Architecture for Affective Social Games

Derek J. Sollenberger and Munindar P. Singh

Department of Computer Science
North Carolina State University
Raleigh, NC 27695-8206, USA
Phone: 1-717-860-1809
Fax: 1-919-515-7896
{djsollen,singh}@ncsu.edu

Abstract. The importance of affect in delivering engaging experiences in entertainment and education is well recognized. We introduce the Koko architecture, which describes a service-oriented middleware that reduces the burden of incorporating affect into games and other entertainment applications. Koko provides a representation for affect, thereby enabling developers to concentrate on the functional and creative aspects of their applications. The Koko architecture makes three key contributions: (1) improving developer productivity by creating a reusable and extensible environment; (2) yielding an enhanced user experience by enabling independently developed applications to collaborate and provide a more coherent user experience than currently possible; (3) enabling affective communication in multiplayer and social games.

1 Introduction

Games that incorporate reasoning about their player's affective state are gaining increasing attention. Such games have been prototyped in military training [6] and educational [11] settings. However, current techniques for building affect-aware applications are limited, and the maintenance and use of affect is in essence handcrafted in each application.

We take as our point of departure the results of modeling affect based on appraisal theory. A fundamental concept of appraisal theory is that the environment of an agent is essential to determining the agent's affective state. As such, appraisal theory yields models of affect that are tied to a particular domain with a defined context. Therefore, each new problem domain requires a new affect model instance. A current and common practice has been to copy and edit a previous application (and, occasionally, to build from scratch) to meet the specifications of a new domain. This approach may be reasonable for research proofs-of-concept, but is not suitable for developing production applications.

Additionally, in order to more accurately predict the user's affective state, many affective applications use physical sensors to provide additional information about the user and the user's environment. The number and variety of sensors continues to increase and they are now available via a variety of public services (e.g., weather and time services) and personal commercial devices (e.g., galvanic skin response units).

Current approaches require each affective application to interface with these sensors directly. This is not only tedious, but also nontrivial as the application must be adjusted whenever the set of available sensors changes.

To address these challenges we propose a service-oriented architecture, called Koko, that compliments existing gaming engines by enabling the prediction of a gamer's affective state. When compared to existing approaches the benefits of our architecture are an increase in developer productivity, an enhanced user experience, and the enabling of affective social applications. Koko is not another model of emotions but a middleware from which existing (and future) models of affect can operate within. It provides the means for both game developers and affect model designers to construct their respective software independently, while giving them access to new features that were previously impossible. Further, Koko is intended to be used by affective models and applications that seek to recognize emotion in a human user. Whereas it is possible to use Koko to model the emotions of nonplaying characters, many benefits, such as using physical sensors, most naturally apply when human users are involved.

The Koko architecture is realized as a service-oriented middleware which runs independently from the game engine. The primary reason for this separation becomes more apparent when we discuss the social and multiplayer aspects of Koko. Such an approach is consistent with existing techniques for multiplayer access to a central game server.

1.1 Contributions

Any software architecture is motivated by improvements in features such as modularity and maintainability: you can typically achieve the same functionality through more complex or less elegant means [16]. Of course, an improved architecture facilitates accessing new functionality: in our case, the sharing of affective information and the design of social applications. Koko concentrates on providing three core benefits to affective game developers. In the remainder of this section, we elaborate on these benefits.

Developer Productivity. Koko separates the responsibility of developing an application from that of creating and maintaining the affect model. In Koko, the application logic and the affect model are treated as separate entities. By creating this separation, we can in many cases completely shield each entity from the other and provide standardized interfaces between them.

Additionally, Koko avoids code duplication by identifying and separating modules for accessing affect models and various sensors, and then absorbs those modules into the middleware. For example, by extracting the interfaces for physical sensors into the middleware, Koko enables each sensor to be leveraged through a common interface. Further, since the sensors are independent of the affect models and applications, a sensor that is used by one model or application can be used by any other model or application without the need for additional code.

Quality of User Experience. Abstracting affect models into Koko serendipitously serves another important purpose. It enables an enhanced user experience by providing data to both the application and its affect model that was previously unattainable, resulting in richer applications and more comprehensive models.

With current techniques it is simply not possible for separate applications to share affective information for their common users. This is because each application is independent of and thereby unaware of other applications in use by that user. By contrast, Koko-based applications can share common affective data through Koko. This reduces each application's overhead of initializing the user's affective state for each session, as well as providing insight into a user's affective state that would normally be outside the application's scope.

Such cross-application sharing of a user's affective information improves the value of each application. As a use case, consider a student using both an education and an entertainment application. The education application can proceed with easier or harder questions depending on the user's affective state even if the user's state were to be changed by participation in some unrelated game.

