

WestminsterResearch

<http://www.westminster.ac.uk/westminsterresearch>

**Self-adaptation via concurrent multi-action evaluation for
unknown context**

Nimalasena, A.

This is an electronic version of a PhD thesis awarded by the University of Westminster.
© Mr Mudiyanseelage Nimalasena, 2017.

The WestminsterResearch online digital archive at the University of Westminster aims to make the research output of the University available to a wider audience. Copyright and Moral Rights remain with the authors and/or copyright owners.

Whilst further distribution of specific materials from within this archive is forbidden, you may freely distribute the URL of WestminsterResearch: (<http://westminsterresearch.wmin.ac.uk/>).

In case of abuse or copyright appearing without permission e-mail repository@westminster.ac.uk

SELF-ADAPTATION VIA CONCURRENT MULTI-ACTION EVALUATION FOR UNKNOWN CONTEXT

ASANGA NIMALASENA

A thesis submitted in partial fulfilment of the
requirements of the University of Westminster
for the degree of Doctor of Philosophy

June 2017

TABLE OF CONTENTS

List of Figures and Tables.....	vi
List of Source Code Listings.....	viii
List of Abbreviations.....	ix
Acknowledgement.....	x
Declaration of Publications	xi
Abstract	xii
1. Introduction	1
1.1 Ubiquitous and Pervasive Computing.....	3
1.2 Context and Context-Awareness	7
1.2.1 Summary of Context Definitions.....	11
1.3 Context-Aware Applications.....	11
1.3.1 Context-Aware Application Development	12
1.4 Self-Adapting Computer Systems.....	14
1.5 Motivation	15
1.5.1 Single Context – Single Action	15
1.5.2 Single Context – Multi-Action	18
1.5.3 Multi-Context – Multi-Action	20
1.6 Objectives and Contributions of the Research	21
1.7 Application Domains	22
1.8 Direction of Research and Thesis Organization.....	22
2. Related Work	24
2.1 Autonomic Systems	25
2.2 Self-Adaptive Systems	30
2.3 Self-Adapting Techniques in Context-Aware Applications	33
2.3.1 Manual Adaptation	35
2.3.2 User Defined Adaptation	35

2.3.3	Learning-Based Adaptation	41
2.3.4	Policy-Based Adaptation	45
2.3.5	Statistics-Based Adaptation	48
2.4	Taxonomy of Context-Aware Adaptation Techniques	51
3.	Generic Framework for Self-Adaptation via Concurrent Multi-Action	
	Evaluation	54
3.1	Motivation	55
3.2	Generic Framework Description – Meta Level.....	56
3.2.1	Context System	57
3.2.2	Inference Systems	62
3.2.3	Action System.....	66
3.3	Formal Modelling	70
3.3.1	Goal Specification.....	70
3.3.2	Action Refinement.....	70
3.3.3	Concurrent Multi-Action Execution	72
3.3.4	Action Outcome Evaluation.....	73
3.4	Application of Framework	73
3.5	Blueprint for Framework Implementation	75
4.	Context-Aware Approach for Experiment-Based Database Performance	
	Tuning	78
4.1	Background	79
4.2	Current Approaches	82
4.3	Formal Modelling	85
4.4	Implementation	89
4.4.1	Context System	90
4.4.2	Inference Systems	93
4.4.3	Action System.....	98
4.4.4	Runtime Execution of Context-Aware Application.....	103

4.5	Experimentation	109
4.5.1	Experiment Setup 1	109
4.5.2	Results.....	111
4.5.3	Experiment Setup 2.....	114
4.5.4	Results.....	115
4.6	Conclusion of the Case Study	119
5.	Context-Aware Approach for Determining the Threshold Price in Name-Your-Own-Price Channels	121
5.1	Background	122
5.2	Name-Your-Own-Price Channel.....	123
5.2.1	Name-Your-Own-Price Strategies	126
5.2.2	Pricing Strategies for Time Sensitive Items	128
5.3	Problem Identification.....	130
5.4	Formal Modelling	136
5.5	Implementation	140
5.5.1	Context System	140
5.5.2	Action System.....	141
5.5.3	Inference System.....	142
5.5.4	Runtime Execution of the Context-Aware Application.....	143
5.6	Experiment Setup	144
5.6.1	Results.....	147
5.6.2	Results for Pessimistic Test Case	147
5.6.3	Results for Optimistic Test Case	148
5.7	Conclusion of the Case Study	150
6.	Future Directions.....	151
6.1	Need for Energy Tuning in Processors	152
6.2	Using the Proposed Framework for Thermal Energy Modelling.....	153

7. Conclusion	156
References	162
Appendix A - Context Acquire Class	176
Appendix B - OWL + SWRL Definitions	180
Appendix C - Knowledge Base Class	188

LIST OF FIGURES AND TABLES

List of Figures

Figure 1. Single-Context Single-Action Model	36
Figure 2. Single-Context Multiple-Action Model.....	44
Figure 3. Taxonomy of Context-Aware Adaptation Techniques.....	51
Figure 4. System Architecture of the Context-Aware Framework	56
Figure 5. Action Space Creation	72
Figure 6. Application of Framework to Achieve Context-aware Adaptation.....	74
Figure 7. Blueprint for Generic Framework Implementation	76
Figure 8. DB CPU Usage During Performance Regression	80
Figure 9. Hourly CPU Usage on the DB Over 3 Month Period.....	80
Figure 10. Adaptation of Context-Aware Framework for Experiment-Based Performance Tuning.....	85
Figure 11. Pre-Populated Knowledge Base	103
Figure 12. Known Context Encountered by Context-Aware Application	104
Figure 13. DBS Updated With Value from Knowledge Base	105
Figure 14. System Detects an Unknown Context	106
Figure 15. Concurrent Multi-Action Evaluation Being Carried Out	107
Figure 16. DBS Updated by the Best Adaptive Action	108
Figure 17. Knowledge base Updated with New Context Information.....	108
Figure 18. Experimentation Setup with an On-site Database	110
Figure 19. CPU Usage of the On-site DBS Without and With Context-Aware Adaptation.....	111
Figure 20. Experimentation Setup with an Off-site Database.....	114
Figure 21. CPU Usage of the Off-site DBS Without and With Context-Aware Adaptation.....	115
Figure 22. Average CPU Usage for Each Configuration Settings	119
Figure 23. NYOP Decision Flow	124
Figure 24. Locality of Hotels Relative to Event Venue	132
Figure 25. Price Variation of Ibis Hotel's rooms.....	133
Figure 26. Price Variation of DoubleTree by Hilton Hotel's Rooms	134
Figure 27. Revenue for a Coach Travel Company between 2009 - 2010.....	135

Figure 28. Flow Diagram of the Concurrent Multi-Action Evaluation System.	136
Figure 29. Run Time Execution of Context-Aware Application in an NYOP Channel	143
Figure 30. Successful Bid Count for the Test Case Where Unknown Context Results in Pessimistic Bid Values	147
Figure 31. Yield for Pessimistic Test Case	148
Figure 32. Successful Bid Count for the Test Case Where Unknown Context Results in Optimistic Bid Values	149
Figure 33. Yield for the Optimistic Test Case	150

List of Tables

Table 1. Summary of Context Definitions	11
Table 2. Experimental Workload Mix in the DBS.....	111
Table 3. Experimental Workload Mix in the Off-Site DBS.....	115
Table 4. Knowledge Base for Thermal Modelling.....	154

LIST OF SOURCE CODE LISTINGS

Listing 1. Context Class for DB Performance Tuning Case	90
Listing 2. Context Acquisition and Context Change Detection.....	92
Listing 3. OWL Class Definition for OLTP Workloads	94
Listing 4. OWL Class Definition for DSS (OLAP) Workloads.....	94
Listing 5. OWL Class Definition for Mix Workloads	95
Listing 6. SWRL Rules for OLTP and OLAP Workloads.....	95
Listing 7. OWL Class Definition for Workload Type	96
Listing 8. OWL Model and Pellet Reasoner Loading.....	97
Listing 9. Dynamic Inference of Workload Types.....	98
Listing 10. Updating of Knowledge Base	98
Listing 11. Private Workbench for Concurrent Action Execution	101
Listing 12. Building of Action Space in Action Refinement Class	102
Listing 13. Context Class for NYOP Channel Case Study	141
Listing 14. Bid Value Evaluation in Action Class	142

LIST OF ABBREVIATIONS

API	Application Programming Interface
AC	Autonomic Computing
AS	Autonomic Systems
BN	Bayesian Network
CBR	Case-based Reasoning
CPT	Conditional Probability Table
DAG	Direct Acyclic Graph
DB	Database
DBA	Database Administrator
DBS	Database System
DSS	Decision Support System
HMM	Hidden Markov Model
JAX-WS	Java API for XML Web Services
MLN	Markov Logic Network
NYOP	Name-Your-Own-Price
OLTP	Online Transaction Processing
OWL	Web Ontology Language
PSO	Particle Swarm Optimization
RDBMS	Relational Database Management System
SLA	Service Level Agreement
SWRL	Semantic Web Rule Language
UML	Unified Modelling Language

ACKNOWLEDGEMENT

First and foremost a special thank you to my director of studies (DoS) Professor Vladimir Getov for his kind imparting of knowledge and guidance for the duration of the research. I would also like to thank my supervisor Dr. Alexander Bolotov and staff at the University of Westminster research office, who over the years facilitated many research related activities.

Secondly I would like to acknowledge with great gratitude my employer CodeGen Ltd, especially the directors Bharat and Harsha who over the years gave me freedom to pursue the research related activities without any hindrance.

Finally, and most importantly I would like to dedicate this thesis to the most important people in my life, *Amma*, *Thatha* and my wife Iliyana. I am forever indebted to their unwavering support and endless encouragements during this long and arduous journey.

DECLARATION OF PUBLICATIONS

Some parts of the work presented in this thesis have been published as research articles with the following details:

1. Nimalasena, A. and Getov, V., (2016). Context-Aware Approach for Determining the Threshold Price in Name-Your-Own-Price Channels. *in: Context-Aware Systems and Applications. Springer.* pp. 83-93.
2. Nimalasena, A. and Getov, V., (2015). Context-Aware Framework for Performance Tuning via Multi-Action Evaluation. *in: 2015 IEEE 39th Annual Computer Software and Applications Conference (COMPSAC) Taichung, Taiwan IEEE.* pp. 318 – 323.
3. Nimalasena, A. and Getov, V., (2014). Performance Tuning of Database Systems Using a Context-Aware Approach. *in: Proc. 9th International Conference on Computer Engineering & Systems (ICCES) Cairo IEEE.* pp. 98 – 103.
4. Nimalasena, A. and Getov, V., (2013). System Evolution for Unknown Context through Multi-Action Evaluation, *in: Proceedings of 2013 IEEE 37th Annual Computer Software and Applications Conference (COMPSAC) IEEE.* pp. 271-276.

ABSTRACT

Context-aware computing has been attracting growing attention in recent years. Generally, there are several ways for a context-aware system to select a course of action for a particular change of context. One way is for the system developers to encompass all possible context changes in the domain knowledge. Other methods include system inferences and adaptive learning whereby the system executes one action and evaluates the outcome and self-adapts/self-learns based on that. However, in situations where a system encounters unknown contexts, the iterative approach would become unfeasible when the size of the action space increases. Providing efficient solutions to this problem has been the main goal of this research project.

Based on the developed abstract model, the designed methodology replaces the single action implementation and evaluation by multiple actions implemented and evaluated concurrently. This parallel evaluation of actions speeds up significantly the evolution time taken to select the best action suited to unknown context compared to the iterative approach.

The designed and implemented framework efficiently carries out concurrent multi-action evaluation when an unknown context is encountered and finds the best course of action. Two concrete implementations of the framework were carried out demonstrating the usability and adaptability of the framework across multiple domains.

The first implementation was in the domain of database performance tuning. The concrete implementation of the framework demonstrated the ability of concurrent multi-action evaluation technique to performance tune a database when performance is regressed for an unknown reason.

The second implementation demonstrated the ability of the framework to correctly determine the threshold price to be used in a name-your-own-price channel when an unknown context is encountered.

In conclusion the research introduced a new paradigm of a self-adaptation technique for context-aware application. Among the existing body of work, the concurrent multi-action evaluation is classified under the abstract concept of experiment-based self-adaptation techniques.

1. INTRODUCTION

Context awareness is a key component in pervasive computing. In simple terms, context awareness is the consideration of an entity's surrounding environment, its properties, other entities in the environment and interaction of these entities and their influence on the change of the environment's properties. In context-aware computing, the computer system's actions are determined by the context and changes in the context. As a consequence of a context change the context-aware system transition through context sensing, context inference, and action phases. A context-aware system does not concern itself with the causality of the context change but the reactive action it must execute in order to maximize the expected goals under the new context.

In this regard, there are several ways a context-aware system would select an action to execute as a result of context change. One way is for the system developers to encompass and embed all possible context changes and corresponding action relevant to the application domain into the context-aware application. When a context change occurs the context-aware application matches it to known context changes and chooses the corresponding action. This method works for small sets of context-change and action pairs and is not feasible when the system could encounter an unknown context. A system is said to have encountered an unknown context when it doesn't know of an action that would make it self-adapt such that the outcome is optimized for that particular context.

Inferences and adaptive learning techniques are used to address the issue of encountering unknown contexts. In a context-aware system employing such techniques, there exists a one-to-many relationship between the context change and action space. The system executes each action iteratively deemed relevant to

the sensed context change and evaluates the outcome and chooses the action which maximizes the expected goals. However, this iterative approach is not feasible when the size of the action space increases as it would result in the adaptive time being increased linearly.

The research project undertaken proposed a novel self-adapting context-aware framework which addresses the issues in the iterative approach through multi-action evaluation and self-adaptation. Under this framework, instead of one action, multiple actions are implemented, evaluated and compared concurrently to find the action that maximizes the expected goals. This concurrent evaluation of actions reduces significantly the time for self-adaptation.

This novel adaptive context-aware framework forms the key contribution of the research. Formal modelling of the framework has been completed and through implementation in two distinct domains, the usability of this novel approach has been demonstrated. The results and findings from each of these implementations have been validated through peer-reviewed publications presented at several conferences.

1.1 Ubiquitous and Pervasive Computing

The emergence of context-aware computing is rooted in the idea of ubiquitous computing, a term first introduced by Weiser [1] as a vision of how computers would be working in the 21st century. Not only the vision has been realized since then but also surpassed in some areas. However, a look back at the characteristics envisioned by Weiser in his paper is needed in order to understand how modern-day context-aware systems sprung up from the ideas of ubiquitous computing systems. One of the first characteristics Weiser mentions is that

“ubiquitous computers must know where they are”. This alludes to the location being a key factor in the interaction between users and the computer (or computer services). The location based aspect of ubiquitous computing was so widely used in context-aware systems creation that some definitions used the location as the only factor determining the context of that particular context-aware system. The definitions have broadened since then which will be looked at in subsequent sections.

The second aspect envisioned by Weiser refers to the specialist nature of ubiquitous computers. In his vision “ubiquitous computers will also come in different sizes, each suited to a particular task”. According to this statement ubiquitous computers were not expected to be general purpose computers that are capable of a multitude of functionalities.

The final characteristic mentions the technology required by the ubiquitous computing which, according to Weiser, “comes in three parts: cheap, low-power computers that include equally convenient displays, software for ubiquitous applications and a network that ties them all together”.

If these last two characteristics are looked at together, what emerges is the context-aware computing for the internet of things (IoT). A set of computing devices specializing in a particular task are connected and orchestrated to meet a computing need otherwise not possible.

The ultimate goal of Weiser’s vision was that people would utilize computing power as they would any other utility, without being aware of who, how, when and where it was generated. This idea of omnipresence computing power was termed pervasive computing. In 1996 Rick Belluzo of HP compared pervasive

computing power to electricity [2] and stated it is “the stage when we take computing for granted. We only notice its absence, rather than its presence”.

A formal definition of pervasive computing is given in [3] where it is defined as “the idea of making ‘computing power’ available anyplace, anytime in a uniform way so that it may be exploited for meeting the challenges faced by society”. Also, identified in this literature are four key areas of advancement needed in order for pervasive computing to fully materialize. They are computing, communication, cognition, and collaboration – called the “4Cs of pervasive computing”.

Much advancement has happened along those 4Cs and the pervasive computing has managed to influence many areas within the past decades from its inception as a vision for 21st-century computers. Some of the areas include ubiquitous knowledge workers [4], intelligent transportation [5], pervasive healthcare [6, 7], and context-aware appliances [8] which gives an indication of the varying landscape within pervasive computing.

The next leap in pervasive computing came with the emergence of smartphones. The key to realizing the full potential of pervasive computing was to achieve mobility. This had to be achieved without a reduction in the quality in terms of computing power while being mobile. What lacked during the early days of pervasive computing was the availability of compact and mobile yet powerful computational devices. This is envisaged by Abowd *et al.* [9] stating that “mobile or smartphone will usher in the real age of ubiquitous computing, While Moore’s Law will lead to impressive computational and storage properties for these devices, it’s their small form factor and constant connectivity that present the most intriguing, and sometimes worrying, possibilities.” This has certainly

become a reality and had led to pervasive service computing, where services are more personalized and customized depending on each individual's preference. An example of realization of the aforementioned vision would be a smartphone application giving location based services, where location is being one aspect of the context. Interestingly there's merging of new technologies to enhance the capabilities of these types of systems. Intel's CARS system is one such example [10] where big data technologies coupled with context-awareness serve drivers (mobile users) with coupons that are most relevant to the customers' immediate preferences onto the car navigation system without disrupting the navigation activities.

With these advances in the technology and support, the next steps in pervasive computing came in the form of service orchestration. With the use of service oriented architecture and web services the traditional model of pervasive computing was transformed into service oriented pervasive computing [11]. The characteristics of a service oriented pervasive computing system were described as "explicit description of user's activity, pervasive nature, semantics, P2P, trust-awareness, and wireless sensor networks". It could be noticed that with the use of semantics and trust-awareness this was an attempt to advance the later 2Cs (cognition and collaboration) in the 4Cs of pervasive computing. The first two of the 4Cs (computing and communications) are also rapidly advancing especially with the introduction of GPS as an embedded service. For a user to truly benefit from pervasive computing, the first two Cs are not enough. The supporting systems need to exhibit cognition - an understanding of the situation.

With the realization of mobility, the next technological barrier to overcome in the pervasive computing landscape is cognition (situation awareness) and

collaboration. Evolutionary system designs [12], evidence based pervasive systems [13], multi-agent design for enabling cognition through reasoning [14] are examples of distinct and parallel efforts taken in addressing the later 2Cs (cognition and collaboration) in pervasive computing.

With the advances in pervasive computing, it could be also expected some level of “undesirable outcomes to privacy, basic freedoms (of expression, representation, demonstration etc.), and even human rights” [15] to emerge. This would mean advances in other areas such as security and trust must also be considered along with the advances in pervasive computing.

In the past, both ubiquitous computing and pervasive computing were considered interchangeable terms due to the similarity of computing power being available everywhere, anytime. However, due to latest developments, these two terms are no longer interchangeable. In current trends, pervasive computing involves compact mobile devices which allow access to computing power anywhere. On the other hand, the ubiquitous computing avoids users having to use computers or devices at all. It acts as computing in the background providing computational power without users ever being aware of its existence. According to Wisner who coined the term ubiquitous computing, it is not the same thing as mobile (pervasive) computing, or a superset or a subset [16].

1.2 Context and Context-Awareness

With mobility becoming prominent in the pervasive computing, location became a key factor in defining software and its interaction with users and devices. The term “context-aware” was first introduced by Schilit and Theimer as a generalization of software which changes its characteristics based on

location. According to Schilit and Theimer [17] “location information is necessary for users and applications that want to query and interact with nearby devices and services. Such information also allows stationary clients to track moving objects. In general, location information enables software to adapt according to its location of use, the collection of nearby people and objects, as well as the changes to those objects over time. We use the term context-aware computing to describe software exhibiting these general capabilities”. This first attempt at defining context-awareness is limiting due to its use of location as the focal point of the context. If the software is not adapting based on location information, it is not considered a context-aware application. Any context-aware model based on this definition will not fit with the multi-action context-aware system as two actions associated with a single location would lead to ambiguity.

The other definitions expanded on this initial definition of “context-aware” by combining other aspects with location. Brown *et al.* [18] defined context as “location, time of day, the season of the year, temperature, and so forth.” Though this definition is broader than that of Schilit and Theimer, it still makes no assumption as to how each of these context aspects would influence the overall context either individually or in combination with each other. Ryan *et al.* [19] define context as “location, time, temperature or user identity” and context-awareness as the term “that describes the ability of the computer to sense and act upon information about its environment” i.e. the context information. Dey [20] not only considers the aforementioned aspects such as location and time of the day but the users’ emotional state, focus of attention, location and orientation, date and time, objects, and people in the user’s environment as well.

While some definitions are location-centric, other definitions are either user or user's environment centric. Hull *et al.* [21] define context as the "user's local environment" and the situation awareness as "ability of computing devices to detect, interpret and respond to aspects of the user's local environment". This definition seems to consider context and situation as synonyms. Franklin & Flaschbart [22] also define context as the situation of the user. Situation of the user is described as "what is happening at this moment" with the user. The moment is further differentiated based on the user's action, expected goals and responses from the system. These aspects could vary from one moment to another.

Brown [23] defined context to be the elements of the user's environment that the user's computer knows about. Brown provides "location, the adjacency of other objects, critical states, imaginary companions, time, and computer states" as examples of elements of a context.

Similar to the user-centric approach in the above definitions a few other definitions look at the context from the application-centric perspective. Ward *et al.* [24] view context as the state of the application's surroundings while Rodden *et al.* [25] define it to be the application's setting.

The aforementioned definitions fit well with certain cases but fail to give a generalized definition of context. The following definition by Schilit *et al.* [26] tries to address this limitation by defining context as a constantly changing execution environment which is a combination of computing, user, and physical environments. According to this definition key aspects of context are "where you are, who you are with, and what resources are nearby". Pascoe [27] defines context as a subjective concept that is defined by the entity that perceives it. In

short, it is to be the subset of physical and conceptual states of interest to a particular entity.

Herein lies the problem of defining what a context is, because what is a context is subjective to the user, application, environment or what the application or user is doing at that time. There is “no clear boundary which divides what is and is not context, but the most interesting kinds of context are those that humans do not explicitly provide” [28]. Dey and Abowd [29] conclude that all of the aforementioned definitions suffer from the problem of being too specific and give a new definition. In [29] Dey and Abowd define context as “any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves”. This definition gives the context-aware system developers much more freedom to decide what constitutes a context. Following this definition, an entity can bring in and remove from the context space any aspect that is not relevant to what is currently being perceived. This definition is accepted and used in this research work due to the flexibility it provides in defining a context.

Once the context is defined, Dey defines context-awareness as using “context to provide relevant information and/or services to the user, where relevancy depends on the user’s task” [29]. Although the context definition in [29] was used in this research project, the definition for the context-awareness was not. This is because it does not mention the adaptation of the system according to the context. In this research definition for context-awareness is expanded to include not only the using of context but also adapting to the context.

1.2.1 Summary of Context Definitions

Table 1 below gives a summary of the context definitions mentioned in previous section. It shows a concise view of context definitions and how the definitions evolved over time to include aspects other than location.

Table 1. Summary of Context Definitions

Reference	Context defined by
Schilit and Theimer [17]	Location
Brown <i>et al.</i> [18]	Location, time of day, the season of the year, temperature
Ryan <i>et al.</i> [19]	Location, time, temperature or user identity
Dey [20]	Location, date and time, users' emotional state, focus of attention, orientation, , objects, people in the user's environment
Hull <i>et al.</i> [21]	User's local environment
Flaschbart [22]	Situation of the user
Brown [23]	Elements of the user's environment that the user's computer knows about
Ward <i>et al.</i> [24]	State of the application's surroundings
Rodden <i>et al.</i> [25]	Application's setting
Schilit <i>et al.</i> [26]	Combination of constantly changing execution environment consisting of computing, user, and physical environments.
Pascoe [27]	Subset of physical and conceptual states of interest to a particular entity
Dey and Abowd [29]	Any information that can be used to characterize the situation of an entity

1.3 Context-Aware Applications

Similar to the definition in [29], some early definitions focused on the use of a context and called such applications (or computing) context-aware applications (or computing). Other such definitions include Hull *et al.* [21], Ryan *et al.* [19] and Pascoe *et al.* [27, 30] where context-aware computing is defined as the ability of computing devices to detect and sense, interpret and respond to aspects of a user's context and the computing devices themselves. Salber *et al.* [31]

define context-aware computing to be the ability to provide a flexible computing service based on real-time sensing of context.

What is missing from these definitions is the adaptation of the application to context. Dey *et al.* [32] introduce the notion of adaptation by defining context-awareness of a system as the work leading to the automation of a software system based on knowledge of the user's context. On the other hand, Ryan [33] defines context-aware applications to be applications that take input from the environment via sensors and allow users to select from a range of physical and logical options according to their current interests or activities. This appears to be a user-driven (manual) adaptation lacking in any automated self-adaptation.

The definitions [17, 18, 26, 34, 35, 36, 37, 38] define context-aware applications to be applications that dynamically adapt/change their behavior based on the context of the user and the application. Fickas *et al.* [13] define context-aware applications to be ones that monitor changes in the environment and adapt their operation according to predefined or user-defined guidelines. Finally, Brown [39] defines context-aware applications as applications that automatically provide information and/or take actions according to the user's present context as detected by sensors. An ideal context-aware application would be one that encompasses key features of all these definitions, where it detects context and changes in context and adapts automatically based on predefined or user-defined guidelines.

1.3.1 Context-Aware Application Development

From the earlier definitions, it is clear that an application must capture the context in order to be considered context-aware. The approaches to capturing the context to make an application context-aware could be classified into two

approaches. One is to use a context-aware tool support to define and capture the context relevant to the application.

CAPpella [40] is one such tool that allows context-aware application developers to define context by demonstration. Castelli *et al.* [41] further extend this to include augmentation-based learning. This method of context-aware application development captures the end-to-end information flow and associated contexts during the demonstration phase. Apart from these types of program-by-demonstration context-aware application development tools, the landscape consists of tools that facilitate some niche requirements in the context application development process.

Examples of these include tools to personalize user experience such as “Persona, a toolkit that provides support for extending context-aware applications with end-user personalization and control features” [42], tools related to context recognition in sensory equipment [43], tools for formal modelling of contexts [44], collaborative programming with rapid prototyping capabilities such as the OPEN framework [45].

The second approach is the context-aware design and implementation through programming. Under this approach, a generic framework or a concrete solution is designed and developed to address a problem in the domain (generic framework) or a specific problem (concrete solution) [46]. Methodologies used in the design and implementation draw from concepts in software engineering and adapt them to suit the context-aware domain. Some examples from context-aware programming landscape are an incremental approach to increasing context awareness in the system [47], similar to incremental development method in software engineering. Other methodologies include the use of web services,

model-driven architecture to develop context-aware web services [48] and semantic-based aspect-oriented programming for composing web services on the fly [49]. Other concepts adapted to context-aware systems include ontology-based context-aware systems [50], semantic-based context-aware systems [51] and intelligence-based context-aware systems [52].

1.4 Self-Adapting Computer Systems

As mentioned earlier, a system is context-aware if it adapts based on the current context. However, self-adaptation is not unique or limited to the context-aware systems. The concept existed on its own right before the concept of context-aware systems. According to [53] a definition of self-adaptive software was provided in a DARPA broad agency announcement as software that “evaluates its own behavior and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible.” As per this definition, the adaptation occurs in the behavior of the software rather than in the structure of the software.

Another definition is given by Oreizy *et al.* [54] in which it is stated that a “self-adaptive software modifies its own behavior in response to changes in its operating environment. By operating environment, we mean anything observable by the software system, such as end-user input, external hardware devices and sensors, or program instrumentation.”

Self-adapting systems are also associated with autonomic computing systems. But according to [55], there’s a key difference between the two, which is “self-adaptive software primarily covers the application and the middleware layers, while its coverage fades in the layers below middleware. On the other hand,

autonomic computing covers lower layers too, down to even the network and operating system”.

With these differences, two trends could be identified in the self-adapting system landscape with autonomic computing on one end dealing with the full stack from software to hardware [56,57,58,59,60,61] and self-adapting software dealing primarily on the software stack [62], both incorporating some common trends such as genetic algorithms and organic computing.

1.5 Motivation

The motivation for the research was rooted in addressing the limitation in existing self-adapting context-aware systems when it comes to using them in multi-context – multi-action scenarios. The existing self-adapting techniques could be broadly classified into two categories based on the context-action relationship. The two categories are single context – single action and single context – multi-action.

1.5.1 Single Context – Single Action

As a subdomain of pervasive computing, context-aware computing has been used widely to create smart environments. One common theme of these smart environments is that based on sensory data from one or more devices another device’s state is changed to bring the environment (in another word context) to a state that is most beneficial to the entity in that context. Three main areas could be identified in these types of context-aware frameworks – they are sensory data acquisition, context inference/management, and action. IBM’s MAPE-K (Monitor, Analyze, Plan, Execute, and Knowledge) loop reference model [196] is an example such a framework. The components of the MAPE-K loop could be

superimposed into the three main areas of sensing, inference and action of a context-aware application. However, a MAPE-K loop still depends on the system developers to formulate the event-condition-action (ECA) rules for self-adaptation [58] which makes it unsuitable for situation where unknown context could be encountered. ECA knowledge comes from human experts or other methods such as concept utility [197], Bayesian techniques [198] or reinforcement learning [100] which suffers from poor scalability when large number of ECA state changes exists.

Due to the nature of its application, a context-aware framework developed for these smart environments has an output action that is of two mutually exclusive states for a given context. In other words, the sensor data and action have a one-to-one relationship. For example, a context-aware application developed to control the ambient temperature in a room would have actions to turn on or off a fan or set the temperature of the air conditioning to one specific value Z when the ambient temperature is X degrees or between X and Y degrees.

It is not possible to have and neither does it make sense to incorporate into the context-aware framework actions such as: when the ambient temperature is X degrees turn on and off the fan or set the air conditioning temperature to P and Q degrees.

The challenge in this type of context-aware systems is that if a sensory data perceives an unknown context, then the system is incapable of deciding which course of action to take if the system developers have not embedded this knowledge in the system. For example, assume due to some freak of nature or some weather pattern that rarely occurs the ambient temperature is X^l and this

was not captured in the domain knowledge and therefore no action is defined for X^1 .

Two possible solutions could be envisaged to address this challenge. The simplest and commonly used (detailed in Chapter 2) is to add the new context to the application knowledge base. This may be fine for a system with few contexts and limited context value ranges but becomes infeasible and impractical when the system has to deal with a high number of contexts and when the probability of encountering unknown context is high due the dynamism and fluidity of the environment.

The other solution would be that the system must be capable of self-adapting. Taking the earlier example, the learning and self-adapting could be thought as:

```
if ambient temperature is  $X^1$  then
start with  $Y = Y^1$ 
label:
    set air conditioning temperature to  $Y$ 
    if not satisfactory
         $Y = Y^2$ 
        go to label:
    if satisfactory
        acquire domain knowledge
        { for  $X^1$  set air condition temperature to  $Y$ }
```

But this self-adaptation is a single action iterative process where each possible action is evaluated one at a time. The time to identify the best action is directly impacted by the number of actions in the action space thus it becomes infeasible when the action space becomes large as the time complexity would become $O(n)$.

This poses the first research challenge, which is, the existing context-aware systems developed for single context – single action environments such as the

ones developed for controlling appliances in smart environments cannot be adopted for multi-context – multi-action environments. Therefore, a new solution must be formulated that does not have the aforementioned limitations.

1.5.2 Single Context – Multi-Action

One possible solution to overcome the research challenge mentioned earlier is to evaluate multiple actions concurrently and find the most beneficial action to execute in the current context. Unlike smart environments, there are domains where when an unknown context is encountered it is possible to output two or more actions concurrently and then evaluate the outcome of each of these actions to arrive at the best action for the given unknown context. In these systems, it is possible to have a single unknown context value associated with two or more actions before converging to a single action. This approach eliminates the limitation associated with the use of an iterative method which evaluates all possible actions to find the best action as explained in the earlier section.

How such a multi-action context-aware system comes into use could be illustrated with the name-your-own-price (NYOP) application. NYOP is a strategy where the buyer suggests the price which he/she is willing to pay for goods or services without knowing the minimum threshold price T which is acceptable. Imagine a hotelier who sells his inventory through such an NYOP channel. If the decision to accept or reject a bid is solely based on the bid value then the hotelier won't have the fluidity to react to the demand uncertainty. In such cases, the decision to accept or forego depends on factors other than the bid value. These factors include the current occupancy rate and expected occupancy rate, which could be influenced by events that are scheduled to take part in the vicinity of the hotel such as conferences, festivals, sporting events, weather and

other seasonal information would be considered as well. These external factors influence the context in which the hotelier's NYOP system makes decisions on accepting or rejecting the bids.

For example, if a new event has been planned in the vicinity of the hotel and there is no historical data or knowledge to rely on (an unknown context), the context-aware NYOP system could be set up such that instead of having one threshold price T the system could have multiple threshold values T^1 and T^2 . It will dynamically evaluate which threshold value is resulting in higher yield. At the end of this dynamic experiment system will update the system's actual threshold price with the one selected through experimentation. Unlike the context aware models used in a smart environment, this approach does not have the constraint that each unknown context is associated with only one action. The hotelier's business model does not suffer when using this approach because "the retailer wants to obtain as much revenue as possible. In the case of perishable or time-dated items (such as Christmas trees or newspapers), the retailer would be willing to accept even a lower bid as long as those willing to pay more actually do pay more" [63].

Although concurrent multi-action execution and evaluation reduce the time complexity compared to single context- single action scenario by evaluating multiple actions in parallel, there is an issue when the number of context values increases. This leads to a high number of permutations to be considered in the context space and action space which poses several research challenges which are elaborated in the next section.

1.5.3 Multi-Context – Multi-Action

When a context-aware application considers multiple contexts, the output action correlates to a set of context attributes rather than to a single context value as in the previous case. A context attribute is an element of the context model describing the context [64]. Taking the earlier example of a hotelier selling his perishable inventory under an NYOP channel, the threshold price used could be influenced by many contexts. For example, a hotelier may decide the threshold price based on any upcoming events in the vicinity of the hotel, weather, and current occupancy rate. Each of these contexts would have multiple values which will be different to how it influences hotelier’s decision-making process. For example, considering the earlier contexts events could be conferences, gathering, and wedding each of these events has different characteristics which must be considered in the decision making. The weather could be defined with concrete parameters such temperature 27°C, humidity 60% or vaguely or using fuzzy terms such as “sunny day” or “mildly chilly day” etc. Permutations of these multiple contexts could be defined as below.

{Conference, Sunny day} → action space {A₁,A₂,...A_n}
{Wedding, {27°C, 60% Humidity},
20% occupancy} → action space {A_a,A_b...A_z}

Each permutation of these contexts would result in multi-action evaluation as the hotelier is unlikely to have encountered all the permutations. The hotelier has to decide on the best course of action whenever a new permutation of context values is encountered. As the number of permutations for the “context value - adaptive action” combination increases exponentially with an addition of each new context, an environment that has to employ multi-context – multi-action

systems faces several challenges and addressing these challenges is the core aim of this research.

The foremost challenge of the research is to reduce the number of action spaces associated with context attribute permutations. Without it, the implementation of a context-aware system using existing self-adapting models would lead to resource starvation and would also impact the system convergence on to a best adaptive action.

1.6 Objectives and Contributions of the Research

The primary objective of the research is to develop a framework for context-aware systems that finds the best adaptive action for unknown context through multi-action evaluation and self-adaptation. This objective addresses the problem of having to deal with a high number of “context-action” permutations that could occur in multi-context – multi-action cases. A formal model of the generic framework will be presented, which will make it possible for the framework to be used across multiple domains.

The research will contribute to the knowledge in the field of self-adapting context-aware systems. The novel generic framework envisaged solves the problems associated with multi-context – multi-action systems through the new approach of concurrent multi-action evaluation.

Through various use cases the research demonstrates that this new approach could be used by multiple domains to address multi-context – multi-action cases.

1.7 Application Domains

The research outcomes are applicable in the domain of self-adapting context-aware system, in the specific case of adapting to the situation when unforeseen contexts are encountered.

What the research delivers is a generic framework of which a concrete implementation could be done to meet the requirements of specific use cases.

During the research two use cases have been implemented in two separate domains and a third use case is being worked on. One use case is in the domain of database performance tuning, where the context-aware approach is used for experiment-based performance tuning when performance regresses due to unknown context.

The second use case shows how a context-aware approach could be used to adjust the threshold prices in name-your-own-pricing channels for unknown contexts.

The third use case is in the domain of thermal modelling. The adoption of the proposed context-aware framework is being investigated for workload specific thermal modelling in high-end processors. Implementation of the framework would be used to dynamically adjust the CPU core count based on encountered workloads to meet the energy/power consumption goals.

1.8 Direction of Research and Thesis Organization

The first step in this research effort is to study and critically analyse the challenges and limitations in the existing self-adapting context-aware application domain for the special case of multi-context – multi-action. The result of this action is Chapter 2 which examines the existing self-adapting techniques used by

the context-aware applications. After studying the existing methods, the conclusion was that they lack the capability and are not a viable solution for multi-context – multi-action cases. In the end, a taxonomy of self-adapting methods used by context-aware systems was created, which allowed a clear positioning for this research among the existing body of work.

Following a critical analysis of existing work, the next step was modelling and formulating a generic framework meeting the research objectives. The proposed generic framework does not assume any domain or application specific constructs or constraints. It acts as a blueprint for a domain specific implementation of the user's choosing. The framework description and formal model are given in Chapter 3.

Chapter 4 describes the first use case which was in the database performance tuning domain. With this implementation of the framework, it was demonstrated how a context-aware approach could be used for experiment-based performance tuning, especially when the reason for performance regression is unknown. The body of work in this chapter is based on published work in [65, 66].

Chapter 5 describes the second concrete implementation of the framework, which is for dynamical adjusting of threshold pricing in name-your-own-price channels. The contents of this chapter are based on the work published in [67, 68].

Chapter 6 gives future directions of the research, including current work being carried out in the area of thermal modelling.

The thesis concludes with Chapter 7 with a summary of research findings and a critical analysis of the contribution of the research.

2. RELATED WORK

2.1 Autonomic Systems

As mentioned in the introduction the concept of self-adaptation is not unique or limited to the context-aware application domain. It has been used in other application domains as well. The concept was rooted in the properties of software (or hardware) agents, which Wooldridge and Jennings [69] first identified as being:

Autonomy: - Agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state.

Social Ability: - Agents interact with other agents (and possibly humans) via some kind of agent-communication language.

Reactivity: - Agents perceive their environment, and respond in a timely fashion to changes that occur in it.

Pro-activeness: - Agents do not simply act in response to their environment; they are able to exhibit goal-directed behavior by taking the initiative.

