Florida International University
FIU Digital Commons

FIU Electronic Theses and Dissertations | University Graduate School

3-29-2011

Abstractions to Support Dynamic Adaptation of Communication Frameworks for User-Centric Communication

Andrew A. Allen
Florida International University, aalle004@fiu.edu

Follow this and additional works at: http://digitalcommons.fiu.edu/etd

Recommended Citation
Allen, Andrew A., "Abstractions to Support Dynamic Adaptation of Communication Frameworks for User-Centric Communication" (2011). *FIU Electronic Theses and Dissertations.* Paper 409.
http://digitalcommons.fiu.edu/etd/409

This work is brought to you for free and open access by the University Graduate School at FIU Digital Commons. It has been accepted for inclusion in FIU Electronic Theses and Dissertations by an authorized administrator of FIU Digital Commons. For more information, please contact dcc@fiu.edu.

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

ABSTRACTIONS TO SUPPORT DYNAMIC ADAPTATION OF COMMUNICATION FRAMEWORKS FOR USER-CENTRIC COMMUNICATION

A dissertation submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Andrew A. Allen

2011

To: Dean Amir Mirmiran
College of Engineering and Computing

This dissertation, written by Andrew A. Allen, and entitled Abstractions to Support Dynamic Adaptation of Communication Frameworks for User-Centric Communication, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

Kaushik Dutta

Ming Zhao

S. Masoud Sadjadi

Zhenyu Yang

Peter J. Clarke, Major Professor

Date of Defense: March 29, 2011

The dissertation of Andrew A. Allen is approved.

Dean Amir Mirmiran
College of Engineering and Computing

Interim Dean Kevin O'Shea
University Graduate School

Florida International University, 2011

© Copyright 2011 by Andrew A. Allen

All rights reserved.

DEDICATION

To Yvette Marie and Alyssa Ashley-Marie, thanks for your patience, your understanding and your unwavering support of my dream.

To Stan, Rohan, Vinnate, Eugenie, Marcia and Mrs. Nick, the encouragement and support you provided during the years of my study, gave me the conviction to carry on.

To my Mom, you have been a great source of inspiration and motivation to me. Your prayers have taken me through this long but successful journey and I thank you sincerely for that.

ACKNOWLEDGMENTS

I would like to express my heart-felt appreciation to all who contributed in many ways to the success of my PhD study and the completion of this thesis. I would especially like to thank the members of my committee, Drs. S. Masoud Sadjadi, Kaushik Dutta, Ming Zhao and Zhenyu Yang, for their insightful comments, and thorough questioning, which helped me focus more astutely on my research ideas in completing my thesis. I am also grateful for the perspectives, personal and professional, provided by Norman Pestaina, William Kraynek and Masoud Milani. To Professor Fábio Costa of the Federal University of Goiânia in Brazil and Professor Jean-Marc Jézéquel of the University of Rennes I in France, I am grateful for the opportunities to participate in research at your universities in the summers of 2009 and 2010 respectively. Many thanks also to the members of the CVM group and my colleagues in the lab, who have been very supportive of my research, very patient during my presentations, and always willing to offer suggestions for improvement.

Finally, I would like to express my deepest gratitude to my advisor, mentor and friend, Dr. Peter J. Clarke, for his guidance and continuous support of my PhD. study and research. I thank him for his patience, stimulating discussions, and encouragement during my preparation for, and writing of this thesis.

The work of this dissertation was supported in part by a FIU Graduate School Dissertation Year Fellowship.

ABSTRACT OF THE DISSERTATION

ABSTRACTIONS TO SUPPORT DYNAMIC ADAPTATION OF COMMUNICATION FRAMEWORKS FOR USER-CENTRIC COMMUNICATION

by

Andrew A. Allen

Florida International University, 2011

Miami, Florida

Professor Peter J. Clarke, Major Professor

The convergence of data, audio and video on IP networks is changing the way individuals, groups and organizations communicate. This diversity of communication media presents opportunities for creating synergistic collaborative communications. This form of collaborative communication is however not without its challenges. The increasing number of communication service providers coupled with a combinatorial mix of offered services, varying Quality-of-Service and oscillating pricing of services increases the complexity for the user to manage and maintain 'always best' priced or performance services. Consumers have to manually manage and adapt their communication in line with differences in services across devices, networks and media while ensuring that the usage remain consistent with their intended goals.

This dissertation proposes a novel user-centric approach to address this problem. The proposed approach aims to reduce the aforementioned complexity to the user by (1) providing high-level abstractions and a policy based methodology for automated selection of the communication services guided by high-level user policies and (2) providing services through the seamless integration of multiple communication service providers and providing an extensible framework to support the integration of multiple communication service providers. The approach was implemented in the Communication Virtual Machine (CVM), a model-driven technology for realizing

communication applications. The CVM includes the Network Communication Broker, the layer responsible for providing a network-independent API to the upper layers of CVM. The initial prototype for the NCB supported only a single communication framework which limited the number, quality and types of services available.

Experimental evaluation of the approach show the additional overhead of the approach is minimal compared to the individual communication services frameworks. Additionally the automated approach proposed out performed the individual communication services frameworks for cross framework switching.

TABLE OF CONTENTS

CHAPTER PAGE

1 INTRODUCTION . . . 1
1.1 Overview of Research Problem . . . 5
1.2 Dissertation Roadmap . . . 6

2 LITERATURE REVIEW . . . 8
2.1 Background . . . 8
2.1.1 Middleware . . . 8
2.1.2 Autonomic Computing . . . 10
2.1.3 User-Centric Communication . . . 14
2.1.4 Communication Virtual Machine . . . 14
2.2 Related Work . . . 17
2.2.1 Policy Languages . . . 18
2.2.2 Network-centric Autonomic Services . . . 20
2.2.3 Composition and Aggregation of Services . . . 21
2.2.4 Dynamic Adaptation . . . 23
2.2.5 User-Centric Communication . . . 26
2.2.6 Service Mashups . . . 27
2.3 Summary . . . 29

3 RESEARCH PROBLEM . . . 30
3.1 Motivation . . . 30
3.2 Problem Statement . . . 36
3.3 Objectives and Evaluation Criteria . . . 37
3.3.1 Near Minimalization of Services API. . . . 37
3.3.2 Intent Based Self-Configuration of Services. . . . 38
3.3.3 Integrated Services through Multiple Communication Frameworks . . . 39
3.4 Summary . . . 40

4 UCC SELF-CONFIGURING APPROACH . . . 41
4.1 Overview of Approach . . . 41
4.2 Feature Analysis for UCC Domain . . . 47
4.3 UCC Policy Definition . . . 50
4.4 UCC Policy Realization . . . 53
4.5 UCC Policy Application on the Illustrative Example . . . 60
4.6 Summary . . . 62

5 UCC FRAMEWORK . . . 63
5.1 Operational Overview . . . 63
5.2 High Level Design . . . 70
5.3 Detailed Design . . . 73
5.4 Implementation Details . . . 76

5.5 Summary 78

6 EVALUATION 79
6.1 Evaluation Goals 79
6.2 Experimental Setup 80
6.3 Experimental Set 1 - Two-way Video Conference: A Comparative Analysis 80
6.4 Experimental Set 2 - Analysis of Candidate Selection Algorithm 84
6.5 Experimental Set 3 - Audio to Audio-Video Conferencing Reconfiguration 86
6.6 Experimental Set 4 - N-way Audio Conference Configuration 87
6.7 Experimental Set 5 -Analysis of Autonomic Response Times 89
6.8 Discussion 90
6.9 Summary 91

7 CONCLUSION 92
7.1 Research Summary 92
7.2 Future Work 93

BIBLIOGRAPHY 97

VITA 104

LIST OF TABLES

TABLE PAGE

3.1 Sampling from Survey of frameworks [7]. 31

3.2 Sample of Per Minute International Calling Rate 32

3.3 Comparison of Per Minute Mobile Calling Rate in the UK 33

4.1 Example Policies for Interpretation. 55

5.1 Sampling from NCB JavaDocs. 77

5.2 Metrics for NCB and ACSTF. 78

LIST OF FIGURES

FIGURE PAGE

2.1 A Traditional View of Middleware 9

2.2 (a)AC Architecture (b)Design of a Autonomic Manager 11

2.3 Layered architecture of the CVM. 15

4.1 Simplified View of the Architecture of Current Approaches 41

4.2 Proposed Architecture 43

4.3 Self-Configuration steps for Frameworks 46

4.4 Survey of Communication Frameworks 48

4.5 Feature Diagram for Frameworks 50

4.6 XML representation for user-centric communication policy. 52

4.7 (a) IETF/DMTF Policy Architecture (b) Runtime Policy Evaluation. . . 56

5.1 NCB Control Flow Diagram. 63

5.2 NCB Autonomic Architecture. 71

5.3 CSM State Machine. 73

5.4 Reusable Autonomic Design. 74

5.5 NCB Detailed Design Diagram. 75

6.1 Analysis of Memory Usage for Two-way Video Conference. 81

6.2 Analysis of Data Transmission for Two-way Video Conference. 82

6.3 Analysis of Processor Utilization for Two-way Video Conference. 83

6.4 Average Times to Select Candidate Communication Framework. 85

6.5 Audio to Video Conferencing Set-up Times. 87

6.6 Analysis of Audio Configuration Times. 88

6.7 Breakdown of Autonomic Components vs Frameworks. 89

6.8 Analysis of Detection and Adaptation Time. 90

LIST OF ALGORITHMS

ALGORITHM PAGE

4.1 Algorithm to select applicable Policies. 58

4.2 Algorithm to produce candidate set of Communication Frameworks. . . . 59

5.1 Algorithm to Configure Communication Services. 66

5.2 Algorithm to Negotiate Communication Services. 68

5.3 WaitforReply Algorithm. 70

LIST OF ACRONYMS

3G	Third Generation
AAA	Authorization, Authentication and Accounting
API	Application Programming Interface
CORBA	Common Object Request Broker Architecture
CVM	Communication Virtual Machine
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IP	Internet Protocol
ITU-R	International Telecommunication Union, Radiocommunication Sector
LAN	Local Area Network
MDE	Model Driven Engineering
MDSD	Model Driven Software Development
NCB	Network Communication Broker
OASIS	Organization for the Advancement of Structured Information Standards
OS	Operating System
P2P	Peer-to-peer
QoS	Quality of Service
RTP	Real-time Transport Protocol
SE	Synthesis Engine
SIP	Session Initiation Protocol
TCP	Transmission Control Protocol
UCI	User-centric Interface
UCM	User-centric Middleware
UDP	User Datagram Protocol
VoIP	Voice over IP
WLAN	Wireless Local Area Network

WWW	World Wide Web
XCML	XML Communication Modeling Language
XMPP	Extensible Messaging and Presence Protocol

CHAPTER 1

INTRODUCTION

The pervasiveness of electronic devices, especially mobile electronic devices, coupled with the increasing resources available in these devices has aided the explosion of available electronic communication services[1]. Some electronic communication services, such as instant messaging (IM) and Voice over IP (VoIP), that were previously seen as trivial and restricted in businesses are now viewed as required services in the organization. Services previously limited to wired networks are transitioning to wireless devices, while bandwidth intensive services such as video conferencing are becoming more common place as the available bandwidth grows. This augmented set of electronic communication services provides a myriad of communication methods for the user.

In addition, the number of entities providing these communication services has increased dramatically. This can best be seen in the VoIP market which started offering services commercially in 1995 with the launch of Vocaltec's InternetPhone application. The range of communication services providers has since expanded, with services offered from the more traditional providers (such as Vonage, Packet8 and Lingo), non-facilities based providers (such as Skype and Google) and cable companies (such as Comcast and Time Warner). Coupled with this are the new players who have taken advantage of the mobile companies' investments in 4G networks to begin the push into the area of mobile VoIP.

While the electronic communication services have become more accessible and cheaper, paradoxically this has led to increases in the complexity for the user with each additional method of communication [41, 45]. This complexity, of the number of communication methods, can be viewed as the contributed effects of the combination

[1]For the purposes of this dissertation, electronic communication services are defined as services which provide users with the ability to send or receive messages (text, audio, video, and other data).

of numerous communication services providers and communication services offered. A user therefore is not only burdened with decisions on what services to use for communication, but also with which provider to use for the service.

With the convergence of electronic communication services on IP networks, opportunities to create elaborate collaborative communication applications are presented. Some communication service providers deliver a highly integrated product generally referred to as *unified communication* (e.g. IBM, Microsoft and Cisco), while others [26, 34, 56, 73] provide a variety of communication tools and communication services for creating synergistic collaborative communication applications. Unified communication attempts to integrate electronic communication media while providing a consistent unified user interface and user experience across multiple devices and media types. The provider-centric one-size-fits-all nature of this integration approach can result in more tools than the organization or individual user needs to accomplish their goals, potentially negating some of the complexity reduction.

In the case of the latter, products such as Skype [73], ooVoo [56] and GoogleTalk [26], which provide commercial-off-the-shelf (COTS) communication APIs, have been made available by their respective companies. Traditionally, development in complex domains such as healthcare, disaster management and other specialized real-time distributed collaborative communication was primarily a 'build, validate, and maintain software systems from scratch' approach [23]. These reusable communication components, that handle the low-level communication concerns, remove many of the tedious and error-prone aspects of creating and managing communication applications for the application developer. These communication APIs support the development of more sophisticated communication applications by third parties.

While both approaches provide converged communication services, users are however still burdened with decisions on what communication services and which communication service provider best serves their current communication needs. Differences

in which services are available [7] per communication services provider exist as well as differences in the QoS of the available services. Furthermore, software functionalities and features are added or enhanced by communication services providers paced by hardware evolution and user demands. The ease of accessibility has also resulted in the commoditization of the communication services, resulting in aggressive pricing models by communication services providers (this is discussed in more details in Chapter 3.1). Price sensitivity and quality-of-service sensitivity therefore become additional variables in the decision burden for the user.

The aforementioned concerns can be viewed in the context of the roles of the users of these collaborative applications. Users of these services have to manage and adapt their communication in line with differences in services across devices, networks and media. Other users (administrators) tasked with managing and, where possible, minimizing the costs associated with the use of the services as a resource have to ensure that the usage remains consistent with the business goals and the potential changes of the business model. The challenge is therefore how to reduce the effort needed by the user to manage their communication needs while ensuring that resulting service selection is near optimal with respect to the user's intent.

One proposed approach to this challenge is user-centric communication which aims to reduce the complexity and offer operating simplicity to users [41]. The Communication Virtual Machine (CVM) technology, proposed by Deng et al. [16], exemplifies this concept with a model-driven and domain specific approach to realizing communication services. Experts and novice users in domains such as healthcare, disaster management and scientific collaboration are presented with a simplified yet powerful way to quickly create and realize communication intensive collaboration. The user's communication needs are specified as a model written in the Communication Modeling Language (CML) [80]. The CML model is then executed on the CVM platform. The CVM is designed as a layered architecture and includes the Network Communica-

tion Broker (NCB) which is the layer responsible for providing a network-independent API to the upper layers of CVM. The initial prototype for the NCB supported only a single communication framework which limited the number, quality and types of services available.

This dissertation proposes an approach for seamless integration and self-configuration of multiple communication frameworks within the NCB. The approach includes a near minimal communication services abstraction and an extensible integration framework that is supported by a policy-driven approach for allocating and self-configuring communication resources. The near minimal abstraction provides a simpler API than that provided by the COTS communication frameworks. This is supported by the extensible integration framework that includes interfacing multiple communication frameworks and policy-based methodologies for selecting the most appropriate services as requested by the user's communication models. Models defined by users are executed on the CVM platform and transformed to the high-level abstract API calls to provide services that are guided by user-defined policies.

The contributions of this dissertation are:

1. Systematic development of high-level abstractions for the communication domain. This abstraction provides a near minimal API in comparison to other communication services APIs such as Skype, Smack and ooVoo. This abstraction provides an API that more closely reflects the communication services that a non-expert user may access.

2. A methodology for selection of services based on user intent to support automated reconfiguration of services and communication frameworks.

3. Development of an extensible integrated architecture for converging multiple communication services and providers. This architecture exposes the API and interfaces with existing communication frameworks as described in (1), and in-

corporates the mechanism to automatically reconfigure communication services as stated in (2).

4. Experimental evaluations that show the additional overhead of the approach is minimal compared to the individual communication frameworks, as well as the automated approach proposed out preforms the individual communication services frameworks for cross framework switching.

1.1 Overview of Research Problem

The research problem explores how to reduce the level of complexity when selecting low-level services provided by a cross-section of communication framework, that is transparent to the user. Specifically, *this study seeks solutions to enable users of collaborative communication to easily access more attractive price/performance options for certain services from amongst multiple service providers.*

Previous work in the areas of collaborative communication tend to (1) be limited in their ability to support reuse of their communication components [10,54,60,62]; (2) lack the support for automated reconfiguration at runtime of these communication components [20], (3) provide low-level interfaces defined for programmers [9, 28, 65, 67, 75] and (4) be limited in support for end user interactions [5, 9, 43].

This work proposes an approach that is domain specific in nature while leveraging component based development, user-centric paradigm, autonomic computing and Feature Oriented Domain Analysis (FODA) approach. The approach utilizes an understanding of the domain to argue that the *derived near minimal abstractions* are sufficient to represent the domain when used in conjunction with the proposed automation. For the purpose of this dissertation, we define a minimal API as the smallest set of ports of the API needed to provide basic services. FODA also informs the support needed for automated integration and reconfiguration.

The proposed solution requires:

1. Systematically developing high-level abstractions for the services provided by a cross-section of communication services frameworks.

2. Formulating a methodology:

   (a) for selection of services based on user intent.

   (b) to support automated reconfiguration of services and communication frameworks.

3. Designing an extensible framework that supports seamless integration of communication frameworks.

The motivation for the work, the research problem and evaluation criteria are elaborated on in Chapter 3.

1.2 Dissertation Roadmap

The rest of the dissertation is organized as follows: Chapter 2 presents a review of literature related to this work. Background concepts of key importance to this work are summarized and presented. Previous and current research are categorized and reviewed in relation to work proposed in this dissertation.

In Chapter 3, motivation for the work of this dissertation is further elaborated. A concise identification of the associated problems is provided as well as an overview of the proposed solution. The objectives and criteria for evaluating the success of the solution with respect to the identified issues are also presented in this chapter.

Chapter 4 is a presentation of the user-centric communication approach proposed in this dissertation. This includes a survey of existing communication services frameworks and domain analysis used to define the domain of user-centric communications via the FODA methodology. The definition of the user-centric communication policy structure is presented as well as the mechanisms to evaluate the policies.

The user-centric communication self-configuring framework is presented in Chapter 5. An overview of the framework is provided and key algorithms that support the self-configuration are presented. A high level view of the architectural approach along with some of the significant components are also highlighted with a detailed design and a discussion of the implementation.

Experimental evaluations are presented in Chapter 6. A prototype of the autonomic NCB was implemented to demonstrate the feasibility of the approach. The goals of the evaluation with respect to the evaluation criteria are presented along with an outline of the experimental setup. The results and a discussion of the results, including threats to the validity of the experiments, are also presented.

The dissertation concludes in Chapter 7. Discussion of future directions for this work is also discussed.

CHAPTER 2

LITERATURE REVIEW

In this chapter, background concepts of key importance to this work are summarized and presented. The chapter concludes with a review of previous and current research that relates to work proposed in this dissertation.

2.1 Background

In this section the concept of user-centric communication is presented, for the purposes of this dissertation the definition of communication is restricted to that of electronic communication. An overview of middleware, autonomic computing and the CVM technology which supports the model creation and realization of user-centric communication services is also presented.

2.1.1 Middleware

The notion of a middleware was born out of a perceived need to enable communication between entities in a heterogeneous distributed computing environment. Bernstein [6] defines middleware as 'a general-purpose service that sits between platforms and applications'. Middleware, however is expected to perform one or more of the following functions:

- Hiding distribution, i.e. the fact that an application is usually made up of many interconnected parts running in distributed locations;
- Hiding the heterogeneity of the various hardware components, operating systems and communication protocols;

- Providing uniform, standard, high-level interfaces to the application developers and integrators, so that applications can be easily composed, reused, ported, and made to interoperate;

- Supplying a set of common services to perform various general purpose functions, in order to avoid duplicating efforts and to facilitate collaboration between applications.

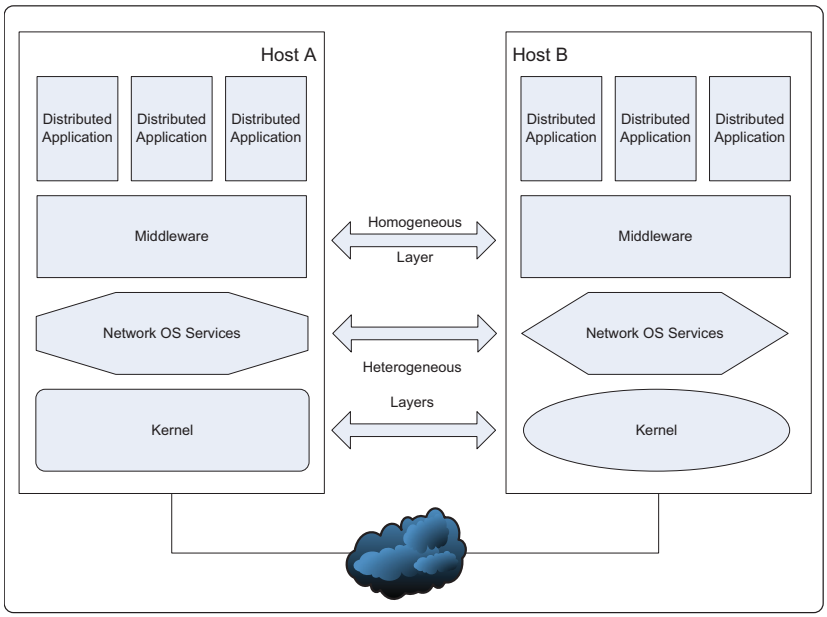


Figure 2.1: A Traditional View of Middleware

Tradionally, middleware is an additional layer between the operating system and the distributed application on every host that deals with communication issues (see Figure 2.1) and attempts to provide a homogeneous view of the world to the distributed application. Since then, a taxonomy [22] of middleware have been proposed that have expanded the scope beyond that of the traditional middleware. However, the primary role of middleware remains the same, that is to make application development easier. This is achieved by providing common programming abstractions, hiding low-level programming details and masking the heterogeneity and the distri-

bution of the underlying hardware operating systems. There still remain challenges for middleware designs, such as:

- the performance penalties due to the inherent indirection and interception techniques of middleware;
- the increased complexity of administration of the middleware as the interconnection and interdependence of applications increases [14];
- and the need for dynamic reconfigurability and adaptation to support the growing trend towards more ubiquitous computing utilizing user context, user mobility and user intent [71].

As the evolutions in software and hardware continue, designs for middleware will need to evolve to meet the needs of the latest technologies while providing vital links to legacy systems.

2.1.2 Autonomic Computing

The every increasing complexity of managing information technology systems and the every evolving nature of software prompted initiatives [37, 49, 81] towards automated solutions to these problems. IBM's proposal [37], named Autonomic computing (AC), portrayed a vision of computing systems that manage themselves according to high-level objectives. The paradigm seeks to alleviate the current burden for human operators tasked with integrating and managing highly complex systems through increased automation and goal specification.

The concept borrowed from the human autonomic nervous system (ANS), which regulates vital bodily functions without the need for conscious human involvement. Similar to the ANS, autonomic systems are expected to respond to changes in their environment according to goals previously set by an administrator. Self-management components are then responsible for maintaining the system in a state that complies

to set goals. Administrators specify system behavior as high-level policies which are transformed to low-level rules and tasks that can be automated. The self-management properties, often referred to as self-* properties, include:

- *Self-Configuration*: provides the means whereby a system can dynamically adapt to its changing environment.
- *Self-optimization*: monitors and fine tunes system to achieve optimal performance.
- *Self-protection*: detects and protects system from various attacks.
- *Self-healing*: identifies problems or potential problems then introduce solutions to ensure the system remains available.