Affective Social Applications. In addition to enabling cross-application sharing of affective data, the Koko architecture enables cross-user sharing of affective data. The concept of cross-user sharing fits naturally into the realm of social and multiplayer games. Through Koko, a user if authorized may view the affective state of other members in their social circle or multiplayer party. Further, that information can potentially be used to better model the inquiring user's affective state.

1.2 Paper Organization

The remainder of this paper is arranged as follows. The background section reviews appraisal theory models. The architecture section then provides detailed description of the components that compose Koko. Finally, the evaluation section demonstrates the merits of the Koko architecture.

2 Background

Smith and Lazarus' [18] cognitive-motivational-emotive model, the baseline for current appraisal models (see Figure 1), conceptualizes emotion in two stages: appraisal and coping. *Appraisal* refers to how an individual interprets or relates to the surrounding physical and social environment. An appraisal occurs whenever an event changes the environment as interpreted by the individual. The appraisal evaluates the change with respect to the individual's goals, resulting in changes to the individual's emotional state as well as physiological responses to the event. *Coping* is the consequent action of the individual to reconcile and maintain the environment based on past tendencies, current emotions, desired emotions, and physiological responses [8].

Koko focuses on a section of the appraisal theory process (denoted by the dashed box), because Koko is intended to model emotions in human subjects. As a result, the other sections of the process are either difficult to model or outside Koko's control. For instance, the coping section of the process is outside Koko's control as it is an action that must be taken by the user.

A situational construal combines the environment (facts about the world) and the internal state of the user (goals and beliefs) and produces the user's perception of the world, which then drives the appraisal and provides an appraisal outcome. This appraisal outcome is made up of multiple facets, but the central facet is "Affect" or current

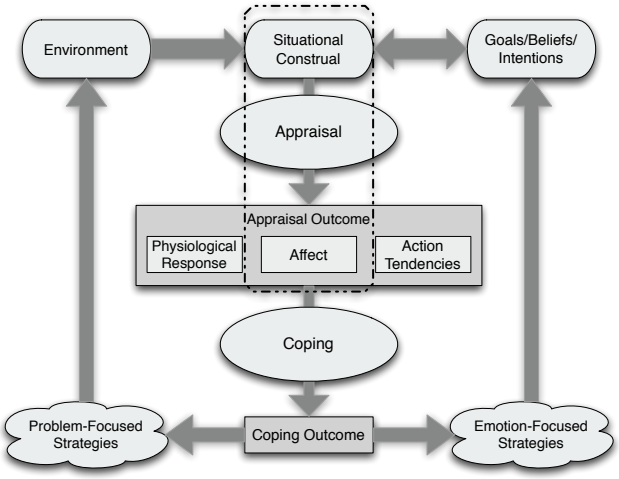

Fig. 1. Appraisal Theory Diagram [18]

emotions. For practical purposes, "Affect" can be interpreted as a set of discrete states with an associated intensity. For instance, the result of an appraisal could be that you are happy with an intensity of α as well as proud with an intensity of β. Unfortunately, selecting the set of states to use within a model is not an easy task as there is no one agreed upon set of states that covers the entire affective space.

Next, we look at three existing appraisal theory approaches for modeling emotion. The first is a classic approach which provides the foundation for the set of affective states used within Koko. The final two are contemporary approaches to modeling emotion with the primary distinction among them being that EMA focuses on modeling emotions of nonplaying characters whereas CARE concentrates on measuring human emotional states.

OCC. Ortony, Clore, and Collins [14] introduced the so-called OCC model, which consists of 22 types of emotions that result from a subject's reaction to an event, action, or object. The OCC model is effectively realized computationally, thus enabling simulations and real-time computations. Further, the OCC's set of emotions have turned out to cover a broad portion of the affective space. Elliot [2] expanded the set of emotions provided by the OCC to a total of 24 emotions. Koko employs this set of 24 emotions as its baseline affective states.

EMA. The EMotion and Adaptation (EMA) model leverages SOAR [12] to extend Smith and Lazarus' model for applications involving nonplaying characters [7]. EMA monitors a character's environment and triggers an appraisal when an event occurs. It then draws a set of conclusions or *appraisal frames*, which reflect the character's perception of the event. EMA then assigns emotions and emotional intensities to each appraisal frame, and passes it to the coping mechanism, which decides the best actions to take based on the character's goals [5].

CARE. The Companion-Assisted Reactive Empathizer (CARE) supports an on-screen character that expresses empathy with a user based on the affective state of the user

[10]. The user's affective state is retrieved in real-time from a pre-configured static affect model, which maps the application's current context to one of six affective states. The character's empathic response is based on these states.

CARE populates its affective model offline. First, a user interacts with the application in a controlled setting where the user's activity is recorded along with periodic responses from the user or a third party about the user's current affective state. Second, the recorded data is compiled into a predictive data structure, such as a decision tree, which is then loaded into the application. Third, using this preconfigured model the application can thus predict the user's affective state during actual usage.

