An Autonomic Framework for User-Centric Communication Services

Andrew A. Allen, Yali Wu and Peter J. Clarke
School of Computing and Information Sciences
Florida International University
Miami, FL 33199, USA
email: {aalle004, ywu001, clarkep}@cis.fiu.edu

Tariq M. King
Department of Computer Science
North Dakota State University
Fargo, ND 58105, USA
email: tariq.king@ndsu.edu

Yi Deng
College of Computing and Informatics
UNC-Charlotte
Charlotte, NC 28223, USA
email: Yi.Deng@uncc.edu

Abstract

The diversity of communication media now available on IP networks presents opportunities to create elaborate collaborative communication applications. However, developing collaborative communication applications can be challenging when using the traditional stovepiped development approach with lengthy development cycle as well as limited utility. One proposed solution to this problem is the Communication Virtual Machine (CVM). CVM uses a user-centric communication (UCC) approach to reduce the complexity while offering operating simplicity to developers and users of collaborative communication services. CVM currently utilizes only one communication framework which limits the number, quality and types of services available.

We extend the CVM to support multiple communication frameworks with a policy-driven approach for the selection and configuration of communication services. Users define policies that, through automation, can maintain high level goals by influencing the selection and configuration decisions. In this paper we provide a policy definition for UCC and a technique to evaluate these UCC policies. We also present our autonomic framework for UCC and experimental evaluation of the implementation.

1 Introduction

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. The diversity of communication media now available on IP networks presents opportunities to create elaborate collaborative communication applications. Various vendors [7, 18, 9] provide a variety of communication tools and communication services for creating synergistic collaborative communication applications. For example, Skype [18] and GoogleTalk [7], which are commercial-off-the-shelf (COTS) communication APIs, have been made available by their respective companies and support the development of more sophisticated communication services by third parties.

However, developing collaborative communication applications can be challenging, especially when using the traditional stovepipe development approach. Traditional stovepipe development approaches result in rigid technology, lengthy and costly development cycles, as well as limited utility. Additionally, users of these services have to manage and adapt their communication in line with differences in services across devices, networks and media. Administrators tasked with managing and, where possible, minimizing the costs associated with the resource have to ensure that the resource usage remains consistent with the business goals and changing business model.

One proposed approach to this challenge is user-centric communication which aims to reduce the complexity and offer operating simplicity to users [13]. The Communication Virtual Machine (CVM) technology, proposed by Deng et al. [5], exemplifies this concept with a model-driven approach to realizing communication services. Domain 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) [4] and executed on the CVM platform. The CVM includes the Network Communication Broker (NCB), the layer responsible for providing a network-independent API to the upper layers of CVM. However, the current CVM depends on a single communication framework and our preliminary studies [2, 3] show that a single communication framework limits the number, quality and types of services available.

The NCB layer of the CVM has been extended to support self-* properties and is the main area of focus in this paper. The extension provides a policy-driven approach for allocating and self-configuring communication resources. This 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, domain experts and novice users, are executed by CVM and provide services that are guided by user-defined policies. The major contributions of this paper are as follows:

1. Development of a policy definition for user-centric communications (UCC).
2. Development of a technique to evaluate UCC policies.
3. Design of an autonomic framework to support UCC.
4. Experimental evaluation of the autonomic framework.

The rest of this paper is organized as follows, Section 2 provides background on supporting technologies and concepts. User-centric communication polices are discussed in Section 4 with our autonomic framework presented in Section 5. Experimental evaluations are discussed in Section 6 with related work in Section 7 and we conclude in Section 8.

2 Background

In this section we introduce user-centric communication and the CVM technology. We also provide an overview of autonomic computing.

2.1 User-Centric Communication

The convergence of various multimedia communications that includes voice, video and data presents many opportunities for enabling unified communication. There are however challenges presented by this model of communication. 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 who is tasked with managing and adapting the communication. Complexity can hinder rather than enhance communication [13] and research is under way in academia and industry to find solutions to such challenges of unified communication.

The user-centric approach [13, 20] is one such research direction. This solution aims to reduce the complexity and offer operating simplicity [13] to users of these unified 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 [20].

The user-centric communications (UCC) approach is therefore about matching the communication resources with the individual's needs at a particular point in time, and adapting accordingly, thereby reducing the complexity to the user.