IBM took this further by being the first to introduce the concept of self-managing computer systems on its autonomic computing (AC) manifesto [70]. The idea was that autonomic computing systems would self-manage themselves with minimum human input or interference. The autonomous nature of the system was analogous to the body's autonomic nervous system which controls key functions without a person being aware or conscious of it.

The main concept behind IBM's autonomic systems was that complex computing systems should also have autonomic properties that would allow them to maintain and optimize themselves, thus reducing the interference from human

users. In this regard, IBM introduced four properties a self-managing autonomic system must have, popularly known as self-* (self-star) or self-CHOP. These are

self-CHOP = {self-configuration, self-healing, self-optimization, self-protection}

Self-Configuration: - refers to the fact that an autonomic system must be able to configure itself towards achieving its high-level goals based on what is desired rather than how it's achieved. For example, this means an autonomic system must be able to replicate or install itself into multiple platforms based on the needs of the users or desired goals. There is no fixed configuration of the system but a dynamic one.

Self-Healing: - refers to an automatic detection, diagnosis, and correction of problems. Fault-tolerance is an important aspect of self-healing. An autonomic system must be reactive to failures or proactively detect early signs of a possible failure and remedy before it materializes. The types of errors detected could be interpreted broadly but they are generally classified as low-level errors and high-level errors. Low-level errors are likely to be encountered as a failure in hardware such as bit errors in memory chips. The high-level error could be an erroneous entry in a directory service (software problem) [71]. If possible, the autonomic system should attempt to fix the problem. Depending on the nature of the fault or failure it could be fixed either by automatically downloading and installing software or by switching to using redundant hardware components. However, it is important that as a result of the healing process the system is not further harmed, for example by the introduction of new bugs or the loss of vital system settings.

Self-Optimization: - this property refers to automatic monitoring and control of resources to ensure the optimal functioning with respect to the defined requirements. The autonomic system may decide to initiate a change to the system proactively (as opposed to reactive behavior) in an attempt to improve performance or quality of service. However, this optimization must be done with respect to the criteria relevant to the needs of a specific user, his or her peer group, and the enterprise [72]. Resource management is just one aspect of self-optimizing behavior.

According to [73], the self-optimizing property allows complex systems to improve themselves over time. For example, middleware (WebLogic, WebSphere) or database systems (Oracle, DB2) have hundreds of tuneable parameters that must be correctly set for them to function optimally. These would then be integrated with equally complex systems. Yet only a few people would know how to tune them for best performance. As a result, performance tuning one subsystem could have unanticipated effects on the entire system. This is where the self-optimizing property of autonomic systems comes useful.

Autonomic systems will continuously improve their operations through constantly monitoring, experimenting, and tuning their own parameters. These systems will seek and identify opportunities that allow improvement and learn to make appropriate choices on what functions to keep, improve, and discard.

Self-Protection: - this relates to the property of an autonomic system to protect itself from both external and internal threats and vulnerabilities. The system is expected to automatically configure and tune itself to achieve security, privacy, function, and data protection goals. The system is expected to defend

itself against large-scale, correlated problems arising from malicious attacks or cascading failures that remain uncorrected by self-healing measures.

In terms of internal threats, the autonomic systems will anticipate problems based on early incidents or threat reports from sensors and take steps to avoid or mitigate them. These also apply to end user actions which could be accidental (inadvertently deleting an important file) or deliberate (introducing vulnerability in the hope of exploiting it later). The system must anticipate such security breaches and prevent them from occurring in the first place.

With these initial properties in place, IBM [70] also proposed 8 conditions a system must meet to be considered autonomic.

1. The system must know itself in terms of what resources it has access to, what its capabilities and limitations are and how and why it is connected to other systems.
2. The system must be able to automatically configure and reconfigure itself depending on the changing computing environment.
3. The system must be able to optimize its performance to ensure the most efficient computing process.
4. The system must be able to work around encountered problems by either repairing itself or routing functions away from the trouble.
5. The system must detect, identify and protect itself against various types of attacks to maintain overall system security and integrity.
6. The system must be able to adapt to its environment as it changes, interacting with neighbouring systems and establishing communication protocols.

7. The system must rely on open standards and cannot exist in a proprietary environment.
8. The system must anticipate the demand on its resources while keeping the complexity of resource management transparent to the users.

Since then others have expanded on the self-* properties by incorporating some of the conditions above. Some of these self-* properties include

Self-learning [74]: - the system possesses ability to carry out unsupervised learning which does not require any external control.

Self-creation also called Self-assembly, Self-replication [75]: - the systems are self-driven and self-motivated in coming up with creative responses to continuously changing demands.

Self-governance [76]: - the system governing itself without external intervention.

Several other self- properties are listed in [77, 78] such as self-awareness where the system knows its resources and the resources it links to and is aware of the internal and external components that it manages. Self-organization, where the system structure is driven by a formal model rather than through explicit external pressures, self-description where the system explains itself in a manner capable of being understood by humans without further explanation, self-regulation where the system operates to maintain some parameter(s) within a pre-set range without external control.

Even with these expansions on self-* there is a debate within the research community as to what self-management systems actually are. The confusion stems from the fact that a query optimizer in DBMS, a resource manager in OS,

or routing software in networks all allow those systems to self-manage. However, the autonomic research community is in agreement that these do not constitute a self-managed system. According autonomic research community these systems simply reflects the change of the query optimizer decision or the network routing decision [58] and not necessarily display self-* capabilities. Therefore, the autonomic research community has been identifying a system as autonomic if it exhibits more than one of the aforementioned self-management properties [79].

2.2 Self-Adaptive Systems

Going by the above consensus in the autonomic research community, it could be said that self-adaptive systems are also autonomic systems. For example, self-adaptive systems have contained elements such as self-optimization and self-configuration for some time. Early use of this self-optimization property could be found on streaming media systems which optimized the music or video playback quality to optimal based on the available bandwidth [80, 81].

However, there's a key difference between the autonomic system and self-adaptive system behavior. The main difference between the two is when each system carries out self-optimization. Self-optimization in autonomic systems is continuous; the system is always on the lookout for opportunities for further optimization. Contrarily, the self-adaptive systems optimization happens due to change in external factors or an external trigger event.

This distinction is clear when looking at the some of the most commonly accepted definitions [82] of self-adaptive systems. An early definition of self-adaptive systems presented by Ravindranathan and Leitch [83] states that “in the context of multi-model systems, adaptation is a procedure or method for

switching between models. Adaptivity can, therefore, be defined as the capability of a system to achieve its goals in a changing environment, by selectively executing and switching between models. This capability contrasts with the conventional use of the term adaptation used in mono-model systems, where design parameters or relations are tuned to fit observed behavior". In this definition, the adaptation is triggered by the change of environment.

Brun *et al.* [84] define self-adaptation as the "capability of the system to adjust its behavior in response to the environment. The "self" prefix indicates that the systems autonomously decide (i.e., with minimal or no interference) how to adapt or to organize themselves so that they can accommodate changes in their contexts and environments".

Naqvi [85] defines self-adaptive by explicitly defining the mechanism used for adaptation. According to this definition "a self-adaptive system consists of a closed-loop system (i.e., modify in runtime itself using feedback due to continuous changes in the system), its requirements and existing tendencies in developing and deploying the complex system, thereby reducing human efforts in the computer interaction. The conception of the self-adaptive system depends on user's requirements, system properties, and environmental characteristics. The self-adaptive software requires high dependability, robustness, adaptivity, and availability". Unlike other definitions, this restricts self-adaptive systems to one containing a closed-loop system. However, this is no longer the case and a wide range of other techniques are also being used for adaptation.

Salehie and Tahvildari's [86] definition states that "a self-adaptive system evaluates its own behavior and changes its own performance when the evaluation indicates that it is not accomplishing what the software is intended to do, or when

better functionality or performance is possible". In this definition, the prominence is given to self-evaluation with a decomposed view of the adaptation mechanism such as:

1. Monitoring software entities (self-awareness)
2. Environment (context-awareness)
3. Analyzing significant changes
4. Planning how to react and executing so that the decisions take effect

The common approach in existing solutions is to have an external adaptation manager which goes through the above four steps and controls the behavior of the adaptable software. Programmers are expected to include the application logic in the adaptable software in order to be self-adaptive.

In [55] Salehie and Tahvildari listed the following questions being elicited to answer the requirements for adaptation. These are collectively known as 5W1H.

- When to adapt?
- Why do we have to adapt?
- Where do we have to implement change?
- What kind of change is needed?
- Who has to perform the adaptation?
- How is the adaptation performed?

However, Krupitzer *et al.* [87] argue that "who has to perform the adaptation" is not required in the self-adaptive system as self-adaptive systems are expected to adapt without user involvement.

In all of the above definitions, self-adaption is triggered by a change in the monitored environment. In [88] self-adaptation is formally defined as a system undergoing a state transition due to self-action. In this definition, Autonomic System (AS) is thought of as a system consisting of multiple states. Self-adaptation is defined as autonomic system transitioning from one states (AS) to another state (AS^1) due to self-*action. Self-*action is the mapping that sends each state in AS to a state of AS^1 . AS called the domain of self-*action and AS^1 the co-domain of self-*action.

2.3 Self-Adapting Techniques in Context-Aware Applications

As mentioned earlier self-adaptive systems differ from autonomic systems based on what triggers the self-adaptation. As stated in the previous section the trigger for the self-adaptation is a change in the environment monitored by the self-adaptive system. Within a context-aware application domain, self-adaptation is triggered by a context change. It could be argued that context could be treated as a synonym for the environment. But the definition of context mentioned in the previous chapter shows that context encompasses not only environmental aspects but user interactions as well. As such adaptation within context-aware domain could be treated as a special case of self-adaption.

Nevertheless, there are similarities between context-aware adaptation and self-adaptation in other domains (what should be called from now as general self-adaptation). For example, according to [29] “context-aware applications look at the who’s, where’s, when’s and what’s (that is, what the user is doing) of entities and use this information to determine why the situation is occurring. An application doesn’t actually determine why a situation is occurring, but the designer of the application does”. The “who, where, when, what and why” in this

case constitute the 5Ws in “5W1H” mentioned in [55]. This similarity in the decision structure used for self-adaption allows techniques used in general adaptation domain to be used with context-aware adaptation domain as well.

However, there are unique set of challenges a self-adaption technique must overcome within the context-aware domain. Top among the challenges is the limited computing resources available for a context-aware application to carry out the self-adaptation related tasks, be it extensive calculations with multiple parameter sets, dynamic experimentation or evaluating the outcome of the adaptive action. Secondly, within the context-aware domain, the adaptation must happen near real time. So, the answer to the question of “when to adapt?” is “near real time” which is different from general self-adaption where adaptation could be proactive or reactive. Therefore context-aware application at times may decide to adapt (self-optimize) to a “locally optimal” state instead of “globally optimal” state if it means adapting in near real time. Depending on the nature of certain context-aware applications, a quick adaptation to a “locally optimal” state may be preferred to a slow adaptation to a “globally optimal” state.

The following section examines the commonly used self-adaptive techniques in context-aware application. As mentioned in the previous chapter the objective of this research is to develop a framework for a context-aware system that finds the best adaptive action for unknown context. The existing body of work is critiqued on the basis of the ability of these techniques to handle unknown contexts. Taxonomy of these existing techniques is given at the end of the chapter, showcasing the position of this research work among the body of the existing work.

2.3.1 Manual Adaptation

The manual adaptation of a context-aware system is not widely used anymore. It's included here for the sake of completion. Under this method, either the end user or the system developers are expected to make changes to the context-aware application to better adapt to the prevailing situation (i.e. current context).

One of the early uses of manual adaptation on the context-aware application is presented in [33] where context-aware applications take input from the environment via sensors and allow users to select from a range of physical and logical options according to their current interests or activities. In this case, the system does not self-adapt.

If this type of method is used in the modern context-aware systems, it would result in context inference failure when an unknown context is encountered. A context inference failure occurs when the context-aware system encounters a context to which it has no adaptive action. The most likely cause for this type of issue is system developers being unable to foresee all possible context changes. In such cases, if manual adaptation is employed user intervention would be needed every time a context inference failure occurs.

2.3.2 User Defined Adaptation

User defined adaptation technique is the most widely used self-adaptation technique in the context-aware application domain. The “user” here could be either the system developer or the end user who configures the context-aware application. The user matches each context change to a corresponding action that would make the system adapt (self-optimization) keeping in-line with the user goals. As shown in Figure 1 there is direct 1:1 correlation between self-adaptive

action and context. This type of model is referred to as single-context single-action model.

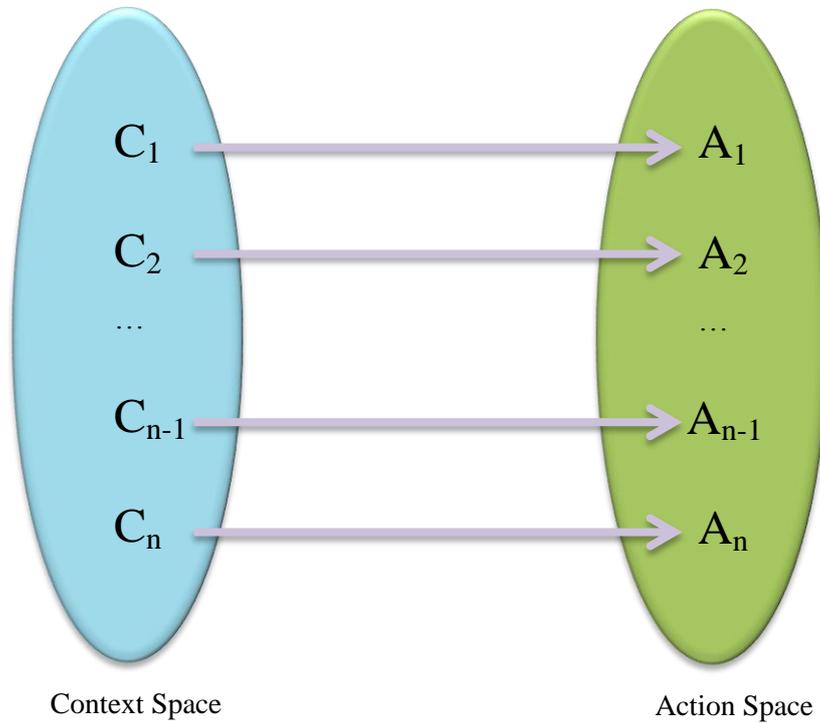


Figure 1. Single-Context Single-Action Model

The action the system takes to self-adapt such that its outcome is optimized to the current context is referred to as a self-adaptive action.

In order to have a 1:1 correlation between the elements in the context space and the action space, the system developer must know of all the contexts the system is likely to encounter. The expected numbers of contexts are few and finite. Therefore, the system developer or user is able to embed the context-action pairs into the context-aware application.

As mentioned in the introduction this type of self-adapting context-aware application is widely encountered in the smart environment. Three main areas could be identified in these types of context-aware frameworks – they are sensory data acquisition, context inference/management, and action. In [20] an early example of such a smart system called “CyberDesk” is presented, which self-

adapts based on user needs. It maintains a registry which consists of systems and interfaces that an end user works with. These could be an email reader, a contact manager, or a Web-based search engine. It uses the user's location as the context and the services available at any particular time depend on the user's context at that time. The context change, in this case, is whether the user is stationary (sitting at a desk) or mobile. The system adapts between a desktop based service and a mobile service depending on the current context. In [21] a similar concept is presented for distinguishing between stationary and mobile users with the use of wearable devices. Sensors are used to detect the user's location and the system adapts the service provided based on the current location. As mentioned in the earlier chapter using location as context was common during the early days of context-aware application.

A smart context-aware space which uses not only location but users and their interactions (which is the current definition of context) is presented in [89]. This smart environment not only detects the users' location to adapt the services but the nature of the interaction between the users. It shows how the users' location is detected based on Bluetooth service on user devices "the context broker concludes that the owners of the detected devices are also located in Room 338". It then infers the context of the current location acquiring information from scheduler service "on 8 September 2004, a presentation is scheduled to take place from 1:00 to 2:30 p.m. in Room 338". When it is time for the meeting to start, the system takes adaptive actions based on what the user has defined. In this case "all the lights in the meeting are currently switched on and the background music is still playing, the agent invokes the dim light method on the light-control service and the stop music method on the music service". This shows 1:1

relationship between current context (this particular meeting) and adaptive action (turn off lights and music). If some other meeting requires lights to be kept switched on and only background music to be turned off (unknown context), then this had to have been envisioned by the developers and programmed into the context-aware application.

Dey *et al.* [40] presents a way to overcome this need for capturing the context-action relationship at programming stage. They propose a programming by a demonstration where users demonstrate the desired adaptive action for a particular change. Though this allows the system to capture the end users' desired action, the action space is limited by what the user demonstrates.

Cao *et al.* [90] present a context-aware tourist guide application which self-adapts the contents served based on the context, in this case, the location of the device and the device itself. The system dynamically adapts the format of the content served to best fit the device capabilities and relevance of the content to best fit the current location. The system is vulnerable to encountering locations previously not visited or new devices. The system tries to overcome this problem by using "social intelligence", in which the user community defines the best adaptive action.

Mizzaro and Vassena [91] also propose the social approach to overcoming context knowledge shortage in applications similar to above [90]. Though this does not directly relate to the smart environment, the concept, if required, could be adopted specifically to increase the domain knowledge space. The idea is to allow "crowd of users to model, control, and manage the contextual knowledge through collaboration and participation, to have a dynamic and user-tailored context representation, and to enhance the process of retrieval based on users'

actual situation, the community of users is encouraged to define the contexts of interest, to share, use” [91]. The downside in using social collaboration to enhance the domain knowledge related to context is that knowledge could become subjective (depends on each individual user’s experience), thus creating many versions of knowledge for the same context. This could lead to ambiguity when it comes to selecting an adaptive action.

The emergences of service-oriented architecture (SOA) paved the way for service orchestration techniques and mashup web services. This allowed developers to build a new system or new capabilities by orchestrating existing services. He *et al.* [92] provide such a use case which combines the semantic web with the ubiquitous web with the use of a mashup web services to create smart (context-aware) plant watering system. The system proposed by [92] will monitor the humidity through a sensor and the rain forecast through a weather service. Based on these aspects the system dynamically determines whether watering is needed. The system uses a plant-cultivation domain-knowledge service which allows calculation of the amount of water the plant requires by understanding the species and life stage. The system is dependent on the designer inputting the domain knowledge “we also take domain knowledge services as a vital component that can explore human-accumulated knowledge related to the target thing” [92]. This information is fed into the system by the end user (an example of end-user defined adaptation). This user configuration, actually, is unified by a plant ontology service, and acts as an input to the plant-cultivation domain-knowledge service. Finally, depending on the context (humidity value is lower than a specified threshold and no rain is forecasted) the system triggers the actuator to water the plant with a specific amount of water. The system will

dynamically adapt the dosage of water based on the context. However, this type of system would be unable to handle an unknown context aspect arising outside the encapsulated domain knowledge.

Though the single-context single-action model is common in the smart environment it could be employed for self-adaption of software as well. A context-ware framework for controlling hardware and software for optimizing power consumption in a mobile phone is provided in [93]. The context, in this case, comprises of the battery level and status, the current location and the time for the next available charging opportunity. The framework proposes three profiles with various self-adaptive features at four distinct levels – hardware & software features adoption, user features adoption and additional optimization. When the battery is critically low, priority is given to maximizing the battery life until the next charging opportunity. The application will offer energy efficient alternatives to the user to minimize the battery consumption while compromising the user experience.

A similar strategy has been put forward [94] for reducing the power consumption of mobile network base stations by adjusting the available bandwidth during low usage times. The first context source comprises static information parameters supplied by network providers or network component manufacturers. Secondly, the user context which is the data coming from mobile devices. Third context source is the contextual information gained from the mobile network components. The final context source is information acquired from third-party context sources, such as web services or databases. Based on the current context, the context-aware application dynamically adjusts the

availability of resources and bandwidth which in turn reduces the overall power consumption in the network.

In all of the above scenarios, the adaptive action was predefined by the system developer or by the end user during a configuration phase. The system will fail to self-adapt if it encounters a context that has not been envisioned and adaptive action embedded into the system. Relying on developers to capture the context could lead to incorrect contexts and inflexible context definitions [95]. So there is a need to enable the system to recognize more contexts and adapt to optimize the computing outcome. Loke [47] proposes a formal method for incremental context awareness. The model defined two monotonic extensions – breadth-monotonic extension for extending the system so that it recognizes more situations (context) than before and depth-monotonic extension for the cases when there is uncertainty and the system extends itself through estimation. “The idea of incremental awareness is, hence, to effectively “grow” the system over time in such a way as to preserve either or both of these monotonicity properties, the system at a later time being a depth- and/or breadth-monotonic extension of the previous version [47].” Though the system increments its context-awareness, it is up to the designers to embed the adaptive action as there are no action evaluation capabilities in this model to verify if the adaptive action results in self-optimization.

2.3.3 Learning-Based Adaptation

Developers had to overcome the limitations presented in the user defined (single-context single-action models) self-adaptive technique by allowing the system to dynamically select the best adaptive action. The use of learning-based adaptation technique provided this capability. However, the learning-based

adaptation technique is not specific to the context-aware system. It has been widely used in general self-adaptive systems and tightly coupled to self-optimization. The learning-based adaptation could be further classified based on the learning approaches. One of the key features of learning-based adaptation is the ability of the system to determine the distance between the expected and the actual goal based on the selected self-adaptive action.

Tesauro *et al.* [96] use the concept of the central arbiter to determine the optimal resource allocation which is computed based on a utility function, which is refined by learning. Elkhodary *et al.* [97, 98] propose a learning and self-optimizing technique purely based on the goals and utility function attached to each goal.

In [99, 100] a collaborative learning approach is proposed, which differs from the earlier mentioned single function learning approach. Collaborative reinforcement learning makes collective adaptation possible by collaborative feedback and lessons learned are broadcasted to all collaborating parties. A collaborative reinforcement learning model for decentralized coordinated self-adaptive components is presented in [101] where the components learn collectively through collaboration but self-adapt individually.

A context-aware adaptation based on the Constraint-Satisfaction Problem (CSP) technique is proposed in [102], handling situations where several configurations are available for the same context. Abdelwahed *et al.* [103] propose a self-adapting technique for distributed systems with varying environmental and operating conditions. It uses prediction-based learning combined with an online control technique to tune the performance of individual system components.

Evolutionary programming and artificial intelligence (AI) based learning techniques are proposed in [54,104,105] to handle encountering uncertain changes and facilitate self-adaptation.

Not all of these learning techniques are equally effective and compatible with the requirements of the context-aware application domain. For instance, the AI and evolutionary programming techniques would be resource intensive and time-consuming to be effective in near real time adaptation in a context-aware application. Moreover, these techniques require that the system is trained beforehand in order to recognize contexts.

The solution was to associate a subset of the action space with each context as possible self-adaptive actions. When the system encounters an unknown context change, it would carry out self-adaptive actions from the associated subset of actions, iteratively measuring the outcome for each action against the expected outcome. The system could finally settle on the action that gives either the expected outcome or has the one that resulted in the least distance to the expected outcome. In this technique, there is a one-to-many relationship between the context and the adaptive action as shown in Figure 2. This type of model is referred to as a single context multiple-action model.

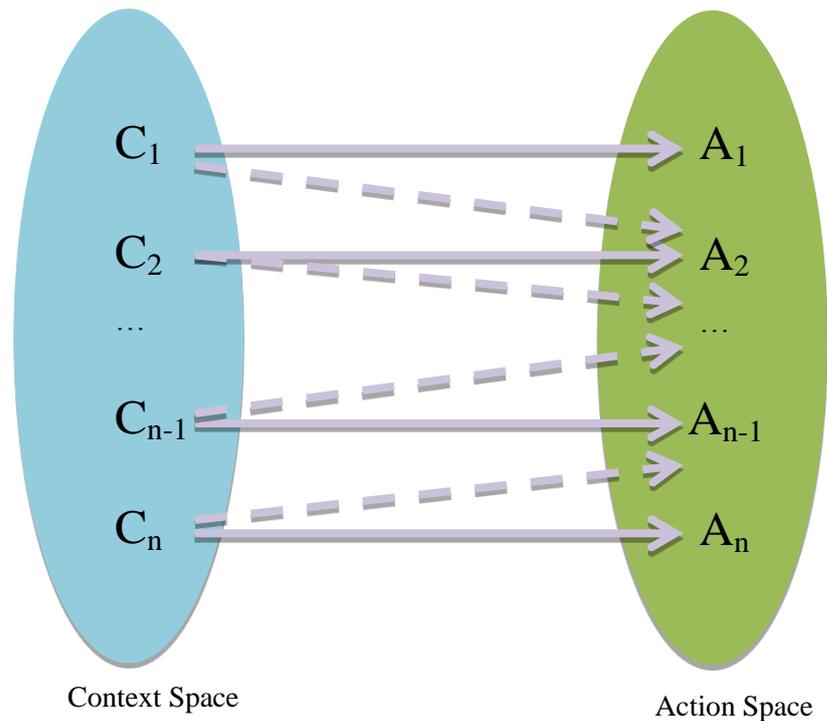


Figure 2. Single-Context Multiple-Action Model

Among the learning techniques, the supervised learning with feedback loop captures these requirements and has been used with context-aware applications. O'Connor *et al.* [95] develop a context-aware heating controller which self-adapts based on temperature readings. The context-aware model uses Q-Learning with a feedback loop to find the best adaptive action for each set of context changes. The application receives a reward from the environment when an action is taken in a particular context. By exploring all combinations of contexts and actions, the application learns the most suitable adaptive action for each context.

Another instance of the use of feedback-based learning for context-aware adaptation is presented in [106]. In this case, context is used to improve the ambient assisted living conditions by using feedback from previous instances of intervention to resolve current issues. The feedback information highlights important criteria such as the quality of sleep during the night and possible breaches of safety during the day in order to better service the residents.

Case-based reasoning (CBR) [199, 200, 201, 202, 203, 204] is a problem solving methodology which uses experience drawn from the encounters of similar kind of problems in the past to solve the current problem. CBR has been widely used in many areas including in autonomic systems to achieve self-configuration capabilities [205, 206].

A context-aware case-based reasoning (CBR) application is presented in [107] which use a combination of a context-aware CBR with general domain knowledge to solving domain specific problems in uncertain contexts. The system consists of three models: situation awareness model, general domain knowledge model, and the case model. The case model is the library containing past cases and their solutions, thus enabling the system to evaluate the effectiveness of the current configurations.

Adaptation based on supervised learning with feedback loop becomes unfeasible when the number of context values increases as this leads to a large number of context and action element combinations. According to [95], the Q-learning itself becomes infeasible when a number of states in context space increases, thus it requires intermediary learning states to reduce the number of states to learn about. As for case-based reasoning, the quality of adaptation depends on the domain knowledge and the number of past cases available. This poses the same problem mentioned in the previous section which is, the system having to depend on the developers to capture context to assign appropriate self-adaptive action.

2.3.4 Policy-Based Adaptation

Policy-based adaptation techniques provide an alternative to the aforementioned two techniques. There is no direct correlation between context

change and the adaptive action. Instead, the system developer formulates a set of policies which determine the adaptive action to take based on what policies are in effect before/during/after the context changes.

In [108] Salomie *et al.* introduces a triple set model (called RAP) where the context C is modelled as a triple $C = \langle R, A, P \rangle$ where R represents a set of context resources, A is a set of actors which interact with context resources and P is a set of real context related policies. This context model is mapped onto different real contexts by populating the sets with real context-specific elements, thus resulting in a specific context model $C_S = \langle R_S, A_S, P_S \rangle$.

Cioara *et al.* [109] expands on the context triple model presented in [108] by introducing self-adapting capabilities to the RAP model. The self-adaptive algorithm in [109] uses a closed feedback loop with four phases which are monitoring, analysis, planning, and execution. During the monitoring phase the system continuously monitors itself and its execution environment in order to capture and represent the relevant information. The analysis phase deals with detecting significant changes in the system itself and/or the execution environment (i.e. context change detection). The planning phase decides and selects the appropriate adaptation actions as a response to the detected changes. Finally, the execution phase modifies the system behavior by enforcing the adaptation actions selected in the planning phase. Looking at the pseudo code of the algorithm presented in [109], it could be seen that this algorithm evaluates each action one at a time in an iterative manner. As mentioned in the previous section of learning-based techniques, using feedback loop for learning could be time-consuming in finding the best adaptive action when there are multiple inputs generating a high number of permutations. Though [109] expands on a policy-

based approach, the introduction of feedback loop has made it unviable for a context-aware application that seeks to self-adapt through multi-action evaluation.

One of the unique uses of the policy-based adaptation techniques is to enforce context constraints by way of self-adapting to restrict user capabilities. In [110] a self-adaptive context-aware system is presented, which restricts user access to the network resources based on a trust policy. The system self-adapts the resources' access criteria dynamically based on the context in which the resource is accessed and the trust policy assigned to the user. A similar policy-based approach is presented in [111] which uses an adaption tree to model to adapt the rules (difficulty level) of a context-aware mathematic game. Samulowitz *et al.* [112] presents a document-based approach with the introduction of Context-Aware Packets (CAPs), which contain context constraints and data to adapt a service request to the prevailing context. In essence CAPs, in this case, act as the policy document. Dey *et al.* [113] use the policy-based approach as a lightweight alternative to continuous adaptation for a context stream. According to [113] an event-based adaptation instead of policy-based would be “quite heavyweight when streams of context are being sent (e.g., from a temperature sensor that produces updates each time the temperature changes by .01 degrees)”.

A formal model for policy based self-adaption is presented in [114] called PobSAM (Policy-based Self-Adaptive Model) for developing and specifying self-adaptive systems that employ policies as the principal paradigm to govern and adapt the system behavior. The main elements of a PobSAM model are actors that perform the main functionality of the system and managers that control the behavior of actors autonomously according to a set of predefined

policies. Conceptually the model has been represented as three layers: managed actors layer, autonomous manager layer and view layer which provides a layer of abstraction between the actors and managers. Formally PobsSAM is denoted by $\Pi = \langle R, V, E, M \rangle$ where R, V, E, and M represent the set of actors, view variables, events, and managers. Similar to RAP triple mentioned in [108], the idea is to populate the PobsSAM with actual instances of actors, views, events and managers. In [114] the implementation of the PobsSAM is given for unmanned autonomous vehicles (UAV) where the model adapts vehicles role (survey, relay, idle) based on context information.

Though policy-based adaption techniques eliminate the issues associated with the single-context single-action model and single-context multi-action model, they introduce their own set of issues when considered for multi-context multi-action application. Due to a large number of context-action pairs that exist in multi-context - multi-action model, a dynamic evaluation policy at run time could become high resource consuming operation.

2.3.5 Statistics-Based Adaptation

The statistical methods use probabilistic reasoning to help the system choose a suitable adaptive action out of many different actions available to the system. Most commonly used statistical methods are Bayesian Network and Hidden Markov model.

Bayesian Network (BN) is one of the models used to represent uncertainty [115]. This feature of BN is used in context-aware systems to adapt amidst the uncertainty in the perceived context. Zheng *et al.* in [116] describe a Bayesian network of n variables consisting of a DAG (Direct Acyclic Graph) of n nodes and a number of arcs. Nodes X_i in a DAG correspond to random variables, and

directed arcs between two nodes represent direct causal or influential relations from one variable to the other. The uncertainty of the causal relationship is represented locally by the CPT (Conditional Probability Table). $P(X_i/Pa(X_i))$ associated with each node X_i , where $Pa(X_i)$ is the parent set of X_i . Under the conditional independence assumption, the joint probability distribution of $X = (X_1, X_2, \dots, X_n)$ can be factored out as a product of the CPTs in the network, namely, the chain rule of BN: $P(X) = \prod_i P(X_i/Pa(X_i))$. With the joint probability distribution, BNs support, at least in theory, any probabilistic inference in the joint space.

Both [117] and [118] propose using probabilistic reasoning coupled with ontological reasoning to self-adapt when uncertainty is encountered in the context. Instead of failing when a developer has not programmed an adaptive action (user defined adaptation) to perceived context, these systems rely on BN to choose the most suitable adaptive action among many.

In [117] the adaptation to a new context comes in the way of restructuring conditional probability table (CPT) by adding a node to the given Bayesian network. However, in [118] probabilistic information derived from BN is used to create instances of ontology classes. The context-aware system uses these instances of ontology classes to infer on the perceived context and adapt.

Hidden Markov Model (HMM) is another statistical method used in context-aware applications to facilitate adaptation. HMM allows representation of probabilistic distribution over a sequence of observations [119, 120] and consists of two main properties that give the “hidden” and “Markov” to its name. In this model, an observation X_t at time t is produced by a stochastic process whose state S_t is hidden from the observer (hidden property). Secondly, this hidden process is

assumed to satisfy the Markov property, where the current state S_t at time t depends only on the previous state S_{t-1} at time $t - 1$ and is independent of all values prior to $t - 1$ referred to as 1st order Markov property.

This ability to predict the current state based only on the previous state has been used in [121] to develop a context-aware application which adapts to the service provided to users based on observed user actions. The system uses previous observations to predict the intention and mental state of the user and adapts the services to best fit the user expectations.

Markov Logic Network (MLN) [122] is another Markov statistical method used in context-aware applications. MLN enables uncertain inference as a result of applying the ideas of a Markov network [122, 123] to first-order logic. The idea behind MLN is that when the system violates one formula in the knowledge base that formula becomes less probable, but not impossible (fewer the violations for a particular formula, the more probable it is). This MLN probability-based reasoning has been used in [124] to develop a context-aware travel recommendation system. The system models the context as the user's current location, destination searched, travel habits. The context-aware application adapts its recommendation such that destinations with a higher probability of being selected are presented to the user.

The issue with statistics or probability based adaptive techniques is that they are prone to erroneous prediction. When the system encounters an unknown context, the accuracy of the adaptive action taken depended on the existing statistical data and the predictive model used. In the multi-context multi-action model, this would require gathering a large data set in order to predict the appropriate adaptive action with high probability.

2.4 Taxonomy of Context-Aware Adaptation Techniques

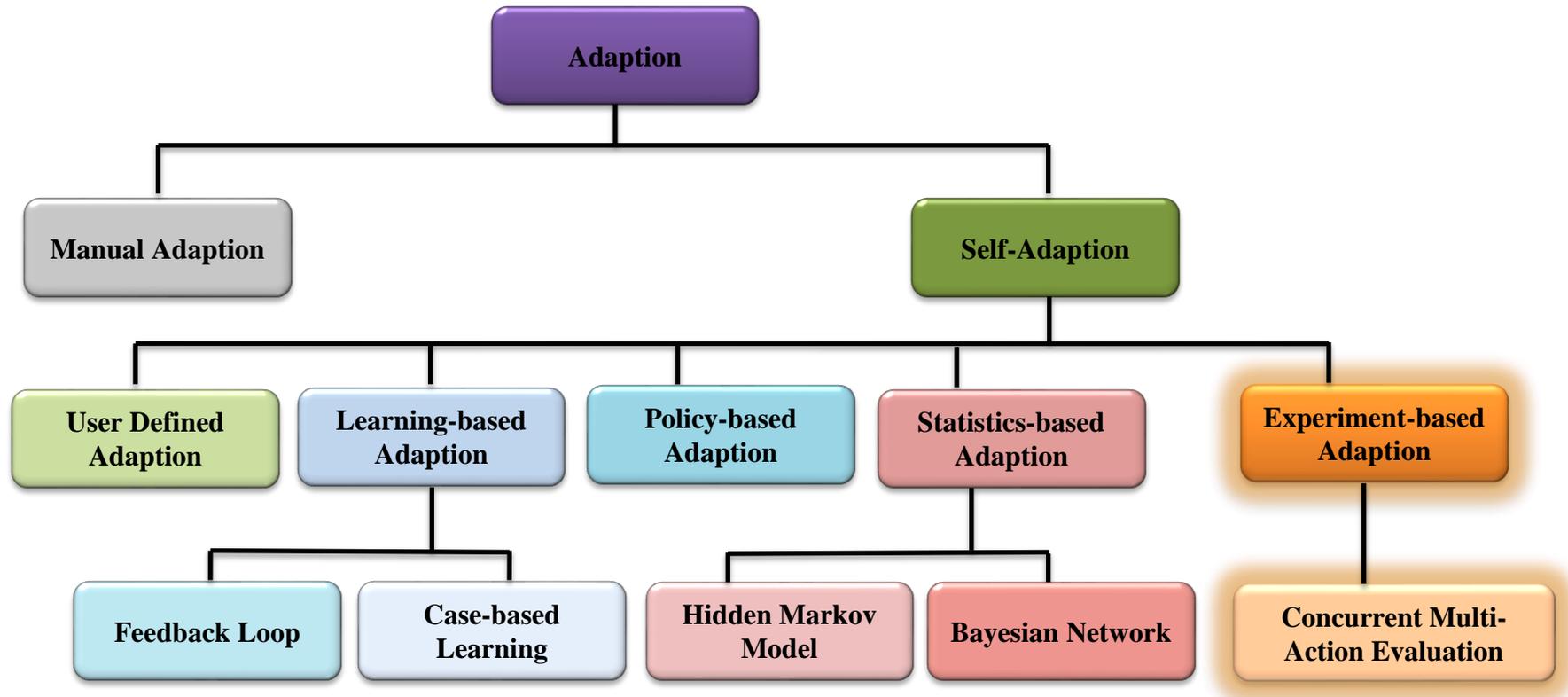


Figure 3. Taxonomy of Context-Aware Adaptation Techniques

The above taxonomy has been created encompassing frequently used context-aware adaptive techniques. Manual adaptation was present in the early context-aware application where users were presented with the option of choosing an adaptive action out of many. However this approach has limited usage nowadays. From the manual adaptation the momentum shifted towards automated self-adaptation.

User defined adaptation was the first technique in the self-adapting context-aware application landscape. It relied on system developers or users to specify or configure the system's adaption prior to use. System was expected to encounter only few context changes and had a single action associated with each context. Naturally this technique suffered from context inference failures when system encountered an unknown context.

This led to the next step up in the self-adaptive techniques in context-aware application domain. The common theme in these techniques is self-adaptation without user intervention or with minimal user intervention when unknown contexts are encountered. Foremost among these techniques is the learning-based adaptation technique. Most common subcategory in learning-based adaptation is feedback loop base learning, widely used in control system. Feedback loop uses error correction mechanisms to narrow the difference between expected and actual goals for each action system iterates over as a result of a context change. When the number of actions the system has to iterate over is large the time for adaptation also increases. The second learning-base technique is case-based reasoning. Case-based reasoning uses past experience (action used in similar situation in the past) to adapt when unknown context is encountered. The quality of the adaptation depends on similarity between past case and current case.

Statistics-based hidden Markov model and Bayesian networks use statistical information from the past context changes to adapt to an unknown context. Similar to case-based reasoning mentioned earlier, in this technique also the quality of adaptation depend the quality of the statistical data available.

