Social Perception

Modeling human interaction for the next generation of communication services

JAMES L. CROWLEY
INRIA RHÔNE-ALPES

A SCENARIO: BOB RUNS A MEETING

Bob manages a team that designs and builds widgets. Life would be sweet, except that Bob's team is distributed over three sites, located in three different time zones. Bob used to collect lots of frequent flyer miles traveling to attend meetings. Lately, however, business travel has evolved into a humanly degrading, wasteful ordeal. So Bob has invested in a high-bandwidth video communications system to cut down on business travel. Counting direct costs, the system was supposed to pay for itself within three months. There is a problem, however.

Bob's videoconferencing system brings together the views from up to 16 cameras to form a four-by-four mosaic of images that can be displayed by up to 16 video channels. The cameras are static, so meeting participants have to position themselves in the center of an image. Naturally, people are not careful, so often just a part of a face appears on the edge of an image. One of the cameras at each site is aimed at a whiteboard, but it is never clear which whiteboard is where. Other cameras present overhead views of individual sketchpads on which participants can draw, but it is never clear who is doing the drawing, or whether they are illustrating a point or simply making notes for themselves. The audio is a composition of up to 16 microphones, all transmitting at the same time. The first half of most meetings is spent "fooling around with the system." Conversations are frequently interrupted by statements such as "Hold on while I adjust the camera to show you." When you factor in the loss in productivity, the system is not likely to pay for itself this year.

Social Perception

INFORMATION TECHNOLOGY IS AUTISTIC

Continued exponential decline in the cost of both communications and information technology would seem to enable a large and diverse array of services for enhancing human-to-human interaction. Examples of such services include:

- Automated camera control for videoconferencing
- Communication tools for collaborative work
- Automated meeting recording
- Tools for automated recording of team sports
- Tools for managing communications to protect privacy and avoid disruption
- Tools for organizing and conducting meetings

This is an open-ended set, limited only by our ability to imagine. Unfortunately, despite the presence of enabling communications technology, none of these services has entered a virtuous spiral of exponential growth.

The use of information technology to enhance human-to-human interaction is currently impractical because of the problem of disruption. Information and communication technologies are autistic. They have no sense of the social roles played by interacting humans, no abilities to predict appropriate or inappropriate service actions, and no sensitivity to the disruption to activity caused by inappropriate service behavior. Disruption renders information and communications services impractical for many applications.

WHAT BOB WANTS: SITUATION-AWARE MEETING TOOLS

When he bought his system, Bob imagined that he was buying a tool that would know when to switch among the cameras and microphones to present the video image and sound of the current speaker. He expected that the image would be perfectly centered on the face of each speaker, and could be switched effortlessly between a view of the whiteboard, a view of a drawing pad, and the view of a group when several people were speaking at once. He even imagined that he could use the video record as minutes of the meeting.

Unfortunately, such automatic recording of audio and video requires an understanding of the roles played by participants in a meeting. For example, the meeting moderator assures that the meeting follows the agenda and stays focused on the meeting's objectives. When the moderator interrupts a speaker, it is important that participants turn their attention to the moderator, even if others are speaking at the same time. The moderator gives the floor to participants so that they may express information to the others. The person in possession of the "floor" is the speaker who should be presented to the others. When he or she speaks directly to the meeting, other participants should see his or her face. When the speaker turns his or her attention to the whiteboard or a sketchpad, participants should see the sketchpad. When the speaker is interrupted by a question, then participants should see both the speaker's view and that of the questioner. When two listeners who do not have the floor exchange small comments, the system should not disrupt the meeting by transmitting their aside to all the other participants. When properly functioning, the system should automatically compose a sequence of video shots that can serve as an audiovisual "record" of the meeting. Off-line speech recognition could even be used to label these shots by topic so that they can be individually recalled without scrolling through the entire meeting record. All of this should happen without any thought or intervention by the meeting participants.

To give Bob his dream system and to save his company money, Bob's videoconferencing system needs to understand what is going on during a meeting. Such understanding is called *situation awareness*. This article proposes a conceptual framework and a software model for situation-aware observation of human activity. The core component of our framework developed at INRIA (French National Institute for Research in Computer Science and Control) Rhône-Alpes is a situation model, which acts as a nonlinear script for interpreting the current actions of humans and predicting the corresponding appropriate and inappropriate actions for services. This framework organizes the observation of interaction using a hierarchy of concepts: scenario, situation, role, action, and entity.

