Towards Policy Driven Self-Configuration of User-Centric Communication

Paola Boettner
Dept. of Computer Science
Wellesley College
Wellesley, MA 02481
pboettne@wellesley.edu

Mansi Gupta
Dept. of Computer Science
Bryn Mawr College
Bryn Mawr, PA 19010
mgupta@brynmawr.edu

Yali Wu and Andrew A. Allen
School of Computing and Info. Sciences
Florida International University
Miami, FL 33199, USA
{aalle004,ywu001}@cis.fiu.edu

ABSTRACT

The convergence of various multimedia communications that includes voice, video and data presents many opportunities for enabling unified communication but paradoxically leads to inefficiencies for the user as the communication may become complex. Model driven technologies such as the Communication Virtual Machine (CVM) propose to reduce such complexity through the use of models to define a user's communication needs. The CVM was extended to utilize multiple common-off-the-self(COTS) communication APIs such as Skype and Smack which we refer to as communication frameworks, to promote service and network independence. However, the user's current priorities and context play no role in the use and configuration of these frameworks.

In this paper we introduce the notion of policy driven self-configuration into the CVM, where high level user policies play a major role in the self-configuration decisions. We show the results of a feature analysis of the domain which guided the policy definition process. We also provide a design of the policy driven self-configuring architecture.

Keywords

Autonomic Computing, Self-Configuration, Communication

1. INTRODUCTION

Unified communication, an amalgamation that includes video, voice and data, provides opportunities for customized communication. This unified communication can paradoxically lead to inefficiencies [12] for the user as the communication may become more complex. Additional complexity is introduced when the users have to manage each new method of communication between parties. Work has begun to address this complex issue with user-centric [12] technologies such as the Communication Virtual Machine (CVM). CVM is a model driven development (MDD) paradigm for the realization of communication intensive application models defined using a Communication Modeling Language (CML).

ACMSE '09 March 19-21, 2009, Clemson, SC, USA.

To further reduce the complexity of developing and using communication intensive application, Allen et al. [1] proposed an architecture for the self configuration of multiple communication frameworks to provide low-level networking support for the Network Communication Broker (NCB), the layer of CVM that provides a network independent API. Communication frameworks, such as Skype [15] and GoogleTalk [5], are commodity-off-the-shelf(COTS) communication APIs that have been made publicly available by their respective companies. They provide a framework for the development of more sophisticated communication services by third parties. Using this architecture, the CVM would select the most appropriate framework to provide the required communication service for the group of users involved in the communication.

To enhance the user-centric behaviour of NCB, a user's current context and priorities should be included in the selection of an appropriate communication framework. Self configuration of communication frameworks in CVM should therefore be directed by high-level user policies, which would possibly express preferences, constraints and conditions for their use. This policy driven self-configuration of the frameworks and services would guide at runtime the realization of communication models at the lowest level of CVM. It would also support decision processes such as switching frameworks in the event that a framework can not satisfy a new request from the user or a framework or its service becomes unavailable. This would require the extension of the architecture and its implementation to support policy driven self-configuration. The major contributions of this paper are the following:

1. A Feature analysis of available communication frameworks
2. Definition of the structure of user-centric communication (UCC) policy
3. Conceptual design of a policy driven user-centric communication layer

In Section 2 we provide background on the main concepts including the CVM technology. Our analysis of the UCC domain is given in Section 3, while in Section 4 we discuss our current work on policy driven self-configuration. Section 5 presents our prototype policy designer, related work in Section 6 and we conclude in Section 7.

2. BACKGROUND

In this section we provide overviews of self-configuration in Autonomic Computing and user-centric communication

as well as introduce the Communication Virtual Machine (CVM).

2.1 Self-Configuration

Autonomic Computing (AC), which addresses the problems associated with the increasing complexity of computing systems as well as the evolving nature of software, is the ability of computing systems to manage themselves and adapt to changes in accordance with business policies and objectives [7, 10]. The essence of autonomic computing systems is self-management and it is derived from a combination of four broad capabilities: self-configuring, self-healing, self-optimizing, and self-protecting.

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 [3]. 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 [13]. While there is still debate surrounding the classification of policies, the existence of the goal and action policy types are generally accepted. *Goal policies* specify a desired state or criteria that characterize a set of desired states, where any member of this set is equally acceptable[11]. An example of such a set would be "Use communication framework that supports audio streaming and video streaming". This set would contain communication frameworks that support audio and video streaming such as Skype [15], GoogleTalk [5] and NCBNative. Any member of this set would satisfy the stated goal. *Action policies* specify the action to be taken when the system is currently in a given state [11]. An example of this would be "If system bandwidth is below 40 percent, replace video steaming with remote user's avatar". The system state of bandwidth less than 40 percent should cause the removal of video streaming.