2.2 Autonomic Computing

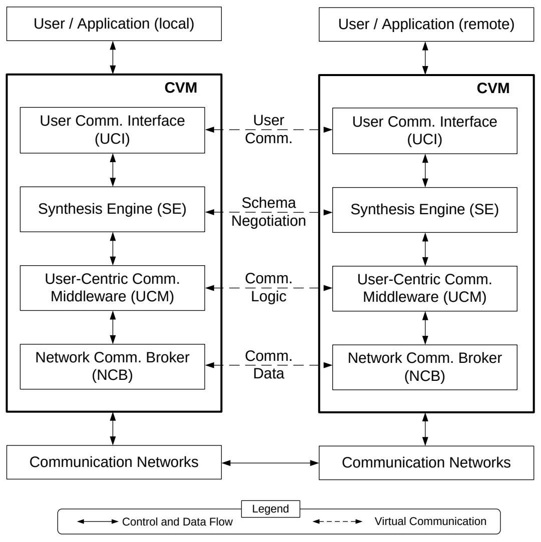
Autonomic computing (AC) [8, 10] refers to computing systems characterized by one or more of the self-management behaviors which include: self-configuration, self-optimization, self-protection, and self-healing. This self-management is achieved by automating low-level tasks while allowing administrators to specify the goals of the system as high-level policies.

The architecture of an AC system is generally composed of managed resources, touchpoints, autonomic managers, and knowledge sources. Managed resources are entities for which self-management services are desired, with touchpoints being the manageability interfaces used to automate low-level management tasks for these entities. Autonomic managers (AM) provide the high level management and can be categorized as either Touchpoint AMs [8], that directly manage resources through their touchpoint interfaces or Orchestrating AMs [8], that manage pools of resources or optimize the Touchpoint AMs for individual resources. Knowledge sources implement registries or repositories that can be used to extend the capabilities of AMs.

A key concept in autonomic computing is the use of closed control loops for self-management. The closed control loops are generally implemented as monitor, analyze, plan, and execute (MAPE) functions [8]. The monitor function collects state information from the managed resource and correlates them into symptoms for analysis. If analysis determines that a change is needed, a change request is generated and a change plan is formulated for execution on the managed resource.

2.3 CVM Technology

CVM technology supports the model creation and realization of user-centric communication services. We present a description of this technology.



User / Application (local)
User / Application (remote)
CVM
CVM
User Comm. Interface (UCI)
User Comm.
User Comm. Interface (UCI)
Synthesis Engine (SE)
Schema Negotiation
Synthesis Engine (SE)
User-Centric Comm. Middleware (UCM)
Comm. Logic
User-Centric Comm. Middleware (UCM)
Network Comm. Broker (NCB)
Comm. Data
Network Comm. Broker (NCB)
Communication Networks
Communication Networks
Legend
Control and Data Flow
Virtual Communication

Figure 1: Layered architecture of the CVM.

2.3.1 Communication Virtual Machine

Deng et al. [5] developed the notion of the Communication Virtual Machine (CVM), which enables the realization of models defined using a Communication Modeling Language[1] (CML). CVM has a layered architecture and lies between the communication network and the user (or application). Figure 1 shows the layered architecture of the CVM. The key components of the CVM are:

1. *User Communication Interface* (UCI), provides a modeling environment for users to specify their communication requirements using a Communication Modeling Language (CML). CML can be used to describe a user communication schema or schema instance, analogous to an object-oriented class and object;
2. *Synthesis Engine* (SE), contains a set of algorithms responsible for (1) automatically synthesizing the user schema instance to an executable communication control script, and (2) negotiating the schema instances with other participants in the communication;

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

3. *User-centric Communication Middleware* (UCM), executes the communication control script and coordinates the delivery of communication services to users, independent of the underlying network configuration;
4. *Network Communication Broker* (NCB), provides a network independent API to the UCM that masks the heterogeneity and complexities of the underlying network for the realization of the communication services.

2.3.2 Communication Frameworks

Instant messaging (IM) applications were originally devised as a way for users to hold real-time conversations on-line, however they have been expanded to include file-sharing, game play, streaming audio and video, and sending text messages to cell phones. To foster further growth, many companies have provided API's to facilitate third party add-ons and extensions of their product's communication framework which essentially become building blocks or *communication frameworks* for more elaborate communication applications. The NCB architecture supports the integration of communication frameworks, such as Skype [18] and Smack [9], providing an extensible set of communication services for the CVM.

