To appear in ACM Queue, 2006

Situation Models for Observing Human Activity

James L. Crowley
Professor I.N.P. Grenoble
INRIA Rhône Alpes
Montbonnot, France
{James.Crowley@inrialpes.fr}

Abstract

Many human activities follow a loosely defined script in which individuals assume roles. Encoding such scripts in a formal representation makes it possible to build systems that observe human activity in order to provide services. In this paper, we present a conceptual framework in which scripts for human activit y are described as scenarios composed of actors and objects within a network of situations. We provide formal definitions for the underlying concepts for situation models, and then propose a layered, component-based, software architecture model for constructing systems to observe human activity using such scripts.

1. Introduction

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 would include

- automated camera control for video-conferencing,
- 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, as well as
- tools for organizing and conducting meetings.

This is an open ended set, limited only by ones ability to imagine. Unfortunately, despite the presence of enabling communications technology, none of these services can be said to have entered a virtuous spiral of exponential growth.

We believe that 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.

In this paper we propose a conceptual framework and a software model for observing and modeling human activity, including human-to-human interaction. The core component of our framework is a situation model. The situation model acts as a non-linear 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 non-disruptive 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.

2. Situated observation of human activity

Most human societies have developed and refined an artistic framework for socially aware encoding and observing of human interaction: the Theater. A theatrical production provides a model for social interaction in the form of a script. Theater can be used as a rich source of concepts for socially aware observation of human activity.

A theatrical production organizes the actions of a set of actors in terms of roles structured as 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 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 class-room teaching, formal meetings, shopping, or dining at a restaurant, follow highly structured scripts that constrain individual actions to a 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 to guide the behavior of services in order 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 non-deterministic branching. The complexity and difficulty of observing human activity is related to the degree of interconnectivity of situations.

3. 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. However, presentation of such a conceptual framework may be top-down or bottom up. We could choose a bottom up presentation, starting with software architectural concepts and constructing a formal hierarchy of increasing abstraction. Alternatively, presentation can begin from progressively refining abstract concepts into concrete programming models. In earlier papers [5], [4], we have tried both approaches. We find that top down, abstract to concrete development leads to a clearer and more engaging presentation.

In this section we propose a hierarchy of definitions for concepts for observing human activity. What we are modeling is sometimes called a "Context model".

Context: The situation within which something exists or happens, and that can help explain it [7]; Any information that can be used to characterize situation. [8].

However, context is a highly overloaded term, meaning different things to different people. To avoid confusion, we propose to refer to models for observing activity as "scenarios". The Cambridge On-line dictionary [7] defines a scenario as:

Scenario: A description of possible actions or events in the future; A written plan for the characters and events in a play or movie; [7]

We propose to define scenario as

Scenario: A network of situations for modeling human activity expressed in terms of relations between entities playing roles.

In common use, situation derives its meaning from the way in which something is placed in relation to its surroundings. For examples, many authors define situation in terms of position and action. The Cambridge on line dictionary defines situation as

Situation: the set of things that are happening and the conditions that exist at a particular time and place. [7].

In our case, the definition of situation requires defining two facets: Observation and Reaction. Observation refers to the concepts with which the system can observe situation. Reaction refers to the set of actions that the system should take based on the current situation.

The observation facet of situation is defined in terms of the configuration of a set of entities playing roles. Configuration is expressed as a set of predicate functions whose arguments are the entities playing the roles.

Situation: A predicate expression of a set of relations over entities assigned to roles.

As a predicate expression, a situation is a form of state. In this case, it refers to the state of the activity as defined by a logical expression. This expression is composed of predicates whose arguments are roles. This concept generalizes and extends the common practice of defining situations based on the relative position of actors and objects.

Relations are predicate (truth) functions with one or more arguments. As predicates, relations are either true or false, depending on the properties of their arguments. 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.).

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

Relations test the properties of entities that have been assigned to roles. Operationally, 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 the set of actions that the entities can enable (props). Formally, role is a function that selects an entity from the set of observed entities.

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

The definition of role can be completed by definitions for actors and props.

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

Prop A role for entities that can not spontaneously act to change the current situation.

A “role” is NOT an intrinsic property of an entity, but rather, is an interpretation assigned to an entity by the system. The role function acts to select an entity from the available set of entities, and not the other way around.

To summarize so far, we have situations defined with logical expressions composed of relations over entities playing roles, and producing a set of actions to be taken by the system. As mentioned above, situations also predict possible future situations. This is captured by the connectivity of a situation network. Changes in the logical expression of relations or in the selection of entities playing roles are represented as changes in situation. Such changes can trigger system actions.