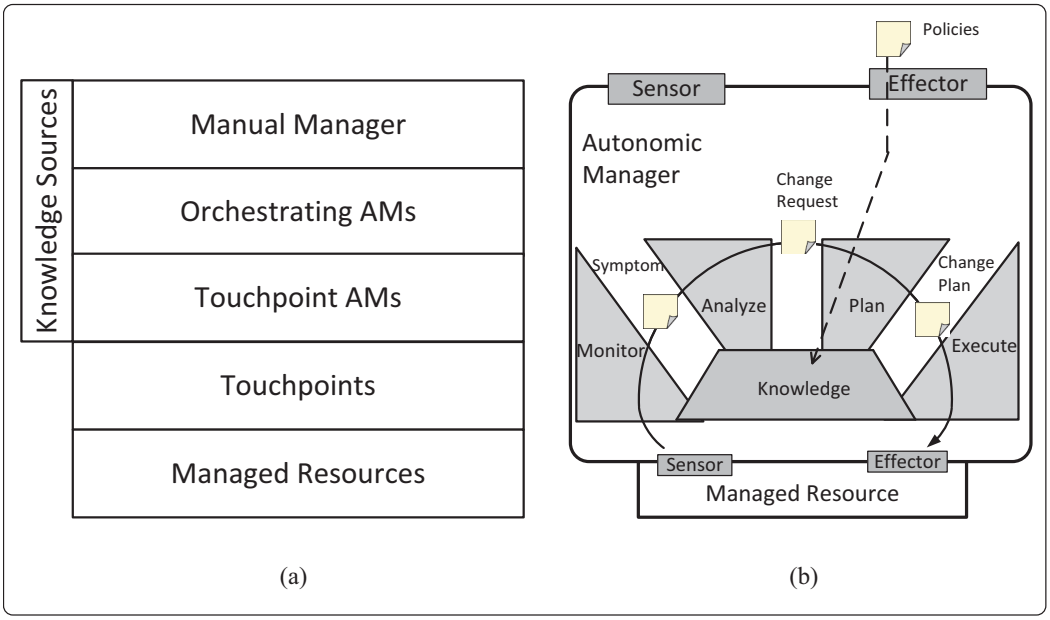


Figure 2.2: (a)AC Architecture (b)Design of a Autonomic Manager

AC's architectural blueprint [31] defines a common layered approach for developing self-managing systems. Figure 2.2(a) presents a view of the AC architecture. The horizontal layers include: managed resources, touchpoints, touchpoint autonomic managers, orchestrating autonomic managers, and a manual manager. A vertical

layer of knowledge sources, shown at the top-left of Figure 2.2(a), interacts with the top three horizontal layers. This facilitate the exchange and archival of management information.

The managed resource layer consists of the entities for which self-management services are being provided. The layer above the managed resources are manageability interfaces called touchpoints. Touchpoints implement the sensor and effector behaviors necessary to automate low-level management tasks [31, 37]. Sensors observe the state of managed resources, while effectors facilitate the implementation of runtime changes. A higher level of management is provided by autonomic managers (AMs). There are two categories of AMs - Touchpoint AMs, and Orchestrating AMs [31]. Touchpoint AMs work directly with managed resources through their touchpoints. Orchestrating AMs manage pools of resources or optimize the Touchpoint AMs for individual resources. The topmost layer is an implementation of a management console, called the manual manager, which facilitates the interaction and intervention of a human administrator. While the vertical layer of knowledge sources implements registries or repositories that may be used to extend the capabilities of AMs, and are directly accessible by the human administrator via the manual manager layer.

Autonomic software systems are characterized by closed loops of control. Figure 2.2(b) presents a conceptual view of the AM's control loop. Sensed changes to managed resources result in the invocation of a set of actions designed to maintain some desired state. Autonomic control loops are implemented as monitor, analyze, plan, and execute (MAPE) functions in AMs.

The MAPE functions of AMs collaborate to manage state changes to the resource as follows:

- Monitor: continuously polls the managed resource for this state information, and correlates it into symptoms for analysis.

- Analyze: determines if the current state is undesirable, and generates a change request to be passed to the plan function.
- Plan: specifies the set of actions needed to remedy the state condition of the managed resource, and formalizes them into a plan for execution.
- Execute: implements change plans on the managed resource through its effectors, for the purpose of acquiring some desired state.
- Knowledge: coordinates access to data shared among the MAPE functions.

High-level coordination of the MAPE functions is achieved through a hierarchical stacking of AMs. As shown at the top of Figure 2.2(b), the state of the MAPE functions and internal knowledge may also be observed and manipulated through sensors and effectors. Orchestrating AMs can therefore detect the generation of MAPE artifacts, and determine alternative courses of action. In addition, the self-management policies that guide the behavior of AMs may be dynamically updated through these top sensors and effectors.

The work in this dissertation is primarily focused on the self-configuration properties of autonomic computing. *Self-configuration* refers to the ability of a system to obtain its configuration parameters and initialize itself in order to provide the expected services. Self-configuration techniques can be viewed as either *initial configuration*, methods for specifying initial configuration requirements or *dynamic configuration*, methods for specifying reconfiguration based on given states [12]. For autonomic systems, self-configuration encompasses the initial configuration of a system as well as dynamic, reactive changes throughout its operational life. Policies are often used to guide these configuration transitions. A policy is a set of considerations designed to guide decisions on courses of action, as such policies are rules that define the choices in the behavior of a system [47].

2.1.3 User-Centric Communication

The convergence of various multimedia communications that includes voice, video and data presents many opportunities for enhancing communication between users. There are however challenges presented by this model of communication which can result in the user being less effective in their interaction with the communication. Interaction can be viewed as any mutual, reciprocal exchange between people, technologies and processes [41]. One such challenge is that with each new communication channel and application a new way of contacting others is introduced. This increases the complexity for the user of the multifaceted communication who is responsible for managing and adapting the communication to her immediate needs. Complexity can hinder rather than enhance communication [41] and research is under way in academia and industry [17, 41, 59] to find solutions to such challenges in multifaceted communication.

The user-centric approach [41, 77] is one such research direction. This solution aims to reduce the complexity and offer operating simplicity [41] to users of these communication services. To be user-centric requires knowledge of the actual 'context' of a user. A context defines a certain relationship of a human being to a particular number of objects of its communication space at a fixed moment of time [77]. The user-centric communications (UCC) approach is therefore about matching the communication resources with the individual's needs at a particular point in time in the context of the specific domain and adapting accordingly, thereby reducing the complexity to the user.

2.1.4 Communication Virtual Machine

Model-Driven Software Development (MDSD) is a software development methodology in which abstract models are created and systematically transformed to concrete implementations. France et al [24] points to the wide conceptual gap between the

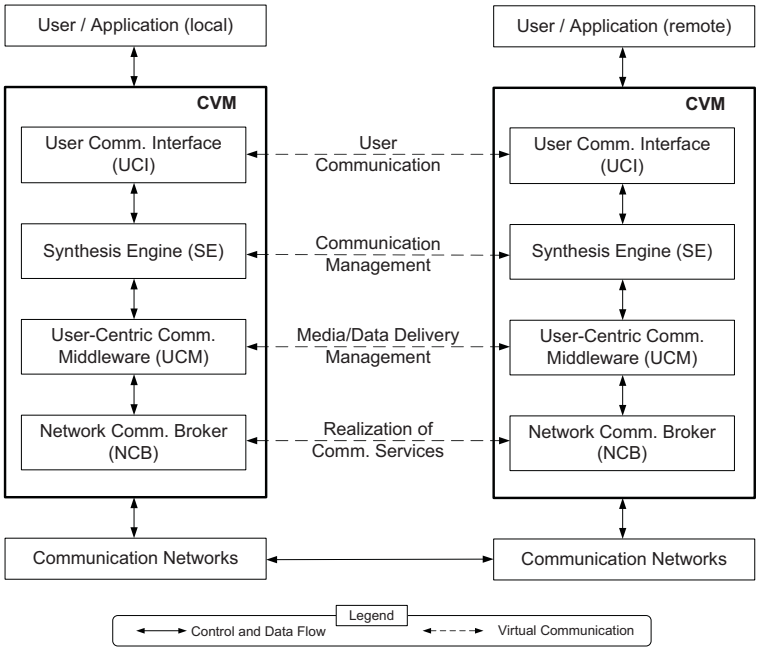


Figure 2.3: Layered architecture of the CVM.

problem and implementation domains as one challenge faced in the development of complex software. MDSD proposes to reduce this gap through the use of technology for systematic transformations of abstract models that consistently represents the problem down to the implementation. The models used would describe the complex system at multiple levels of abstractions with the models becoming a primary artifacts for development instead of just documentation as in traditional methodologies.

Deng et al. [16] developed the notion of the Communication Virtual Machine (CVM), a model-driven paradigm for specifying and realizing user-centric collaborative communication. Models are defined using a Communication Modeling Language[1] (CML). CML is a domain-specific modeling language used to create models for user-centric communication applications. Unlike many domain-specific languages that generate code before the application is executed, CML models are directly interpreted by CVM. CVM has a layered architecture and lies between the communication

[1]http://www.cis.fiu.edu/cml/

network and the user (or application). Figure 2.3 shows the layered architecture of the CVM. The key components of the CVM are:

User Communication Interface (UCI), provides a modeling environment for users to specify their communication requirements using CML. CML can be used to describe a user *communication schema* or *schema instance*, analogous to an object-oriented class and object. In order to realize a communication application two types of communication models are required: a *control schema* (or instance) that defines the configuration of the connections in a communication, and a *data schema* (or instance) that defines the media being transferred across a connection. The term media is used to refer to both streaming media e.g., video and data e.g., files. During a communication, schemas are shared with the parties in the connection.

Synthesis Engine (SE), implements a set of algorithms responsible for (1) automatically synthesizing schema instances into executable communication control scripts, (2) negotiating the schema instances with other participants in the communication, and (3) realizing media transfer between participants in the communication. The semantics to support the interpretation of CML models are based on changes to models (schemas) at runtime and defined using state machines [80]. As the state machines for schema negotiation and media transfer are executed the appropriate control scripts are generated for processing in the UCM.

User-centric Communication Middleware (UCM), executes the communication control script and manages the delivery of media to participants in the communication, independent of the underlying network configuration. Based on the control script received by the UCM, macros are loaded and executed either synchronously or asynchronously. Managing the delivery of media requires the UCM to store data in temporary locations, retrieve data from remote UCMs on-demand, enforce security policies

associated with media, among other tasks. Currently, only a skeletal UCM has been implemented in the CVM prototype.

Network Communication Broker (NCB), provides a network independent API to the UCM that masks the heterogeneity and complexities of the underlying network to support the realization of the communication services. The NCB interacts with the underlying communication frameworks to ensure that the request from the UCM are realized, including request negotiation between participants, delivery of media and the enforcement of low-level policies.

2.2 Related Work

The diversity of media capacity, service provider and user preference and the increasing demand for collaborative work requires multimedia middleware to provide seamless integration of various system components. Proposals such as the Next Generation Network (NGN) as proposed by the International Telecommunication Union Standardization Sector (ITU-T) [32] aims to provide the infrastructure necessary for meeting these challenges. The challenges of such middleware include: the independence of specific communication systems, the interoperability between different devices and media, the support for QoS-aware adaptation during run-time, and the common and easy interface to configure, navigate and monitor the environment.

Researchers have attempted to address some of these challenges by applying such concepts as autonomic computing, service composition, reflective middleware and user-centered paradigms. Previous work done in this area was surveyed and a summary of some of the more significant contributions is presented. Based on the review of the available literature in this area three key categories were identified, and each work was associated with the category that best represented their strengths. A review of work in the related areas of this dissertation is presented in this section.

2.2.1 Policy Languages

Policy-Based Management (PBM) separates the rules governing the behavior of a system from its functionality. The aim is to reduce the maintenance costs of information and communication systems while improving flexibility and runtime adaptability. Boutaba et al. [8] provide a historical perspective on policy-based management's evolution from the early applications as security models to today's elaborate frameworks, languages and tools. Boutaba et al. also notes the significant role PBM plays in various paradigms including SLA-driven, Business-driven, autonomous, adaptive, and self-* management.

Policy languages offer a common means of specifying rules for the behavior that can be mapped to some implemented control mechanisms. There is much diversity and very little agreement as to the structure and elements that defines a policy language but most approaches in this area either propose general purpose or specialized policy languages. There are general purpose policy languages such as Ponder [15], a declarative, object-oriented language for specifying security and management policy for distributed object systems. Ponder is described as a expressive, extensible and flexible policy language aimed at supporting the specification of the wide range of requirements needed for security and management of distributed systems. Another general purpose policy language is the eXtensible Access Control Markup Language (XACML) [58]. XACML is an OASIS standard for a policy language written in XML. XACML is used to describe general access control requirements, and additional has standard extension points for such things as defining new functions and data types. Queries can be formed to ask whether or not a given action should be allowed, with an interpretable response value of either Permit, Deny, Indeterminate or Not Applicable. Policy Management for Autonomic Computing (PMAC) [1], is another policy language for the management of aspects (quality of service (QoS), configuration) of large-scale distributed system. PMAC additionally includes integratable software

components to ease the development of software applications that use the policy management. The benefits ascribed to the use of these general purpose policy languages are:

- standard, easier interoperability with other systems that are using the same standard language.
- generic, can easily be used in many different domains and environments since it is not specific to anyone domain.
- and powerful, high numbers of extensions, hooks and functions that support many different ways to use the language.

This is however, countered by some [78] who argue that specialization results in semantically richer policy languages. Specialized policy languages such as WS-Policy4MASC [79], that specifies management policies for Web services and their compositions, are deeply based on their domain (in this case Web Services Description Language (WSDL) [50]). Others such as Web Services Policy Language (WSPL) [4], used for specifying web services policies, including application-specific service options, are specialized refinements of more general policy languages (WSPL is a subset of XACML).

The approach to policies in this dissertation is in the vein of domain specific specialization, focused more on providing a semantic rich language for specifying domain specific policies. The systematic approach to understand and define policies detailed later in this dissertation can be applied to other domains of interest. The author notes that while an explicit policy specification is described in this dissertation, other policy languages could be used to specify the artifacts of systematic domain analysis.

2.2.2 Network-centric Autonomic Services

There is a plethora of research in the area of autonomic communications. Dobson et al. [19] provide a comprehensive survey of the current state of autonomic communications research and identify significant emerging trends and techniques. Most of the research in the autonomic communications area focuses on applying autonomic capabilities to the network infrastructure including self-management of the topology, load, task, physical and logical characteristics of the networks. For example, Gu et al. describes the *Architecture of Network Autonomy* (ANA) system [28], where high-level abstractions are achieved through the interface of a policy-based management to provide autonomic multimedia communication with minimum human administration. Sousa et al. [75] described a task-aware system where users explicitly specified goals and quality attributes through a set of user interfaces. Based on that, the underlying self-managing system dynamically queried the task to trigger reconfiguration if needed. In Boutaba et al [9], they proposed SELFCON as an architecture for self-configuration of networks, in which configuration policies are defined for easing the management of network elements and maintenance of relationships among network components during network operation.

These works focus on the interacts of the network infrastructure and not on the end user communication applications or the communication middleware layers above the infrastructure. These techniques tend to be network-centric as opposed to user-centric communication, which deals with associating and adapting available communication resources with a user's communication needs in the context of a specific domain. To support the transition from network-centric to a user-centric level, further abstractions will be needed. The approach described in this dissertation leverages the user-centric paradigm to support greater involvement of the non-technical users and context in the decisions of reconfiguration.

2.2.3 Composition and Aggregation of Services

Academia and industry have investigated ways to systematically build and reuse services. In this subsection we discuss some of the representative efforts in this area. Composable communication software (CCS) utilizes component oriented concepts in the design and implementation decisions. Aggregated communication software (ACS) can be defined as collaborative communication applications that support protocols of multiple communication providers simultaneously but independently. For the purposes of this discussion, we narrow the scope of ACS to the support of reusable third party component-based designed application to present a differentiation to CCS.

Aggregated communication software (ACS) include integration of reusable frameworks (or clients) such as Skype, GoogleTalk and MSN messenger protocols. The main purpose of the ACS is to provide its user with a single user interface that can display and access the basic services (instant messaging, file sharing, audio conferencing) of these protocols. There are products such as Trillian [10], Qnext [62], Pidgin [60] and Eclipse Communication Framework (ECF) [20] that provide platforms to support multiple communication providers, while aggregating the accounts of the providers into one interface. ECF provides a set of high-level abstractions,which facilitates the reuse of high-level communication components and provides a cross-protocol API [20] that utilizes plug-ins from various communication providers.

Products like Trillian and Qnext while offering ways of adding new providers to their platforms, are proprietary and closed source with no way to reuse their communication components for building more extensive communication applications. Pidgin is open source but suffers the same limitations as Trillian and QNext. While ECF allows the reuse of high-level communications components in various application contexts, it however does not provide the self-configuration of the plug-ins from the various communication providers therefore lacking the flexibility for choosing the most cost effective communication framework on-the-fly.

Nicols et al. [54] uses a Commercial-off-the-shelf (COTS) approach to build an early internet application for collaborative multimedia communication. Nicol et al.'s prototype distributed multimedia application used ready-made component technology. Nicol approach supports the reuse of these COTS style communication components. However it does not address two issues, (1) the aggregation of multiple communication providers; and (2) a methodology for component selection or replacement.

On the other hand, Stiller et al. [76] uses a custom-built but dynamically reusable approach. Stiller et al presents Da CaPo++, a framework embracing both low-level communication subsystems and high-level APIs to support distributed multimedia applications based on a formal description of protocol graphs to configure, validate and execute different media flows at each participating peer system. Da CaPo++ configures end system protocols based on the requirements of the application using the middleware, local resources and the network prerequisites stated as QoS values. While DaCaPo++ packages the protocols as modules, the reliance on the core 'lift system' requires modules that are custom built. DaCaPo++ therefore does not facilitate reuse of COTS style communication frameworks.

Similar to most of the work in this subsection, the approach described in this dissertation utilizes a component-based software design approach. Unlike [76], the NCB support COTS as well as custom-built components. This work also differs from [10,54,60,62] as the NCB is viewed as a composite component and hence provides converged reusable services. While [20] provides reuse of it's services, the use of each service and provider pair needs to be explicitly stated. The NCB needs only the service to be specified for service reuse.

2.2.4 Dynamic Adaptation

As stated earlier, one of the expected functionalities of middleware is to *provide uniform, high-level interfaces to the application developers and integrators so that applications can be easily composed, reused, ported and made to interoperate.* For this functionality to support the continuously evolving nature of software, several issues will need to be addressed:

- How to provide middleware with the ability to integrate diverse underlying high-level services, both existing and new ones. This issue defines the extensibility of the middleware. The second part of this issue is how to support the integration under a common infrastructure. This sub issue defines how to abstract away the details of the underlying service implementations while focusing on the services attributes.

- How to provide middleware with the ability to adapt service provision according to user/organization requirements and context. As applications becoming more personalized, middleware must evolve to support a more user/organization centered paradigm. To support this middleware need the ability to adapt autonomously to reduce the management complexity to the user. Middleware need to understand context and requirements so it can infer needed behavior and adapt accordingly.

- How to provide middleware with the ability to support the earlier discussed issues in dynamic and uncertain environments.

There have been extensive research conducted in the area of self-adaptive systems [39, 43, 61, 66] with adaptive middleware solutions proposed to address one or more of the issues outlined earlier. However many challenges still exist in this area [13, 30, 48, 70]. Two such challenges are support in dynamic and uncertain environments (runtime variability) and dynamic decision making. Generally, runtime variability supports

postponing the binding decisions on certain components or code until runtime. This contrast with traditional variability management techniques where the binding occurs at design, implementation or deployment phases.

Transparent shaping [65, 67, 68, 74] was proposed as a way to introduce adaptive behavior into legacy systems, with the potential benefits of extending the life of the application beyond its original design. Transparent shaping supports the design and development of new software artifacts from existing applications without the need to modify the existing application's source code. Aspect oriented programming is used to introduce new functionality at development time with reflective techniques used to support reconfiguration at runtime. The Adaptive CORBA Template, proposed by Sadjadi et al. [65], is a transparent shaping approach to runtime adaptation of CORBA applications. ACT's generic interceptor is registered at the ORB of the CORBA application at startup time. Request/reply is then redirected through the generic interceptor to the ACT Core which supports the runtime registration/unregistration of dynamic interceptors. The TRAP family [68, 74] provide language-based approaches for transparent shaping. A set of classes are targeted at compile time and adapt-ready programs generated with hooks to support reflective use. While transparent shaping supports runtime adaption, a priori knowledge and targeting of explicit low-level objects and classes is required to support the runtime adaption. This requires the developer to have an understanding of the low-level objects (in the case where transparent shaping is supported in the middleware) or classes (in the case of transparent shaping of an application).

More recently, the explicit use of models to support runtime adaptation have been proposed. Morin et al. [51] seek to address the challenge of runtime variability with the use of an explicit base model at runtime and the use of aspect oriented modeling techniques to weave new aspects into the existing base model. Configurations are generated and used at runtime to support reconfiguration. The aforementioned

approach provides strong support for validation of configuration, however it does not address issues of runtime element changes that are outside of the explicit model used at runtime. While runtime element changes can be detected, use of new elements and functionalities can only be done after new specific associated aspects are woven in. Component based approaches are one way of providing dynamic integration beyond design and deployment time.

Bencomo et al. [5] proposes a component based approach that utilizes reflection to address the challenges of runtime variability. Reflective mechanisms are used to inspect the system and event-action policies to reconfigure the system. However, the action element of the policy state a specific architectural configuration. This work requires the design time creation of a model as well as valid variants described in the policies to support the runtime adaptation. Additionally, this work requires custom components to support aspect weaving.

Another approach for dynamic reconfiguration of distributed multimedia middleware and applications was proposed by Provensi et al. in [61], where models are used at runtime as a means to structure the reflection mechanisms used for adaptation. The authors propose the use of an autonomic loop and policies for the monitoring of applications and their environment and for directing the reflection mechanisms. This work however, takes a more generic middleware approach with no focus on the user-centric aspects in supporting the dynamic adaptations. Additionally, this work lacks a comprehensive solution for policy definition and interpretation.

Dynamic adaption approaches described in the preceding works are aimed at specialists with strong background in data communication and middleware application programming. An a priori understanding of the set of possible valid configuration (exhaustive or bounded) can be useful for such specialist but will possibly be useless for non-technical users as they they may not comprehend or care since it is outside their domain of expertise. Runtime generation of configurations, on the other hand,

can be useful to both technical and non-technical users as it provides better support for high variability and unknown environments. The work in this dissertation aims to provide similar support for the non-technical users of communication services.

2.2.5 User-Centric Communication

There has been an upsurge in research in the area of user-centric communication service creation and use. In 2005 Lasserre and Kan [41] reported on the fragmented customer and employee experience when people try to contact each other using the myriad of available technologies. As a result of the study Alcatel developed an enterprise communication architecture with the following key principles: *user-centric* - focusing on the benefits to the user and offering operating simplicity to mask the complexity of the underlying technology; *openness* - is the basis for choice, which extends from the users choice of device, to product components, to the deployment model; and *unified* - unifying voice and data, fixed and wireless, and applications, enables services to be delivered seamlessly across different media.

Location-awareness is also leveraged to support user-centric communication. Lewis et al. in [42] present an approach for the management of user centric adaptive services for adaptive service composition and policy based management of adaptive system behavior. Through context-awareness, a centralized system is used to compose the services needed based on the user location and policies specified. The managed services are however limited to resource allocation services, such as printer assignments.

