Service Learning and Service Risk Management in Internet Telephony

Xiaotao Wu
Department of Computer Science
Columbia University
New York, New York 10027
Email: xiaotaow@cs.columbia.edu

Henning Schulzrinne
Department of Computer Science
Columbia University
New York, New York 10027
Email: hgs@cs.columbia.edu

***Abstract*— Internet telephony can introduce many novel communication services, however, this novelty puts learning burden on users. We have developed an intelligent service creation environment which can automatically create services by learning from communication behavior. The service creation environment models communication services as decision trees and uses the Incremental Tree Induction (ITI) algorithm for learning. We use our Language for End System Services (LESS) to describe learned results and have implemented a simulation environment to verify the learning algorithm. We noticed that automatically generated services may introduce undesirable side-effects that can cause users to lose calls, money, or privacy. Thus, we did an analysis on service risk management for LESS-based services and proposed several approaches for fail-safe services.**

I. Introduction

One of the key advantages of Internet telephony is its ability to provide many innovative communication services, however, novelty is a barrier to entry. Many users may not be aware of what services are available and not know how to customize or create their own services. It will be a great help to users if there is a service creation environment that can automatically generate desired services.

We consider the learning process for automatic service creation is applicable in Internet telephony systems because of the following reasons. Internet telephony signaling can convey more information to end users. The information allows people to make sensible call decisions which are usually impossible in PSTN networks. In addition, Internet telephony end systems usually have more computational capabilities so they can execute programmable call handling services and can easily collect users' communication behaviors for learning. The extended abilities of Internet telephony end systems make service learning practical and useful.

We believe many services can be generated automatically, for example, phone spam filtering, call handling based on callee's location (as defined in GEOPRIV location object format [8]), media capabilities (as defined in User Agent Capability Extension to Presence Information Data Format [7]) and status (as defined in Rich Presence Extensions to the Presence Information Data (RPID) [12]), call routing based on caller's address, time based call handling, and call handling based on priority, preferred language and subject of calls. There are a lot of parameters for call decision making. Users may not even know some of the parameters, not to speak of how to use them. In addition, creating such services manually is tedious and error-prone. Thus, we describe how proxies and end systems can create call handling rules automatically by learning from observed user behavior. We focus on services using the Session Initiation Protocol (SIP) [11].

We need to handle four tasks for service learning: *representing communication behaviors, finding a learning algorithm, representing the learned results, and handling service risk management.* We choose to use decision trees to represent communication behaviors and detail the rational of the choice in Section II. Section III gives the criteria on which decision tree learning algorithm to use. Because the Language for End System Services (LESS) [18] is simple, safe, and tree-like, we decide to use LESS scripts to describe learned results. Section IV shows how to convert a decision tree to a LESS script. In most cases, communication services bring great conveniences to users. However, services may have undesirable side-effects and users may have the risk of losing calls, money, or privacy. In Section V, we give a comprehensive analysis on service risks and propose several approaches to create fail-safe services. Section VI describes how we integrate the service learning functions in our SIP user agent, SIPC [17]. Section VII concludes the paper and discusses our future work.

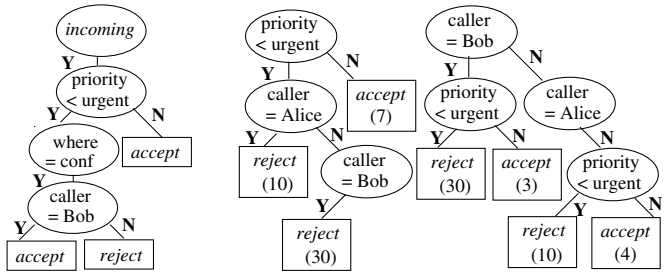
II. Representing users' communication behaviors

We consider using binary decision trees to represent communication behaviors is the most suitable way for service learning, among other possible user behavior representations, such as finite state machines, use case maps [2], rule sets, and Bayesian networks [6].

