

Self-Configuring User-Centric Communication Services

Andrew A. Allen, Sean Leslie, Yali Wu and Peter J. Clarke
 School of Computing and Information Sciences
 Florida International University
 Miami, FL 33199, USA
 email: {aalle004, slesl001, ywu001, clarkep}@cis.fiu.edu

Ricardo Tirado
 Department of Computer Engineering
 University of Puerto Rico-Mayaguez
 Mayaguez, P.R. 00681-9000
 email: rtn23821@uprm.edu

Abstract—

The functionality of communication applications, such as instant messaging, has dramatically improved due to market competition. The quest for the competitive edge has resulted in the development of open platform APIs that allow communication applications to become building blocks, or communication frameworks for more elaborate communication applications. The services available through these different communication frameworks can however be quite dissimilar, as revealed from a survey of some of these frameworks.

In this paper we propose an architecture that utilizes multiple communication frameworks and the autonomic computing capability of self-configuration of these frameworks. This architecture extends Network Control Broker (NCB) which is the layer of the Communication Virtual Machine (CVM) that interacts with the underlying communication networks. We provide a detailed design of the proposed architecture and show how a communication scenario can be realized using a prototype of the NCB.

I. INTRODUCTION

Designing and implementing collaborative multimedia applications used to be a formidable challenge. Collaborative multimedia application pioneers had little choice but to custom-build most of the technology needed for their distributed multimedia applications [1]. Major strides have been made over the last decade to improve the methodologies for the development of these applications, as well as the availability of multimedia toolkits to support such development.

The spectacular growth in popularity of the Web [2] has also been marked with the increased utilization of multimedia communication applications such as Skype [3], GoogleTalk [4] and Windows Live Messenger [5], with high user volumes leading to a rapid evolution in their functions and quality. Many of these companies have in recent years made their APIs available to allow third parties to extend and enhance their communication services, which essentially become building blocks or communication frameworks for more elaborate communication applications.

Deng et al. [6] proposed a new approach that supports the rapid conception, construction and realization of new application-specific communication services. This approach, based on the *Communication Virtual Machine* (CVM), exploits

model-driven development and abstracts the underlying network infrastructure completely from the user. The lowest layer of the CVM is the *Network Communication Broker* (NCB) [7] and it is responsible for encapsulating the networking complexity and heterogeneity of basic multimedia and multi-party communication.

In this paper we survey several communication frameworks to identify the services they provide. These communication frameworks include: NCB [7], Skype [3], Java MSN Messenger Library (JML) [8] and GoogleTalk [4]. The results showed that no one communication framework surveyed provides a complete package of communication functions and features. For collaborative technologies like CVM which utilizes the convergence of various multimedia communications that include voice, video and data, the limitations of the individual communication frameworks are too significant to be ignored. However, if the function that the individual communication framework brings is valued, it warrants inclusion. To this end we extend the NCB layer of the CVM to include the use of multiple communication frameworks using the paradigm of self-management in autonomic computing.

The major contributions of this paper are as follows:

- 1) Applying a novel approach to self-configuring media services with multiple communication frameworks.
- 2) Providing a detailed design for the self-configuration by extending the NCB architecture.
- 3) Showing the feasibility of the extended NCB by developing a prototype and realizing a communication scenario.

In the next section we present background on self-configuration and CVM. In Section 3 we motivate this research and Section 4 describes self-configuration of the NCB. Section 5 describes the communication scenario realized using the prototype. The related work and concluding remarks are presented in Sections 6 and 7, respectively.

II. BACKGROUND

In this section we provide an overview of self-configuration in Autonomic Computing and introduce the Communication Virtual Machine (CVM).

A. Self-Configuration

Autonomic Computing (AC) is a computing environment with the ability to manage itself and dynamically adapt to change in accordance with business policies and objectives [9]. AC, proposed by IBM, addresses the problems associated with the increasing complexity of computing systems, and the evolving nature of software. AC systems are characterized by intelligent closed loops of control that are typically implemented as the monitor, analyze, plan, and execute (MAPE) functions of autonomic managers.

