Methodology for Engineering Affective Social Applications

Derek J. Sollenberger and Munindar P. Singh

Department of Computer Science
North Carolina State University
Raleigh, NC 27695-8206, USA
{djsollen,singh}@ncsu.edu

Abstract. Affective applications are becoming increasingly mainstream in entertainment and education. Yet, current techniques for building such applications are limited, and the maintenance and use of affect is in essence handcrafted in each application. The Koko architecture describes middleware that reduces the burden of incorporating affect into applications, thereby enabling developers to concentrate on the functional and creative aspects of their applications. Further, Koko includes a methodology for creating affective social applications, called Koko-ASM. Specifically, it incorporates expressive communicative acts, and uses them to guide the design of an affective social application. With respect to agent-oriented software engineering, Koko contributes a methodology that incorporates expressives. The inclusion of expressives, which are largely ignored in conventional approaches, expands the scope of AOSE to affective applications.

1 Introduction

Representing and reasoning about affect is essential for producing believable characters and empathic interactions with users, both of which are necessary for effective agent based entertainment and education applications. Leading applications of interest include pedagogical tools [4, 9], military training simulations [6], and educational games [10].

As evidenced by the above applications, incorporating affect into applications is an active area of research. The primary focus of the existing research has been modeling the affective state of a single agent. Our work builds on that foundation but goes further by incorporating affective computing with multiagent systems (MAS). By focusing our attention on the communication of affect between agents, we can develop a new class of applications that are both social and affective. To achieve the fusion of affect with MAS, two challenges must be overcome. First, we need a medium through which agents can exchange affective data. Second, we must define a methodology for creating affective social applications via that medium.

The first challenge is addressed by using Koko [14], a middleware that facilitates the sharing of affective data. Koko is a multiagent middleware whose agents manage the affective state of a user. Further, Koko is intended to be used by applications that seek to recognize emotion in human users. Although it is possible to use Koko in systems

that model emotion in virtual characters, many of its benefits most naturally apply when human users are involved.

Importantly, Koko enables the development of affective social applications by introducing the notion of expressive communicative acts into agent-oriented software engineering (AOSE). Using the multiagent environment provided by Koko, agents are able to communicate affective information through the exchange of expressive messages. The communication of affective information is naturally represented as an *expressive* communicative act, as defined by Searle [12]. However, expressive acts are a novelty in both AOSE and virtual agent systems, which have traditionally focused on the assertive, directive, and commissive communicative acts (e.g., the FIPA inform command). Further, Koko enables us to create a methodology for engineering affective, social applications.

Contributions. This paper describes a methodology, Koko-ASM, centered on expressive communicative acts, the first such methodology to our knowledge. Using this methodology application developers can construct applications that are both social and affective. As such, the combination of the Koko middleware and this methodology enable AOSE to expand into the design and creation of affective social applications.

Paper Organization. The remainder of this paper is arranged as follows. Section 2 reviews appraisal theory affect models. Section 3 provides an overview of the Koko middleware. Section 4 describes the Koko-ASM methodology. Section 5 demonstrates its merits via a case study.

2 Background

This section provides a synopsis of two areas that are fundamental to Koko-ASM: (1) appraisal theory as a foundation for modeling affect and (2) communicative acts and their relevance to AOSE.

2.1 Appraisal Theory

Smith and Lazarus' [13] cognitive-motivational-emotive model, the baseline for current appraisal models (see Fig. 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].

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 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 simultaneously happy (at an intensity of α) as well as proud (at an intensity of β).

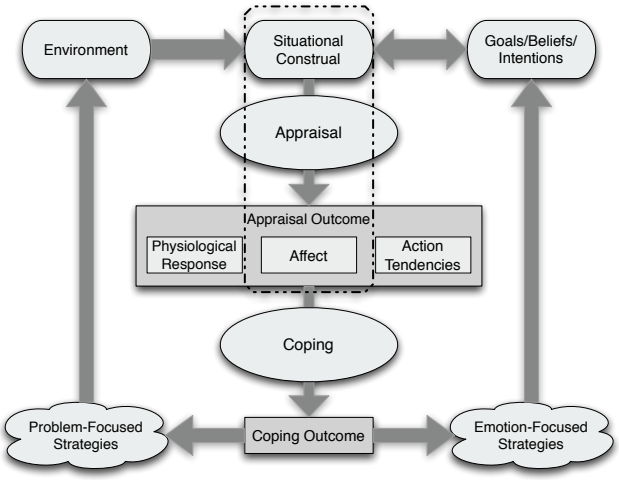

Fig. 1. Appraisal theory diagram [13]

2.2 Communicative Acts

The philosophers Austin [1] and Searle [12] developed speech act theory founded on the principle that communication is a form of action. In other words, when an agent communicates, it alters the state of the world. Communicative acts are grouped based on their effects on the agent's internal state or social relationships. Specifically, assertive acts are intended to inform, directive acts are used to make requests, and expressive acts allow agents to convey emotion.