3 Overview of NCB

Network communication broker (NCB) is the lowest layer of the CVM architecture. NCB utilizes communication frameworks such as Skype [18], Smack [9] and its own NCBNative [2] to provide collaborative communication services. A policy authoring tool [3] is used to create user-defined policies, these policies are stored in the policy repository of NCB. Figure 2 shows the flow of control during the normal operation of the NCB. A user's communication model would be accepted and transformed by the upper layers of the CVM into a series of requests for the NCB as in step 1 in Figure 2.

The requests are inserted into a priority queue where the highest priority request is evaluated against stored policies and the current operating environment state (steps 2 and 3 of Figure 2). The result of the evaluation is an identified communication framework that satisfies the user's request and does not violate the active policies. If the evaluation process returns a new framework for the connection, the *TouchPoint Manager* (TPM) issued commands to setup the new framework for use and updates the framework-to-connection mapping table, step 4, 5 and 6 of Figure 2. Section 4 provides details on the policy definition and its interpretation.

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 (step 7 of Figure 2) to the CSM. The CSM looks up the framework for the connection in the framework-to-connection mapping table, step 8, and executes the command on the specific framework (step 9 of Figure 2).

Any resulting events or exceptions are passed to the NCB Manager, step 10 of Figure 2. Any out-of-band or *reactive events* that cannot be handled by the TPM are encapsulated and forwarded to the OAM. The OAM processes this event 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. The details of this approach are provided in Section 5.

4 UCC Policies

The interfacing of multiple communication frameworks expands the available communication services. However, utilizing this method in isolation raises the complexity for users and developers of CVM who will now have to become aware of each communication framework's service listings and limitations. This would be contrary to one of the key concepts of CVM, which is to reduce the complexity associated with the development of collaborative communication applications. We therefore investigated methodologies for influencing service selection and automating configuration. After analysis of the user-centric communication do-

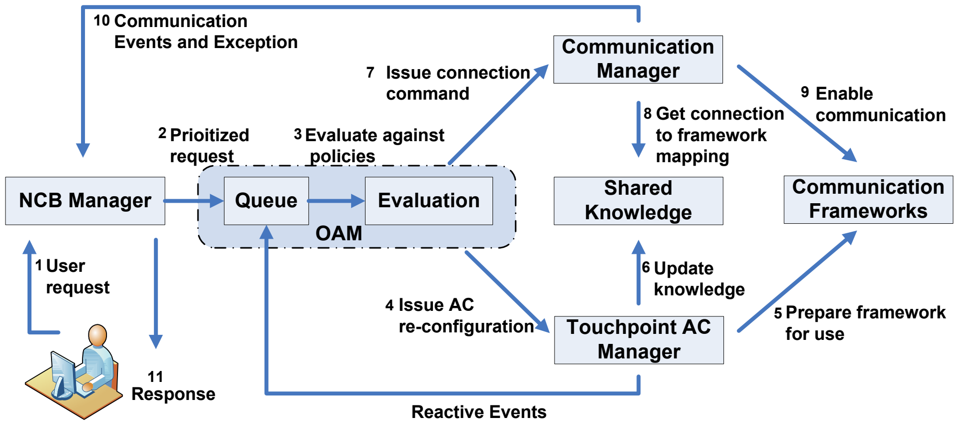


Figure 2: NCB Control Flow Diagram.

main [3], a policy-driven approach was identified as the best fit.

Policies are rules that define the choices in the behavior of a system [16]. Separating the rules from the implementation facilitates dynamic modification of these rules, yielding flexibility to change the management strategy of the system and hence modify the behavior of a system without changing its underlying implementation [19]. In this section, we define user-centric communication policies and present a structural design for representing these policies.

4.1 Policy Definition

To fully realize the vision of autonomic computing [10] there is a need to express high level business goals for the managed systems. Policies are one way to express these goals. 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 [16]. Agrawal et. al [1] defines 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 $V_i \dots V_n$ to the set of system attributes A_S.