The essence of autonomic computing systems is self-management [10] and it is derived from a combination of four broad capabilities: self-configuring, self-healing, self-optimizing, and self-protecting. In this paper our focus will be on self-configuring. *Self-Configuration* provide the means whereby a system can dynamically adapt to its changing environment. Such changes could include insertion or the removal of components. Dynamic adaptation helps ensure continuous strength and productivity of the IT infrastructure, resulting in business growth and flexibility [9].

B. Communication Virtual Machine

Deng et al. [6] developed the notion of the Communication Virtual Machine (CVM) which enables the realization of models defined using a Communication Modeling Language (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 synthesizes the user schema instance to an executable communication control script, and (2) negotiating the schema instances with other participants in the communication;
- 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 Control Broker (NCB), provides a network independent API to the UCM that mask the heterogeneity and complexities of the underlying network for the realization of the communication services.

III. MOTIVATION

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

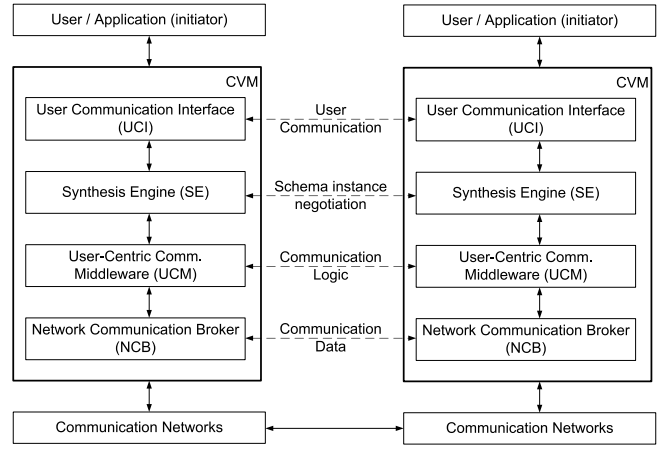


Fig. 1. Layered architecture of the CVM.

TABLE I
AVAILABILITY OF FEATURES IN FOUR COMMUNICATION SERVICE FRAMEWORKS.

Criteria	JML	NCB	Skype	Smack/ GoogleTalk
Core Features				
Chat (one-to-one)	1	1	1	1
Chat (group)	1	1	1	1
Contact list	1	1	1	1
Audio (one-to-one)	0	1	1	1
Audio (conference)	0	1	1	0
Video (one-to-one)	0	1	1	0
Video (conference)	0	1	0	0
File Transfer	1	1	1	1
Additional Features				
Scrolling marque text	0	0	0	1
Emoticons	1	0	1	0
Online status	1	0	1	1
Avatar images	1	1	1	1
Voicemail	0	0	1	0

of their product's communication framework. For applications such as CVM with limited manpower to dedicate to the maintenance and enhancement of its NCB, such communication frameworks can be considered for the underlying network support of NCB. To this end we surveyed three communication frameworks:

- 1) GoogleTalk [4], is XMPP compliant with source code provided and supported by Google. It allows any XMPP compliant client to communicate with its servers.
- 2) Java MSN Messenger Library (JML) [8], supports Microsoft Notification Protocol (MSNP) versions 8 to 12. This is a Java based open source communication framework that provides connection to the MSN Messenger network. Its library is currently used in at least three communication applications.
- 3) Skype [3], peer-to-peer Internet telephony network that offers free and paid communication services through proprietary client application. An API is provided to developers but it must use the proprietary client to access the network.

Table I shows the available features for each of the communication frameworks including the NCB. A "1" indicates

that feature is present and “0” the feature is absent. Our investigations covered the core features and additional features as listed in Table I.