We can clearly identify learning targets in a binary decision tree due to its simple structure. As shown in Figure 1(a), the learning target is to find non-leaf tree nodes that can best partition a user's behavioral data. It is hard to identify learning targets with using finite state machines or use case maps.

The cost of simplicity is to sacrifice some functionality. A binary decision tree is not Turing-complete so it cannot represent all possible communication behaviors. However, we believe service learning should only focus on commonly used services. Learning of complicated services is error prone and usually causes confusion.

0-7803-8939-5/05/$20.00 (C) 2005 IEEE



(a) Service decision tree (b) Different tree node order

Fig. 1. Representing communication behaviors as decision trees

Using rule sets to represent communication behaviors is another viable way. Rules sets and decision trees are exchangeable, but decision trees are more efficient for execution.

Bayesian networks [6] are commonly used in machine learning, however, we found Bayesian networks are not appropriate for communication service learning because when making a communication decision, different factors may not be independent of each other. For example, domain-based call rejection is not independent of caller-based call rejection. It is hard to build a Bayesian network with co-related decision factors [6].

III. Decision tree learning

To perform learning, we need to identify input variables as well as expected outputs. As we mentioned earlier, the learning target is to find non-leaf tree nodes that can best partition a user's behavioral data. We define possible user communication behavior as the actions a user can perform, such as `accept`, `reject`, `proxy`, `redirect` for incoming call handling, `call` for outgoing call setup, `transfer`, `hold`, `mute` for mid-call handling, `notify` for event handling, and `power-on`, `power-off` for networked appliance control. The actions will be leaf tree nodes. The non-leaf nodes represent the parameters of a call and its context to match, such as the identity, status, location, media capabilities, and device type of the caller and callee, the time, priority, subject, preferred language of the call. The value of the parameters can be acquired from SIP INVITE messages and SIP event notification messages [10]. Once we get these data, we can perform decision tree learning.

A. Criteria for choosing a learning algorithm

There are several existing decision tree learning algorithms [9], [15]. To choose an appropriate algorithm for our communication decision tree learning, we defined several requirements for the algorithm.

The algorithm should perform incremental learning. Behavior learning is a dynamic process. New samples and new rules may get introduced from time to time, and old rules may get broken. Building decision trees in an incremental way reflects the dynamics of people's communication behaviors.

The algorithm should have an appropriate tree quality measurement mechanism [16]. As shown in Figure 1(b), two trees represent the same user behavior over 47 calls. They have the same height. The left tree has fewer leaves and fewer nodes, but the right tree has a shorter path for most of the training samples (e.g., 30 matches for rejecting calls from Bob with priority lower than urgent). In terms of simplicity, the left tree is better. In terms of efficiency, the right tree is better. Because service learning is to help users to create and understand communication services, a simpler decision tree is preferable. For example, in Figure 1(b), the left tree is the desired result.

The algorithm must have an effective pruning method to filter decision errors (noise) because there are so many random factors that can affect communication behaviors. In addition, due to the dynamics of people's communication habits, the algorithm should be able to save and reactivate pruned branches because a pruned branch may get restored with additional learning samples.

B. Incremental Tree Induction (ITI) algorithm

Based on the above requirements, we chose to use the Incremental Tree Induction (ITI) [15] algorithm. ITI can map an existing tree and a new training example to a new tree based on several tree transformation mechanisms. The average incremental cost of updating a tree is much lower than the average cost of building new trees from scratch each time.

ITI uses an algorithm called *direct metric tree induction* to map one tree to another based on tree quality measurements. The algorithm introduces four tree quality measurement matrices, namely *expected number of tests*, *minimum description length*, *expected classification cost*, and *expected misclassification cost* [15]. We choose to use the *expected number of tests* matrix for our service learning process because that reflects the simpleness of a decision tree.

ITI also introduces a pruning technique named *virtual pruning*. Instead of deleting pruned branches, it uses one bit on every node to mark prune decision. If the bit value is true, the branch rooted at the node should be hidden from users. When new learning samples added, ITI will use the unpruned tree for learning so pruned branches may get reactivated for useful services.