Rasche et al. [63] presents a framework for dynamic component configuration and reconfiguration based on Microsoft.NET. The approach utilizes an XML-based configuration description language (CDL) which is a specialized architecture description language (ADL). A CDL instance is designed and used to instantiate the component, or generate reconfiguration actions such as the addition, removal or modification of a

component, if necessary. However, their reconfiguration algorithm is inflexible, once chosen, it remains fixed at runtime.

In the realm of user-centric communication policies, some initial work has been proposed. Gorton [27] shows how users could set up their personal preferences and policies for controlling their communication services. For instance, a personal policy could define key controls of service delivery including the service access control, post-paid spending limits, device or network access impact on services. While their policies allow control over the who, what, where, when and how much of service delivery, it does not provide guidance on the configuration of resources to best support users' services. This work also lacks a formal representation of the policy.

The initial NCB was developed by Zhang et al. [83] and the focus was to provide a higher level of abstraction that encapsulated the complexity of the underlying network. The NCB also provided some self-management capabilities such as dynamic adaptation in response to changes in the network conditions. Sadjadi et al. [69] provided more details on the design and implementation for the initial version of the NCB focusing on how the internal network addresses were mapped to the globally accessible network addresses. Sadjadi et al. [69] described how the Network Address Translation (NAT) method and Simple Traversal of User Datagram Protocol through NATs (STUN) were used during self-configuration to realize end-to-end communication. As previously stated, the new implementation of the NCB uses the initial version of the NCB referred to as NCBNative.

2.2.6 Service Mashups

More recently there has been a major initiative to mix traditional telecommunication services and web capabilities to provide so called *web + telco mash-ups*. Sienel et al. [72] describes the OPUCE service architecture that supports the interoperation of telecommunications (telco) and information technology (IT) applications. The

OPUCE platform integrates the following: a service creation environment, a service lifecycle manager and a service execution environment (SEE). The SEE seamlessly integrates the orchestration of services running on top of several technologies, including Java 2 Enterprise Edition (J2EE), Java API for integrated networks service logic execution environment (JAIN SLEE), SIP servlets [64], .NET or client-side widgets. The OSA/Parlay by Open API Solutions [57] is a suite of open, standard, APIs designed to facilitate easier access to core network capabilities from outside of the network. The OSA/Paraly APIs support the creation of telecommunications services by developers. while these approaches provide easy-of-use benefits, they tend to be more heavyweight solutions, requiring server side as well as client side development. Additionally the level of abstraction would not be ideal for domain experts and end users with little programming background. Composition of these services are manually done through some service creation environment, requiring the users of these tools to have a high level of expertise in the area of telecommunication.

As ubiquitous computer trends upwards, O'Droma et al. [55] put forward a vision for a Consumer Based Model (CBM) to replace the Subscriber Based Model (SBM) in 4th Generation wireless telephony. This vision is seen as one of the grand goals for research in such areas as vertical handover technologies from mobile wireless networks to unlicensed spectrum access points such as WiFi hotspots. The concept proposes an 'always best connected' (ABC) approach where handover is not only based on a user's location but on other factors such as price and performance. Some work has begun towards realizing the vision through the network-independent infrastructure proposal included in Next Generation Network (NGN) [32] and an Always-Best-Served Music Distribution [25] prototype.

The work proposed in this dissertation has similar goals but applied at higher level. The full vision of ABC is currently limited by the lack of a network-independent in-

frastructure, this work however does not have that limitation as the services described in this dissertation are provided on the neutral virtual infrastructure of the Internet.

2.3 Summary

Extensive research has been done with respect to automating reconfiguration efforts. While many attempted to provide a general solution to the problem, some have seen the benefits of bounding the problem within a specific domain. Much of this work has been applied at the network layer with significant success, however efforts aimed at middleware and application layers tend to be general purpose. Previous work, described in this chapter, in the area of collaborative communication tend to (1) be limited in their ability to support reuse of their communication components; (2) lack the support for automated reconfiguration at runtime of these communication components, (3) provide low-level interfaces defined for programmers and (4) be limited in support for end user interactions.

In this chapter, the AC concept and it's grand challenge for reducing complexity to users of IT through self-managed automation was presented. The CVM technology, a user-centric model-driven paradigm aimed at reducing complexity to user in the collaborative communication domain, was also presented. Related works were reviewed and several deficiencies found which would limit the use of these works as solutions to the problems outlined in this dissertation. These include limitations in their ability to support reuse of their communication components for building more extensive communication applications, lack of automated reconfiguration at runtime of these communication components and limited support for end user interactions. While the initial NCB provided a higher level of abstraction, it however lacked a framework for systematically integrating additional low-level communication services and mechanisms to support the dynamic selection and reconfiguration of services by users.

CHAPTER 3

RESEARCH PROBLEM

In this chapter we present motivation for the work in this dissertation. We identify concisely the associated problems in the problem statement and provide an overview of the proposed solution. The objectives and criteria for evaluating the success of the solution with respect to the identified issues are also presented in this chapter.

3.1 Motivation

The use of traditional approaches for the design and implementation of communication-intensive software is expensive and error-prone. Development in more complex domains such as healthcare, disaster management and other specialized real-time distributed collaborative communication was primarily a 'build, validate, and maintain software systems from scratch' approach [23]. The popularity of IP based communication applications has led to the creation of 'commercial-off-the-shelf' (COTS) reusable communication components that handle the low-level communication concerns, removing many of the tedious and error-prone aspects of creating and managing communication applications for the application developer.

However, not all COTS reusable communication frameworks provide the same services or same quality-of-service. Table 3.1 shows a sampling from a survey (done as part of a feature-oriented domain analysis) of such communication frameworks [7] where a 1 indicates this feature is present while a 0 indicates an absence. The term 'conference' is used in the table to indicate multi-party support involving three or more users. As can be seen from the table, NCBNative [1](a framework developed in the CVM project) is the only framework that supports video conferencing involving more than two users. NCBNative is however an experimental framework which currently

[1]The initial version of the NCB is referred to as NCBNative

Features	NCBNative	Skype	JML	Gtalk	Android	Blackberry OS
Core Features						
Chat (one-to-one)	1	1	1	1	1	1
Chat (Conference)	1	1	1	1	1	1
Audio (one-to-one)	1	1	0	1	1	1
Audio (Conference)	1	1*	0	0	0	1
Video (one-to-one)	1	1	0	0	1	1
Video (Conference)	1	0	0	0	0	0
File Transfer	1	1	1	1	1	1
Contact List	1	1	1	1	1	1
API	Java	Java	Java	C++	Java	(HTTP) Java
Additional Features						
Emoticon	1	1	1	1	0	1
Online Status	1	1	1	1	1	1
Avatar Images	0	1	1	1	1	1
PC to Phone	0	$	0	$	$	$
Phone to PC	0	$	0	0	$	$
Voicemail	0	1	0	1	1	1

Table 3.1: Sampling from Survey of frameworks [7].

lacks the robustness and efficiencies of the other commercially developed frameworks. Coupled with this is the pace at which new software features are being added and functionalities enhanced as the hardware support improves. Since the time the survey that produced Table 3.1 was done, Andriod variants now include audio conferencing capabilities and more recently a simple video chat application was built using Flex 4[2] and deployed on AIR for Android[3] that allows multiple users in a video chat room. For application designers and developers who reuse the COTS communication services in products for more complex domain, the ability to easily switch communication components as well as integrate new services and service providers becomes a key concern to keep the product current. For feature-sensitive and quality-sensitive user of these products, keeping abreast of the latest technological development by all the available providers of services may not be a feasible option. Additionally this would take the focus away from their domain of expertise as they would instead need to focus on maintaining the tool.

[2]http://www.adobe.com/products/flex/flex_framework/
[3]http://www.adobe.com/products/air/

Country	Skype		ooVoo		Gvoice		VoIPVoIP	
	LandLine	Mobile	LandLine	Mobile	LandLine	Mobile	LandLine	Mobile
USA	$0.023	$0.023	$0.020	$0.020	free	free	$0.019	$0.019
Canada	$0.023	$0.023	$0.020	$0.020	free	free	$0.019	$0.019
Germany	$0.023	$0.253	$0.024	$0.290	$0.020	$0.230	$0.019	$0.317
Italy	$0.023	$0.308	$0.028	$0.390	$0.020	$0.300	$0.019	$0.450
UK	$0.023	$0.259	$0.024	$0.330	$0.020	$0.180	$0.019	$0.430
Isreal	$0.023	$0.265	$0.024	$0.156	$0.020	$0.230	$0.019	$0.207
France	$0.023	$0.209	$0.030	$0.230	$0.020	$0.150	$0.019	$0.129
Saudia Arabia	$0.250	$0.268	$0.270	$0.270	$0.110	$0.190	$0.108	$0.146
Egypt	$0.188	$0.159	$0.200	$0.200	$0.130	$0.110	$0.119	$0.119
Iraq	$0.390	$0.390	$0.370	$0.370	$0.090	$0.160	$0.056	$0.139
Lebanon	$0.126	$0.250	$0.140	$0.250	$0.100	$0.190	$0.115	$0.187
UAE	$0.275	$0.275	$0.280	$0.280	$0.190	$0.190	$0.152	$0.156
Morroco	$0.259	$0.355	$0.270	$0.350	$0.090	$0.290	$0.046	$0.345
Kuwait	$0.132	$0.132	$0.160	$0.160	$0.090	$0.120	$0.071	$0.121
Russia	$0.052	$0.089	$0.060	$0.060	$0.040	$0.080	$0.048	$0.062
China	$0.023	$0.023	$0.020	$0.030	$0.020	$0.020	$0.019	$0.019
Brazil	$0.058	$0.221	$0.050	$0.250	$0.040	$0.150	$0.040	$0.156
India	$0.092	$0.092	$0.100	$0.100	$0.060	$0.060	$0.019	$0.024

Table 3.2: Sample of Per Minute International Calling Rate

A further consideration is the economic cost or potential savings from the diversity of electronic communication service providers. As each service matures, competitive premiums are applied to the communication service by competing providers as is the case with Voice-over-IP (VoIP) telephone services. Businesses and individuals who wish to take full advantage of these prices will have to manually monitor the oscillating pricing of competing vendors. A simple example would be per minute cost for VoIP service between two countries or cities. Table 3.2 contains a small sampling of per minute international rates[4] for several countries. The lowest per minute rate for calling a landline or mobile service in each country is underlined in the table. Calling parties in the USA and Canada are free using GoogleVoice[5]. Calling a landline in Israel is cheapest using VoIPVoIP[6] while calls to mobiles in Israel are cheapest using ooVoo[7]. The use of GoogleVoice for calls to landline parties in Lebanon is cheapest while VoIPVoIP is lowest for mobile parties in Lebanon. Even

[4]Pricing information retrieved on February 15th 2011.
[5]https://www.google.com/voice/b/0/rates
[6]http://rates.voipvoip.com/
[7]http://www.oovoo.com/

Country	Country Area Code	Rate/Min
United Kingdom - Mobile - H3G	447400, 447401, 447402, 447403, 447411, 447412, 447413, 447414, 447575, 447576, 447577, 447578	$0.17
United Kingdom - Mobile - O2	44754, 44756	$0.12
United Kingdom - Mobile - O2	447404, 447405	$0.44
United Kingdom - Mobile - Orange	447409, 447410, 447529, 447556, 447579, 447580, 447581, 447582, 447583, 4475320, 4475321, 4475322, 4475323, 4475324	$0.12
United Kingdom - Mobile - T - Mobile	447538, 447539, 447550, 447572, 447573, 447574, 447942, 447943, 447945, 447948	$0.11
United Kingdom - Mobile - T - Mobile	4478921	$0.12
United Kingdom - Mobile - Vodafone	447407, 447551, 447553, 447554, 447555, 447557, 447570, 447584, 447585, 447586, 447587, 4475374	$0.12
United Kingdom - Mobile - Vodafone	447552	$0.15
...	...	...
...	...	...
...	...	...

Table 3.3: Comparison of Per Minute Mobile Calling Rate in the UK

within a single provider, the diversity in pricing can be an issue as well. In Table 3.3 a sampling of the comparative mobile rates for the United Kingdom is presented. The mobile service provider O2 has been highlighted in the table to show the possible disparity in rates as provided by VoIPVoIP where an almost four fold difference exist depending on the area code of the called party. A decision on a specific service provider made today based on price and QoS may not be the best fit tomorrow or next month. This is further complicated by the complexity and size of the APIs for the communication services frameworks. As an example, the skype API has close to two hundred commands while smack API has over one hundred commands. A static selection or manual monitoring and changing of service provider may not be a feasible option, especially with the large combinatorial set of possible terminating locations, pricing and QoS demands.

To further highlight the potential issues involved in such collaborations, one scenario from the collaborators at Miami Children's Hospital is presented. For illustrative purposes the selected scenario has been condensed. It however, still highlights the applicable nontrivial concerns for collaborative communication in domains such as healthcare.

In the scenario, Dr. Burke has a weekly conference with Drs. Sanchez and Monteiro to discuss patients' status. All three doctors are in their respective offices during these meetings and have had their technical staff configure appropriate video and audio conferencing tools to support the meetings. However during this particular meeting they need to get additional information from an incoming patient's primary care physician who is at another clinic, Dr. Clarke, via a land-line connection. Based on the hospital's directive to ensure cost containment at all levels, they hope to use the lowest cost services for the four party conference. The technical department has the following:

- Specialized IP based Audio-Video conferencing tools with no associated per minute cost but no support for landline or mobile telephones.
- IP based Peer-to-Peer Audio-Video conferencing tools with variable per minute cost to landline or mobile phones and limitations on party size.
- VoIP SIP Server with bundled minutes that serve the entire hospital (no associated per minute cost for local calls, variable per minute cost otherwise) and no limitation on party size.

While there are many requirements that could be extracted from the scenario, based on the focus of this dissertation only the explicit requirements highlighted in the scenario will be selected. Three requirements that need to be satisfied have been identified in the scenario:

- Requirement 1. Initial provisioning of a three way audio-video conference
- Requirement 2. Transitioning from a three way audio-video conference to a four way audio conference
- Requirement 3. Complying with the hospital's directive for minimizing cost

Technology currently exist that can support such scenarios, however such support usually comes at the cost of over provisioning of services, customized software and the need for specialized support staff. The scenario would be further complicated by the need to share the patient's record, which would require decisions on how much of the record to share, what media types constitute this record and what are the suitable delivery methods for all or part of the record. Coordination of these meetings would therefore involve not only the doctors, but also the technical staff who would need to know beforehand all the possible media and members that could be involved in the conference. Additionally, compliance with requirement 3 would require a priori knowledge of Dr. Clarke's area code and current per minute rates for calling his area code for the options that support landline and mobile calling. Any solution in this scenario would therefore need to address the following questions:

- How can the most appropriate provisioning of service be guaranteed?
- How can transitions of service and service providers be appropriately supported?
- How can compliance with the hospital's directive be ensured?

The previous questions themselves raise further questions:

1. What defines 'most appropriate' for a specific user in a specific context in a non-complex fashion?
2. Can all service transitions be handled the same?
3. How can 'compliance directives' be stated in a non-complex fashion?
4. How can these directives be interpreted and enforced?

For the purposes of this dissertation, we will define *appropriate services* as either best performance or best price services. The fundamental problem highlighted by this scenario is therefore, **how can a user dynamically access more attractive price/performance options for certain services from service providers**.

3.2 Problem Statement

The research problem being explored crosscuts the areas of software engineering, collaborative communication and adaptive middleware. This dissertation investigates the problem of how to reduce the level of complexity when selecting low-level services provided by a cross-section of communication framework, that is transparent to the user. Specifically, *this study seeks solutions to enable users of collaborative communication to easily access more attractive price/performance options for certain services from amongst multiple service providers*. The study, framed in a broader scope, focuses on the investigation of an approach to address the following issues within the context of the communication services domain: 1) how to provide middleware with the ability to integrate diverse underlying high-level services, both existing and new ones under a common infrastructure while abstracting away the details of the underlying service implementations; 2) how to provide middleware with the ability to adapt service provision according to user/organization requirements and context; and 3) how to provide middleware with the ability to support the earlier discussed issues in dynamic and uncertain environments.

Work in the area of user-centric collaborative communication tend to be limited in their ability to support reuse of their communication components for building more extensive communication applications. Automated reconfiguration at runtime of these communication components are also lacking in these class of applications. Middleware that support dynamic adaptation in some form are generally focused on low-level infrastructure with interfaces defined for programmers and provide few avenues for end user interactions. The initial NCB [83] provided a higher level of abstraction that encapsulated the complexity of the underlying network. The initial version of NCB however lacks a framework for systematically integrating additional low-level communication services and mechanisms to support the dynamic selection and reconfiguration of services by users.

The research problem is divided into three sub-problems:

1. Methodology to ensure the completeness or near completeness of the current simplified NCB high-level API to support it's reuse on other communication services frameworks.

2. Formulate a methodology:

   (a) for selection of services based on user intent.

   (b) to support automated reconfiguration of services and communication frameworks.

3. Design an extensible framework that supports seamless integration of communication frameworks.

3.3 Objectives and Evaluation Criteria

This section describes the specific, measurable objectives of this research and the criteria for the evaluation of the objectives. The research problem as described in Section 3.2 was divided into three distinct sub-problems. Objective 1, focuses on sub-problem 1 of Section 3.2, Objective 2 focuses on the second sub-problem and Objective 3 focuses on the third sub-problem.

3.3.1 Near Minimalization of Services API.

To support the goals of the user-centric paradigm discussed in Section 2.1.3, another level of abstraction, above the currently available set of APIs for communication services will be needed by non-technical users in the domain. Skype, with a near two hundred command API, and Smack, with over one hundred commands in it's API, will become too burdensome for non-technical users. An analysis of the domain is needed to identify the minimum elements and relationships need to describe communication

within this domain. This identification process must also be viewed from the users' perspective. It is hypothesized that the near minimal API will be equivalent to the initial NCB API.

Objective 1: Perform a domain analysis of the domain of communication services frameworks aimed at identifying a minimal API to support service requests. The results

Evaluation Criteria: the analysis shall:

1. characterize the domain of user-centric communication
2. formalize the abstraction that represents a minimal set of elements and associations.
3. validate the completeness of the NCB API with respect to near minimal API for communication services.

3.3.2 Intent Based Self-Configuration of Services.

An outcome of understanding the domain lies in identifying attributes of features that contain malleable states and those that do not. Through observation or modification of the variables of these attributes, system behavior can be manipulated to ensure that the system achieves or maintains predefined levels based on cost or services. A structured representation for policies would be needed as well as an efficient methodology for evaluation and enforcement of these policies.

Objective 2(a): Develop a methodology for selection of services based on user preferences; Develop a policy-based methodology for evaluating best cost and/or best performance services that satisfies the user's intent. The user's intent will be specified as abstract high-level goals.

Evaluation Criteria: the methodology shall:

1. characterize the domain of user-centric communication
2. formalize the description of the user intent with respect to the domain of user-centric communication to support automated evaluation techniques
3. support efficient evaluation of the formalized description of user intent

Objective 2(b): Develop a methodology for automating the configuration of communication services and communication service providers. Policies previously defined will guide the automated re-configuration of communication services. The real-time nature of the domain requires efficient evaluation of policies at runtime.

Evaluation Criteria: the methodology shall:

1. provide support for selection algorithms based on selection metrics identified in Objective 1(a).
2. provide algorithms that can be leveraged to evaluate such policies at runtime.
3. provide evaluation results that can can be leveraged to support decisions and enforcement mechanisms.

3.3.3 Integrated Services through Multiple Communication Frameworks

With an understanding of the domain of interest that facilitates identification and characterization of it constituent features, abstraction can be developed that supports extensibility of services. An architecture and appropriate algorithms can them be developed to support the analysis of the abstractions of the services, independent of the communication application frameworks. A set of general communication services can be derived that is representative of the class of communication application frameworks. This ensures that any considerations for abstraction represents a through examination of the user-centric communication domain.

Objective 3: Design an extensible architecture for integrating communication services provided by multiple communication application frameworks. These services include audio, video, file transfer and chat with a communication application framework providing one or more of these services.

Evaluation Criteria: the design shall:

1. be extensible to support integration of additional communication frameworks and services.

2. provide the independent mapping of communication services abstractions.

3. support adaptability and self-configuration of services at run-time.

3.4 Summary

In this chapter, this focus of the study was defined as 'the investigation of an approach that includes the provision of a high-level abstraction and automation techniques that support an adaptive runtime reuse of communication services'. The work was motivated through a discussion and a scenario derived from collaborators of the project. The sub-problems highlighted as:

1. Formulate a methodology:

   (a) for selection of services based on user intent.

   (b) to support automated reconfiguration of services and communication frameworks.

2. Design an extensible framework that supports seamless integration of communication frameworks.

Criteria for evaluating success of meeting the objectives are also presented in this chapter. The next chapter presents the details of the approach.

CHAPTER 4

UCC SELF-CONFIGURING APPROACH

In this chapter is a presentation of the survey of existing communication frameworks. The results of the survey are used to define the domain of user-centric communications via the FODA methodology. The definition of the user-centric communication policy structure is presented as well as the mechanisms to evaluate the policies.

4.1 Overview of Approach

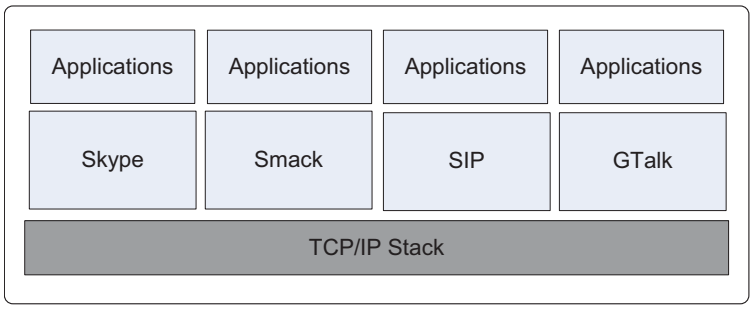


Figure 4.1: Simplified View of the Architecture of Current Approaches

This dissertation proposes a user-centric approach to address the problems outlined in Chapter 3. The proposed approach aims to reduce the complexity to the user by providing services through the seamless integration of multiple communication service providers. This goal will be supported by sub-goals that (1) provide an extensible framework to support the integration of multiple communication service providers, and (2) provide a methodology for automated selection of the communication services guided by high-level user preferences.

Figure 4.1 presents a simplified view of the architecture used in the current designs that support reuse of communication applications. The lighter colored layers in the architecture represent the existing state-of-the-practice, which is a vertical approach

to building communication applications. Essentially a user application is layered directly on top of one of the communication services framework utilizing the TCP/IP stack. The creation of multiple variants of the application would be needed to support the application's use on multiple communication frameworks. A user's conceptual intent can be viewed as a model that is defined as a user connection in the application. In this regard the association between user connection and the realization as a low-level session is immutable across communication services frameworks. The user's intent which conceptually can seamlessly transition based on needs and preferences, would in this architecture require the manual destruction of the user connection on one communication services framework and the recreation on another communication services framework. Fundamental changes are needed to support the goals stated in the earlier paragraph.

The abstract architecture for the proposed solution is shown in Figure 4.2. The proposed runtime adaptable middleware is composed of the following functional layers from top to bottom: user *Applications* that reuse the collaborative communication services; *Middleware API layer* that provides a high-level abstraction to applications and users; *Mapping Layer* that associates a high-level to low-level communication session; *Selection Layer* that selects the communication services framework that best serves a user's needs; and *Convergence Layer* that aggregates the communication services and the communication services frameworks that provide the services. At the bottom of the architecture is the TCP/IP stack that is utilized by all the communication services frameworks and to the center left is a horizontal layer, the *Knowledge Repository*, that handles repository and registry services for the connecting layers.

The black layers in Figure 4.2 are the proposed extensions for a collaborative communication middleware architecture. The purpose of these layers are now further elaborated.