Existing agent communication languages and methodologies disregard expressives. AOSE methodologies specify messages at a high level and therefore are not granular enough extract the meaning of the messages [2]. On the other hand, agent communication languages specify messages at the appropriate level of detail, but omit expressives. Instead, they have focused on other communicative acts, such as assertives and directives, which can be can be readily incorporated into traditional agent BDI frameworks [15].

3 Koko

Koko's purpose is twofold. It serves as both an affect model container and an agent communication middleware [14]. The affect models that Koko maintains focus on a section of the appraisal theory process (denoted by the dashed box in Fig. 1) in which the models absorb information about the agent's environment and produce an approximation of the agent's affective state. Koko then enables that affective information to be shared among agents via expressive messages. It is important to note that since Koko is designed to model human users, the environment of the agent extends into the physical world. To better report on that environment, Koko supports input from a variety of physical sensors (e.g., GPS devices).

Koko promotes the sharing of affective information at two levels: *cross-user* or interagent communication and *cross-application* or intraagent. For a social (multi-agent)

application, Koko enables agents to communicate via expressives. Expressives enable agents to share their current affective state among other agents within Koko (the format of an expressive message is outlined in Section 4). Koko also provides a basis for applications—even those authored by different developers—to share information about a common user. This is simply not possible with current techniques because each application is independent and thereby unaware of other applications being employed by a user.

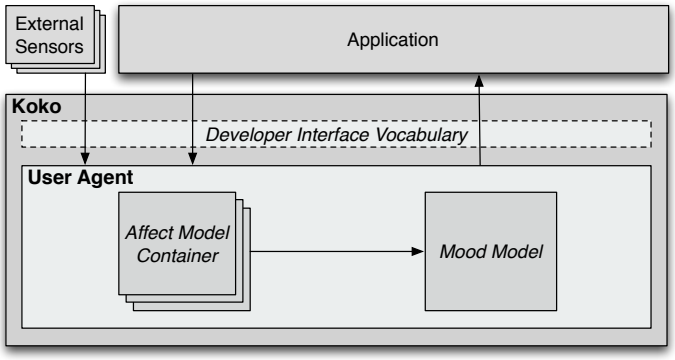

Fig. 2. Koko basic architectural overview

Fig. 2 shows Koko's basic architecture using arrows to represent data flow. The following sections summarize a few of Koko's key components.

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

Affect Model Container. This container manages one or more affect models for each user agent. Each application must specify exactly one affect model, which is instantiated for each user of the application. The container then manages that instance for the user agent. As Fig. 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 in Section 4) at runtime.

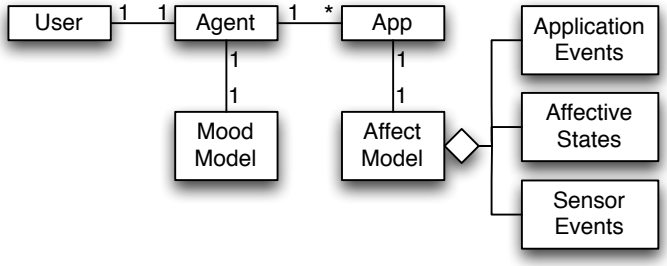

Fig. 3. Main Koko entities

Koko follows CARE's [10] supervised machine learning approach for modeling affect by populating predictive data structures with affective knowledge. This enables Koko to support affect models that depend on an application's domain-specific details, while allowing Koko as a whole to maintain a domain-independent architecture.

For each affect model, the container 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 the emotion 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 [7], 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.

For simplicity, Koko's model for mood 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, the user's mood is a cross-application measurement of the user's emotional state.

Developer Interface Vocabulary Koko provides a vocabulary through which the application 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). If needed, the ontologies are designed to grow to meet the needs of new 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 captures the 24 emotional states proposed by Elliot [3], including 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 at a given moment. When the event arrives at the affect model, it is decomposed using the agreed upon event definition.