So how does the role assignment process select among the available entities? Brdizcka has 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 only see 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 below. Operational definitions for property and entity are grounded in the software components for observation of activity.

4. A Software architecture for observing activity

We propose a layered architectural model for services based on human activity. Four layers of this model are 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.

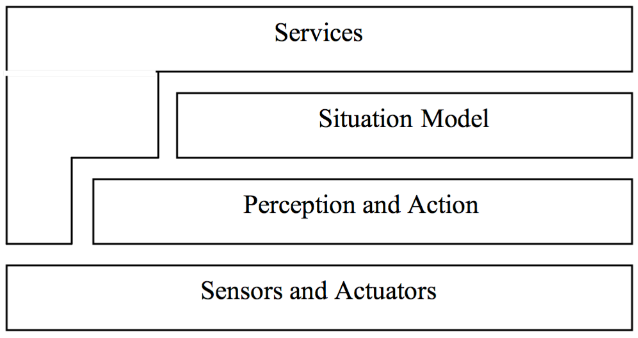

Fig. 1. *A layered model for non-disruptive user services.*

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 API's, 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 described above. 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.

5. The perception-action layer

At the perception-action layer, we propose a data-flow process architecture for software components for perception and action [9], [12], [6]. Component based architectures, as described in Shaw and Garlan [14], are composed of auto-descriptive functional components joined by connectors. Such an architecture is well adapted to interoperability of components, and thus provides a framework by which multiple partners can explore design of specific components without having to rebuild the entire system.

Within the perception-action layer, we propose three distinct sub-layers, as shown in figure 2. These three sub-layers are the Module layer, the Components layer and Federation layer. 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.

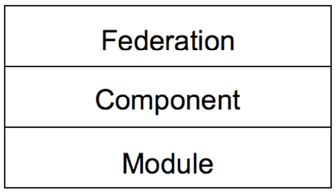

Fig. 2. *Three sublayers within the perception action layer.*

5.1 Modules

Modules are auto-descriptive components formally defined as synchronous transformations applied to a certain class of data or event, as illustrated in Figure 3. Modules generally have no state. They are executed by a call to a method (or function or subroutine) accompanied by a vector of parameters. The vector of parameters 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 to 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.

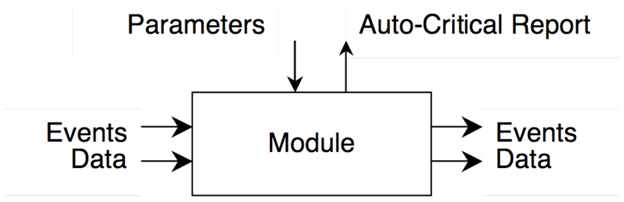

Fig. 3. *Modules apply a transformation to data or events and return a report.*

An example of a module is provided by 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 as a lookup table representing a ratio of color histograms [13]. A common use for such a module is to detect skin colored pixels within regions of an image as shown in figure 4.

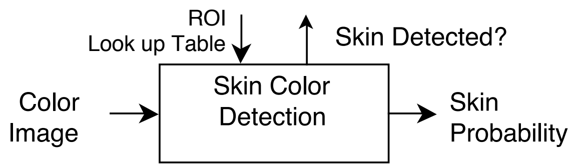

Fig. 4. *A module for detecting skin pixels*

5.2 Peception and Action Components

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

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 of processing by selecting 1 pixel of N [11] or to selectively delete targets judged to be uninteresting.

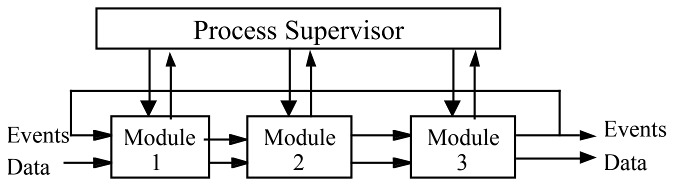

Fig. 5. A perceptual component integrates a set of modules to transform data streams or events.

5.3 A model for perceptual components

An architectural model for perceptual components is shown in figure 6. The functional core in this model provides tracking of entities. Tracking is a cyclic process of recursive estimation of the properties of a set of targets. A well-known framework for such estimation is the Kalman filter. An early version of this architecture used for tracking faces is described in [17].