The following section outlines the conceptual framework for this theory. This is followed by a proposal for a layered architecture for nondisruptive services. Within this layer we present a component-based architectural model that uses concepts from autonomic computing to provide observation of human activity that robustly adapts to changes in the environment.

SITUATED OBSERVATION OF HUMAN ACTIVITY

Most human societies have developed and refined an art form for describing human action and social interaction:

the theater. Theater can be used as a rich source of concepts for socially aware observation of human activity.

A theatrical production provides a model for social interaction in the form of a script. The production organizes the actions of a set of actors in terms of roles, structured as a series of scenes composed, in turn, of a series of situations. A role is more than a set of lines. A role defines a space of allowed actions, including dialog, movement, and emotional expressions. The audience understands the production by recognizing the roles using social stereotypes and relating these to individual social experiences.

In a similar manner, everyday human actions and interactions can be observed and described in terms of situations in which individuals play roles. Depending on the activity, actions and interactions may be more or less constrained and limited by implicit compliance with a shared script. Deviating from the script is considered impolite and can often provoke conflict or even terminate the interaction. Some activities, such as classroom teaching, formal meetings, shopping, or dining at a restaurant, follow highly structured scripts that constrain individual actions to highly predictable sequences. Other human activities occur in the absence of well-defined scripts and are thus less predictable. We propose that when a stereotypical social script does exist, it can be used to structure observation and guide the behavior of services to avoid disruption.

One important difference exists between theater and life. A theater script is composed of a fixed sequence of situations. Real life is much less constrained. For many activities, situations form a network rather than a sequence and may often exhibit loops and nondeterministic branching. The complexity and difficulty of observing human activity are related to the degree of interconnectivity of situations.

CONCEPTUAL FRAMEWORK FOR OBSERVING ACTIVITY

Translating theatrical concepts into software requires formal expression. To be meaningful, this formal expression must ultimately be grounded in procedures and actions for real systems. In this section we propose a hierarchy of definitions for concepts for observing human activity, sometimes called a *context model*.[1] *Context* is a highly overloaded term, however, meaning different things to different people. To avoid confusion, this article refers to models for observing activity as *scenarios*.

In common use, *situation* derives its meaning from the way in which something is placed in relation to its surroundings—for example, in terms of position and action. In our case, the definition of *situation* requires two aspects: perception and action. *Perception* refers to the ability to sense and recognize situation; *action* refers to the way in which the system reacts to the current situation. Bob's ideal videoconferencing system would interpret the signals from cameras and microphones to determine who is playing the role of moderator and who is currently speaking or asking a question, then would use this information to switch between possible views of participants.

A situation is a form of state. Situation represents the state of the activity as defined by relations between the actors playing roles. Changes in the situation trigger actions by the system, such as changing the current camera or microphone or enabling face-tracking to keep a camera centered on a meeting participant. When a

Glossary

Context: The situation within which something exists or happens and that can help explain it;[1] any information that can be used to characterize situation.

Scenario: A description of possible actions or events in the future; a written plan for the characters and events in a play or movie;[1] a network of situations for modeling human activity expressed in terms of relations between entities playing roles.[2]

Situation: The set of things that are happening and the conditions that exist at a particular time and place;[1] a predicate expression of a set of relations over entities assigned to roles.[3]

Relation: A predicate test on properties of one or more of the entities playing roles.

Role: A function that selects an entity from the set of observed entities.

Actor: A role for entities that can spontaneously act to change the current situation.

Prop: A role for entities that cannot spontaneously act to change the current situation.

1. Cambridge online dictionary of the English Language; http://dictionary.cambridge.org.
2. Coutaz, J., Crowley, J. L., Dobson, S., Garlan, D. 2005. Context is key. *Communications of the ACM* (Special issue on the Disappearing Computer) 48(3): 49-53.
3. Brdiczka, O., Maisonnasse, J., Reignier, P. 2005. Automatic detection of interaction groups. International Conference on Multimodal Interaction (ICMI '05), Trento, Italy (October).

Social Perception

speaker in Bob's system points out information on a projected slide, the system should switch from presenting the speaker to presenting the slide.

Relations are truth functions (predicates) with one or more arguments. The truth of a relation depends on properties that may be observed by a machine perception system. Unary relations apply a test to some property or set of properties of an individual entity. Binary and higher-order relations test relative values of properties of more than one entity. Examples would include spatial and temporal relations (in front of, beside, higher than, etc.), or other perceived properties (lighter, greener, bigger, etc.). Relations test the properties of entities that have been assigned to roles.