An alternative to above self-adapting techniques was the policy-based adaptation. Rather than explicitly specifying the exact action to undertake in order to adapt, the policy-based adaptation technique only specify set of policies to follow to adapt. The policy rules governs which actions are applied for a context changes. Similar to user defined adaptation, the policy-based adaptation depends on system developers or users to formulate the policies that should govern the adaptation.

The critique of the existing techniques has shown the need for a new approach to handling adaptation when an unknown context is countered in a multi-context multi-action model.

This research proposes an experiment-based adaptation technique which uses concurrent multi-action evaluation to find the adaptive action, applicable to a multi-context multi-action model when a context-aware application encounters an unknown context.

The research envisages a new paradigm of experiment-based adaptation techniques in the context-aware domain. Self-adaptation via concurrent multi-action evaluation would be one construct of this new paradigm.

3. GENERIC FRAMEWORK FOR SELF-ADAPTATION VIA CONCURRENT MULTI- ACTION EVALUATION

3.1 Motivation

The inability to use existing context-aware adaptation techniques with multi-context multi-action models is the primary motivation for proposing a generic framework which uses a novel experiment-based adaptation technique. The proposed framework carries out context-aware self-adaptation via concurrent multi-action evaluation.

The motive of the novel framework is to address the limitation in the existing context-aware adaptation techniques which were detailed in Chapter 2. In summary, the new framework addresses the following issues in the existing context-aware adaptation techniques.

1. Dependency on system developers to pre-define context and context elements which cause rigid context declarations.
2. Dependency on system developers to encompass and embed all possible context change-adaptive action pairs into the system.
3. Inability to carry out system adaptation when an unknown context is encountered.
4. The time for system adaptation being dependent on the size of the action space (i.e. the number of adaptive actions to be evaluated).
5. The need of user intervention to expand the knowledge base which stores context and adaptive action information.

The secondary motivation of the proposed framework was to have an implementation agnostic context-aware platform which could be used across multiple domains. The existing context-aware adaptive techniques work well for certain use cases in a particular domain but cannot be ported to other domains or different use cases. By being implementation-agnostic the proposed generic

framework allows cross domain implementations using the generic framework as a blueprint. Sections 3.5 gives a generic UML class diagram which could be used as a blueprint for a concrete implementation of the framework in object oriented programming language. It must be stated that these diagrams are guidelines that captures the core concept of the framework. Developers are free to use these blueprints or create their own implementation strategy depending on the domain and the actual use case for which the framework is implemented for.

3.2 Generic Framework Description – Meta Level

The framework encompasses the three major aspects a self-adapting context-aware system must have, that is context sensing, action, and self-adaptation. With these three aspects in place, any concrete implementation of the framework would qualify as a self-adapting context-aware application. The conceptual framework was designed with three distinct systems each with a unique purpose. Figure 4 shows a high-level view of the system architecture.

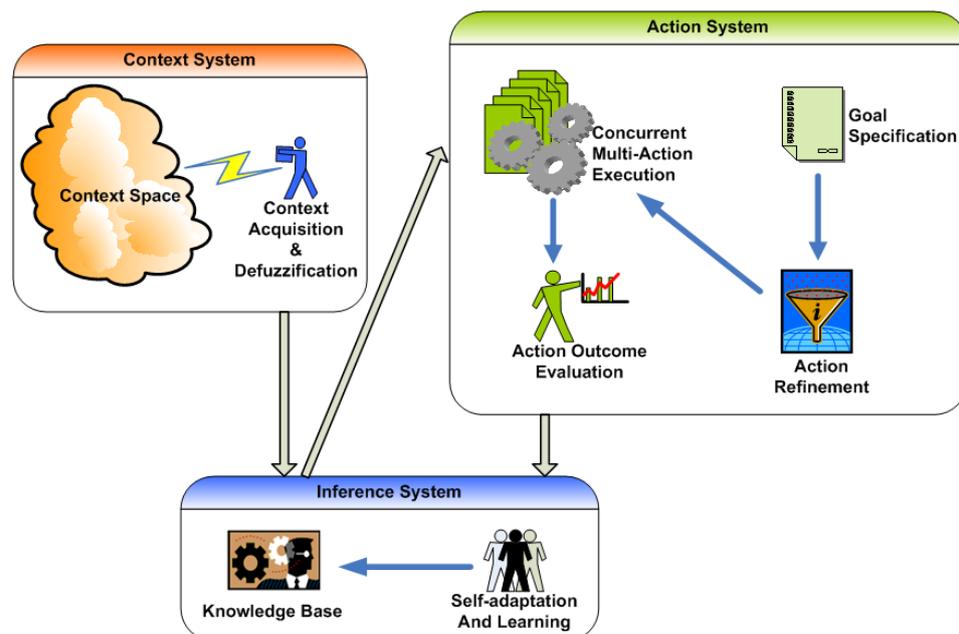


Figure 4. System Architecture of the Context-Aware Framework

The novelty of the framework is how these systems are orchestrated to find the best action for self-adaptation through concurrent multi-action evaluation. As seen in Figure 4 the generic framework consists of three systems, they are the context system, the inference system, and the action system. The subsequent sections give a detail description of each of these systems.

3.2.1 Context System

The context system carries out two responsibilities. First, it defines the boundaries of the context space. The context space boundary is determined based on the context definition in [29] the definition chosen to model context in this research. Primarily “any information used to characterize the situation of an entity” is considered within the context boundary, thus within the context space. Any information that does not fit this definition is considered outside of the context space. Any context that is outside the context boundary is considered not relevant to the context-aware application. Realistically the context space consists of a concrete subset of the context that is attainable from sensors, applications and users and able to be exploited in the execution of the task. The context space for a given context-aware application is usually explicitly specified by the application developer, but may evolve over time [64]. Information that was relevant at one point in time may become obsolete or new information could become relevant in characterizing the context. With advancements in sensor technologies, collecting low-level sensor data has become significantly easier and cheaper. This has resulted in the vast amount of data being collected and requiring non-traditional reasoning techniques to be employed to understand the sensory data [125].

Within a context space, each piece of information that characterizes a situation of an entity could be defined as a context attribute. A context attribute consists of an identifier, type, and a value, and optionally a collection of properties describing specific characteristics [64].

This research uses the concept of context attributes to overcome the problem of rigid context definitions by system developers. In the proposed generic framework, the context space is represented as a collection of context with each context consisting of set of context attribute values and identifiers. The context attribute values are presented by $v_{xy}, x = \{1..n\}, y = \{1..k\}$ and the attribute identifier is represented as $a_y, y = \{1..k\}$. With this notation the context space could be represented as a collection of contexts $\{C_1, C_2, \dots C_n\}$ where each context is represented as

$$C_1 = \{v_{11}a_1, v_{12}a_2, v_{13}a_3, \dots, v_{1k}a_k\}$$

$$C_i = \{v_{i1}a_1, v_{i2}a_2, v_{i3}a_3, \dots, v_{ik}a_k\}$$

$$C_n = \{v_{n1}a_1, v_{n2}a_2, v_{n3}a_3, \dots, v_{nk}a_k\}$$

When taken as sets each attribute and value pairs could be used to uniquely represent each context. To illustrate this representation in real world example assume a context consists of three attributes such as temperature, humidity and pollen level. Context space consisting of multiple values for each of these context attribute could be represented as

$$C_1 = \{20\text{ }^\circ\text{C}: \textit{temperature}, 70\%: \textit{humidity}, \textit{high}: \textit{pollen Level}\}$$

$$C_2 = \{25\text{ }^\circ\text{C}: \textit{temperature}, 67\%: \textit{humidity}, \textit{medium}: \textit{pollen Level}\}$$

$$C_3 = \{22\text{ }^\circ\text{C}: \textit{temperature}, 72\%: \textit{humidity}, \textit{low}: \textit{pollen Level}\}$$

If required this could be even represented as below for simplification.

$$\begin{bmatrix} 20 & 70 & high \\ 25 & 67 & medium \\ 22 & 72 & low \end{bmatrix} \begin{bmatrix} temperature \\ humidity \\ pollen level \end{bmatrix} = \begin{bmatrix} C_1 \\ C_2 \\ C_3 \end{bmatrix}$$

With this representation of the context space developers are not constrained by rigid context definitions. Evolution of the context space is carried out by modifying the elements in each set that represents a unique context.

The introduction of a new context attribute to the context space will result in the addition of a new element to the set representing the context. With this approach developers would not have to make any changes to the existing context definition as they will be unaffected by the addition of a new context attribute. This expansion could be illustrated as below where the context space of n contexts is expanded to $n+1$ with the addition of a new context attribute a_{k+1} .

$$\begin{aligned} C_1 &= \{v_{11}a_1, & v_{12}a_2, & \dots, v_{1k}a_k\} \\ C_i &= \{v_{i1}a_1, & v_{i2}a_2, & \dots, v_{ik}a_k\} \\ C_n &= \{v_{n1}a_1, & v_{n2}a_2, & \dots, v_{nk}a_k\} \\ C_{n+1} &= \{v_{(n+1)1}a_1, & v_{(n+1)2}a_2, & \dots, v_{(n+1)k}a_k, & v_{(n+1)(k+1)}a_{(k+1)}\} \end{aligned}$$

Taking the earlier example, context space after the addition of a new context attribute wind speed could be illustrated as below.

$$\begin{bmatrix} 20 & 70 & high & n/a \\ 25 & 67 & medium & n/a \\ 22 & 72 & low & n/a \\ 28 & 60 & low & 30 \end{bmatrix} \begin{bmatrix} temperature \\ humidity \\ pollen level \\ wind speed \end{bmatrix} = \begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ C_4 \end{bmatrix}$$

Similarly, the removal of a context attribute from the context space could be represented by a reduction in element count in the context. The context set transformation shown below illustrates the removal of a context attribute a_k from an existing context space.

$$\begin{aligned} C_1 &= \{v_{11}a_1, & v_{12}a_2, & \dots, v_{1(k-1)}a_{(k-1)}\} \\ C_i &= \{v_{i1}a_1, & v_{i2}a_2, & \dots, v_{i(k-1)}a_{(k-1)}\} \end{aligned}$$

$$C_n = \{v_{n1}a_1, \quad v_{n2}a_2, \quad \dots, v_{n(k-1)}a_{(k-1)}\}$$

For the earlier mentioned example, simplified representation of the context space after the removal of a context element pollen level could be written as

$$\begin{bmatrix} 20 & 70 \\ 25 & 67 \\ 22 & 72 \end{bmatrix} \begin{bmatrix} temperature \\ humidity \end{bmatrix} = \begin{bmatrix} C_1 \\ C_2 \\ C_3 \end{bmatrix}$$

The removal of the context attribute does not reduce the size of the context space. However, the context attribute is now placed outside the context space boundary and is not used to characterize the situation of an entity. The new matrices represent this change in context representation.

The context space representation proposed provides a domain independent method for formulating context spaces. At the same time, it addresses the issue of dependency on system developers for context space formulation and rigid context definitions.

The second objective of the context system is context sensing. This has been broken down into two steps - context acquisition and defuzzification. The context system was formed with the assumption that the context space is heterogeneous where context attributes are acquired from various heterogeneous sources. This is in keeping with reality where modern day context-aware applications gather context attributes from various sources such as hardware sensors (hard sensors), web resources [126, 127], databases and host-to-host services (soft sensors). Acquiring context attributes from heterogeneous sources presents a problem of having each of these attributes in different types or even in fuzzy terms which would make it difficult to calculate a closeness measure between contexts. For example, a context-aware application used for controlling ambient temperature in

a room may get room temperature in crisp values through a hardware sensor, such as 10, 20 or 30 Celsius. However, same could be sensed through a soft sensor (i.e. Web service API) in fuzzy terms such as “very cold”, “cold”, “hot”, “very hot”. At times, use of fuzzy terms may provide a higher degree of accuracy than the use of a value range. An example of such a case could be found in defining a buyer type in a context-aware application which differentiates buyers based on their spending. A value range based buyer type classification would define a buyer spending less than \$100 “low buyer”, \$101-\$200 “medium buyer” and anyone spending more than \$200 as “high-end buyer”. In this case, buyers who spend \$10 and \$100, both will be classified as “low buyer”. This can lead to unfair classification of the users as the buyer who spent \$100 is much closer to a medium buyer than the one who spent \$10. Fuzzy sets address this problem with the use of the membership.

However, sensing using fuzzy terms is not feasible therefore the generic framework introduced a defuzzification layer into the context system to address this problem. Defuzzification allows different data types and values to be transformed into crisp values. Defining a context with crisp values allows differentiating between contexts and detection of unknown context. This differentiation is possible because, as explained previously, each value set within context is unique to that particular context. Furthermore, the action system uses the distance between context attribute values as a measurement for determining the nearest known context to any unknown context encountered. In this research the Euclidean distance between each context is used to measure closeness between contexts. As each context element is represented with a crisp value, the

Euclidean distance method provides a convenient way of calculating distance by considering each value as a coordinate point in multi-dimensional space.

There are many defuzzification methods available such as Centre-of-Gravity, Centre-of-Sum, First-of-Maxima, Last-of-Maxima, and Middle-of-Maxima [128,129,130]. The proposed generic framework is not prejudiced towards any particular defuzzification technique. It is left to the implementers of the framework to decide on an actual defuzzification method based on application and domain specific properties [131].

Finally, the generic framework has been architected such that the context system is fully decoupled from the action system. This makes the addition and removal of a context attribute from the context space agnostic to the concurrent action implementation in the action system. As a result of this decoupling the action space does not grow or shrink based on context space but only on the goal specification (goal specification is explained in the subsequent section detailing the action system). This negates the need for system developers to encompass and embed all context change – adaptive action pairs into the system. Both context and action space can grow and shrink independently of each other.

3.2.2 Inference Systems

The second system in the proposed framework is the inference system. The primary objective of this system is context-inference, which is to identify whether the context the system is currently encountering or operating under is already known or unknown. A context is considered unknown if the system is unaware of an adaptive action to execute in that particular context. Similarly a context is considered a known context if it has an associated adaptive action.

In existing context-aware systems, the inference system would consist purely of a knowledge base. In these systems, the knowledge base must be pre-populated by system developers with all “context change – adaptive action” pairs the system is likely to encounter. If the context-aware system is expected to encounter a new context, then the knowledge base must be expanded preemptively. Failure to do so would result in context inference failure and the system would be unable to execute an adaptive action.

The primary use of the knowledge base is to infer if the currently perceived context is known or unknown. If the context is a known context, then the knowledge base provides the corresponding adaptive action suited to optimize the goal expectation under the said context. If the context is unknown, then the inference system invokes the actions system to carry out the concurrent multi-action evaluation. The inference system provides the action system with the initial adaptive action around which the action space for evaluation is built. This initial adaptive action corresponds to the adaptive action of the closest known context to the unknown context.

The closest known context is found by calculating the distance between each known context’s attribute and the unknown context’s attribute. However, it is possible that two or more contexts could have equal distances. To overcome this problem the research expands the idea of context attributes in [64] by introducing a “priority” as a property of a context attribute. The priority is based on the degree of influence each context attribute has in the prevailing context. It is possible certain context attributes may have a higher degree of influence than others. With the introduction of the priority, if there are two or more contexts with equal distances to the unknown context, then the priority of each context

attribute is considered. In this case, the context containing high priority attributes is considered the closest context to the unknown context.

However, it is still possible that two or more contexts could have context attributes with the same degree of influence. In such cases, the decision of identifying the closest known context is based on the output value (scalar value) of the adaptive action. This presents the application developer with two options. One is to consider the context for which the corresponding adaptive action has the highest configuration parameter among other equal-distance peers which is termed as an optimistic approach. Another option is to choose the context whose action has the lowest configuration parameter as a method of risk aversion, which is termed pessimistic approach. Taking the earlier example of a context-aware application used for controlling ambient temperature, two equal-distance contexts could have adaptive actions which set the temperature to 20° or 30° Celsius. In the optimistic approach the context whose adaptive action sets the room temperature to 30° could be chosen as the closest known context. Under the pessimistic approach the context whose adaptive action sets the room temperature to 20° could be chosen as the closest known context.

The framework makes no assumption as to which approach to use and it is up to the application developer to choose a pessimistic or optimistic approach based on the context-aware system requirement. During the implementation of the second case study of the research both these two approaches were used and tested.

The techniques used for modelling the knowledge base within a context-aware application have evolved over time. The current trend is geared towards modelling context and the knowledge base using ontologies [132-141]. The

advantage of modelling the knowledge base using ontology is that it allows leveraging of ontological inference techniques to be used to carry out context inference. However, the disadvantage of this approach is that the knowledge base model created would have domain specific constructs. This would necessitate a higher degree of rework before this knowledge base model is adapted to another domain.

In the proposed framework, the implementation of the knowledge base is transparent to the other systems in the framework. No assumptions are made with regard to how the contexts are modeled and the knowledge is organized within the knowledge base. The two case studies implemented used two different knowledge base modelling approaches. One uses the current trend of using ontologies to model the knowledge base, while the other uses the knowledge base model inspired by data warehouse technique.

Apart from the knowledge base the inference system also consists of a self-adapting mechanism. One of the two aspects of the self-adaptation mechanism is to expand the knowledge base with results of the concurrent multi-action evaluation. Expanding the knowledge base allows the context-aware system to recognize more and more context over time. A separate self-adaptive mechanism also addresses the issue of requiring user intervention for knowledge base expansion. The second aspect is that the self-adaptive system will be responsible for executing the adaptive action on the external system which the context-aware system manages. In essence, the self-adaptive mechanism acts as an abstraction between the context-aware application and the external application managed by it, allowing ease of portability.

3.2.3 Action System

The action system is invoked when an unknown context is inferred by the inference system. The action system carries out the concurrent action evaluation to find the best adaptive action to take in the currently perceived unknown context. The best adaptive action is chosen based on how well it adapts the application in line with expected goals of the application.

The action system consists of four components (i) goal specification, (ii) action refinement, (iii) concurrent action execution, and (iv) action evaluation. Together these four components encompass the novel concurrent multi-action adaptive technique proposed by the research. The action system would address the two remaining issues identified in the motivation section. The objective of the action system is to carry out concurrent multi-action evaluation when an unknown context is encountered and to find the best adaptive action for that context. The novelty of the framework lies in the fact this adaptation could be carried out with minimal user input during the setup and execution of the context-aware application. The aforementioned four components are tasked with unique roles in achieving this objective.

The ultimate result of an adaptive action in a context-aware application is introducing change to the system it manages. The change introduced by the action manifests as the action's output. This research proposes the concept of defining adaptive action as a function of the parameter manipulated by it. This could be explained by taking the smart plant watering system mentioned in [92] as an example. The adaptive action, in this case, is the watering of the plant and the parameter manipulated by the adaptive action is the volume of water delivered to the plant. In this case, the self-adaptive action A could be defined as

a function of the volume of water delivered: A (*volume of water delivered*). The research uses this parametrized action concept to carry out concurrent multi-action adaptation with minimum user intervention.

The goal specification component of the action system encapsulates the extremities of the parameters used in the action space. The premise is that within a particular domain the set of parameter values interesting for the application is far less compared to the universe of parameter values within that domain. Therefore, in any implementation of the generic framework, the application developer only has to specify minima and maxima of the parameter values that the context-aware application must use for concurrent action execution. This releases the application developer from having to foresee all possible context-change and adaptive action pairs.

The action refinement component builds the action space used for the concurrent action execution. It uses the goal specification as the boundary for parameter values used for creating actions within the action spaces. In addition to the goal specification, three other action refinement related control values are introduced in the generic framework as means of limiting the number of actions in the action space, as well as means of fine tuning the concurrent execution. The action refinement receives the parameter value associated with a closest known context, discovered in the context inference phase as an input. This closest known context is used as the seed for expanding the action space with the use of three control values defined in the action refinement. The three control values of the action refinement are:

1. A value to specify the number of actions created with parameter values towards the maxima of the goal specification.

2. A value to specify the number of actions created with parameter values towards the minima of the goal specification.
3. A value to specify the difference between two neighbouring actions' parameter value.

The introduction of these three control values eliminates the need for the context-aware system to employ a brute force method which will evaluate every parameter value between the minimum and maximum goal specification. Since there's no correlation between the action system and the context space, the size of the action space is controlled independently with these control values. It is assumed that values for these control parameters are set based on the expected outcomes and the domain where the context-aware application is used in. The total number of actions to be executed and evaluated is determined by how large or small the three control parameters are set for, which allows fine tuning of the action execution. Besides prepopulating of the knowledge base with facts using system developers expertise of the domain, these five values (the two goal specification values and the three action refinement values) are the only values required from the system developers.

The action space built using the five values defined in the action refinement mentioned earlier. Once the action space is built it is handed over for concurrent multi-action experimentation. The generic framework proposes a private workbench for the concurrent action execution. A private workbench is an experimental area where the current state of the system is shielded from the effects of the action under evaluation. Unlike the iterative approach used by existing self-adapting context-aware systems, the generic framework proposed by the research uses concurrent action execution to evaluate all actions at the same

time. The iterative approach where actions are evaluated sequentially will have a time complexity of $O(n)$. The concurrent action evaluation will have a time complexity $O(1)$ as actions are executed independently of each other. This reduction in time complexity translates to short self-adaptation time. Depending on the action's parameter the execution end time of each action may vary. In context-aware application domain it's expected that all action would complete around the same time. But this is no guarantee as depending on the domain the framework is implemented, certain parameter values could make some action take longer to complete than others. To overcome this a cut-off time for experimentation could be introduced. Any action that has not completed by this cut-off time could be removed from being considered for the action evaluation phase. Introduction of the cut-off time could further reduce the time for adaptation.

There are research attempts to parallelize the adaptation [142] with the use of parallel optimization techniques, such as particle swarm optimization (PSO) [143]. However, there is a key difference between the PSO based technique and the concurrent multi-action evolution method of this research. In the PSO technique each particle must update their velocity and position relative to the particle with the global optimal for iteration of the optimization task. In the proposed framework, each action (similar to the particle in PSO domain) is treated with equal importance and is a candidate to become a "best adaptive action". Moreover, all actions evaluate the problem space independent of other actions (particles) without any relation to other actions in the action space.

At the end of the concurrent experimentation phase the outcome of each action is evaluated and the best action for the unknown context is chosen. The

action outcome evaluation component contains the optimization centric function used for deciding the best action based on the action outcomes. The evaluation function used is domain specific. For example, the context-aware application used to determine the best threshold price in a name-your-own-price (NYOP) e-commerce site would be looking to choose an action that maximizes the future profits (NYOP case study is presented in Chapter 5), whereas a context-aware application used for performance tuning may look for minimizing resource usage such as CPU and memory (case study presented in Chapter 4).

3.3 Formal Modelling

A formal model of the proposed generic framework's action system is presented, which could be used as a blueprint for any domain-specific implementation.

3.3.1 Goal Specification

As mentioned earlier, the goal specification encapsulates the extremities of the parameters used in the action space. The system developer is expected to specify the minimum and the maximum of the parameter values that the context-aware application must use for concurrent action execution. These extremities are denoted as G_{lo} for minimum and G_{hi} for maximum. The goal specification values are considered elements of the action's parameter space, which is used to differentiate one action from another.

$$(G_{lo}, G_{hi}) \in \{action\ parameter\ space\}$$

3.3.2 Action Refinement

The action refinement limits the number of actions used for concurrent multi-action evaluation. It builds the action space using the action of the closest known

context as the seed. This seed action is denoted as A_k and defined as a function of its outputs $parameter_k$.

$$Seed\ action = A_k(parameter_k)$$

The three control values of the action refinement component are denoted as

1. The lower bound expansion range denoted by p , which specifies the number of actions to define in the direction of G_{lo} .
2. The upper bound expansion range denoted by q specifies the number of actions to define in the direction of G_{hi} .
3. The distance between two neighboring actions' parameter values denoted by Δ .

These are the only inputs that depend on system developers and are based on their knowledge of the application domain. In practice the values dictate how many actions the user wants the context-aware application to consider for concurrent multi-action evaluation. With these values in place, the action space for concurrent multi-action evaluation could be defined as a union of three sets.

$$\begin{aligned}
 \text{Action space} = \{ & A_k(parameter_k) \cup \\
 & A_p(parameter_p) \cup \\
 & A_q(parameter_q) \\
 | & p = \{1 .. n\}, n > 0, q = \{1 .. m\}, m > 0, \\
 & parameter_k - p\Delta \geq G_{lo}, \\
 & parameter_k + q\Delta \leq G_{hi}, \\
 & \Delta > 0 \\
 & \}
 \end{aligned}$$

Figure 5 illustrates how the five parameters in action refinement come together to create the action space. As mentioned previously the creation of the action space starts of by identifying the seed action (denoted $A(k)$, black arrow). Around this other actions are generated with equal distance of Δ between each neighbouring actions' parameter values. The expansion continues until p and q

number of actions are created on either side of $A(k)$. These are denoted in green arrows in Figure 5. If any of the parameter values within an action falls outside of G_{lo} or G_{hi} then these are not considered as part of the action space. In Figure 5 these are denoted by the red arrows. From this it is clear that by adjusting the five parameters in the action refinement users of the context-aware application can fine tune the action space used for concurrent multi-action evaluation.

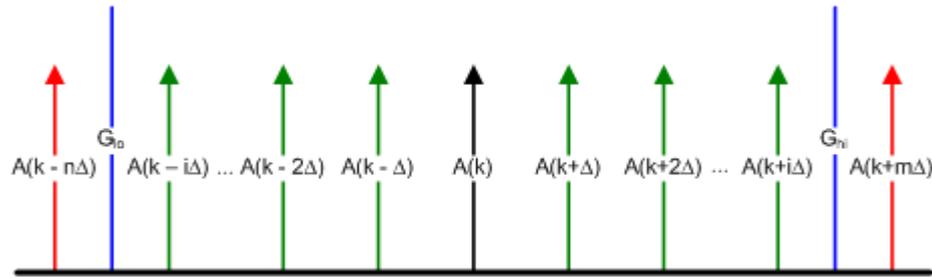


Figure 5. Action Space Creation

3.3.3 Concurrent Multi-Action Execution

As implied by the name, the concurrent multi-action execution is to execute all the actions in the action space at the same time. The research uses the notation used by Vinh [88] for autonomic systems (AS) to formally model the parallel execution of the action space.

$$AS_1 \begin{array}{c} \xrightarrow{\text{self-*action}_1} \\ \xrightarrow{\hspace{1.5cm}} \\ \xrightarrow{\text{self-*action}_n} \end{array} AS_2$$

Self-*action[1-n] are parallel self-adaptive actions executed in the concurrent action space while AS_1 and AS_2 are two states of the autonomic system. One key aspect is that the system adaptation is carried out in private workbench and is opaque to the users until the action outcome evaluation has taken place.

3.3.4 Action Outcome Evaluation

As mentioned earlier, the evaluation criteria for choosing the action that results in the highest benefit depends on the user expectations and the domain in which the context-aware system is implemented.

Thus the best parameter to use with the adaptive action executed as a result of the unknown context change could be formally defined as

$$\begin{aligned} parameter_{best} = & \{ \\ & \forall parameter_i \in \{action\ space\ parameters\} \\ & \exists A_i(parameter_i): \text{Maximum Benefit } (A_i) \\ & \} \end{aligned}$$

What above formal definition means is that for all parameters that were in the actions that made up the action space, there exists one parameter such that delivers the maximum benefit (best outcome) out of all the concurrently executed actions.

The best parameter to use with the adaptive action is the one that yields the highest benefit in line with the application's goal expectations. At the time of the experimentation the user is unaware of the optimal goal outcome. What is expected from the system is to adapt so that the outcome is the best possible under current context.

3.4 Application of Framework

As mentioned in the Section 3.1 the framework has been designed to be implementation and domain agnostic. Figure 6 shows application of the framework to achieve context-aware adaptation through concurrent multi-action evaluation. Figure 6 in this case depicts a hypothetical context-aware application which acquires context attributes from three sources. One is through social media APIs, the other from mobile phones and lastly set of hard sensors. The initial

context boundary consists of these three sources. The context-aware application applies the adaptive action on the adapted system (or system managed by context-aware application).

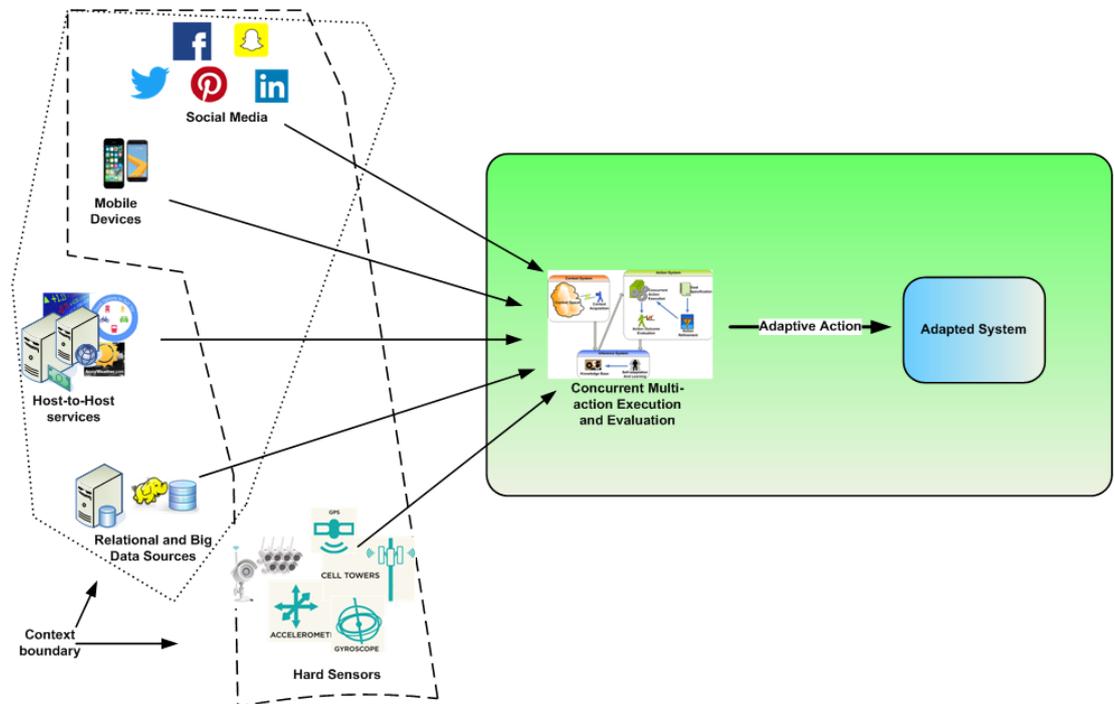


Figure 6. Application of Framework to Achieve Context-aware Adaptation

In the early days of context-aware computing the context attributes were acquired primarily from hard sensors. Examples of such context-aware applications were mentioned in Chapters 1 and 2.

The smart phones enabled easy access to context information while being mobile. In fact both Apple and Google have numerous products [210] that customize the services based on user's context. The product customization is not based just on the location of the user and includes other aspect such as personal preference, past usage.

Somewhat new into the mixture is social network based context gathering. Users of social networks are increasingly sharing personal information which both implicitly and explicitly gives contextual information about the user. This social media based context information has been used to enhance communication between

user groups [211], create augmented reality that is context-aware [212] and predict future events which has been used to detect potential crime [213, 214].

As mentioned earlier the initial context boundary consists of above three context sources. Section 3.2.1 described how the context boundary would change when new context sources become relevant and existing context sources are made irrelevant. This is shown in Figure 6 with the second context boundary which brings in Host-to-Host (H2H) services and relational and big data sources as context sources. At the same time the hard sensors are made irrelevant thus placing them outside the context boundary. XML/SOAP, JSON, RESTful APIs are used to acquire context information from external sources to create mashup web services. Example of a mashup web services which act as soft sensors for context-aware application was mentioned in [92]. How the framework handles removing and adding of context attributes was described previously in Section 3.2.1.

As the knowledge base does not have any facts with new context attributes, context inference will result in concurrent multi-action execution. Creation and execution of concurrent action space was explained on Section 3.2.3. As a result of the concurrent action evaluation the best adaptive action for prevailing context will be applied to the adapted system. In Chapter 4, the case study shows use of context-aware application to adapt a database to different work load mixes while in Chapter 5 the context-aware application adapts a hoteliers NYOP channel.

3.5 Blueprint for Framework Implementation

The nature of the concrete implementation of the framework is highly subjective and depends largely on the domain it is been implemented. Apart from the domain the programming languages used (i.e. object oriented, sequential programming) and aspect such as implementation architecture (i.e. object oriented architecture, aspect

oriented architecture, service oriented architecture) will heavily influence the implementation strategy. Therefore it is a difficult task to have a “one size fits all” implementation of the framework. However, considering the core systems and modules it is possible to give a generic blueprint which could be further expanded to fit the domain specific constructs. Figure 7 shows a UML diagram which captures the core systems and modules and interaction between them.

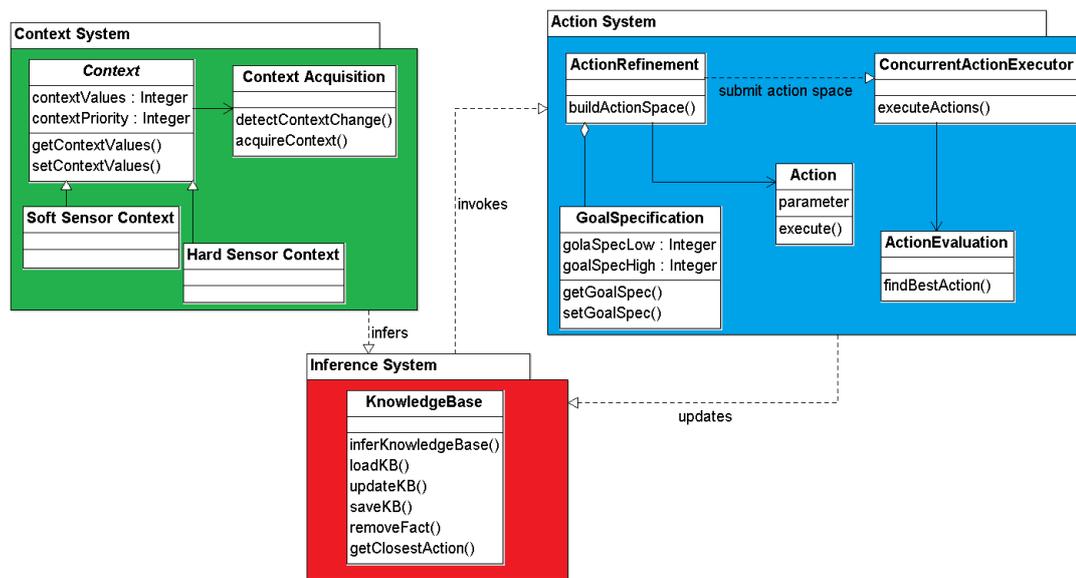


Figure 7. Blueprint for Generic Framework Implementation

The three packages in the UML have a one-to-one mapping to the three system described in Section 3.2.

The context system package in the diagram consists of four classes. An abstract context class which has two attributes which are context values and context priority. The abstract context class could be extended to specialist sub classes of soft sensor context and hard sensor context. These could be further sub classed based on the concrete implementation. The context acquisitions class the other class in the package. This class could be used for acquiring and detecting context changes.

The inference system package contains a single knowledge base class. The blueprint only specifies the operations on the knowledge base. The developers have the freedom to choose the mechanism for storing knowledge facts that is most convenient for the actual implementation. To demonstrate this capability, the case studies provided in Chapter 4 and 5 uses two different mechanism for storing knowledge facts.

The action system package consists of five classes. At the centre of it is the action class. It has the method for executing the action assigned to it and the attribute to hold the value of the parameter on which action is parametrized (parameterized action is explained on Section 3.2.3). The other four classes have a one-to-one mapping to the modules in action system on Figure 4. The action refinement class builds the action space using values from goal specification. The actions in the action space are concurrently executed once submitted to the action executor. The concurrent action executor simulates a private workbench area for experimentation where actions under evaluation do not affect the system state. The action evaluation class find the best course of action and update the knowledge base with the context-action pair facts.

What's omitted from the action system package is the applying of best action on the adapted system as depicted on Figure 6. This is highly dependent on the implementation and nature of the interface between context-aware application and the adapted system.

Once the generic framework was formed, the next step in the research was the implementation of a context-aware system based on the proposed framework. Two use cases from two different domains were used for this purpose. Chapter 4 and 5 of the thesis detail these case studies.

**4. CONTEXT-AWARE
APPROACH FOR
EXPERIMENT-BASED
DATABASE
PERFORMANCE TUNING**

4.1 Background

Performance problems in the database system (DBS) could have a cascading effect on all aspects of an enterprise application. Database vendors have provided some help in identifying and resolving performance regression by way of automation and manageability [144]. But ultimately it is the responsibility of the database administrator (DBA) to fine tune the DBS to perform optimally. A properly tuned DBS is more likely to enable a DBA to achieve the service level agreements (SLA) than a non-tuned DBS.

However, database performance tuning is not a one-off task. A DBA has to constantly keep an eye on the DBS performance as a multitude of reasons could cause the tuning work carried out earlier to become invalid or inadequate. Changes in hardware or faulty hardware related issues (i.e. failed memory banks reducing the memory available to the system, network bandwidth reduction, failed I/O controllers reducing I/O bandwidth) could easily be detected with modern monitoring systems and replacing the component will rectify the issue. But there are other factors that pose much more subtle performance problems. This type of changes include database upgrades (i.e. change in optimizer behavior), change in configuration parameters (i.e. depreciated configuration settings or change in a default value set at installation), change in workloads and business requirements (i.e. a DBS system used for online transaction processing (OLTP) now being used for transaction processing and decision support system (DSS)), stale statistics on data or a combination of these factors. As a result, the database query performance could regress, remain unaffected or improve [145].

Figure 8 and 9 show a real-world example¹ of performance regression due to a runaway query consuming high CPU. Figure 8 shows the CPU usage at the time of the incident, while Figure 9 shows the CPU consumptions on that hour compared with the hourly CPU consumptions over a 3-month period.