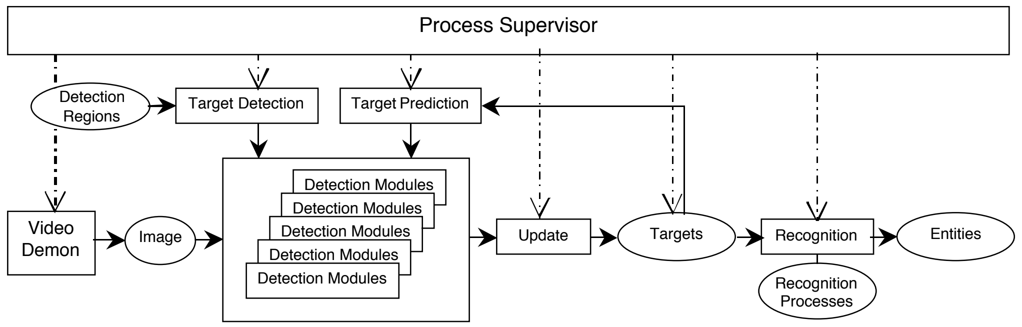

***Fig. 6**. Architectural model for perceptual components for entity detection and tracking*

Tracking is classically composed of four phases: Predict, observe, detect, and update. The prediction phase updates the previously estimated attributes for a set of targets to a value predicted for a specified time. The observation phase applies the prediction to the current data to update the state of each target. The detect phase detects new targets. The update phase updates the list of targets to account for new and lost targets.

These phases may be completed by a recognition phase, an auto-regulation phase, and a communication phase. In the recognition phases, the tracker interprets recognition methods that have been downloaded to the component during configuration. Such methods are bits of code that may be expressed in a declarative language such as scheme, CLIPS or C++ to be interpreted by a component interpreter [10].

The auto-regulation phase determines a "quality of service" measure, such as total cycle time or resolution, and adapts the module parameters as well as the list of targets to maintain a desired quality, as described below. During the communication phase, the supervisor may respond to requests from other components. These requests may ask for descriptions of process state, process capabilities, or may provide specification of new recognition methods.

The supervisory component of a process provides command interpretation, execution scheduling, event handling, parameter regulation, and reflexive description. 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. The supervisor acts as a scheduler, invoking execution of modules in a synchronous manner. The supervisor handles event dispatching to other components, and reacts to events from other components. The supervisor monitors and regulates module parameters based on the execution results. Auto-critical reports from modules permit the supervisor to dynamically adapt processing. Finally, the supervisor responds to external queries with a description of the current state and capabilities. We formalize these abilities as the autonomic properties of self-monitoring, auto-regulation, auto-description and auto-criticism.

Homeostasis or "autonomic regulation of internal state" is a fundamental property for robust operation in an uncontrolled environment. A process 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 using the auto-critical reports. 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.

A self-monitoring component maintains a model of its own behavior in order to estimate both quality of service and confidence for its outputs. Monitoring allows a process to detect and adapt to degradations in performance due to changing operating conditions by reconfiguring its component modules and operating parameters. Monitoring also enables a process to provide a symbolic description of its capabilities and state. The description of the capabilities includes both the basic command set of the controller and a set of services that the controller may provide to a more abstract supervisor. Such descriptions are useful for both manual and automatic composition of federations of components.

5.4 Component federations

A federation [15] is a collection of independent components that cooperate to perform a task. We have designed a middle-ware 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, output data structures, as well as current operational state. Three classes of channels exist for communication between components: events, streams and requests. Events are asynchronous symbolic messages that are communicated through a publish and subscribe mechanism provided by the Federation Supervisor. Streams provided serial high bandwidth data between two components. Requests are asynchronous messages that ask for the current values of some process variables.

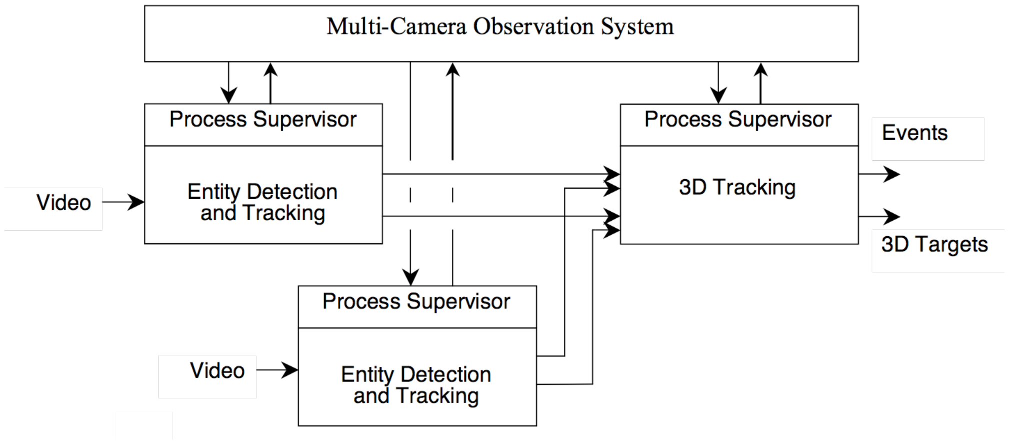

Fig. 6. A process federation using multiple entity detection components for 3D tracking.