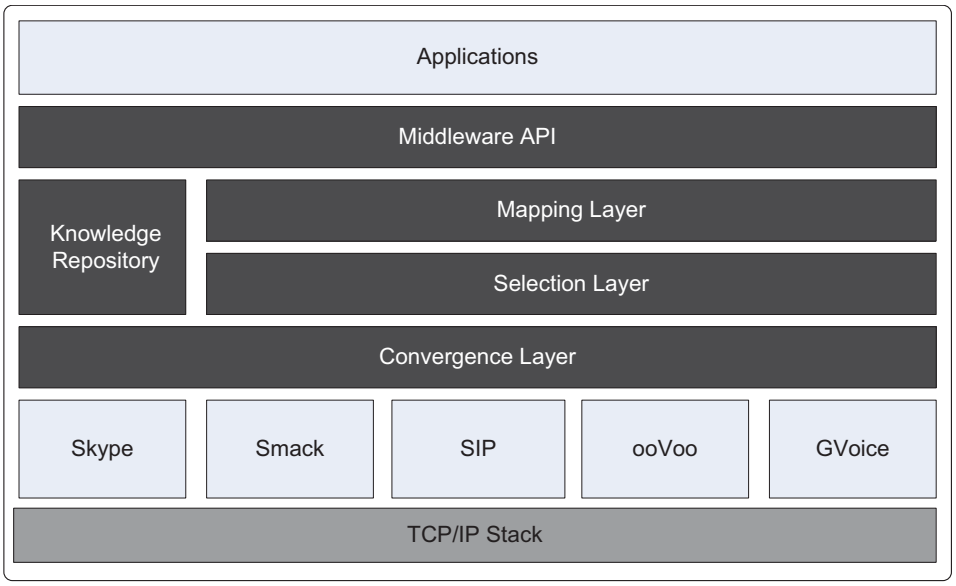


Figure 4.2: Proposed Architecture

Middleware API layer: - provides a high-level abstraction to applications and users. The API signatures need to be sufficient to support the mix of possible underlying communication services frameworks. API calls are translated to or modify an *intent model* representing a desired set of communication services for the user. This layer includes ports for injecting high-level user defined preferences that will aid in constraining the transitions of the intent model.

Mapping Layer - provides the logic for managing association between two distinct communication models. In this case, the mapping is from a model of intent expressed at the API layer to a runtime model executing on a communication framework. This should facilitate a decoupling of user intent from the actual communication execution thereby allowing dynamic adaptation of services.

Selection Layer - provides algorithmic reasoning for selecting best served communication framework and services. The algorithms will need to simple and fast as they will be bounded by the real time nature of the domain. The algorithms will also need to be flexible to support the goal of dynamic adaptability in the architecture. Key inputs

for the decision algorithms will be the user preferences as well as representations of the available services and communication services frameworks.

Convergence Layer - provides an aggregated set of communication frameworks and their available services. Distinct communication services frameworks are integrated at this layer. A common abstracted view for accessing communication services offered by these communication services frameworks will therefore need to also be provided.

Knowledge Repository - provides a repository for all data shared by the previously described layers. As can be seen at center left in Figure 4.2, knowledge repository interfaces with the API layer (high-level user defined preferences and the intent model), the mapping layer (model associations), the selection layer (input and output for selection logic) and the convergence layer (set representing aggregation of communication services).

The proposed architecture is the foundation for the proposed approach. The approach simplifies the experience for the users of the middleware but moves the complexity from the user into the middleware. Complexities introduced relate to:

Switching metrics - In more traditional areas that utilize handover, such as cellular networks, the switching decision is generally based on Relative Signal Strength and on call drop rate. The complexity is elevated in this situation. In addition to QoS type metrics, other criteria such as cost which will be based on the network, location and time of day of the called party become factors that affect the decision as well. This will need to be balanced against user preferences which may dictate priorities and weights for some set of metrics. Moreover, for a switch to occur the communicating users must have some common subset of communication services providers as well. The switching metrics in this situation should include user preference, available services frameworks, peers available services frameworks, application types, cost, network conditions and other QoS metrics.

Switching decision algorithm - Based on the switching metrics mentioned above, the decision about when and how to switch the intent model to which communication services framework will be made. How individual metrics are weighted and the relationship of weights for all metric are as important as the metric themselves for the reasoning algorithms. The level of interaction that the user has in the weighting process and the need to ensure that the process does not become burdensome for the user must also be balanced. How to switch will need to tackle complexities such as ordering (destroy old session first or create new session first?) and impacting constraints (limited resources to support old session and new session at the same instance). Trade offs may need to be made in such cases. The algorithms will need to resolve these and other conflicts that may arise, potentially even at runtime.

Mobility management - When a remapping for an intent model is required, a transfer from one communication framework to another may occur. All peer users involved in the communication need to correctly switch communication services framework and do so in a timely, ordered way. Peer notification is therefore critical in ensuring all peers are provided the correct 'next' framework. The state of the session may also need to be preserved and recreated to ensure substantive continuation of the communication. While outside the scope of this dissertation, mobility management must also investigate the challenges of enabling on-the-fly loading and unloading of communication services frameworks.

To assure the feasibility of the approach, the introduced complexities need to be ameliorated. To ensure a clear understanding of the potential elements that can aid in the reduction of the complexities as well as eventual solutions, a clear understanding of the domain is necessary. In Figure 4.3 the inputs and output for a self-configuration request is shown.

- Step 1: An *initial configuration*, a user's request for a new connection or a new service would be evaluated.

- Step 2: A set of configuration commands is generated for the configuration of the communication services framework.

- Step 3: A *dynamic configuration*, a user's request to reconfigure an existing connection or service, would be evaluated.

- Step 4: A set of configuration commands are generated to reconfigure a framework or (as in this example) replace a framework. To ensure that the required goals of the initial and dynamic configuration were met, relevant states would be included in the monitored set of states to provide feedback.

- Step 5: A *reactive configuration*, monitored states of the framework that are out-of-band , would be evaluated.

- Step 6: A set of configuration commands are generated to reconfigure or replace a framework.

User-centric communication policies would have to be developed for the classes of configuration discussed previously. Policy definition is presented in the next section.

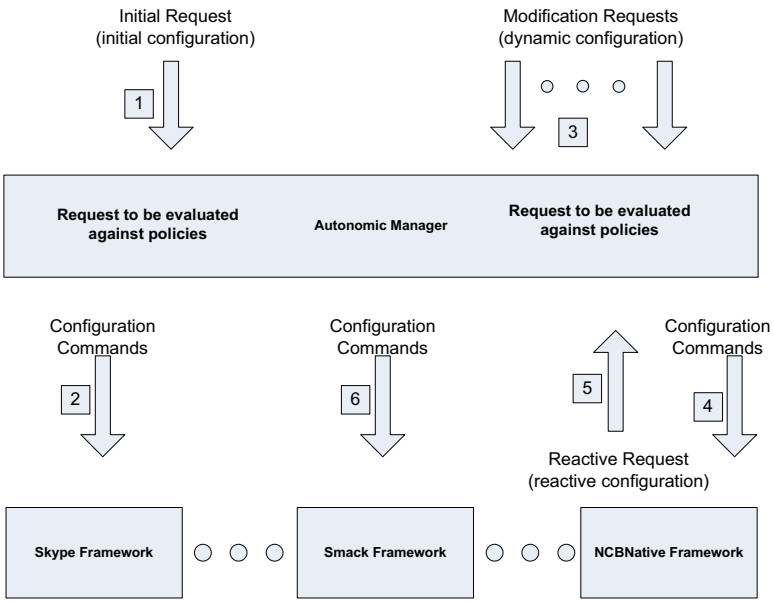


Figure 4.3: Self-Configuration steps for Frameworks

The architecture and approach were further refined and the refinements are presented next. The approach was implemented in the Communication Virtual Machine (CVM), a model-driven technology for realizing communication applications. The CVM includes the Network Communication Broker, the layer responsible for providing a network-independent API to the upper layers of CVM.

4.2 Feature Analysis for UCC Domain

To define policies for guiding user-centric communication, a detailed domain analysis needs to be performed to extract the essential 'characteristics' of the communication domain. *Feature Oriented Domain Analysis* (FODA) [36] provides a systematic approach to address this problem. As a method for discovering and representing commonalities among related software systems, the primary focus of FODA is the identification of prominent or distinctive features of software systems in a domain. It leads to the creation of a set of products that define the domain in terms of its mandatory, optional and alternative characteristics of related systems. In this subsection the application of FODA method to the user-centric communication domain is presented. The focus is on domain modeling for the purpose of this study, which is an important phase of FODA that defines the problems within the domain addressed by software.

Successful FODA practices builds on understandings of both the common aspects, as well as differences of related systems in a targeted domain. This requires a detailed survey of existing systems to capture the commonalities and variabilities as comprehensively as possible. An initial survey of three communication frameworks [3] was performed to provide a view of the domain with respect to each one's supporting features. Based on the increased interest in the domain by new service providers, a wider and more representative set of communication frameworks that are currently popularly used in industry was used to extend the original survey. Moreover, more

distinguishing features of these frameworks was incorporated, such as message archiving and importing contact list. Figure 4.4 shows the result of the extended survey.

In this table, a 1 indicates this feature is present while a 0 indicates an absence. Entries with a 1* show that the feature comes with costs, R shows the feature has restrictions. Besides the basic features of communication services such as contact list and chat, which all frameworks support, additional features were incorporated that are of potential interest to the user. The additional features provide a rich set of fine grained properties that complement basic features in various aspects. Although they are not essential in delivering basic communication services, they bring more variabilities of the systems that could affect their potential usage.

Core Features	NCB Native	Skype	JML	Gtalk	Android	Yahoo!	Windows Live Messenger	Blackberry OS	AOL	Palm OS
Chat (one-to-one)	1	1	1	1	1	1	1	1	1	1
Chat (Conference)	1	1	1	1	1	1	1	1	1	1
Audio (one-to-one)	1	1	0	1	1	1	1	1	1	1
Audio (Conference)	1	1	0	0	0	1	1	1	0	R(<= 3)
Video (one-to-one)	1	1	0	0	1	1	1	1	1	0
Video (Conference)	1	0	0	0	0	1	1	0	0	0
File Transfer	1	1	1	1	1	1	1	1	1	1
Contact List	1	1	1	1	1	1	1	1	1	1
API	Java	Java	Java	C++	Java	JavaScript /C++	JavaScript (HTTP)	Java	Java C/C++	HTML
Additional Features										
Emoticon	1	1	1	1	0	1	1	1	1	1
Online Status	1	1	1	1	1	1	1	1	1	1
Avatar Images	0	1	1	1	1	1	1	1	1	1
Voicemail	1	1	0	0	0	1	1	1	0	1
PC to Phone	1	1*	1*	1	0	1*	1*	0	1	1
Phone to PC	1*	1*	0	0	0	1*	0	1*	1	0
Message Archive	0	0	0	1	1	1	1	1	0	1
Plug-Ins	0	1	1	1	1	1	1	1	1	1
Importing Contact List	0	1	0	1	1	1	1	1	1	1
IM forwarding to cell phone	0	1	0	0	0	1	1	1	1	1
Radio	0	1	0	0	0	1	0	1	1	1
Importing Contact List	0	1	0	0	0	1	1	1	1	1

Figure 4.4: Survey of Communication Frameworks

The domain models describe elements of systems in a given domain from the point of view of the 'problem space' [36]. An important artifact of domain modeling is the feature model. Features are the attributes of the system that directly affects the end-users. The feature model of the framework gives us a logical grouping of the features

of systems in the domain that are of interest. Figure 4.5 shows the feature diagram resulting from feature modeling. The focus is on user centric communication applications as the interested family of systems. It includes several mandatory features graphically denoted by solid circles above or beside the feature name, as in `contact list` (top right feature in Figure 4.5), and optional features denoted by empty circles, as in `file transfer`(second left feature in Figure 4.5).

Alternative features are connected by an empty arc, showing one and only one of the sub features must be present, while a filled arch connecting features denote or-features, meaning you can have one or several of such features. For instance, user-centric communications could have chat, or audio, or video, or any combination of them, but a PC2Phone is either free or at a cost. Each top feature of user-centric communications, which is call the *main feature*, have their own sub features, either optional or mandatory, representing properties of the main features. The hierarchical structure goes down until there are no further properties to be captured. The feature diagram is extensible, as the FODA process can be refined through iterations in the future.

Feature analysis helps to capture the domain model in terms of the various characteristics or considerations of the domain. The results of the feature analysis will be used as the basis for designing policies that will guide how such "considerations" are satisfied by means of self-configuration. Details of the user-centric communication policies will be explained in the next subsection.

Figure 4.4 shows a diverse set of features that characterize the user-centric communication domain, however there exist users who have no interest in a manually evaluating fine grained configuration of their communication. This work on the feature analysis of the domain suggests the potential for automation that can reduce the complexity to the user. A user's request for communication services can be guided by a combination of goal and action policies, which is referred to in this disserta-

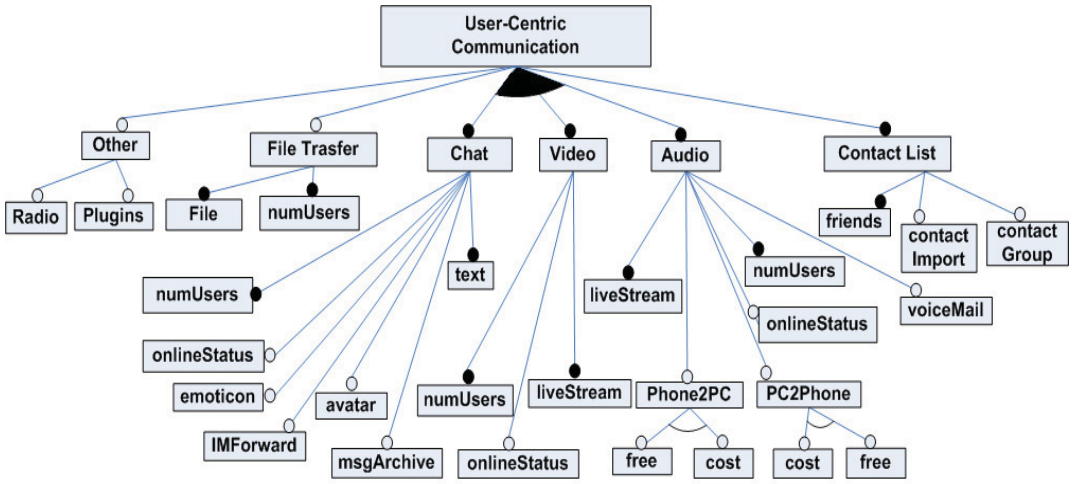


Figure 4.5: Feature Diagram for Frameworks

tion as *User-Centric Communication Policies*. User-centric communication policies are policies that aid the simplification of communication while enhancing the user experience.

4.3 UCC Policy Definition

Policies are rules that define the choices in the behavior of a system [47]. Agrawal et. al [2] define a system's behavior to be 'a continuous ordered set of states where the order is imposed by time'. Each state can be viewed as a mapping of some values ${V^t}_i \ldots {V^t}_n$ to the set of system attributes A^t_S where ^{t} is some ordered time slice.

Let $B(S)$ be the set of all possible behaviors that the system S can exhibit.

$$B(S) = \{\{{V^0}_i, ..., {V^0}_n\} \mapsto A^0_S, \{{V^1}_i, ..., {V^1}_n\} \mapsto A^1_S, ..., \{{V^t}_i, ..., {V^t}_n\} \mapsto A^t_S\}$$

There should exist some set of constraints C such that when C is applied to $B(S)$, it maps to a subset of desired behaviors $B_{desired}(S) \subset B(S)$.

Then

$$g : (C, B(S)) \rightarrow B_{desired}(S)$$

Policy P is defined as the function g:

$$P(C, B(S)) \rightarrow B_{desired}(S)$$

Therefore

$$P(C, B(S)) \subset B(S)$$

Where policy P characterizes a subset of the possible behaviors $B(S)$ of a target system S that satisfies a set of constraints C; that is, it defines a subset of $B(S)$ of acceptable behaviors for S [2]. Policies can be defined using only a small number of attributes of system state and do not require the determination of the complete state a priori [2]. Constraint C is defined as a 4-tuple $\langle C_O, C_N, C_V, C_D \rangle$ where:

- C_O: narrows the scope of the constraint to identify subcomponents of S to which the constraint is applied;
- C_N: represents the condition(s) under which the constraint is triggered;
- C_V: facilitates the ranking of multiple applicable constraints based on some expected business value or utility; and
- C_D: associates a condition C_N with an action that achieves some desired behavior.

The aforementioned elements are extended based on the results of domain analysis of the collaborative communication domain [3,7]. Figure 4.6 is an example of the XML version of a user-defined policy that guides the establishment of video conferences. The elements are interpreted as:

- *Scope (C_O)* identifies the applicable communication component (the subject of the communication) using the service attribute. A second attribute indicates the status of the policy as being active or not. e.g *Communication Object*.

```
<csmPolicy policyName="selectComm_Video_01">
    <scope>
        <service>Communication Object</service>
        <active>true</active>
    </scope>
    <condition>
        <feature>Video</feature>
        <operation>request</operation>
        <literal></literal>
    </condition>
    <businessValue>
        <businessGroup>general</businessGroup>
        <value>96</value>
    </businessValue>
    <decision>
        <param>Enabled</param>
        <operation>equalTo</operation>
        <value>conID.enabled</value>
    </decision>
</csmPolicy>
```

Figure 4.6: XML representation for user-centric communication policy.

- *Condition (C_N)* represents the trigger for the application of the policy in terms of: (1) feature - the carrier of the intended information to be communicated; (2) operation - the action to be performed on the proposed medium; and optionally, (3) some value provided for comparison. e.g. *when a request for video is received* (feature:video, operation:request).

- *Business value (C_V)* facilitates a ranking of triggered polices. It is represented by (1) the business group attribute which provides a way to associate related policies; and (2) a numeric value that represents the policy's priority in the group. e.g. *general group with priority 96.*

- *Decision (C_D)* indicates the policy's desired outcome or expected behavior of the communication. It is expressed as a triple (consisting of parameter, operation and value) that specifies the criteria that the communication must satisfy. e.g. *select communication framework whose medium supports at least the connection's users count.*

Details on the evaluation of this example are provided in the following Subsection.

4.4 UCC Policy Realization

Recall from Section 4.3 that a user-centric policy is characterized as:

$$P(C, B(S)) \subset B(S)$$

where constraint C is a 4-tuple $\langle C_O, C_N, C_V, C_D \rangle$ and

$C_O = \langle sc, b \rangle$ where subcomponent sc exists in System S and boolean b is an indicates if a policy is available;

$C_N = \langle f, oper_1, l \rangle$ where feature f is in the set of features of subcomponent sc, $oper_1$ is an applicable operation of feature f and literal l is a discrete value that may optionally be applied to $oper_1$;

$C_V = \langle g, r \rangle$ where g is an associable business group and r a discrete rank within the business group;

$C_D = \langle param, oper_2, val \rangle$ where $param$ is a valid port in the subcomponent sc of S, $oper_2$ is comparison operator and val is a value or range of values for $param$;

Given a runtime system S and a set of events T that occurs in S where

$\{S.fs\}$ is a set of features active for the a given connection;

$\{sc.fs\}$ = is a set of features supported by sc

$T = \{e|$ monitored event in $S\}$.

A single policy P_1 of form $\langle C_O, C_N, C_V, C_D \rangle$ on (S, T) is interpreted as follows:

$$C_O \wedge C_N \implies apply(C_D)$$

which can be further refined as:

$$C_O := (C_O.sc \in S \wedge \{S.fs\} \subseteq \{C_O.sc.fs\} \wedge C_O.b = ACTIVE)$$

that is: C_O implies subcomponent sc exists in System S at runtime, the set of active features in $\{S.fs\}$ are a subset of $\{C_O.sc.fs\}$ and b indicates that Policy P_1 is active. Note that the absence of support for an active feature in the specific connection on the running system will remove the subcomponent from consideration. It will be assumed that the user will explicitly remove unwanted features; and

$$C_N := \{\exists e \in T : C_N.f = e.f \wedge C_N.op_1 = e.op\}$$

that is: C_N implies Event e is of the monitored type T and the feature $e.f$ of Event e and the operation $e.op$ match the feature $C_N.f$ and operation $C_N.oper_1$ of C_N.

$$apply(C_D) := enforce(C_O.param, C_O.oper_2, C_O.val)$$

where $C_D.param$ is a port on $C_O.sc$, is a desired state of $C_D.param$ and $C_D.oper_2$ defines the relationship of val to the $param$. It is noted that C_D describes goal states for some set of variables in subcomponents of C_O.

Interpretation of multiple policies build on the previously described interpretation of single policies. Given policies $P_1 \ldots P_n$ each of the form $\langle C_O, C_N, C_V, C_D \rangle$ on (S, T), interpretation is as follows:

$$\forall P_i : C_{Oi} \wedge C_{Ni} \implies PS \cup P_i$$

where P_i is a policy in $P_1 \ldots P_n$ and PS is a set of applicable policies. The set PS is further refined by

$$CFPS = resolveConflict(PS)$$
$$RankedPS = rank(C_v.v, rank(C_V.g, CFPS))$$

where $CFPS$ is a set of conflict free policies, and $RankedPS$ is a ordered set of policies based on $C_V.g$ is an ordering on business group which is further ordered by

Line Number	CO		CN			CV		CD		
	sc	b	f	op_1	lit	g	v	param	op_2	val
1	CommObject	ACTIVE	VIDEO	START	xAzv34	A	10	enableMedium	EQUALTO	ENABLED
2	CommObject	ACTIVE	VIDEO	STOP	xAzv35	A	12	disableMedium	NOTEQUALTO	ENABLED
3	CommObject	ACTIVE	VIDEO	STOP	xAzv36	A	12	disableMedium	NOTEQUALTO	DISABLED
4	CommObject	ACTIVE	VIDEO	REQUEST		B	13	getFrameworkSet	MINIMIZE	PRICE
5	CommObject	ACTIVE	VIDEO	REQUEST		B	14	getFrameworkSet	MAXIMIZE	PROFORMANCE
6	CommObject	ACTIVE	VIDEO	REQUEST		B	15	getFrameworkSet	MAXIMIZE	PRICE
7	System	ACTIVE	MEMORY	LOW		C	16	getFrameworkSet	MINIMIZE	PROFORMANCE

Table 4.1: Example Policies for Interpretation.

$C_V.v$, the value within the group. The methodology for conflict resolution will be discussed later in this section.

$$\forall P_i \in RankedPS\ apply(C_{Di})$$

where for all P_i, policies in $RankedPS$ the policy set, apply the decision C_{Di}.

In Table 4.1 is a potential set of policies. A row in Figure 4.1 represents one policy. Column 2 of the table list the subcomponent targeted by the policy, $C_O.sc$ and column 3 shows the active status of the policy, $C_O.b$. The feature of the subcomponent of interest ($C_N.f$), the operation on the feature ($C_N.op_1$) and the associated literal ($C_N.l$) are listed in columns 4,5 and 6. Similarly, $C_V.b$, the business group for the policy, and $C_V.v$, the rank within the group is listed in colunms 7 and 8. Columns 9, 10 and 11 list the goal parameters (($C_D.param$) - port, ($C_D.op_2$) - operator and ($C_D.val$) - range).

