An Ontology for Video Event Representation

Ram Nevatia, Jerry Hobbs and Bob Bolles

Institute for Robotics and Intelligent Systems
University of Southern California
Los Angeles, CA 90089-0273
nevatia@iris.usc.edu

USC Information Sciences Institute
4676 Admiralty Wa,
Marina del Rey, CA 90292
hobbs@isi.edu

SRI International
333 Ravenswood Avenue
Menlo Park, CA 94040
bolles@ai.sri.com

Abstract

Representation and recognition of events in a video is important for a number of tasks such as video surveillance, video browsing and content based video indexing. This paper describes the results of a "Challenge Project on Video Event Taxonomy" sponsored by the Advanced Research and Development Activity (ARDA) of the U.S. Government in the summer and fall of 2003. The project brought together more than 30 researchers in computer vision and knowledge representation and representatives of the user community. It resulted in the development of a formal language for describing an ontology of events, which we call VERL (Video Event Representation Language) and a companion language called VEML (Video Event Markup Language) to annotate instances of the events described in VERL. This paper provides a summary of VERL and VEML as well as the considerations associated with the specific design choices.

1. Introduction

Representation and recognition of events in a video is important for a number of tasks such as video surveillance, video browsing and content based video indexing. It would be highly useful if the research community and the users agreed on a common representation for describing events as it would then allow easy interchange of video annotations and sharing of video recognition modules among researchers. Advanced Research and Development Activity (ARDA) of the U.S. Government sponsored a "Challenge Project on Video Event Taxonomy" to further this goal in the summer and fall of 2003. The project brought together more than 30 researchers in computer vision and knowledge representation and representatives of the user community. It resulted in the development of a formal language for describing an ontology of events, which we call VERL (Video Event Representation Language) and a companion language called VEML (Video Event Markup Language) to annotate instances of the events described in VERL. This paper provides a summary of VERL and VEML as well as the considerations associated with the specific design choices. We hope that continued dialog and use of these tools will help in their further refinement and broader acceptance.

Our work draws on previous research in computer vision as well as in knowledge representation. In the computer vision community, Ivanov and Bobick [5] suggested use of context-free grammars. Nevatia, Hongeng and Zhao [7] advocated use of hierarchical decomposition and suggested use of single/multiple thread terminology. Vu, Bremond and Thonnat have developed similar concepts [8].

In the artificial intelligence literature, Narayanan [6] develops a formalism for the execution of actions and then applies it to several problems in linguistics, including the problem of representing different aspects, *e.g.*, continuing or completed. More recently, this work was an important influence in the development of the process component of the web services language DAML-S [2].

There has been a great deal of research on temporal reasoning. Much of it has been based on the work of Allen and his colleagues [1] and, we incorporate that work as well. Much of the work on temporal reasoning has focused on temporal constraint satisfaction (*e.g.*, [3]) but we did not concentrate on this aspect in our project.

2. Issues in the design of a Representation Language

An event representation language needs to be able to represent the wide variety of events we observe in our everyday world. The representation needs to be formal but still be natural for human users. The representation language needs to be flexible to allow new event classes to be added incrementally. The representation should be useful for annotating instances in a video, to allow causal and other inferences to be made from the annotations and be useful for automatic recognition of events from video data.

An ontology of events requires a means of describing the structure and function of events. The structure tells us how an event is composed of lower-level states and events. This structure can involve a single agent or multiple agents and objects. For a specific domain, composition has to start somewhere, so we need to state what the primitive events are. The function tells us about the roles an event plays in its environment, and how it, in turn, participates in larger-scale events.

In our representation, we make the critical assumption that complex events can be decomposed into simpler events. *Primitive* events are the simplest type of events inferred directly from the observables in the video data. We define different types of compositions of events with *sequencing* (one after the other) being the most common. We further distinguish between *single thread* events where all sub-events can be placed in a linear order and *multi-thread* events where at least some of the sub-events occur simultaneously; the former are typically performed by a single agent, the latter are typically, but not necessarily, performed by multiple agents.