3 Architecture

Existing affective applications tend to be monolithic where the affective model, external sensors, and application logic are all tightly coupled. As in any other software engineering endeavor, monolithic designs yield poor reusability and maintainability. Similar observations have led to other advances in software architecture [16].

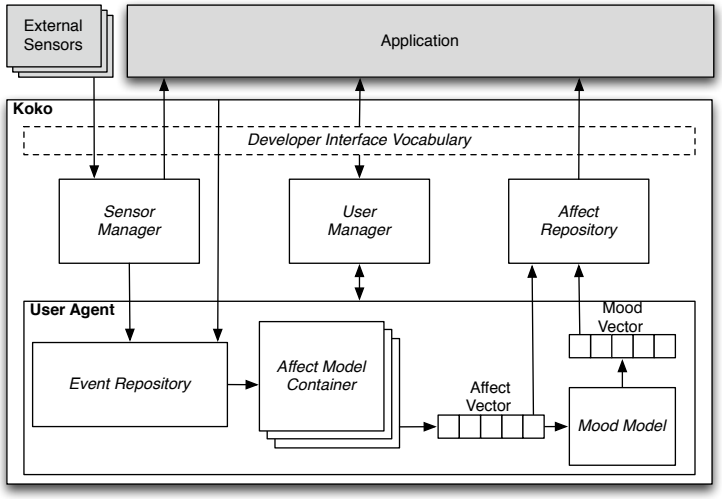

Fig. 2. Koko architectural overview

Figure 2 shows Koko's general architecture using arrows to represent data flow. The following portion of this section provide details on each of Koko's major components. Then after the groundwork of the architecture has been explained we elaborate on the formal interface definitions for the remainder of the section.

3.1 Main Components and their Usage

Koko hosts an active computational entity or *agent* for each user. In particular, there is one agent per human, who may use multiple Koko-based applications. Each agent has access to global resources such as sensors and messaging but operates autonomously with respect to other agents.

The Affect Model Container This container manages the affect model(s) for each agent. Each application specifies exactly one affect model, whose instance is then managed by the container. As Figure 3 shows, an application's affect model is specified in terms of the affective states as well as application and sensor events, which are defined in the application's configuration (described below) at runtime.

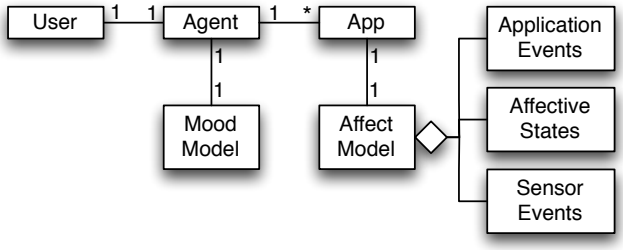

Fig. 3. Main Koko entities

Configuring the affect model at runtime enables us to maintain a domain-independent architecture with domain-dependent affect model instances. Further, in cases such as CARE, we can construct domain-independent models (generic data structures) and provide the domain context at runtime (object instances). This is the approach Koko has taken by constructing a set of generic affect models that it provides to applications. These affect models follow CARE's supervised machine learning approach of modeling affect by populating predictive data structures with affective knowledge. These affect models are built, executed, and maintained by the container using the Weka toolkit [20]. The two standard affect models that Koko provides use Naive Bayes and decision trees as their underlying data structures.

To accommodate models with drastically different representations and mechanisms, Koko encapsulates them via a simple interface. The interface takes input from the user's physical and application environment and produces an *affect vector*. The resulting *affect vector* contains a set of elements, where each element corresponds to an affective state. The affective state is selected from an ontology that is defined and maintained via the developer interface vocabulary. Using this ontology, each application developer selects the emotions to be modeled for their particular application. For each selected emotion, the vector includes a quantitative measurement of the emotion's intensity. The intensity is a real number ranging from 0 (no emotion) to 10 (extreme emotion).

Mood Model Following EMA, we take an *emotion* as the outcome of one or more specific events and a *mood* as a longer lasting aggregation of the emotions for a specific user. An agent's mood model maintains the user's mood across all applications registered to that user.

Koko's model for mood is simplistic as it takes in affect vectors and produces a *mood vector*, which includes an entry for each emotion that Koko is modeling for that user. Each entry represents the aggregate intensity of the emotion from all affect models associated with that user. Consequently, if Koko is modeling more than one application

for a given user, then the user's mood is a cross-application measurement of the user's emotional state.