We assume that the subcomponents described as 'CommunicationObject' and 'System' exist in S and an event e (start video for conID:dffdf) is in T. Based on the previous discussion, policy 1 in Table 4.1 would be interpreted as $C_O.sc$ and $C_O.b$ is satisfied. For policy 1, $C_N.f$ is 'VIDEO' and $C_N.op_1$ is 'START' which matches the event e. An additional parameter is provided by the event, that is the specific connection identifier ''. This value is substituted in $C_N.l$ placeholder 'conID'. C_O and C_N are satisfied for policy 1 and C_D must now be applied. The state derived

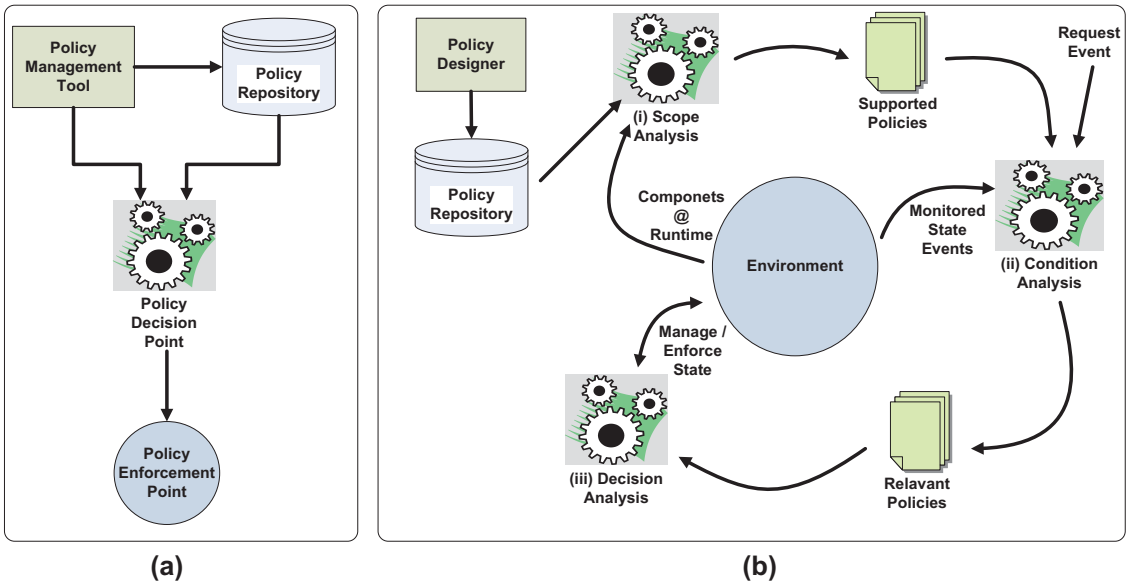


Figure 4.7: (a) IETF/DMTF Policy Architecture (b) Runtime Policy Evaluation.

from the port defined in $C_D.param$ (enableMedium) must satisfy ($C_D.op_2$ that is, 'EQUALTO') as defined in $C_D.val$ ('ENABLED').

In the case of multiple policies to be interpreted, (S, T) must be such that the subcomponents and conditions are satisfied by more than one policy. Each policy would be evaluated based on the current state of the system to test if it satisfies the targeted subcomponent and necessary conditions to be a fdf policy. Policies that evaluate to true in this respect are add to the policy set. For the purposes of this example, we assume that to be the case for policies 4 - 6 in Figure 4.1. Note the conflicting decisions of Policies 4 and 5. Conflict resolution will be needed to select one of the two opposing choices, this would be addressed in the $ReseolveConflict(PS)$ function mentioned earlier (in this case we defer to minimization for prices in this scenario). A ranking of the remaining policies follows and the policies would be applied in the following order .

Refinement of this conceptual view of policy interpretation is the basis for the policy evaluation mechanism used in the NCB.

While policy definition describes the 'what', policy interpretation represents the 'how' that ensures stated goals of the policies are met. The IETF/DMTF policy architecture identifies four core elements to support a policy framework (see Figure 4.7 (a)). These four elements are 1) the policy management tool, used to create and define the policies for the system; 2) the policy repository, which provides storage and retrieval mechanisms for policies; 3) the policy decision point (PDP), which has logical entities that decide applicability of policies and what is needed to comply with the policies; and 4) the policy enforcement point (PEP), which consists of logical entities charged with enforcing the policy decisions. This architecture represents the fundamental model supporting much of the standards work in this area [2, 52, 53]. As such the approach for supporting policies in the NCB was influenced by this architecture.

Figure 4.7 (b) presents an overview of the lightweight policy evaluation mechanism used in satisfying the users communication needs. User policies are created with the Policy Designer [7], a simplified policy management tool that supports the policy definition presented in Section 4.3, and stored in the *policy repository*. At runtime policies are retrieved from the *policy repository* as a first step in selecting applicable policies. The policy decision point, as identified in Figure 4.7 (a), is logically divided into the three analysis processes described in Figure 4.7 (b) and identified as (i,ii,iii). The policies and the available *components at runtime* are input for *scope analysis* (Figure 4.7 (b)(i)), where the policy scopes matching active components are selected, resulting in a subset of *supported policies*. In the context of NCB the components are the frameworks and the state of their services.

Events such as user requests or monitored state (reactive events) trigger further refinement of the supported policies via *condition analysis* (Figure 4.7 (b)(ii)). Policies with matching conditions to the event or environment state are selected to create the set of *relevant policies*. The decisions of the set of *relevant policies* (Figure 4.7 (b)(iii))

Algorithm 4.1 Algorithm to select applicable Policies.

```
1: createPolicySet(evt, commFWSet, polRepos)
   /*Input: evt - events from reactive requests or user requests
            polRepos -available policies in the policy repository
            commFWSet - set of available communication frameworks
     Output: policySet- set of policies ordered by Business Value */
2: policySet ← {}
3: for all pol ∈ polRepos do
4:    if pol.scope(commFWSet) = true then
5:       if pol.condition(evt) = true then
6:          policySet ← policySet ∪ {pol}
7:       end if
8:    end if
9: end for
10: return policySet
```

are the goals or range of states that the system should support. Policy enforcement is provided by the Orchestration Autonomic Manager (OAM) which is included in the representation of the environment shown in Figure 4.7 (b). The details of the OAM are discussed in Chapter 5.

In Algorithm 4.1, the algorithm for policy selection is outlined. This can be viewed as the algorithmic equivalent of Figure 4.7(b) (i) and (ii). Each round of policy selection begins with an empty policy set, step 2 of Algorithm 4.1, to which relevant policies are added. Policies are deemed relevant if the request attributes (encapsulated in an event as described in Algorithm 4.1's input) match the policy values for service, feature or operation (step 4 - 5 of Algorithm 4.1). When relevant policies are looked up and retrieved from the repository, an equivalent object representation is built with all the tags and values extracted and stored as object attributes. Policies are stored as XML in the *policy repository* (recall from Figure 4.7(b)).

For a set of relevant policies, the policies are processed in the order of their business values. Since UCC policies define how a user's request is mapped to an underlying communication framework, it would be straightforward to go through all the currently available frameworks and use the policy to decide which one satisfies the request. In its simplest form this set reduction technique is equivalent to the naive set theory

Algorithm 4.2 Algorithm to produce candidate set of Communication Frameworks.

```
1: setReduction_CommFW (evt, commFWSet, policySet)
   /*Input: evt - events from reactive requests or user requests
            policySet -policies relevant to the event
            commFWSet - set of available communication frameworks
     Output: commFWSet- candidate set of frameworks */
2: if policySet.hasNext ≠ false then
3:    if commFWSet ≠ empty then
4:       pol ← policySet.next - get and removes the element from the set
5:       for all frmwk ∈ commFWSet do
6:          if eval(pol, frmwk) ≠ true then
7:             commFWSet ← commFWSet \ {frmwk}
8:          end if
9:       end for
10:      commFWSet ← SetReduction_CommFW(evt, commFWSet, policySet)
11:   end if
12: end if
13: return commFWSet
```

for intersection [18]: start with a full set, and gradually reduce it until all policies are processed. Our algorithm for set reduction is shown in Algorithm 4.2. The input for the algorithm includes the policy set resulting from the policy selection described earlier, a set of available communication frameworks and the request encapsulated in an event. Each policy in the policy set is extracted (line 4) and compared against the communication frameworks with support for the stated feature, lines 5 - 10. Communication frameworks that do not support the stated feature or cannot satisfy the decision component of the policy are removed from the set of communication frameworks (lines 10 - 11 and 14). The next policy from the policy set is applied against the resulting communication framework set until all policies are evaluated (line 17). The evaluator produces a set of candidate frameworks as its output for each request.

As with almost all non-trivial policy-based approaches there exists the possibility of policy conflicts. Approaches for the static detection and resolution of policy conflicts have been proposed [35, 46]. The area of dynamic detection and resolution of policy conflicts at runtime is still fertile ground for research, although recent work [11] has suggested renewed interest in this area. We reuse some of the techniques pro-

posed in [11, 35] to address the policy conflicts concern in the evaluation mechanism to adequately support correct interpretation.

4.5 UCC Policy Application on the Illustrative Example

In Section 3.1 we presented an illustrative scenario and indicated three requirements that needed to be satisfied for the scenario. We restate the three requirements here for ease of reference. They are:

- Requirement 1. Provisioning of a three-way audio-video conference
- Requirement 2. Provisioning of a four-way audio conference
- Requirement 3. Complying with the hospital's directive for minimizing cost

These requirements can be expressed as constraints to be applied to the system. We further express these requirements in two forms. The first is a goal-oriented or declarative view, a highly abstract and implementation-independent expression. The second is a less abstract imperative view, which is bounded to the specifics of the interpretation approach. We present both views to underscore the subtle differences in how a policy is defined as against how it is evaluated at runtime and also to highlight the independence of the definition process.

Requirements 1 and 2 are conceptually similar. We therefore provide a discussion on Requirement 1 only, however noting the applicability to Requirement 2. In the case of Requirement 1, an equivalent declarative expression would be:

$$\{\forall f \in CSet_{cid} | AudioVideo \in f.services\}$$

This can be read as: a candidate set instance $CSet_{cid}$ is a set of communication frameworks where each member f includes $AudioVideo$ in the set of available services $f.services$ at runtime.

The interpretation of a policy to support Requirement 1 would be expressed differently when viewed within the context of the set reduction methodology used in the implementation. An equivalent imperative expression would be:

$$\{\forall f \in CSet_{cid} | if(AudioVideo \notin f.services) then(CSet_{cid} - \{f\})\}$$

This is interpreted as: for each member f of the candidate set instance $CSet_{cid}$ an evaluation is done; the non inclusion of $AudioVideo$ in the set of available services of $f.services$ at runtime requires the removal of that communication framework f from the candidate set instance $CSet_{cid}$.

While the class of policies exemplified by Requirements 1 and 2 produce sets that satisfy a desired goal, polices of the class that reflects Requirement 3 will produce at most a single candidate. The single candidate produced by this class of policies results from some maximum or minimum operation on a set of possible candidates. Requirement 3 describes a minimization objective, below we note an equivalent declarative expression of this requirement:

$$\{\exists g \forall f \in CSet_{cid} | g.Audio.cost \leq f.Audio.cost\}$$

This is interpreted as: there exists a member g of the candidate set instance $CSet_{cid}$ such that some property value, $g.Audio.cost$, is less than or equal to the property value, $f.Audio.cost$, of every other member of the set. The interpretation of Requirement 3 expressed in an equivalent imperative form would be:

$$f_{candidate} \leftarrow \underset{0 \leq t \leq z}{ARGMIN} \left(\sum_{i=0}^{n} f_t.services(Audio).cost(uloc, urem_i) \right)$$ [1]

[1] ARG MIN $f(x)$ gives a position x_{min} at which f is minimized

Which is: $f_{candidate}$ is the result of a minimization function, $ARGMIN$, over the set of candidates indexed from 0 to z. This minimization function uses the summation of the per minute cost for n many connection pairs of local user, $uloc$, to remote users, $urem$, as the comparative value for each candidate.

4.6 Summary

This chapter presented a FODA based domain analysis of the user-centric domain. This domain analysis informed the structural design of a policy language to support high-level specifications by users. The mechanisms for correctly interpreting the user-centric policies were discussed and algorithms used in the process were outlined. The next Chapter focuses on the framework that utilizes the user-centric policies.

CHAPTER 5

UCC FRAMEWORK

This Chapter presents the self-configuring user-centric communication framework. An overview of the framework is provided followed by a more detailed but high level view of the architectural approach. Some of the significant components are also highlighted with a detailed design and a discussion of the implementation.

5.1 Operational Overview

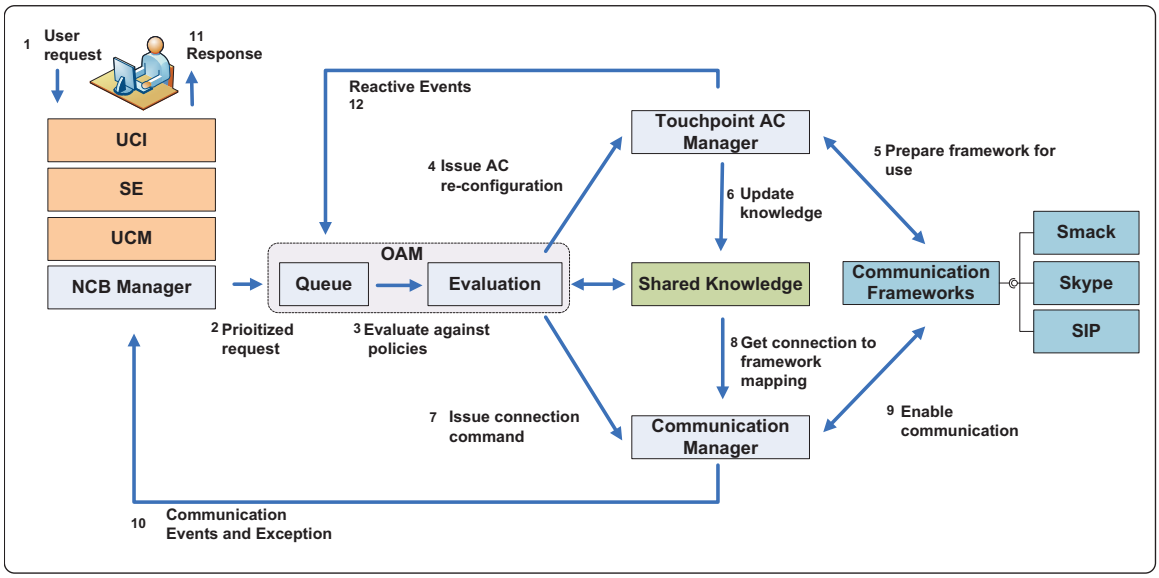


Figure 5.1: NCB Control Flow Diagram.

Figure 5.1 shows the flow of control for the proposed approach introduced in the NCB. The numbered steps in Figure 5.1 highlight a typical sequence of operations to initiate a collaborative communication.

Step:

1. A user's intent model for initiating a communication with a remote peer would be accepted by the upper layers of the CVM (see top left). The high-level intent

model is transformed to different level *communication models* as it descends the CVM stack. The interested reader can find details on the processing of the user's communication models in the UCI, SE and UCM layers in [80,82]. At the NCB Manager, the user' communication model becomes a series of API calls or *requests* (example:*createConnection*, *addParty*) for the NCB.

2. The requests are inserted into a priority *Queue*. A priority queue is used to ensure that high valued request (reactive events such as failed services) are handled before less valued request (proactive events such as send text file).

3. The highest priority request is evaluated against policies and the current operating environment state retrieved from the *shared knowledge*. The result of the evaluation is an identified communication framework that satisfies the user's request and does not violate the active policies.

4. If the evaluation process returns a new framework for the connection (that is, one that is different from the current one in use), the *TouchPoint AC Manager* (TPM) is issued commands to setup a new session on the proposed replacement communication services framework.

5. The TPM uses the services of the *Communication Framework Manager* to access and direct the appropriate communication services framework in preparing the required services.

6. On the successful completion of the session setup, the TPM updates the user connection-to-session mapping table in the shared knowledge.

7. The evaluation is one sub-component of the *Orchestration Autonomic Manager* (OAM), which is also responsible for usage synchronization of the communication frameworks. OAM blocks the *Communication Services Manager* (CSM)

while these reconfiguration steps are in progress, then issues the connection commands to the CSM.

8. The CSM looks up the session and associated communication services framework for the connection in the user session-to-connection mapping table in the shared knowledge.

9. The the communication commands are then executed on the specific communication services framework accessed through the communication framework manager.

10. Any notification events or communication based exceptions generated through the communication manager are passed to the NCB Manager.

11. These events and exceptions are expected to be resolved, actioned or viewed interactively by the user.

12. Any out-of-band or *reactive events* that cannot be handled by the TPM are encapsulated and forwarded to the OAM. The OAM processes these events guided by policies from the shared knowledge and directs the TPM appropriately. This effectively provides a stacked approach for the autonomic design with the OAM performing the role of manager for the TPM.

A systematic representation of the processing that occurs in the OAM is presented in Algorithm 5.1. The *evaluate* function describes the coordination methodology for the OAM. Two important objectives guided the design decisions of the algorithm. The first was the need to maintain the existing CSM interface and operations. This facilitates easy reuse and more importantly backward compatibility of previous versions of the CSM. Secondly, As highlighted in Section 3.1, the distributed nature and user-centric focus present complexities which have to be addressed for a feasible implementation. With each peer supporting independently defined and enabled user

Algorithm 5.1 Algorithm to Configure Communication Services.

```
1: evaluate(PQ, KS, TPM, CSM)
   /*Input: PQ - PriorityQueue of events from reactive requests or user requests
            KS -Knowledge Source containing repositories such as policy and Frameworks
            TPM -Touchpoint AC Manager
            CSM -Communication Services Manager
           commFWSet - set of available communication frameworks
   */
2: loop
3:    if PQ not empty then
4:       evt ← PQ.next
5:       polRepos ← KS.getPolices()
6:       commFWSet ← KS.getCommFWs()
7:       policySet ← createPolicySet(evt, commFWSet, polRepos) {Algorithm 4.1}
8:       policySet ← createPolicySet() {Algorithm 4.1}
9:       candidateSet ← commFWSet {create temporary copy of commFWSet }
10:      candidateSet ← setReduction_CommFW (evt, candidateSet, policySet) {Algorithm 4.2}
11:      candidateSet ← setReduction_CommFW () {Algorithm 4.2}
12:      candidateFW ← negPrimaryFW (candidateSet)
13:      if TPM.configure(conID, candidateFW) == DIFF then
14:         reConFigCommands(CSM, getConnection(conID)) {re-generate & execute previous calls}
15:      end if
16:      CSM.execute(evt.cmd) {execute current call for CSM}
17:   end if
18: end loop

19: reConFigCommands(CSM, ConObject) {ConObject - current abstract state of connection}
20: for all party such that party ∈ ConObject.partyList do
21:    commands ∪ {addParticipant(party)}
22: end for
23: for all medium such that medium ∈ ConObject.mediaList do
24:    commands ∪ {enableMedium(medium)}
25: end for
26: for all cmd such that cmd ∈ commands do
27:    CSM.execute(cmd)
28: end for
```

policies in the middleware, selection of a framework amongst peers become a global decision. We reduce this complexity to that of the problem of consensus in distributed systems, we present the details next.

The evaluate algorithm uses the priory queue, *PQ*, shared knowledge, *SK*, touchpoint AC manager, *TPM* and the communication services manager, *CSM*, to effect reconfiguration. Previously queued events, *evt*, are dequeued from the PQ (step 4) and the policies and set of available communication services frameworks retrieved from the SK (step 5 and 6). The items retrieved in steps 4 - 6 are used as input for

the createPolicySet Algorithm (Algorithm 4.1) producing the set of currently applicable policies, policySet, shown in step 7. The setReduction_CommFW (Algorithm 4.2) is then applied to reduce the available set of framework to the subset that satisfies the user's current requirements, step 9.

This candidate set produced in step 9 contains all the available frameworks that can support the features and users defined in the specific connection. Step 10 , negPrimaryFW, identifies the selected communication services framework that resulted from the negotiation amongst the peers. The negPrimaryFW describes the protocol used for consensus, we present details on the protocol later in this section. The selected communication framework identified in step 10 is passed to the TPM to pre-prepare the framework (ensure framework availability, create connection to session mappings and potentially this could include testing of services) in step 11.

Two results are possible with the TPM configure, the first is a change of framework associated with the specified connection identifier and the second is a reuse of a framework that is already associated with the connection identifier. In the case of the former, a remapping of connection identifier to framework identifier is needed. The previously associated framework identifier is denoted as 'old' and will be used to destruct the previous low-level session as needed. For the latter the 'new' and 'old' frameworks are the same, hence no change of framework is required. The function returns a *DIFF* enumeration when frameworks need to change and a *SAME* enumeration is no change is necessary. A return of *SAME*, step 14, will execute the event specific command.

If a *DIFF* is returned, then the OAM must generate the required actions or calls to re-establish the services on the newly selected communication framework. Creation and execution of the re-establishment actions will be the responsibility of the *reConFigCommands* function in the OAM, see step 11. The steps of reConFigCommands are listed from step 18 - 27. A *connection object (ConObject)*, which represents the

Algorithm 5.2 Algorithm to Negotiate Communication Services.

```
negPrimaryFW (candidateSet, partyList)
if LEADER == TRUE then {leader possesses negotiation token}
    for all party such that party ∈ partyList do
        sendMsg(preferredOrderFW(candidateSet), party) {ranked version of candidateSet}
    end for
    NEGOTIATION ← INTERMEDIATE
    return waitForReplies(candidateSet, partyList)
else {Does not possess negotiation token}
    if NEGOTIATION == INITIAL then
        for party such that party ∈ partyList ∧ party == LEADER do
            sendMsg(preferredOrderFW(candidateSet), leader)
        end for
        NEGOTIATION ← INTERMEDIATE
    else if NEGOTIATION == INTERMEDIATE then
        if membership(candidateSet) == DIFF then
            for party such that party ∈ partyList and party == LEADER do
                sendMsg(preferredOrderFW(candidateSet), leader)
            end for
        else
            for party such that party ∈ partyList and party == LEADER do
                sendMsg(acceptedOrderFW(candidateSet), party)
            end for
            NEGOTIATION ← FINAL
        end if
    else {NEGOTIATION == FINAL}
        NEGOTIATION ← INITIAL
        return candidateSet
    end if
end if
```

current state of the specific connection, is input for the function. In this dissertation, a connection is a high-level model of the user's communication intent comprising a set of users and a set of features(example media). Reconfiguration will generate re-invitations to the set of users in the connection and re-enabled the set of media in the connection.

Algorithm 5.2 presents the peer negotiation algorithm, negPrimaryFW. The algorithm leverages a three phase protocol and a majority function to support agreement amongst peers on a communication services framework. Possession of the negotiation token is required to initiate a request for consensus with the initiator in the role of leader. There are three states for negotiation defined by the enumerations *INITIAL*, *INTERMEDIATE* and *FINAL* with *INITIAL* being the default starting state for

a communication. Steps 2 - 7 are actioned by the initiator while steps 8 - 23 are actioned by the other peers in the communication.

Phase 1: *INITIAL* The leader sends to each peer in the communication a ranked version of the *candidateSet* (step 3 - 4) with the preferred ranking based on the user's policies in the policy repository. The leader will then wait for the replies (`waitForReplies` at step 7) from all peers. Each peer on receiving the condidateSet, will also reorder the candidateSet from the perspective of the specific peer's set of user policies. Each peer returns a preferred ranked candidateSet to the leader, step 9 - 11.

Phase 2: *INTERMEDIATE* The *INTERMEDIATE* stage for the leader is elaborated in the `waitForReplies` algorithm (see Algorithm 5.3). After receiving all the peer replies, steps 2 - 4 of Algorithm 5.3, the candidateSets are compared. If the candidateSets are the same, step 6, the set is sent to all peers to all peers and the state of negotiation moves to *FINAL*. If differences exist between the peer candidateSets, a majority function is used to produce a new proposed candidateSet (step 13 - 15). The new proposed candidateSet is sent again to the peers.

For a peer, receipt of a candidateSet in this phase will only be checked for consistency of membership against the previous version of the candidateSet, step 13 of Algorithm 5.2. This is done in case the set of communication services frameworks changes, in which case the user preferences will need to be re-evaluated against the new candidateSet, step 14 of Algorithm 5.2. If the membership is consistent, then an annotated candidateSet is returned to the leader to denote acceptance (step 16 of Algorithm 5.2).