As a simple example of a federation of perceptual components, consider a system that integrates targets from multiple cameras to provide 3-D target tracking, as shown in figure 6. This system uses an entity and detection tracking process that can use background difference subtraction and motion to detect and track blobs in an image stream. The set of tracked entities is sent to a composition process that tracks targets in 3D scene coordinates.

6. Defining situation models

One of the challenges of specifying a scenario is avoiding the natural tendency towards complexity. Over a series 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, we have noticed that 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 behaviour, 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 then gradually add new situations. This leads to avoiding the definition of perceptual components for unnecessary entities.

Principle 2: Behaviour 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 behaviour. For an 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. However, there can only be one action list for any situation.

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. Roles act 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 the 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 results 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 appropriately perform new actions.

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

7. Conclusions

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

Socially aware observation of activity and interaction is a key requirement for development of non-disruptive 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 in this paper are intended as a foundation for such observation.

Acknowledgment

This work has been supported by the EC IST projects FAME project (IST-2000-28323) and IP CHIL as well as French national project RNTL/ProAct ContAct. This work has been performed in active collaboration with Joelle Coutaz, Gaetan Rey, Patrick Reignier, Dave Snowdon, Jean-Luc Meunier and Alban Caporossi.

References

[1] J. Allen, "Maintaining Knowledge about Temporal Intervals", Journal of the ACM, 26 (11) 1983.

[2] O. Brdiczka, J. Maisonnasse, P. Reignier, Automatic Detection of Interaction Groups, 2005 International Conference on Multimodal interaction, ICMI '05, Trento It., october 2005

[3] J Coutaz, J. L. Crowley, S. Dobson, and D. Garlan, "Context is Key", Communications of the ACM, Special issue on the Disappearing Computer, Vol 48, No 3, pp 49-53 March 2005.

[4] J. L. Crowley, J. Coutaz and F. Berard, "Things that See: Machine Perception for Human Computer Interaction", Communications of the A.C.M., Vol 43, No. 3, pp 54-64, March 2000.

[5] J. L. Crowley, J. Coutaz, G. Rey and P. Reignier, "Perceptual Components for Context Aware Computing", UBICOMP 2002, International Conference on Ubiquitous Computing, Goteborg, Sweden, September 2002.

[6] J. L. Crowley, "Integration and Control of Reactive Visual Processes", Robotics and Autonomous Systems, Vol 15, No. 1, decembre 1995.

[7] Cambridge On-line dictionary of the English Language, http://dictionary.cambridge.org

[8] Dey, A. K. “Understanding and using context”, Personal and Ubiquitous Computing, Vol 5, No. 1, pp 4-7, 2001.

[9] Software Process Modeling and Technology, edited by A. Finkelstein, J. Kramer and B. Nuseibeh, Research Studies Press, John Wiley and Sons Inc, 1994.

[10] A. Lux, "The Imalab Method for Vision Systems", International Conference on Vision Systems, ICVS-03, Graz, april 2003.

[11] J. Piater and J. Crowley, "Event-based Activity Analysis in Live Video using a Generic Object Tracker", Performance Evaluation for Tracking and Surveillance,

[12] J. Rasure and S. Kubica, “The Khoros application development environment “, in Experimental Environments for computer vision and image processing, H. Christensen and J. L. Crowley, Eds, World Scientific Press, pp 1-32, 1994.

[13] K. Schwerdt and J. L. Crowley, "Robust Face Tracking using Color", 4th IEEE International Conference on Automatic Face and Gesture Recognition", Grenoble, France, March 2000.

[14] M. Shaw and D. Garlan, Software Architecture: Perspectives on an Emerging Disciplines, Prentice Hall, 1996.

[15] J. Estublier, P. Y. Cunin, N. Belkhatir, "Architectures for Process Support Ineroperability", ICSP5, Chicago, 15-17 juin, 1997.

[17] J. L. Crowley and F. Berard. "Multi-modal tracking of faces for video communications". In Proc. of Computer Vision and Pattern Recognition, CVPR 97, pages pp 640-645, June 1997.