A caveat is in order regarding the terms "primitive" and "composite." An event that is primitive from one perspective can be composite from another. We can regard walking from one point to another as a primitive component in a larger action. But from another point of view, walking is a complex action consisting of repetitions of moving one leg forward and then moving the other leg forward. Moving a leg forward can itself be viewed as lifting the foot, swinging the leg forward, and placing the foot down. If we had sensors on the muscles of the leg, we could break this down into even smaller scale events. In principle, this kind of analysis could be extended almost arbitrarily far down to lower levels or up to higher levels.

If our hypothesis of decomposing complex events into simpler events is valid and if the number of primitive events is limited, we can overcome the complexity of representing the wide variety of events seen in the real world. It will allow us to not only share annotations but also the modules for recognizing primitive and composite events in a unified framework thus greatly speeding up the process of developing automated video event recognition systems.

3. A Video Event Representation Language (VERL)

We now describe the formal Video Event Representation Language called VERL.

3.1. Objects, States, and Events

We start with the notion of *objects* and distinguish *Mobile Objects* from *Contextual* Objects.

Objects have *properties* or *attributes*; logically we can think of these as one-argument predications. They also stand in *relations* to other objects; we can think of these as predications with two or more arguments. Properties, attributes, and relations can be thought of as *states*. The term "state" as used in computer science, means the aggregate of all the properties and relations of all relevant entities *at a given moment in time.* We can call this a *w-state*, for world state. In ordinary language, however, we talk about the state of a single entity that persists over time, e.g., the state of John's being sick. We can call this an *e-state*, for entity state. In this paper, when we refer to state, it is the e-state that we have in mind.

An *event* is a change of state in an object. Thus when a rock rolls down a hill, its location changes. We can write: *change*(*p*(*x*),*q*(*x*)) or *change*(*at*(*x*,*y*),*at*(*x*,*z*)).

Events generally have a time instant or interval where or during which they occur. In Section 3.8, we mention an ontology of time that can be imported into VERL. Events generally have locations as well, inherited from the locations of their participants. An ontology of location could also be imported.

States and events can *cause* other states and events. For example, a rock hitting a window can cause the window to shatter. We can have a predicate *cause* relating states or events (the cause) to other states or events (the effect), for example:

$cause(chang(p_1(x), q_1(x)), change(p_2(y), q_2(y)))$

This says that a change of state in x from p_1 being true to q_1 being true causes a change in y from p_2 being true to q_2 being true. An example of this would be when a baseball (x) changing its location (*i.e.*, moving) causes a window (y) to change from being intact to being broken.

3.2. Types

We take an annotation to be a pair consisting of a *thing* in a VERL ontology and a designation of a *location* in the video data.

<*thing, loc*>

The *thing* describes a state or event, or an entity such as a physical object. We will not say anything here about how the locations are to be specified.

There are three basic types in the language. Everything is a *thing*. There are two types of things. The type *ent* encompasses entities, and generally may be thought of as physical objects, although in some applications it can be used more broadly. The type *ev* encompasses states and events, where by "state" we mean the ordinary language notion of e-state, not the computer science notion of w-state. Normally, a person would be of type *ent*, and his or her running would be of type *ev*. The type hierarchy is

thing
/ \
ent ev

It is possible in specific applications to expand this hierarchy to more specific types. For example, one might introduce *person* and *vehicle* as subtypes of type *ent*.

We will refer loosely to things of type *ent* as entities and things of type *ev* as events.

3.3. VERL Expressions

Constants may be of any one of the three types. Variables may range over any one of the three types.

A VERL expression (*vexpr*) is defined as follows:

A constant or variable is a vexpr.

vexpr --> constant | variable

For example, "John," "X1," "Fire-1," and "E1" may all be vexprs. The type of the *vexpr* is the type of the constant or variable. Thus, "John" is an entity constant, "E1" will be an event variable if, for example, it refers to John's running, and so on.

A function symbol applied to the appropriate number of vexprs as arguments is a *vexpr*.

vexpr --> fct "(" [vexpr { "," vexpr }] ")"*

(Square brackets [...] indicate something is optional; here the function may have no arguments. Curly brackets {...} group elements together. The *Kleene star* * means zero or more instances of.) The arguments must be of the right type. For example, if *head-of* is a function taking one entity *vexpr* as its argument, then *head-of* (*John*) is a *vexpr*. The function determines what type of thing the result is. Thus, *head-of* would be an entity function and *head-of* (*John*) would be an entity.

