A PROGRAMMABLE POLICY MANAGER FOR CONVERSATIONAL BIOMETRICS

Ganesh N. Ramaswamy  Ran D. Zilca  Oleg Alecksandrovich
IBM Thomas J. Watson Research Center
Yorktown Heights, New York

ABSTRACT

Conversational Biometrics combines acoustic speaker verification with conversational knowledge verification to make a more accurate identity decision. To manage the added level of complexity that the multi-modal user recognition approach introduces, this paper proposes the use of verification policies, in the form of Finite State Machines, which can be used to program a policy manager. Once a verification policy is written, the policy manager interprets the policy on the fly, and at every turn in the session decides dynamically whether to accept the user, reject the user, or continue to interact and collect more data. The policy manager allows for any number of verification engines to be plugged-in, thereby adding to the flexibility of the framework. As a result, system developers have significant freedom to design user verification solutions, and a wide variety of application-specific, transaction-specific and user-specific constraints can be addressed using a generic system. The paper also describes a prototype implementation of a Conversational Biometrics solution with the proposed programmable policy manager.

1 INTRODUCTION

The IBM Conversational Biometrics technology [5][6] enables highly accurate and convenient speaker recognition, by combining text-independent acoustic voiceprint match with conversational knowledge match. Conversational Biometrics brings several key advantages to the application by combining who the user is (represented by the voice) with what the user knows (represented by passwords or personal/application-related knowledge). The combination is obtained using a single conversational natural language interface including both acoustic text-independent speaker verification and speech recognition engines, with additional resources for natural language understanding, dialog management and text-to-speech synthesis. Other examples of combining acoustic speaker verification and speech recognition could be found in [2], [3] where verbal information (knowledge) verification is used to collect enrollment data for subsequent text-dependent acoustic verification, and in [4] where a nonstandard SVM was used to combine speaker verification with verbal information verification. Another example of combining speaker and speech recognition for applications such as name-based identity claim recognition could be found in [1].

In this paper, we introduce a major new component to the Conversational Biometrics framework that allows for programmability and customizability of Conversational Biometrics solutions, while also increasing the accuracy, flexibility and convenience aspects. The incorporation of the new component, termed verification policy manager, is a significant departure from earlier static methods to combine voiceprint and knowledge that remain fixed for all applications, transactions and users. The freedom to program the policy manager is enabled through the use of verification policies that implement a dynamic verification session, as well as through the ability to specify and add verification objects and user models. Complete programming and customization of the verification process now becomes possible, thereby addressing a wide variety of user-specific constraints (e.g. a “goat” user with high acoustic false reject rates may prefer lower weights for the voiceprint match), transaction-specific requirements (e.g. a million dollar transaction will require a higher degree of security as opposed to a ten dollar transaction, etc.), application-specific requirements (e.g. a banking application may need different security questions compared to a travel application), and other similar constraints. As a result, system developers and administrators have significant control over the system behavior.

In our initial prototype, the verification policies are implemented as Finite State Machines (FSM). Different goals can be accomplished using different verification policies or the same verification policy can be used to accomplish different system behavior in response to dynamically changing session context. Although in our initial implementation we have focused on voiceprint match and knowledge match, additional verification objects (caller-id, possession of key, fingerprint, face, etc.) can be added at any time since the flexible architecture supports plug-in verification engines. Additional verification engines can also be used for the same verification object (e.g. we can have more than one voiceprint verification engine for the same voiceprint object). Finally, additional customization is possible by incorporating user-specific preferences in the individual user models which can further modify the system behavior.

The remainder of this paper is divided into four sections. In Section 2, we introduce the components of the proposed framework and describe an example high-level architecture. Section 3 contains the details of our prototype implementation, including example verification policies, verification objects and user models. Section 4 concludes the paper.

2 COMPONENTS AND ARCHITECTURE

We suggest the following general framework for user verification. When a verification session is initiated, a policy is first loaded and the verification session commences in the attempt to verify the identity of the user up to the degree of certainty that the policy allows. The policy is comprised of a set of rules, operations and conditions on the session context. The context records all relevant variables for the verification process, such as the user’s name, the current state in the verification policy that is in effect, the history pertaining to the verification objects (e.g. questions) that have been invoked, scores and outcomes associated with the invocations, transaction specific requirements (e.g. desired level of accuracy, etc.), user specific requirements (e.g. a user having a cold may prefer not to rely on voiceprint match, etc.), and other physical and logical variables (e.g. quality of a voice channel, etc.). The context may also record other variables that represent verification scores from external verification sources (e.g. fingerprint recognition score from an external engine). In addition to system default variables, user-defined variables may be added at any time to support new
verification policies and verification objects. At every turn, the context is updated based on the user’s response. The rules and conditions of the policy are applied to the values in the context, resulting in one of three possible actions: accept the user, reject the user, or continue to interact – in which case the policy determines the next verification object to be invoked.