Koko comes preloaded with an event ontology (partially shown in Fig. 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

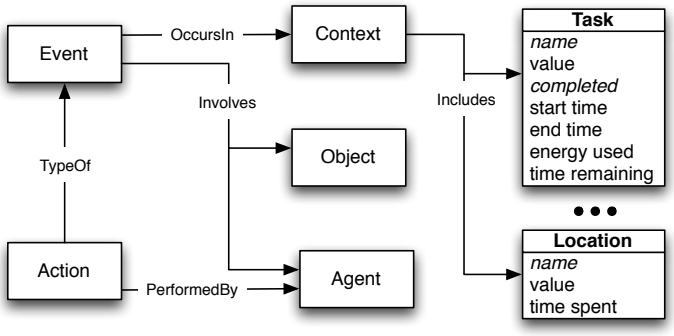

Fig. 4. Event ontology example

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).

Runtime API. The runtime API is the interface through which applications communicate with Koko at runtime. The API can be 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 Section 4 discusses.

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 specified in the developer interface. Using the ontology, the developer encodes snapshots of the application's environment. At runtime the snapshots capturing the user's view of the application environment are passed into Koko for processing. Upon receipt, Koko stores each event 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 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 depend upon the set of emotions chosen by the application when it is configured, whereas the mood vector's entries aggregate all emotions modeled for a particular user. As such, a user's mood is relevant across all applications. Suppose a user, Alice, reads an email that makes her angry and the email client's affect model recognizes this. All of Alice's affect enabled applications can benefit from the knowledge that the user is angry even if they cannot infer that it is from some email. Such mood sharing is natural via Koko because Koko maintains the user's mood and can supply it to any application.

4 Methodology

Now that we have laid the architectural foundation we describe Koko-ASM, a methodology for configuring a social (multiagent) affective application using Koko. 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 that are required by Koko. Table 1 systematically lists Koko-ASM's steps to create an affective social application. The following documentation concentrates on Steps 1-4, which are of primary interest to AOSE.

Table 1. Koko-ASM: a methodology for creating an affective, social application

Step	Description	Artifacts Produced
1	Define the set of possible roles an agent may assume	Agent Roles
2	Describe the expressives exchanged between roles	Expressive Messages
3	Derive the emotions to be modeled from the expressives	Emotions
4	Describe the set of possible application events	Application Events
5	Select the sensors to be included in the model	Sensor Identifier(s)
6	Select the desired affect model	Model Identifier

Step 1 requires the developer to identify the set of roles an agent may assume in the desired application. Possible roles include TEACHER, STUDENT, PARENT, CHILD, and COWORKER. A single agent can assume multiple roles and a role can be restricted to apply to the agent only if certain criteria are met. For example, the role of COWORKER may only apply if the two agents communicating work for the same company.

Step 2 requires the developer to describe the expressive messages or *expressives* exchanged between various roles [12]. Searle defines expressives as communicative acts that enable a speaker to express his or her attitudes and emotions towards a proposition. Examples include statements like "Congratulations on winning the prize!" where the attitude and emotion is congratulatory and the proposition is winning the prize. Formally, we define the structure of an expressive to match that of a communicative act in general:

$$\langle sender, receiver, type, proposition \rangle \tag{1}$$

The *type* of the expressive refers to the attitude and emotion of the expressive and the *proposition* to its content, including the relevant events. The *sender* and *receiver* are selected from the set of roles defined in Step 1. The developer then formulates the expressives that can be exchanged among agents assuming those roles. The result is the set of all valid expressive messages allowed by the application.

Step 3 requires the developer to select a set of emotions to be modeled from the emotion ontology. The selected emotions are based on the expressives identified in the previous step. To compute the set of emotions, we evaluate each expressive and select the most relevant emotions from the ontology for that particular expressive. We add the selected emotions to the set of emotions required by the application. This process is repeated for every expressive and the resulting emotion set is the output of this step.

Koko offers support for expressives by providing a well-delineated representation for affect. Koko can thus exploit a natural match between expressives and affect to help designers operationalize the expressives they employ in their applications. Our recommended approach to selecting an emotion is to structure Elliot's set of emotions as a tree (Fig. 5). Each leaf of the tree represents two emotions, one that carries a positive connotation and the other a negative connotation. Given an expressive, you start at the top of the tree and using its *type* and *proposition* you filter down through the appropriate branches until you are left with only the applicable emotions. For example, say that you have a message with a *type* of *excited* and a *proposition* equal to "I won the game." Now using the tree you determine that winning the game is an action the user would have taken and that *excited* has a positive connotation, so the applicable emotion must therefore be *pride*. In general, the sender and receiver would have different interpretations. For example, if the recipient of the above message is the agent who lost the game, then the emotions that are relevant to the recipient would be *admiration* and *reproach* depending on their perception of the winner.