To ensure that a user's mood is up to date with respect to recent events, we introduce a *mood decay formula* [15] as a way to reduce the contribution of past events. This formula reduces the effect that a given emotion has on the user's mood over time. We further augment our mood model with the concept of *mood intensity* from the work of Dias and Paiva [1]. The mood intensity sums all positive and negative emotions that make up the user's current mood, which is used to determine the degree to which a positive or negative emotion impacts a user. For example, if a user has a positive mood intensity then a slightly negative event may be perceived as neutral, but if the event were to recur the mood intensity would continue to degrade, thereby amplifying the effect of the negative event on the user's mood over time.

Affect Repository The affect repository is the gateway through which all affective data flows through the system. The repository stores both affect vectors (application specific) and mood vectors (user specific). These vectors are made available to both external applications as well as the affect and mood models of the agents. This does not mean all information is available to a requester, as Koko implements security policies for each user (see user manager). An entity can request information from the repository but the only vectors returned are those they have the permission to access.

The vectors within the repository can be retrieved in one of two ways. The first retrieval method is through a direct synchronous query that is similar to an SQL SELECT statement. The second method enables the requester to register a listener which is notified when data is inserted into the repository that matches the restrictions provided by the listener. This second method allows for entities to have an efficient means of receiving updates without proactively querying and placing an unnecessary burden on the repository.

Event Repository The event repository is nearly identical to the affect repository with respect to storage, retrieval, and security. Instead of storing vectors of emotion, the event repository stores information about the user's environment. This environmental information is comprised of two parts: information supplied by the application and information supplied by sensors. In either case, the format of the data varies across applications and sensors as no two applications or sensors can be expected to have the same environment. To support such flexibility we characterize the data in discrete units called *events*, which are defined on a per application or sensor basis. Every event belongs to an ontology whose structure is defined in the developer interface.

Sensor Manager Information about a user's physical state (e.g., heart rate and perspiration) as well as information about the environment (e.g., ambient light, temperature, and noise) can be valuable in estimating the user's emotional state. Obtaining that data is a programming challenge because it involves dealing with a variety of potentially arcane sensors. Accordingly, Koko provides a unified interface for such sensors in the form of the sensor manager. This yields key advantages. First, a single sensor can be made available to more than one application. Second, sensors can be upgraded trans-

parently to the application and affect model. Third, the overhead of adding sensors is amortized over multiple applications.

The sensor manager requires a plugin for each type of sensor. The plugin is responsible for gathering the sensor's output and processing it. During the processing stage the raw output of the sensor is translated into a sensor event whose elements belong to the event ontology in the developer interface vocabulary. This standardized sensor event is then provided as input to the appropriate affect models.

User Manager The user manager keeps track of the agents within Koko and maintains the system's security policies. The user manager also stores some information provided by the user on behalf of the agent. This includes information such as which other agents have access to this agent's affective data and which aspects of that data they are eligible to see. It is the user manager's responsibility to aggregate that information with the system's security policies and provide the resulting security restrictions to the sensor manager and the affect repository. Privacy policies are crucial for such applications, but their details lie outside the scope of this paper.

Developer Interface Vocabulary Koko provides a vocabulary through which the application developer interacts with Koko. The vocabulary consists of two ontologies, one for describing affective states and another for describing the environment. The ontologies are encoded in OWL (Web Ontology Language). The ontologies are designed to be expandable to ensure that they can meet the needs of new models and applications.

The *emotion ontology* describes the structure of an affective state and provides a set of affective states that adhere to that structure. Koko's emotion ontology provides by default are the 24 emotional states proposed by Elliot [2]. Those emotions include states such as *joy*, *hope*, *fear*, and *disappointment*.

The *event ontology* can be conceptualized in two parts, event definitions and events. An event definition is used by applications and sensors to inform Koko of the type of data that they will be sending. The event definition is constructed by selecting terms from the ontology that apply to the application, resulting in a potentially unique subset of the original ontology. Using the definition as a template, an application or sensor generates an event that conforms to the definition. This event then represents the state of the application or sensor at a given moment. When the event arrives at the affect model it can then be decomposed using the agreed upon event definition.

Koko comes preloaded with an event ontology (partially shown in Figure 4) that supports common contextual elements such as time, location, and interaction with application objects.

Consider an example of a user seeing a snake. To describe this for Koko you would create an event *seeing*, which involves an object *snake*. The context is often extremely important. For example, the user's emotional response could be quite different depending on whether the location was in a *zoo* or the user's *home*. Therefore, the application developer should identify and describe the appropriate events (including objects) and context (here, the user's location).