Figure 1 shows an example of a Conversational Biometrics system with programmable policy management. In the client-server architecture, the verification client interfaces with the application and maintains the session context described earlier, either locally or at a central location. The verification server manages the following three basic types of persistent data, accessed through the data manager component:

- **Verification Objects**: a pool of verification challenges that can be used for the purpose of verifying the identity of users, such as the user’s voiceprint and topics for knowledge verification.
- **Verification Policies**: policies that govern the interaction between the user and the system and between the various verification objects. Any number of verification policies could be written to satisfy a wide variety of user-specific, transaction-specific or application-specific authentication needs, including needs that change in real-time.
- **User Models**: models created when a user enrolls in the system, using the inputs provided by the user (e.g. samples of voice, answers to personal questions, etc.), or acquired through other means (details of past transactions, balance in most recent bill, etc.).

The user models may be updated in real-time when needed, such as when a new bill is issued and the balance changes or when more voice samples are available. An individual user model contains information regarding all verification objects relevant to that user, including any user preferences related to the verification objects.

The verification server consists of a verification policy manager and a set of verification engines. Each verification engine operates on a given verification object or a family of verification objects. The flexible architecture allows for easy addition of new verification engines and verification objects. Verification engines to be added could be of a new type or an existing type. For example, a face recognition engine could be added to a verification server that previously consisted of voiceprint and fingerprint recognition engines, or we could add a second voiceprint recognition engine (which could be from a different manufacturer, for example). Similarly, new verification objects could be added to new verification engines or existing verification engines (such as adding a new question to an existing knowledge verification engine). Verification objects can be invoked only if a corresponding verification engine exists.

The verification policy manager interprets a verification policy for a given user model, and drives the entire authentication process. It receives the current context from the verification client, operates on it, incorporates updated status of current verification objects, and returns an updated context to the verification client along with the specification of the next step to be taken during the verification process. In the implementation to be described in Section 3, the verification policy manager is responsible for invoking states in a Finite State Machine, interpreting the conditions of the state machine and branching to the next state. The verification policy manager is the entity that makes the final accept or reject decision for the authentication process and in some cases may also make intermediate decisions when appropriate.

### 3 Example Implementation

This section contains a description of a prototype Conversational Biometrics Server with programmable policy management built at IBM. The prototype consists of two verification engines: acoustic text-independent speaker verification engine based on [7] and a knowledge verification engine based on [6]. In this implementation, the verification client communicates with the

---

**Figure 1: Example System Architecture**

**Figure 2: Example Registry of Verification Objects**

```xml
<verification_object_base>
  <object name="DOB" Engine="Knowledge" Type="QA" Prompt=none Perplexity="100"></object>
  <object name="CALLER_ID" Engine="Telephony" Type="Caller_ID" Prompt=none Perplexity="20"></object>
  <object name="VOICE_PRINT" Engine="Voiceprint" Prompt=none Perplexity="1000"></object>
  <object name="COLOR" Engine="Knowledge" Type="QA" Prompt="What is your favorite color?" Perplexity="5"></object>
  <object name="CAR_COLOR" Inherit_from="COLOR" Engine="Knowledge" Type="APP_NUM" Prompt="What is the color of your car?" Perplexity="100"></object>
  <object name="CUR_BALANCE" Engine="Knowledge" Type="APP_NUM" Prompt="What is the approximate balance in your account?" Perplexity="100"></object>
  <object name="LAST_TRANSACTION_DATE" Engine="Knowledge" Type="APP_STR" Prompt="What is the date of your last transaction?" Perplexity="100"></object>
</verification_object_base>
```
verification server using an XML message interface. Example functions supported by the interface include starting and ending enrollment sessions and verification sessions; updating existing user models; continuing a verification session to determine the next state within policy; and adding, deleting and updating verification policies, verification objects, and context variables. Query functions to get information on active verification objects, verification policies, user models and context variables are also supported.