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

$$B(S) = \{\{V_i, ..., V_n\} \mapsto A_S^0, ..., \{V_j, ..., V_m\} \mapsto A_S^t\}$$

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

Then

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

We define policy P as the function $g(x)$.

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

Therefore

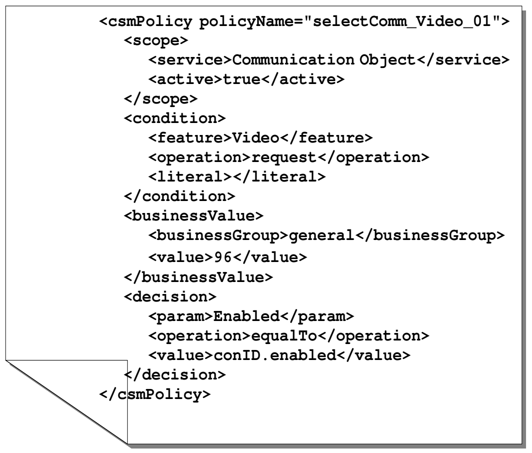
$$P(C, B(S)) \subset B(S)$$

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 [1]. 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 [1].

We define constraint C as a 4-tuple (C_O, C_N, C_V, C_D) where:

- C_O: narrows the scope of the constraint to identify subcomponent 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
- C_D: associates a condition C_N with some action that achieves some desired behavior