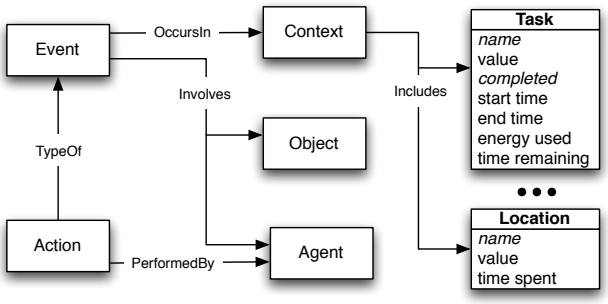

Fig. 4. Event ontology example

3.2 The Architecture Formally

Now that we have laid the groundwork, we describe the Koko architecture in more formal terms from the perspective of an application developer. The description is divided into two segments, with the first describing the runtime interface and the second describing the configuration interface. Our motivation in presenting these interfaces conceptually and formally is to make the architecture *open* in the sense of specifying the interfaces declaratively and leaving the components to be implemented to satisfy those interfaces.

Application Runtime Interface The application runtime interface is broken into two discrete units, namely, *event processing* and *querying*. Before we look at each unit individually, it is important to note that the contents of the described events and affect vectors are dependent on the application's initial configuration, which is discussed at the end of this section.

Application Event Processing. The express purpose of the application event interface is to provide Koko with information regarding the application's environment. During configuration, a developer defines the application's environment via the event ontology in the developer interface. Using the ontology, snapshots of the application's environment are then encoded as events, which are passed into Koko for processing. The formal description of this interaction is as follows.

$$\mathsf{userID} \times \mathsf{applicationID} \times \mathsf{applicationEvent} \mapsto \bot \tag{1}$$

Upon receipt, Koko stores the event in the agent's event repository, where it is available for retrieval by the appropriate affect model. This data combined with the additional data provided by external sensors provides the affect model with a complete picture of the user's environment.

Application Queries. Applications are able to query for and retrieve two types of vectors from Koko. The first is an application-specific affect vector and the second is a user-specific mood vector, both of which are modeled using the developer interface's emotion ontology. The difference between the two vectors is that the entries in the affect

vector are dependent upon the set of emotions chosen by the application when it is configured, whereas the mood vector's entries are an aggregation of all emotions modeled for a particular user.

When the environment changes, via application or sensor events, the principles of appraisal theory dictate that an appraisal be performed and a new affect vector computed. The resulting vector is then stored in the affect repository. The affect repository exposes that vector to an application via two interfaces. Formally, (here a square bracket indicates one or more)

$$\mathsf{userID} \times \mathsf{applicationID} \mapsto \mathsf{affectVector} \tag{2}$$

$$\mathsf{userID} \times \mathsf{applicationID} \times \mathsf{timeRange} \mapsto [\mathsf{affectVector}] \tag{3}$$

Additionally, an application can pass in a contemplated event and retrieve an affect vector based on the current state of the model. The provided event is not stored in the event repository and does not update the state of the model. This enables the application to compare potential events and select the one expected to elicit the best emotional response. The interface is formally defined as follows.

$$\mathsf{userID} \times \mathsf{applicationID} \times \mathsf{predictedEvent} \mapsto \mathsf{affectVector} \tag{4}$$

Mood vectors, unlike affect vectors, aggregate emotions across applications. As such, a user's mood is relevant across all applications. Suppose a user is playing a game that is frustrating them and the game's affect model recognizes this. The user's other affect-enabled applications can benefit from the knowledge that the user is frustrated even if they cannot infer that it is from a particular game. Such mood sharing is natural in Koko because it maintains the user's mood and can supply it to any application. The following formalizes the above mechanism for retrieving the mood vector.

$$\mathsf{userID} \mapsto \mathsf{moodVector} \tag{5}$$

Application Configuration Interface Properly configuring an application is key because its inputs and outputs are vital to all of the application interfaces within Koko. In order to perform the configuration the developer must gather key pieces of information and then supply that information to Koko using the following interface:

$$[\mathsf{affectiveState}] \times [\mathsf{eventDefinition}] \times [\mathsf{sensorID}] \times \mathsf{modelID} \mapsto \mathsf{applicationID} \tag{6}$$

The *affectiveStates* are the set of states (drawn from the emotion ontology) that the application wishes to model. The *eventDefinitions* describe the structure (created using the event ontology) of all necessary application events. The developer can encode the entire application state using the ontology, but this is often not practical for large applications. Therefore, the developer must select the details about the application's environment that are relevant to the emotions they are attempting to model. For example, the time the user has spent on a current task will most likely effect their emotional status, where as the time until the application needs to garbage collect its data structures is

most likely irrelevant. The *sensorIDs* and *modelID* both have trivial explanations. Koko maintains a listing of both the available sensors and affect models, which are accessible by their unique identifiers. The developer must simply select the appropriate sensors and affect model and record their identifiers.