Phase 3: *FINAL* A final confirmation of the agreed ranking of the candidateSet is sent to all peers, see Algorithm 5.3. The algorithm ends by returning the candidateSet to the calling algorithm, Algorithm 4.1.

Algorithm 5.3 WaitforReply Algorithm.

```
waitForReplies(candidateSet, partyList)
while numOfReplies < sizeOf(partyList) do
    wait()
end while
if NEGOTIATION == INTERMEDIATE then
    if diffAllReplies == SAME then
        for all party such that party ∈ partyList do
            sendMsg(candidateSet, party)
        end for
        NEGOTIATION ← FINAL
        waitForReplies(partyList)
    else
        for all party such that party ∈ partyList do
            sendMsg(majorityFunct(candidateSet), party)
        end for
    end if
else
    for all party such that party ∈ partyList do
        sendMsg(candidateSet, party)
    end for
    NEGOTIATION ← INITIAL
    return candidateSet
end if
```

The algorithms discussed have been realized in the logic of the AOM. The designs and implementations are presented in the next subsection.

5.2 High Level Design

The list of self-management properties continues to expand as researchers identify new behavior [39] for systems that will either directly or indirectly support the autonomic concepts. The expanding and evolving nature of self-* behaviors reinforced the need for an extensible design vertically (to add other self-* behavior) and horizontally (to extend self-management to the upper layers of CVM). The approach used for supporting vertical extensibility in this work is by laterally stacking components that share a managed resource. Liu et. al [44] specifies an autonomic component's interface as three ports: *functional* - traditional program inputs and outputs; *control* - sensors and effectors; and *operational* - rule injection and rule management. This concept is utilized in the design of the NCB autonomic architecture shown in

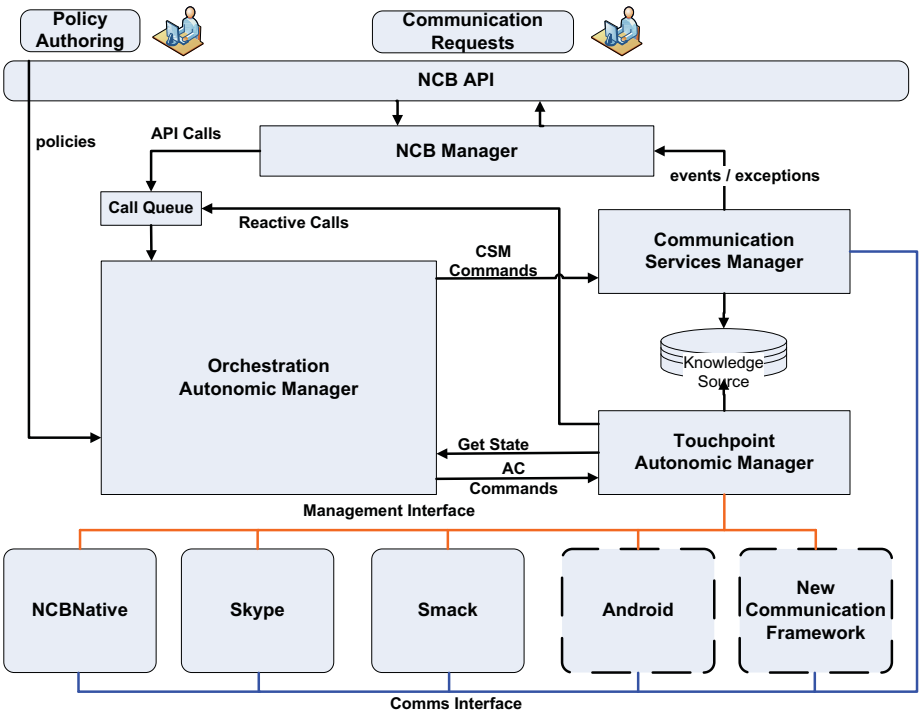


Figure 5.2: NCB Autonomic Architecture.

Figure 5.2. At the top right of Figure 5.2 is the *Communication Services Manager* (CSM) which utilizes the functional port of the communication frameworks (bottom of Figure 5.2) to provide communication services such as creating a connection or enabling a particular medium. The communication frameworks also include a control port, indicated as management interface shown above the frameworks in Figure 5.2, which is used by the *touchpoint autonomic manager* (TPM). TPM directly interacts with the sensors and actuators of the communication frameworks providing low-level management to the resource.

It is acknowledged that conflicts can occur between standard application execution and adaptive behavior [44]. For highly multi-threaded and asynchronous systems such as communication, coordination becomes even more challenging. The approach presented in this dissertation reduces the potential for such conflicts through the use of a high level coordinator which is refer to as the *orchestration autonomic manager* (OAM). The OAM (left in Figure 5.2) provides safe access to the shared managed resources by monitoring the states of the CSM and TPM components. The OAM can

delay sending commands to a component as well as cause the component to suspend pending actions while the other completes a non-concurrent task.

The OAM also has responsibility for evaluating requests, retrieved from the *call queue* (top left of Figure 5.2), with respect to stated policies. The OAM is the main policy decision point in the NCB (recall Figure 4.7 (b) (i) and (ii)) with requests triggering the evaluation process. A request is one of two forms. The first is a user's explicit desire for service such as '*video conference with Bob, Mary and me*'. The second form can be out-of-band events that fall outside the scope of the TPM. An example of such an event would be '*Failure in Skype framework*'. The TPM detects the failure but it would be the responsibility of the OAM to identify all the connections that were using this framework, select an alternative and restore the state on the new framework. Session creation and addition of parties lie in the functional port, so OAM will need to regenerate the required commands (based on the connection's previous state) for CSM to restore the service for the connection. With the TPM escalating these events to the OAM, the OAM therefore serves as a high level autonomic manager.

Figure 5.3 shows the state machine representing the dynamic behavior for the CSM. Execution starts with the invocation of the `loginAll` method that puts the CSM in the `READY` state. This causes initial transitions to `UPDATED_KNOWLEDGE` state, as policies are loaded via `UPDATED_POLICIES` and the knowledge is updated with the inventoried communication frameworks. The knowledge will continue to be updated as policies are added and modified and changes occur within the communication frameworks. The `getCapablities` method uses this knowledge to return a list of available services with elements derived from any of the supporting communication frameworks. This supplied list of services defines the user's communication space by listing all the available forms of communication at that instance of time.

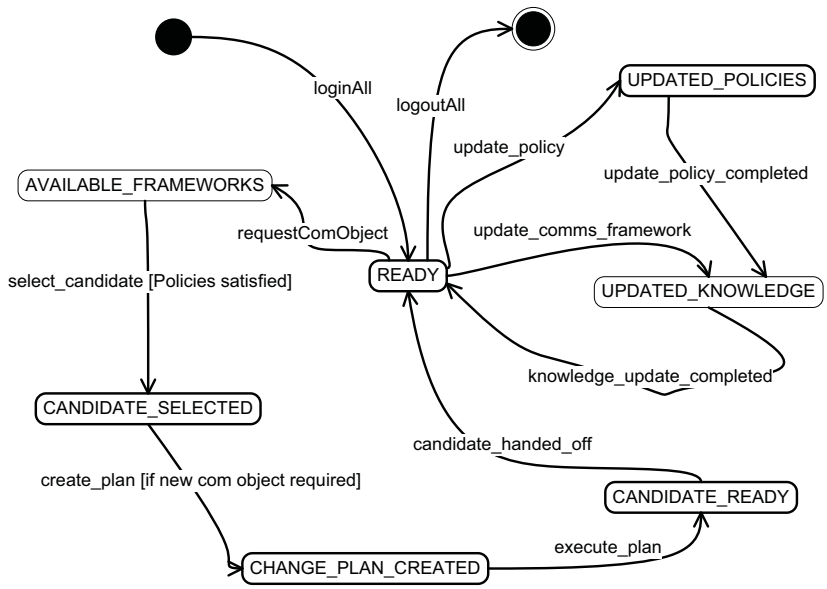


Figure 5.3: CSM State Machine.

The `getCommunicationObject` generates a `requestComObject` and transitions to `AVAILABLE_FRAMEWORKS` after getting the list of inventoried communication frameworks. A candidate is then selected from the list that does not violate the active policies of the CSM and transitions to the state `CANDIDATE_SELECTED`, see Figure 5.3. A change plan is generated and the state transitioned to `CHANGE_PLAN_CREATED`, if the candidate communication framework is already being used then the plan is empty. The plan is executed to effect the directives of the change plan moving to the `CANDIDATE_READY` state and the NCB uses the returned communication framework to realize the communication model that was invoked by CVM.

5.3 Detailed Design

Figure 5.4 depicts the revision to King et al. [40] *Reusable Autonomic Manager* design. As with the original design, the MAPE functionalities (*Monitor*, *Analyzer*, *Planner* and *Executer*) are encapsulated in individual classes with each class threaded for inde-

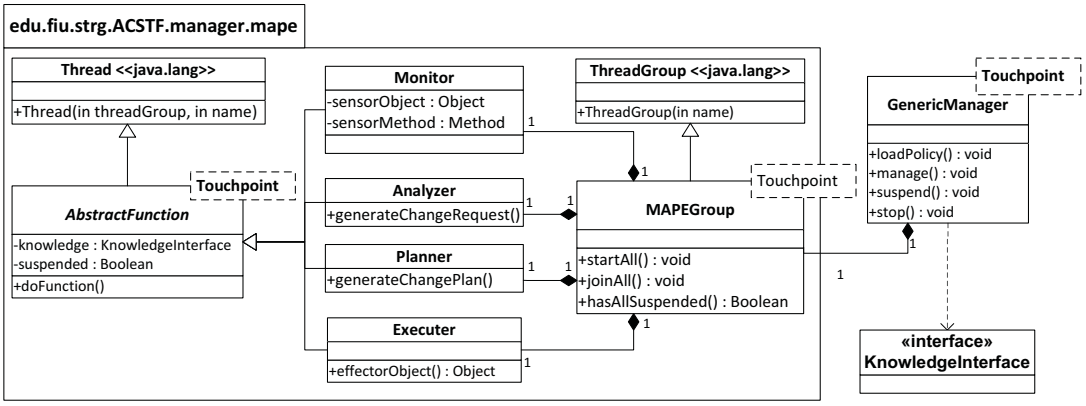


Figure 5.4: Reusable Autonomic Design.

pendence of operation. Each thread can be individually started, stopped, suspended or resumed giving fine grained control of the loop. The revised design includes a thread group, *MAPEGroup* in the *edu.fiu.strg.ACSTF.manager.mape* package in Figure 5.4, which provides the mechanisms to safely suspend and resume the MAPE threads as a single unit for coarse grained control (recall the discussion in Section 5.2 where the OAM may need to suspend MAPE activities). The controller class *GenericManager*, see top right of Figure 5.4, coordinates the operations and services as well as maintain the knowledge source (*KnowledgeInterface*, bottom right of Figure 5.4) used by the MAPE functions. *GenericManager*, *MAPEGroup* and the MAPE classes (through inheritance from *AbstractFunction*, bottom left of Figure 5.4) are parameterized with the template class, *Touchpoint*. Self-management will be carried out by classes representing the *Touchpoint* template class.

Figure 5.5 shows the core components of the new NCB Design, a refinement of Figure 5.4. At the bottom left of Figure 5.5 is the package *edu.fiu.cvm.ncb.tpm* which contains *CommTPManager*, the new specialization of the *GenericManager*. *CommTPManger* is parametrized with a communication specific touchpoint, *CommFWTouch*, which includes monitor methods such as `checkFW` - which iterates through the set of frameworks, polling each for error state; and execute methods such as

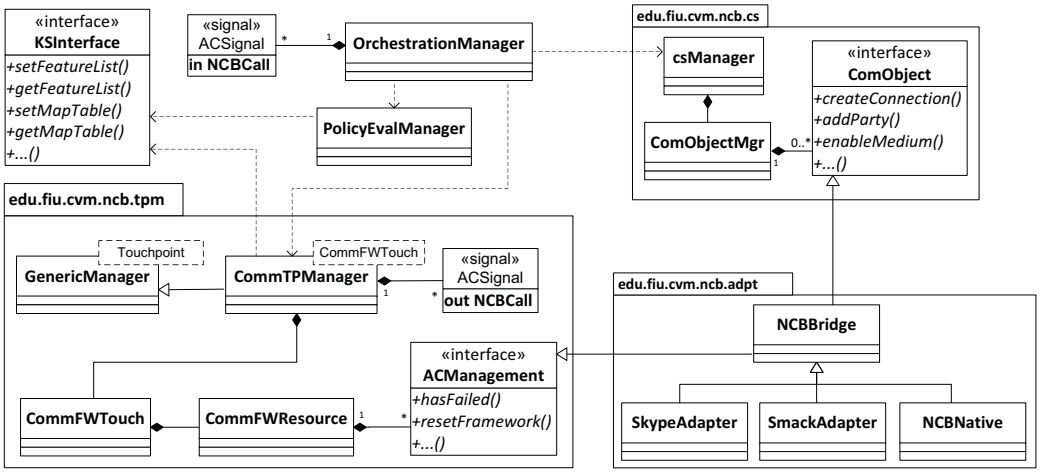


Figure 5.5: NCB Detailed Design Diagram.

`resetFramework` - which reinstantiates a specific framework. Out-of-band events that cannot be handled by the *CommTPManager* are encapsulated and sent out asynchronously via *ACSignal*, to be handled by the *OrchestrationManager*.

Shown at the top right of Figure 5.5 is the package *edu.fiu.cvm.ncb.cs* which has the components responsible for managing the communication services (CSM). The *ComObject* interface defines the operations for the communication framework such as `createConnection`, which creates a new communication framework session for a specific connection identifier, and `addParty`, which adds a new member to a communication session. The *ComObjectMgr* maintains the set of available communication frameworks, while the *csManager* handles the high level coordination to effect the communication.

Package *edu.fiu.cvm.ncb.adpt* at the bottom right in Figure 5.5 includes the adapter classes for the communication frameworks. For black-box communication frameworks, such as Skype, the adapter classes function as managed resource wrappers. SmackAdapter is the managed resource wrapper that reuses the open source Smack libraries, and NCBNative is the original NCB as proposed by Zhang et al. Each adapter class implements the common sensor, effector and communication methods of the *NCBBridge* interface. The *NCBBridge* is an amalgamation of the functional and control

ports defined by the *ComObject* interface of the *edu.fiu.cvm.ncb.cs* package and the *ACManagement* interface of the *edu.cvm.ncb.tpm* package.

The *OrchestrationManager*, shown at top center of Figure 5.5, coordinates the use of the shared resources by autonomic (TPM) and non-autonomic (CSM) components. Additionally, the use of the knowledge source by the two components (specifically the limits on the access to the MappingTable as a *reader-writer lock pattern*) adds to the safeness. CSM uses the MappingTable (`getMapTable` method of the *KSInterface*, at the top left of Figure 5.5) as read-only via indirection through the OAM, allowing the OAM to monitor and queue when necessary the access to the shared resource. Tthe TPM is responsible for updating the table using locks when writing to the table. The *PolicyEvalManager*, below the *OrchestrationManager* in Figure 5.5, is the high level policy decision point. Implementations of the algorithms described in Chapter 4 are utilized in the *PolicyEvalManager* to produce a candidate communication framework.

5.4 Implementation Details

The architecture and designs for the autonomic NCB described in the previous sections have been implemented as a prototype. A subset of the API[1] for the autonomic NCB is presented in Table 5.1 representing the interface available to the upper layers of the CVM. The `login` method returns an object (UserObject) containing profile information such as user roles, saved schemas, and contact lists. The profile information is reused by the upper layers of CVM including the population of the displayed user interface. The `createSession` method creates a connection-to-session object, providing a mapping to the high level concept of a connection to one or more communication framework sessions. The `addParty` method adds persons from the user's contact list to a previously created session. Streaming media or non-streaming data

[1]The full API is available as JAVADocs at http://www.cis.fiu.edu/cml/

Method Summary	
void	`addParty`(String sessionID, String participantID) This function adds the participants specified to the specific session.
void	`removeParty`(String sID, String participant) This method removes the list of participants from the given session.
void	`createSession`(String sessionID) This function creates a session with the specific session ID
void	`destroySession`(String sessionID) This function destroys the specified session ID
void	`disableMedium`(String connectionID, String mediumName) Stop sending the specified medium to all the participants in the connection
void	`enableMedium`(String connectionID, String mediumName) Start sending the specified medium to all the participants in the connection
UserObject	`login`(String userName, String password) This method will attempt login of the given user
void	`logout`(String userName) Logs the user out.
void	`resetNCB`() Resets the ncb instance.
void	`applyPolicy`(String policy) Add and apply specified policy to the NCB.

Table 5.1: Sampling from NCB JavaDocs.

is sent to members of a session by the `enableMedium` method, while policies are introduced by the `applyPolicy` method. Each method invocation is added internally to a priority call queue and applied to a specific connection object during the evaluation process to produce a best fit selection of communication framework.

Table 5.2 shows some of the static metrics from the implementation which was collected using the Eclipse Metrics 1.3.8 plugin. As discussed in Section 5.3 the autonomic NCB exploits the reusable autonomic manager of King et al. [40] (referred to as ACSTF in Table 5.2). The reusing of ACSTF meant NCB needed only 4 additional classes to support self-configuration, the collection of communication frameworks (`CommFWResource`) that extends `AbstractResource`; the touchpoint that uses the resource and extends `CommFWTouch`; and the entity responsible for the management of the touchpoint, the `CommTPManager` which extends the `GenericManager` class. Additionally, the `OrchManager` also extends the `GenericManager` class as it manages the `CommTPManager` and the `CSManager` as discussed in the previous section. Subsequent self-* behavior can easily be included with the addition of another ex-

	NCB	ACSTF	Total
Total Lines of Code	9461	1091	10552
Number of Packages	27	5	32
Number of Interfaces	6	1	7
Number of Classes	156	17	173

Table 5.2: Metrics for NCB and ACSTF.

tension of the `GenericManager` class. Of the 156 classes in NCB, 47 are within the adapter package and an additional 28 relates to events and event handling. Total lines of code for the full autonomic NCB inclusive of the ACSTF was 10552.

5.5 Summary

In this chapter we presented the self-configuring user-centric communication framework. The integration of autonomic computing support coupled with the user-centric focus through user defined policies was presented. The extensibility of the design in terms of ease of addition of new communication frameworks and expansion of self-* behavior was also highlighted in this chapter.

CHAPTER 6

EVALUATION

I have implemented a prototype of the autonomic NCB that demonstrates the feasibility of the approach and incorporates the designs discussed in the previous chapters. In this chapter, I present the evaluation of the approach with respect to objectives 2 and 3 as presented in Section 3.3. For completeness, the author notes that objective 1 was evaluated through the results of the feature oriented domain analysis in Section 4.2. The policy shown in Figure 4.6 was used in all experiments. In this chapter I outline our experimental setup and present the results of the experiments. A discussion of the results including threats to the validity of the experiments is also presented.

6.1 Evaluation Goals

The goals of the evaluation are reflective of the objectives and criteria defined in Chapter 3. The criteria guided the design of the set of experiments discussed in this chapter. The evaluation of the implementation was based on the following hypotheses

- the autonomic NCB has minimal overhead over baseline communication frameworks;
- the automated reconfiguration time is less than the manual reconfiguration time for cross communication services framework switching.

The prototype was evaluated with respect to its efficiency (compared to previous versions of CVM) and the scalability of the selection mechanism.

6.2 Experimental Setup

I used three versions of the NCB prototype during the experimental runs, where each run was composed of twenty iterations of the specific scenario. The highest and lowest reading were thrown out and the remaining eighteen averaged. The first two versions were based on a single-framework developed prototype NCB without self-configuring behavior: one version was configured to only use the Skype version 4.2 communication framework API, and the other was configured to only use the Smack version 3.0.4 communication framework API. The third version was the new prototype of the autonomic NCB with Skype 4.2 API and Smack 3.0.4 API as the supporting communication frameworks. A common driver application was used to simulate CVM type requests for all experimental sets.

Environment: All computers used during the experiments met the following specifications: Pentium 4 3.00GHz, 1 GB RAM, web camera and microphone with 100 Mb Ethernet Adapters. Windows XP SP3, Java 1.6.0_18 and Eclipse Version 3.5.2 were installed on all computers. TPTP version 4.6 [21], a Java profiling tool, and windows' perfmon, the windows performance monitoring tool, were the data gathering tool used during the evaluation process.

6.3 Experimental Set 1 - Two-way Video Conference: A Comparative Analysis

Purpose: This experimental set assess the overhead of the autonomic NCB with respect to the reused communication frameworks. The evaluations are based on a two way video communication. The evaluation is applied to the autonomic NCB, Skype native windows client and a Smack native implementation. The metrics used in this evaluation are memory utilization, processor utilization and data transmissions. Data transmission was included since the autonomic NCB has to negotiate for consensus amongst the parties involved in the communication, indication of the impact of this

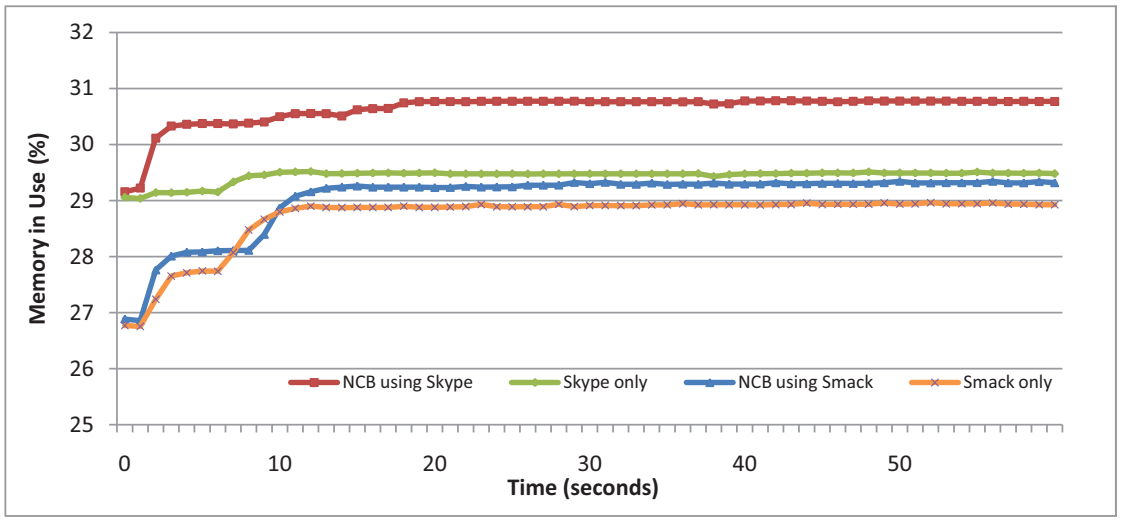


Figure 6.1: Analysis of Memory Usage for Two-way Video Conference.

negotiation is therefore valued. The data used in the analysis was gathered using perfmon. The specification for the computer used for the performance monitoring was a Dual Core 2.33 Ghz processor with 3 GB of RAM and a 100 Mb Ethernet adapter. Additionally, an analysis of the prototype's autonomic detection and reconfiguration times is presented. It is expected that some reconfiguration activities will be dynamically driven by changes in the underlaying layers, such as failure and faults. A initial look and analysis of the cost of the autonomic aspects is presented as well.

With the additional indirection, it was expected that additional memory, processor and network bandwidth resources would be needed to support the additional processes. The expected overhead for memory should be within five percent of the base communication services frameworks based on the specifics of the design and the implementation. While the processor utilization and additional data transmission overhead should be within twenty percent of the base communication services framework with the autonomic polling responsible for most of this overhead. We calculate the percentage difference as $\%DIFF = |\frac{x1-x2}{(x1+x2)/2}| * 100$.