We extended these elements based on the results of domain analysis of the collaborative communication domain [2, 3]. Where:



```
<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 3: XML representation for user-centric communication policy.

- *Scope* 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.
- *Condition* represents the trigger for the application of the policy represented by (1) medium - 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.
- *Business value* facilitates a ranking of conflicting polices. It is represented by (1) the business group attribute which provides a way to associate policies, and (2) a numeric value that represents the policy's priority in the group.
- *Decision* indicates the policy's desired outcome or expected behavior of the communication. It is expressed as triple (consisting of parameter, operation and value) that specify the criteria that the communication must satisfy.

Figure 3 is an example of a user-defined policy that guides the establishment of video conferences, the elements are interpreted as:

- Scope: *selection of Communication Object*
- Condition: *request for video*
- Business value: *general group with priority 96*
- Decision: *select communication framework whose medium supports at least the connection's users count*

We provide details on the evaluation of this example in Section 4.2.

4.2 Policy Interpretation

While policies describe the 'what', policy interpretation represents the 'how' that ensures stated goals in the policies are maintained. We provide a discussion of our lightweight policy evaluation mechanism used in satisfying communication needs.

Policy selection: Policies are stored as Extensible Markup Language (XML) representations in the *policy repository*. Policies are deemed relevant if the request attributes match the policy values for service, feature or operation. 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.

Using set reduction in framework selection: For a set of relevant policies, they 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 guide which one satisfies the request. This is in fact a set reduction technique based on the naive set theory for intersection [6]: start with a full set, and gradually reduces it until all policies are processed. The evaluator produces a set of candidate frameworks as its output for each request.

Evaluation details: Besides the policy, additional input to the policy evaluation process is the currently available set of communication frameworks (the environment) and the request information from the user. We will use Figure 3 as a referring example. Details of the policy

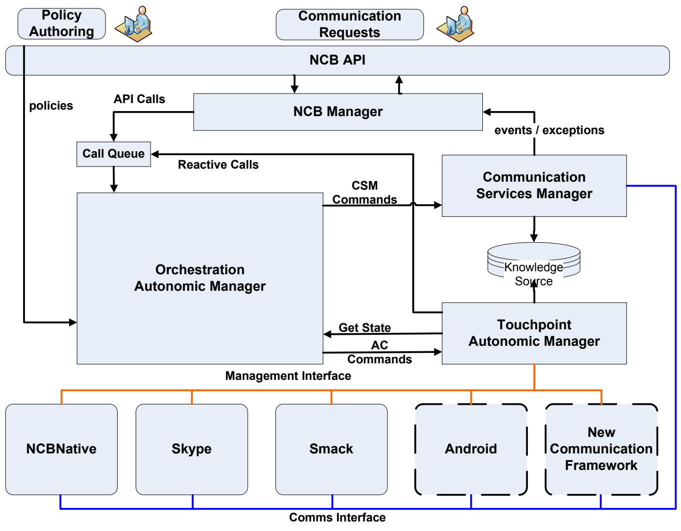


Figure 4: NCB Autonomic Architecture.

evaluation works as follows: The scope component of each policy object is used to identify the interested object, in the case of the example of Figure 3 it is the selection of a communication object (communication framework). The condition component determines the applicability of the policy for the current request, in this case this policy is valid only for video selection. The decision component is responsible for stating the desired goal of the policy, for our referring example the desired outcome is a communication framework that supports video.

In our video conferencing example in Figure 3 the following occurs: (1) the decision component of the policy is extracted; (2) the number of participants in the conference is extracted as well as the media desired; (3) the framework information instantiated for each of the currently available communication frameworks. This includes the maximum number of participants supported by each communication framework and the supported media capabilities; and (4) a comparison is made between the request with the conferencing capabilities of each available framework based on the operators specified in the policies, such as "greaterThan" or "equalTo". If the comparison shows that a communication framework cannot satisfy the goal as stated by the decision component, then this communication framework will be removed from the set. Each policy will rule out the set of frameworks that do not satisfy that policy, and the final resulting set would be the candidate set of communication frameworks that satisfy all communication needs for the user. If the set contains more than one members, a member will be chosen randomly.

5 UCC Autonomic Framework

A high level view of the architectural approach is presented in this section. We also highlight some of the significant components with a detailed design.

5.1 Architecture