C. Accuracy of the ITI algorithm

There are two ways to measure the accuracy of the ITI algorithm for communication service learning: one is to perform real world usage testing and the other is to do simulation.

To perform real world usage testing, we need to perform service learning based on the data collected from real users, and submit the learned services to users for evaluation. This requires a large deployment of VoIP systems and to have people using the systems for their daily communications. Currently, people are making efforts to deploy VoIP systems, such as the SIP.edu [14] initiatives in Internet2 community and many companies have deployed VoIP solutions in their intranet. However, most of the deployments are still in their testing stage, not used as primary communication means for people's daily usage. Thus, we consider simulation a more

appropriate way, but real world usage testing is in our future work plan when our local VoIP deployment becomes available.

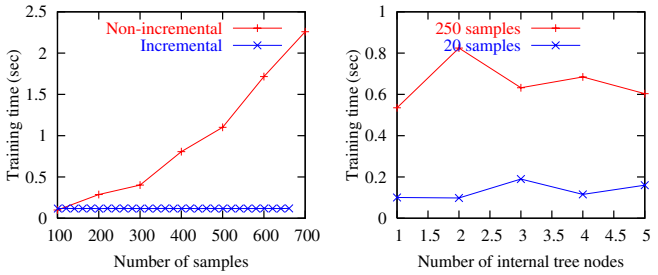
We have built a simulation environment to generate random simulated calls. In the simulation environment, the distribution of call arrival time, priority of calls, and caller's address are adjustable. By default, we use Poisson distribution for call arrival time, uniform distribution multiplied by weights for setting the priority of calls and picking callers: 98% of the calls are of normal priority, 0.9% urgent calls, 0.1% emergency calls, and 1% non-urgent calls. 30% of the calls are from `user1`, 10% each from `user2`, `user3`, and `user4`, the rest from all the other users. The simulation environment also simulates people's daily life, such as time for meal and sleep, and can load calendars in iCal [5] format for meeting and appointment information.

We then randomly created several expected services, applied the services to the simulated calls, and simulated the call handling process. For example, if we have an expected service `"reject all calls from sip:bob@example.com"`, when we apply the service to the generated calls, all calls from `sip:bob@example.com` get rejected. The other calls will be handled based on the default simulation setup. With the default setup, if a call arrives when the simulated user is sleeping, 50% of the calls will not get answered; when the user is in meal, 20% of the calls will not get answered; if the user is in an appointment, 40% of the calls get rejected; in normal cases, 85% of the calls get accepted. The default setup will introduce reasonable noise for service learning. Once the simulation completes, we save the simulated communication behavior data in C4.5 [9] format. We can then use the ITI algorithm to learn from the behavior data. The pruned trees generated by ITI algorithm should match the expected services. We tested 40 expected services, each applied to 300 simulated calls. The tests show that 80% of the learning results exactly match their expected services, 10% of the learning results represent the expected services in different ways, and 10% of the learning results do not match the expected services. The mismatch comes from the randomness of the simulation data. For example, if an expected service is `"accept all emergency calls"`, however, since only 0.1% of calls are emergency, there may not be emergency calls in the simulated call set, the learning algorithm cannot infer the expected service. Based on the simulation, we consider the ITI algorithm fits our need.

D. Performance of the ITI algorithm

Figure 2(a) and Figure 2(b) show the performance of the ITI algorithm based on 700 simulated calls, running on an IBM ThinkPad laptop with Linux operating system, a 1GHz Intel Pentium III Mobile CPU, and 256MB memory. The expected services in the testing make call decisions based on the priority of calls, caller's addresses, and callee's ongoing activities when receiving calls.

Figure 2(a) shows that the training time for non-incremental training increases polynomially as more training samples are added in, while the training time for incremental training is a constant if only training on the new added samples. For 20 new added samples, the training time for incremental training is about 0.12 seconds, which is quick enough to provide automatic service creation to users.