A *role* is an abstract generalization for a class of entities. Role classes are typically defined based on the set of actions that entities in the class can take (actors) or can enable (props). A role is *not* an intrinsic property of an entity, but rather an interpretation the system assigns to an entity. So how can the role assignment process select among the available entities? Some have proposed to view this process as a filter.[2] In this view, a filter acts as a kind of sorting function for the suitability of entities based on their properties. The most suitable entity wins the role assignment.

The lowest-level concepts in this framework are entity and property. A property refers to any value that can be observed, or inferred from observations. An entity is a correlated collection of properties. This solipsistic viewpoint admits that the system can see only what it knows how to see. At the same time, it sidesteps existential dilemmas related to how to define notions of object and class. In this view, a chair is anything that can be used as a chair, regardless of its apparent form. More formal definitions for these two concepts are rooted in the software architectural model described later. Operational definitions for property and entity are grounded in the software components for observation of activity.

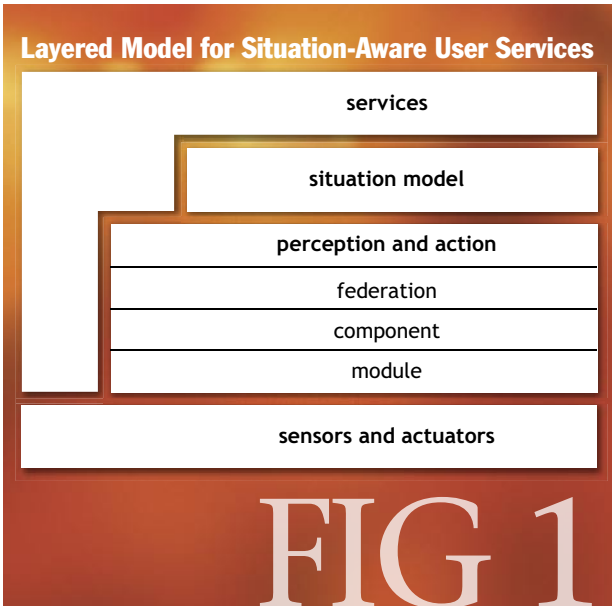
A SOFTWARE ARCHITECTURE FOR OBSERVING ACTIVITY

We have constructed several examples of situation-aware systems that provide information and communication services for human-to-human interaction. Our systems use a layered architectural model, as shown in figure 1. At the lowest layer, the service's view of the world is provided by a collection of physical sensors and actuators. This corresponds to the *sensor-actuator layer*. This layer depends on the technology and encapsulates the diversity of sensors and actuators by which the system interacts with the world. Information at this layer is expressed in terms of sensor signals and device commands.

Hard-wiring the interconnection between sensor signals and actuators is possible and can provide simplistic services that are hardware-dependent and have limited utility. Separating services from their underlying hardware requires that the sensor-actuator layer provide logical interfaces, or standard APIs, that are function-centered and device-independent. Hardware independence and generality require abstractions for perception and action.

Perception and action operate at a higher level of abstraction than sensors and actuators. While sensors and actuators operate on device-specific signals, perception and action operate in terms of environmental state. Perception interprets sensor signals by recognizing and observing entities. Abstract tasks are expressed in terms of a desired result rather than actions to be blindly executed.

For most human activities, there are a potentially infinite number of entities that could be observed and an infinite number of possible relations for any set of entities. The appropriate entities and relations must be determined with respect to the service to be provided. This is the role of the situation model, as described in the previous section. The situation model allows the system to focus perceptual attention and computing resources in



order to associate the current state of the activity with the appropriate system action.

Services specify a scenario composed of a situation model, as just described. The scenario determines the appropriate entities, roles, and relations to observe, acting in a top-down manner to launch (or recruit) and to configure a set of components in the perception-action layer. Once configured, the situation model acts as a bottom-up filter for events and data from perceptual components to the service.

THE PERCEPTION-ACTION LAYER

At the perception-action layer, we propose a data-flow process architecture for software components for perception and action.[3,4,5] Component-based architectures consist of auto-descriptive functional components joined by connectors.[6] Such architecture is well adapted to interoperability of components and thus provides a framework by which multiple partners can explore design of specific components without rebuilding the entire system.