The introduction of autonomic behavior in the NCB is a first step towards enabling an autonomic CVM through a bottom up approach. While our current focus is on the extension of the NCB to include self-configuration behavior, we perceive that there will be significant benefits for user-centric communication from the addition of other self-management behavior such as self-optimization and self-healing. Additionally, the list of self-management properties continues to expand as researchers identify new behavior [11] for systems that will either directly or indirectly support the autonomic concepts. Considerations such as those just outlined reinforced the need for an extensible design vertically (to add other self-* behavior)

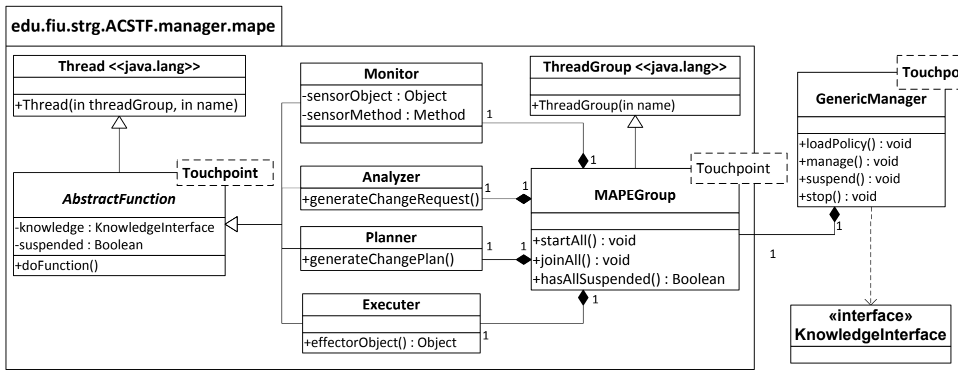


Figure 5: Reusable Autonomic Design.

and horizontally (to extend self-management to the upper layers of CVM).

Our approach for supporting vertical extensibility is by laterally stacking components that share a managed resource. Liu et. al [15] 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. We utilize this concept in our design. At the top right of Figure 4 is the *Communication Services Manager* (CSM) which utilizes the functional port of the communication frameworks (bottom of Figure 4) 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 framework in Figure 4, which is used by the *touchpoint autonomic managers* (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 [15]. For highly multithreaded and asynchronous systems such as communication, coordination becomes even more challenging. Our approach reduces the potential for such conflicts through the use of a high level coordinator which we refer to as the *orchestration autonomic manager* (OAM). The OAM (left in Figure 4) 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 4), with respect to stated policies. The OAM is the main policy decision point in the NCB 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 falls 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 generate the required commands 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.

5.2 Detailed Design

Figure 5 depicts the revision to King et al.'s [12]'s *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 independence of

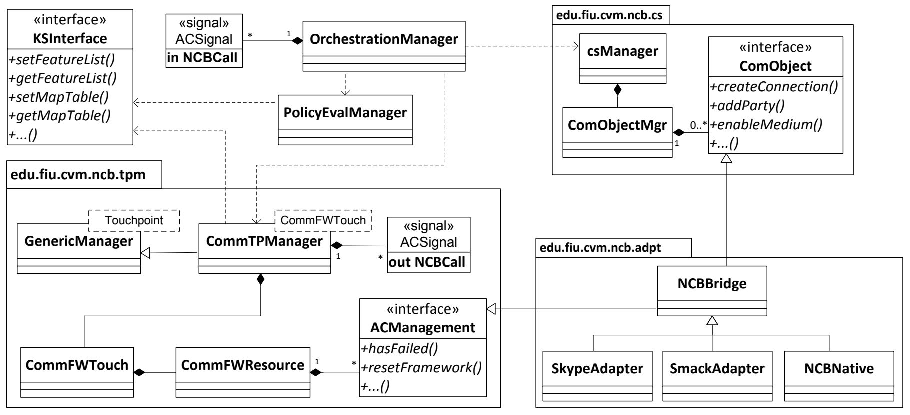


Figure 6: NCB Detailed Design Diagram.

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, which provides the mechanisms to safely suspend and resume the MAPE threads as a single unit for coarse grained control (recall our discussion in Section 5.1 where the OAM may need to suspend MAPE activities). The controller class *GenericManager*, see top right of Figure 5, coordinates the operations and services as well as maintain the knowledge source (*KnowledgeInterface*, bottom right of Figure 5)used by the MAPE functions. *GenericManager*, *MAPEGroup* and the MAPE classes (through inheritance from *AbstractFunction*, bottom left of Figure 5) are parameterized with the template class, *Touchpoint*. Self-management will be carried out by classes representing the *Touchpoint* template class.

Figure 6 shows the core components of our NCB Design. At the bottom left of Figure 6 is the package *edu.fiu.cvm.tpm* which contains *CommTPManager*, our specialization of the *GenericManager*. We parametrize *CommTPManger* 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 `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 *Orchestration-Manager*.

We show at the top right of Figure 5 the package *edu.fiu.cvm.cs* which has the components responsible for managing the communication services. 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 bottom right in Figure 5 includes the adapter classes for the communication frameworks. For black or gray box communication frameworks such as Skype, the adapter classes function as managed resource wrappers. 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 6, coordinates autonomic (TPM) and non-autonomic (CSM) components use of the shared resources. Additionally, the use of the knowledge by the two components adds to the safeness. CSM uses the MappingTable (`getMappingTable` method of the *KSInterface*, top left of Figure 6) as read-only while the TPM is responsible for updating the table using locks when writing to the table. The *PolicyEvalManager*, below the *OrchestrationManager* in Figure 6, is the high level policy decision point.

6 Evaluation

We have implemented a prototype that incorporates the designs discussed in this paper. The prototype was evaluated with respect to its efficiency compared to previous versions of CVM and the scalability of the selection mechanism. The policy shown in Figure 3 was used in both experiments. In this section we outline our evaluation methodology and present the experimental results.

6.1 Experiment Set 1

