and is supplied through a dialog like shown in figure A-8

(ii) **Relevant Context Variables** refers to the context-entities selected from context model and is used within this context-aware activity to decide for selection of suitable activity from workflow pool at runtime. A comprehensive dialog is provided by CAWD for this operation as shown in figure A-9

(iii) **Ruleset** refers to the rule-set that contains the necessary rules based upon the relevant context entities and are used to evaluate the selection of appropriate sub-workflow from workflow pool. The rules may be added, deleted and edited using the rule editor as shown in figure A-10

(iv) **Child Workflows I/O Args** is optional information that may be passed from a context-aware activity to a selected sub-workflow pool if needed.

![Figure A-7: SmartActivity and its related information](image)
Figure A-8: SmartActivity Context Group ID Args dialog

Figure A-9: SmartActivity Context variables management dialog
(7) Finally a workflow may be validated for errors and warnings. Figure A-11 and figure A-12 shows unsuccessful and successful validation of workflows respectively.
The CAWD could be used while a workflow model’s instances are running and rules and workflow pool may be modified at runtime, giving it an edge over other workflow products.