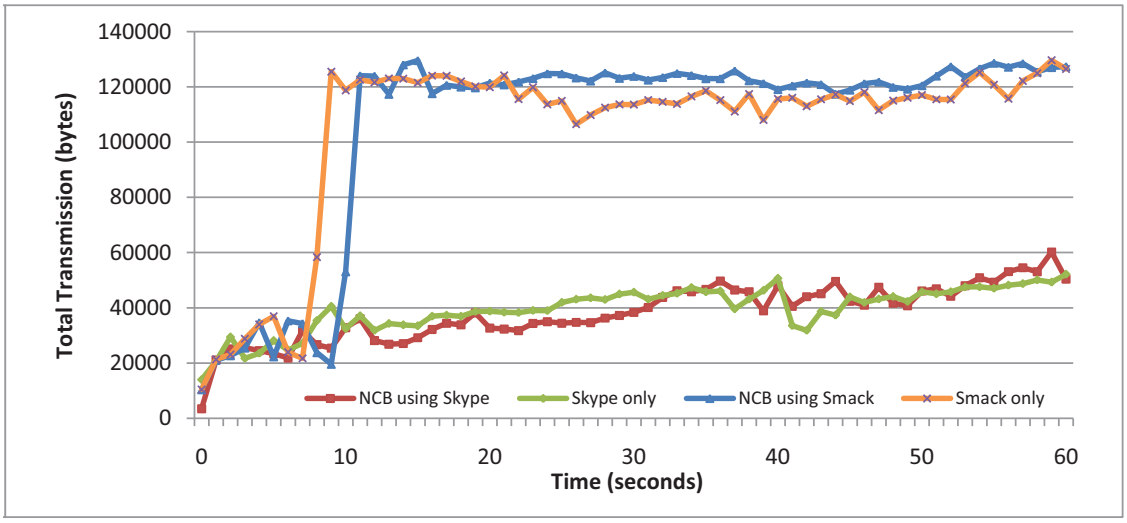


Figure 6.2: Analysis of Data Transmission for Two-way Video Conference.

Results: Figure 6.1 shows the average processor utilization for the two way video scenario. The y-axis shows the percentage of the processor utilized while the x-axis shows the time within the scenario run for completing each scenario in the experiment. The lines represents the performance of the reused communication services frameworks (Skype only and Smack only) clients and the NCB prototype either reusing Skype or Smack. A comparative look at the Smack versus NCB with Smack shows that on average, there is a 1.37% increase in memory utilization by NCB over that of the Smack native client. NCB using Skype, however shows a 4.31% increase in memory utilization over the Skype native client.

While the difference is less than our expected maximum memory utilization, we conducted a trace to identify reasons for the differences in comparative percentages. Our investigation shows additional intermediary objects used for the NCB to Skype communication. Skype is a C++ library, thus requiring the use of JNI wrappers to facilitate the communication which results in the additional memory usage. Smack on the other hand, is native Java.

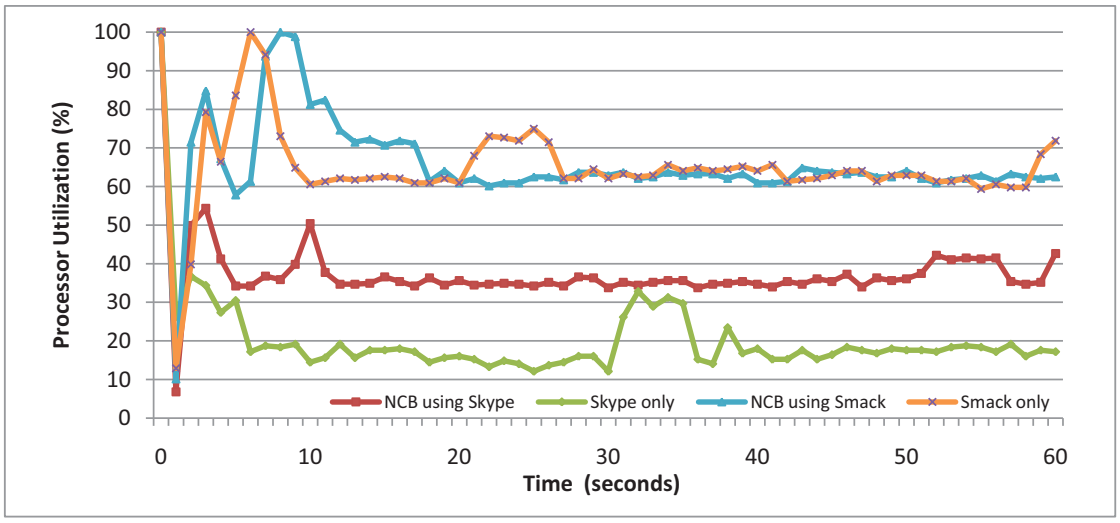


Figure 6.3: Analysis of Processor Utilization for Two-way Video Conference.

average bytes transmitted at discrete points in the experiments are shown in Figure 6.2. The y-axis shows the transmission in bytes with the x-axis showing the time within the scenario run for completing each scenario in the experiment. Again, as in the previous experiment, the lines represents the performance of the reused communication services frameworks (Skype only and Smack only) clients and the NCB prototype either reusing Skype or Smack. Calculation of the worst case difference in the graph for the Smack versus NCB with Smack shows a 18.18% additional data transmission with an on average value of 9.7%. NCB using Skype showed a smaller difference than the Smack comparisons in this experiment. The comparitive difference between the native Skype client and the NCB with Skype was 12.17% for the worst case and 7% on average.

Figure 6.3 shows the graph of processor utilization at intervals during the experiments. the percentage of cup utilized is marked on the y-axis and the x-axis shows the time within the scenario run for completing each scenario in the experiment. The largest percentage difference between the Skype and the NCB using Skype was calculate to be 73%. This is quite significant and out traces reveals two heavily used

loops in the Java interfacing implemtation for Skype. The first loops for attaching to the WIN32 library for Skype and the second loops when it awaits confirmation on commands sent to the Skype library. The Smack and NCB with Smack peaked at 28.57% for processor utilization differences with an average of 15.38%.

We have found that the autonomic monitoring influences the relatively high CPU utilization. The trace and analysis of the autonomic components will be discussed later in this chapter. An interesting result of the experiments in this section is the clearly visible measured delay introduced by the indirection of NCB. In Figure 6.1 and even more so in Figure 6.3 is an offset of approximately two seconds for the graph of the NCB compared to the graphs for Smack and Skype. This delay represents the time for NCB to process the user request, evaluate the potential communication services frameworks and then prepare the framework for use. Optimization of the implementation should potentially reduce this delay.

6.4 Experimental Set 2 - Analysis of Candidate Selection Algorithm

Purpose: I evaluated the technique for selecting the candidate communication framework using the new NCB prototype. I measured the elapsed time from receiving a user's request to the returning of a candidate communication framework. Varying numbers of communication frameworks were included in the pool for selection during the experimental runs and each run was composed of ten iterations of a specific pool size. The experimental runs for each pool size was then averaged and tabulated. The initial hypothesis with respect to the current algorithm is that at a worst, the selection algorithm employed scales polynomially. While the current algorithm employed is not an optimized algorithm, the expectation is that it performs at worst in polynomial time. The reader is reminded that the design easily supports the replacement of the selection algorithms.

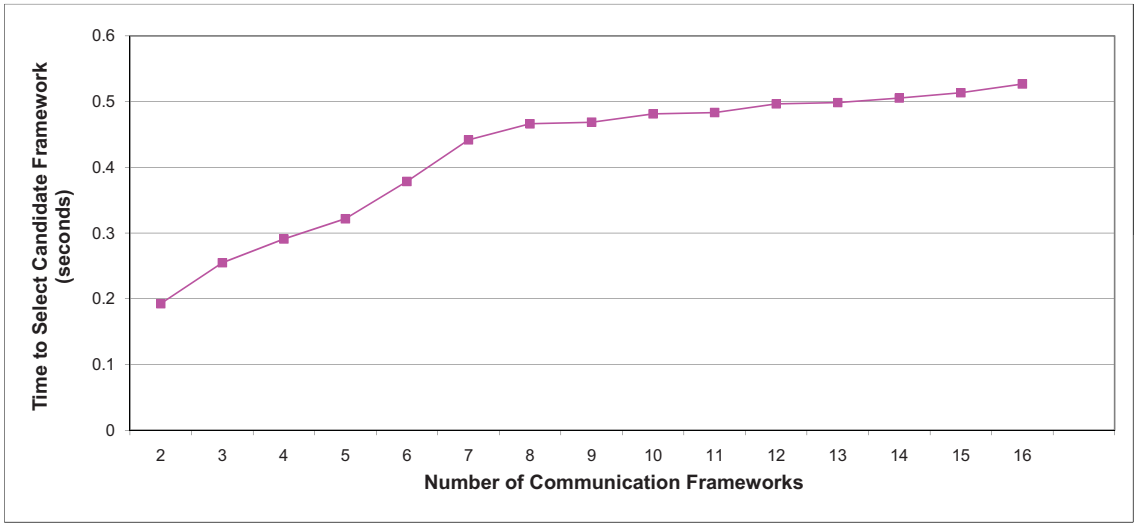


Figure 6.4: Average Times to Select Candidate Communication Framework.

Results: The pool sizes range from 2 to 16 and the averages for the runs are shown in Figure 6.4. The graph represents the average time for the selection process with respect to the number of communication frameworks available. The x-axis shows the number of communication frameworks available at the start of the selection process while the y-axis shows the time taken to complete the policy evaluation and return a candidate communication framework. The graph shows a non linear increase in the selection times with respect to the increase in the communication framework pool. The predicted increase was a typical linear increase based on the time complexity analysis of $2n - 1$ for the recursive algorithm. However, a detailed review of the data shows a maximum combination of the feature attributes used in the experiments of six. Therefore, multiple communication frameworks are guaranteed to be removed at this threshold. On average a reduction in the set of communication frameworks occurs with each feature attribute evaluated. Note however, that a request that can be satisfied by all communication frameworks in the set may not exhibit this behavior.

6.5 Experimental Set 3 - Audio to Audio-Video Conferencing Reconfiguration

Purpose: In the event a decision on function, price or performance warrants a change of framework, it is expected that the proposed automated approach will be comparatively faster than a manual switching process. The three scenarios (2-way, 3-way, 4-way) measured the time to setup the configuration to serve a user's request for an initial audio conference and then switch the configuration to video conferencing.

Results: Figure 6.5 shows the average time for completing each scenario in the experiment. The cluster of bars represents the performance of the non-autonomic (Skype only and Smack only) variants and the autonomic variant of the NCB. The y-axis is the measured time in seconds for the experiment. The bar representing the non-autonomic variant includes the times for starting the Skype version of NCB initially, stopping and starting the Smack version of NCB if necessary. The x-axis shows the three scenarios (2-way, 3-way and 4-way), while the y-axis shows the time to complete the switch and setup of a video conference. The non-autonomic implementations of NCB performed better than the autonomic variant for two way audio conferencing to video conferencing. From our analysis, this was due to Skype's support for two way video conferencing. Therefore there was never a need to switch frameworks. The autonomic version of NCB, however, had the additional overhead of the AC framework threads.

It is shown in Figure 6.5 that as the number of participants increase the autonomic NCB scales better than the non-autonomic NCB. Skype video conferencing support is limited to two way. As such, the Skype implementation of NCB required a switch to the Smack implementation of NCB. For the single framework implementation, this requires conference disconnection, shutdown and a restart with the new framework implementation. The time taken to manually stop and start an implementation was averaged with the highest and lowest values discarded to compensate for differences in the speed of users in the experiments. The author also noted from the data that

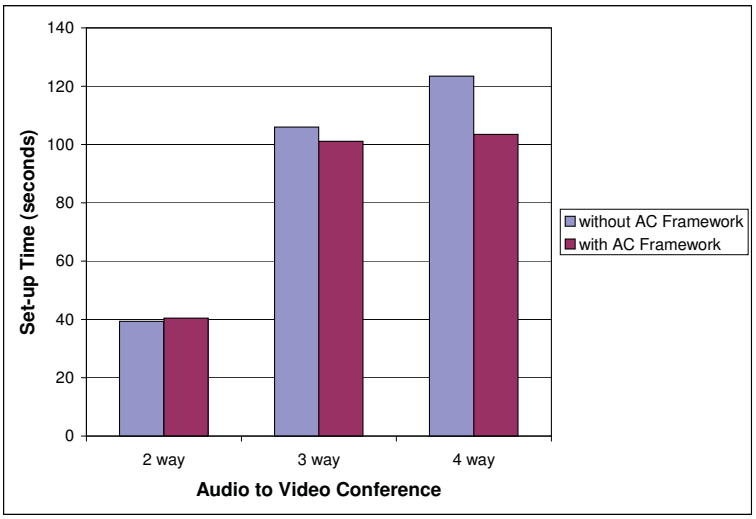


Figure 6.5: Audio to Video Conferencing Set-up Times.

the AC Framework accounted for noticeably less of the overall execution time as the conference's participant count increased.

6.6 Experimental Set 4 - N-way Audio Conference Configuration

Purpose: The scenarios (n-way and n+1-way audio up to 5-way) measured setting up the configuration to serve a user' request for an initial audio conference and to switch to N+1 way for the experimental implementations. A fourth implementation (Asterisk-only) was added to the initially described three implementations (Skype-only, Smack-only and AC NCB).

Results: The averages for each scenario in the experiment are displayed in Figure 6.6. The graph shows the average time inclusive of startup, negotiation and establishment of the streams. The implementation with the best responsive time was Asterisk-only. Our investigations revealed that this was a result of the client-server nature of the implementation compared to the peer-to-peer architecture of the other implementations. Negotiation time was minimal, as the initiator simply created a 'room' on the asterisk server and then sent invitations with the room number to the

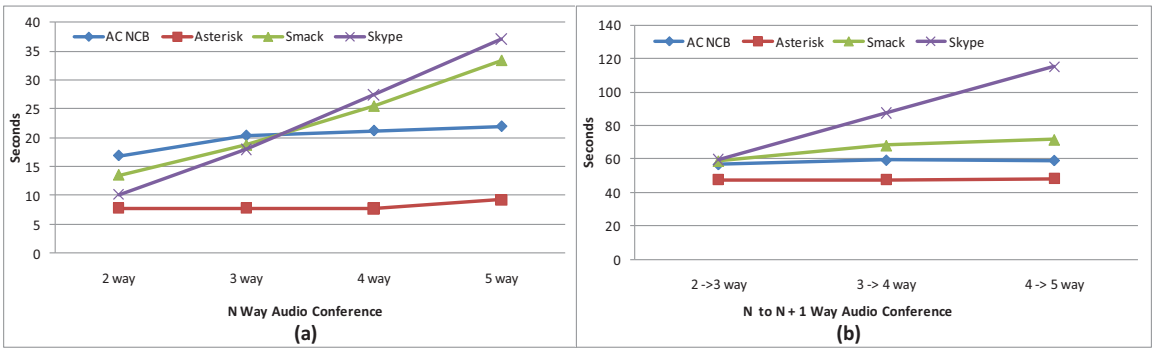


Figure 6.6: Analysis of Audio Configuration Times.

other parties. The AC NCB times included the initialization times for all communication frameworks, which is responsible for the significant difference compared to the lighter Asterisk-only implementation times shown in Figure 6.6(a). Figure 6.6(b) shows the results for starting a conference and then the subsequent addition of one other party to the conference. A 'steady-state' period of 40 seconds was included to simulate a conversation before requesting the additional user. As with the N-way conferences, the AC NCB implementation mirrored the Asterisk-only implementation in the N+1-way conferences. Asterisk-only was the framework candidate selected to be used by the AC NCB based on the policies used in these scenarios. The Smack-only and Skype-only were linear with respect to the number of parties involved in the conversation as these implementations incrementally added the parties.

The results were further analyzed to see what was the impact of the autonomic components in the time analysis. Figure 6.7 indicates the proportion of time dedicated to the autonomic processes as compared to the communication framework processes. This is on average 26% of the processing time consumed by autonomic processes. A detailed inspection of the results showed the monitor threads responsible for a significant portion of the consumed time. The current implementation of the monitor threads poll at specified intervals. Optimized alternatives to the naive polling implementation can further reduce the time consumed by the autonomic components.

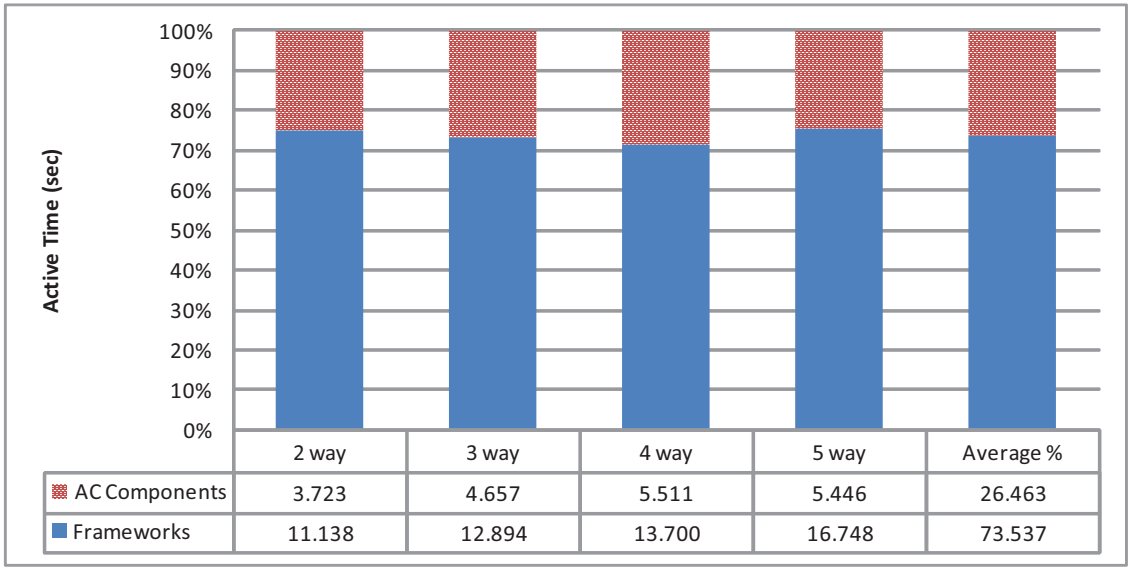


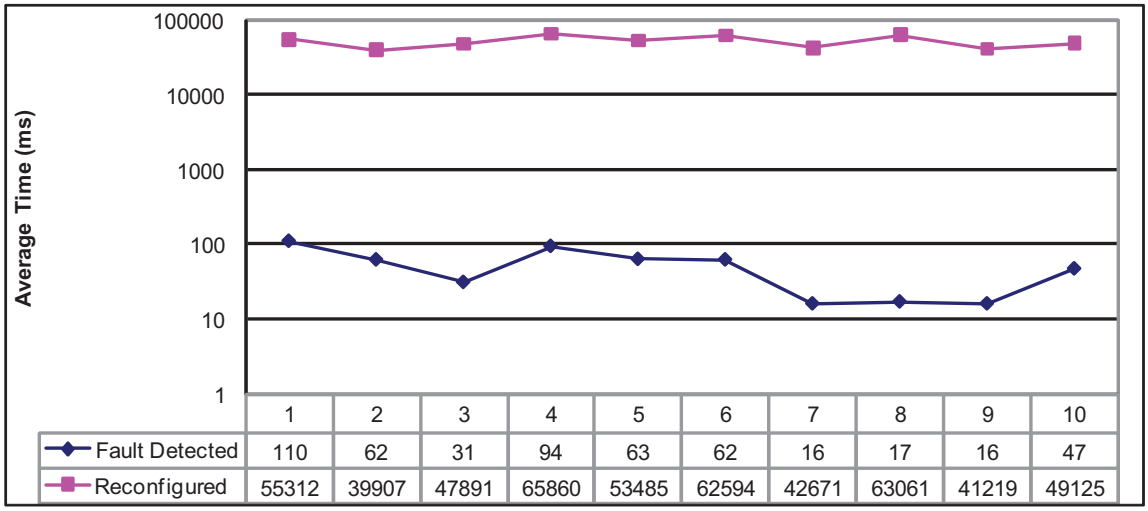
Figure 6.7: Breakdown of Autonomic Components vs Frameworks.

6.7 Experimental Set 5 -Analysis of Autonomic Response Times

Purpose: The purpose of this experiment was to evaluate the responsiveness of the fault detection and adaptation triggering components of the implementation. Each run is an injected fault for an active communication framework for some random but bounded time interval. The difference between the fault injection and fault detection times are recorded, as well as the time difference until reconfiguration occurs.

Results: The performance results for the fault detection and adaptation times are shown in Figure 6.8. The x-axis shows the runs while the y-axis shows the time scale in milliseconds for each of the runs. Using the autonomic framework, faults were able to be discovered at most 110 ms after detection and as early as 16 ms after fault injection. The mean for fault detection was 51 ms. The experiments used a polling interval of 250 ms for the monitoring thread of the MAPE functions for the autonomic framework with a predicted mean of 125 ms.

The ITU-T [33] proposed a one-way delay of less than 400 ms as one of its performance targets for voice and video applications. The significance of the mean detection



	1	2	3	4	5	6	7	8	9	10
Fault Detected	110	62	31	94	63	62	16	17	16	47
Reconfigured	55312	39907	47891	65860	53485	62594	42671	63061	41219	49125

Figure 6.8: Analysis of Detection and Adaptation Time.

time of 51 ms compared to the ITU-T's performance target of 400 ms suggests that the proposed framework can be beneficial to self-healing audio and video applications as repairs could begin before a human can recognize that a fault happened. A comparison of the detection against reconfiguration times is presented in Figure 6.8. It must be noted that there was a substantial difference in times for detection and reconfiguration. Reconfiguration involves stopping and restarting some services, as resources such as cameras and microphones may not support shared access by differing communication frameworks. To the best of our knowledge, there are no generally accepted performance targets for startup or establishing audio and video application.

6.8 Discussion

The prototype of the NCB is an initial proof of concept for the introduction of extensible services to the CVM. The prototype NCB currently includes four communication frameworks, Skype, Smack, Asterisk and NCBNative. For Subsection 6.4 experiments, additional communication frameworks were created as stubs to support the simulation. This however does not reflect a limitation of the prototype as additional communication frameworks can be interfaced in the future. The number of active

communication frameworks will be dependent on the limitation of the specific communication device. While it may seem unrealistic to support that many frameworks on a device, This exercise can be viewed as a proof of concept where different elements of concern that may impact the final configuration maybe selected from such a large pool of options.

The experimental set for Subsection 6.7 used a polling interval of 250 ms for the monitoring thread of the MAPE functions for the autonomic framework. This value was derived from earlier experiments aimed at finding a balance between minimizing the effects of the MAPE polling and adequately evaluating system changes. Work on more resource friendly methods such as event driven signaling may provide an alternative to the current implementation's use of polling.

Tools for performance evaluation generally do not introduce significant overhead to the monitored environment. For real-time systems this tends not to be the case as the sampling period can influence the application performance. In our evaluation process, Eclipse Test and Performance Tools Platform was used for monitoring and profiling the prototype. This introduced additional memory and computation overhead, which in turn introduced delays in the participants negotiation process. I compensated for this overhead by reducing the sampling period and focused the profiling and monitoring based on the criteria of the specific experiment.

6.9 Summary

Results of experiments conducted on the prototype was presented in this chapter. The results of the experiments were discussed and threats to the validity of the experiments were highlighted.

CHAPTER 7

CONCLUSION

In this chapter I present a summary of the research conducted in this dissertation. The contributions of the work are revisited as well as the evaluation of the work. A discussion of some future directions of this work is presented in the future work section.

7.1 Research Summary

In this dissertation I investigated the problem of how to provide an always-best-served solution for users of collaborative communication services. The combinatorial mix of large numbers of communication service providers, communication services and platforms coupled with expanding metrics for valuing 'best fit' services presents challenges for user of theses communication services. In my research, I proposed a user-centric solution that alleviates the burden of users having to monitor and manage these communication services by interfacing multiple communication frameworks and policy-driven selection of communication services. The approach included defining user-centric policies for users to describe high-level intent for communication needs. These high-level policies guide the automated selection algorithms for reconfiguration of services to support the user's intent.