We used three version of the NCB prototype during the experimental runs, where each run was ten iterations of the specific scenario. Two versions included the initial developed prototype NCB without self-configuring behavior, one version was configured to only use the Skype version 3.4 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 3.4 API and Smack 3.0.4 API as the supporting communication frameworks. The three scenarios (2 way, 3 way, 4 way) measured the time to setup a user' request for an initial audio conference and then a switch to video conferencing. A common driver application was used to simulate CVM type requests.

Figure 7 shows the average for 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 bar representing the non-autonomic includes the times for starting the Skype version of NCB initially, stopping the Skype version of NCB 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

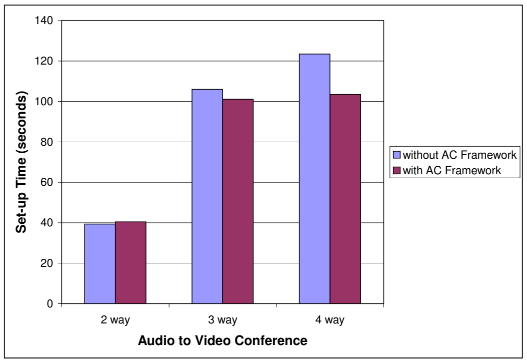


Figure 7: Audio to Video Conferencing Set-up Times.

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.

As shown in Figure 7, 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 single framework implementation this requires conference disconnection, shutdown and a restart with the new framework implementation. We averaged the time taken to manually stop and start an implementation to compensate for differences in the speed of users in the experiments. We also noted from the data that the AC Framework accounted for noticeably less of the overall execution time as the conference's participant count increased.

6.2 Experimental Set 2

We evaluated the technique for selecting the candidate communication framework using the new NCB prototype. We 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 ten iterations of a specific pool size. The experimental runs for each pool size was

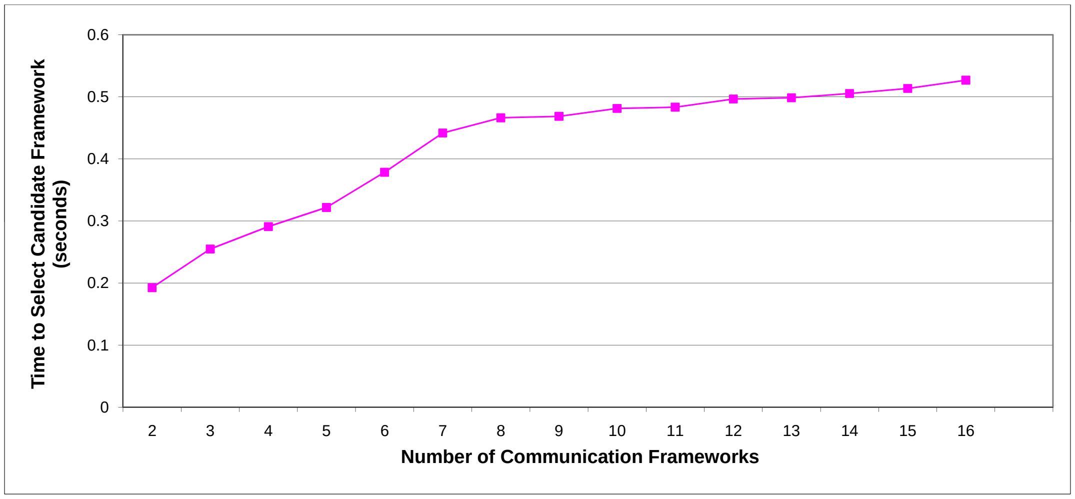


Figure 8: Average Times to Select Candidate Communication Framework.

then averaged and tabulated. We also used a common driver application to simulate CVM type requests.

The averages for the pool sizes ranging from 2 to 16 are shown in Figure 8. The graph represents the average time of 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. This was expected since on average a reduction in the set of communication frameworks occurs with each attribute evaluated. We note however that a request that can be satisfied by all communication frameworks in the set may not exhibit this behavior.

6.3 Discussion

Our prototype of the NCB is an initial proof of concept for the introduction of extensible services to the CVM. The prototype NCB currently includes three communication frameworks, Skype, Smack and NCBNative. For experiment 2, 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.

Tools for performance evaluation generally do not introduce significant overhead to the monitored environment. For real-time systems this tend 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 introduced delays in the participants negotiation process. We compensated for this overhead by reducing the sampling period and focused the profiling and monitoring based on the criteria of the current experiment.

7 Related Work

In [17], a framework for dynamic component configuration and reconfiguration based on Microsoft.NET is implemented and evaluated. Their work is similar to ours in that an XML-based configuration description language is designed and used to instantiate the component, or generate reconfiguration actions such the addition, removal or modification of a component, if necessary. However, their reconfiguration algorithm, once chosen, remains fixed at runtime while our design is inherently more flexible since reconfiguration algorithms could be wrapped as internal policies.

Lewis et al. in [14] present an approach for the management of user centric adaptive services. This work shares some similar traits with our autonomic framework in its adaptive service composition and policy based management of adaptive system behavior. However, a centralized system is used to compose the services needed. Our approach is based on a peer to peer model, with the flexibility to change this model.

King et. al [12] present a reusable object-oriented design for developing self-testable autonomic software. Their design provides a generic manager that can be instantiated to provide both autonomic and testing services based on high-level policies. King et. al [12] apply their design to a small autonomic prototype simulation. Our approach borrows from the manager design in [12]. However, due to the high safety requirements of the NCB, we extended the work in [12] by incorporating additional mechanisms for suspending and resuming the generic manager. Furthermore, our autonomic CVM design applies the generic manager design to the implementation of a real autonomic system rather than a simulation.