(a) Fast vs. incremental training (b) 20 vs. 250 incremental samples

Fig. 2. Training time of the ITI algorithm

Figure 2(b) shows that for incremental training, the training time is independent of the number of internal nodes in the expected services. This is because ITI algorithm uses the *virtual pruning* technique to handle the overfitting problem so it always constructs complete decision trees. The number of nodes in a pruned tree will not affect the training time of building a complete decision tree.

IV. Using LESS to Represent Learned Results

Since the Language for End System Services (LESS) [18] has a tree-like structure, it is straightforward to convert learned results to LESS scripts. The non-leaf tree nodes can be converted to LESS `switches`, the leaf nodes can be converted to LESS `actions`, and the root nodes can be converted to `triggers`. For example, we can convert the decision tree in Figure 1(a) to the LESS script below:

```
<less><incoming>
<priority-switch><priority less="urgent">
 <where-switch type="civil"><where LOC="conf">
  <address-switch field="origin"
   subfield="display">
   <address is="Bob"><accept/></address>
   <otherwise><reject/></otherwise>
  </address-switch>
 </where></where-switch>
 </priority>
 <otherwise><accept/></otherwise>
</priority-switch>
</incoming></less>
```

The generated LESS scripts can be loaded into a user agent's service engine to automate call processing.

V. Service Risk Management

Automatically generated services may introduce unexpected side-effects which users may not be aware of. Users using the services may have to take the risk of losing calls, money, or privacy. However, risk is a part of any activities and can never be eliminated, nor can all risks ever be known. The opportunity for better communication experience cannot be achieved without taking risk. What we should do is "to balance

the possible negative consequences of risk against the potential benefits of its associated opportunity."[13]

We consider that service risk should be handled in the service creation stage. As Charette [4] has said, "Risk engineering does not deal with future decisions, but with the future of present decisions."

There are three key elements for risk management: [13][3]

Identify: We must find the cause of service risks.
Analyze: We must evaluate potential loss of risks.
Resolve: We must help users to act on the risks.

A. Identify service risks

There are two factors that comprise a risk: [13] "loss resulting from its occurrence and probability or likelihood that it will occur." We will identify possible losses in this section and analyze the probability in next section. We consider the following losses are possible for communication services, namely losing communication, compromising privacy, costing money, and distracting attention.

Since we use LESS to represent services, the loss can only occur when performing LESS actions. There are only very limited actions defined in LESS. Thus, we can easily identify the cause of service risks by checking the relationship between LESS actions and potential losses, as shown in Table I.

Loss	LESS actions may cause the loss
Losing communication	reject, redirect, transfer, disconnect, accept on a wrong branch media-update, hold
Releasing privacy	call, accept, notify
Costing money	accept, redirect, or transfer calls to a device with higher charge rate
distracting attention	alert, accept, appliance control

TABLE I
LESS ACTIONS MAY CAUSE LOSS

B. Analyze service risks

Three main steps can be used to analyze service risks: estimating the probability of a risk, evaluating the impact of a risk, and determining the overall risk of a service.

1) Probability: We can estimate the probability of a risk quantitatively as well as qualitatively. When we use the ITI algorithm for learning, it can associate the number of matching instances to each tree branch. We can use these numbers to perform quantitative analysis to estimate how likely a risk may happen in a tree branch. The bigger the number, the more likelihood the branch may introduce service risks. We can also perform qualitative analysis. In a decision tree, the internal nodes along the path to a leaf node comprise the conditions under which a corresponding service action will execute. To estimate the probability of a risk, we must analyze the internal nodes. In LESS, the internal nodes are switches, such as `address-switch`, and `time-switch`. Different arguments of a switch have different risk characteristics. For example, in an `address-switch`, domain matching or wildcard matching is more likely to introduce risk than specific URL matching. Changing the arguments with higher risk probability to the arguments with lower risk probability can help to resolve service risks. We will not detail the analysis for every LESS switch in this paper.