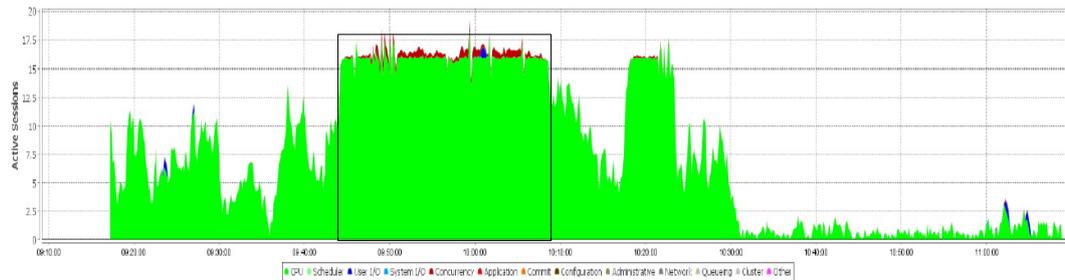


Figure 8. DB CPU Usage During Performance Regression

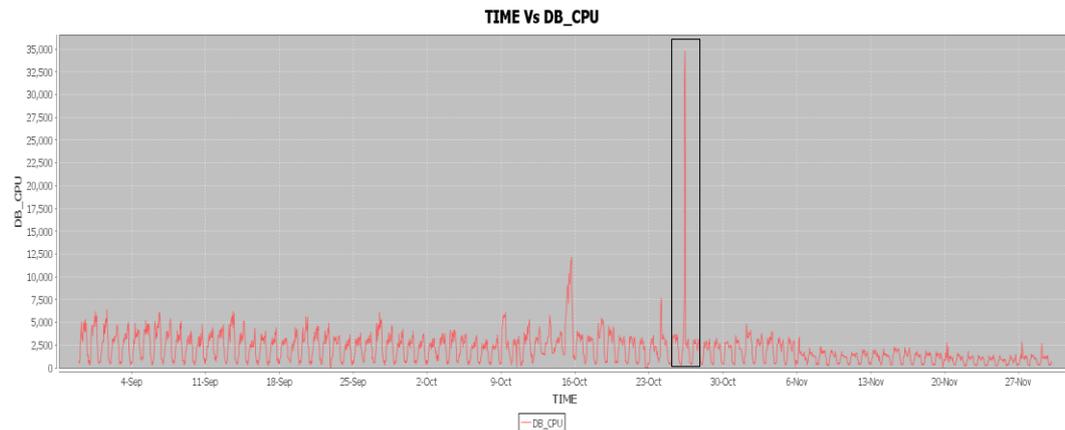


Figure 9. Hourly CPU Usage on the DB Over 3 Month Period

If any of the aforementioned reasons result in performance regression, the DBA would have to rectify it. Prior to starting performance tuning activities, the DBA prioritizes which performance tuning tasks to carry out first. This prioritization is done based on a prediction the DBA makes as to which tuning action would yield the highest performance benefit with the lowest cost. The cost could be in the form of monetary losses due to regressed performance, as well as time limit imposed by SLA to resolve a performance issue. The DBA uses her knowledge of the DBS, past experiences of handling a similar situation and even

¹Graphs obtained by the author as part of his work at CodeGen Ltd (www.codegen.co.uk).

intuition to make this prioritization. However, this prediction may not always be accurate, thus resulting in elongated performance resolution time.

As there are many factors to consider, it is impossible for a DBA to predict the performance changes without actually trying the changes on a system [146]. Therefore, the experiment-based performance tuning models give a DBA an accurate understanding on the returns of the tuning effort as opposed to predictive models [147] which consider data statistics or algorithmic complexities. In such cases, the DBA would create a replica of the production database, run a similar workload to recreate the problem behavior and then hypothesizes the root cause and the set of potential solutions and evaluate them one by one with different sets of experiments [148,149,150]. Experimenting with a representative system or on the production system itself allows a DBA to accurately gauge the performance gain for a particular tuning task before applying it to a production DBS.

There are similarities between how a context-aware system reacts to a context change and how a DBA employs an experiment-based performance tuning model. Self-adapting context-aware systems have three main aspects, context sensing, actions or actuators and self-adaptation/self-learning based on action outcomes. These aspects could be superimposed to experiment-based DBS performance tuning as follows. The DBA has to monitor the DBS to determine if the performance has regressed (i.e. context sensing). Performance regression is detected by way of increased query execution time, high user response time, change in the CPU pattern, memory usage patterns, etc. (i.e. context inference). The DBA would carry out a tuning action depending on the symptoms. If the DBA is aware of the reason for the current performance change, this would be a

known context. Since there are many factors that could affect the database performance, the DBA may not always know the root cause outright (i.e. unknown context). In an unknown context, DBA would need to engage in multiple tuning experiments (i.e. actions in the context-aware model), evaluate the outcome of them all and then would decide the most beneficial tuning activity to apply on the production DBS. Once a fix is found, the DBA would be knowledgeable to detect similar situations in the future and fix them quickly, the same way a context-aware system uses self-adaptation and evolves to recognize more and more contexts.

4.2 Current Approaches

Experiment-based database performance tuning models are actively developed and researched into by both the academic initiatives and by database vendors. Among the related work, two types of experiment-based database performance models could be identified. The first model type is where experimentation takes place offline or offloaded from the production system, while the second model runs experimentation on the production system itself in real-time. During a performance crunch, configuration parameter tuning could give the quickest fix with minimum effort. This is because if the effect of changing the configuration parameter is already known then the change could be implemented without any pre-processing activities. Depending on the configuration parameter database may not even need to be restarted, ensuring no downtime for the application. After this initial breather, the DBA could further investigate the root cause of the performance regress and provide a more substantiated solution. But modern databases have hundreds of parameters and deciding which parameter to tune and what value to set for it requires careful testing. A wrong parameter or parameter

value could worsen an already bad situation. Database vendor literature provides insight into the parameters a DBA is most likely to use for tuning the workload. This vendor literature could be considered as guidelines but may need to be tested and validated for each individual application.

A framework proposed in [151] provides a DBS independent experiment-based approach to database tuning via configuration parameter settings. With the use of adaptive sampling, it identifies and brings high impact and high-performance configuration parameters into the experiment workbench. Using a cycle-stealing paradigm it executes experiments directly on the production database for accurate gauging of the benefits to configuration parameters changes. Another configuration parameter related framework is presented in [149], in which starting off with a small number of experiments, the framework expands the experiment base depending on the estimated benefits from each experiment. A rapid experiment-defining framework and a high-level language that enable the DBA to define experiments using the framework are proposed in [152]. The framework orchestrates the scheduling, running and tuning of the system in an automated fashion. The DBA is free to devise experiments to check the impact of configuration changes. An offline experiment-based performance model is presented in [153], which trains neural network to recognize the workload patterns on a test system before letting it predict performance on the production database. As training of the neural network is required beforehand, the dynamic adaptation to performance workloads that were not in the training set could be unpredictable. The neural network internal weight adjustments are opaque to a DBA, thus validation of the prediction would require a DBA to carry out further tests.

If tuning cannot be achieved through configuration parameter changes, then the next option for achieving good performance is for a DBA to tune the relevant database queries. To successfully tune queries a DBA must be aware of the characteristics of the data being queried, statistics and the business requirements. In this regard, an experiment-based SQL tuning framework is presented in [148]. Using cardinality sets the framework quickly develops new plans that are in the same neighborhood but with better execution plans. Cardinality sets represents the cardinality values that are needed for costing a particular SQL execution plan. The approach used by Oracle is based on an initial SQL plan which is considered the baseline, while other plans are generated and compared against this baseline plan [145, 154]. If better plans are found, then they are added to the baseline after verification allowing query performance to always improve and never regress beyond the baseline. Oracle has automated this tuning process with the use of automatic tuning optimizer [144] in Oracle database versions of 11g and with automatic re-optimization techniques in 12c [155].

A vendor-specific experiment-based performance tuning tool is described in [146], called SQL Performance Analyzer. The SQL Performance Analyzer (SPA) was introduced by Oracle in the 11g database version and allows the DBA to measure performance changes or the impact of configuration parameter changes, database version upgrades, updating statistics, and the creation of database objects such as indices and to materialize views. SPA could be used on the production database or a SQL set from production could be transferred to a test system to be used with SPA. However, SPA does not provide any way to automatically run the tests and requires the DBA to execute each experiment. A more robust workload capture and replay system is presented in [156, 157]. The

DBA could capture the production workload with minimum overhead and use the captured workload to conduct experiments on a test system (i.e. offline) to the same level of concurrency as the production system. The database replay works for proactive performance tuning but is not sufficient for reactive performance tuning as the workload capture has to happen before the problem period.

4.3 Formal Modelling

The proposed framework was adapted to implement a context-aware experiment-based database performance tuning system as shown in Figure 10.

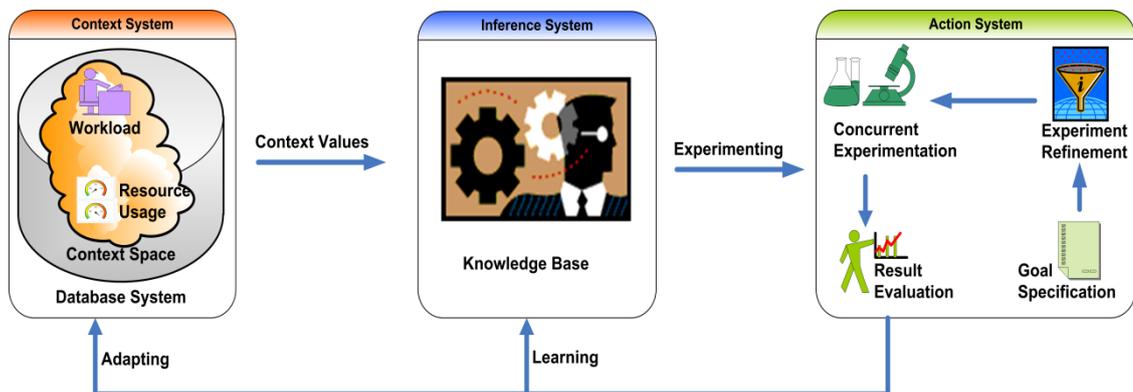


Figure 10. Adaptation of Context-Aware Framework for Experiment-Based Performance Tuning

The three systems (context, action, and inference) of the proposed framework are mapped to the experiment-based database performance tuning case study as follows. Following the definition in [29], which was adopted by the research, the database system (DBS) is nominated as the entity that is of concern and the user workload W and the resource usage R are defined as the contexts describing the state of the DBS. At any given time, the workload W could consist of zero or more user workload types. Zero workload type represents no user work being done on the database. The background work needed for the upkeep of the DBS is not

considered as this is of no practical use or concern for the tuning effort. The resource usage R describes the level of consumption of system resources by the DBS at a given point in time. The resource types include CPU, memory and DBS memory components such as buffer pools, I/O utilization, network utilization and any other resource type the DBS uses to service the workload.

The context acquisition is responsible for sensing and acquiring these context values. Based on the context values acquired the context system decides if a context change has occurred. How the context sensing happens is implementation specific. However, as with any instrumentation activity, care must be taken not to induce any overhead due to the context sensing. The CPU cycle stealing mechanism in [151] is one possible mechanism that could be employed for context value sensing. The cycle stealing mechanism uses idle computing capacity to carry out context sensing work with near zero overhead on production loads. Furthermore, the context acquisition is expected to transform the heterogeneous context value types in multiple units into a single unit of measurement, allowing comparison of contexts. This context comparison is used in the action system to find the closest known context to an unknown context. Two workload types were defined for W ; $W=\{OLTP,DSS\}$. The online transaction processing (OLTP) workload type has the characteristic of accessing a small percentage of tuples on few tables for a given query. The Decision Support System (DSS) workload types are ad-hoc queries that access a large percentage of tuples in multiple tables for management information or business intelligence reports. The CPU load for each workload type was considered as the resource usage R ; $R=\{CPU_{OLTP}, CPU_{DSS}\}$. It's not uncommon for both these workload types to be present in the same DBS. It is typical in enterprise scenarios that the usage pattern remains constant over

time as the application uses a similar set of queries [153, 158]. Therefore, any change to the workload type or the resource usage pattern is considered a context change and context inference is carried out.

The action system is responsible for concurrent action execution and evaluation when an unknown context is encountered. The goals of the action system are to reduce the number of required actions (experiments) and to complete the actions in a single pass as opposed to iterative manner. To achieve this first goal the action system uses goal specification and action refinement. The goal specification defines the extremities of the variable parameter used in the actions. In this implementation, the performance tuning is carried out by way of database configuration changes. The experimentation consisted of executing a representative workload using a different set of configuration values for each experiment. The representative workload is created by capturing the top resource consuming queries for each workload type. The goal specification, in this case, would be the minimum and maximum values to be used in the experiments, but not necessarily the maximum and minimum values the parameter could be configured for. These extremities are denoted as G_{lo} and G_{hi} and are considered elements of the configuration parameter space.

$$(G_{lo}, G_{hi}) \in \{ DB \text{ configuration parameter space} \}$$

The action refinement limits what action qualifies to be in the action space. Without the limiting effects of the action refinement, the context-aware system would have to experiment with every value between G_{lo} and G_{hi} which would be a resource and time intensive endeavour. The action limiting process starts by identifying from the knowledge base, the context that is closest to the unknown

context. The closeness is measured by the difference of the context values. If more than one context is found to be the closest, then the priority of each context is considered. The configuration parameter setting of this known context is used to devise the initial action. This is denoted as A_k and defined as a function of the configuration parameter $configuration_k$ of the closest known action.

$$\text{Initial action} = A_k(DB\ configuration_k)$$

To derive other actions in the action space three integer parameters are introduced. They are the lower bound expansion range denoted p , which specifies the number of actions to define in the direction of G_{lo} . The upper bound expansion range denoted by q specifies the number of actions to define in the direction of G_{hi} and finally the distance between each configuration parameter is denoted by Δ . These three parameters and the goal specification are the only inputs that depend on system developers, effectively eliminating the need to identify all possible context changes. Having defined these values, the total number of actions (or experiments) which need to be executed could be defined as a union of three action sets.

$$\begin{aligned} \text{Action space} = \{ & A_k(DB\ configuration_k) \cup \\ & A_p(DB\ configuration_p) \cup \\ & A_q(DB\ configuration_q) \\ | & p = \{1 .. n\}, n > 0, q = \{1 .. m\}, m > 0, \\ & DB\ configuration_k - p\Delta \geq G_{lo}, \\ & DB\ configuration_k + q\Delta \leq G_{hi}, \\ & \Delta > 0 \\ & \} \end{aligned}$$

The defined actions are then executed concurrently in a private workbench. The private workbench ensures that the configuration changes in each action are

opaque to and do not affect the current state of the DBS. As all actions are executed concurrently, the outcome of each action is known at the same time as opposed to an iterative approach where the analysis of the results has to be delayed until all actions have finished.

The final phase of the action system is the outcome evaluation. The evaluation criteria for choosing the action that result in the highest benefit depend on the domain the context-aware system is implemented in. For the database performance, the tuning actions are evaluated based on a minimizing function. That is, the best action is the one with the configuration parameter that resulted in minimum resource usage such as the CPU usage for the execution of queries, the number of data blocks accessed, memory utilization or even the response times or a combination of metrics as used by [146]. Thus the best configuration parameter for the unknown context could be formally defined as

$$\begin{aligned}
 DB\ configuration_{best} = \{ \\
 \forall DB\ configuration_i \in \{DB\ configuration\ parameters\ in\ action\ space\} \\
 \exists A_i(DB\ configuration_i):\ minimum\ (Resource\ Usage\ (A_i)) \\
 \}
 \end{aligned}$$

Once the best setting for the configuration parameter is known for the current context, it could be used to update the context-aware system so it recognizes this context in the future (learning and adaptation) and at the same time, the DBS configuration setting is changed to optimize the workload execution.

4.4 Implementation

The context-aware experiment-based performance tuning system was developed as a Java desktop application. The key components of the Java and OWL implementations of the three systems (context, inference, and action) in the research framework are given in the following sections.

4.4.1 Context System

The context-aware system would sense the context-space for any changes. As mentioned in an earlier section, the context space consists of resource usage and workload. Two Oracle DB connection services were created in the database for each of the workload types $W=\{OLTP,DSS\}$, thus enabling the measuring of the resource usage $R=\{CPU_{OLTP}, CPU_{DSS}\}$ of each of these workload types.

Constructor and class variables of the Context class are shown in Listing 1.

```
public class Context {  
  
    private int oltp; //oltp load  
    private int dss; //dss load  
    private String name;  
    private String workloadType;  
  
    private Connection oltpConnection;  
    private Connection dssConnection;  
  
    private int C1_PRIORITY=1;  
    private int C2_PRIORITY=2;  
  
    private int optimizer_index_cost_adj;  
  
    public Context(int c1, int c2, int optiIndCostAdj){  
  
        this.oltp = c1;  
        this.dss = c2;  
  
        this.optimizer_index_cost_adj = optiIndCostAdj;  
  
    }  
}
```

Listing 1. Context Class for DB Performance Tuning Case

Context sensing was carried out by polling the database every 5 seconds and evaluating the CPU usage of each service assigned to the workload types. This polling only incurred 0.000015% of the total hourly CPU time available on the DBS server, which is considered to be negligible. When new workloads are added to or removed from the DBS, the resource usage pattern would vary based on the final workload that exists in the DBS. This change in the resource usage pattern is considered a context change if there was a 3% difference in the CPU usage

between 3 consecutive samples. The value of 3% was based on DBA expertise of the database application. Furthermore, with the help of the DBA's understanding of the domain, it was found that a 5-second polling interval and the 3 samples gathered provided the ideal polling frequency and sampling rate to avoid taking too long to detect a context change versus introducing an additional overhead to the system.

Listing 2 shows the Context acquisition class which was implemented as a runnable java thread (Full code listing is given in Appendix A). It shows the continuous sampling of the DB workload and detecting a context change when there's a 3% difference on the load.

```

public class ContextAcquire extends Thread {

    private Connection con;
    private final String SQL = "select service_name,value from
                                v$sqlservice_stats where stat_name='DB CPU'
                                AND SERVICE_NAME IN ('dsssrv','oltpsrv)";
    ...
    private GUIObjects guiObj;

    public ContextAcquire(GUIObjects guiObj) throws SQLException {

        con = DBConnectionPool.getDBConnection(DBConnectionPool.POOL);

        valueMap.put(CURRENT, new ContextValue());
        valueMap.put(PREVIOUS, new ContextValue());
        this.guiObj = guiObj;
        setName("Context Acquire");
    }

    @Override
    public void run() {

        while (true) {

            if (i == 0) {
                value = valueMap.get(PREVIOUS);
            } else {
                value = valueMap.get(CURRENT);
                twosamplecollected = true;
            }
        }
        ...
        long oltpValue = 0, dssValue = 0; // no load on the DB

        if (i == 1) {
            ContextValue before = valueMap.get(PREVIOUS);

```

```

        oltpValue = ((value.getOltpValue() -
        before.getOltpValue()));
        dssValue = ((value.getDssValue() -
        before.getDssValue()));
    }

    if (i == 0 && twosamplecollected) {

        ContextValue current = valueMap.get(CURRENT);
        oltpValue = ((value.getOltpValue() -
        current.getOltpValue()));
        dssValue = ((value.getDssValue() -
        current.getDssValue()));
    }

    if (dssValue > 0 || oltpValue > 0) {

        BigDecimal oltpBD = new BigDecimal(oltpValue);
        BigDecimal dssBD = new BigDecimal(dssValue);
        BigDecimal oltpPct =
        oltpBD.multiply(hundred).divide(oltpBD.add(dssBD),
        mc).round(mc1);

        boolean isContextChanged =
        hasContextChanged(oltpPct.doubleValue(),
        dssPct.doubleValue());
        ...
    }

public boolean hasContextChanged(double oltp, double dss) {

    if (preDSSValue < 0) {

        prevOLTPValue = oltp;
        preDSSValue = dss;

        return true;
    }

    double ot = Math.abs(oltp - prevOLTPValue);
    double ol = Math.abs(dss - preDSSValue);

    if (ot >= 3 || ol >= 3) {

        ++samples;
    }

    if (samples == 3) {

        samples = 0;
        prevOLTPValue = oltp;
        preDSSValue = dss;
        return true;
    }

    return false;
}
}

```

Listing 2. Context Acquisition and Context Change Detection

4.4.2 Inference Systems

The knowledge base was modelled using java implementation of Protégé OWL API and for inference Java API for Protégé Pellet Reasoner was used.

The context inference is carried out by the inference system which consists of a knowledge base and a self-adaption and learning mechanism. When a context change is encountered, the knowledge base is queried to identify if the context is known. If the inference system is unable to find the context in the knowledge base, the action system is invoked. The other component in the inference system is the self-adaptation and learning mechanism which updates the context-aware system and the knowledge base with the outcome from the action system.

The knowledge base represents each context in terms of the earlier defined context values, which is workload type and associated resource usage. The knowledge base was modelled using web ontology language (OWL). The objective was to have an OWL representation of context [159] that would allow leveraging of OWL inherent inference capabilities for context inference. With the use of the OWL only the distinct workload types need to be defined, in this case the OLTP and DSS workload types. Taking into consideration the resource usage distribution of the current context, the DBS is said to have an OLTP workload type if there is no resource usage for DSS and all existing resource usage is for serving the OLTP queries. The Listing 3 illustrates how this is defined using OWL restriction.

```
<owl:Class rdf:ID="OLTP">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:FunctionalProperty rdf:ID="oltpLoad"/>
      </owl:onProperty>
      <owl:hasValue datatype="int">100</owl:hasValue>
    </owl:Restriction>
  </rdfs:subClassOf>
</rdfs:subClassOf>
```

```

<owl:Restriction>
<owl:onProperty>
<owl:FunctionalProperty rdf:ID="dssLoad"/>
</owl:onProperty>
<owl:hasValue datatype="int">0</owl:hasValue>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>

```

Listing 3. OWL Class Definition for OLTP Workloads

Similarly, the current context is said to have a DSS workload if there is zero usage for OLTP queries and all existing resource usage is for serving the DSS queries. OWL class for DSS workload type is presented in Listing 4.

```

<owl:Class rdf:ID="DSS">
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty>
<owl:FunctionalProperty rdf:ID="oltpLoad"/>
</owl:onProperty>
<owl:hasValue datatype="int">0</owl:hasValue>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty>
<owl:FunctionalProperty rdf:ID="dssLoad"/>
</owl:onProperty>
<owl:hasValue datatype="int">100</owl:hasValue>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>

```

Listing 4. OWL Class Definition for DSS Workloads

Using OWL it is possible to model other workload types as complex classes based on the distinct workload types OLTP and DSS defined earlier. The current context is said to have a mixed workload if the resource usage has an OLTP portion as well as a DSS portion. This is defined in OWL using OLTP and DSS types as following. The OWL specification for Mix workload type is given in Listing 5 below.

```

<owl:Class rdf:about="#Mix">
<owl:disjointWith rdf:resource="#DSS"/>
<owl:disjointWith rdf:resource="#OLTP"/>
<rdfs:subClassOf>
<owl:intersectionOf rdf="Collection">
<owl:complementOf rdf:resource="#DSS"/>
<owl:complementOf rdf:resource="#OLTP"/>
</owl:intersectionOf>
<rdfs:subClassOf
</owl:Class>

```

Listing 5. OWL Class Definition for Mix Workloads

Mix workload type is specified as the complement of both the DSS workload type and OLTP workload type, which could be inferred as a system having CPU usage on OLTP and DSS workloads. Though the mix workload type is created as a disjoint of DSS and OLTP, this definition alone isn't enough to infer a mix workload type. This is because OWL currently adopts a standard logical model of open world assumptions. Under open world assumptions, a statement cannot be assumed true on the basis of failure to prove it [160].

To overcome this limitation of OWL two rules using SWRL (Semantic Web Rule Language) were created. With the use of OWL + SWRL the context-aware system was able to correctly infer that the perceived load is OLTP, DSS or a mixture of both.

```

Workload(?x) ^ dssLoad(?x,0) ^ oltpLoad(?x,100) → OLTP(?x)
Workload(?x) ^ dssLoad(?x,100) ^ oltpLoad(?x,0) → DSS(?x)

```

Listing 6. SWRL Rules for OLTP and DSS Workloads

The SWRL rules refer to a “workload” class which is a generic workload type defined as a union of all three aforementioned workload types. The OWL specification of this generic workload type is given below. The full OWL + SWRL definitions are presented in Appendix B.

```

<owl:Class rdf:ID="Workload">
<rdfs:subClassOf>
<owl:Class>

```

```

<owl:unionOf rdf:parseType="Collection">
<owl:Class rdf:about="#Mix"/>
<owl:Class rdf:about="#DSS"/>
<owl:Class rdf:about="#OLTP"/>
</owl:unionOf>
</owl:Class>
</rdfs:subClassOf>
</owl:Class>

```

Listing 7. OWL Class Definition for Workload Type

Because of this generic workload type specification, the domain or application specific workload types (e.g. Order search workload, sales report workload and etc.) could be inferred upon and deduced to one of the three aforementioned workload types, thus allowing the implementation of the framework to be independent of any domain specific constructs. Unlike the other context-aware systems, where developers had to capture all possible context changes, in the proposed self-adapting model the knowledge base starts off with few known contexts. This initial knowledge could be derived from the DBAs knowledge of the system, past experience and even intuition (rule of thumb) and is not expected to be extensive. Once the knowledge base has sufficiently expanded, the DBA could even invalidate these initial knowledge base settings and let the system populate it with experiment-based results. This would ensure all knowledge facts are validated through experimentation not based on intuition of the DBA.

The following code listing shows Java implementations of loading and populating the knowledge base with initial knowledge and initializing rule engine for inference.

```

private void createReasoner() {
    reasoner =
    ReasonerManager.getInstance().createProtegeReasoner
        (owlModel, ProtegePelletOWLAPIReasoner.class);
}

private void createRuleEngine() throws SWRLParseException,
SWRLRuleEngineBridgeException, SWRLRuleEngineException {

```

```

    factory = new SWRLFactory(owlModel);
    bridge = BridgeFactory.createBridge("SWRLJessBridge",
        owlModel);
}

private void loadKnowledgeBase() throws OntologyLoadException
{
    owlModel = ProtegeOWL.createJenaOWLModelFromURI(URI);

    workload = owlModel.getOWLNamedClass("Workload");
    hasName = owlModel.getRDFProperty("hasName");
    dssLoad = owlModel.getRDFProperty("dssLoad");
    oltpLoad = owlModel.getRDFProperty("oltpLoad");
    optiIndCostAdj =
        owlModel.getRDFProperty("optiIndexCostAdjValue");

    for (Context c : mainList) {

        OWLIndividual w1 =
            workload.createOWLIndividual(c.getName());
            w1.setPropertyValue(hasName, c.getName());
            w1.setPropertyValue(oltpLoad, c.getC1());
            w1.setPropertyValue(dssLoad, c.getC2());
            w1.setPropertyValue(optiIndCostAdj,
                c.getOptimizerIndexCostAdj());

    }
}

```

Listing 8. OWL Model and Pellet Reasoner Loading

The context-aware application infers whether the perceived workload type is a known context or unknown context by dynamically executing SWRL against the knowledge base.

```

public ResponseObj checkContextKnown(Context c) throws
    SQWRLException, SWRLParseException, SWRLRuleEngineException {

    ResponseObj response = new ResponseObj();
    SWRLImp imp = factory.createImp("Query-1",
        "Workload(?x)  $\wedge$  dssLoad(?x, " + c.getC2() + ")  $\wedge$ 
        oltpLoad(?x, " + c.getC1() + ")  $\rightarrow$  sqwrl:select(?x)");

    bridge.infer();
    SQWRLResult result = bridge.getSQWRLResult("Query-1");

    while (result.hasNext()) {

        response.setChecked(true);
        OWLIndividual ind =
            owlModel.getOWLIndividual(result.getObjectValue("?x").toString());

        ...
        Context existingContext = new Context(oltp, dss, opti,
            contextName.toString());
        response.setContext(existingContext);
    }
}

```

```

        result.next();
    }
    imp.delete();
    return response;
}

```

Listing 9. Dynamic Inference of Workload Types

The self-adapting mechanism updates the knowledge base with newly discovered knowledge. Along with the update to persistent media, the context-aware application updates the in-memory OWL model with this newly discovered knowledge.

```

public void updateKnowledgeBase(Context unknownContext, Connection
    connection) throws SWRLRuleEngineException, SQLException{

    OWLIndividual w1 =
    workload.createOWLIndividual(unknownContext.getName());
    w1.setPropertyValue(hasName, unknownContext.getName());
    w1.setPropertyValue(oltpLoad, unknownContext.getC1());
    w1.setPropertyValue(dssLoad, unknownContext.getC2());
    w1.setPropertyValue(optiIndCostAdj,
        unknownContext.getOptimizerIndexCostAdj());
    bridge.infer();
    saveKnowledge(unknownContext, connection);
}

```

Listing 10. Updating of Knowledge Base

Appendix C has the full Java code listing of the knowledge base class.

4.4.3 Action System

When an unknown context is encountered, the action system would identify the high CPU consuming queries for each workload type and bring them to the private workbench for experimentation. In this case, the experimentation would be the concurrent execution of queries under various configuration parameter settings. The context-aware application adapts the database by setting the configuration parameter which results in the lowest resource consumption under current the context (i.e. workload mix).

Out of hundreds of parameters in Oracle, the configuration parameter `optimizer_index_cost_adj` [161] was chosen as the parameter to demonstrate how the proposed framework could be adopted for experiment-based performance tuning. In practice, when database tuning is done by way of configuration parameter changes it is carried out one parameter at a time. As such use of a single configuration parameter to demonstrate the context-aware approach is similar to a DBA carrying out the very first configuration parameter change on a long list of configuration parameters. The `optimizer_index_cost_adj` parameter influences the optimizer behavior based upon the value it is set for. By default, it is set to 100 in which case the optimizer counts the cost of using index access path same as using table access paths. Lower values make the optimizer to consider index access as a less costly operation compared to table access path and on the other hand, higher values make it favour full table scans over index access paths. However, there are no clear guidelines to setting this parameter in practice and DBAs are likely to leave it at the default value for fear of setting it to an incorrect value [162]. The rule of thumb is to set it to a lower value for OLTP workloads, which makes query plans bias for index access and set it to a higher value for DSS workloads, in favour of full table scans. But this rule of thumb may not work for every application, thus this parameter makes an ideal candidate for experiment-based performance tuning. In addition to the aforementioned reason, this parameter also made a good candidate to demonstrate the proof of concept for the following reasons.

1. This parameter affects the query execution plans by influencing the optimizer's decision to use indexes versus full table access. Based on the selected plan, the resource usage (CPU, memory, disk I/O) will vary between

each query and as such the resource usage of the query is indicative of how well it is performing. Therefore, this parameter gives a direct correlation between the resource usage of each plan and the parameter value being used. By comparing resource usage of various queries under different parameter value settings, it is possible to identify which parameter is beneficial to be used for the current context (i.e. workload type).

2. The value that could be set for the parameter has a wide range (1-10000) but for an enterprise application, the applicable range is much narrower, which is the concept demonstrated by the goal specification.
3. Adaptability to the action refinement model proposed in the framework modelling section allows the automated experiment creation and execution.
4. The parameter could be set at the system level or session level. System level setting is applicable to the entire DBS, while session level setting is visible only to the candidate session and opaque to other database sessions. As such it provides a private workbench to carry out the concurrent experimentation.
5. The setting of the parameter does not require the database to be restarted, allowing rapid application prototyping and experimentation without any delay.

Although `optimizer_index_cost_adj` is used here, any configuration parameter could have been used such as `optimizer_index_caching` [161] or `optimizer_dynamic_sampling` [161] with the context-aware model.

The private workbench for concurrent experimentation was implemented as a runnable Java thread which creates the action space using values for goal specification and action refinement.

```

public class ConcurrentActionExecutor extends Thread {
    ...

    @Override
    public void run() {
        ...

        Action closestKnownAction =
            knowledgeBase.getClosestAction(unknownContext);
        int optimizerIndCostAdj =
            closestKnownAction.getOptimizerIndexCostAdj();

        ActionRefinement refine = new ActionRefinement();
        ArrayList<Action> actionSpace =
            refine.getRefinedActionSpace(optimizerIndCostAdj,
            unknownContext, list, producerLock);
        ...
    }
}

```

Listing 11. Private Workbench for Concurrent Action Execution

Each action is also implemented as a runnable java thread, which allows multiple instances of the action class to be executed with low overhead. Each action class is encapsulated with the values of the unknown context and a configuration parameter. Each action has a unique configuration parameter to experiment with and SQL executed by one action is not influenced by the configuration parameters of other actions. The actions and the concurrent action executioner are arranged in a producer-consumer configuration, allowing fixed set of experimentation for all actions.

The goal specifications (G_{lo}, G_{hi}) were set as (25,250). Furthermore, values for (p, q, Δ) were set at (3,3,25). How the values of (p, q, Δ) related to the concurrent multi-action evaluation was explained in the previous formal model section.

These values were used to create the action space. Apart from any inputs to the knowledge base, these 5 values are the only input expected from developers or users of the context-aware application. This reduces the reliance on system developers to capture all possible context changes. The following listing shows the dynamic creation of action space for concurrent experimentations, starting off with the closest known context's configuration parameter.

```

ArrayList<Action> actionSpace = new ArrayList<>();

for (int i = getOptimisticCount(); i >= 0; i--) {

    if (optimizerIndCostAdj + (delta * i) > goal.getMaxValue()) {
        continue;
    }
    Action action = new Action(new Context(unknownContext.getC1(),
        unknownContext.getC2(),
        (optimizerIndCostAdj + (delta * i)),
        DBConnectionPool.getDBConnection(DBConnectionPool.
            OLTP),
        DBConnectionPool.getDBConnection(DBConnectionPool.
            DSS)),list, producerLock);
    actionSpace.add(action);
}

for (int i = 1; i < getPessimisticCount() + 1; i++) {

    if (optimizerIndCostAdj - (delta * i) < goal.getMinValue()) {
        continue;
    }

    Action action = new Action(new Context(unknownContext.getC1(),
        unknownContext.getC2(),
        optimizerIndCostAdj - (delta * i),
        DBConnectionPool.getDBConnection(DBConnectionPool.
            OLTP),
        DBConnectionPool.getDBConnection(DBConnectionPool.
            DSS)),list, producerLock);
    actionSpace.add(action);
}

return actionSpace;

```

Listing 12. Building of Action Space in Action Refinement Class

The system has the options of choosing either CPU, response time or the number of blocks accessed as the evaluation criteria for determining the best configuration parameter for the perceived context. For the experiments of this research, CPU usage was chosen as the evaluation criteria. The reason for this is

that as the experimentation is carried out concurrently there could be contention for shared resources (memory, I/O, network), which may increase the overall experimentation elapsed time. Furthermore, all experiments may not face the same level of resource contention as well. However, the CPU time represents the total time consumed by a query on CPU and not affected by contention issues and is provided by Oracle DB internal statistics gathering process [163]. Therefore comparing the CPU time instead of the elapsed time of each experiment eliminates any discrepancies due to resource contention.

4.4.4 Runtime Execution of Context-Aware Application

The following text describes the runtime execution of the context-aware application developed based on the proposed framework. At inception, the knowledge base was populated with an initial set of facts. These facts were not validated and they are used to represent the DBA setting the initial configuration based on the DBA's experience and intuition. These initial knowledge facts, shown in Figure 11 include one relating to OLTP only workloads, another DSS only workloads and one for mix workload type where the CPU usage was distributed (90:10) among OLTP and DSS workloads.

Context Name	OLTP Load	OLAP Load	Optimizer Value
OLTP_100_0	100	0	100
MIX_90_10	90	10	90
OLAP_0_100	0	100	20

Figure 11. Pre-Populated Knowledge Base

The context-aware application starts of by sensing the context (workload and resource usage) and infers upon the currently perceived context. The outcome of this inference has two possibilities; either the system has encountered a known

context or an unknown context. If the context is known and if the current configuration setting is not the ideal based on the knowledge base facts, the system adapts the database by setting the ideal configuration parameter.

The following section demonstrates what happens at run time when a database that is currently running a mixed workload and the DSS workload portion completes. In order to optimize the query for OLTP, the DBS must be adapted by setting the configuration parameter to 100, which is the ideal parameter for OLTP only workloads. In Figure 12, the system outputs the status of a known context being encountered.

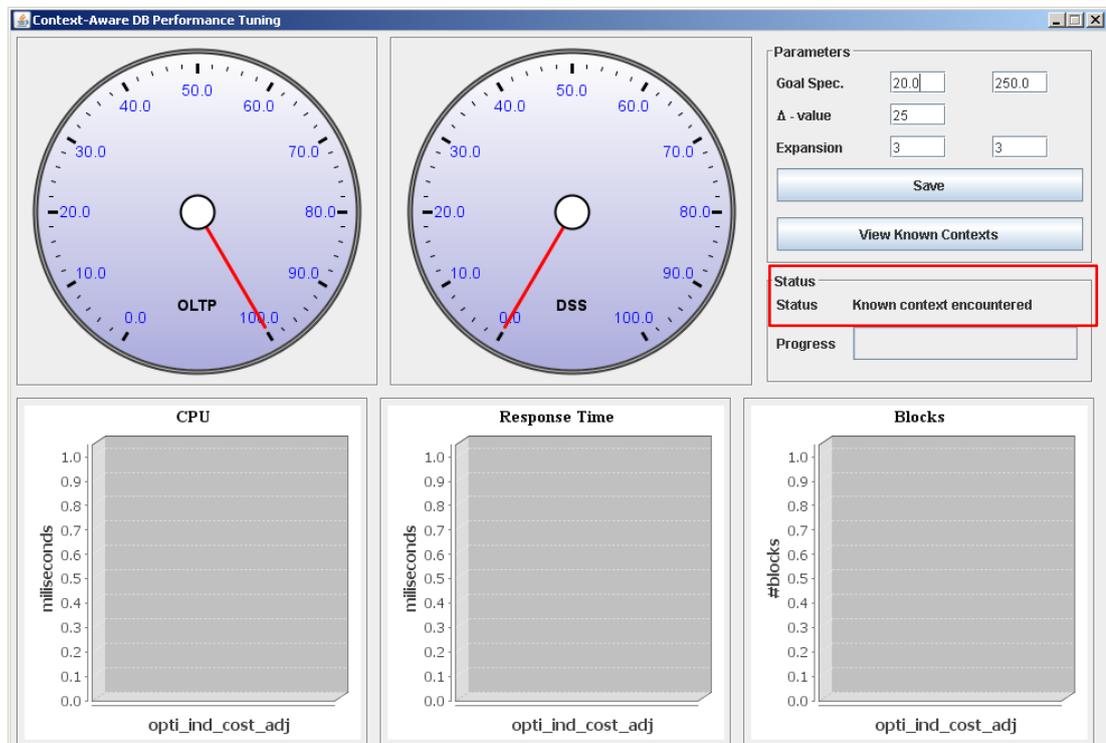


Figure 12. Known Context Encountered by Context-Aware Application

Figure 13 shows the system output status where the system has been updated with the configuration parameter value ideal for OLTP only workloads. Once the system is updated, the application goes back to context sensing mode.