A predicate symbol applied to the appropriate number of arguments is a vexpr.

*vexpr --> pred "(" [vexpr { "," vexpr } *] ")"*

The arguments must be of the right type. The result is always of type *ev*. For example, if *change* is a predicate symbol relating two things of type *ev*, then *change* (*E1,E2*) is a *vexpr* of type *ev*.

A logical operator applied to the appropriate number of vexprs of type *ev* is a vexpr.

*vexpr --> "AND" "(" vexpr {"," vexpr } * ")" |*
"OR" "(" vexpr { "," vexpr }) " |*
"IMPLY" "(" vexpr "," vexpr ") " | "NOT" "(" vexpr ") " |
"EQUIV" "(" vexpr "," vexpr ") "

AND and *OR* take one or more arguments. *IMPLY* and *EQUIV* take two arguments. *NOT* takes one argument. The result is always of type *ev*.

Things of type *ev* can be events types or event tokens. *NOT* takes an event type as its argument. Thus, if we say that *NOT* (*run*(*John*)) occurs at a location in the video data, then we are saying that no event of type (*run*(*John*)) occurs there.

A constant or variable can be used as a *label* on a vexpr.

vexpr --> {constant | variable} ":" vexpr

The resulting vexpr refers to the same thing as its constituent vexpr and of course is of the same type. The label can then be used elsewhere to refer to that thing. If we did not have labels, ambiguities could result. Suppose twice in a file we refer to John's running.

run(John) ... run(John)

These may or may not be the same instance of running. If we say *E1*: *run(John) ... E1,* they definitely are the same instance.

3.4. Defining Composite Events in VERL

The basic operator for defining composite events is *PROCESS*. It takes a predication and a vexpr as its two arguments. The predication is a predicate applied to the appropriate number of arguments, where the arguments have an optional type specification.

defn --> "PROCESS" "(" pred "(" [argspec {"," argspec} ")"*
["," vexpr] ")"
argspec --> [type] variable

The second argument of *PROCESS* is optional, and if it is missing, it is assumed the process is primitive, i.e., in the given application it is directly implemented in software.

For example, if we have the predicate *located-at* relating a thing to an entity, and a predicate *change* relating two things of type *ev*, then we can define the predicate *move* as follows:

PROCESS(move(thing x, ent y, ent z),
change(located-at(x,y),located-at(x,z)))

That is, for a thing *x* to move from entity *y* to entity *z*, there is a change in *x*'s location from *y* to *z*.

Labels defined inside a *PROCESS* statement are local to that *PROCESS* statement. Thus, if we write

PROCESS(move(thing x, ent y, ent z),
change(e1:located-at(x,y),e2:located-at(x,z)))

We can't write "*e1*" or "*e2*" outside of the *PROCESS* statement and expect it to refer to the same thing it refers to inside the *PROCESS* statement.

Three other operators can be used optionally in place of *PROCESS—PRIMITIVE, SINGLE-THREAD,* and *MULTITHREAD*. *PRIMITIVE* can be used when there is no second argument to the *PROCESS* definition. *SINGLE-THREAD* means that all constituent events in the definition happen sequentially, without overlap. *MULTI-THREAD* can be used when there is no such constraint.

For example, suppose *located-at* and *change* are primitive predicates. Then a *move* event is a single-thread event. So we could rewrite the above example as

PRIMITIVE(located-at(thing x, ent y))
PRIMITIVE(change(ev e1, ev e2))
SINGLE-THREAD(move(thing x, ent y, ent z),
change(located-at(x,y),located-at(x,z))

3.5. Defining Subtypes

The user may want to define subtypes of the types *ent* and *ev*. This can be done with the *SUBTYPE* operator. It takes two arguments, the name of the subtype and the name of the supertype.