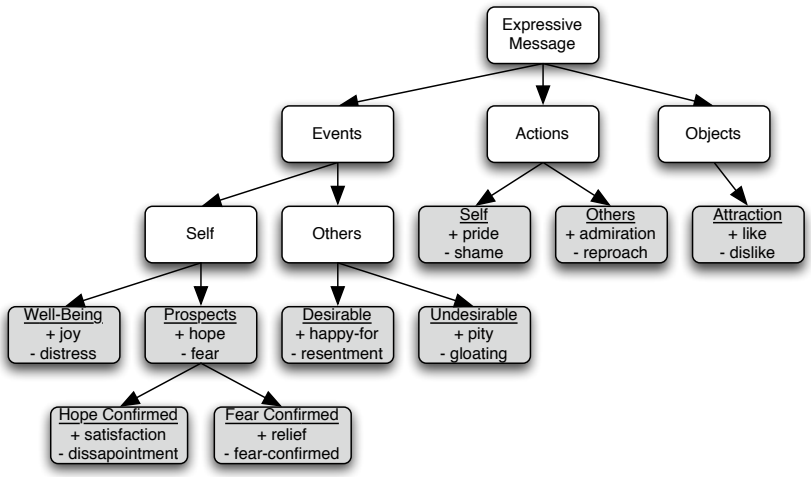

Fig. 5. Expressive message hierarchy

If the *proposition* of the expressive message is composite or even ambiguous as to whether or not the *type* applies to an event, action, or object, then more than one path of the tree may apply. Such is the case when an agent conveys its user's mood via an expressive message. Mood is an aggregation of emotions and therefore does not have a unique causal attribution. For example, an expressive might convey that a user is generally happy or sad without being happy or sad at something. Therefore, we do not select any specific emotion when evaluating a expressive pertaining to mood as the emotions that comprise the mood are captured when evaluating the other expressives. In other words, mood is not treated directly upon the reception of an expressive.

Step 4 requires the developer to describe the application events using the event ontology. The events described are a combination of the expressives in Step 2 and additional details about the application environment. An expressive is modeled as two

application events, one for sending and another for receipt. Each event is modeled as an action (see Fig. 4) in which the sending of a message is described as an action performed by the sender that involves the receiver with the expressive in the context. Similarly, one can envision the receipt of the message as an action performed by the recipient that involves the sender. The decomposition of a message into two events is essential because we cannot make the assumption that the receiving agent will read the message immediately following its receipt and we must accommodate for its autonomy.

The additional details about the application's environment are also modeled using the event ontology. The developer can encode the entire application state using the ontology, but this may not be 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, whereas the time until the application needs to garbage collect its data structures is likely irrelevant. The resulting events are combined with the events derived from the expressive messages to form the set of application events that are needed by Koko.

Step 5 and *Step 6* 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 sensor and affect model identifiers.

Based on the artifacts generated by the above methodology we now have sufficient information to configure the Koko middleware. Upon configuration Koko supports affective interactions among agents (using the expressive messages) as well as enables applications to query for the affective state of an agent.

5 Evaluation

Using expressives to communicate an agent's affective state extends traditional AOSE into the world of affective applications. We evaluate Koko-ASM by conducting a case study that steps through the methodology and produces a functional affective social application. The subject of our 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.

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 Google OpenSocial platform [11] 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 communication and display of its user's *energy levels* and *emotional status*.

Each user is assigned an *energy level* that is computed using simple heuristics from data retrieved from the GPS sensor on board the phone. Additionally, each user is assigned an *emotional status* generated from the affect vectors retrieved from Koko. The emotional status is represented as a real number ranging from 1 (sad) to 10 (happy). 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 the members of a user's social circle. The activities are classified as either