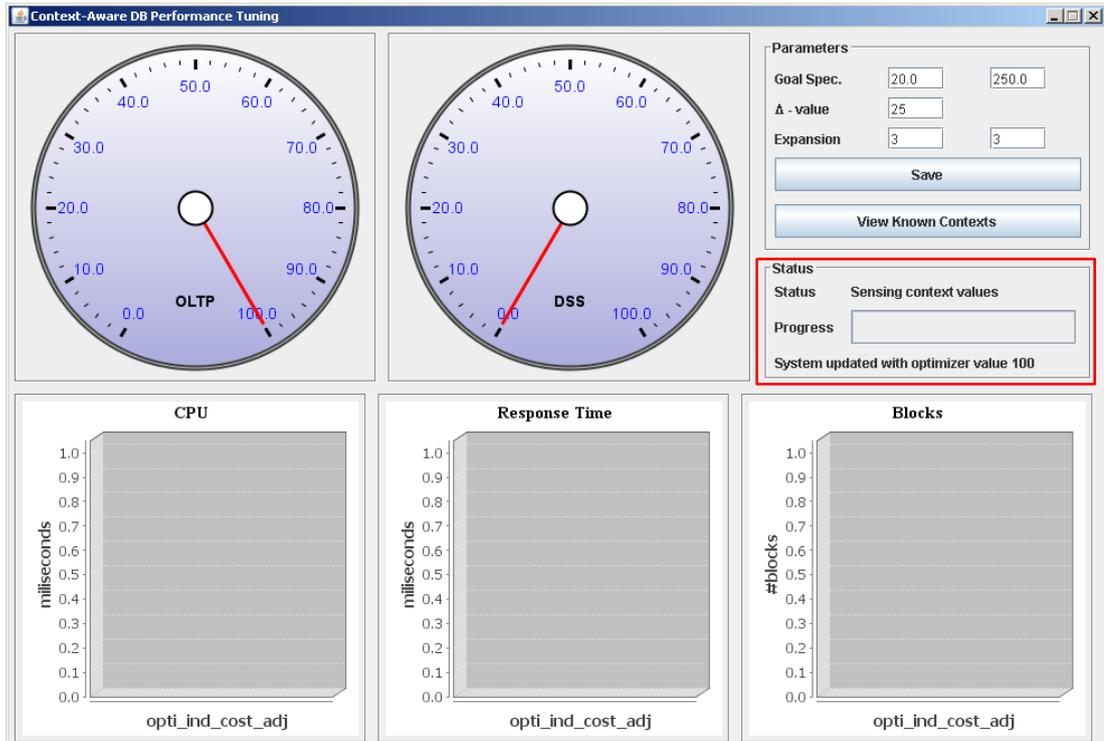


Figure 13. DBS Updated With Value from Knowledge Base

The section below describes the runtime activity when the system encounters an unknown context. In this particular case, the system was running only an OLTP workload and a DSS workload is introduced into the database. As a result the resource usage pattern changes and the context-aware application infers a context change as shown in Figure 14.

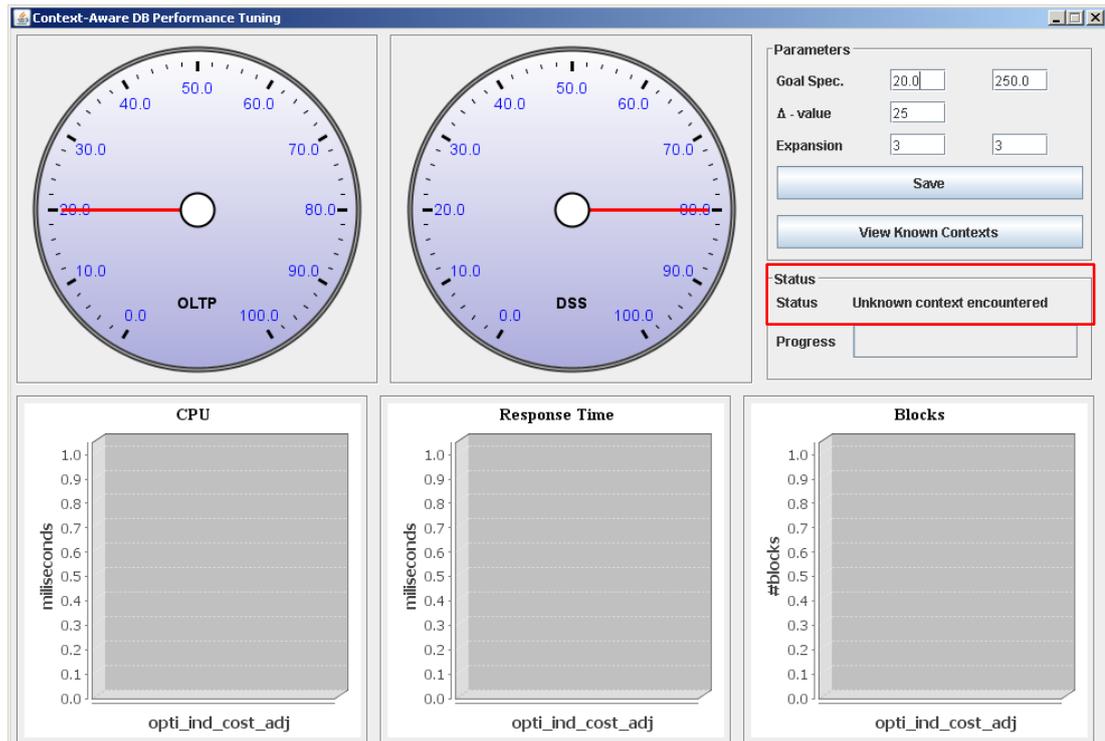


Figure 14. System Detects an Unknown Context

Once the unknown context is detected the action system carries out the concurrent multi-action evaluation. In Figure 15 the workload distribution dials show the workload distribution of 10:90 between OLTP:DSS workload types. Looking at the knowledge base (Figure 11) it could be seen that the closest know context for this unknown context is OLAP_0_100, which has a workload distribution of 0:100 between OLTP:DSS. The database configuration parameter associated with this context is 20. The action space is built around the closest known context to the perceived unknown context. The action space consists of 4 actions with configuration values of 20, 45, 70 and 90. Each graph shows the

amount of CPU, response time and number of database blocks used during the experimentation under different configuration parameters.

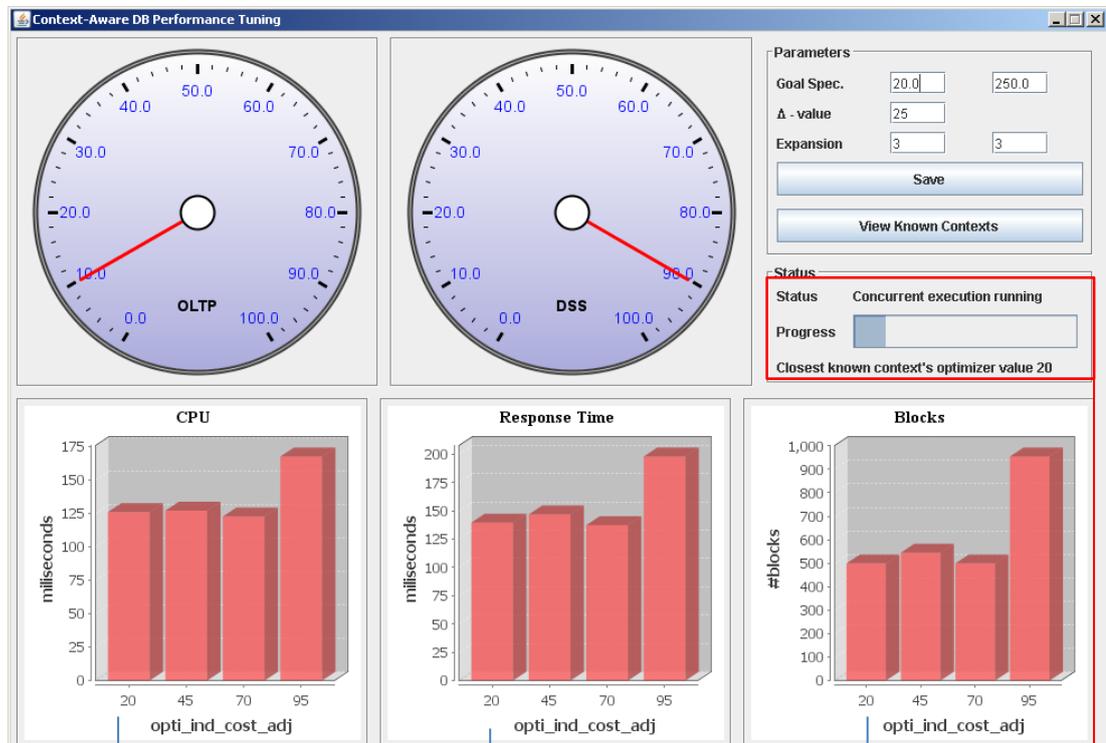


Figure 15. Concurrent Multi-Action Evaluation Being Carried Out

DB Configuration parameter uses by the closets known

DB configuration parameters used by each action in the action space. Using the formal specification mentioned in Section 3.3.2 these could be written as A(25), A(45), A(70), A(90).

At the end of the concurrent multi-action evaluation, the context-aware application updates the DBS with the configuration parameter that yields the highest benefit in terms of resource reduction as shown in Figure 16.



Figure 16. DBS Updated by the Best Adaptive Action

Finally, the context-aware application updates the knowledge base with the configuration parameter information and associated context as shown in Figure 17. In this case the unknown context had a workload distribution of 8:92 between OLTP:DSS. Concurrent multi-action evaluation had identified that DB Configuration parameter 20 to be the best setting for this context (i.e. workload distribution) The context-aware application would be able to recognize the current context in the future, which is no longer unknown and set the ideal configuration parameter for it.

Context Name	OLTP Load	OLAP Load	Optimizer Value
OLTP_100_0	100	0	100
Mix_90_10	90	10	90
Mix_8_92	8	92	20
OLAP_0_100	0	100	20

Figure 17. Knowledge base Updated with New Context Information

4.5 Experimentation

The context-aware application experimentation was conducted with two different setups. In setup one, the experimentation is conducted on the same database (on-site) that is being sensed by the context system. In the second experiment setup, the experimentation is conducted away (off-site) from the database being sensed by the context system. Apart from where the experimentation takes place, all other aspects of the experimentation are the same between the two setups.

Workloads were generated against two of Oracle database's sample schemas namely OE (order entry) and SH (sales history) [164]. The query to OE schema was used to simulate OLTP workload type. This schema contains order entry related tables, which are accessed using order numbers. The query access paths are index based. The queries to SH schema was used for simulating DSS workload type, which runs queries against the sales history for generating sales reports. Query access paths on this workload type tend to use the index and full table scans. The data volumes of the sample schemas were increased with the use of Swingbench [165] load generator, such that the OE schemas main table to have over one million rows and SH main table to have over four million rows.

4.5.1 Experiment Setup 1

Figure 18 shows the first experiment setup, where the experimentation is on-site. This setup represents a configuration where an enterprise system consists of the production database and no standby database. In such cases, in order for findings of the experiments to be realistic, the experimentation must be conducted on the production database itself.

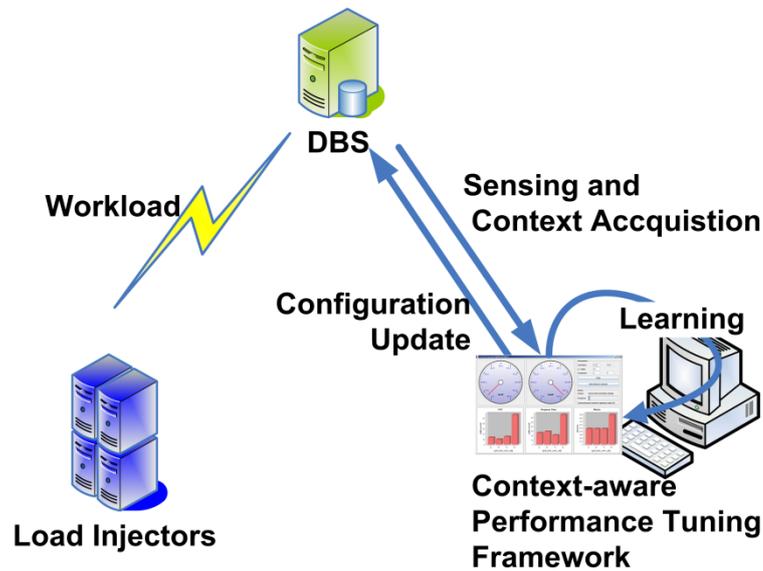


Figure 18. Experimentation Setup with an On-site Database

For this setup, the Oracle 12.1 was chosen as the production database which ran on a server with 12GB RAM, 2.0GHz Intel quad-core processor and 500GB SAS disks running on RedHat Linux 5.5. The workload was generated using a load injector server which had the specifications of 16GB RAM, 2.4GHz Intel quad-core processor, and 500GB SAS disk running RedHat Linux 6.4 All machines were connected via LAN.

Three different workloads were created for the OLTP and DSS workload types (OLTP₁, DSS₁, DSS₂) and injected to and removed from the DBS at 5-minute regular intervals. The OLTP₁ workload represents the regular online transaction processing workload the DBS is tuned for. The DSS₁ and DSS₂ represent two different decision support queries that are executed infrequently. This is representative of a practical situation such as an airline reservation system where OLTP₁ would represent a search for availability of seats which is executed the majority of times and DSS₁ and DSS₂ would represent ad-hoc reporting queries such as mid-day sales report for a particular destination or a date. Table 2 shows the

experimental workload mix on the DBS at various times corresponding to Figure 19.

Table 2. Experimental Workload Mix in the DBS

Time	Workload Mix
T ₀ , T ₆	OLTP ₁
T ₁ , T ₇	OLTP ₁ , DSS ₁
T ₂ , T ₉	OLTP ₁ , DSS ₁ , DSS ₂
T ₃ , T ₁₁	OLTP ₁ , DSS ₂
T ₄ , T ₁₃	OLTP ₁

4.5.2 Results

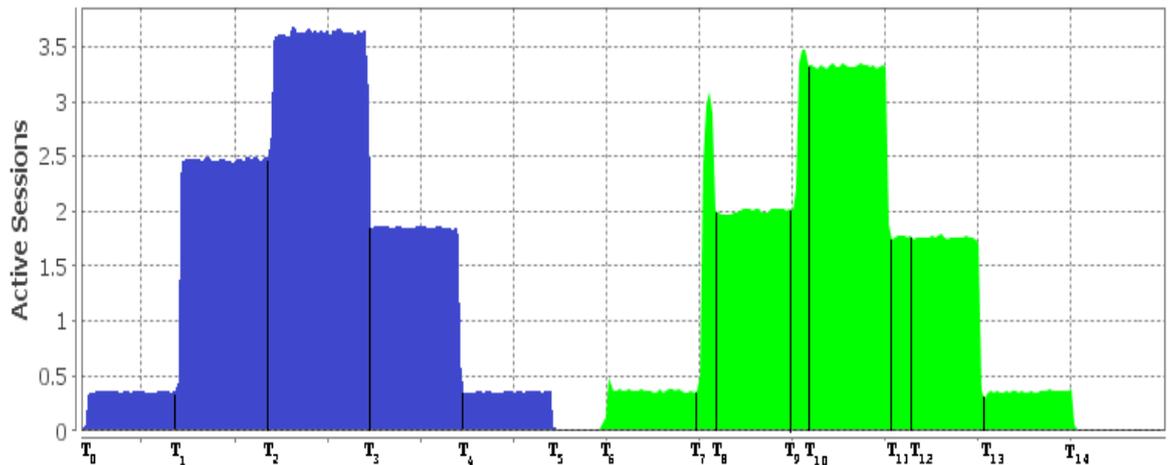


Figure 19. CPU Usage of the On-site DBS Without and With Context-Aware Adaptation

On Figure 19 the graph on the left in blue colour represents the CPU usage pattern when the DBS configuration was optimized only for OLTP workloads and the graph on right in green colour represents when the context-aware application is able to experiment and updates the configuration parameter to best fit the workload mix the DBS is serving at the time. The Oracle Enterprise Manager (OEM) plots the CPU usage in terms of active sessions [163] (Y-axis), the value of which is calculated as:

$$active\ sessions = \frac{CPU\ used\ by\ all\ non - idle\ DB\ sessions}{wall\ clock\ time}$$

This represents the ratio of CPU time used by all active sessions in a given time interval. The theoretical maximum for this value is the total number of CPU cores available on the server where the DBS is running. There were no other workloads on the test DBS except for the experimental workloads generated. Therefore, the two graphs on Figure 19 are directly comparable. The X-axis unit is minutes and time-steps denote the workload injection and experiment/adaptation points.

The context-aware system knows the configuration parameter to use when OLTP only workload is present as this was inserted to the knowledge base, based on the DBAs experience of the system. At T_6 the system is updated with the corresponding setting for OLTP workload type. When at T_7 the second workload is introduced the load characteristic changes, the context-aware system infers it has encountered an unknown context and the current configuration is not suited for the mixed workload. Concurrent experimentation and adaptation are carried out by the context-aware system between $T_7 - T_8$. During the concurrent experimentation, the CPU usage on the DBS system spikes higher than the CPU usage at T_1 when the DBS is not employing a context-aware system to adapt to changing workloads.

Once the adaptation has taken place at T_8 and the DBS is updated with the best configuration for the current workload mix the CPU usage is reduced ($T_8 - T_9$) compared to what the CPU usage would have been if there had been no adaptation. At T_9 the second DSS workload is introduced. This results in increasing the CPU usage in the DBS, however, since the system has already

adapted to a DSS workload type, the increase is less compared to the increase in the CPU usage when the DBS was non-adaptive.

The change in the CPU usage results in a context change and experiment-based adaptation takes place between $T_9 - T_{10}$. During the on-site concurrent experimentation, the CPU usage spikes since but even then the overall CPU usage is less compared to when the DBS was non-adaptive ($T_2 - T_3$). Once the DBS adapts to the new workload mix, as a result of the concurrent experimentation the CPU usage reduces ($T_{10} - T_{11}$) compared to had there been no adaptive tuning. At T_{11} DSS_1 is removed from the workload mix which results in a reduction of the CPU usage. The resulting workload mix is identified as a context change and experimentation and adaptation take place between $T_{11} - T_{12}$. Since the system has already adapted, the workload change does not result in a CPU usage spike compared to the previous two concurrent experimentations. At the end of each experiment-based adaptation, the knowledge base is updated with the configuration parameter identified as the best for the unknown context encountered. This enables the context system to recognize similar contexts in the future without any experimentation and to adapt the DBS configuration to better suit that context.

The results showed that when the DBS is able to adapt according to the workload being served, this results in less CPU usage compared to when it was non-adaptive. However, the on-site concurrent experimentation results in additional load to the DBS (time period $T_7 - T_8$). This could be overcome by either using a cycle-stealing mechanism [151] or conducting the concurrent experimentation off-site.

4.5.3 Experiment Setup 2

A second experimentation setup was devised where the concurrent experimentation is conducted off-site away from the product DBS on the standby database. Enterprise application uses standby databases for business continuity as part of the disaster recovery strategy. The primary and standby DBS are kept in synchronization by shipping redo records from the primary DBS to the standby DBS. As such any experimentation on the standby DBS will have the same load characteristic as an experimentation on primary DBS.

The chosen DBS was Oracle 12.1 for both primary DBS and standby DBS which ran on a server with 12GB RAM, 2.0GHz Intel quad-core processor and 500GB SAS disks running on RedHat Linux 5.5. The workload was generated using multi-threaded load injector scripts which ran out of server which had the specifications of 16GB RAM, 2.4GHz Intel quad-core processor, and 500GB SAS disk running RedHat Linux 6.4. All machines were connected via LAN. Figure 20 shows this off-site concurrent experimentation setup.

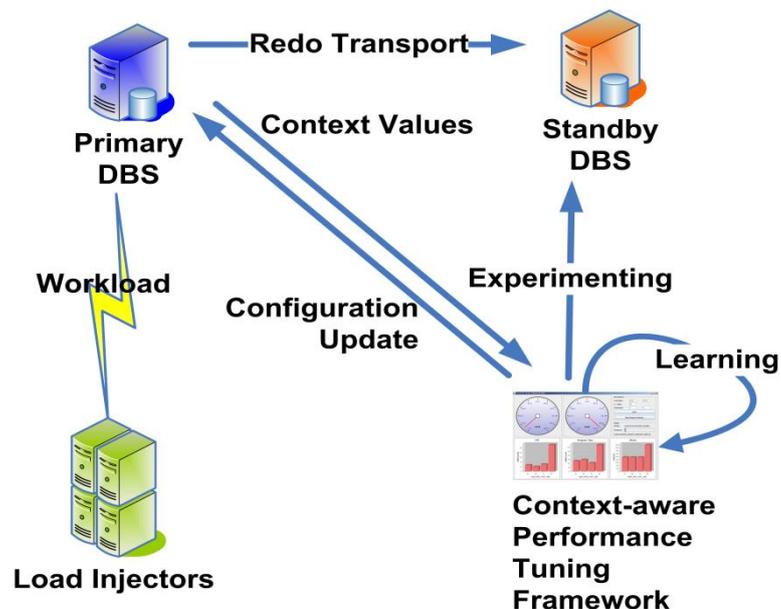


Figure 20. Experimentation Setup with an Off-site Database

Similar to the on-site concurrent experimentation setup, three different workloads were created for the OLTP and DSS workload types (OLTP₁, DSS₁, DSS₂) and injected to the primary DBS at the same rate used for the previous experiment setup. Table 3 shows the experimental workload mix on the primary DBS at various times corresponding to Figure 21.

Table 3. Experimental Workload Mix in the Off-Site DBS

Time	Workload Mix
T ₀ , T ₆	OLTP ₁
T ₁ , T ₇	OLTP ₁ , DSS ₁
T ₂ , T ₉	OLTP ₁ , DSS ₁ , DSS ₂
T ₃ , T ₁₁	OLTP ₁ , DSS ₂
T ₄ , T ₁₃	OLTP ₁

4.5.4 Results

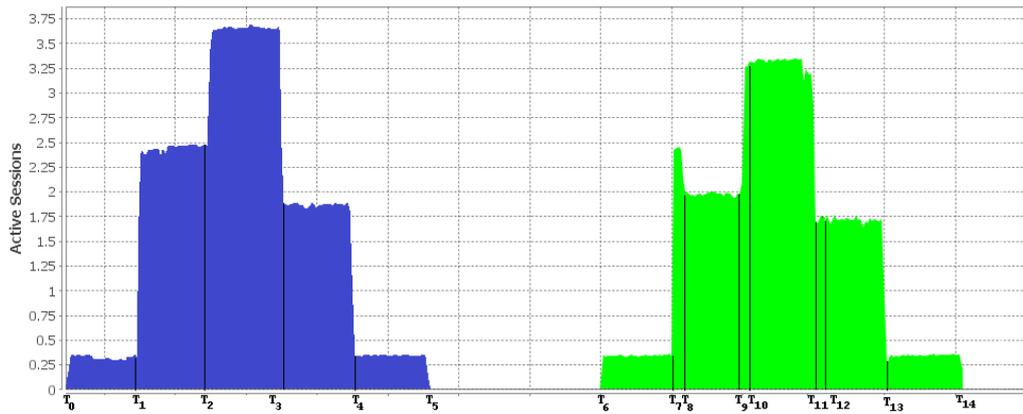


Figure 21. CPU Usage of the Off-site DBS Without and With Context-Aware Adaptation

Same as before, the graph on the left in blue represents the CPU usage pattern when the DBS configuration optimized only for OLTP workloads and the graph on right in green represents when the context-aware application is able to experiment and updates the configuration parameter to best fit the workload mix the DBS is serving at the time.

The experimentation starts off with the system containing only the OLTP₁ workload and first DSS workload (DSS₁) is injected at T₇. This results in a change

of the CPU usage distribution and is detected by the context-aware application as a context change. Inference of this context change reveals this is an unknown context which results in concurrent experimentation to be carried out on the standby DB. During this experimentation time period ($T_7 - T_8$) the CPU usage reaches the same level as in the test without context-aware adaptation ($T_1 - T_2$). However, once the experiment results have been evaluated and adaptation has taken place, the overall CPU usage is lower for the rest of the time period of the ($T_8 - T_9$) compared to without experiment-base adaptation. As the concurrent experimentation is off-site there no additional overhead introduced to the primary DBS. Using the equation used to calculate the active session count, the difference in the absolute CPU usage was found to be 150 CPU seconds. Since the workload injection rate is constant the two tests cases (without and with adaptation) had the same amount of work. Therefore the reduction in the CPU usage implies the new query plan resulting from the adaptation of the optimizer parameter to fit the current workload mix used less CPU to complete the same workload. Calculating this reduction in CPU usage showed that when database was adaptive it used 20% less CPU to execute the same workload compared to non-adaptive instance.

The knowledge base is expanded with the current context space values and the parameter setting suited for it. The second DSS workload (DSS_2) is injected at T_9 but as the DBS has already been adapted to one DSS workload the increase in the overall CPU usage is lower compared to the same workload mix ($T_2 - T_3$) without experiment-base adaptation. But the injection of this second DSS workload results in a change in the overall CPU usage. This in turn triggers a context inference that leads to unknown context being detected and concurrent experimentation to take place in the time period of ($T_9 - T_{10}$). Evaluation of the experiment result

indicated that parameter setting that's already being used results in the lowest overall CPU usage for the current context. Therefore, the adaptation did not result in lowering of the CPU usage for this time period of $(T_9 - T_{11})$ but it is still lower compared to $(T_2 - T_3)$, which had a similar workload mix. The knowledge base is expanded to include the current context and parameter setting suited for it, so the context-aware system is capable of detecting and adapting irrespective of the order of the workload occurrence. At T_{11} one of the DSS workloads (DSS_1) is removed from the DBS, which results in a change in the CPU usage distribution and context inference to take place. Similar to earlier cases, since the DBS has already been adapted to the OLTP and DSS workload mix, the overall CPU usage is lower $(T_{11} - T_{13})$ compared to same workload period without experiment-based adaptation $(T_3 - T_4)$. As the context inference reveals an unknown context, the concurrent experimentation take place in the time period $(T_{11} - T_{12})$. Similar to the situation with the previous workload injection, the adaptation based on the experimental results does not lower the overall CPU usage as the current configuration values and the values derived from the experiment results are the same. However, the knowledge base is updated to reflect the current context and configuration parameter setting so that the context-aware system is capable of identifying this particular workload mix in the future.

These observations for both experimental setups could be validated by examining the results of the concurrent experiments, which is the CPU used per single execution of a query under different parameter settings. These results are used by the action evaluation to determine the action that maximizes the goal expectations. In this case, the expected goal is to reduce the CPU used for query execution. In order to avoid workload bias, for the mixed workload types, the

result evaluation uses a weighted average to calculate average CPU used during the experimentation. For example, assume that the DBS is primarily used for OLTP workloads. Therefore it could have more OLTP query executions than DSS query executions. As such using the weighted average ensures the configuration parameter change does not negatively affect the workload type that is most active at the time of the context change.

The system started with OLTP only workload type which is a known context based on the initial facts the knowledge base is populated with, where configuration value of 100 was considered the best configuration. When the DSS workload is injected into the DBS, the CPU usage pattern gets changed and the context inference did not yield a known context. This resulted in the top resource consuming queries for each workload type being executed in the private workbench under six different parameter settings. Each parameter reflects an action (A(20), A(45), A(70), A(95), A(120), A(145)) which used that parameter to create a distinct private workbench area for experimentation. SQL executed in these private workbench areas would use the configuration parameter set by each action on that particular private workbench area. Figure 22 shows the weighted-average CPU usage per execution of a query during the concurrent multi-action evaluation under various configurations settings.

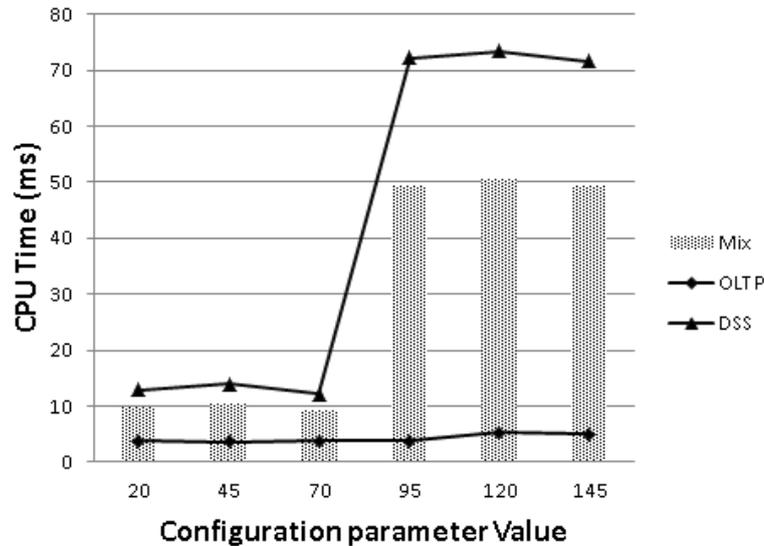


Figure 22. Average CPU Usage for Each Configuration Settings

It shows that setting the configuration parameter to 70 results in the lowest CPU usage to execute the current workloads. However, if only the OLTP workload type is considered, then the best configuration parameter setting would be 95. But in a mix workload environment, this would lead to regressed performance on the DSS workload queries. Furthermore, this result evaluation could be used by the DBA to validate the initial knowledge base setting and rectify if they are not the ideal configuration values.

4.6 Conclusion of the Case Study

The experimental results have shown that with the use of goal specification and action refinement together with the concurrent multi-action evaluation, the implemented system enables the DBS to adapt to changing workload types and resource usage patterns. As a result of this work, it is hoped to introduce a new paradigm of context-aware database performance tuning where instead of one setting fits all approach, the DBS updates its settings to best suit the current workloads and assists DBAs in performance tuning tasks.

As future work, the context-aware model could be extended to include multiple parameters for tuning. Some of these parameters involve system resource related parameter tuning such as memory parameters (buffer pool sizes, shared pool size) I/O parameters (synchronous IO, asynchronous IO, IO schedulers), network socket related parameters. This would require a multi-node grid experimentation setup for concurrent experimentation to be carried out as resource settings are not opaque between the experiments.

The lessons learned from this case study have implication beyond database performance tuning. The ability to adapt the database and run the same workload with less CPU has commercial implication as well. One such item is reduced licensing cost as system is able to handle more workload with adaptation. Therefore requirement for server scale up or scale out would be infrequent. This in turn reduces the power consumption in the data centres. If workloads could be completed with few number of servers (or CPUs) using application adaptation, that will have a major impact on power consumption. In an era where environmental concerns are front page news due to power (or energy) generation and consumption, this context-aware approach could be used to address some of those concerns.

**5. CONTEXT-AWARE
APPROACH FOR
DETERMINING THE
THRESHOLD PRICE IN
NAME-YOUR-OWN-PRICE
CHANNELS**

5.1 Background

The Name-your-own-price (NYOP) sales strategy gained wide popularity with the rise of the e-commerce. The web and related technologies overcame the technical barriers that made such a sales channel to be unfeasible in the traditional brick and mortar operators. In brief, under an NYOP strategy, the seller does not disclose the price of the goods or services being provided. The buyer would bid based on the perceived value of the goods or the service. If this bid value is higher than a threshold price that the seller has set, then and only then a transaction would occur.

However, employing an NYOP strategy for selling time-sensitive items provides additional challenges. A time-sensitive item, in this case, is an entity the value of which becomes insignificant to both the buyer and the seller after a certain point in time. Examples of such items are the common perishable goods, sports or concert tickets, airline tickets and alike, which after a certain time no longer have a commercial value. One example of a challenge when employing an NYOP for selling time-sensitive items is that the seller has to consider the actual and perceived price depreciation of the items if an NYOP bid is rejected. At the same time if there's no demand uncertainty, then the seller could afford to reject a bid with little risk and hope for a higher yield from another bid.

In essence, the seller has to be aware of the current bidding trends and external forces that influence the buyer's bid values before deciding to accept or reject an NYOP bid. The current bidding trends and the external forces are the contexts of the NYOP bidding system, as they have a direct influence on the execution of the bidding system. In order to make maximum profit with changing context, the seller must adapt the threshold price to best fit the current context as

one threshold price may not suit all eventualities. This is a problem for the seller as it is not possible to know the best threshold price for all the contexts the NYOP system may encounter.

A context-aware application implemented based on the framework proposed by this research provides a possible solution to this problem with its concurrent multi-action evaluation when an unknown context is encountered.

5.2 Name-Your-Own-Price Channel

Name-your-own-price (NYOP) is a strategy where the buyer suggests the price which he/she is willing to pay for a good or service without knowing the minimum threshold price which is acceptable. This technique has been employed for commercial activity throughout history. In fact, when in “1653 George Fox suggested that everyone should pay the same price for the same good it was considered a radical notion and by and large ignored” [166]. This practice is still being used by street vendors in various parts of the world where no trade is done without some form of haggling, in another word, naming your price.

However, the technique has emerged as another viable mainstream sales channel thanks to the internet economy (or e-commerce), which has eliminated the issues involved with manual NYOP strategy. Terwiesch *et al.* [166] list some of the issues eliminated by technology, allowing online haggling to be viable. Foremost in the absence of posted prices, the owner of the retail store had to provide detailed instructions to his clerks on how to conduct the haggling process. They had to be given advice on how to give a price so it is not too high, so customers aren't turned away. At the same time, they had to be taught how to discount in favour of a sale during low demand times. This process required

extensive training, which is no longer needed with electronic sales channels (e-channels). The second issue had to do with scalability; the capacity of the clerk dealing with customers was constrained by the lengthy haggling for every transaction. Therefore, it is possible some customers may miss out on a reduced or haggled price. In order to avoid customer dissatisfaction [167], the owner would have to hire additional clerks. Finally, the communication between owner and clerks require close supervision and monitoring to avoid principal-agent problems. The decision flow for a NYOP channel could be modeled as below where T is the acceptable threshold price.

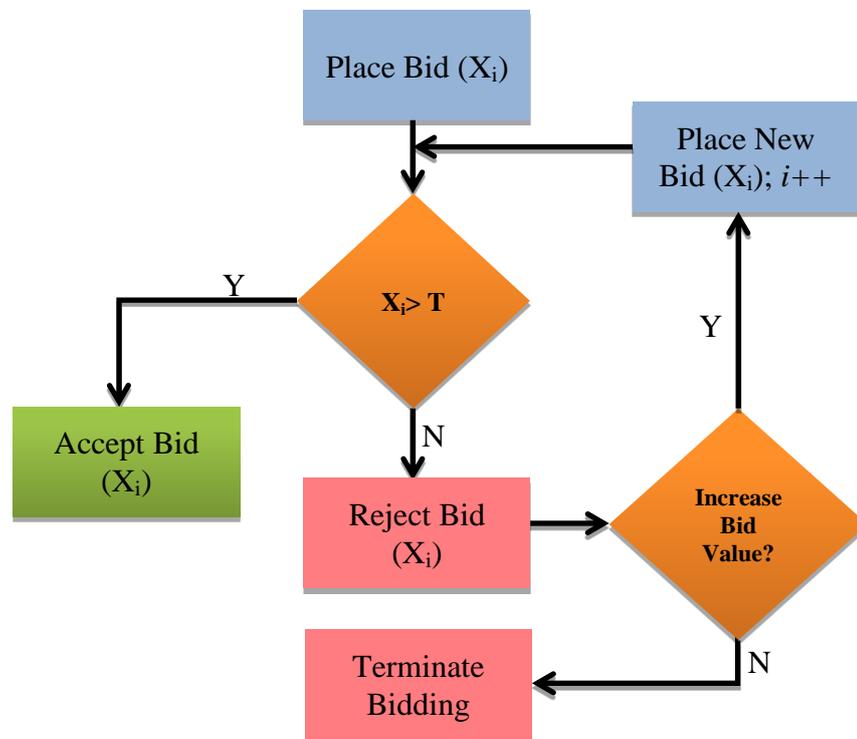


Figure 23. NYOP Decision Flow

A buyer places a bid with value X_i and if this value is higher than the acceptable threshold price T then a transaction occurs. If the bid value is lower than threshold price then the buyer is informed of this fact and either allowed to make another bid or the bidding process terminates. The buyer incurs “frictional cost” [168, 169, 170] when a bid is resubmitted after a rejection. Frictional cost is

the additional cost (time, effort and emotional strain) a buyer has to bear during a subsequent bid after having being rejected previously.

In a situation where NYOP is used for non time-sensitive items, the buyer's focus is to have a high consumer surplus, because the seller isn't incentivized to adjust the price of the item. Consumer surplus is the difference between buyer's willingness to pay (the maximum price he/she will pay) and the actual price paid with low frictional cost [171]. However, when NYOP is used for time-sensitive items the seller will also be concerned about the producer surplus, which is the difference between the lowest price the seller is willing to sell at and the actual selling price [172, 173].

There are similarities and differences between NYOP and traditional auction mechanisms. Vickrey [207] identified four basic auction mechanisms. They are the English, Dutch, first-price sealed-bid and second-price sealed-bid auctions. English auction also called ascending auction [208] is where bidders would start at a lower bid value and go on increasing the bid value until one bidder is successful in completing the auction transaction. Dutch auctions [209] on the other hand are auctions in which the offering price of the item being sold is set high and then lowered until a bidder makes a bid and wins it. The two types of auction mechanism are also called real-time auctions to distinguish them from the sealed-bid strategies explained next.

In first-price sealed-bid auctions a bidder would submit a sealed bid before a particular deadline. Once the deadline has passed all the bids are evaluated and highest bidder is chosen as the winner. In the second-price sealed-bid auction, the winner doesn't pay the amount that she submitted but that of the second highest bid.

The NYOP is similar to sealed-bid auction as each NYOP bid is invisible to other bidders. Bidders in a NYOP mechanism are driven by consumer surplus rather than any other influences. To encourage higher bid values sellers using NYOP channels may not allow subsequent bids.

However response to NYOP could be instantaneous making it similar to real-time auction mechanisms such as English or Dutch auctions. Unlike the ascending or descending auctions the NYOP allow a user to change their bid values as they please. In practice a NYOP seller may force a user to only submit a second bid if it's higher than previous one. But users are known to circumvent this restriction on online NYOP channels by using multiple credit/debit cards.

But the fundamental difference between traditional auction mechanisms and NYOP is that in auctions, bidders would know the starting price of an item whereas in NYOP bidders have no knowledge of the price of the item at any stage. Thus in auctions bidders can decide how much more or less to bid based on the reserved price or competing bids. In NYOP a bidder would be guided only by his consumer surplus.

5.2.1 Name-Your-Own-Price Strategies

Companies like Priceline (www.priceline.com - sells airline tickets and hotel rooms) employ a random element, where it randomly selects two hotels from a set of hotels, rather than compare the bid with the lowest rate available. This ensures that the information gained through previous successful bids is negated. Therefore, each bid's willingness to pay is not influenced by other successful or unsuccessful bids [174]. In this NYOP strategy even if two buyers bid the same value, it's possible that only one bid is successful and other one is rejected. Hinz *et al.* [175] compare the strategy of using fixed thresholds versus adaptive

thresholds, whereby the seller is learning the buyer's willingness to pay with each bid that is being rejected.