Within the perception-action layer, we propose three distinct sub-layers, as shown in figure 1: modules, components, and federations. The components within each layer are defined in terms of the components in the layer below. Each layer provides the appropriate set of communications protocol and configuration primitives. The following describes the components within each layer.

MODULES

Modules are auto-descriptive components formally defined as synchronous transformations applied to a certain class of data or event, as illustrated in figure 2. Modules generally have no state. They are executed by a call to a method (or function or subroutine), accompanied by a vector of parameters that specifies the data to be processed and describes how the transform is to be applied. Output is also generally accomplished by writing to a stream or by posting events to other modules or an event dispatcher.

Modules return a result that includes a report of the results of processing. Examples of information contained in this report include elapsed execution time, confidence in the result, and any exceptions that were encountered.

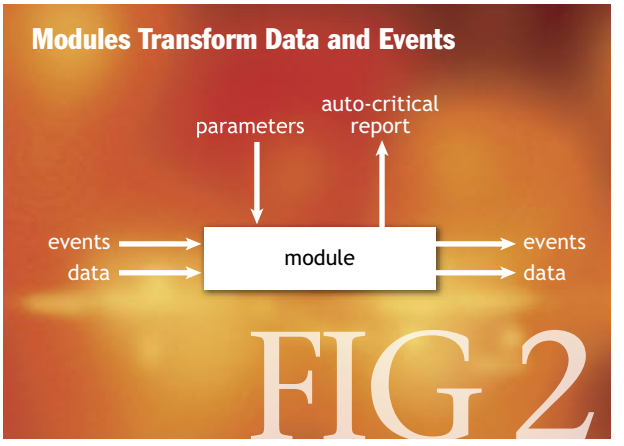
An example of a module is a procedure that transforms RGB color pixels into a scalar value at each pixel that represents the probability that the pixel belongs to a target region. Such a transformation may be defined using a lookup table representing a ratio of color histograms.[7] A common use for such a module is to detect skin-colored pixels within regions of an image, as shown in figure 3.

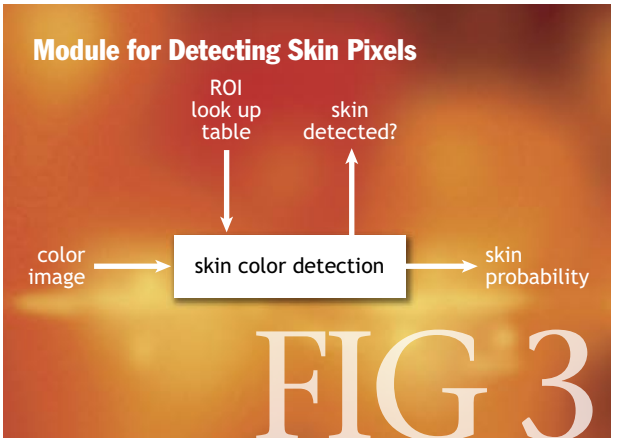
Such a module can be used to find faces in Bob's system in order to steer the camera to keep each face centered in the image.

SOFTWARE COMPONENTS FOR PERCEPTION AND ACTION

The second layer in the architecture concerns perception and action components, autonomous assemblies of modules executed in a cyclic manner by a component supervisor. Components communicate via synchronous data streams and asynchronous events to provide software services for action or perception.

As shown in figure 4, the component supervisor interprets commands and parameters, supervises the execution of the transformation, and responds to queries with a description of the current state and capabilities of the component. The auto-critical report from modules allows a component supervisor to monitor the execution time and to adapt the schedule of modules for the next cycle so as to maintain a specified quality of service, such as execution time or number of targets tracked. Such monitoring can be used, for example, to reduce the resolution





Social Perception

of processing by selecting one pixel of N[8] or to selectively delete targets judged to be uninteresting.