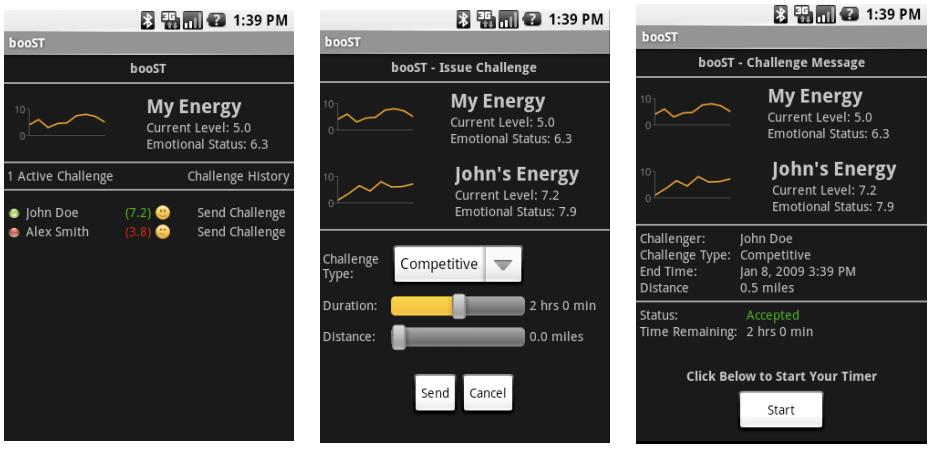

Fig. 6. booST buddy list and activities screenshots

competitive or cooperative. Both types of activities use heuristics based on the GPS sensor readings 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 first user to reach the goal is the winner, whereas in a cooperative activity both parties must reach the goal in order for them to win.

As described in Section 3, Koko hosts an agent for each booST user. A user's agent maintains the affect model that is used to generate the user's emotional status. Further, booST provides the agent with data about its environment, which in this case incorporates the user's social interactions and his or her participation in the booST activities. The user agent processes the data and returns the appropriate emotional status. Further, Koko enables the exchange of affective state between booST agents (representing a social circle of users). This interaction can been seen in Figure 6 in the emoticons next to the name of a buddy. The affective data shared among the members of a social circle provides 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.

5.1 Configuring booST

Table 1 outlines Koko-ASM's process for creating an affective, social application. We now demonstrate this process using booST.

Step 1 requires that we identify the set of roles an agent may assume. The booST application involves only two roles, FRIEND and SELF. An agent A assumes the role of agent B's FRIEND if and only if the users represented by agents A and B are members of each others social circle. The social circle is maintained by booST and can be equated to the friend list in popular social applications such as Facebook and MySpace.

Step 2 requires that we identify and describe all expressives that occur between the two roles. Below is an example of what a few such messages would look like depending

on the outcome of a competitive activity within booST. The remaining messages would be defined in a similar fashion.

$$\langle \text{SELF}, \text{FRIEND}, \textit{happy}, \textit{"I won the game"} \rangle \quad (2)$$

$$\langle \text{FRIEND}, \text{SELF}, \textit{sad}, \textit{"I lost the game"} \rangle \quad (3)$$

$$\langle \text{FRIEND}, \text{SELF}, \textit{happy}, \textit{"I ran a good race"} \rangle \quad (4)$$

Step 3 requires that we select a set of emotions to model from the emotion ontology. As Section 3 shows, the ontology is based on Elliot's expansion of the OCC model, which categorizes emotions based on the user's reaction to an action, event, or object. When inspecting each expressive, we find that booST focuses on measuring *happiness* and *sadness* of the user with respect to actions and events. Therefore, we can narrow our selection to only emotions that meet those criteria. As a result, we select four emotions: two are focused on the actions of the user (*pride* and *shame*) and two on the events (*joy* and *distress*). The booST application uses these emotions to compute the user's emotional status by correlating (1) *pride* and *joy* with happiness and (2) *shame* and *distress* with sadness.

Step 4 requires that we describe the application events using the event ontology. Each expressive message yields two events: a sending event and a receiving event. The remaining events provide additional details about the application's environment (Table 2 shows some examples). Since an event in booST is merely an instantiation of the event ontology, the event descriptions are trivial. For example, the "Competitive Exercise Challenge" message can be described as an action that involves another agent. When an event occurs at runtime, the context associated with its occurrence would specify attributes such as the time of day, challenge information, and the user's energy level as calculated by the application.

Table 2. Representative booST events

#	Event Description
1	View my energy level and emotional state
2	View my friend's energy level and emotional state
3	Send or receive "Cooperative Exercise Challenge" message
4	Send or receive "Competitive Exercise Challenge" message
5	Complete or fail a cooperative activity
6	Complete or fail a competitive activity

Step 5 requires that we select the sensors to be included in the model. As we have already noted booST requires a single sensor: a GPS. The first time a sensor is used a plugin must be created and registered with Koko. The GPS plugin converts latitude and longitude vectors into distance covered over a specified time period. This data is then maintained by Koko and made available for consumption both by the application and the affect model.