Allowing or restricting repeat bids is another mechanism used to influence the buyer's bids. Fay [176] examines whether it is profitable to restrict buyers to a single bid similar to the strategy employed by Priceline which restrict one bid per buyer for 24 hour period. However, on time-sensitive items repeat bids would be negative for the seller surplus as the perceived value of the item diminishes with time, leading to the buyer's assumption that the threshold price T at time t_1 is less than the threshold price at time t_2 when $t_2 > t_1$. Therefore, the seller must infer the potential future gain or loss when rejecting a bid with the hope that the buyer will submit another bid.

This restriction on when a subsequent bid could be placed is designed to play on the buyer's impatience [177]. Impatient bidders would like to conclude the transaction at a lower fractional cost. An example of this strategy could be the Dutch auctions where bidders prefer to purchase an item sooner at a higher price when faced with a positive cost of bidding at a later time. As a result, on average, Dutch auctions have higher revenue than sealed-bid auctions [178]. Unlike in a Dutch auction, in a time sensitive NYOP scenario it is the seller's surplus that is affected as the ultimate owner of the item.

Customer impatience could also be used to segment buyers as high and low spenders. For example, business travellers would have a high willingness to pay [179] to secure their bids due to the nature of the travel, which is time sensitive. Corporate budgets may also allow them to bid at higher values than individuals, who have to pay out of their own pocket.

Opaqueness is another method employed to influence the buyer's behavior. Opaqueness refers to the fact that buyers cannot have complete information about the product or prices. In this regard, Priceline's deals are opaque as buyers cannot know the exact hotel when they bid for hotel rooms in a specified area, nor do they know whether the bidding prices will be accepted [170]. In [174] it is stated that this opacity makes it difficult for price rivalry, as buyers cannot be certain about the outcome of their bid or if a substitute item (e.g. same flight from a different website) from a rival would succeed for the same bid value.

Information leakage to the buyer through interface design is also used to influence the buyers' bids [180]. Due to the restricted number of bids (in this case one), the buyer is given information on the probability of succeeding if certain values are used. This method may not be suitable when using NYOP for time-sensitive items due to the depreciative nature of the items.

5.2.2 Pricing Strategies for Time Sensitive Items

Time-sensitive items are items the price of which decreases over time due to the perishable nature of the item. Well-known perishable items are food items such as milk, meat, etc. But this is not limited to consumables. There are items that become perishable due to seasonality such as seasonal clothing (winter coats, summer beach-ware, etc.), seasonal decorative (Christmas trees), or due to being replaced by another, such as magazines, or an empty seat on a departing airplane or each day a hotel room is unoccupied (a lost opportunity to earn revenue).

Berk *et al.* [181] developed a dynamic programming model which determines the selling prices dynamically by considering the dynamic pricing of perishable assets and explicitly incorporates the menu cost into the model with the objective of maximizing the discounted expected profit for an initial inventory. The menu

cost is the costs associated with price changes with reference to the particular physical costs. For example, printing new menus at a restaurant or changing labels at a store every time the prices are changed could be considered as menu costs. For an NYOP the menu cost would exist implicitly as the internal threshold price and any subsequent change to threshold price is opaque to the bidder.

Two of the pricing strategies that are widely used on perishables are adjusting the order quantity to influence price [182] or adjusting the price based on demand or both price and order quantity [183] and [184].

Chun in [183] employs the later strategy of adjusting supply. This model is primarily concerned with the case where the retailer wants to maintain the same selling price on a perishable product throughout the entire sales period by adjusting the ordering policy rather than the price. This strategy is not directly applicable to NYOP as there is no posted price, hence the buyer has no restriction to adjust the “posted price” in the form of the threshold price.

Becher [184] tries to inject intelligence into the price and order adjustment process through the use of fuzzy controllers and hope to identify the revenue potential of a rule-based implementation. However, when a buyer identifies a new factor that needs to be incorporated into the decision structure, it would require introducing new fuzzy rules which could be time-consuming.

The strategy in [182] could be broken down into two: control price based on buyer discrimination and quantity of items left on the shelf. The buyer discrimination is implicit in clearance sales as explained by [185]. The buyers with high valuation make their purchases early on during the sale (with wider choice), while buyers with low valuation purchase during discounted time period (sales time). Though the time sensitive items in NYOP channel could be

considered a form of a clearance sale, the opaqueness of the price and the outcome of the bid bring uncertainty which is not present in a clearance sale.

5.3 Problem Identification

As it doesn't hold its own inventory, the revenue loss to Priceline from foregoing the transactions that could have been successful is very limited. The loss of revenue occurs to the hoteliers who decide to release some of their perishable inventory through the Priceline's NYOP channel. Priceline makes no assumption about the demand rate or the occupancy rate of the hotel.

On the other hand, if it is the hotelier that makes the decision, as the owner of the perishable inventory, the decision to forego a successful bid would depend on factors other than the value of the bid. When there is demand certainty, the hotelier could forego a successful bid in the hope of higher yield with other successful bids. But when there is demand uncertainty, he must consider the context of his transactional environment to determine if the bid should be accepted or not.

In these situations, the hotelier's own factors would be considered, such as the current occupancy rate, expected occupancy rate which could be influenced by an event that is scheduled to take part in the vicinity of the hotel such as conferences, festivals, sporting events, weather and other seasonal information. It would be beneficial for the hotelier to have the rooms occupied at "some rate" (breakeven being the minimum value) when there's demand uncertainty and the rate of occupancy is low, than to have unoccupied rooms incurring operational costs. According to [180] "the retailer wants to obtain as much revenue as possible. In the case of perishable or time-dated items (such as Christmas trees or

newspapers), the retailer would be willing to accept even lower bids as long as those willing to pay more actually do pay more”.

The hotelier is unlikely to know the best threshold value to set for all the combinations of factors (i.e. contexts). Therefore, the hotelier’s NYOP system needs to employ a context-aware system that adapts the bid value based on the current context. Currently, hoteliers manually adapt their prices for known contexts. How hoteliers adapt the price for known contexts could be demonstrated in the following real world example. The example is based on the room price variation of two hotels based in the vicinity of an exhibition centre when three different types of events are held at the venue.

The exhibition centre in question is Excel London² which hosts many events such as conferences, trade and private shows throughout the year. Out of many hotels in the vicinity, two hotels were chosen based on the closeness to the venue of the events. One of the hotels chosen is Ibis Style Hotel³ and the other is DoubleTree by Hilton⁴. Figure 24 shows the locality of these hotels relative to the event venue.

²<http://excel.london/>

³<http://www.ibis.com/gb/hotel-8712-ibis-styles-london-excel/index.shtml>

⁴<http://doubletree3.hilton.com/en/hotels/united-kingdom/doubletree-by-hilton-hotel-london-docklands-riverside-LONNDDI/index.html>

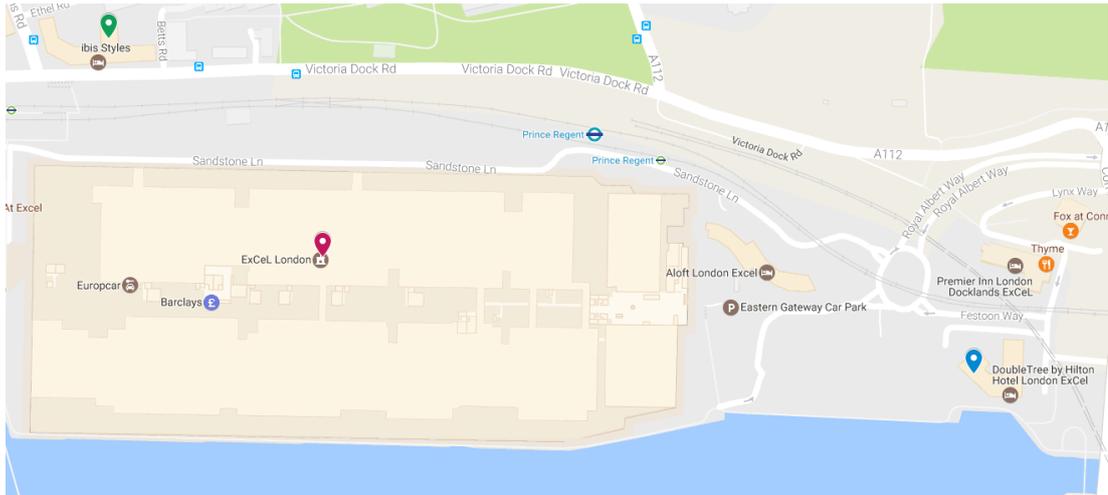


Figure 24. Locality of Hotels Relative to Event Venue

Three distinctly different events were chosen to compare how the room price varies during the days of the events. One event chosen was ComiCon⁵ which is held twice a year in May and in October. For comparison, October date range was chosen as it is closer to the dates of other events considered here. The other two events chosen were World Travel Market (WTM)⁶ and Defence & Security Equipment International (DSEI)⁷, both held once a year in November and September respectively. WTM is a trade show which brings together travel and tour operators from around the world. DSEI is an arms expo which showcases the latest in defence technologies.

Considering a typical attendee at each of these events, their “buyer surplus” (willingness to pay) for a hotel could be arranged in increasing order as ComiCon < WTM < DSEI. ComiCon is mainly attended by teenagers and young adults with limited buyer surplus, while the other two events are mainly attended by corporate workers and government employees with higher buyer surplus.

⁵<http://www.mcmcomiccon.com/london/>

⁶<http://www.wtm.com>

⁷<http://www.dsei.co.uk/>

The price of a single room with one adult was obtained through both hotels' public websites⁸, for the duration of the events. The price for two days before and one day after the event was obtained as well. This allows detecting the change of price before, during and after the event. Since each event happens at different dates the x-axis of the graphs had to be changed to reflect all events. Figure 25 and Figure 26 show the price variation during each event by each hotel. On these graphs ComiCon is held from days 3-5, WTM from days 3-5 and DSEI from days 3-6.

Looking at the graphs it is clear that price adaptation is happening based on the hotel guests' buying power. The price increases are not by the same degree for all of the events but based on the expected buyer surplus. The hoteliers know the context of the event and the type of attendees and adapt the price to gain high revenue.

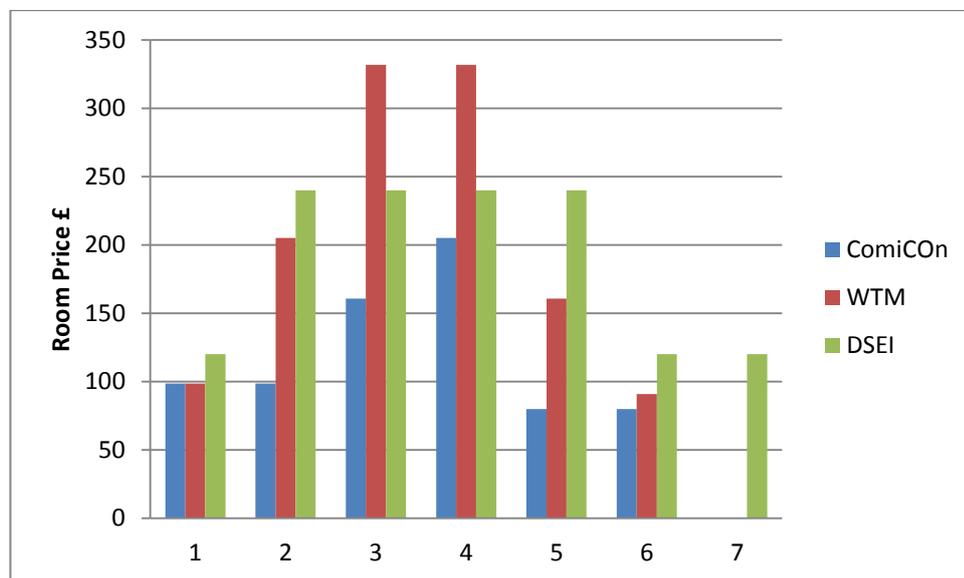


Figure 25. Price Variation of Ibis Hotel's rooms

⁸Price valid as of 11th October 2016

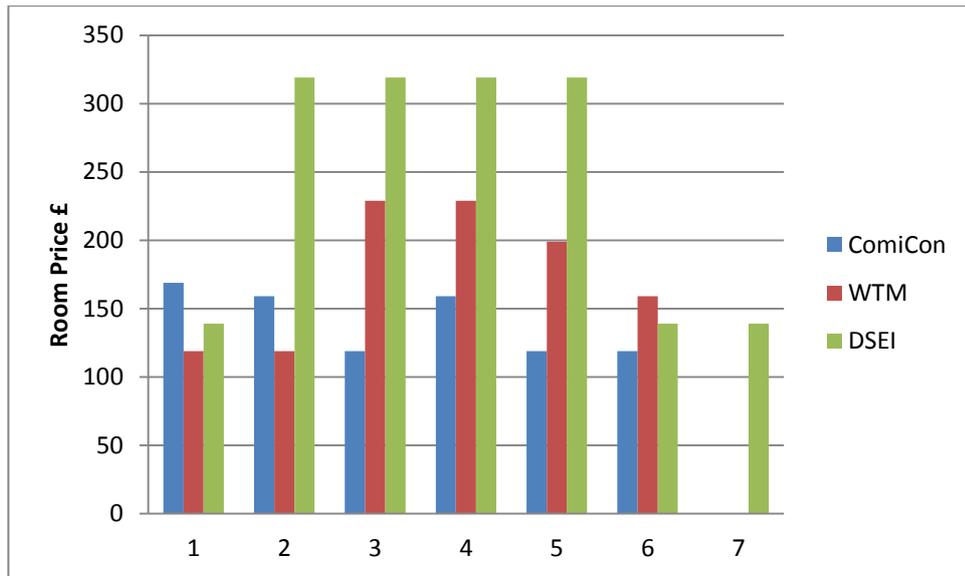


Figure 26. Price Variation of DoubleTree by Hilton Hotel's Rooms

The problem the hoteliers face is how to set the price in an unknown context, so that it results in high revenue but with low customer dissatisfaction. The unknown context could be an event that's held in the vicinity for the first time and historical information on buyers' surplus does not exist. On the other hand, the unknown context could arise due to seemingly unrelated events. An example of such unknown context is the change in demand for coach travel during the Icelandic volcanic eruption in 2010 [186]. Due to the grounding of flights during this period other forms of transport had an increased demand. Figure 27 shows the revenue for 2009 and 2010 for a coach company⁹ which shows a higher than normal revenue during the time of the disruption.

⁹Graph courtesy of CodeGen Ltd.

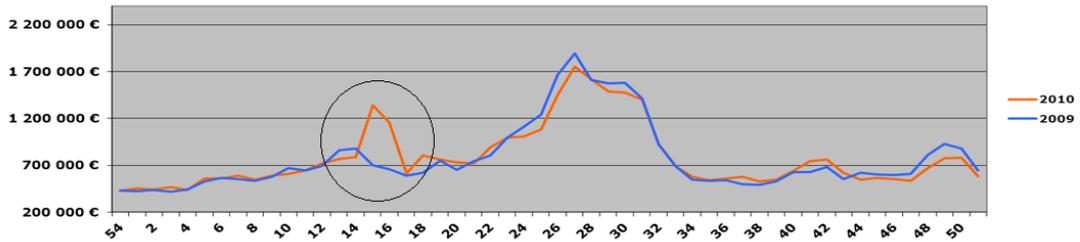


Figure 27. Revenue for a Coach Travel Company between 2009 - 2010

On the other hand, increasing the price due to an increase in demand could result in customer dissatisfaction [187]. Such was the case when Uber decided to increase the fares during London underground rail strikes [188, 189].

In any NYOP channel adaptation of the threshold, the price must be transparent to the end user and applicable uniformly to avoid unsavoury effects on customers. For example, Amazon used price discrimination based on customer type (new vs existing), which led to a situation “when a buyer deleted the cookies on his computer that identified him as a regular Amazon customer, the price of a DVD offered to him for sale dropped from \$26.24 to \$22.74” [190]. Due to the customer outrage Amazon had to issue a public apology and refund all customers who had paid higher prices.

The research offers a solution to this problem based concurrent multi-action evaluation technique to adapt the threshold price. The context-aware approach gauges the buyers’ surplus dynamically when an unknown context is encountered and adjusts the threshold price to yield high revenue. This experiment-based adaptation technique differs from NYOP techniques used in the travel industry [191].

5.4 Formal Modelling

The proposed framework was adapted to implement a context-aware NYOP channel system as shown in Figure 28.

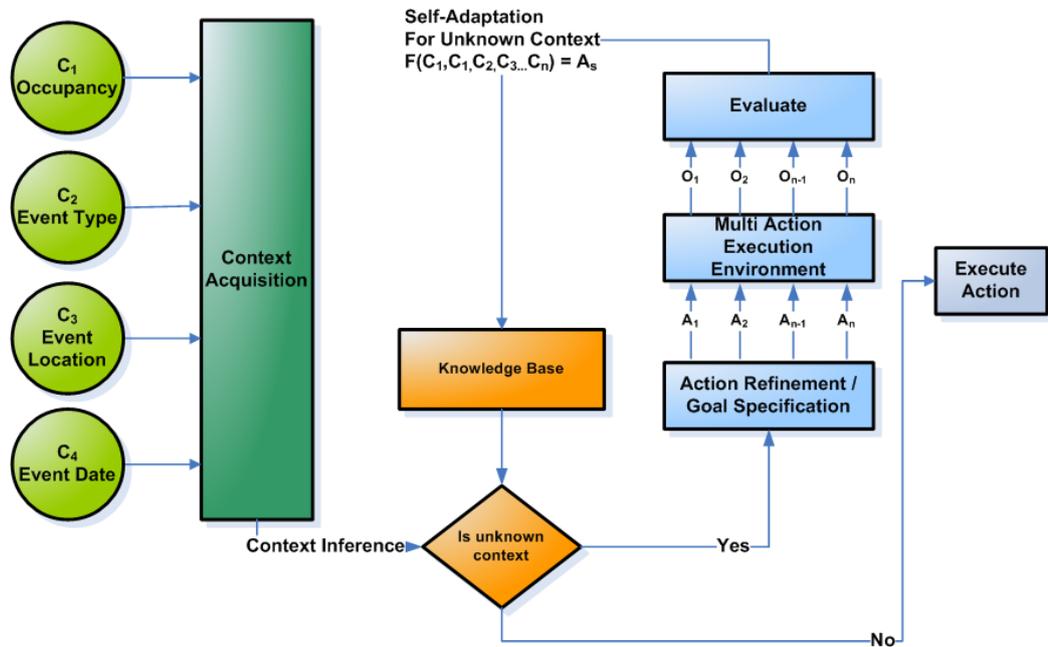


Figure 28. Flow Diagram of the Concurrent Multi-Action Evaluation System

The multi-context space was modeled as consisting of four soft contexts. The first context is the current occupancy of the hotel, which acquired through hotel's internal reservation database. The other contexts are related to an event planned near the vicinity of the hotel. These include event type, event location, and date of the event. Though they are related to the single event, each influences the bid values and the hotelier's NYOP channel in its own way. Treating the event related information as the context is in line with the definition in [29] adopted by the research. The event type could be a corporate event, a trade show or an entertainment-related event. As shown in the problem identification section each of these event types influences the bid value by way of buyer surplus. In essence, the buyer type is inferred based upon the event type. If the event venue is not

near the hotel then this would have less influence on the value of the bid compared to an event at a venue near the hotel. Therefore, the event location is considered a context. The final context is the temporal information of the event, as the demand for hotel rooms likely to increase during the days of the event. There are many avenues for sensing these event-related contexts, such as consuming Web Services APIs, Social media feeds (Twitter, Facebook) or traditional web pages.

This demonstrates the modelling of context system to acquire context from heterogeneous sources. The context acquisition and defuzzification layer are responsible for formalizing how each type of context fits into the model. In the NYOP use case, the occupancy would be a numerical value; the event type is a classification (of string type). The events must be interpreted and quantified, so that the distance between contexts could be measured. When an unknown context is encountered this inter-context distance is used to identify the closest known context.

The knowledge base was modeled using a relational database system (RDBMS), with each knowledge fact represented as a tuple in the RDBMS. This approach was chosen to demonstrate the versatility of the proposed generic framework when it comes to choosing technologies for implementations. A knowledge base implementation using OWL was demonstrated in the previous use case (Chapter 4). When a threshold value is identified for unknown context, this fact is stored in the knowledge base. So the context-aware system recognizes more and more contexts and is able to self-adapt.

The action system is modelled such that each action evaluated a sample set of bid values under various threshold prices. The evaluation criterion was set to the

threshold price with the highest number of successful bids. It is possible that some bids would be successful in more than one threshold. In such cases, the bid would be considered successful only in the highest threshold it exceeds.

Taking the NYOP threshold price as the configuration parameter, the formal modelling of the proposed context-aware framework was carried out as below.

The goal specification is a sub-range of the entire bid value range. For example, if the universe of prices for a hotel room is considered, it could vary between \$0 (100% discounted) and thousands of dollars (based on luxury). But for a particular hotelier, such a large value range is irrelevant. His interest lies in a small range of threshold values, so that accepted bids do not result in a loss ($< G_{lo}$) or the threshold price is too high ($> G_{hi}$) resulting in lower conversions (uncompetitive) and unsold rooms. The lower limit of the goal specification sets the minimal price at which the hotelier is willing to set the threshold price when there's high demand uncertainty. The upper limit sets the highest margin that will enable high yield and still be competitive in terms of other hotels in the area. With these constraints in mind G_{lo} and G_{hi} are modelled as elements of the seller's interested price range.

$$(G_{lo}, G_{hi}) \in \{ \text{seller's interested price range} \}$$

In the NYOP context-aware application, the action system adapts the threshold price of the hotel's bid evaluation system. The actions devised for concurrent experimentation are defined as a function of the threshold price. Similar to an earlier database performance use case, the action of the context closest to the unknown context is denoted as A_k and set as the initial action.

$$\text{Initial action} = A_k(\text{threshold_price}_k)$$

Besides the goal specification, the only other user inputs are the values for action refinement (p, q, Δ). In this case, the lower bound expansion value range p (in the direction of G_{lo}) represents the minimum threshold price the hotelier is prepared to set. Similarly, the upper bound expansion value range q (in the direction of G_{hi}) represents the maximum threshold the hotelier is willing to set without being priced out of the competition. Finally, the difference between two neighbouring threshold prices is denoted by Δ .

Having defined these parameters, the total number of actions (or the experiments) which need to be executed could be defined as a union of three action sets.

$$\text{Action space} = \left\{ \begin{array}{l} A_k(\text{threshold_price}_k) \cup \\ A_p(\text{threshold_price}_p) \cup \\ A_q(\text{threshold_price}_q) \\ | \quad p = \{1 .. n\}, n > 0, q = \{1 .. m\}, m > 0, \\ \text{threshold_price}_k - p\Delta \geq G_{lo}, \\ \text{threshold_price}_k + q\Delta \leq G_{hi}, \\ \Delta > 0 \end{array} \right\}$$

Once the action space is created, it is passed on to the concurrent action execution module. In this case, the action consists of the hotelier's context-aware system evaluating a sample set of live bids with different threshold values to see which threshold value would give the highest yield.

The action evaluation consists of the evaluation criteria used to determine the outcome of which action is the most beneficial. The hotelier's evaluation criterion is a maximizing function which is to select the action that gives the highest yield.

$$\begin{aligned}
threshold_price_{best} = \{ \\
& \forall threshold_price_i \in \{action\ space\ threshold\ prices\} \\
& \exists A_i (threshold_price_i): \text{Maximum} (Benefit(A_i)) \\
& \}
\end{aligned}$$

The best threshold price identified through the concurrent action evaluation is then associated with the perceived context. This context-action information is added to the knowledge base, allowing the context-aware application to carry out adaptation when the same context is encountered in the future.

5.5 Implementation

The context-aware application was implemented as a standalone Java (JDK 1.6) application running on a quad-core 2.5 GHz server with 8 GB RAM, and RedHat Linux 5. 1.

5.5.1 Context System

The context system had to acquire the four contexts mentioned in Figure 28. The occupancy was sensed through querying the reservation system of the database and the context of the event was sensed through a Java Web Service (JAX-WS) API. In a production environment, the context-aware application would be working in connection with the hoteliers' reservation system to acquire context (occupancy) and update the threshold price for bid evaluation. In lieu of a reservation system, mock up services and database tables were used as sources for context acquisition detecting context changes. The constructor and variables of the context class are shown in Listing 13.

```

public class Context {

    private double occupancy;
    private double eventType;
    private double eventLocation;
    private double eventDate;

```

```

private int C1_PRIORITY=1;
private int C2_PRIORITY=2;
private int C3_PRIORITY=3;
private int C4_PRIORITY=4;

private double thresholdValue;

public Context(double c1, double c2, double c3,double c4,
double thresholdValue){

    this.occupancy = c1;
    this.eventType = c2;
    this.eventLocation = c3;
    this.eventDate = c4;
    this.thresholdValue = thresholdValue;
}

```

Listing 13. Context Class for NYOP Channel Case Study

5.5.2 Action System

The action class was created as a runnable Java thread. Each action has a unique threshold value against which it evaluates the bid values. The bid values originally submitted to the hotelier's NYOP channel are cloned and fed into a common basket from which the action class thread obtains the bid values for evaluation. The Listing 14 shows the run method of the action class.

```

public class Action extends Thread {

    ...

    @Override
    public void run() {

        synchronized (getBasket()) {
            try {

                if (getBasket().size() > 0) {

                    while (getBasket().size() > 0) {

                        double bidValue = getBasket().remove(0);
                        checked++;

                        if (bidValue < getThresholdValue()) {
                            rejected++;
                        } else {
                            sucess++;
                            amount = new
                                BigDecimal(getAmount()).add(new
                                    BigDecimal(bidValue)).setScale(2,
                                        RoundingMode.UP);
                        }
                    }
                }
            }
        }
    }
}

```

```

        }
    }
} catch (Exception ex) {
    ex.printStackTrace();
}
}
}

```

Listing 14. Bid Value Evaluation in Action Class

The action refinement and action space creation, in this case, is similar to that of Section 4.4.3. Starting off with the closest known context’s action, the action space is expanded within the constraints of the action refinement values.

The action resulting in the highest revenue is evaluated to be the most beneficial and its threshold price is used to adapt the NYOP channel.

5.5.3 Inference System

A relational database management system (RDBMS) was used to model the knowledge base for this implementation of the proposed context-aware framework. This was mainly to demonstrate the technology independent feature of the framework. The use of RDBMS posed several challenges compared to the use of OWL in the previous use case. The main challenge was being able to remove existing contexts or bring in new contexts when they become relevant for the application. Schema-less NoSQL databases provided one possible solution. However, this was rejected as the inference operation in NoSQL database would be computationally intensive. The final solution was the adoption of the star schema from the data warehouse domain to represent the knowledge facts. A fact table in the star schema is used to store the threshold price and measure tables are used to store each of the contexts. When a new context is introduced to the system, it is added to the knowledge base as a new measure. This doesn’t affect

any of the existing context facts and the same is true for removing a measure table (context) as well.

5.5.4 Runtime Execution of the Context-Aware Application

With the help of Figure 29, the run time execution of the context-aware application could be described as follows.

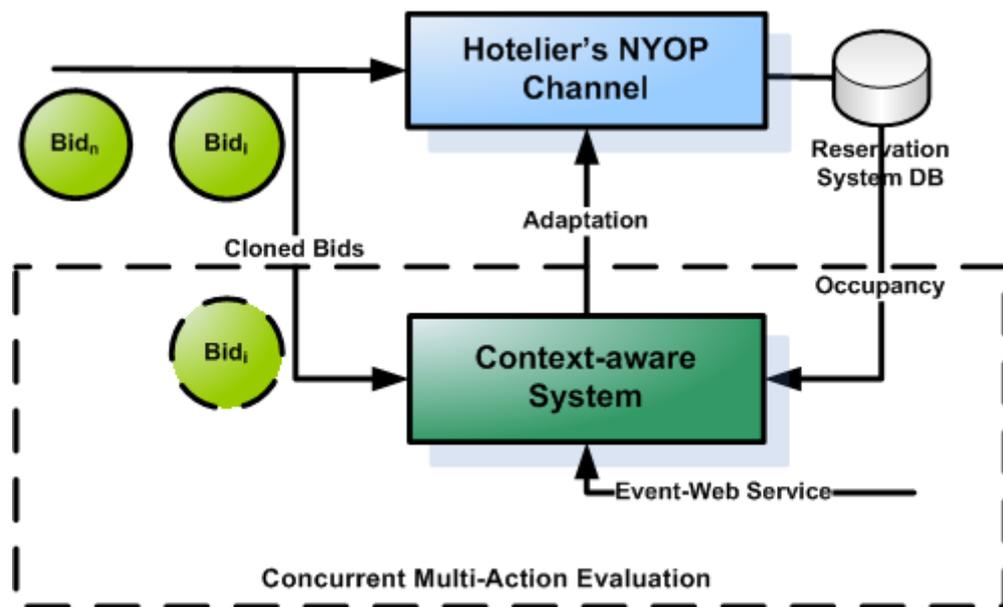


Figure 29. Run Time Execution of Context-Aware Application in an NYOP Channel

The context-aware application detects a context change through Event-WS (Web Service) when an event is scheduled at a nearby venue (event-WS contain location information). If the perceived context is unknown, this triggers concurrent multi-action evaluation to adapt the hotelier's NYOP system. Under these conditions, the action system acquires clones of the bid values submitted to the hotelier's NYOP channel in real time. These bid values are then evaluated against multiple threshold prices during concurrent multi-action evaluation. Once the concurrent multi-action evaluation has completed, the context-aware system

updates the hotelier's NYOP channel with the threshold price that would yield the highest revenue for the perceived context.

5.6 Experiment Setup

The experimentation was conducted without the use of any NYOP strategies mentioned in Section 5.2.2, such as allowing multiple bids, time restriction on subsequent bids etc. Instead, each value is considered as an individual bid and not as a subsequent bid part of a bidding transaction.

The knowledge base was populated with facts of known context and threshold values to be used for bid evaluation when the system is under each known context.

The experimentation consisted of two tests and a control test to go along with each of the two tests. One of the tests simulated an unknown context in which the mean value of the bids is lower than the threshold value of the closest known context. This case was referred to as the pessimistic case. This represents a situation the buyer surplus is lower than what the hotelier expected. The hotelier has to adapt the threshold price by lowering it in order to capture more bids in this context. An un-adapted threshold price would mean the hotelier loses out under the current context.

The second test simulated an unknown context under which the mean bid values are considerably higher than the threshold value of the closest known context. This case was referred to as the optimistic case where the buyer surplus is higher compared to what the hotelier anticipated. Under this context, the hotelier has to increase the threshold value to achieve higher revenue. This prevents some bids from succeeding, which encourages higher bidding values. Though no assumptions were made about the bidding strategies, this test case

was included in the experimentation for the completeness of the evaluation, by showing that the framework works for both the optimistic and the pessimistic cases.

A sample set of bid values were generated using the normal distribution class of Apache Common Math library¹⁰. The mean value of the sample set was adjusted to make the generated bid values fit either the pessimistic case or the optimistic case. The Δ value from the action refinement was chosen as the value for standard variation when the sample bid value set was created. This ensured a wide range of bid values that would fall into different threshold price ranges. Though the normal distribution was used to generate input bid values, it makes no difference to the outcome of the test case even if the bid values are of a different distribution. This is because the best course of action is chosen after evaluating all the bid values with all the actions in the action space.

Control tests were devised to evaluate the effectiveness of concurrent multi-action evaluation against an existing self-adaptive context-aware model. For this purpose, a context-aware system with learning based self-adapting technique was used. Such a system would iterate over a set of threshold prices using a sample set of bids and finally evaluates the outcomes to find the best threshold price. For control tests case, each action (evaluation of bids against threshold price) was given a sample of bids. The sample size was calculated as:

$$sample\ size = \frac{total\ number\ of\ bids\ into\ the\ system}{action\ space\ size}$$

This ensured the total number of bids used for evaluation is the same for all the tests and each bid is only evaluated by one action and not repeated.

¹⁰<http://commons.apache.org/proper/commons-math/>

The action evaluation is done based on the number of successful bids for each threshold value. Although not part of the evaluation criteria, the monetary value gained or lost based on each test is also compared.

For the test case, the goal specification was set as $(G_{lo}, G_{hi}) = (190, 300)$. The action refinement parameters (p, q) were set as equal numbers of pessimistic and optimistic expansion - $(p, q) = (2, 2)$ and the distance between two nearest threshold prices (Δ) was set to 15.

For the pessimistic test case, 1000 bid values were generated using a normal distribution function with a mean value of 212.50 and standard deviation of 15. For the optimistic test case, another 1000 bid values were generated with a mean value of 243.50 and standard deviation of 15. For both test cases, it was assumed the hotelier's NYOP system had threshold price set at 225. This had to be known beforehand in order to explicitly generate bid values to match the pessimistic and the optimistic cases.

Each of the test cases (optimistic and pessimistic tests) had two control tests. The two control tests differ from each other based on how the action space is traversed. The two possibilities here are, first evaluating the threshold price in the increasing order of the threshold price value and then followed by decreasing order (optimistic direction). The second option is the reverse of it, where the test starts with decreasing order and then increasing the order of threshold price values (pessimistic direction). What this represents is the hotelier first increasing and then decreasing the threshold price or vice versa in order to gauge the buyer surplus. These tests also allowed to check if the order of choosing an action (optimistic action first or pessimistic action first) has any effect on the overall outcome, compared with the concurrent multi-action evaluation technique.

5.6.1 Results

The expansion from the closest known context's adaptive action's threshold value resulted in five actions that will evaluate bids with 5 different threshold values. The threshold values were 195, 210, 225, 240 and 255. These are denoted as A(195), A(210), A(225), A(240) and A(250) in the graphs below.

5.6.2 Results for Pessimistic Test Case

The result of the pessimistic test case (Figure 30) shows that under the current unknown context the majority of successful bid values were evaluated by an action that had a threshold value of 210.

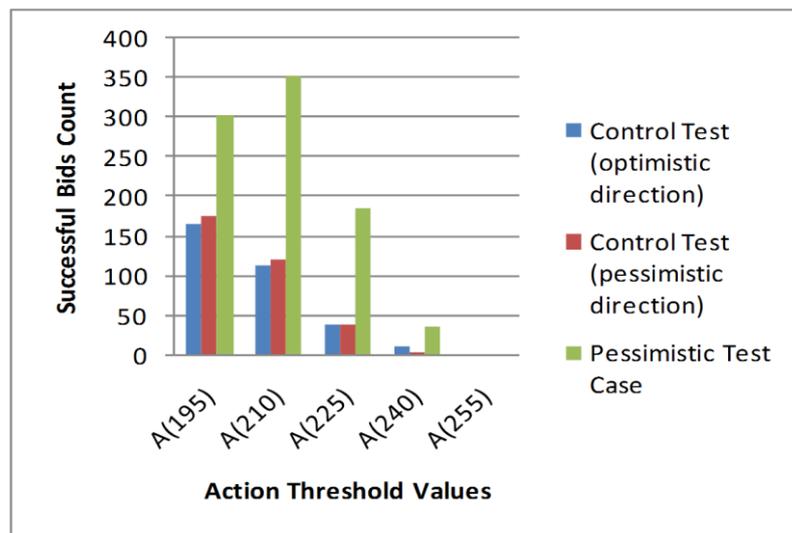


Figure 30. Successful Bid Count for the Test Case Where Unknown Context Results in Pessimistic Bid Values

In this context the hotelier has to lower his threshold price to 210 in order to have a high revenue. In essence, the hotelier could associate the current unknown context with the threshold value 210 for future evaluation, thus effectively evolving the system to recognize the current unknown context in the future. This conclusion is known to be correct as the bid values for the pessimistic test case were generated using a normal distribution with a mean value of 212.50.

In the pessimistic test case, 880 out of 1000 bid values succeeded in one threshold value or another. The remaining values were less than 195, thus rejected. However, looking at the control test cases in the optimistic direction, there were only 332 successful bid values while in the pessimistic direction there were 342 bid values. The low success rate is due to employing only a single action, which evaluates the bids by using only a single threshold value.

Furthermore, the results of the test case erroneously show 195 to be the threshold value under which the majority of bids are successful. This would result in unnecessarily lowering the hotelier's profit margin. Figure 31 shows the highest yield is obtained using the adaptive threshold price, which was made possible by the concurrent multi-action evaluation technique.

This confirms that the concurrent multi-action evaluation gives the fluidity and flexibility needed to react to an unknown context compared to the self-learning adaptive techniques with an iterative approach.

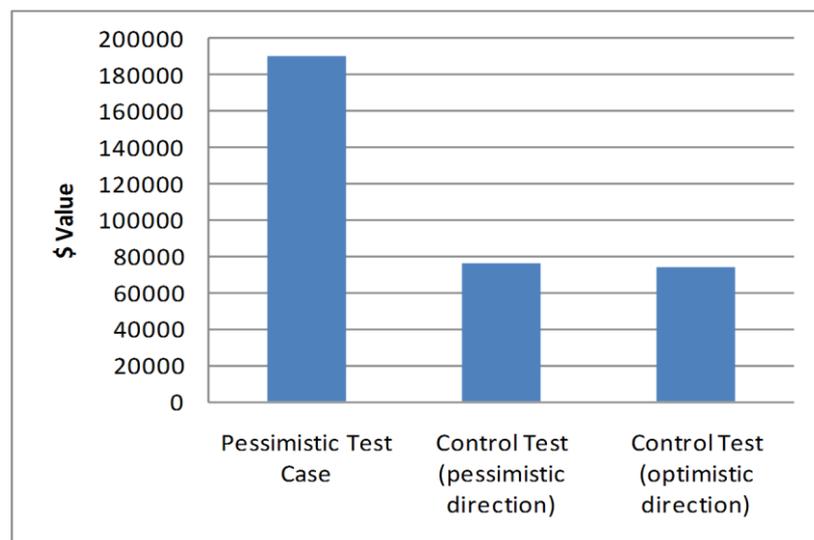


Figure 31. Yield for Pessimistic Test Case

5.6.3 Results for Optimistic Test Case

The results of the optimistic test case (Figure 32) show that under the current unknown context the majority of successful bids occur with an action that

evaluated them with a threshold value of 240. This is known to be true as the bid values were generated under the normal distribution that had a mean value of 243.50. Thus, the hotelier's NYOP channel system could be evolved by associating the context with the action of setting the threshold price to 240. Out of 1000 bids, 999 were successful, while the remaining one value was less than 195 and therefore was rejected.