SUBTYPE (type, type)

For example, to say that *ent* has subtypes *mobile* and *immobile* and *mobile* has subtypes *vehicle* and *human*, we would write

SUBTYPE(mobile, ent)
SUBTYPE(immobile, ent)
SUBTYPE(vehicle, mobile)
SUBTYPE(human, mobile)

Sibling types in the type hierarchy should be mutually exclusive. Thus, if you were to specify *mobile* and *container* as subtypes of *ent*, you could not have mobile containers. It is often better to treat such concepts as properties rather than as types. In this case, the definitions would be

PRIMITIVE(container(ent x))
PRIMITIVE(mobile(ent x))

3.6. Inference Rules

In addition to annotations of specific events and definitions of composite properties, relations, and events, we may also want to specify inference rules that allow us to draw conclusions from what we recognize in the data. For this we use the operator *RULE*, which takes two vexprs of type *ev* as its arguments.

RULE(vexpr,vexpr)

A *RULE* is an implication; the first vexpr implies the second. For example, suppose we define *carry(x,y,a,b,t)* (*x* carries *y* from *a* to *b* during time interval *t*), as *x* holds *y* during *t* and *x* moves from *a* to *b* during *t*.

PROCESS(carry(x,y,a,b,t),
AND(hold(x, y,t), move(x,a,b,t)))

Then if we want to say that when *x* carries *y* from *a* to *b* during *t*, *y* also moves from *a* to *b* during *t*, we can say

RULE(IMPLY(carry(x,y,a,b,t), move(y,a,b,t)))

Variables in the antecedent of the implication will be interpreted as universally quantified; variables that occur in the consequent but not in the antecedent will be interpreted as existentially quantified. Thus,

RULE(p(x,z), q(x,y))

will be interpreted as

(A x,z)[p(x,z) --> (E y) q(x,y)]

3.7. Control Structures in VERL

Constructing complex events by composition of simpler events is central to our representation scheme. We can distinguish between *single-thread* events, in which only one thing is happening at a given time, and *multithread* events in which more than one thing is happening.

The most fundamental relation among component events is one of *sequence*. First one event happens and then another event happens. We can encode this if we reify events*, i.e.,* treat events as individuals that can be referred to by constants and variables in our logic or language. The expression *Sequence(e1,e2)* describes the composite event consisting of event *e1* happening followed by event *e2* happening; the events occur in sequence and do not overlap. Longer sequences can be constructed, so we allow *Sequence* to take an arbitrary number of arguments:

Sequence (*e1,e2,e3,...*).

The resulting vexpr describes the composite event consisting of all the argument events occurring in sequence.

But *Sequence* is not enough for recognizing higher-level events. Walking is not best described as a sequence of moving the right foot forward, then moving the left foot forward, then moving the right foot forward, and so on. It is in fact an iteration or a loop of the sequence of two actions, moving one foot forward, and then moving the other foot forward. So an account of event composition requires the concept of loops or *iteration*.

We can write *Repeat-Until(e1,e2)* to describe the composite event of an iteration of event type *e1* happening until state *e2* holds. *Repeat-Until* takes two things of type *ev* as its arguments. The resulting vexpr describes the composite event in which the first argument is repeated until the second argument holds or obtains.

Repeat-Until(e1, e2)

Normally, *e1* is the sort of event that changes the world in a way that affects whether or not *e2* holds or obtains.

A *While-Do(e1,e2)* construct is also part of VERL. *While-Do* takes two things of type *ev* as its arguments. The resulting vexpr describes the composite event in which the second argument is repeated as long as the first argument holds or obtains.

While-Do(e1,e2)

While-Do can be defined in terms of other operators:

PROCESS(While-Do(ev e1, ev e2),
Conditional(e1,
Repeat-Until(e2, NOT(e1))))

Alternation is also required. Suppose we are watching a man on an assembly line taking objects off the conveyer belt and tossing them into Bin A or Bin B. We may not be able to determine a better description of this action than as an iteration of alternative actions. We can express alternation in terms of the logical operator *OR*.

Conditionals are required as well. We may be able to determine that the man is looking at the objects on the conveyer belt and tossing the round ones into Bin A and the square ones into Bin B. In this case, the best description of the composite event is as an iteration with a conditional in the loop. The man is able to make observations and conditionalize his actions on what is observed.

In VERL, *Conditional* takes two or three events as its arguments. It says that if the first argument holds or obtains, then the second argument happens; otherwise the third argument happens, if there is a third argument.