The most significant finding from Table I was that no single framework provided all the services required by many communication scenarios. Although NCB has a “1” for all the core features (third column) these features are restricted based on certain constraints, including the presence of some firewalls. In addition, we also found dissimilar service levels as well as tiered structures for service availability. For example, Skype’s audio connections are free between PCs but involve a charge for calls to cell phones or land line phones. The limitations of the individual communication frameworks could be addressed by extending the framework with custom implementations of the missing functionalities, however issues of maintainability of such custom implementations would arise.

To obtain the best service to the CVM from the available communication frameworks motivated us to integrate multiple communication frameworks within NCB. This new multi-protocol NCB will provide all the functionalities required by CVM while providing an extensible infrastructure for new communication frameworks. Since the NCB abstracts the network layer from CVM, CVM is unaware of which framework is providing the requested media service. As such self-configuring media services as needed based on constraints for that media service was also included in the design. This novel approach provides an extensible yet independent platform for the realization of communication applications.

IV. SELF-CONFIGURING NCB

In this section we present a high-level autonomic design view of the NCB focusing on the components used during self-configuration. Figure 2 shows the autonomic design view of the NCB highlighting the Communications Services Manager (CSM) and the NCB bridge. The other components shown in Figure 2, using dashed lines, illustrate how the NCB interacts with the UCM and underlying communication networks [6]. The structure of the policy used to self-configure the communication frameworks and a detailed design of the CSM is also presented. We have used the design presented in this section to construct a prototype of the NCB for the CVM.

A. High-Level Autonomic Design View

The Communication Services Manager (CSM) self-configures the communication services needed by NCB for the realization of a communication model. The CSM is an autonomic manager and as such carries out the MAPE functions. The manageability endpoints are the supporting communication frameworks implemented with a common interface that includes standard sensor/effector interfaces as shown at the bottom of Figure 2. In the current prototype these communication frameworks include: GoogleTalk [4], Java MSN Messenger Library (JML), Skype [3] and a modified version of the original NCB [6] referred to as the *NCBNative*. The CSM also has sensor/effector interfaces to provide the *NCB Manager* with the ability to monitor and manage the CSM.

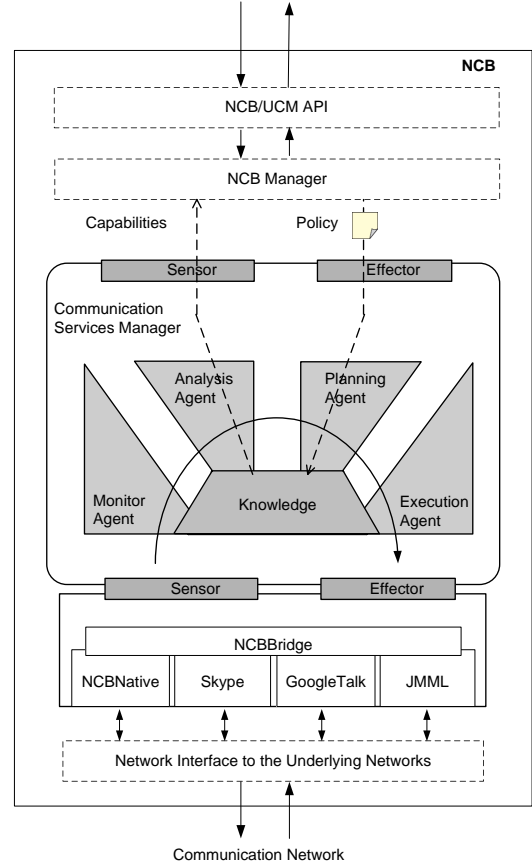


Fig. 2. Autonomic design view of the NCB

The NCB Manager provides policies that govern the behavior of the CSM and request communication services from the CSM. *Policies* are added to the knowledge via the planning agent which set the policies active or inactive based on the information passed within the policy. Active policies direct the decisions of the CSM in the manipulations of the underlying communication frameworks. The *monitor agent* does an inventory collection of the available communication frameworks for NCB. The monitor agent collects this information as the communication frameworks register with the agent on start up and also polls the communication frameworks at specified intervals for state updates such as removed frameworks, enabled or disabled services or reduced service levels on existing frameworks. These state updates provide feedback that can be measured against the dictates of the policies. New frameworks can be integrated dynamically by registering with the agent through the framework’s Bridge interface.