2.2 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 as the user may be less effective if the interaction with the communication becomes unnatural. Interaction can be viewed as any mutual, reciprocal exchange between people, technologies and processes [12]. Interaction becomes unnatural as the context and priority of the communication will change according to the specific domain(healthcare, disaster management). Additionally each new communication channel and application increases the complexity as it introduces a new way of contacting others. Complexity can hinder rather than enhance communication [12] with the user having the responsibility for managing their communication and adapting them as needed. 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

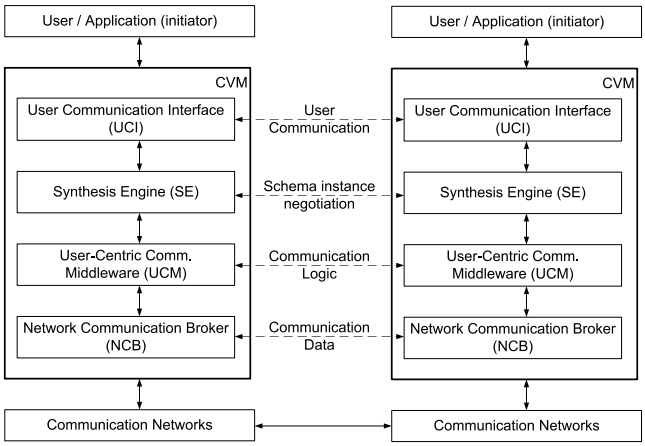

Figure 1: Layered architecture of the CVM.

of objects of its communication space at a fixed moment of time [16]. The user-centric communication approach is about matching the communication resources with the individual's needs in the context of the specific domain and adapting accordingly, reducing the complexity to the user.

2.3 Communication Virtual Machine

Deng et al. [4] 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: *User Communication Interface* (UCI) - which provides a modeling environment for users to specify their communication requirements using CML ; *Synthesis Engine* (SE) - which 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; *User-centric Communication Middleware* (UCM) - which executes the communication control script and coordinates the delivery of communication services to users; and *Network Communication Broker* (NCB) - which provides a network independent API to the UCM that masks the heterogeneity and complexities of the underlying network.

3. UCC DOMAIN ANALYSIS

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) [9] 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 [9]. 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 section we present our survey of existing communication frameworks, and use the result of the survey to define the domain of user-centric communications via the FODA methodology, the result of which is used as the basis for designing user-centric communication policies.

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
Chat (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	0	1*	1*	0	1	1
Phone to PC	?	0	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 2: Survey of Communication Frameworks

3.1 Survey of Communication Frameworks

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. In a previous paper [1], we had surveyed three communication frameworks with respect to each one's supporting features. The small coverage of surveyed systems hinders the usefulness of this method. To extend our sample set to yield more meaningful results, we have chosen a wider and more representative set of communication frameworks that are currently popularly used in industry. Moreover, we incorporated more distinguishing features of these frameworks, such as message archiving and importing contact list. Figure 2 shows the result of our 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, we incorporated additional features 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. We present the feature analysis on this surveyed group in section 3.2.

3.2 Analysis of Communication Framework

In this subsection, we present the application of FODA method to the user-centric communication domain. We focus 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. The domain models describe elements of systems in a given domain from the point of view of the "problem space" [9]. 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 3 shows the feature diagram resulting from feature modeling. We focus on user centric communication applications as our 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 3), and optional features denoted by empty circles, as in `file transfer`(second left feature in Figure 3).

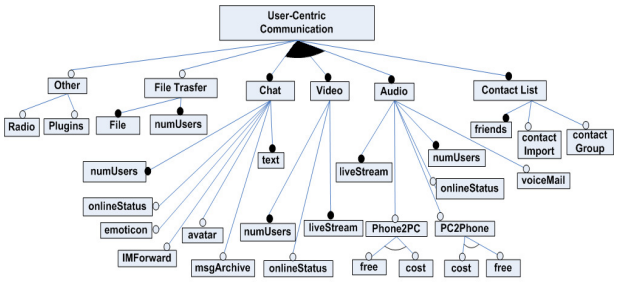

Figure 3: Feature Diagram for Frameworks

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 we call the main features, have their own sub features, either optional or mandatory, representing properties of the main features. The hierarchical structure goes down until we do not have further properties to be captured. The feature diagram is extensible as we refine and iterate the FODA process in the future.