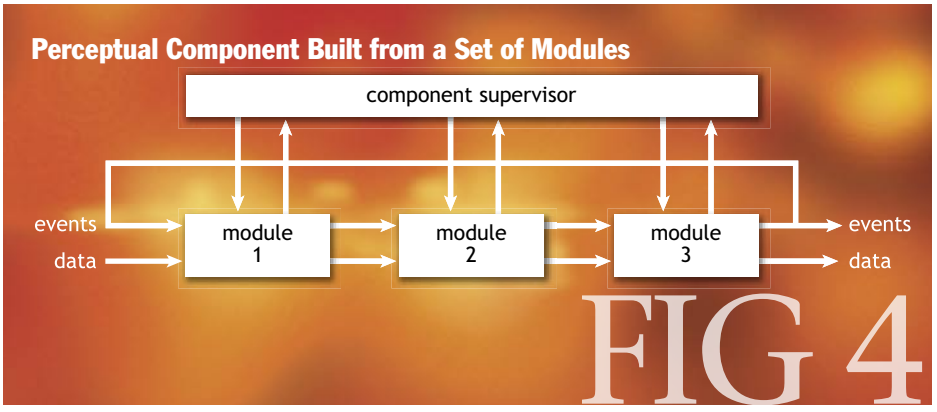
A simple example of a perceptual component is shown in figure 5. This component takes in color images and produces the current position of a skin blob. The supervisory controller, labeled "skin blob tracker," invokes and coordinates observational processes for skin detection, pixel moment grouping, and tracking. This federation provides the transformation component for a composite observation process. The skin region tracker provides the supervisory control for this federation.

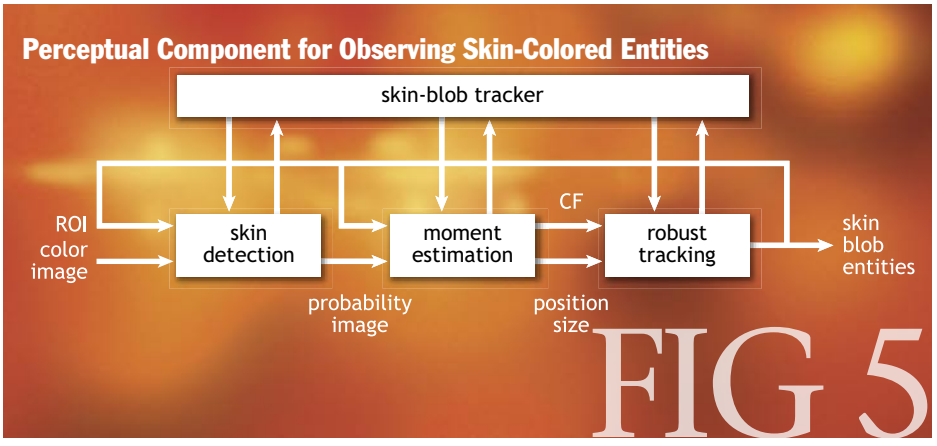
A MODEL FOR PERCEPTUAL COMPONENTS.
A general architectural model (programming pattern) for robust perceptual components is shown in figure 6. Components constructed with this model implement a recursive estimation process to track entities. A well-known framework for such estimation is the Kalman filter. An early version of this architecture used for tracking faces was described in "Multi-modal tracking of faces for video communications," delivered in June 1997 at CVPR '97 (Computer Vision and Pattern Recognition).[9]

Tracking is classically composed of three phases: predict, detect, and estimate. The prediction phase projects the previously estimated attributes for each of a set of targets to a predicted value for the current time. The predicted target state is used by a detection process to interpret the current data to locate each target. The estimation phase updates the properties for tracked targets and edits the list to account for new and lost targets.

In the model in figure 6, these three classical tracking phases are completed by a recognition phase, an autoregulation phase, and a communication phase. In the recognition phase, the component executes recognition procedures to interpret the individual entity or groups of entities. Recognition procedures are interpreted by a lightweight language interpreter for the Lisp-like language Scheme.[10] In our implementation, such procedures may be preprogrammed, or they may be downloaded to the component during configuration as snippets of code.

In addition to recognition, the supervisory component provides execution scheduling, self-monitoring, parameter regulation, and communications. The supervisor also acts as a scheduler, invoking execution of modules in a synchronous manner. For self-monitoring, a component applies a model of its own behavior to estimate both quality of service and confidence for its outputs. Monitoring allows a process to detect and adapt to degradations in performance resulting from changing operating conditions. It does so by reconfiguring its component modules and operating parameters. Monitoring also enables a





process to provide a symbolic description of its capabilities and state.

Homeostasis, or "autonomic regulation of internal state," is a fundamental property for robust operation in an uncontrolled environment. A component is auto-regulated when processing is monitored and controlled so as to maintain a certain quality of service. The process supervisor maintains homeostasis by adapting module parameters to maximize estimated quality of service. For example, processing time and precision are two important state variables for a tracking process. Quality-of-service measures such as cycle time, number of targets, or precision can be maintained by dropping targets based on a priority assignment or by changing resolution for processing of some targets.