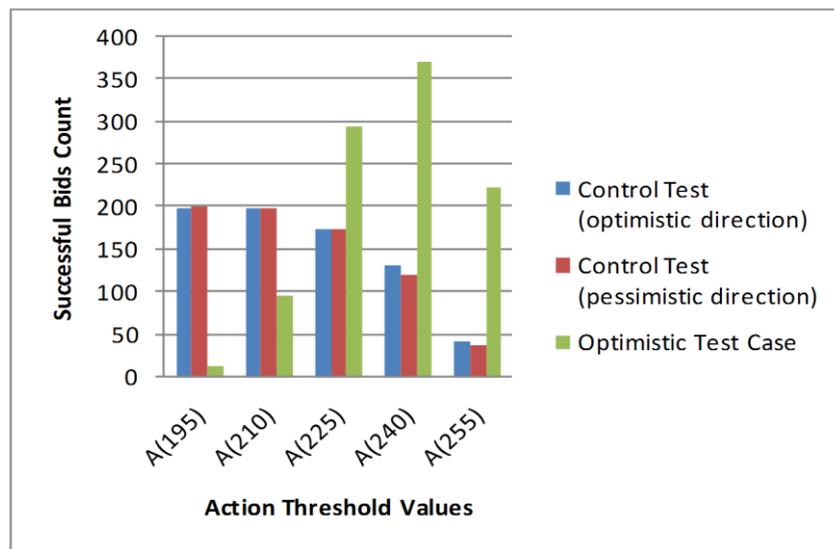


Figure 32. Successful Bid Count for the Test Case Where Unknown Context Results in Optimistic Bid Values

The control tests in both pessimistic and optimistic directions resulted in erroneously identifying that using a threshold value of 195 is the correct course of action. In fact, this indicates that the context change or the unknown context encountered has a pessimistic effect on the bid values, when in fact the opposite is known to be true. Out of 1000 bid values only 732 and 745 were successful in the pessimistic direction and the optimistic direction test cases respectively.

The sum of successful bid values in (Figure 33) shows that the concurrent multi-action evaluation yielded higher revenue than the self-learning techniques using iterative approach. This affirms the fact that the concurrent multi-action

evaluation is able to accurately predict the impact the change of context has on the bid values. A context-aware self-adapting NYOP channel system would stand to gain higher revenue compared to a system which uses self-learning techniques with an iterative approach for adaptation.

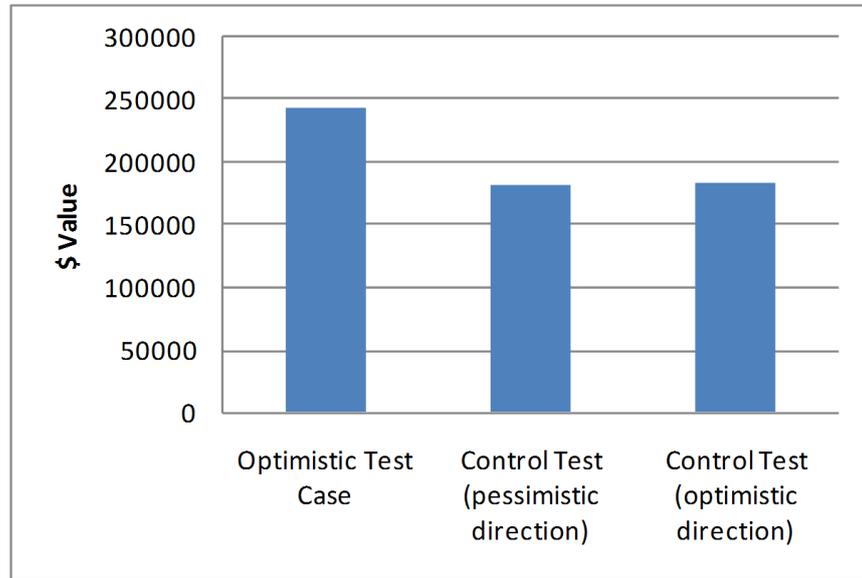


Figure 33. Yield for the Optimistic Test Case

5.7 Conclusion of the Case Study

An implementation of the proposed generic framework was completed for the NYOP scenario. The experimental results from both test cases showed that concurrent multi-action execution and evaluation is able to identify the best action for an unknown context, thus evolving the system to work with previously unknown contexts. These promising results complement very well the much faster evaluation time for our concurrent multi-action framework when compared to context-aware systems based on the iterative approach.

Though this use case was implemented for NYOP channel, the proposed framework could be easily adopted for any domain that allows concurrent multi-action evaluation.

6. FUTURE DIRECTIONS

The versatility of the proposed framework has been demonstrated with the implementation of two case studies from two distinct domains. A solution based on the proposed framework has been envisaged to achieve thermal energy efficiencies in high-end processors.

6.1 Need for Energy Tuning in Processors

Thermal energy-related issues, such as overheating, are a serious concern in modern high-end processors. The high temperatures not only reduce the reliability [192] and the lifetime of the underlying hardware but also increase the power consumption. High power consumption is a major issue for commercial data centres as this leads to higher cooling bills and high carbon footprints. All major processor manufacturers correlate the maximum expected performance with the thermal design power (TDP) [193]. TDP is the maximum amount of temperature and power that could be sustained by the processor over a long execution period of a typical workload.

The problem in managing the thermal energy in processors comes from the fact that there are physical properties that influence the thermal behavior of the processor. Besides the physical properties, the nature of the workload also affects the power consumption of the processor. For a user running a workload on a processor, the physical property is something immutable. As such research work has been geared towards application-specific thermal energy models. Such a model is presented in [194] where asymptotic equilibrium temperature (T_{∞}^h) is presented as

$$T_{\infty}^h = \tau_{\infty}^h + T_0$$

Aggregated asymptotic equilibrium temperature includes three components, which are ambient (power-off) temperature, idle equilibrium temperature, and the

asymptotic equilibrium temperature. The ambient (power-off) temperature and the idle equilibrium temperature (T_0) are static components. However, the asymptotic equilibrium temperature (τ_∞^h) is transient and solely dependent on the application workload and representative of additional power consumed during the execution of the workload.

The experimentation in [194] was carried out to find the lowest power consumption for a constant workload by varying the core count and the CPU scaling governor of the processor. The CPU scaling governor controls the P-states of the CPU of which the two extreme cases are the power save scaling governor (low P-state) and the on-demand scaling governor (high P-state). Through a brute force method of experimentation, the power consumption for all cores and P-states combinations were obtained for a constant workload. With this result in hand, thermal energy efficiencies are achieved by configuring the CPU to the core count that resulted in minimum power consumption during the experimentation.

6.2 Using the Proposed Framework for Thermal Energy Modelling

However, the approach in [194] has two problems:

1. The use of brute force method to vary the core count.
2. The use of constant workload.

Having to vary the core count for each P-state means longer experimentation time. For example, for a 16-core processor, this would mean running 32 experiments (16 for each P-state) to find the ideal core count for low power consumption. Secondly, whenever workload changes the experimentation has to

be carried out again as asymptotic equilibrium temperature is solely dependent on the workload characteristics.

These issues could be addressed with an implementation of the proposed framework with its use of concurrent multi-action evaluation. A description of a potential solution is given here which is awaiting implementation and testing.

The knowledge base is modelled as in Table 4 where each workload is associated with a core count, asymptotic equilibrium temperature, and power consumption.

Table 4. Knowledge Base for Thermal Modelling

Workload	Core Count	Asymptotic Equilibrium Temperature	Power Consumption
W_1	C_1	τ_1	P_1
W_k	C_k	τ_k	P_k
W_n	C_n	τ_n	P_n
W_i	$C?$	$\tau?$	$P?$

What is expected of the concurrent multi-action evaluation system is to find out the core count ($C?$) which results in minimum power (or temperature) consumption for unknown workload W_i . An analogy could be drawn between the above model and the Chapter 4 case study of database performance tuning. By defining the workload as the context and adaptive action as a function of the core count being set $A(C_i; i= 1 - n)$, an action space for concurrent multi-action evaluation could be devised. The initial core count could be obtained using the closeness measure of the workloads. This approach reduces the number of experiments needed to find the core count that achieved desired goal expectations.

The same experimentation model could be used to answer optimization queries such as “what is the minimum CPU count required to complete the work

under given SLAs but with minimum power consumption” or for identification of the highest performance for a different number of CPUs and different consumed power. The implementation of this model requires special hardware equipment and the author is waiting for the next available opportunity.

7. CONCLUSION

Context-aware computing has come a long way since sprouting out as part of a vision for ubiquitous computing. At a time when mobile computing was a novelty, context-awareness was the key to keeping computing power uniform and relevant from a user's perspective. The expectation was for computers to be aware of their location, dedicated to the particular task and finally users to be unaware of the origins of the computing power they are consuming. The early models developed for context-aware computing had location as the only aspect when it came to defining a context. This led to narrow and constrained definitions of context and context-aware applications. Chapter 1 collates the evolution of the definitions over the years as new aspects were considered and attempts were made to arrive at a wide-reaching generic definition.

The research community's effort had resulted in a broader context definition, however, the self-adaptation techniques used by the context-aware applications had limitations resulting from the legacy context definitions they were modeled on. The existing self-adapting techniques are specific to either single context – single action or single context – multi-action use cases. The single context – single action model worked well with legacy context definitions which considered location as the only aspect defining the context. Each context change had exactly one corresponding adaptive action. The context and action spaces were limited and finite in size. This required the context-aware application developers to embed all possible “context change – adaptive action” pairs into the system. Context inference failure would occur if the system encounters a context change that was not foreseen. Context-aware applications developed using the single context – single action model were unable to adapt to unknown context. The single context – multi-action model was introduced as a solution to the above limitations. In this model when an unknown context is

encountered the system would iterate over a finite set of pre-defined actions to find the best adaptive action. However, the iterative approach suffers from serious scalability limitations particularly when the size of the action space increases in multi-context – multi-action scenarios.

Thus, the objective of this project was to address the limitation in existing self-adapting techniques used by the context-aware systems when it comes to multi-context – multi-action scenarios. The research identified five issues in the existing body of work to address.

1. Dependency on system developers to pre-define context and context elements which cause rigid context declarations.
2. Dependency on system developers to encompass and embed all possible context change-adaptive action pairs into the system.
3. Inability to carry out system adaptation when an unknown context is encountered.
4. The time for system adaptation being dependent on the size of the action space (i.e. the number of adaptive actions to be evaluated).
5. The need of user intervention to expand the knowledge base which stores context and adaptive action information.

The research introduced a novel generic framework which solves the above issues by achieving self-adaptation via concurrent multi-action evaluation. The framework consists of three systems called context, inference, and action. The project addresses the above objectives by decoupling context and action spaces, creating a dynamic action space for evaluation and using just five user inputs and a self-updating knowledge base model. Chapter 3 provides an extensive description of

each of the systems along with how each system either individually or together with another system addresses the research objectives.

The domain agnostic formal model as derived in this project delivers an ease of adoption and portability for context-aware developers when carrying out concrete implementations of the novel concurrent multi-action evaluation technique. Two case studies were carried out to validate the concept of self-adaptation via concurrent multi-action evaluation and to showcase the ease of cross domain adaptability of the framework.

The first case study uses an implementation of the framework for performance tuning of a database when it encounters a previously unknown workload mix. The experiment resulted in a 20% reduction in CPU usage when self-adaptation via concurrent multi-action valuation is in use, compared to the non-adaptive setting. This achievement validates the soundness of the formal model and its implementation.

The second case study involves finding the threshold price for name-your-own-price channels when a hotel booking system encounters an unknown context. On this occasion, the implementation of the framework is compared against an existing adaptation technique with an iterative approach. The results show that self-adaptation via concurrent multi-action evaluation correctly identified the nature of the context change and is able to set the appropriate threshold price. On the contrary, the iterative approach resulted in either erroneous interpretation of the context change or setting threshold prices unsuitable to the detected context change. This achievement cemented the superiority of the novel self-adaption technique proposed in this research project compared to existing self-adapting techniques when it came to multi-context – multi-action scenarios.

As a further proof of the versatility of the framework, future work involving thermal energy modelling in high-end processors has been planned.

The implementations of the framework could be used in other domains to solve problems associated with multi-context multi-action model. For example, a cloud provider could use the proposed framework to adjust the market value of spot instances [195] using wide variety of contexts such as current resource availability, current demand and future demand for spot instances and even react quickly to dynamic changes in the competitors' spot instance market values. Another possible area of use is dynamic resize of a computer cluster based on context. In this case, the multi-context space could consist of workload and user types and aspects of SLA related constraints. The concurrent multi-action evaluation technique could be used to dynamically shrink or expand the cluster through concurrent multi-action evaluation. In short, due to the domain agnostic nature of the framework it could be implemented in any domain to achieve self-adaptation in multi-context multi-action cases.

The contributions to knowledge by this research could be summarized as below.

- The generic framework provides a domain independent method for formulating context spaces. It eliminates the dependency on system developers for context space formulation and rigid context definitions. Facilitating context addition and removal with ease in multi-context models.
- The novel context-aware architecture of the framework fully decouples the context-system from the action system. The decoupling makes changes made in the context space agnostic to the concurrent action implementation in the action system. Both context and action space can grow and shrink independently of each other. This

feature negates the need for system developers to encompass and embed all context change – adaptive action pairs into the system.

- The research expanded the concept of context attribute by introducing a “priority” as a property of a context attribute. The priority is based on the degree of influence each context attribute has in the prevailing context. With the introduction of the priority, if there are two or more contexts with equal distances to the unknown context, then the priority of each context could be used to discriminate between the equidistance contexts.

- This research introduced the novel concept of defining adaptive action as a function of the parameter manipulated by it. This concept allows the dynamic creation of action space based on just five pre-defined user inputs.

- The novel experiment-based self-adaption technique formulated by the research uses concurrent multi-action evaluation to speeds up self-adaption in context-aware application when in multi-context multi-action models.

In conclusion, the thesis presents a novel context-aware application framework which enables self-adaptation via concurrent multi-action evaluation when an unknown context is encountered. The proposed framework eliminates the limitations of the currently existing approaches by employing an experiment-based adaptation mechanism. A concrete implementation of this framework was carried out showing how it could be used in a variety of domains. As a result of this work, a new paradigm of a self-adaptation technique for context-aware application has risen. The taxonomy presented in Chapter 2 shows the unique placement of this novel self-adaptation technique among the existing body of work. It is believed that this new contribution to knowledge will lead to further experiment-based self-adaptation techniques.

REFERENCES

1. M. Weiser, (1991). "The computer for the 21st century," *Scientific American*, vol. 265, no. 3, pp. 66–75.
2. M. Rouse, (2010). Pervasive computing (ubiquitous computing). [online] Available from: <<http://internetofthingsagenda.techtarget.com/definition/pervasive-computing-ubiquitous-computing>> [Accessed 10 October 2016].
3. Reddy, Y.V., (2006). Pervasive Computing: Implications, Opportunities and Challenges for the Society, *Pervasive Computing and Applications, 2006 1st International Symposium on*. 2006, pp. 5-5.
4. Voids, S. *et al.*, (2002). Integrating virtual and physical context to support knowledge workers. *Pervasive Computing, IEEE*. **99** (3), 73-79.
5. Fei-Yue Wang *et al.*, (2006). Smart Cars on Smart Roads: An IEEE Intelligent Transportation Systems Society Update. *Pervasive Computing, IEEE*. **5** (4), 68-69.
6. Bayraktar, C. *et al.*, (2011). Diagnosing internal illnesses using pervasive healthcare computing and neural networks. *Procedia Computer Science*. **3** (0), 584-588.
7. Coronato, A. and Pietro, G.D., (2010). Formal specification of wireless and pervasive healthcare applications. *ACM Trans.Embed.Comput.Syst.* **10** (1), 12:1-12:18.
8. Saxena, N. *et al.*, (2009). Near-Optimal Tracking for Residents' Comfort in Context-Aware Heterogeneous Smart Environments. *The Computer Journal*. **52** (8), 878-889.
9. Abowd, G.D. *et al.*, (2005). Guest Editors' Introduction: The Smart Phone--A First Platform for Pervasive Computing. *Pervasive Computing, IEEE*. **4** (2), 18-19.
10. Intel, (2014). Using Apache Hadoop* for Context-Aware Recommender Systems. [online] Available from: <<http://intel.ly/1nRRoUZ>> [Accessed 14 July 2014].
11. Jiehan Zhou *et al.*, (2010). Pervasive Service Computing: Visions and Challenges, *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*. 2010, pp. 1335-1339.
12. Ruimin Liu *et al.*, (2004). An evolutionary system development approach in a pervasive computing environment, *Cyberworlds, 2004 International Conference on*. 2004, pp. 194-199.
13. Zhang, D. *et al.*, (2010). Context reasoning using extended evidence theory in pervasive computing environments. *Future Generation Computer Systems*. **26** (2), 207-216.
14. Padovitz, A. *et al.*, (2008). Multiple-Agent Perspectives in Reasoning About Situations for Context-Aware Pervasive Computing Systems. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*. **38** (4), 729-742.

15. Paechter, B. *et al.*, (2011). Heaven and Hell: Visions for Pervasive Adaptation. *Procedia Computer Science*. 7 (0), 81-82.
16. Judge, R., (2000). Ubiquitous? Pervasive? Sorry, they don't compute. [online] Available from: < <http://www.computerworld.com/article/2593079/ubiquitous--pervasive--sorry--they-don-t-compute.html>> [Accessed 13 October 2016].
17. Schilit, B.N. and Theimer, M.M., (1994). Disseminating active map information to mobile hosts. *Network, IEEE*. 8 (5), 22-32.
18. Brown, P.J. *et al.*, (1997). Context-aware applications: from the laboratory to the marketplace. *Personal Communications, IEEE*. 4 (5), 58-64.
19. Ryan, N., Pascoe, J., Morse, D., (1997). Enhanced Reality Fieldwork: the Context-Aware Archaeological Assistant. Gaffney, V., van Leusen, M., Exxon, S. (eds.) *Computer Applications in Archaeology*, [online] Available from: < <https://www.cs.kent.ac.uk/pubs/1998/616/content.html>> [Accessed 20 October 2016].
20. Dey, A.K., (1998). Context-aware computing: The CyberDesk project. AAAI 1998 *Spring Symposium on Intelligent Environments*, Technical Report SS-98-02 (1998) 51-54.
21. Hull, R., Neaves, P., Bedford-Roberts, J., (1997). Towards Situated Computing. 1st *International Symposium on Wearable Computers*, 146-153.
22. Franklin, D., Flaschbart, J., (1998). All Gadget and No Representation Makes Jack a Dull Environment. AAAI 1998 *Spring Symposium on Intelligent Environments, Technical Report SS-98-02*. 155-160.
23. Brown, P.J., (1996). The Stick-e Document: a Framework for Creating Context-Aware Applications. *Electronic Publishing '96*. 259-272.
24. Ward A., Jones, A., Hopper, A., (1997). A New Location Technique for the Active Office. *IEEE Personal Communications* 4(5). 42-47.
25. Rodden, T., Cheverst, K., Davies, K. Dix, A., (1998). Exploiting Context in HCI Design for Mobile Systems. Workshop on Human Computer Interaction with Mobile Devices.
26. Schilit, B., Adams, N. Want, R., (1994). Context-Aware Computing Applications. 1st International Workshop on Mobile Computing Systems and Applications. 85-90.
27. Pascoe, J., (1998). Adding Generic Contextual Capabilities to Wearable Computers. 2nd International Symposium on Wearable Computers. 92-99.
28. Abowd GD, Ebling MR, Gellersen HW (2002) Context-Aware Pervasive Computing. *IEEE Wireless Communications*. 9(5): 8-9.
29. Dey, A. K and Abowd G. D., (1999). Towards a better understanding of context and context-awareness, in In HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing. *Springer-Verlag*, pp. 304-307.
30. Pascoe, J., Ryan, N.S., Morse, D.R., (1998). Human-Computer-Giraffe Interaction – HCI in the Field. Workshop on Human Computer Interaction with Mobile Devices.

31. Salber, D., Dey, A.K., Abowd, G.D., (1998). Ubiquitous Computing: Defining an HCI Research Agenda for an Emerging Interaction Paradigm. *Georgia Tech GVU Technical Report GIT-GVU-98-01*.
32. Dey, A.K., Abowd, G.D., Wood, A., (1999). CyberDesk: A Framework for Providing Self-Integrating Context-Aware Services. *Knowledge-Based Systems*, 11 3-13.
33. Ryan, N., (1997). Mobile Computing in a Fieldwork Environment: Metadata Elements. Project working document, version 0.2.
34. Abowd, G.D., Dey, A.K., Orr, R., Brotherton, J., (1997). Context-Awareness in Wearable and Ubiquitous Computing. 1st International Symposium on Wearable Computers. 179-180.
35. Davies, N., Mitchell, K., Cheverst, K. Blair, G., (1998). Developing a Context Sensitive Tourist Guide. 1st Workshop on Human Computer Interaction with Mobile Devices, *GIST Technical Report*. G98-1.
36. Dey, A.K., Abowd, G.D., (1997). CyberDesk: The Use of Perception in Context-Aware Computing. 1st Workshop on Perceptual User Interfaces. 26-27.
37. Korteum, G., Segall, Z., Bauer, M., (1998). Context-Aware, Adaptive Wearable Computers as Remote Interfaces to “Intelligent” Environments. 2nd International Symposium on Wearable Computers. 58-65.
38. Ward, A., Jones, A., Hopper, A., (1997). A New Location Technique for the Active Office. *IEEE Personal Communications* 4(5). 42-47.
39. Brown, P.J., (1998). Triggering Information by Context. *Personal Technologies*. 2(1) 1-9.
40. Dey, A.K. *et al.*, (2004). a CAPpella: programming by demonstration of context-aware applications, *Proceedings of the SIGCHI conference on Human factors in computing systems*. Vienna, Austria. New York, NY, USA: ACM, 2004, pp. 33-40.
41. Castelli, V. *et al.*, (2007). Augmentation-Based Learning combining observations and user edits for Programming-by-Demonstration. *Knowledge-Based Systems*. **20** (6), 575-591.
42. Kawsar, F. *et al.*, (2010). A portable toolkit for supporting end-user personalization and control in context-aware applications. *Multimedia Tools Appl.* 47, 3 (May 2010), 409-432.
43. Figo, D. *et al.*, (2010). Preprocessing techniques for context recognition from accelerometer data. Springer London.
44. Cipriani, N. *et al.*, (2011). Tool support for the design and management of context models. *Information Systems*. **36** (1), 99-114.
45. Guo, B. *et al.*, (2011). Toward a cooperative programming framework for context-aware applications. *Personal Ubiquitous Comput.* 15, 3 (March 2011), 221-233.
46. Lian Yu. *et al.*, (2011). Building Customizable Context-aware Systems, *Service Sciences (IJCSS), 2011 International Joint Conference on*. 2011, pp. 252-256.

47. Loke, S.W., (2010). Incremental awareness and compositionality: A design philosophy for context-aware pervasive systems. *Pervasive and Mobile Computing*. **6** (2), 239-253.
48. Abeywickrama, D.B. and Ramakrishnan, S., (2012). Context-aware services engineering: Models, transformations, and verification. *ACM Trans.Internet Technol.* **11** (3), 10:1-10:28.
49. Li, L. *et al.*, (2011). Semantic based aspect-oriented programming for context-aware Web service composition. *Information Systems*. **36** (3), 551-564.
50. Hwang, Z. *et al.*, (2007). A Context Model by Ontology and Rule for Offering the User-Centric Services in Ubiquitous Computing, *Convergence Information Technology, 2007. International Conference on*. 2007, pp. 77-82.
51. Ejigu, D. *et al.*, (2007). Semantic approach to context management and reasoning in ubiquitous context-aware systems, *Digital Information Management, 2007. ICDIM '07. 2nd International Conference on*. 2007, pp. 500-505.
52. Jaewoo Chang *et al.*, (2008). Intelligent Context-Aware System Architecture in Pervasive Computing Environment, *Parallel and Distributed Processing with Applications, 2008. ISPA '08. International Symposium on*. 2008, pp. 745-750.
53. Laddaga, R., (1997). Self-adaptive software: A Position Paper. Tech. Rep. 98-12, DARPA BAA.
54. Oreizy, P. *et al.*, (1999). An architecture-based approach to self-adaptive software. *Intelligent Systems and their Applications, IEEE*. **14** (3), 54-62.
55. Salehie, M. and Tahvildari, L., (2009). Self-adaptive software: Landscape and research challenges. *ACM Trans.Auton.Adapt.Syst.* **4** (2), 14:1-14:42.
56. Kephart, J.O. and Chess, D.M., (2003). The vision of autonomic computing. *Computer*. **36** (1), 41-50.
57. Montana, D. and Hussain, T., (2004). Adaptive reconfiguration of data networks using genetic algorithms. *Applied Soft Computing*. **4** (4), 433-444.
58. Huebscher, M.C. and McCann, J.A., (2008). A survey of autonomic computing—degrees, models, and applications. *ACM Comput.Surv.* **40** (3), 7:1-7:28.
59. Bahati, R.M. and Bauer, M.A., (2008). Adapting to Run-Time Changes in Policies Driving Autonomic Management, *Autonomic and Autonomous Systems, 2008. ICAS 2008. Fourth International Conference on*. pp. 88-93.
60. Kalyvianaki, E. *et al.*, (2009). Self-adaptive and self-configured CPU resource provisioning for virtualized servers using Kalman filters, *Proceedings of the 6th international conference on Autonomic computing*. Barcelona, Spain. New York, NY, USA: ACM, 2009, pp. 117-126.
61. Ramirez, A.J. *et al.*, (2009). Applying genetic algorithms to decision making in autonomic computing systems, *Proceedings of the 6th international conference on Autonomic computing*. Barcelona, Spain. New York, NY, USA: ACM, 2009, pp. 97-106.
62. Schmeck, H. *et al.*, (2010). Adaptivity and self-organization in organic computing systems. *ACM Trans.Auton.Adapt.Syst.* **5** (3), 10:1-10:32.

63. Wilson, J.G. and Zhang, G., (2008). Optimal design of a name- your- own price channel. *Journal of Revenue and Pricing Management*. 7 (3), 281-290.
64. Henricksen, K.,(2012). A framework for context-aware pervasive computing applications, Ph.D Thesis, The University of Queensland, [online] Available from <<http://henricksen.id.au/publications/phd-thesis.pdf>> [Accessed 20 October 2016].
65. Nimalasena, A. and Getov, V., (2014). Performance Tuning of Database Systems Using a Context-aware Approach. *in: Proc. 9th International Conference on Computer Engineering & Systems (ICCES) Cairo IEEE*. pp. 98 – 103.
66. Nimalasena, A. and Getov, V., (2015). Context-Aware Framework for Performance Tuning via Multi-action Evaluation. *in: 2015 IEEE 39th Annual Computer Software and Applications Conference (COMPSAC) Taichung, Taiwan IEEE*. pp. 318 – 323.
67. Nimalasena, A. and Getov, V., (2013). System evolution for unknown context through multi-action evaluation, *in: Proceedings of 2013 IEEE 37th Annual Computer Software and Applications Conference (COMPSAC) IEEE*. pp. 271-276.
68. Nimalasena, A. and Getov, V., (2016). Context-aware Approach for Determining the Threshold Price in Name-Your-Own-Price Channels. *in: Context-Aware Systems and Applications*. Springer. pp. 83-93.
69. Wooldridge, M. and Jennings, N. R., (1995). Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, vol. 10(2) pp. 115-152.
70. Horn, P., (2001). Autonomic computing: IBM's perspective on the state of information technology. IBM T.J. Watson Lab.
71. Paulson, L.D., (2002). Computer system, heal thyself. *Computer*, 35(8), pp.20-22.
72. Bantz, D. F. *et al.*, (2003). Autonomic personal computing in *IBM Systems Journal*, vol. 42, no. 1, pp. 165-176.
73. Kephart, J. O. and Chess D. M., (2003). The vision of autonomic computing, *Computer*, vol. 36, no. 1, pp. 41-50.
74. Tianfield, H., (2003). Multi-agent autonomic architecture and its application in e-medicine. *In IEEE/WIC International Conference on Intelligent Agent Technology (IAT 2003)*, pages 601–604.
75. Tesauro, G. *et al.*, (2004). A Multi-agent systems approach to autonomic computing. *In IBM Press*, pages 464–471.
76. Sterritt, R. *et al.*, (2005). A concise introduction to autonomic computing. *In Advanced Engineering Informatics*, volume 19, pages 181–187.
77. Poslad, S., (2009). Autonomous systems and Artificial Life, In: *Ubiquitous Computing Smart Devices, Smart Environments and Smart Interaction*. Wiley. pp. 317–341.
78. Nami, M.R. and Bertels, K., (2007). A survey of autonomic computing systems - *Intelligent Information Processing III: IFIP TC12 International Conference on Intelligent Information Processing (IIP 2006)*, 26–30.

79. Ganek, A. and Friedrich, R.J., (2006). The road ahead—achieving wide-scale deployment of autonomic technologies. *In Proceedings of the 3rd IEEE International Conference on Autonomic Computing.*
80. Lippman, A., (1999). Video coding for multiple target audiences. *In Proceedings of the IS&T/SPIE Conference on Visual Communications and Image Processing.* 780–784.
81. McCann, J. A. and Crane, J. S., (1998). Kendra: Internet distribution and delivery system: An introductory paper. *In Proceedings of the SCS EuroMedia Conference,* 134–140.
82. Macías-Escrivá, F.D. *et al.*, (2013). Self-adaptive systems: A survey of current approaches, *research challenges and applications, Expert Systems with Applications,* Volume 40, Issue 18, 7267-7279.
83. Ravindranathan, M. and Leitch, R., (1998). Heterogeneous intelligent control systems. *In IEE Proceedings – Control Theory and Applications,* Vol. 14S, pp. 551–558.
84. Brun, Y. *et al.*, (2009). Engineering self-adaptive systems through feedback loops. *In Software engineering for self-adaptive systems,* Vol. 5525, pp. 48–70.
85. Naqvi, M., (2012). Claims and supporting evidence for self-adaptive systems – A literature review. [online] Available from: < <http://nu.diva-portal.org/smash/get/diva2:512439/FULLTEXT01.pdf>> [Accessed 20 October 2016].
86. Salehie, M. and Tahvildari, L., (2012). Towards a goal-driven approach to action selection in self-adaptive software. *Softw. Pract. Exper.* 42, 2, 211-233.
87. Krupitzer, A.C. *et al.*, (2015). A survey on engineering approaches for self-adaptive systems. *Pervasive and Mobile Computing.* 17, 184–206.
88. Vinh, P.C., (2016). Finite Limits and Colimits in Autonomic Systems. *in: Context-Aware Systems and Applications.* Springer. pp. 10-20, ISBN 9783319292359.
89. Chen, H. *et al.*, (2004). Intelligent agents meet the semantic web in smart spaces. *Internet Computing, IEEE,* 8(6), pp.69-79.
90. Cao, Y. *et al.*, (2008). Follow Me, Follow You - Spatiotemporal Community Context Modeling and Adaptation for Mobile Information Systems. *In Proceedings of the Ninth International Conference on Mobile Data Management (MDM '08).* 108-115
91. Mizzaro, S. and Vassena, L., (2011). A social approach to context-aware retrieval. *World Wide Web,* 14(4), pp.377-405.
92. He, J. *et al.*, (2012). A smart Web service based on the context of things. *ACM Transactions on Internet Technology (TOIT),* 11(3), p.13.
93. Datta, S. K., Bonnet, C. and Nikaein, N., (2014). Self-adaptive battery and context aware mobile application development, *2014 International Wireless Communications and Mobile Computing Conference (IWCMC),* pp. 761-766.
94. Göndör, S., Uzun, A. and Küpper, A., (2011). Towards a dynamic adaption of capacity in mobile telephony networks using context information, *2011 11th International Conference on ITS Telecommunications,* pp. 606-612.

95. O'Connor, N. *et al.*, (2007). Self-Adapting Context Definition, *Self-Adaptive and Self-Organizing Systems, 2007. SASO '07. First International Conference on. 2007*, pp. 336-339.
96. Tesauro, G. *et al.*, (2004). A multi-agent systems approach to autonomic computing, *in: Proc. AAMAS—Vol. 1, IEEE*, pp. 464–471.
97. Elkhodary, A., Malek, S., Esfahani, N., (2009). On the role of features in analyzing the architecture of self-adaptive software systems, *in: Proc. MRT, ACM/IEEE*, pp. 41–50.
98. Elkhodary, A., Esfahani, N., Malek, S., (2010). FUSION: a framework for engineering self-tuning self-adaptive software systems, *in: Proc. FSE, ACM*, pp. 7–16.
99. Fisch, D., Kalkowski, E., Sick, B., (2011). Collaborative learning by knowledge exchange, *in: Organic Computing—A Paradigm Shift for Complex Systems, Springer*, pp. 267–280.
100. Sutton, R. and Barto, A., (1998). Reinforcement Learning. MIT Press, 1998.
101. Dowling, J. and Cahill, V., (2004). Self-managed decentralised systems using K-components and collaborative reinforcement learning, *in: Proc. WOSS, ACM*, pp. 39–43.
102. Parra, C. *et al.*, (2012). Using constraint-based optimization and variability to support continuous self-adaptation, *in: Proc. SAC, ACM*, pp. 486–491.
103. Abdelwahed, S., Kandasamy, N., Neema, S., (2004). A control-based framework for self-managing distributed computing systems, *in: Proc. WOSS, ACM*, pp. 3–7.
104. Prothmann, H. *et al.*, (2009), Organic traffic light control for urban road networks, *in: Int. J. Autonomous and Adaptive Communications Systems*, Vol. 2, No. 3, pp. 203–225.
105. Bäck, T. and Schwefel, H.-P., (1993). An overview of evolutionary algorithms for parameter optimization, *Evol. Comput. 1 (1)*, 1–23.
106. McNaull, J. *et al.*, (2014). Flexible context aware interface for ambient assisted living, *Human-Centric Computation and Information Sciences*, vol. 4, pp. 1–41.
107. Nwiabu, N. *et al.*, (2011). Situation awareness in context-aware case-based decision support, *2011 IEEE International Multi-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support (CogSIMA)*, pp. 9-16.
108. Salomie, I. *et al.*, (2008). RAP-a basic context awareness model, *In Intelligent Computer Communication and Processing, 2008. ICCP 2008. 4th International Conference on*, pp. 315-318.
109. Cioara, T. *et al.*, (2010). A self-adapting algorithm for context aware systems, *In Roedunet International Conference (RoEduNet)*, pp. 374-379.
110. Saidane, A., 2007, June. Adaptive context-aware access control policy in ad-hoc networks. (p. 13).

111. Zheng, M., Xu, Q. and Fan, H., (2016). Modeling The Adaption Rule in Context-aware Systems. *International Journal of Ad hoc, Sensor & Ubiquitous Computing (IJASUC)*, Vol.7.
112. Samulowitz, M., Michahelles, F. and Linnhoff-Popien, C. (2001). Adaptive interaction for enabling pervasive services. *In Proceedings of the 2nd ACM international workshop on Data engineering for wireless and mobile access (MobiDe '01)*, 20-26.
113. Dey, A.K., Abowd, G.D. and Salber, D., 2001. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-computer interaction*, 16(2), pp.97-166.
114. Khakpour, N. *et al.*, (2012). Formal modeling of evolving self-adaptive systems. *Sci. Comput. Program.* 78, 1, 3-26.
115. Ranganathan, A., Al-Muhtadi, J. and Campbell, R. H. (2004). Reasoning about Uncertain Contexts in Pervasive Computing Environments. *IEEE Pervasive Computing*, 62-70.
116. Zheng, H., Kang, B. and Kim, H, (2007). An Ontology-based Bayesian Network Approach for Representing Uncertainty in Clinical Practice Guidelines, *6th International Semantic Web Conference 2007 Workshop on Uncertainty Reasoning for the Semantic Web.* pp. 161–173.
117. Kwang-Eun, K. and Kwee-Bo, S., (2008). Development of context aware system based on Bayesian network driven context reasoning method and ontology context modeling, *Control, Automation and Systems, 2008. ICCAS 2008. International Conference on.* 2008, pp. 2309-2313.
118. Gu, T., Pung, H.K. and Zhang D.Q., (2004) A Bayesian approach for dealing with uncertain contexts, *In the Proceeding of the Second International Conference on Pervasive Computing (Pervasive 2004)*, Vienna, Austria, April 2004.
119. Ghahramani, Z., (2001). An introduction to hidden Markov models and Bayesian networks. *In Hidden Markov models. World Scientific Series In Machine Perception And Artificial Intelligence Series*, Vol. 45, 9-42.
120. Degirmenci, A., (2014) Introduction to Hidden Markov Models, Harvard University, [online] Available from: <http://scholar.harvard.edu/files/adeqirmenci/files/hmm_adeqirmenci_2014.pdf> [Accessed 10 October 2016].
121. Chang, C.K., Jiang, H.Y., Ming, H. and Oyama, K., (2009). Situ: a situation-theoretic approach to context-aware service evolution. *Services Computing, IEEE Transactions on*, 2(3), pp.261-275.
122. Richardson, M. and Domingos, P., (2006). Markov Logic Networks, *Machine Learning J.*, pp. 107-136.
123. Pearl, J. (1988). Probabilistic reasoning in intelligent systems: Networks of plausible inference. San Francisco, CA: Morgan Kaufmann.
124. Choi, C. *et al.*, (2016). Travel Destination Recommendation Based on Probabilistic Spatio-temporal Inference. *in: Context-Aware Systems and Applications.* Springer. pp. 94-100.

125. Perera, C. *et al.*, (2014). Context Aware Computing for The Internet of Things: A Survey, in *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 414-454.
126. Derczynski, L.R., Yang, B. and Jensen, C.S., (2013). March. Towards context-aware search and analysis on social media data. In *Proceedings of the 16th international conference on extending database technology*. pp. 137-142.
127. Hu, X. *et al.*, (2014). Multidimensional context-aware social network architecture for mobile crowdsensing. *IEEE Communications Magazine*, 52(6), pp.78-87.
128. Rao, D. H. and Saraf, S. S., (1996). Study of defuzzification methods of fuzzy logic controller for speed control of a DC motor, *Proceedings of International Conference on Power Electronics*, vol.2, pp. 782-787.
129. Saade, J.J. and Diab, H.B., (2004). Defuzzification methods and new techniques for fuzzy controllers. *Iranian journal of electrical and computer engineering*, 3(2), p.161.
130. Naaz, S., Alam, A. and Biswas, R., (2011). Effect of different defuzzification methods in a fuzzy based load balancing application. *IJCSI*, 8(5), pp.261-7.
131. Runkler, T.A., (1997). Selection of appropriate defuzzification methods using application specific properties. *IEEE Transactions on Fuzzy Systems*, 5(1), pp.72-79.
132. Luo, H. *et al.*, (2010), December. Adaptive wireless multimedia communications with context-awareness using ontology-based models. In *Global Telecommunications Conference (GLOBECOM 2010)*. pp. 1-5.
133. Hosseinzadeh, S. *et al.*, (2016). A semantic security framework and context-aware role-based access control ontology for smart spaces. In *Proceedings of the International Workshop on Semantic Big Data*. p8.
134. Kim, E.J. *et al.*, (2016). A Study on Knowledge-based Context Aware Framework using Machine Learning. *Advanced Science and Technology Letters* Vol.139 (FGCN 2016), pp.90-94.
135. Orciuoli, F. and Parente, M., (2016). An ontology-driven context-aware recommender system for indoor shopping based on cellular automata. *Journal of Ambient Intelligence and Humanized Computing*, pp.1-19.
136. Gassen, J.B. *et al.*, (2017). An experiment on an ontology-based support approach for process modeling. *Information and Software Technology*, 83, pp.94-115.
137. Tarus, J.K., Niu, Z. and Mustafa, G., (2017). Knowledge-based recommendation: a review of ontology-based recommender systems for e-learning. *Artificial Intelligence Review*, pp.1-28.
138. De Backere, F. *et al.*, (2017). The OCarePlatform: A context-aware system to support independent living. *Computer Methods and Programs in Biomedicine*. pp.111-120.
139. Lee, K., Lee, J. and Kwan, M.P., (2017). Location-based service using ontology-based semantic queries: A study with a focus on indoor activities in a university context. *Computers, Environment and Urban Systems*, 62, pp.41-52.