2) Impact: Different risks have different impact to users. The impact can be categorized as negligible, marginal, critical, and catastrophic [1], or as very low, low, moderate, high, and very high in commonly-used classification. In general, for communication services, we consider the risks causing irreversible loss have higher impact than the other risks. Thus, compromising privacy, losing communication, and costing money are more severe than distracting attention. We can set the impact of compromising privacy and losing communication as high, costing money as moderate, and distracting attention as low.

3) Overall risk: Service risks are not independent to each other. For example, when a user trys to preserving communications, he may take the risk of losing privacy, money, or attention. To resolve service risks, we must try to avoid or mitigate risks with higher impact, even though it may introduce risks with lower impact, the overall risk can still get lowered.

C. Resolve risks

Since a risk is composed of the probability of its occurrence and the loss of its outcome [3], to resolve or mitigate it, we can reduce its probability or reduce its potential loss. The design rule of the resolution is to ensure that either the overall impact of service risks is low, or users can get alerted of risk occurrence and the correction is viable. There are several risk resolution options we can use [3], e.g., risk avoidance, risk transfer, risk impact reduction, and building contingency plan.

1) Risk avoidance: We can avoid a service risk by reducing the probability of its occurrence. As we discussed in Section V-B.1, we can adjust the arguments of switches in a LESS decision tree to reduce risk probability.

2) Risk transfer: Service risks can be transferred to another person. For example, in a meeting, instead of rejecting calls, a boss can transfer calls to his secretary so the risk of losing calls decreased, but the risk of being distracted is transferred to his secretary.

3) Risk impact reduction: For risk impact reduction, we only focus on risks with high loss impact, such as the risk of losing communication and compromising privacy, as defined in Section V-B.2. Table I shows the action that may cause these risks. In this paper, we use `reject` action as an example showing how to reduce the risk of losing communication.

Automatically rejecting a call is dangerous. People reject calls due to many different reasons, but in most cases, due to people's availability and the subject of calls. Nowadays, there are many different communication methods. We noticed that different communication methods have different characteristics in terms of disturbance factors and expected response time as shown in Table II. This gives us a hint that when a call gets rejected, we may use other communication methods to mitigate

the potential loss. For example, if a user has a service to automatically reject all calls when he is in a meeting. We can modify the service as below to make it safer: In an unimportant meeting, change the alerting style from ringing to vibrating; in an important conference, forward the call to a voicemail and provide voicemail indication. If the user is doing a presentation in a conference and does not want to be disturbed in any way, forward the call to voicemail without providing any indication. Note that vibrating a user's end device or showing voicemail indication may somewhat distract the user's attention. But as we discussed in Section V-B.3, the overall risk will still get lowered.

Action	Disturbed entities	Expected response time
Call/ringing	Callee and others	seconds - minutes
	Immediate attention otherwise lose the call	
Call/vibrating	Callee	seconds - minutes
	Immediate attention otherwise lose the call	
Instant messaging	Callee	minutes - hours
	Immediate attention but delay is ok	
Voicemail/Email with indication	Callee	minutes - days
	Immediate attention but delay is ok	
Voicemail/Email without indication	Callee	hours - days
	No immediate attention	

TABLE II
THE CHARACTERISTICS OF DIFFERENT COMMUNICATION METHODS

4) Building contingency plan: It is very important to make the correction viable when service risk occurred. To achieve this, we should log all the actions performed by service scripts. For each service action, we should define a convenient way for users to remedy the potential loss caused by the action. For example, if a call gets automatically rejected, the service execution environment should allow users easily retrieve the caller's number and call back.

VI. IMPLEMENTATIONS

We have implemented a SIP user agent called SIPC [17]. SIPC has a built-in LESS engine that can execute LESS scripts and provide communication services. SIPC can support a large set of functions and collect all the required information we need for service learning [19]. SIPC records all the call related information and user performed actions in a file in C4.5 [9] format, the same format as what we used in the simulation environment we introduced in Section III-C. Once the number of new call records reaches a threshold, e.g., 20 new call records, SIPC will use the ITI algorithm to perform service learning. The learned result will be saved in two files, one is a LESS script that representing the pruned tree, the other is a binary file that representing the whole tree with virtual pruning marks. SIPC will then load the LESS script in its LESS engine to provide communication services. We are still working on building a user friendly interface to help users to handle communication service fail-safe.