The *analysis agent* characterizes the inventory, which serves to inform the upper layers of the CVM of the *capabilities* or the types of media services that are available overall and assist in the selection of requisite services. When a request for media services is received, this request will state the service required, the participants for the service or optionally could state constraints for QoS. An analysis will be performed by the analysis agent on the request against the active policies in the knowledge manager to select a candidate communication

framework that is able to provide the service requested. Additionally if a state update received through the monitor agent indicates that the current media service level has changed, a re-analysis is initiated. A change request is generated if there is a need to change the existing communication framework or to initiate communication with a framework.

After the analysis agent creates a change request it is passed to the *planning agent* for processing. The planning agent creates a list of necessary commands to effect the utilization of the candidate communication framework. The planning agent will evaluate the actions required for providing the service, such actions could be to destroy the current conference connection and create a new connection if the addition of participants requires a new connection, then create a change plan for execution. The change plan is executed by the *execution agent* and ends with the handing off of the candidate communication framework to the NCB Manager for the realization of the communication.

B. Specifying Communication Policies

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 [11]. By applying AC self-configuring techniques we seek to decouple that choice in the behavior of the system from the actual implementation. Separating the policy from the implementation of a system permits the policy to be modified in order to dynamically change the strategy for managing the system and hence modify the behavior of a system without changing its underlying implementation [12]. This separation facilitates the addition of new communication frameworks that will be governed by the existing policies, as well as the introduction or modification of policies as business requirements change. There are four common elements identified when studying the various policy standards [13]:

- Scope: what is or is not the subject of the policy
- Condition: when a policy is to be applied
- Business value: express the relative priority of a policy and allows a system to make economic trade-offs
- Decision: describes the observable behavior or desired outcome of a policy

The XML schema for the communication service policy consist of the following components:

- *Scope* the subject of the policy, in our design it defines the management operation to be performed on the specified communication component. The XML representation of the scope, as shown in Figure 3, consist of two parts: (1) *service* - the applicable communication component, and (2) *operation* - the intended management action.
- *Condition* represents the trigger for the consideration of the policy represented in two parts: (1) *medium* - the carrier of the intended information to be communicate, and (2) *operation* - the action to be performed on the proposed medium.
- *Business value* prioritizes conflicting polices and is represented by: (1) *businessGroup* - the associated grouping

```
<csmpPolicy>
  <scope>
    <service>"Communication Object"</service>
    <operation>"selection"</operation>
    <active>"true"</active>
  </scope>
  <condition>
    <medium>"video"</medium>
    <operation>"request"</operation>
  </condition>
  <businessValue>
    <businessGroup>"general"</businessGroup>
    <value>96</value>
  </businessValue>
  <decision>
    <mediumAttribute>"numberOfUsers"</mediumAttribute>
    <connectionID>"connectionID"</connectionID>
    <minVal>"connectionID.users"</minVal>
  </decision>
</csmpPolicy>
```

Fig. 3. Example of communication service policy.

for the specific policy, and (2) *value* - a numeric value that represents the policy's priority in the group.

- *Decision* defines the policy's desired outcome and expected behavior of the communication. This is represented as (1) *mediumAttribute* - the property of the medium that is to be focused on, (2) *connectID* - optionally specify a connection to be targeted, and (3) either *maxVal* and/or *minVal* - a set of parameters that state the acceptable range for the specified medium attribute.

Figure 3 shows an example of a communication service policy using the XML schema previously described.

Policy:

- 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*

In Figure 3 the service is a Communication Object, an instance of a communication framework class that will be provided to NCB to realize a requested communication service. The management operation we wish to perform is that of selecting an instance of a Communication Object. The condition that will cause this policy to be used is the action of requesting a medium of type video. The business value has priority 96 in the general group. The decision that satisfy this policy is that the attribute "numberOfUsers", which provides the numeric value for the number of users supported on that medium for a particular communication framework, not be less than the number of users currently in the specified connection.