140. Machado, A., *et al.*, (2017). Reactive, proactive, and extensible situation-awareness in ambient assisted living. *Expert Systems with Applications*, 76, pp.21-35.
141. Chakrabarty, A. and Roy, S., (2017). An Ontology Based Context Aware Protocol in Healthcare Services. *In Proceedings of the First International Conference on Intelligent Computing and Communication*. pp. 137-146.
142. Rani, P.K. and Kannan, E., (2014). FFA-context aware energy efficient routing using fast reactive and adaptive algorithm. *American Journal of Applied Sciences*, 11(2), p.301.
143. Qi, B. and Shen, F., (2014). Performance comparison of partial swarm optimization variant models. *In Information Technology: New Generations (ITNG)*. pp. 575-580.
144. Belknap, P. *et al.*, (2009). Self-Tuning for SQL performance in Oracle database 11g. *ICDE '09. IEEE 25th International Conference on Data Engineering*. pp.1694, 1700.
145. Ziauddin, M. *et al.*, (2008). Optimizer plan change management: improved stability and performance in Oracle 11g. *VLDB '08*.
146. Yagoub, K. *et al.*, (2008). Oracle's SQL performance analyzer. *IEEE Data Engineering Bulletin*. **31(1)**.
147. Chi, C. *et al.*, (2013). Performance prediction for performance-sensitive queries based on algorithmic complexity, *Tsinghua Science and Technology* , vol.18, no.6, pp.618,628.
148. Herodotou, H. and Babu, S., (2009). Automated SQL tuning through trial and (sometimes) error. *In Proceedings of the Second International Workshop on Testing Database Systems (DBTest '09)*.
149. Babu, S. *et al.*, (2009). Automated experiment-driven management of (database) systems. *In Proceedings of the 12th conference on Hot topics in operating systems (HotOS'09)*, 19-19.
150. de Moraes Barbosa, E. B. and Silva, M. B., (2013). Influential Parameters to the Database Performance—A Study by Means of Design of Experiments (DoE). *DESIGN OF EXPERIMENTS-APPLICATIONS*, 41. [online] Available from: <<http://cdn.intechopen.com/pdfs-wm/45313.pdf>> [Accessed 14 January 2014]
151. Duan, S. *et al.*, (2009). Tuning database configuration parameters with iTuned. *Proceedings of the VLDB Endowment*, **2(1)**, 1246-1257.
152. Borisov, N.,and Babu, S. (2013). Rapid experimentation for testing and tuning a production database deployment. *In Proceedings of the 16th International Conference on Extending Database Technology (EDBT '13)*. 125-136.
153. Khattab,A.A. *et al.*, (2013). NNMonitor: performance modeling for database servers. *Computer Engineering & Systems (ICCES), 2013 8th International Conference*, pp.301, 306.
154. Oracle, (2010). SQL plan management in Oracle Database 11g. [online] Available from: <<http://www.oracle.com/technetwork/database/focus-areas/bi-datawarehousing/twp-sql-plan-management-11gr2-133099.pdf>> [Accessed 14 January 2014].

155. Oracle, (2013a). Optimizer with Oracle database 12c. [online] Available from: <<http://www.oracle.com/technetwork/database/bi-datawarehousing/twp-optimizer-with-oracledb-12c-1963236.pdf>> [Accessed 14 January 2014].
156. Galanis *et al.*, (2008) Oracle database replay. *SIGMOD Conference*. Pg. 1159-1170.
157. Colle, R. *et al.*, (2009) Oracle database replay. Proceedings of the VLDB Endowment 2, 1542–1545.
158. Holze, M., (2012). Self-Management Concepts for Relational Database Systems. Ph.D Thesis, University of Hamburg, [online] Available from <<http://ediss.sub.uni-hamburg.de/volltexte/2012/5503/pdf/Dissertation.pdf>> [Accessed 04 January 2014].
159. Ejigu, D. *et al.*, (2007). Semantic approach to context management and reasoning in ubiquitous context-aware systems, *Digital Information Management, 2007. ICDIM '07. 2nd International Conference on*. 2007, pp. 500-505.
160. Antoniou, G. and van Harmelen, F., (2004). A Semantic Web Primer. Cambridge: MIT Press, pp.145
161. Oracle, (2014). Database Reference 12c Release 1. [online] Available from: <<http://docs.oracle.com/database/121/REFRN/E41527-14.pdf>> [Accessed 14 January 2014].
162. Xu, T. *et al.*, (2013). Do not blame users for misconfigurations. *In Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP '13)*, 244-259.
163. Dias, K. *et al.*, (2005). Automatic performance diagnosis and tuning in Oracle. *In Proc. Conf. on Innovative Data Systems Research (CIDR)*. [online] Available from: <<http://www-db.cs.wisc.edu/cidr/cidr2005/papers/P07.pdf>> [Accessed 04 January 2014]
164. Oracle, (2013b). Database Sample Schemas. [online] Available from: <<http://docs.oracle.com/database/121/COMSC/E15979-04.pdf>> [Accessed 14 January 2014].
165. Dominic, G. (2010). Swingbench load generator. [online] Available from: <<http://dominicgiles.com/swingbench.html>> [Accessed 04 January 2014].
166. Terwiesch, C. *et al.*, (2005). Online Haggling at a Name- Your- Own- Price Retailer: Theory and Application. *Management Science*. 51 (3), 339-351.
167. Gelbrich, K. (2011). I have Paid Less than You! The Emotional and Behavioral Consequences of Advantage Inequality, *Journal of Retailing*, 87 (June) (2011), pp. 207–224.
168. Hann, I. and Terwiesch, C., (2003). Measuring the frictional costs of online transactions: The case of a name-your-own-price channel. *Manag. Sci.* **49**, 1563–1579.
169. Spann, M. *et al.*, (2004), Measuring Individual Frictional Costs and Willingness-to-Pay via Name-Your-Own-Price Mechanisms, *Journal of Interactive Marketing*, 18 (4), 22-36.

170. Cai, G.S. *et al.*, (2009). Optimal Reserve Prices in Name-Your-Own-Price Auctions with Bidding and Channel Options. *Production and operations management*. **18** (6), 653-671.
171. Chang, X., (2009). Impact on consumer price cheapness conception from involvement degree and price perception experience. *IEEE Computing, Communication, Control, and Management*. **7**, 387-390.
172. Helmberger, P. and Rosine, J., (1980). Measuring Producer's Surplus. *Southern Economic Journal*. **46** (4), 1175.
173. Mishan, E.J., (1968). What is Producer's Surplus?, *American Economic Review*. **58** (5), 1269
174. Fay, S., (2009). Competitive reasons for the Name- Your- Own- Price channel. *Marketing Letters*. **20** (3), 277-293.
175. Hinz, O. *et al.*, (2011). Price discrimination in e- commerce? An examination of dynamic pricing in name-your-own price markets. *Mis quarterly*. **35** (1), 81-98.
176. Fay, S., (2004). Partial-repeat-bidding in the nameyour-own-price channel. *Marketing Letters*. **23** (3), 407-418.
177. Karahan, N. G. and Abbas,A. E., (2012). Measuring Consumer Impatience and the Effects of Timing in Name-Your-Own-Price Channels, *in IEEE Transactions on Engineering Management*, vol. 59, no. 2, pp. 226-239.
178. Carare, O. and Rothkopf, M., (2005). Slow Dutch auctions. *Manag Sci*. **51** (3), 365–373.
179. Gal-Or, E., (2011). Pricing practices of resellers in the airline industry: posted price vs. Name-your-own-price models. *Journal of economics & management strategy*. **20** (1), 43-82.
180. Wilson, J.G. and Zhang, G., (2008). Optimal design of a name- your- own price channel. *Journal of Revenue and Pricing Management*. **7** (3), 281-290.
181. Berk, E. *et al.*, (2009). On pricing of perishable assets with menu costs. *International Journal of Production Economics*. **121** (2), 678-699.
182. Ishii, H. and Toyokazu N., (1996). Perishable inventory control with two types of customers and different selling prices under the warehouse capacity constraint. *International Journal of Production Economics*. **44** (1), 167-176.
183. Chun, Y.H., (2003). Optimal pricing and ordering policies for perishable commodities. *European Journal of Operational Research*. **144** (1), 68-82.
184. Becher, M., (2009). Simultaneous capacity and price control based on fuzzy controllers. *International Journal of Production Economics*. **121** (2), 365-382.
185. Nocke, V. and Peitz, M., (2007). A theory of clearance sales. *Economic Journal*. **117** (522), 964-990.
186. Loughlin, S., (2010). Eyjafjallajökull eruption, Iceland | April/May 2010. [online] Available from: <
http://www.bgs.ac.uk/research/volcanoes/icelandic_ash.html> [Accessed 12 January 2017].

187. The Economist, (2016). A fare shake. [online] Available from: <
<http://www.economist.com/news/finance-and-economics/21698656-jacking-up-prices-may-not-be-only-way-balance-supply-and-demand-taxis>> [Accessed 12 January 2017].
188. Bullen, J., (2017). Uber accused over 'rip-off' fares four times higher than normal during Tube strike, *EveningStandard*. Available from: <
<http://www.standard.co.uk/news/transport/uber-slammed-for-ripping-off-londoners-by-quadrupling-fares-amid-tube-strike-chaos-a3435891.html>> [Accessed 12 January 2017].
189. Spence, P., (2015). Why Uber is right to triple its fares during London's tube strike, *The Telegraph*, Available from: <
<http://www.telegraph.co.uk/finance/economics/11728546/Why-do-economists-support-Ubers-fare-rise-during-the-London-tube-strike.html>> [Accessed 12 January 2017].
190. Ramasastry, A., (2005). Web Sites Change Prices Based on Customers' Habits, *CNN.com*, [online] Available from:
 <<http://edition.cnn.com/2005/LAW/06/24/ramasastry.website.prices/>> [Accessed 11 January 2017].
191. Wang, T., Gal-Or, E., and Chatterjee, R., (2009). The Name-Your-Own-Price Channel in the Travel Industry: An Analytical Exploration. *Manage. Sci.* 55, 6, 968-979.
192. Micron Tech, (2004). Uprating Semiconductors for High-Temperature Applications, [online] Available from: <
<https://www.micron.com/~media/documents/products/technical-note/.../tn0018.pdf>> [Accessed 11 January 2017].
193. Zhang, K. *et al.*, (2015). Minimizing thermal variation across system components. In *Parallel and Distributed Processing Symposium (IPDPS)* .pp. 1139-1148.
194. Getov, V. *et al.*, (2015). Towards an application-specific thermal energy model of current processors. In *Proceedings of the 3rd International Workshop on Energy Efficient Supercomputing*. p-5.
195. Amazon, (2017). Amazon EC2 Spot Instances, [online] Available from:
 <<http://aws.amazon.com/ec2/spot-instances>>[Accessed 11 January 2017].
196. IBM, (2005). An architectural blueprint for autonomic computing, IBM Corp., Available from: < <http://ibm.co/1IP7TvG> > [Accessed 11 January 2017].
197. Bhola, S. *et al.*, (2006). Utility-aware resource allocation in an event processing system, International Conference on Autonomic Computing., pp. 55.
198. Guo, H., (2003). A bayesian approach for autonomic algorithm selection. In *Proceedings of the IJCAI workshop on AI and autonomic computing: developing a research agenda for self-managing computer systems*.
199. Aamodt, A. and Plaza, E., (1994). Case-based reasoning: Foundational issues, methodological variations, and system approaches. In *AI Communications*, volume 7:1, pages 39–59.
200. Watson, I. and Marir, F., (1994). Case-based reasoning: A review. *The Knowledge Engineering Review*, 9(4):327–354.

201. Cunningham, P., (1998). CBR: Strengths and weaknesses. *In Proceedings of 11th Int. Conf. on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pages 517–523.
202. Leake, D. B., (1996). Case-Based Reasoning, Experiences, Lessons and Future Directions. *AAAI Press / MIT Press*.
203. Watson, I., (1997). Applying Case-Based Reasoning: Techniques for Enterprise Systems. *Morgan Kaufmann Publishers*.
204. Spalazzi, L., (2001). A survey on case-based planning. *Artificial Intelligence Review*, 16:3–36.
205. Khan, M.J., Awais, M.M. and Shamail, S., (2008). Enabling self-configuration in autonomic systems using case-based reasoning with improved efficiency. *In Autonomic and Autonomous Systems, 2008. ICAS 2008*. pp. 112-117.
206. Khan, M.J., Awais, M.M. and Shamail, S., (2007). Achieving self-configuration capability in autonomic systems using case-based reasoning with a new similarity measure. *Advanced Intelligent Computing Theories and Applications. With Aspects of Contemporary Intelligent Computing Techniques*, pp.97-106.
207. Vickrey, W., (1961). Counter speculation, Auctions, and Competitive Sealed Tenders. *Journal of Finance*, 16(1), pp. 8-37.
208. Klemperer, P., (2004). Auctions: theory and practice. Available from: <<http://dx.doi.org/10.2139/ssrn.491563> > [Accessed 11 January 2017].
209. Lucking-Reiley, D., (1999). Using field experiments to test equivalence between auction formats: Magic on the Internet. *American Economic Review*, pp.1063-1080.
210. Borowicz, W., (2015). The next battle between Apple and Google is all about context. Available from: < https://thenextweb.com/insider/2015/06/09/the-next-battle-between-apple-and-google-will-be-all-about-context/#.tnw_882je42r> [Accessed 11 January 2017].
211. Joly, A., Maret, P., and Daigremont, J., (2009). Context-awareness, the missing block of social networking. *International Journal of Computer Science and Applications*, 4 (2).
212. Grubert, J. *et al.*, (2017). Towards pervasive augmented reality: Context-awareness in augmented reality. *IEEE transactions on visualization and computer graphics*, 23(6), pp.1706-1724.
213. Chen, X. *et al.*, (2015). Crime Prediction Using Twitter Sentiment and Weather. *in Proceedings of the Systems and Information Engineering Design Symposium*, pp.63-68.
214. Beresin, A.M., (2017). From Crime to Weather, Twitter Is Literally Predicting the Future. Available from: < <http://observer.com/2017/06/twitter-predicts-crime-better-than-police/>> [Accessed 11 July 2017].

APPENDIX A - CONTEXT ACQUIRE CLASS

```
package org.wmin.asanga.context;

import java.math.BigDecimal;
import java.math.MathContext;
import java.math.RoundingMode;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.HashMap;
import org.wmin.asanga.action.ConcurrentActionExecutor;
import org.wmin.asanga.util.Context;
import org.wmin.asanga.util.ContextValue;
import org.wmin.asanga.util.DBConnectionPool;
import org.wmin.asanga.util.GUIObjects;

/**
 *
 * @author Asanga
 */
public class ContextAcquire extends Thread {

    private Connection con;
    private final String SQL = "select service_name,value from
v$service_stats where stat_name='DB CPU' AND SERVICE_NAME IN
('dsssrv','oltpsrv')";
    //above sql overhead 3 microseconds
    private String CURRENT = "current", PREVIOUS = "previous";
    private HashMap<String, ContextValue> valueMap = new
HashMap<>();
    int i = 0;
    double prevOLTPValue = -1, preDSSValue = -1;
    int samples = 0;
    private GUIObjects guiObj;

    public ContextAcquire(GUIObjects guiObj) throws
SQLException {

        con =
DBConnectionPool.getDBConnection(DBConnectionPool.POOL);

        valueMap.put(CURRENT, new ContextValue());
        valueMap.put(PREVIOUS, new ContextValue());
        this.guiObj = guiObj;
        setName("Context Acquire");
    }

    @Override
    public void run() {

        ContextValue value = null;

        boolean twosamplecollected = false;

        while (true) {

            if (i == 0) {
```

```

        value = valueMap.get(PREVIOUS);
    } else {
        value = valueMap.get(CURRENT);
        twosamplecollected = true;
    }

    try {

        PreparedStatement pr =
con.prepareStatement(SQL);
        ResultSet rs = pr.executeQuery();

        while (rs.next()) {

            value.setValues(rs.getString(1),
rs.getLong(2));
        }

        rs.close();
        pr.close();

        long oltpValue = 0, dssValue = 0;

        if (i == 1) {

            ContextValue before =
valueMap.get(PREVIOUS);

            oltpValue = ((value.getOltpValue() -
before.getOltpValue())); // / (double) (5 * 1000000));
            dssValue = ((value.getDssValue() -
before.getDssValue())); // / (double) (5 * 1000000));
        }

        if (i == 0 && twosamplecollected) {

            ContextValue current =
valueMap.get(CURRENT);

            oltpValue = ((value.getOltpValue() -
current.getOltpValue())); // / (double) (5 * 1000000));
            dssValue = ((value.getDssValue() -
current.getDssValue())); // / (double) (5 * 1000000));
        }

        if (dssValue > 0 || oltpValue > 0) {
            // System.out.println("dss value
"+dssValue+ " oltp value"+oltpValue);
            BigDecimal hundred = new BigDecimal(100);
            MathContext mc = new MathContext(2,
RoundingMode.HALF_EVEN);
            MathContext mc1 = new MathContext(1,
RoundingMode.HALF_UP);
            BigDecimal oltpBD = new
BigDecimal(oltpValue);
            BigDecimal dssBD = new
BigDecimal(dssValue);

            BigDecimal oltpPct =
oltpBD.multiply(hundred).divide(oltpBD.add(dssBD),
mc).round(mc1);

```

```

//                                     BigDecimal dssPct =
dssBD.multiply(hundred).divide(oltpBD.add(dssBD),mc);

//                                     BigDecimal oltpPct2 =
oltpBD.divide(oltpBD.add(dssBD),mc).multiply(hundred).round(mc1);
//                                     BigDecimal
//                                     dssPct
//                                     =
hundred.subtract(oltpPct);

//                                     System.out.println("oltp " +
oltpPct.doubleValue() + " dss " + dssPct.doubleValue());
//                                     System.out.println("oltp " +
oltpPct.doubleValue() + " dss " + dssPct.doubleValue());
//                                     System.out.println("oltp " + oltpValue +
" dss " + dssValue);

guiObj.getOltpMeter().updateMeterValue(oltpPct.doubleValue());

guiObj.getDssMeter().updateMeterValue(dssPct.doubleValue());

boolean isContextChanged =
hasContextChanged(oltpPct.doubleValue(), dssPct.doubleValue());
System.out.println(isContextChanged);

if (isContextChanged) {

    guiObj.clearBarCharts();
    ConcurrentActionExecutor m = new
ConcurrentActionExecutor(new Context(oltpPct.intValue(),
dssPct.intValue()), guiObj);
    m.start();
    m.join();

}

System.out.println("back to sensing the
context change ");
guiObj.getStatus().updateProgress(0);
guiObj.getStatus().updateStatus("Sensing
context values");

} else {
    System.out.println("no load on system");
    guiObj.getStatus().updateStatus("No load
on system");

    guiObj.getDssMeter().updateMeterValue(0);
    guiObj.getOltpMeter().updateMeterValue(0);
    prevOLTPValue = -1;
    preDSSValue = -1;
}
i = ++i % 2;

Thread.sleep(5 * 1000);
} catch (InterruptedException ex) {
    ex.printStackTrace();
} catch (SQLException ex) {
    ex.printStackTrace();
}
}

```

```

    }
}
public boolean hasContextChanged(double oltp, double dss)
{
    //    return true;
    if (preDSSValue < 0) {
        prevOLTPValue = oltp;
        preDSSValue = dss;

        return true;
    }

    double ot = Math.abs(oltp - prevOLTPValue);
    double ol = Math.abs(dss - preDSSValue);

    if (ot >= 3 || ol >= 3) {
        ++samples;
    }

    if (samples == 3) {
        samples = 0;
        prevOLTPValue = oltp;
        preDSSValue = dss;
        return true;
    }

    return false;
}
}

```

APPENDIX B - OWL + SWRL DEFINITIONS

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
  xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:sqwrl="http://sqwrl.stanford.edu/ontologies/built-ins/3.4/sqwrl.owl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:swrl="http://www.w3.org/2003/11/swrl#"
  xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns="http://www.owl-ontologies.com/Ontology1394035041.owl#"
  xmlns:swrla="http://swrl.stanford.edu/ontologies/3.3/swrla.owl#"
  xml:base="http://www.owl-ontologies.com/Ontology1394035041.owl">
  <owl:Ontology rdf:about="">
    <owl:imports rdf:resource="http://swrl.stanford.edu/ontologies/3.3/swrla.owl"/>
    <owl:imports rdf:resource="http://sqwrl.stanford.edu/ontologies/built-ins/3.4/sqwrl.owl"/>
  </owl:Ontology>
  <owl:Class rdf:ID="OLTP">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:FunctionalProperty rdf:ID="oltpLoad"/>
        </owl:onProperty>
        <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
          >100</owl:hasValue>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:FunctionalProperty rdf:ID="dssLoad"/>
        </owl:onProperty>
        <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
          >0</owl:hasValue>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Class>
```

```

<owl:intersectionOf rdf:parseType="Collection">
  <owl:Class>
    <owl:complementOf>
      <owl:Class rdf:ID="DSS"/>
    </owl:complementOf>
  </owl:Class>
  <owl:Class>
    <owl:complementOf>
      <owl:Class rdf:ID="Mix"/>
    </owl:complementOf>
  </owl:Class>
</owl:intersectionOf>
</owl:Class>
</rdfs:subClassOf>
<rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
<owl:disjointWith>
  <owl:Class rdf:about="#DSS"/>
</owl:disjointWith>
<owl:disjointWith>
  <owl:Class rdf:about="#Mix"/>
</owl:disjointWith>
</owl:Class>
<owl:Class rdf:ID="Workload">
  <rdfs:subClassOf>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Mix"/>
        <owl:Class rdf:about="#DSS"/>
        <owl:Class rdf:about="#OLTP"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>
<owl:Class rdf:about="#DSS">
  <owl:disjointWith rdf:resource="#OLTP"/>
<owl:disjointWith>
  <owl:Class rdf:about="#Mix"/>
</owl:disjointWith>
<rdfs:subClassOf>

```

```

<owl:Restriction>
  <owl:onProperty>
    <owl:FunctionalProperty rdf:about="#oltpLoad"/>
  </owl:onProperty>
  <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
  >0</owl:hasValue>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    >100</owl:hasValue>
    <owl:onProperty>
      <owl:FunctionalProperty rdf:about="#dssLoad"/>
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Class>
    <owl:intersectionOf rdf:parseType="Collection">
      <owl:Class>
        <owl:complementOf rdf:resource="#OLTP"/>
      </owl:Class>
      <owl:Class>
        <owl:complementOf>
          <owl:Class rdf:about="#Mix"/>
        </owl:complementOf>
      </owl:Class>
    </owl:intersectionOf>
  </owl:Class>
</rdfs:subClassOf>
<rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>
<owl:Class rdf:about="#Mix">
  <owl:disjointWith rdf:resource="#DSS"/>
  <owl:disjointWith rdf:resource="#OLTP"/>
</rdfs:subClassOf>
<owl:Class>
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class>

```

```

        <owl:complementOf rdf:resource="#DSS"/>
    </owl:Class>
    <owl:Class>
        <owl:complementOf rdf:resource="#OLTP"/>
    </owl:Class>
    </owl:intersectionOf>
</owl:Class>
</rdfs:subClassOf>
<rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>
<owl:DatatypeProperty rdf:ID="optiIndexCostAdjValue">
    <rdfs:domain>
        <owl:Class>
            <owl:unionOf rdf:parseType="Collection">
                <owl:Class rdf:about="#DSS"/>
                <owl:Class rdf:about="#Mix"/>
                <owl:Class rdf:about="#OLTP"/>
                <owl:Class rdf:about="#Workload"/>
            </owl:unionOf>
        </owl:Class>
    </rdfs:domain>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
</owl:DatatypeProperty>
<owl:FunctionalProperty rdf:about="#dssLoad">
    <rdfs:domain>
        <owl:Class>
            <owl:unionOf rdf:parseType="Collection">
                <owl:Class rdf:about="#DSS"/>
                <owl:Class rdf:about="#Mix"/>
                <owl:Class rdf:about="#OLTP"/>
                <owl:Class rdf:about="#Workload"/>
            </owl:unionOf>
        </owl:Class>
    </rdfs:domain>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="hasName">
    <rdfs:domain>

```

```

<owl:Class>
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#DSS"/>
    <owl:Class rdf:about="#Mix"/>
    <owl:Class rdf:about="#OLTP"/>
    <owl:Class rdf:about="#Workload"/>
  </owl:unionOf>
</owl:Class>
</rdfs:domain>
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:about="#oltpLoad">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#DSS"/>
        <owl:Class rdf:about="#Mix"/>
        <owl:Class rdf:about="#OLTP"/>
        <owl:Class rdf:about="#Workload"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
</owl:FunctionalProperty>
<swrl:Imp rdf:ID="Rule-1">
  <swrl:head>
    <swrl:AtomList>
      <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
      <rdf:first>
        <swrl:ClassAtom>
          <swrl:argument1>
            <swrl:Variable rdf:ID="x"/>
          </swrl:argument1>
          <swrl:classPredicate rdf:resource="#OLTP"/>
        </swrl:ClassAtom>
      </rdf:first>
    </swrl:AtomList>
  </swrl:head>

```

```

<swrl:body>
  <swrl:AtomList>
    <rdf:rest>
      <swrl:AtomList>
        <rdf:rest>
          <swrl:AtomList>
            <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
            <rdf:first>
              <swrl:DatavaluedPropertyAtom>
                <swrl:argument1 rdf:resource="#x"/>
                <swrl:argument2 rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
                >100</swrl:argument2>
                <swrl:propertyPredicate rdf:resource="#oltpLoad"/>
              </swrl:DatavaluedPropertyAtom>
            </rdf:first>
          </swrl:AtomList>
        </rdf:rest>
      <rdf:first>
        <swrl:DatavaluedPropertyAtom>
          <swrl:propertyPredicate rdf:resource="#dssLoad"/>
          <swrl:argument1 rdf:resource="#x"/>
          <swrl:argument2 rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
          >0</swrl:argument2>
        </swrl:DatavaluedPropertyAtom>
      </rdf:first>
    </swrl:AtomList>
  </rdf:rest>
</rdf:rest>
<swrl:ClassAtom>
  <swrl:classPredicate rdf:resource="#Workload"/>
  <swrl:argument1 rdf:resource="#x"/>
</swrl:ClassAtom>
</rdf:rest>
</swrl:AtomList>
</swrl:body>
</swrl:Imp>
<swrl:Variable rdf:ID="oltp"/>
<swrl:Imp rdf:ID="Rule-2">
  <swrl:head>
    <swrl:AtomList>

```

```

<rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
<rdf:first>
  <swrl:ClassAtom>
    <swrl:argument1 rdf:resource="#x"/>
    <swrl:classPredicate rdf:resource="#DSS"/>
  </swrl:ClassAtom>
</rdf:first>
</swrl:AtomList>
</swrl:head>
<swrl:body>
  <swrl:AtomList>
    <rdf:first>
      <swrl:ClassAtom>
        <swrl:classPredicate rdf:resource="#Workload"/>
        <swrl:argument1 rdf:resource="#x"/>
      </swrl:ClassAtom>
    </rdf:first>
    <rdf:rest>
      <swrl:AtomList>
        <rdf:rest>
          <swrl:AtomList>
            <rdf:first>
              <swrl:DatavaluedPropertyAtom>
                <swrl:propertyPredicate rdf:resource="#oltpLoad"/>
                <swrl:argument1 rdf:resource="#x"/>
                <swrl:argument2 rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
                >0</swrl:argument2>
              </swrl:DatavaluedPropertyAtom>
            </rdf:first>
            <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
          </swrl:AtomList>
        </rdf:rest>
      </swrl:AtomList>
    </rdf:rest>
  </swrl:AtomList>
  <rdf:first>
    <swrl:DatavaluedPropertyAtom>
      <swrl:argument2 rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >100</swrl:argument2>
      <swrl:argument1 rdf:resource="#x"/>
      <swrl:propertyPredicate rdf:resource="#dssLoad"/>
    </swrl:DatavaluedPropertyAtom>
  </rdf:first>

```

```
</swrl:AtomList>
</rdf:rest>
</swrl:AtomList>
</swrl:body>
</swrl:Imp>
<swrl:Variable rdf:ID="xoltp"/>
<swrl:Variable rdf:ID="dss"/>
</rdf:RDF>
```

APPENDIX C - KNOWLEDGE BASE CLASS

```
package org.wmin.asanga.inference;

import
edu.stanford.smi.protege.exception.OntologyLoadException;
import edu.stanford.smi.protege.owl.ProtegeOWL;
import
edu.stanford.smi.protege.owl.inference.pellet.ProtegePelletOWLAP
IReasoner;
import
edu.stanford.smi.protege.owl.inference.protegeowl.ReasonerManage
r;
import
edu.stanford.smi.protege.owl.inference.reasoner.ProtegeReasoner;
import
edu.stanford.smi.protege.owl.inference.reasoner.exception.Proteg
eReasonerException;
import edu.stanford.smi.protege.owl.model.*;
import
edu.stanford.smi.protege.owl.swrl.bridge.BridgeFactory;
import
edu.stanford.smi.protege.owl.swrl.bridge.SWRLRuleEngineBridge;
import
edu.stanford.smi.protege.owl.swrl.bridge.exceptions.SWRLRuleEngi
neBridgeException;
import
edu.stanford.smi.protege.owl.swrl.exceptions.SWRLRuleEngineExcep
tion;
import edu.stanford.smi.protege.owl.swrl.model.SWRLFactory;
import edu.stanford.smi.protege.owl.swrl.model.SWRLImp;
import
edu.stanford.smi.protege.owl.swrl.parser.SWRLParseException;
import edu.stanford.smi.protege.owl.swrl.sqwrl.SQWRLResult;
import
edu.stanford.smi.protege.owl.swrl.sqwrl.exceptions.SQWRLExceptio
n;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
import org.wmin.asanga.action.Action;
import org.wmin.asanga.util.Context;
import org.wmin.asanga.util.ResponseObj;

/**
 *
 * @author Asanga
 */
public class KnowledgeBase {

    ArrayList<Context> mainList = new ArrayList<>();

    private boolean isPessimisticApproach = true;
    private final String CONTEXT_SQL = "select oltp_load,
dss_load, opti_ind_cost_adj,name from context";
```

```

        private final String SAVE_CONTEXT_SQL="insert into context
values(?,?,?,?)";
        private OWLModel owlModel;
        private SWRLRuleEngineBridge bridge;
        private SWRLFactory factory;
        private ProtegeReasoner reasoner;
        private final String URI = "file:/// " +
System.getProperty("user.dir").replace("\\", "/") +
"/workload3.owl";
        private OWLNamedClass workload;
        private RDFProperty hasName;
        private RDFProperty dssLoad;
        private RDFProperty oltpLoad;
        private RDFProperty optiIndCostAdj;

        public KnowledgeBase(Connection connection) throws
SQLException, OntologyLoadException, SWRLParseException,
SWRLRuleEngineBridgeException, SWRLRuleEngineException {

            loadContexts(connection);
            loadKnowledgeBase();
            createRuleEngine();
            createReasoner();

        }

        private void createReasoner() {

            reasoner =
ReasonerManager.getInstance().createProtegeReasoner(owlModel,
ProtegePelletOWLAPIReasoner.class);

        }

        private void createRuleEngine() throws SWRLParseException,
SWRLRuleEngineBridgeException, SWRLRuleEngineException {

            factory = new SWRLFactory(owlModel);
            bridge = BridgeFactory.createBridge("SWRLJessBridge",
owlModel);

        }

        private void loadKnowledgeBase() throws
OntologyLoadException {

            owlModel = ProtegeOWL.createJenaOWLModelFromURI(URI);

            workload = owlModel.getOWLNamedClass("Workload");
            hasName = owlModel.getRDFProperty("hasName");
            dssLoad = owlModel.getRDFProperty("dssLoad");
            oltpLoad = owlModel.getRDFProperty("oltpLoad");
            optiIndCostAdj =
owlModel.getRDFProperty("optiIndexCostAdjValue");

            for (Context c : mainList) {

                OWLIndividual w1 =
workload.createOWLIndividual(c.getName());
                w1.setPropertyValue(hasName, c.getName());
                w1.setPropertyValue(oltpLoad, c.getC1());
            }
        }
    }
}

```

```

        w1.setPropertyValue(dssLoad, c.getC2());
        w1.setPropertyValue(optiIndCostAdj,
c.getOptimizerIndexCostAdj());
    }
}

private void loadContexts(Connection connection) throws
SQLException, OntologyLoadException {
    PreparedStatement pr =
connection.prepareStatement(CONTEXT_SQL);
    ResultSet rs = pr.executeQuery();
    while (rs.next()) {
        Context c = new Context(rs.getInt(1),
rs.getInt(2), rs.getInt(3), rs.getString(4));
        mainList.add(c);
    }

    rs.close();
    pr.close();
    connection.close();
}

public Action getClosestAction(Context c) {

    double diff = -1;
    ArrayList<Context> diffList = new ArrayList<>();

    for (Context e : mainList) {

        double d = e.getDifference(c);
        // System.out.println("dddddd "+d+"
"+e.getThresholdValue() );
        if (diff < 0) { // first occurrence

            diff = d; //current minimum value
            diffList.add(e);
        } else {

            if (d < diff) { // a lower value found remove
all
                diffList.clear();

                diff = d;
                diffList.add(e);
            } else if (d == diff) { // same value found

                diffList.add(e);
            }
        }
    }

    if (diffList.size() > 1) {

```

```

diff = -1;
ArrayList<Context> c1List = new ArrayList<>();

for (Context e : diffList) {

    double c1diff = e.getC1Difference(c);

    if (diff < c1diff) {
        diff = c1diff;
        c1List.add(e);
    }
}

// check the high priority first
if (c1List.size() > 1) {

    diff = -1;
    ArrayList<Context> c2List = new ArrayList<>();

    for (Context e : c1List) {

        double c2diff = e.getC2Difference(c);

        if (diff < c2diff) {
            diff = c2diff;
            c2List.add(e);
        }
    }

    if (c2List.size() > 1) {

        diff = -1;
        ArrayList<Context> c3List = new
ArrayList<>();

        for (Context e : c2List) {

            double c3diff = e.getC2Difference(c);

            if (diff < c3diff) {
                diff = c3diff;
                c3List.add(e);
            }
        }

        if (c3List.size() > 1) {

            Context e = c3List.get(0);

            for (int i = 1; i < c3List.size() - 1;
i++) {

                Context k = c3List.get(i);

                if (isPessimisticApproach) {

                    e =
e.getPessimisticThresholdContext(k);
                } else {

```

```

        e
        =
e.getPessimisticThresholdContext(k);
    }

    }

    return new Action(e);

    } else {

        return new Action(c3List.get(0));
    }

    } else {

        return new Action(c2List.get(0));
    }

    } else {

        return new Action(c1List.get(0));
    }

    } else {

        return new Action(diffList.get(0));
    }

    }

    public ResponseObj checkContextKnown(Context c) throws
SQWRLEException, SWRLParseException, SWRLRuleEngineException {

        ResponseObj response = new ResponseObj();

        SWRLImp imp = factory.createImp("Query-1",
"Workload(?x)  $\wedge$  dssLoad(?x, " + c.getC2() + ")  $\wedge$  oltpLoad(?x, " +
c.getC1() + ")  $\rightarrow$  sqwrl:select(?x)");
        bridge.infer();
        SQWRLResult result = bridge.getSQWRLResult("Query-1");

        while (result.hasNext()) {

            response.setChecked(true);

            OWLIndividual ind =
owlModel.getOWLIndividual(result.getObjectValue("?x").toString());
;

            StringBuilder contextName = new
StringBuilder(ind.getBrowserText());
            int oltp = (Integer)
ind.getPropertyValue(oltpLoad);
            int dss = (Integer) ind.getPropertyValue(dssLoad);

```

```

        int opti = (Integer)
ind.getPropertyValue(optiIndCostAdj);

        Context existingContext = new Context(oltp, dss,
opti, contextName.toString());
        response.setContext(existingContext);
        result.next();
    }
    imp.delete();
    return response;
}

    public ResponseObj getWorkloadTypes(Context context)
throws SWRLRuleEngineException, ProtegeReasonerException {

        ResponseObj response = new ResponseObj();

        OWLIndividual w1 =
workload.createOWLIndividual("unknown-workload");
        w1.setPropertyValue(hasName, "unknown-workload");
        w1.setPropertyValue(oltpLoad, context.getC1());
        w1.setPropertyValue(dssLoad, context.getC2());

        bridge.infer();
        Collection<OWLClass> cls =
reasoner.getIndividualDirectTypes(w1);
        Iterator<OWLClass> itr = cls.iterator();

        while (itr.hasNext()) {

response.addWorkloadTypes(itr.next().getBrowserText());
        }

        w1.delete();
        return response;
    }

    public void updateKnowledgeBase(Context
unknownContext, Connection connection) throws
SWRLRuleEngineException, SQLException{

        OWLIndividual w1 =
workload.createOWLIndividual(unknownContext.getName());
        w1.setPropertyValue(hasName,
unknownContext.getName());
        w1.setPropertyValue(oltpLoad, unknownContext.getC1());
        w1.setPropertyValue(dssLoad, unknownContext.getC2());
        w1.setPropertyValue(optiIndCostAdj,
unknownContext.getOptimizerIndexCostAdj());

        bridge.infer();

        saveKnowledge(unknownContext, connection);
    }
}

```

```

        private void saveKnowledge (Context
unknownContext, Connection connection) throws SQLException{

        connection.setAutoCommit (false);
        PreparedStatement pr =
connection.prepareStatement (SAVE_CONTEXT_SQL);
        pr.setInt (1, unknownContext.getC1 () );
        pr.setInt (2, unknownContext.getC2 () );

pr.setInt (3, unknownContext.getOptimizerIndexCostAdj ());
        pr.setString (4, unknownContext.getName ());

        pr.execute ();
        connection.commit ();
        pr.close ();
        connection.close ();
    }
}

```