Feature analysis helps us to capture the domain model in terms of the various characteristics or considerations of the domain, which will be used as the basis for designing policies that will guide how such "considerations" are satisfied by means of self-configuration. We will explain details of user-centric communication policies in the next subsection.

3.3 User-Centric Communication Policy

Section 3.2 shows a diverse set of features that characterize the user-centric communication domain, however there exist users who have no interest in a fine grained configuration of their communication. Our work on the feature analysis of this 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 we call *User-Centric Communication Policies*. User-centric communication policies are policies that aid the simplification of communication while enhancing the user experience.

In Figure 4 we show the inputs and output for a self configuration request in the CVM. An *initial configuration*, a user's request for a new connection or a new service (step 1 in Figure 4), would be evaluated and a set of configuration commands (step 2 in Figure 4) generated for the configuration of the framework. A *dynamic configuration* (step 3 in Figure 4), a user's request to reconfigure an existing connection or service, would be evaluated producing a set of configuration commands (step 4 in Figure 4) 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. A *reactive configuration* (step 5 in Figure 4), monitored states of the

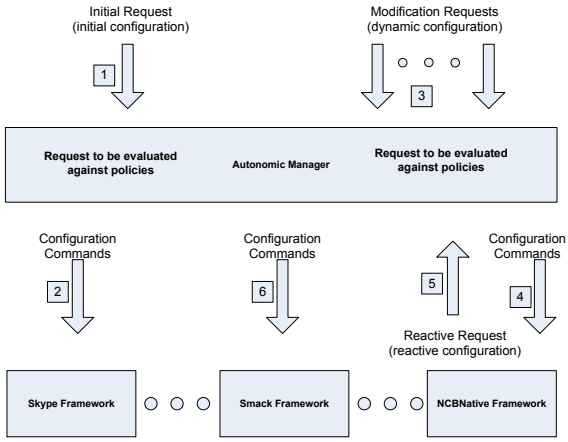

Figure 4: Self-Configuration steps for Frameworks

framework that are out-of-band , would be evaluated and a set of configuration commands (step 6 in Figure 4)generated to reconfigure or replace a framework.

User-centric communication policies would have to be developed for the classes of configuration discussed previously. We present the policy definition in section 4.1.

4. POLICY DRIVEN SELF-CONFIGURATION

We present our work on policy definition for user-centric communication and the application of such policies.

4.1 Communication Policy Definition

There are four common elements identified when studying the various policy standards [8]:

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

We extended these elements to the collaborative communication domain based on the results of our feature analysis and the need to express the range of user-centric communication policies discussed in Section 3.3:

- *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 5, consist of: *service* - the applicable communication component, and *operation* - the intended management action.
- *Condition* represents the trigger for the consideration of the policy represented as: *medium* - the carrier of the intended information to be communicated, and *operation* - the action to be performed on the proposed medium.
- *Business value* prioritizes conflicting polices and is represented by: *businessGroup* - the associated grouping for the specific policy, and *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: *mediumAttribute* - the property of the medium that is to be focused on, *connectID* - optionally specify a connection to be targeted, and either

```
<csmPolicy>
    <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>
</csmPolicy>
```

Figure 5: Communication Service Policy.

maxVal and/or *minVal* - a set of parameters that state the acceptable range for the specified attribute.

Figure 5 shows an example of a communication service policy using XML.

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*

4.2 Conceptual Design

Figure 6 shows the conceptual view of the evaluation process. The domain specific nature of CVM assures us of a much more narrow focus in terms of features and attributes. With that in mind, our evaluation process uses a lightweight policy decision mechanism at its core as detailed in [1]. The *policy authoring tool*, an instance of which is our simplified designer (see Section 5), creates policies that will guide the management of the communication resources. These policies are stored in the *policy repository* (left in Figure 6).

As shown in Figure 6, an incoming user's request for service is first handled by the *request decomposition* component. A request is decomposed into the feature, sub-features and attributes to be evaluated. This process also identifies the target of the request which will aid in the lookup and selection of the relevant policies. The policy evaluator takes this decomposed set along with the returned set of relevant policies and the current state of the targeted resource to produce a set of configuration commands as its decision. These commands are passed to the touchpoint manager.

The *touchpoint manager* (right side in Figure 6) is responsible for the low-level management of the communication resources, these resources are the communication frameworks as well as the specific services offered by the frameworks. The touchpoint manager will also generate reactive requests, request generated due to out-of-band states, which are evaluated in a similar manner as the user's request.