Figure 2 shows an example registry of verification objects, implemented in XML. The specification contains a description of all registered verification objects. The first object in this example is the date-of-birth (DOB) object, of type question-answer (QA) (i.e. used with the knowledge verification engine). A suggested prompt is included, and the “perplexity” representing the difficulty associated with the verification object is specified. The second object in this example is Caller-ID, which can be used in telephony applications. The third object is the voiceprint object. The fourth and fifth objects illustrate the hierarchical nature of the specification, whereby the CAR_COLOR object inherits default properties from the parent object COLOR. Finally, the last two objects are examples of dynamic verification objects, for which no value is present in the user model, and the correct responses are obtained from the application.

Figure 3 shows an example of a user model, also implemented in XML. The user model contains description of verification objects for which the user has provided enrollment data. The first object is the Caller-ID object, for which this user’s correct response is 914-945-3000 in this example. The user’s preference for this object may be optionally included and used by the verification policy in selecting objects with higher preference when possible. The second and third objects are similar. The fourth object (color of car) has two responses, perhaps because this user has two cars and either response may be accepted as the correct answer. The fifth object is the voiceprint object, for which model parameters are needed, which may be stored in a file, and the filename is included. The last two objects do not have any responses included because they are dynamic verification objects, and the current

![Figure 3: Example User Model](image)

responses have to be obtained from the application. Any of these objects can be added, deleted or updated in real-time.

Figure 4 shows the state machine for an example verification policy and the corresponding XML specification could be found in Figure 5. First, in Figure 5, context variables such as “curBalance” and “lastTransactionDate” are initialized with default values to handle dynamic verification objects such as “CUR_BALANCE” and “LAST_TRANSACTION_DATE”. These variables will subsequently be updated using inputs from the application. The variable “minVoiceprintScore” is used to specify the minimum score acceptable for the voiceprint match. This default value for this variable is specified by the verification policy and different policies may have different default values (e.g. a stricter policy may have higher minimum score). The default value may be overwritten by values obtained from either the user model (to account for user-specific requirements) or by the application (to account for transaction-specific or application-specific requirements).

Next, a set of conditions relevant to the policy are specified, which will be subsequently used to determine state transitions or evaluate verification objects. For example, the condition “ONE_OK” is satisfied if the total number of verification objects invoked so far is 1 (_curObjectNum = 1) and it was a match or there were no mismatches (_curWrongNum=0). Another example of a different kind is the condition “CUR_BALANCE_TEST” which can be used to evaluate the current balance (“CUR_BALANCE”) verification object, and in this case a 5% error is allowed.

Following the conditions, a set of states are defined. In this example, there are four states: ACCEPT, REJECT, START and ACCOUNT. ACCEPT and REJECT are the terminal states where the final verification decision is made. START is the initial state and contains verification objects “CALLER_ID”, “DOB” and “CAR_COLOR”. By default, verification objects are selected at random, but relative weights may be specified optionally to modify the probability that an object may be selected. Transition to the ACCOUNT state occurs if one of the first two conditions (ONE_OK or TWO_OK_ONE_BAD) is satisfied, and transition to the REJECT state occurs if the third

![Figure 4: State Machine for Example Verification Policy](image)
condition (TWO_BAD) is satisfied. In the ACCOUNT state, no weights are specified, so all three objects have equal probability of being selected at random. Further, evaluating these objects requires different tests to be used, and these are selected from the previously defined list. Based on the conditions satisfied, transition to either the ACCEPT state or the REJECT state occurs and the corresponding final decision is sent to the verification client. In some cases, intermediate decisions could be made at the intermediate states. For example, a temporary decision to accept the user for low security transactions can be made if certain conditions are satisfied in the START state, and the final decision can be made at the ACCEPT state.

4 CONCLUDING REMARKS

In this paper, we presented a novel policy manager for Conversational Biometrics that allows for programmability and customizability of multi-modal user verification solutions. In the initial prototype implementation, verification policies are implemented as Finite State Machines and specified using XML. The proposed framework offers significant freedom to system developers, and enables a wide variety of application-specific, transaction-specific and user-specific requirements to be supported using a single generic user verification system. Future work may include developing tools for policy design and analysis, as well as tools for estimating errors rate that can be used by the policy manager during the verification session.

Another direction for the future is to develop prototype systems with additional plug-in verification engines.

REFERENCES