I defined user-centric communication policies and outlined the technique for their interpretation. I elaborated on the algorithms to support the automated reconfiguration and runtime selection of candidate communication frameworks. The architecture of the self-configuring framework was presented and the flow of the approach was explained in the context of the design. Details of the design of the self-configuring framework were presented and the extensibility of the architecture to support other autonomic behavior was also discussed.

The evaluation of the self-configuring framework was presented. The evaluation was done with respect to the evaluation criteria discussed in Chapter 3. Objective 2 is primarily addressed in the evaluations, showing the extensibility and adaptability at runtime with the framework. Objective 1 is achieved with the defined policy structure, interpretation mechanism and algorithms and supported by the Analysis of Candidate Selection Algorithm evaluation. Based on the QoS performance recommendations discussed in the paper, we believe however that this approach is also feasible for other collaboration intensive multimedia applications.

7.2 Future Work

The autonomic NCB is positioned to be a useful tool for researchers in collaborative communication and self-managed systems. The design of the autonomic NCB presents a real world application that is extensible, therefore providing a testbed for researchers in many areas. Already there is some preliminary work on integrating self-testing support in the prototype. In the future, the NCB can be extended to include other autonomic behavior, such as self-healing properties to provide recovery options for a collaborative communication session.

Efficient strategies for session recovery on failed communication frameworks or services will aid the resilience of research in this area. The challenge will be providing support for proprietary communication frameworks in the designs for session recovery. The reduction of handover latency will also be an additional issue that needs to be addressed in moving the always-best-served paradigm forward.

The approach proposed includes a layered design, where one of the layer is the convergence layers. Works such as [20] which already provide some form of convergence can also benefit from the application of the concepts in the upper layers to move to an dynamically configurable implementation. In fact, potentially some of these

convergence products could be investigated as possibly alternate implementation of the convergence layer of NCB.

Additionally, we will be investigating the use of models to further improve the performance of the evaluation and generation of new configurations at runtime. The real-time nature of the domain requires minimization of all possible latency points. The use of abstract models to evaluate and potentially enforce through casual connections will be an interesting direction that could add value to other research areas in computer science.

Next Generation Networks (NGN) promises to revolutionize the wireless network. O'Droma et al. [55] outlines a vision that replaces the current Subscriber Based Model (SBM) with a Consumer Based Model (CBM) in 4th Generation wireless telephony. This vision is seen as one of the grand goals for research in such areas as vertical handover technologies from mobile wireless networks to unlicensed spectrum access points such as WiFi hotspots. O'Doma et al. proposes an 'always best connected' (ABC) approach where handover is not only based on a user's location but on other factors such as price and performance. As articulated by O'Droma et al. [29] there are six key aspects for realizing the vision:

1. Access discovery
2. Access selection
3. Authentication, Authorization and Accounting (AAA) support
4. Mobility management
5. Profile handling
6. Content adaptation

The approach presented in this dissertation can provide the ground work for items 2 and 5 in the previous list. Item 2, Access selection, is the process of deciding over

which access network to connect at any point in time. Item 5, refers to solution for supporting application adaptation based on context, resources and metrics in real-time. This is an area that has a lot of promise for potential innovations and will be investigated in the future.

Cloud computing, which is the practice of provisioning computational and storage resources on demand over the Internet, has seen significant adoption and still continues to evolve. Cloud computing provides cheaper computing but is expected to provide even more benefits, with faster, more flexible and mode effective computation [38]. Currently, most cloud computing service providers have some pricing structure that is based on a predefined set of resources. As more options in terms of granularity of service compositions are provided, users of these services will want the flexibility to dynamically compose services not just within a single cloud service provider, but also across providers. Intuitively, this could produce additional cost savings, but would have to address issues of weighting communication latencies between providers, per packet cost for inbound and outbound traffic, cost of resource and cost of migration.

The work described in this dissertation can become a first step in providing such an approach to support the additional dynamism in cloud computing. This work can complement approaches that define an abstraction for cloud computing resource provision and use [38]. Since most pricing is available on the Internet, mining techniques can be employed to gather pricing information. Simple load tests can be used to ascertain bandwidth constrains and utility / cost functions designed to algorithmically inform appropriate service composition.

The Communication Virtual Machine Research Group at Florida International University has already begun to investigate the use of the adaptive approach describes in the dissertation to support energy management. A microgrid is a self contained unit that includes energy loads and sources within a larger smart grid. Specifically,

the group is investigation an approach for managing energy microgrids using models. A modeling language and a modeling environment will be developed for graphically expressing a desired energy architecture. Also a virtual machine will be developed to analysis the modeling artifacts and provide software management of the microgrid hardware. An investigation of how and what will be needed to transfer the conceptual ideas of the NCB from the domain of communication services to that of energy management is currently underway.

BIBLIOGRAPHY

[1] D. Agrawal, Kang-Won Lee, and J. Lobo. Policy-based management of networked computing systems. *Communications Magazine, IEEE*, 43(10):69 – 75, 2005.

[2] Dakshi Agrawal, Seraphin Calo, Kang-won Lee, Jorge Lobo, and Dinesh Verma. *Policy technologies for self-managing systems*. IBM Press, 2008.

[3] Andrew A. Allen, Sean Leslie, Yali Wu, Peter J. Clarke, and Ricardo Tirado. Self-configuring user-centric communication services. In *(ICONS 2008)*, pages 253–259. IEEE, April 2008.

[4] Anne H. Anderson. An introduction to the web services policy language (wspl). *Policies for Distributed Systems and Networks, IEEE International Workshop on*, 0:189, 2004.

[5] Nelly Bencomo, Gordon S. Blair, Carlos A. Flores-Corts, and Peter Sawyer. Reflective component-based technologies to support dynamic variability. In *VaMoS'08*, pages 141–150, 2008.

[6] Philip A. Bernstein. Middleware: a model for distributed system services. *Commun. ACM*, 39:86–98, February 1996.

[7] Paola Boettner, Mansi Gupta, Yali Wu, and Andrew A. Allen. Towards Policy-Driven Self-Configuration of User-Centric Communication. In *(ACM Southeast Conference 2009)*. ACM, April 2009.

[8] Raouf Boutaba and Issam Aib. Policy-based management: A historical perspective. *Journal of Network and Systems Management*, 15:447–480, 2007. 10.1007/s10922-007-9083-8.

[9] Raouf Boutaba, Salima Omari, Ajay Pal, and Singh Virk. Selfcon-an architecture for selfconfiguration of networks. *Journal of Communications and Networks*, 3:317–323, 2001.

[10] Cerulean Studios. Trillian software, Sept. 2009. `http://www.ceruleanstudios.com`.

[11] M. Charalambides, P. Flegkas, G. Pavlou, J. Rubio-Loyola, A.K. Bandara, E.C. Lupu, A. Russo, M. Sloman, and N. Dulay. Dynamic Policy Analysis and Conflict Resolution for DiffServ Quality of Service Management. In *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*, volume , pages 294 –304, april 2006.

[12] Huoping Chen, Salim Hariri, and Fahd Rasul. An innovative self-configuration approach for networked systems and applications. In *(CTS 2006)*, pages 537–544. IEEE, 2006.

[13] Betty Cheng, Rogrio de Lemos, Holger Giese, Paola Inverardi, and Jeff Magee. Software engineering for self-adaptive systems: A research roadmap. 5525:1–26, 2009.

[14] Adrian Colyer, Gordon Blair, and Awais Rashid. Managing complexity in middleware. In *In The Second AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software (ACP4IS), In AOSD 2003*, 2003.

[15] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The ponder policy specification language. In *POLICY '01: Proceedings of the International Workshop on Policies for Distributed Systems and Networks*, pages 18–38, London, UK, 2001. Springer-Verlag.

[16] Yi Deng, S. Masoud Sadjadi, Peter J. Clarke, Vagelis Hristidis, Raju Rangaswami, and Yingbo Wang. CVM - A Communication Virtual Machine. *J. Syst. Softw.*, 81(10):1640–1662, 2008.

[17] Yi Deng, S. Masoud Sadjadi, Peter J. Clarke, Chi Zhang, Vagelis Hristidis, Raju Rangaswami, and Nagarajan Prabakar. A communication virtual machine. In *COMPSAC 06*, pages 521–531. IEEE Computer Society, 2006.

[18] Keith Devlin. *The Joy of Sets: Fundamentals of Contemporary Set Theory.* 1993.

[19] Simon Dobson, Spyros Denazis, Antonio Fernández, Dominique Gaïti, Erol Gelenbe, Fabio Massacci, Paddy Nixon, Fabrice Saffre, Nikita Schmidt, and Franco Zambonelli. A Survey of Autonomic Communications. *ACM Trans. Auton. Adapt. Syst.*, 1:223–259, December 2006.

[20] ECF. Eclipse communication framework project, September 2007. `http://www.eclipse.org/ecf/`.

[21] Eclipse Test & Performance Tools Platform Project. Eclipse Test & Performance Tools Platform, September 2010. `http://www.eclipse.org/tptp/`.

[22] Wolfgang Emmerich. Software engineering and middleware: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 117–129, New York, NY, USA, 2000. ACM.

[23] Mohamed Fayad and Douglas C. Schmidt. Object-Oriented Application Frameworks. *Commun. ACM*, 40(10):32–38, October 1997.

[24] Robert France and Bernhard Rumpe. Model-driven development of complex software: A research roadmap. In *2007 Future of Software Engineering*, FOSE '07, pages 37–54, Washington, DC, USA, 2007. IEEE Computer Society.

[25] V. Ghini, P. Salomoni, and G. Pau. Always-best-served music distribution for nomadic users over heterogeneous networks. *Communications Magazine, IEEE*, 43(5):69 – 74, May 2005.

[26] Google. GoogleTalk, Nov. 2010. `http://www.google.com/talk/`.

[27] David Gorton. Transforming the customer experience with user centric networking. 2008.

[28] X Gu, J. Strassner, J Xie, L.C. Wolf, and T. Suda. Autonomic Multimedia Communications: Where Are We Now? *Proceedings of the IEEE*, 96(1):143 –154, jan. 2008.

[29] E. Gustafsson and A. Jonsson. Always best connected. *Wireless Communications, IEEE*, 10(1):49 – 55, feb. 2003.

[30] S. Hallsteinsen, M. Hinchey, Sooyong Park, and K. Schmid. Dynamic software product lines. *Computer*, 41(4):93 –95, 2008.

[31] IBM Autonomic Computing Architecture Team. An architectural blueprint for autonomic computing. Technical report, IBM, Hawthorne, NY, June 2006.

[32] ITU-T. Y.2001 General Overview of NGN. International Telecommunication Union, Dec. 2004.

[33] ITU-T SG 12. G.1010 - Multimedia QoS/Performance requirements . International Telecommunication Union, Nov. 2001.

[34] Jive Software. Smack API, Nov. 2008. `http://www.igniterealtime.org/projects/smack/`.

[35] Hiroaki Kamoda and Krysia Broda. Policy Conflict Analysis Using Free Variable Tableaux for Access Control in Web Services Environments. In *In Policy Management for the Web Workshop*, pages 5–12, 2005.

[36] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, Carnegie-Mellon University Software Engineering Institute, November 1990.

[37] J.O. Kephart and D.M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–52, Jan. 2003.

[38] Y. Khalidi. Building a cloud computing platform for new possibilities. *Computer*, 44(3):29 –34, march 2011.

[39] Tariq M. King, Djuradj Babich, Jonatan Alava, Ronald Stevens, and Peter J. Clarke. Towards self-testing in autonomic computing systems. In *8th International Symposium on Autonomous Decentralized Systems(ISADS '07)*, 2007.

[40] Tariq M. King, Alain E. Ramirez, Peter J. Clarke, and Barbara Quinones-Morales. A reusable object-oriented design to support self-testable autonomic software. In *SAC*, pages 1664–1669, 2008.

[41] Philippe Lasserre and Dennis Kan. User-centric interactions beyond communications. Alcatel Telecommunications Review, 2005. `http://alcaesd-f.nl.francenet.fr/docs/1/S0503-UCBB_interactions-EN.pdf` (March 2007).

[42] D. Lewis, T. O'Donnell, K. Feeney, A. Brady, and V. Wade. Managing user-centric adaptive services for pervasive computing. pages 248–255. IEEE Computer Society, 2004.

[43] Hua Liu and S. HARIRI. A component-based programming model for autonomic applications. In *Proceedings of the First International Conference on Autonomic Computing*, pages 10–17, Washington, DC, USA, 2004. IEEE Computer Society.

[44] Hua Liu and Manish Parashar. A programming system for autonomic self-managing applications. In Manish Parashar; Salim Hariri, editor, *Autonomic Computing: Concepts, Infrastructure, and Applications*, pages 211–235. CRC Press, 2006.

[45] C. Low. Integrating Communication Services. *Communications Magazine, IEEE*, 35(6):164 –169, June 1997.

[46] E. Lupu and M. Sloman. Conflict Analysis for Management Policies. In *Proceedings of the fifth IFIP/IEEE international symposium on Integrated network management V : integrated management in a virtual world*, pages 430–443, London, UK, UK, 1997. Chapman & Hall, Ltd.

[47] M. J. Masullo and S. B. Calo. Policy management: An architecture and approach. In *IEEE First International Workshop on Systems Management*, pages 13–26, April 1993.

[48] P.K. McKinley, S.M. Sadjadi, E.P. Kasten, and B.H.C. Cheng. Composing adaptive software. *Computer*, 37(7):56 – 64, 2004.

[49] Microsoft. Microsoft dynamic systems initiative overview, Feb. 2004. `http://www.microsoft.com/hk/windowsserversystem/mn20581/wp\_dsi.mspx(February2011)`.

[50] Jean J. Moreau, Roberto Chinnici, Arthur Ryman, and Sanjiva Weerawarana. Web services description language (WSDL) version 2.0 part 1: Core language. Candidate recommendation, W3C, March 2006.

[51] Brice Morin, Olivier Barais, Gregory Nain, and Jean-Marc Jezequel. Taming dynamically adaptive systems using models and aspects. In *Proceedings of the*

31st International Conference on Software Engineering, ICSE '09, pages 122–132, Washington, DC, USA, 2009. IEEE Computer Society.

[52] Network Working Group. The COPS (Common Open Policy Service) Protocol: Request for Comments: 2748 Category: Standards Track, Jan 2000.

[53] Network Working Group. Policy Core Information Model (PCIM) Extensions: Request for Comments: 3460 IBM Updates: 3060 Category: Standards Track, Jan 2003.

[54] John R. Nicol, Yechezkal S. Gutfreund, Jim Paschetto, Kimberly S. Rush, and Christopher Martin. How the internet helps build collaborative multimedia applications. In *Communications of the ACM*, pages Vol. 42, No.1,79–85, Jan. 1999.

[55] M. O'Droma and I. Ganchev. Toward a ubiquitous consumer wireless world. *Wireless Communications, IEEE*, 14(1):52 –63, 2007.

[56] ooVoo LLC. ooVoo Developers, Oct. 2010. `http://www.oovoo.com/Developers.aspx?pname=DevelopersMain`.

[57] Open API Solutions. OSA/Parlay, Nov. 2010. `http://www.openapisolutions.com/brochures/OSAParlayOverview.pdf`.

[58] Organization for the Advancement of Structured Information Standards (OASIS). eXtensible Access Control Markup Language (XACML) Version 2.0. Technical report, OASIS Access Control TC, February 2005.

[59] Marty Parker and Don Van Doren. Achieving cost and resource savings with unified communications. Technical report, Microsoft, March 2009.

[60] Pidgin. Pidgin development project, February 2011. `http://developer.pidgin.im/`.

[61] Lucas L. Provensi, Fabio M. Costa, and Vagner Sacramento. Management of Runtime Models and Meta-Models in Meta-ORB Reflective Middleware Architecture. In *Proceedings of the 4th International Workshop Models@Run.time*, volume 509 of *CEUR Workshop Proceedings*, pages 81–88, Denver, 2009. CEUR-WS.org.

[62] Qnext Corporation. Qnext, Sept. 2007. `http://www.qnext.com/`.

[63] Andreas Rasche and Andreas Polze. Configuration and dynamic reconfiguration of component-based applications with microsoft .net. In *Sixth International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'03)*, page 164. IEEE Computer Society, 2003.

[64] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. Technical report, Network Working Group, June 2002. RFC 3261.

[65] S. M. Sadjadi and P. K. McKinley. ACT: An adaptive CORBA template to support unanticipated adaptation. In *Proceedings of the 24th IEEE International Conference on Distributed Computing Systems (ICDCS'04)*, pages 74–83, Tokyo, Japan, March 2004. (acceptance rate 17.7

[66] S. M. Sadjadi, Philip K. McKinley, Eric P. Kasten, and Zhinan Zhou. Metasockets: Design and operation of run-time reconfigurable communication services. *Software: Practice and Experience (SP&E). Special Issue: Experiences with Auto-adaptive and Reconfigurable Systems.*, 36:1157–1178, 2006.

[67] S. Masoud Sadjadi, Philip K. McKinley, and Betty H. C. Cheng. Transparent shaping of existing software to support pervasive and autonomic computing. *SIGSOFT Softw. Eng. Notes*, 30:1–7, May 2005.

[68] S. Masoud Sadjadi and Fernando Trigoso. TRAP.NET: A realization of transparent shaping in .NET. *International Journal of Software Engineering and Knowledge Engineering*, 19(4):507–528, 2009.

[69] S.M. Sadjadi, S. Kalayci, and Yi Deng. A Self-Configuring Communication Virtual Machine. In *IEEE International Conference on Networking, Sensing and Control, 2008. ICNSC 2008.* , volume , pages 739 –744, April 2008.

[70] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.*, 4:14:1–14:42, May 2009.

[71] M. Satyanarayanan. Pervasive computing: vision and challenges. *Personal Communications, IEEE*, 8(4):10–17, August 2002.

[72] Jürgen Sienel, Alberto León Martín, Carlos Baladrón Zorita, Laurent-Walter Goix, Álvaro Martínez Reol, and Belén Carro Martínez. OPUCE: A Telco-driven Service Mash-up Approach. *Bell Lab. Tech. J.*, 14:203–218, May 2009.

[73] Skype Limited. Skype, Nov. 2010. `http://www.skype.com/intl/en-us/home`.

[74] **S. Masoud Sadjadi**, Philip K. McKinley, Betty H.C. Cheng, and R.E. Kurt Stirewalt. TRAP/J: Transparent generation of adaptable Java programs. In *Proceedings of the International Symposium on Distributed Objects and Applications (DOA'04)*, volume 3291, pages 1243–1261, Agia Napa, Cyprus, October 2004.

[75] J.P. Sousa, V. Poladian, D. Garlan, B. Schmerl, and M. Shaw. Task-based Adaptation for Ubiquitous Computing. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 36(3):328 –340, may 2006.

[76] Burkhard Stiller, Christina Class, Marcel Waldvogel, Germano Caronni, Daniel Bauer, and Bernhard Plattner. A flexible middleware for multimedia communication: Design, implementation and experience. *IEEE Journal of Selected Areas in Communications*, 17:1614–1631, 1999.

[77] Stefan Arbanowski Sven van der Meer, Stephan Steglich. User-centric communications. In *IEEE International Conference on Telecommunications*, pages 425–444. Special Sessions, 2001.

[78] Gianluca Tonti, Jeffrey Bradshaw, Renia Jeffers, Rebecca Montanari, Niranjan Suri, and Andrzej Uszok. Semantic web languages for policy representation and reasoning: A comparison of kaos, rei, and ponder. In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, *The Semantic Web - ISWC 2003*, volume 2870 of *Lecture Notes in Computer Science*, pages 419–437. Springer Berlin / Heidelberg, 2003.

[79] Vladimir Tosic, Abdelkarim Erradi, and Piyush Maheshwari. Ws-policy4masc - a ws-policy extension used in the masc middleware. *Services Computing, IEEE International Conference on*, 0:458–465, 2007.

[80] Yingbo Wang, Yali Wu, Andrew Allen, Barbara Espinoza, Peter J. Clarke, and Yi Deng. Towards the Operational Semantics of User-Centric Communication Models. In *Proceedings of the 33th Annual International Computer Software and Applications Conference (COMPSAC 09)*, pages 254–262. IEEE Computer Society, July 2009.

[81] M. Wilkinson. Designing an adaptive enterprise architecture. *BT Technology Journal*, 24:81–92, 2006.

[82] Yali Wu, Andrew A. Allen, Frank Hernandez, Yingbo Wang, and Peter J. Clarke. A user-centric communication middleware for cvm. In *The 12th IASTED International Conference on Software Engineering and Applications (SEA 2008)*, pages 210–215. ACTA, Nov. 2008.

[83] C. Zhang, M. Sadjadi, W. Sun, R. Rangaswami, and Y. Deng. A user-centric network communication broker for multimedia collaborative computing. In *2nd IEEE/ACM CollaborateCom*, Nov. 2007.

VITA

ANDREW A. ALLEN

2005	B.Sc. Computer Science Florida International University Miami, Florida
2009	M.Sc. Computer Science Florida International University Miami, Florida

PUBLICATIONS AND PRESENTATIONS

Yingbo Wang, Peter J. Clarke, Yali Wu, Andrew A. Allen and Yi Deng: *Realizing Communication Services Using Model Driven Development*. Proceedings of the IASTED International Conference on Software Engineering and Applications (SEA 2007)

Andrew A. Allen, Sean Leslie, Ricardo Tirado, Yali Wu and Peter J. Clarke: *Self-Configuring User-Centric Communication Services*. The Third International Conference on Systems (ICONS 2008)

Yali Wu, Andrew A. Allen, Frank Hernadez, Yingbo Wang and Peter J. Clarke: *A User-Centric Middleware for CVM*. Proceedings of the IASTED International Conference on Software Engineering and Applications (SEA 2008)

Yingbo Wang, Peter J. Clarke, Yali Wu, Andrew A. Allen and Yi Deng: *Runtime Models to Support User-Centric Communication*. Proceedings of the 3rd International Workshop on Models@runtime(Sept 2008)

Paola Boettner, Mansi Gupta, Yali Wu and Andrew A. Allen: *Towards Policy Driven Self-Configuration of User-Centric Communication*. Proceedings of the 47th ACM Southeast Conference (ACMSE 09)

Yingbo Wang, Yali Wu, Andrew A. Allen, Barbara Espinoza, Peter J. Clarke and Yi Deng: *Towards the Operational Semantics of User-Centric Communication Models*. Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC 09)

Peter J. Clarke, Yali Wu, Andrew A. Allen and Tariq M. King: *Experiences of Teaching Model-Driven Engineering in a Software Design Course*. Proceedings of the Models 2009 Educators' Symposium (Oct 2009)

Andrew A. Allen, Yali Wu, Peter J. Clarke, Tariq M. King and Yi Deng: *An Autonomic Framework for User-Centric Communication Services*. Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative research (CASCON 09)

Peter J. Clarke, Andrew A. Allen, Tariq M. King and Edward L. Jones and P. Natesan: *A Web-Based Repository of Software Testing Tools to Support Pedagogy*. SPLASH 2010 Educators' and Trainers' Symposium(Oct 2010)

Tariq M. King, Andrew A. Allen, Yali Wu, Peter J. Clarke, and Alain E. Ramirez: *A Comparative Case Study on the Engineering of Self-Testable Autonomic Software*. Accepted to the 8th IEEE International Conference on the Engineering of Autonomic and Autonomous Systems(Apr 2011)

Yali Wu, Andrew A. Allen, Frank Hernandez, Robert France and Peter J. Clarke. *A Model-Driven Approach to Realizing User-Centric Communication Services*. SOFTWARE PRACTICE AND EXPERIENCE (journal accepted for publication 2011)