C. Detailed Design

The detailed design for the self-configuring component of NCB is presented in this subsection focusing on both the static and dynamic views. Figure 4 shows the structure of the CSM and the NCB bridge including the classes representing the APIs for the communication frameworks. The main classes in Figure 4 are:

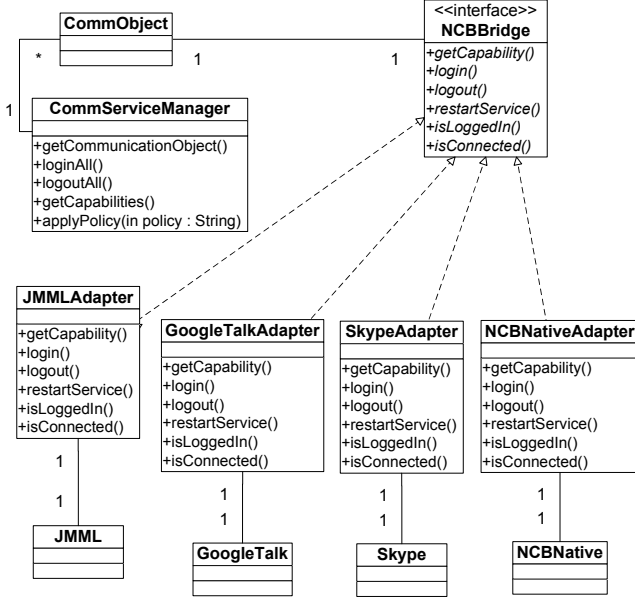


Fig. 4. Class diagram of CSM and NCB bridge.

- `CommServiceManager` - is the controller class responsible for coordinating the self-configuration activities. The sensor interface consist of: `getCapabilities` - returns a list of all media types available with the enabled communication frameworks, and `getCommunicationObject` - returns an instance of a communication framework that implements the `NCBBridge`. The effector interface consist of: `applyPolicy` - adds and activates policies that direct the configuration decisions of the CSM, and `loginAll`, `logoutAll` - enables the NCB to login to and logout from all underlying communication frameworks.
- `CommObject` - contains a single instance of a communication framework object wrapped in an adapter. We do not show the specifics of the `CommObject` class.
- `NCBBridge` - defines the methods for the sensor/effector interfaces used by the adapters, see Figure 2. The sensor interface consist of: `getCapability` - provides a list of all media types available for the communication framework, `isLoggedIn` - checks if the NCB is currently logged in to a specific communication framework, and `isConnected` - checks if the NCB is currently connected to a specific communication framework. The effector interface for the manageability endpoints are: `login`, `logout` and `restartService`. The latter method, `restartService` restarts the existing communication service.
- Adapter classes (`JMMLAdapter`, `GoogleTalkAdapter`, `SkypeAdapter`, and `NCBNativeAdapter`) - implementations of the `NCBBridge` interface that wraps their respective underlying API's providing manageability and service to CSM.
- API classes (`JMML`, `GoogleTalk`, `Skype`, and

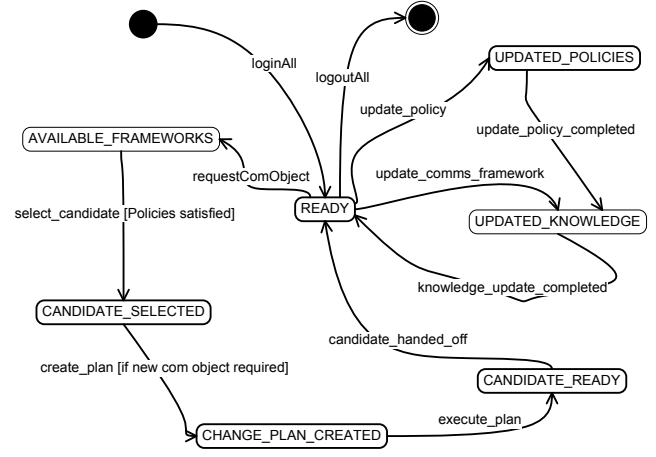


Fig. 5. CSM State Machine.