Further, Koko enables affect models to perform online, supervised learning by classifying events via a set of emotions. Applications can query the user directly for the user's emotional state and then subsequently pass that information to Koko. In general, many applications have well-defined places where they can measure the user's responses in a natural manner, thereby enabling online learning of affective state. Applications that do exercise the learning interface benefit from improved accuracy in the affect model. The formal definition of this interface is as follows. Notice there is no output because this interface is used only to update Koko's data structures not to retrieve information.

$$\mathsf{userID} \times \mathsf{applicationEvent} \times \mathsf{emotionClassifier} \mapsto \bot \tag{7}$$

4 Evaluation

Our evaluation mirrors our claimed contributions, namely, the benefits of the Koko architecture. First, we evaluate the contributions of the architecture in itself. Subsequently, we demonstrate the usefulness of the architecture with case studies of both single and multiplayer games.

4.1 Architecture Evaluation

A software architecture is motivated not by functionality but by so-called "ilities" or nonfunctional properties [3]. The properties of interest here—reusability, extensibility, and maintainability—pertain to gains in developer productivity over the existing monolithic approach. In addition, by separating and encapsulating affect models, Koko enables sharing affective data among applications, thereby enhancing user experience. Thus we consider the following criteria.

Reusability. Koko promotes the reuse of affect models and sensors. By abstracting sensors via a standard interface, Koko shields model designers from the details of how to access various sensors and concentrate instead on the output they produce. Likewise, application developers can use any installed affect model.

Maintainability. Koko facilitates maintenance by separating the application from the affect model and sensors. Koko supports upgrading the models and sensors without changes to the components that use them. For example, if a more accurate sensor has been released you could simply unregister the corresponding old sensor and register the new sensor using the old sensor's identifier. Any model using that sensor would begin to receive more accurate data. Likewise, a new affect model may replace an older model in a manner that is transparent to all applications using the original model.

Extensibility. Koko specifies generic interfaces for applications to interact with affect models and sensors as well as providing a set of implemented sensors and models. New sensors and models can be readily installed as long as they respect the specified interfaces.

User Experience. Koko promotes sharing at two levels: *cross-application* or intraagent and *cross-user* or interagent communication. For a social or multi-player application Koko enables users—or rather their agents—to exchange affective states. Koko also provides a basis for applications—even those authored by different developers—to share information about a common user. An application may query for the mood of its user. Thus, when the mood of a user changes due to an application or otherwise, this change becomes accessible to all applications.

4.2 Case Studies

In this section we present two case studies that demonstrate Koko in operation. The first study is the creation of a new social application, called booST, which illustrates the social and multiplayer aspects of Koko. The second study is the re-creation of the affect-enabled Treasure Hunt game, which helps us illustrate the differences between the Koko and CARE architectures.

booST The subject of our first case study is a social, physical health application with affective capabilities, called booST. To operate, booST requires a mobile phone running Google's Android mobile operating system that is equipped with a GPS sensor. Notice, that while booST does take advantage of the fact that the sensor and application run on the same device this is not a restriction that is imposed by Koko.

The purpose of booST is to promote positive physical behavior in young adults by enhancing a social network with affective capabilities and interactive activities. As such, booST utilizes the OpenSocial platform [13] to provide support for typical social functions such as maintaining a profile, managing a social circle, and sending and receiving messages. Where booST departs from traditional social applications is in its use of the *energy levels* and *emotional status* of its users.

Each user is assigned an *energy level* that is computed using simple heuristics from data retrieved from the GPS. Additionally, each user is assigned an *emotional status* generated from the affect vectors retrieved from Koko. The emotional status is represented as a number ranging from 1 to 10. The higher the number the happier the individual is reported to be. A user's energy level and emotional status are made available to both the user and members of the user's social circle.

To promote positive physical behavior, booST supports interactive physical activities among members of a user's social circle. The activities are classified as either competitive or cooperative. Both types of activities use the GPS to determine the user's progress toward achieving the activities goal. The difference between a competitive activity and a cooperative activity is that in a competitive activity the user to first reach the goal is the winner, whereas in a cooperative activity both parties must reach the goal in order for them to win.

Koko's primary function in booST is to maintain the affect model that is used to generate the user's emotional status. The application provides the data about its environment, which in the case of booST is the user's social interactions and the user's participation in the booST activities. Koko passes this information to the appropriate user agent, who processes the data and returns the appropriate emotional status. Further, Koko enables the exchange of affective state with the members of a user's social