4.3 Applying Communication Policies

As stated in section 2.3, CVM utilizes the model driven approach for the creation and the realization of collaborative communication. User-centric communication policies provide a way to introduce user specific high level goals to

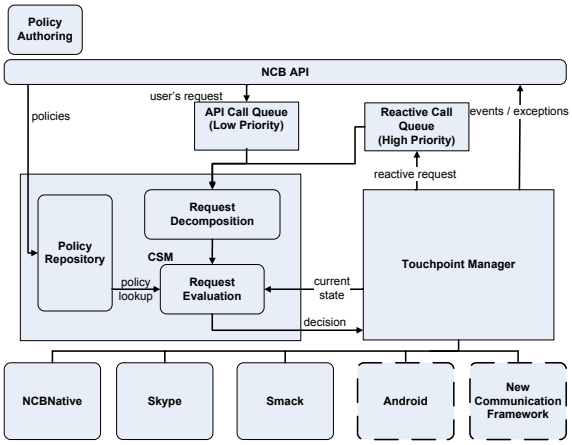

Figure 6: Conceptual Design for Policy Evaluator

guide this realization process. We now present a communication scenario that details how this is achieved.

Scenario: There is a collaborative conference call between two members of a project group, Paola and Mansi. The two-way conference utilizes the PC-to-PC audio feature. After some discussion the group need to bring another person, Yali, into the conference via PC-to-Phone. The user-centric communication policies defined for the CVM are:

- `P1:{S:con C:any D:cUsers<=fUsers }`
- `P2:{S:con C:PC2Ph D:useByDef NCBNative}`

Where S:Scope, C:condition, D:decision. *P1*, a goal policy, states that for any type of request (for example audio or video) the frameworks considered should support the number of users in the connection. We effect this goal by creating a set of candidate frameworks. *P2*, an action policy, states that for a specific type of request (PC2Phone) the default framework should be NCBNative. In this scenario, NCBNative is our most cost effective framework for fixed line calls.

Realization of Communication: A communication model is created for a two-way audio-video conference and goes through a series of transformations that results in API calls to the NCB. The interested reader can see [17, 14] for the details of the model creation and transformation processes. 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. We assume that Mansi and Yali are all in Paola's contact list for each communication provider. NCB receives this series of request for the realization of a two-way communication as follows, with Paola as the initiator:

- `createSession(conID)`
- `addParty(conID, "Mansi")`
- `sendMedia(conID, "audio")`

The three user requests are queued and serviced in order by the *request decomposition* component. The `createSession` request is to create and map a session in this layer of the CVM to the upper layer's connection identifier *conID*.

The `addParty` call provides connection and remote users identification and is decomposed as such. Connection is identified as the scope (targeted concern) and so policies *P1* and *P2* are retrieved from the *policy repository* by the *request evaluation* component, see Figure 6. Policy *P2* specifically targets connections which requested PC2Phone feature, however we have not yet added that feature to the connection so this policy would not be relevant at this time. Using policy *P1* the connection's count of users is evaluated against the possible candidate frameworks retrieved by the *request evaluation* component from the *touchpoint manager*. This results in a candidate set whose members all satisfy policy *P1*.

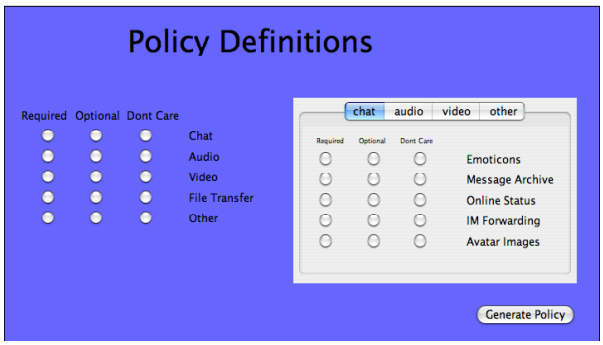

Figure 7: Screenshot of Simplified Policy Designer

The `sendMedia` call adds the audio feature to the connection and policy *P1* is needed to guide the decisions for this request. The candidate set is further reduced to those that support the "audio" feature with at least two users, since JML does not satisfy this requirement it is no longer included in the set. From the resulting candidate set the first member of the set, Skype, is chosen and the communication realized using that communication framework.

When the communication model changes to a three-way audio conference with the addition of the fixed line call, it results in another set of `addParty` and `sendMedia` calls being generated. The `sendMedia` call is decomposed as the targeted connection with features "audio" and "PC2Phone". Policy *P2* is evaluated resulting in a third analysis of the candidate set. Since NCBNative is included in the candidate set the decision is passed to the *touchpoint manager* to use NCBNative as the communication framework. The *touchpoint manager* is responsible for safely switching the communication framework from its current framework of Skype to the new framework, NCBNative.