Step 6 is trivial as booST employs an affect model that is provided by Koko. In particular, booST employs the model that implements decision trees as its underlying data structure.

Using the artifacts generated by the methodology we now have obtained sufficient information to configure the Koko middleware. Upon configuration, Koko maintains an agent for each booST user. The agent is responsible for modeling the affective state of the user as well as communicating that state to other agents within the user's social circle. With Koko supporting the affective aspects of booST, application developers are free to focus on other aspects of the application. For instance, they may focus on developing the creative aspects of the game, such as how to alter gameplay based on the user's affective state.

6 Discussion

An important contribution of Koko and Koko-ASM is the incorporation of expressive communicative acts. These acts, though well-known in the philosophy of language, are a novelty both in agent-oriented software engineering and in virtual agent systems. The incorporation of expressives enable agents to interpret the difference between an expression of feelings and a statement of fact, thus enabling agents to better model their users and their environment.

Existing Methodologies. Existing AOSE methodologies specify messages at a level that does not describe the contents of the message and therefore are not granular enough to support expressives [2]. Koko-ASM is restricted to applications that are both affective and social, thus its applicability has a much narrower scope than existing methodologies. These distinctions are simply the result of a difference in focus. It is quite possible, given the narrow scope of Koko-ASM, that it could be integrated with broader methodologies in order to leverage their existing processes and tools. For example, many methodologies [2, 16] have detailed processes by which they help developers identify all possible messages that are exchanged among agents. Koko-ASM would benefit by integrating those processes, thereby making it easier to identify the expressive messages.

Virtual Agents. Koko and, in particular, Koko-ASM have focused on human-to-human social interactions. This does not inherently limit the methodology only to such interactions. We have begun to explore the application of our methodology on human-to-virtual agent interactions. Using this modified version of Koko-ASM we envision a scenario where the virtual agents will have access to the user's affective state via Koko. Applications that leverage this technique could manipulate a virtual agent's interactions with a user, based on the user's affective state.

Enhanced Social Networking. Human interactions rely upon social intelligence [5]. 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 networking 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 and Koko-ASM open up promising areas for future research. As an architecture, it is important that Koko fits in with existing architectures such as game engines. We have described some efforts in a companion paper [14]. Association with other architectures would not only facilitate additional applications but would lead to refinements of the present methodology, which we defer to future research. Further, we are actively working on formalizing the notion of expressive communication with respect to agent communication languages.

References

1. J. L. Austin. *How to Do Things with Words*. Oxford University Press, London, 1962.
2. S. A. Deloach, M. F. Wood, and C. H. Sparkman. Multiagent Systems Engineering. *Journal of Software Engineering and Knowledge Engineering*, 11(3):231–258, 2001.
3. C. Elliott. *The Affective Reasoner: A Process Model of Emotions in a Multi-agent System*. PhD, Northwestern University, 1992.
4. C. Elliott, J. Rickel, and J. Lester. Lifelike pedagogical agents and affective computing: An exploratory synthesis. In *Artificial Intelligence Today, LNAI*, 1600:195–212, 1999.
5. D. Goleman. *Social Intelligence: The New Science of Human Relationships*. Bantam Books, New York, 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, Dec 2004.
8. R. S. Lazarus. *Emotion and Adaptation*. Oxford University Press, New York, 1991.
9. S. Marsella, W. L. Johnson, and C. LaBore. Interactive pedagogical drama. In *International Conference on Autonomous Agents*, pages 301–308, 2000.
10. S. McQuiggan and J. Lester. Modeling and evaluating empathy in embodied companion agents. *International Journal of Human-Computer Studies*, 65(4), April 2007.
11. OpenSocial Foundation. Opensocial APIs, http://www.opensocial.org, 2009.
12. J. R. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, Cambridge, UK, 1970.
13. 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. New York, Guilford Press, 1990.
14. D. J. Sollenberger and M. P. Singh. Architecture for Affective Social Games. In *Proceedings of First International Workshop on Agents for Games and Simulations*, LNAI. In Press. Berlin, Springer, 2009.
15. R. Vieira, A. Moreira, M. Wooldridge, and R. H. Bordini. On the formal semantics of speech-act based communication in an agent-oriented programming language. *Journal of Artificial Intelligence Research*, 29:221–267, 2007.
16. F. Zambonelli, N. R. Jennings, and M. Wooldridge. Developing Multiagent Systems: The Gaia Methodology. *ACM Transactions on Software Engineering and Methodology*, 12(3):317–370, July 2003.