An initial idea for self-configuring user-centric communication services was described in [2] and further refined in [3] with the concept of policy-driven self-configuration. Conceptual designs for the architecture were presented along with a policy authoring tool for UCC. We extend this work with an architecture and detailed designs for the autonomic functionalities in NCB through the use of our autonomic framework. Additionally, we provided a formal definition for UCC policies with a technique for evaluating the policies as well as experimental evaluations of the implementation.

8 Conclusion

In this paper we presented an extension of the NCB that supports interfacing of multiple communication frameworks and policy-driven selection of communication services. We defined user-centric communication policies and outlined our technique for the evaluation of the user-centric communication policies. Details of the design of our autonomic framework were discussed and evaluations of the autonomic framework presented. We plan in the future to further extent the NCB to include self-healing properties to provide recovery options for a collaborative communication session. An extended version of this paper will be prepared, with more exhaustive experiments that will include analysis of the policy definition process as well as fault injection for analysis of the self-healing properties.

Acknowledgements

This work was supported in part by a Florida International University Foundation grant and the National Science Foundation under grant HRD-0833093.

About the Authors

Andrew A. Allen is a PhD student at Florida International University's School of Computing and Information Sciences. His areas of interest are software engineering, autonomic computing, model-driven development and collaborative communication. His Internet address is aalle004@cis.fiu.edu.

Yali Wu is a PhD student at Florida International University's School of Computing and Information Sciences. Her areas of interest are software engineering, model-driven development, workflow and collaborative communication. Her Internet address is ywu001@cis.fiu.edu.

Peter J. Clarke is an Associate Professor at Florida International University's School of Computing and Information Sciences. His areas of interest are software testing, model-driven development and autonomic computing. His Internet address is clarkep@cis.fiu.edu.

Tariq M. King is a Assistant Professor at North Dakota State University's Department of Computer Science. His areas of interest are software testing, autonomic computing, and model-driven development. His Internet address is tariq.king@ndsu.edu.

Yi Deng is a Professor and Dean at the University of North Carolina-Charolette's College of Computing and Informatics. His Internet address is Yi.Deng@uncc.edu.

References

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

[2] 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.

[3] 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.

[4] Peter J. Clarke, Vagelis Hristidis, Yingbo Wang, Nagarajan Prabakar, and Yi Deng. A declarative approach for specifying user-centric communication. In *CTS 2006*, pages 89–98. IEEE, 2006.

[5] 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.

[6] Keith Devlin. *The Joy of Sets: Fundamentals of Contemporary Set Theory*. Springer, NY, second edition, 1993.

[7] Google. Googletalk, Sept. 2007. http://www.google.com/talk/.

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

[9] Jive Software. Smack api, Nov. 2008. http://www.igniterealtime.org/projects/smack/.

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

[11] Tariq M. King, Djuradj Babich, Jonatan Alava, Ronald Stevens, and Peter J. Clarke. Towards self-testing in autonomic computing systems. In *ISADS '07*, 2007.

[12] 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.

[13] Philippe Lasserre and Dennis Kan. User-centric interactions beyond communications. Alcatel Telecommunications Review, 1st Quarter, 2005.

[14] D. Lewis, T. O'Donnell, K. Feeney, A. Brady, and V. Wade. Managing user-centric adaptive services for pervasive computing. In *ICAC 2006*, pages 248–255. IEEE Computer Society, 2004.

[15] 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.

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

[17] Andreas Rasche and Andreas Polze. Configuration and dynamic reconfiguration of component-based applications with microsoft .net. In *IEEE Sixth Intn'l Symposium on Object-Oriented Real-Time Distributed Computing*, page 164, 2003.

[18] Skype Limited. Skype developer zone, Feb. 2007. https://developer.skype.com/.

[19] M. Sloman. Policy driven management for distributed systems. *Journal of Network and Systems Management*, 2:333–360, 1994.

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