5. A SIMPLIFIED POLICY DESIGNER

Figure 7 shows the current implementation of the policy designer prototype. It allows users to define the various characteristics of the communication that he wants to incorporate, in terms of "optional", "required" and "do not care" features. These features are the expected characteristics of the system that the policy evaluator would enforce when self-configuring the underlying communication framework. Due to the hierarchical nature of features, the required sub-features of a main feature also need to be specified. Note that currently, the policy designer is only able to generate goal policies that define the expected features of system behaviors. When action policies and other more complex types of policies are put into consideration, the policy designer could be extended correspondingly, while the architecture for policy evaluation and self-configuration remains.

When the administrator is done with selection of the fea-

tures and the "Generate Policy" button is pressed, the policy designer generates policies in the form of XML documents and store them in the local policy repository. At runtime, these goal policies would be transformed to action policies that will guide system behaviors via self-configuration.

6. RELATED WORK

Self-configuration based upon policies to eliminate labor-intensive processes performed by experts has been in existence for decades. However, most of these configuration tasks are limited to network level configuration management. In Boutaba et al [2], 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, as opposed to our focus of 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.

In the realm of user-centric communication policies, some initial work has been proposed. In Gorton [6], 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. Their work differs from ours in two ways: (1) the scope of the policy: while their policies allow control over the who, what, where, when and how much of service delivery, our policies focuses more on the configuration of communication resources to best support users' services and (2) they lack a formal representation of the policy, while our policy definition is based on a FODA approach with an XML definition.

In [1], an initial idea for self-configuring user-centric communication services was been presented. However, the policies are defined in an ad-hoc manner, with no systematic process to help identify the essential characteristics of the domain that policies are based on. We complement that work by using the FODA approach to capture the various characteristics of the domain that form the basis for policy definition, and a simplified GUI policy authoring tool for producing XML policies based on the definition.

7. CONCLUDING REMARKS

In this paper we have extended the self-configuration capabilities of the CVM by introducing user-centric policies that guide its autonomic behaviour. These user-centric policies provide a way for users to state preferences and concerns that influence service provisioning. We presented the results of our survey of popular communication frameworks and the feature analysis of the user-centric communication domain which guided our policy definition. We also presented a design of the NCB that highlights the policy driven self-configuration and described a communication scenario that uses the prototype of the NCB. Our future work involves extending user-centric policy definition to represent other autonomic features such as self-healing and self-optimization. Additionally we will be evaluating the efficacy of the design with respect to performance and configuration effort.

Acknowledgments

This work was supported by NSF grants IIS-0552555 and HRD-0317692. The authors would like to thank their mentor Dr. Peter J. Clarke and the members of the 2008 REU program.

8. REFERENCES

[1] A. A. Allen, S. Leslie, Y. Wu, P. J. Clarke, and R. Tirado. Self-Configuring User-Centric Communication Services. In *(ICONS 2008)*, pages 253–259. IEEE, April 2008.

[2] R. Boutaba, S. Omari, A. Pal, and S. Virk. Selfcon-an architecture for selfconfiguration of networks. *Journal of Communications and Networks*, 3:317–323, 2001.

[3] H. Chen, S. Hariri, and F. Rasul. An innovative self-configuration approach for networked systems and applications. In *(CTS 2006)*, pages 537–544. IEEE, 2006.

[4] Y. Deng, S. M. Sadjadi, P. J. Clarke, C. Zhang, V. Hristidis, R. Rangaswami, and N. Prabakar. A communication virtual machine. In *COMPSAC 06*, pages 521–531. IEEE Computer Society, 2006.

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

[6] D. Gorton. Transforming the customer experience with user centric networking. 2008.

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

[8] 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)`.

[9] 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, CMU Software Engineering Institute, November 1990.

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

[11] J. O. Kephart and W. E. Walsh. An artificial intelligence perspective on autonomic computing policies. *POLICY 2004*, 0:3, 2004.

[12] P. Lasserre and D. 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).

[13] 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.

[14] R. Rangaswami, S. Sadjadi, N. Prabakar, and Y. Deng. Automatic generation of user-centric multimedia communication services. In *IPCCC 2007*, pages 324–331. IEEE, 2007.

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

[16] S. A. Sven van der Meer, Stephan Steglich. User-centric Communications. In *ICT 2001*, pages 425–444. Special Sessions, 2001.

[17] Y. Wang, P. J. Clarke, Y. Wu, A. A. Allen, and Y. Deng. Realizing communication services using model-driven development. In *(SEA 07)*, 2007.