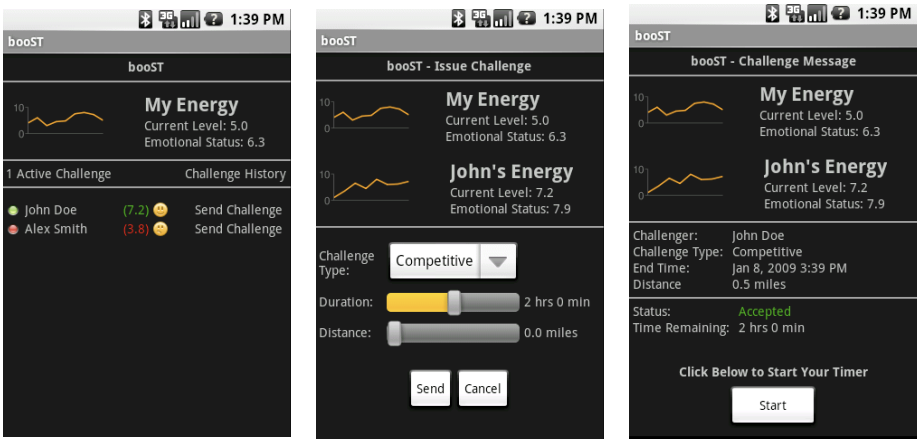

Fig. 5. booST buddy list and activities screenshots

circle. This interaction can be seen in Figure 5 in the emoticons next to the name of a buddy. The affective data shared among members of a social circle is also used to provide additional information to the affect model. For instance, if all the members of a user's social circle are sad then their state will have an effect on the user's emotional status.

Treasure Hunt We showed above how a new application, such as booST, can be built using Koko. Now we show how Koko can be used to retrofit an existing affective application, resulting in a more efficient and flexible application.

Treasure Hunt (TH) is a educational game that demonstrates the CARE model [10] wherein the user controls a character to carry out some pedagogical tasks in a virtual world. TH appraises the emotional state of the user based on a combination of (1) application-specific information such as location in the virtual world, user's objective, and time spent on the task and (2) data from physiological sensors providing information on the user's heart rate and skin conductivity. TH conducts an appraisal every time the user's environment changes and produces one of six perceived emotional states is output.

To reconstruct TH using Koko, we first abstract out the sensors. The corresponding sensor code is eliminated from the TH code-base because the sensors are not needed by the application logic. After the sensors have been abstracted, we select the six emotional states from the emotion ontology that match those already used in TH. The final step is to encode the structure of the application's state information into application events. This basic format of the state information is already defined, as the original TH logs the information for its internal affect model.

Both CARE and Koko models can be thought of as having three phases, as outlined in Table 1. The difference between the two architectures is how they choose to perform those phases. For example, in CARE's version of TH the environmental data and emotional classifiers are written to files. The data is then processed offline and the resulting

affect model is injected into the application allowing TH to produce the emotional probabilities.

Table 1. Three phases of CARE affect models

#	Description
1	Gather environmental data and emotional classifiers
2	Perform supervised ML techniques on the data and classifiers
3	Given environmental data produce emotional probabilities

Koko improves on the CARE architecture by eliminating the need for the application to keep record of its environmental data and also by performing the learning online. As a result, the Koko-based TH can move fluidly from phase 1 to phase 3 and back again. For example, if TH is using Koko then it can smoothly transition from providing Koko with learning data, to querying for emotional probabilities, and return to providing learning data. In CARE this sequence of transitions while possible, results in an application restart to inject the new affect model.

This improvement can be compared to an iterative software engineering approach versus the more rigid waterfall approach. CARE corresponds to the waterfall approach, in that it makes the assumption that you have all the information required to complete a stage of the process at the time you enter that stage. If the data in a previous stage changes, you can return to a previous state but at a high cost. Koko follows a more iterative approach by removing the assumption that all the information will be available ahead of time and by ensuring that transitions to a previous state incur no additional cost. As a result, Koko yields a more efficient architecture than CARE.

5 Discussion

This paper shows how software architecture principles can be used to specify a middleware for social affective computing. If affective computing is to have the practical impact that many hope it will, advances in software architecture are crucial. Further, the vocabulary of events and context attributes introduced here can form the basis of a standard approach for building and hosting affective applications.

Game Integration. Due to the social and multiplayer nature of Koko, it cannot be contained within a traditional gaming engine. However, Koko can interoperate with gaming engines in a loosely coupled manner. To incorporate Koko into an existing game engine API, the engine can simply provide a façade (wrapper) around the Koko API. The façade is responsible for maintaining a connection to the Koko service and marshalling or unmarshalling objects from the engine's data structures to those supported by the Koko. Currently, Koko has service endpoints that support the communication of data structures encoded as Java objects, XML documents, or JSON objects.

Affect Modeling. Existing appraisal theory applications are developed in a monolithic manner [2] that tightly couples application and model. As a notable exception, EMA

provides a domain-independent framework that separates the model from the application. Whereas EMA focuses on modeling virtual characters in a specific application, Koko models human emotion in a manner that can cross application boundaries.

We adopt appraisal theory due to the growing number of applications developed using that theory. Our approach can also be applied to other theories such as Affective Dimensions [15], whose models have inputs and outputs similar to that of an appraisal model. Likewise, we have adopted Elliot's set of emotions because of its pervasiveness throughout the affective research community. Its selection does not signify that Koko is bound to any particular emotion ontology. Therefore, as the field of affective computing progresses and more well-suited ontologies are developed, they too can be incorporated into the architecture.