VII. CONCLUSION AND FUTURE WORK

This paper proposes a model for communication service learning. Because usually end users are not trained for communication service creation, they may not know how to customize or create their own services. Service learning can help them by generating communication services automatically based on their communication behaviors. Because people are still rarely using VoIP systems as primary means for their daily communications, we cannot perform real world testing for the ITI learning algorithm, but we had done accuracy and performance measurements for the algorithm in our simulation environment and proved that it fits our learning model. We noticed that communication services may introduce unexpected side-effects and proposed several options for service risk management. But we still need to find out how to present potential risks and possible solutions to users in a friendly and easy to understand way.

REFERENCES

[1] Air Force Systems Command/Air Force Logistics Command Pamphlet 800-45. Software risk abatement. Technical report, United States Air Force, 1988.

[2] Daniel Amyot. Use case maps as a feature description notation. In *FIREwork Feature Constructs Workshop*, May 2000.

[3] Barry W. Boehm. Software risk management: principles and practices. *IEEE Software*, 8(1):32–41, January 1991.

[4] R.N. Charette. Information technology risk engineering. In *SEI/NSIA Workshop on Software Risk*, February 1991.

[5] F. Dawson and D. Stenerson. Internet calendaring and scheduling core object specification (icalendar). RFC 2445, Internet Engineering Task Force, November 1998.

[6] Finn V Jensen. Bayesian networks and decision graphs. In *Bayesian networks and decision graphs*. Springer, 2001.

[7] M. Lonnfors and K. Kiss. User agent capability presence status extension. Internet Draft draft-ietf-simple-prescaps-ext-00, Internet Engineering Task Force, February 2004. Work in progress.

[8] J. Peterson. A presence-based GEOPRIV location object format. Internet Draft draft-ietf-geopriv-pidf-lo-01, Internet Engineering Task Force, February 2004. Work in progress.

[9] J. R. Quinlan. *C4.5: Programs for machine learning*. Morgan Kaufman, 1993.

[10] A. B. Roach. Session initiation protocol (sip)-specific event notification. RFC 3265, Internet Engineering Task Force, June 2002.

[11] J. Rosenberg, Henning Schulzrinne, G. Camarillo, A. R. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: session initiation protocol. RFC 3261, Internet Engineering Task Force, June 2002.

[12] Henning Schulzrinne. RPID – rich presence information data format. Internet Draft draft-ietf-simple-rpid-01, Internet Engineering Task Force, February 2004. Work in progress.

[13] Roger L. Van Scoy. Software development risk: Opportunity, not problem. Technical report, Carnegie Mellon University, September 1992.

[14] SIP.edu. Sip.edu cookbook. http://web.mit.edu/sip/sip.edu/.

[15] Paul E. Utgoff, Neil C. Berkman, and Jeffery A. Clouse. Decision tree induction based on efficient tree restructuring. *Machine Learning*, 29(1):5–44, 1997.

[16] S. M. Weiss and C.A. Kulikoswski. *Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems*. Morgan Kaufman, 1991.

[17] Xiaotao Wu. Columbia university SIP user agent (sipc). http://www.cs.columbia.edu/IRT/sipc.

[18] Xiaotao Wu and Henning Schulzrinne. Programmable end system services using SIP. In *Conference Record of the International Conference on Communications (ICC)*, May 2003.

[19] Xiaotao Wu and Henning Schulzrinne. sipc, a multi-function SIP user agent. In *7th IFIP/IEEE International Conference, Management of Multimedia Networks and Services (MMNS)*, pages 269–281. IFIP/IEEE, Springer, October 2004.