During the communication phase, the supervisor may respond to requests from other components. These requests may ask for descriptions of process state and process capabilities, or they may provide specifications of new recognition methods. The supervisor acts as a programmable interpreter, receiving snippets of code script that determine the composition and nature of the process execution cycle and the manner in which the process reacts to events.

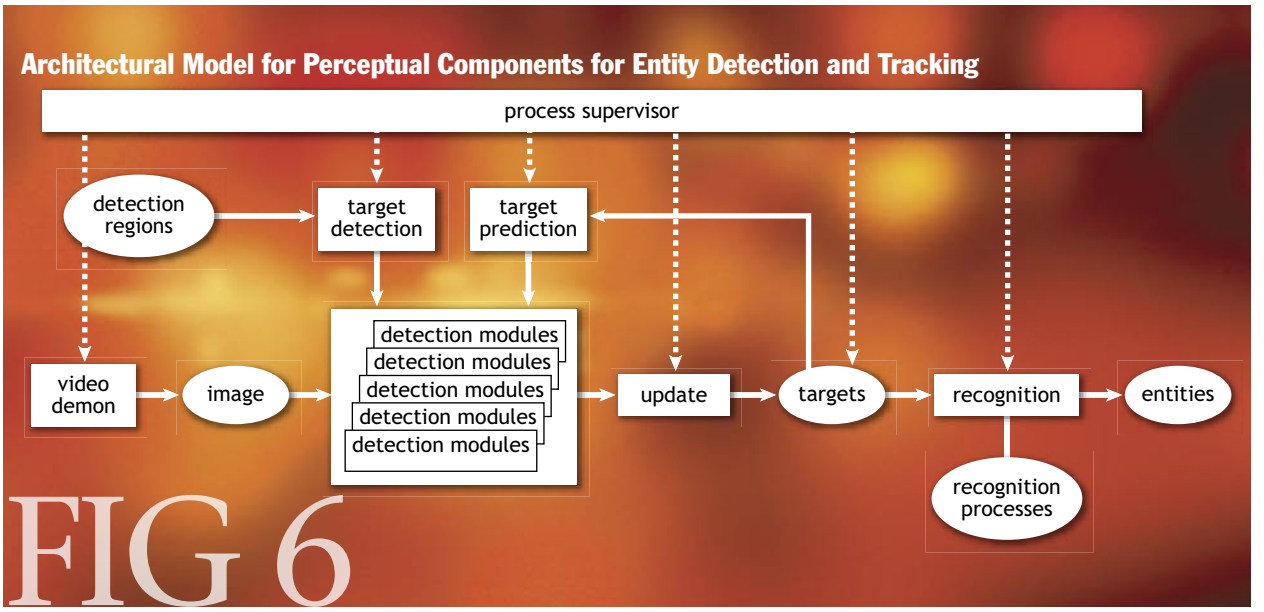
FEDERATIONS OF PERCEPTUAL COMPONENTS

A *federation*[11] is a collection of independent components that cooperate to perform a task. We have designed a middleware environment that allows us to dynamically launch and connect components on different machines. This environment provides an XML-based interface that allows components to declare input command messages and output data structures, as well as current operational state. Three classes of channels exist for communication between components: *events* are asynchronous symbolic messages that are communicated through a publish-and-subscribe mechanism provided by the federation supervisor; *streams* provide serial high-bandwidth data between two components; *requests* are asynchronous messages that ask for the current values of some process variables.

Bob's ideal videoconferencing system is a good example of a user service provided by a federation of perceptual components. Figure 7 shows the automatic recording and communications system that we have constructed for the European IST (Information Society Technologies) FAME (Facilitating Agent for Multicultural Communication) project. Such a system could be used to provide the automatic audiovisual composition for Bob's ideal system. At each site, a federation of components is assembled to detect information about meeting participants, such as their position, head orientation, speech events, and arm movements, as well as changes in a slide that is being discussed by a speaker. Events from this federation are filtered through role-assignment components to inform a situation model about the current configuration of actors and props. The current situation dictates the selection and composition of cameras to be transmitted, as well as the accompanying audio composition.