Virtual Agents. Koko's interagent communication was developed with a focus on human-to-human social interactions (e.g., booST). This does not limit Koko to only those interactions and we have begun to explore the usage of Koko with human-to-virtual agent interactions. Given the correct permissions, virtual agents (operating outside of Koko) could request the user's affective state in the same manner as agents internal to Koko can. For example, the virtual agents models used in the ORIENT [9] and NonKin Village [17] applications could access the affect state of the player and use that information to enhance the agent's decision making process.

Enhanced Social Networking. Human interactions rely upon social intelligence [4]. Social intelligence keys not only on words written or spoken, but also on emotional cues provided by the sender. Koko provides a means to build social applications that can naturally convey such emotional cues, which existing online social network tools mostly disregard. For example, an advanced version of booST could use affective data to create an avatar of the sender and have that avatar exhibit emotions consistent with the sender's affective state.

Future Work. Koko opens up promising areas for future research. In particular, we would like to further study the challenges of sharing affective data between applications and users. In particular we are interested in exploring the types of communication that can occur between affective agents.

In a companion paper [19], we describe a methodology for building affect-aware, social applications. We are interested in refining and enhancing that methodology for gaming applications that incorporate affect.

References

1. J. Dias and A. Paiva. Feeling and reasoning: A computational model for emotional characters. In *Progress in Artificial Intelligence, LNCS*, 3808:127–140. Springer, 2005.
2. C. Elliott. *The Affective Reasoner: A Process Model of Emotions in a Multi-agent System*. PhD thesis, Northwestern, 1992.
3. R. E. Filman, S. Barrett, D. D. Lee, and T. Linden. Inserting ilities by controlling communications. *Communications of the ACM*, 45(1):116–122, 2002.
4. D. Goleman. *Social Intelligence: The New Science of Human Relationships*. Bantam Books, New York, 2006.

5. J. Gratch, W. Mao, and S. Marsella. *Modeling Social Emotions and Social Attributions*. Cambridge University Press, Cambridge, UK, 2006.
6. J. Gratch and S. Marsella. Fight the way you train: The role and limits of emotions in training for combat. *Brown Journal of World Affairs*, Vol X (1):63–76, Summer/Fall 2003.
7. J. Gratch and S. Marsella. A domain-independent framework for modeling emotion. *Journal of Cognitive Systems Research*, 5(4):269–306, 2004.
8. R. S. Lazarus. *Emotion and Adaptation*. Oxford University Press, New York, 1991.
9. M. Y. Lim, J. Dias, R. Aylett and A. Paiva. Intelligent NPCs for Education Role Play Game. In F. Dignum, B. Silverman, J. Bradshaw, and W. van Doesburg, editors, *Agents for Games and Simulations LNAI*, In this volume, Springer, 2009.
10. S. McQuiggan, S. Lee, and J. Lester. Predicting user physiological response for interactive environments: An inductive approach. In *Proceedings of the Second Artificial Intelligence and Interactive Digital Entertainment Conference*, pages 60–65. AAAI Press, 2006.
11. S. McQuiggan and J. Lester. Modeling and evaluating empathy in embodied companion agents. *International Journal of Human-Computer Studies*, 65(4):348–360, 2007.
12. A. Newell. *Unified Theories of Cognition*. Harvard University Press, Harvard, 1990.
13. OpenSocial Foundation. Opensocial APIs, http://www.opensocial.org, 2009.
14. A. Ortony, G. L. Clore, and A. Collins. *The Cognitive Structure of Emotions*. Cambridge University Press, Cambridge, MA, 1988.
15. R. W. Picard. *Affective Computing*. MIT Press, Cambridge, MA, 1997.
16. M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice-Hall, Upper Saddle River, NJ, 1996.
17. B. Silverman, D. Chandrasekaran, N. Weyer, D. Pietrocola, R. Might, and R. Weaver. NonKin Village: A Training Game for Learning Cultural Terrain and Sustainable Counter-Insurgent Operations. In F. Dignum, B. Silverman, J. Bradshaw, and W. van Doesburg, editors, *Agents for Games and Simulations LNAI*, In this volume, Springer, 2009.
18. C. Smith and R. Lazarus. Emotion and adaptation. In L. A. Pervin and O. P. John, editors, *Handbook of Personality: Theory and Research*, pages 609–637. Guilford Press, New York, 1990.
19. D. J. Sollenberger and M. P. Singh. Methodology for engineering affective social applications. In *Proceedings of the Tenth International Workshop on Agent-Oriented Software Engineering*, LNCS. In Press, Springer, 2009.
20. I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, 2005.