`NCBNative`) - provides the actual APIs for the different communication frameworks. The specifics of these APIs are not shown in the Figure 4.

Figure 5 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 `getCapabilities` 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.

The `getCommunicationObject` generates a `requestComObject` and transitions to `AVAILABLE_FRAMEWORKS` after getting the list of inventoried communication frameworks. Thereafter it selects a candidate from the list that does not violate the active policies of the CSM, this transitions to the state `CANDIDATE_SELECTED` in Figure 5. 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.

V. COMMUNICATION SCENARIO

In this section we present a communication scenario that details how our prototype realizes a communication service using the self-configuring NCB.

Scenario: There is a collaborative conference call to four members of a project group that includes Peter, Yingbo, Yali and Andrew. The four-way conference utilizes PC-to-PC voice

and chat features. After some discussion the group need to bring another person into the conference, this party will be communicating with the group via a fixed line phone.

Realization of Communication: A model is designed for a four way audio conference and is transformed to a series of platform specific requests by the UCM, the layer of the CVM responsible for that process. We assume that Yingbo, Yali and Andrew are all in Peter's contact list for each communication provider. The NCB receives this series of request for the realization of a four way communication as follows, with Peter as the initiator:

- `createSession(sessID)`
- `addParty(sessID, "Yingbo, Yali, Andrew")`
- `sendMedia(sessID, "audio")`

NCB calls `getCommunicationObject` with the media type parameter set to "audio" and the partylist, "Yingbo,Yali, Andrew", from `addParty`. Note the partylist contains the user ids for the users by the communication service providers. There are four supporting communication frameworks used in the prototype which include: Skype, supporting five users for audio; NCBNative, supporting unlimited users for audio; JML, no audio support; and GoogleTalk, supporting two users for audio. The high level policies specified are:

- 1) Choose a communication framework that provides conferencing on the required medium supporting at least the party size.
- 2) When the addition of a new party exceeds the support of an existing session, choose a new communication framework.
- 3) If a call is billable, use NCBNative support
- 4) Choose Skype if it is one of the candidate frameworks.

With these policies active in the CSM, the following occurs:

- From the inventory collected and categorized, Skype and NCBNative would be candidates that do not violate the first policy.
- The second policy would not be violated by any of the candidates.
- The third policy is not violated by the two candidates
- The fourth policy is satisfied by Skype.

Skype is chosen as the preferred communication framework and the communication started between the participants.

When the communication model changes to a five way audio conference with the addition of the fixed line call, NCB is sent a series of request to effect the change. CSM through its inventory will know that Skype bills for fixed line calls, so a new communication framework is needed to realize the five way audio communication.

- From the inventory collected and categorized, Skype and NCBNative would be the candidate that do not violate the first policy.
- The second policy would not be violated by the two candidates.
- The third policy is satisfied by NCBNative only.

- The fourth policy is not violated by this sole candidate.

NCBNative can support five way, the remote users are informed of the change in framework and the Skype session is discarded with a new session created on NCBNative with the five participants. The communication is resumed with the group continuing their discussions.

VI. RELATED WORK

Nicol et al. [1] developed a prototype distributed multimedia application using ready-made component technology [1], the methodology however does not address multiple communication providers. There are several products such as Trillian [14], Qnext [15], and Eclipse Communication Framework (ECF) [16] that provide platforms to support multiple communication providers. These products facilitate the aggregation of the accounts of the communication providers into one interface. They offer some way of adding additional communication providers to their platforms, usually as a plug-in. Products like Trillian and Qnext are free to download, but proprietary and closed source with no way to reuse their communication components for building more extensive communication applications.