DEFINING SITUATION MODELS

One of the challenges of specifying a scenario is avoiding the natural tendency toward complexity. Over a series



Social Perception

of experiments, we have evolved a method for defining scenarios for services based on observing human activity. Our method is based on two principles and leads to a design process composed of six phases.

Principle 1: Keep it simple. In real examples, there is a natural tendency for designers to include entities and relations in the situation model that are not really relevant to the system task. It is important to define the situations in terms of a minimal set of relations to prevent an explosion in the complexity of the system. This is best obtained by first specifying the system behavior, then for each action specifying the situations, and for each situation specifying the entities and relations. Finally for each entity and relation, we determine the configuration of perceptual components that may be used.

The idea behind this principle is to start with the simplest possible network of situations and gradually add new situations. This leads to avoiding the definition of perceptual components for unnecessary entities.

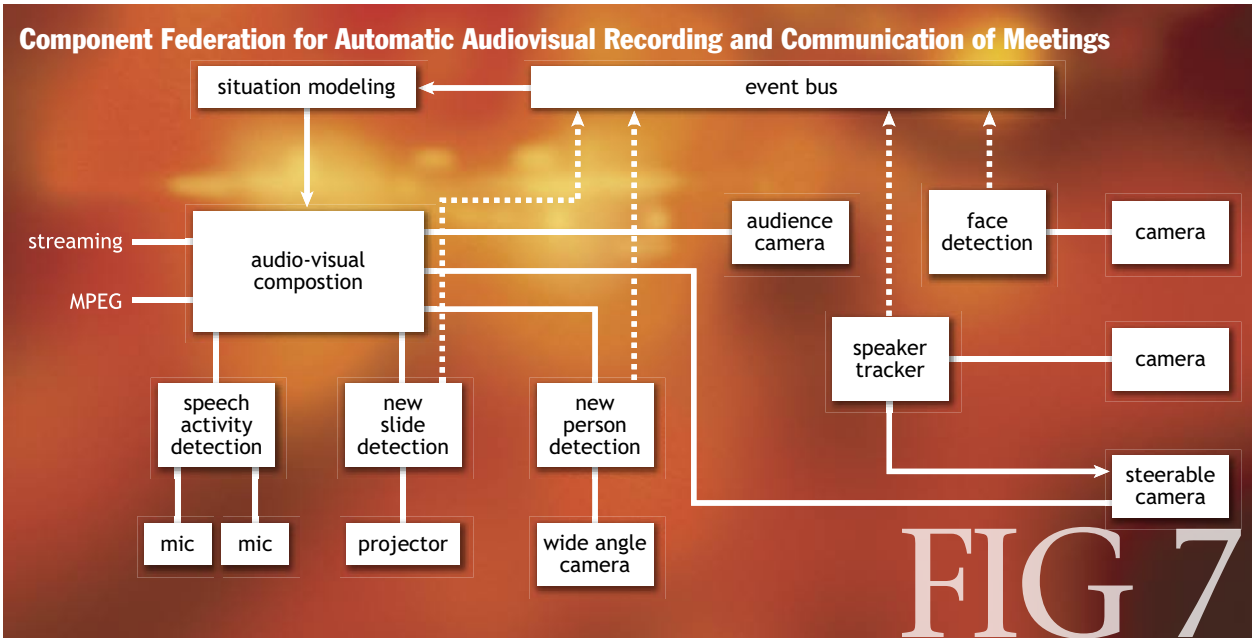
Principle 2: Behavior drives design. The idea behind this principle is to drive the design of the system from a specification of the actions that the service is to take. The first step in building a situation model is to specify the desired service behavior. For ambient informatics, this corresponds to specifying the set of actions that can be taken and formally describing the conditions under which such actions can or should be taken. For each action, the service designer lists a set of possible situations, where each situation is a configuration of entities and relations to be observed in the environment. Situations form a network, where the arcs correspond to changes in the roles or relations between the entities that define the situation. Arcs define the reaction to events.

These two principles are expressed in a design process composed of six phases.

Phase 1: Map actions to situations. The actions to be taken by the system provide the means to define a minimal set of situations to be recognized. The mapping from actions to situations need not be one-to-one. It is perfectly reasonable that several situations will lead to the same action. There can be only one action list for any situation, however.

Phase 2: Identify the roles and relations required to define each situation. A situation is defined by a set of roles and a set of relations between entities playing roles. A role acts as a kind of variable so that multiple versions of a situation played by different entities are equivalent. Determine a minimal set of roles and the required relations between entities for each situation.

Phase 3: Define filters for roles. Define the properties that must be true for an entity or agent to be assigned to a role, and design a similarity measure. Use this to sort the entities and select the most appropriate for each role.



Phase 4: Define components for observation. Define a set of perceptual components to observe the entities required for the roles and to measure the properties required for the relations. Define components to assign entities to roles and to measure the required properties.

Phase 5: Define the events. Changes in situations generate events. Events may be the result of changes in the assignment of entities to roles or changes in relations between the entities that play roles.

Phase 6: Implement, then refine. Given a first definition, implement the system. Extend the system by seeking the minimal perceptual information required to perform new actions appropriately.

A situation graph implements a finite-state machine. Human behavior is, of course, drawn from an unbounded set of actions, and therefore can never be entirely predicted by a finite-state machine. Thus, our model is most appropriate for tasks in which human behavior is regulated by a well-defined, commonly followed script. The lecture scenario is such an activity.

CONCLUSIONS

A situation model is a network of situations concerning a set of roles and relations. Roles are abstract classes for actors or props. An entity may be interpreted as playing a role, based on its current properties. Relations between entities playing roles define situations. This conceptual framework provides the basis for designing software services that can offer nondisruptive information and communication services.

Socially aware observation of activity and interaction is a key requirement for development of nondisruptive services. For this to come true, we need methods for robust observation of activity, as well as methods to automatically learn about activity without imposing disruptions. The framework and techniques described here are intended as a foundation for such observation. Q

REFERENCES

1. Dey, A. K. 2001. Understanding and using context. *Personal and Ubiquitous Computing* 5(1): 4-7.
2. Brdiczka, O., Maisonnasse, J., Reignier, P. 2005. Automatic detection of interaction groups. International Conference on Multimodal Interaction, Trento, Italy.
3. Finkelstein, A., Kramer, J., Nuseibeh, B., eds. 1994. *Software Process Modeling and Technology*. Research Studies Press, John Wiley and Sons Inc.
4. Rasure, J., Kubica, S. 1994. The Khoros application development environment. In *Experimental Environments for Computer Vision and Image Processing*, ed. H. Christensen and J. L. Crowley, 1-32. World Scientific Press.
5. Crowley, J. L. 1995. Integration and control of reactive visual processes. *Robotics and Autonomous Systems* 15(1).
6. Shaw, M., Garlan, D. 1996. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall.
7. Schwerdt, K., Crowley, J. L. 2000. Robust face tracking using color. 4th IEEE International Conference on Automatic Face and Gesture Recognition, France.
8. Piater, J., Crowley, J. L. 2002. Event-based activity analysis in live video using a generic object tracker. Performance Evaluation for Tracking and Surveillance (PETS-2002), Copenhagen (June).
9. Crowley, J. L., Berard, F. 1997. Multi-modal tracking of faces for video communications. In *Proceedings of Computer Vision and Pattern Recognition*: 640-645.
10. Lux, A. 2003. The Imalab method for vision systems. International Conference on Vision Systems (ICVS-03), Graz.
11. Estublier, J., Cunin, P. Y., Belkhatir, N. 1997. Architectures for Process Support Interoperability (ICSP5), Chicago.

ACKNOWLEDGMENTS

This work is supported by the EC IST projects, FAME and IP CHIL, as well as French national project RNTL/ProAct ContAct. This work has been performed in collaboration with Joelle Coutaz, Gaetan Rey, Patrick Reignier, Dave Snowdon, Jean-Luc Meunier, and Alban Caporossi.

LOVE IT, HATE IT? LET US KNOW

feedback@acmqueue.com or www.acmqueue.com/forums

JAMES L. CROWLEY directs the GRAVIR laboratory at the INRIA Rhône-Alpes research center in Montbonnot, France. He holds the post of professor at the Institut National Polytechnique de Grenoble (INPG), where he teaches courses in computer vision, signal processing, pattern recognition, machine learning, and artificial intelligence at l'ENSIMAG (Ecole National Superieure d'Informatique et de Mathematiques Appliquées). Over the past 25 years, Crowley has edited two books and five special issues of journals, and written more than 180 articles on computer vision and mobile robotics. He and his collaborators are developing techniques for acoustic and visual perception of human activity, with applications to interactive environments and new forms of man-machine interaction. He has a Ph.D in electrical engineering from Carnegie Mellon University.