ECF encourages the reuse of high-level communication components and provides a cross-protocol API [16] that utilizes plug-ins from various communication providers. ECF provides a set of high-level abstractions, rather than yet another messaging API, which allows the reuse of high-level communications components (instant messaging, file sharing, video conferencing, etc.) in varying application contexts and UIs. ECF however does not provide the self-configuration of the plug-ins from the various communication providers therefore lacking the flexibility for choosing the best communication framework on-the-fly. It is also worth mentioning that the CVM uses a model-driven approach to creating communication applications.

VII. CONCLUDING REMARKS

In this paper we leverage the open platform APIs provided by three communication application providers to improve the ability of the Communication Virtual Machine (CVM) to provide a more comprehensive set of communication services. In order to realize these services we applied the concept of self-configuration to the Network Communication Broker (NCB), the layer of the CVM that interacts with the underlying communication networks. We presented a design of the NCB that highlights self-configuration and describe a communication scenario that uses the prototype of the NCB. Our future work involves extending the autonomic feature of the NCB to include self-healing and self-optimization using the concept of self-testing to guarantee the goodness of the feedback's decision before reconfiguration. Additionally we will be evaluating the efficacy of the design with respect to performance and configuration effort.

VIII. ACKNOWLEDGMENTS

This work was supported by the National Science Foundation under grants IIS-0552555 and HRD-0317692. The authors would like to thank the participants of the FIU REU Summer 2007 program for their contributions to this work.

REFERENCES

- [1] J. R. Nicol, Y. S. Gutfreund, J. Paschetto, K. S. Rush, and C. Martin, "How the internet helps build collaborative multimedia applications," in *Communications of the ACM (January 1999)*, 1999, pp. Vol. 42, No.1,79–85.
- [2] T. Berners-Lee, "Www: Past, present, and future," *Computer*, pp. 69–77, 1996.
- [3] Skype Limited, "Skype developer zone," Feb. 2007, <https://developer.skype.com/>.
- [4] Google, "Google talk," September 2007, <http://www.google.com/talk/>.
- [5] Microsoft Corp., "Windows live messenger," September 2007, <http://get.live.com/messenger/overview>.
- [6] Y. Deng, S. M. Sadjadi, P. J. Clarke, C. Zhang, V. Hristidis, R. Rangaswami, and N. Prabakar, "A communication virtual machine," in *Proceeding of COMPSAC 06*. IEEE Computer Society, 2006, pp. 521–531.
- [7] C. Zhang, M. Sadjadi, W. Sun, R. Rangaswami, and Y. Deng, "A user-centric network communication broker for multimedia collaborative computing," in *2nd International Conference on Collaborative Computing (IEEE/ACM CollaborateCom)*, Nov. 2007.
- [8] JML Development Team, "Java msn messenger library," September 2007, <http://sourceforge.net/projects/java-jml>.
- [9] IBM Autonomic Computing Architecture Team, "An architectural blueprint for autonomic computing," IBM, Hawthorne, NY, Tech. Rep., June 2006.
- [10] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–52, January 2003.
- [11] M. J. Masullo and S. B. Calo, "Policy management: An architecture and approach," in *Proceedings of the IEEE First International Workshop on Systems Management*, April 1993, pp. 13–26.
- [12] M. Sloman, "Policy driven management for distributed systems," *Journal of Network and Systems Management*, vol. 2, pp. 333–360, 1994.
- [13] D. Kaminsky, "An introduction to policy for autonomic computing," IBM Autonomic Computing, Mar. 2005, <http://www.ibm.com/developerworks/autonomic/library/ac-policy.html>(September2007).
- [14] Cerulean Studios, "Trillian software," Sept. 2007, <http://www.ceruleanstudios.com>.
- [15] Qnext Corporation, "Qnext," Sept. 2007, <http://www.qnext.com/>.
- [16] ECF Development Team, "Eclipse communication framework project," September 2007, <http://www.eclipse.org/ecf/>.