Conditional(e1, e2)
Conditional(e1, e2, e3)

The resulting vexpr describes a piece of behavior that has been recognized to operate in this fashion.

This completes the specification of the syntax of VERL. We next present a class of predicates that are very

useful in building up and relating complex structured events.

3.8. Temporal Relations

Representing the temporal relations among component events is crucial in recognizing composite events. For most applications, describing the relations among the temporal intervals occupied by the component events, according to Allen's interval algebra (Allen and Ferguson, 1997), is sufficient. This is because agents are responding primarily to the actions of other agents or the behavior of moving objects. Even in a case where one of the threads is precisely timed, such as a conveyor belt in a factory, the worker is responding primarily to the appearance of the part rather than to the passage of a certain amount of time.

Times come in two varieties—*instant*s and *interval*s. Thus,

SUBTYPE(temporal-entity, thing)
SUBTYPE(instant, temporal-entity)
SUBTYPE(interval, temporal-entity)

Of two distinct instants, one is *before* the other. The predicate *after* is the inverse of *before*.

before(t1,t2), after(t1,t2), t1=t2

An instant *t* and an interval *T* can be in several possible relations:

begins(t,T), inside(t,T), ends(t,T)

It may be that none of these is true.

There are six possible basic relations that can obtain between two intervals:

before(T1,T2), meets(T1,T2), overlaps(T1,T2), begins(T1,T2), contains(T1,T2), ends(T1,T2)

These form the basis of Allen's interval algebra. They can be defined in terms of *begins, inside*, and *ends* relations between instants and intervals.

There are two possible relations between events and times: Some events happen instantaneously. In this case, we say

at-time(e,t)

where *t* is an instant. Some events happen across intervals, with a duration. In this case we say

during(e,T)

where *T* is an interval.

The OWL-Time ontology [4] provides a rich elaboration of these concepts and includes treatments of measures of duration, clock and calendar terms, and temporal aggregates. In some cases, this richer theory of time is required.

Many actions are rhythmic, and recognizing this is an important part of recognizing the higher-level event. A rhythmic event can be characterized as an iterative event in which the iterations occupy time intervals of equal duration. Rhythm is often used to coordinate multithread iterative action. Finally, it is sometimes necessary to relate events to the clock and calendar.

3.9. The Semantics of VERL

One of our design goals was to give VERL expressions a clear semantics, by translating them into equivalent first order logic sentences. In most cases, the semantics should be obvious, but the are several hidden subtleties. We are unable to include the details of this semantics due to space limitations but expect to document them in a forthcoming report.

4. An Example: An Abstract Ontology of Mobile Objects

In this section, we give an example of devising an ontology for a particular, though abstract, domain—the domain of mobile objects, such as one that surveillance applications would be concerned about. This will illustrate some of the concepts discussed above and the approach to constructing an ontology for an application.

In surveillance applications, there will be many objects, but we are primarily interested in Mobile Objects; we will restrict these to objects that are capable, insofar as we can tell, of independent motion. People, animals, and cars are examples. There are also Portable Objects. These are objects that a mobile object can pick up and carry to another location, but they are not necessarily capable of moving of their own accord. A toddler is a portable object that is capable of independent motion; a book is a portable object that isn't.

Among the contextual objects are containers. By this we mean anything that something else can be inside or outside of. Rooms, trash containers, and fenced-in areas are all containers.

For each of the categories Object, Mobile Object, Container, and Portable Object, we list the properties they have and the relations they can participate in. By considering the changes in these properties and relations, we arrive at the primitive events that can happen to them

A Mobile Object has a velocity and a direction. The primitive events are:

Speed-Up	Change from a lower to a higher velocity
Slow-Down	Change from a higher to a lower velocity
Start	Speed-Up when the start velocity is 0
Stop	Slow-Down when the finish velocity is 0
Turn-Right	Clockwise change in direction
Turn-Left	Counter-clockwise change in directions

Two Objects can be in a "distance-from" relation. When at least one of those Objects is a Mobile Object, this relation can change. This gives rise to the following primitive events:

Move-Toward(A,B)	A moves and changes the distance from B to a smaller value
Move-Away-From(A,B)	A moves and changes the distance from B to a larger value

When the distance between *A* and *B* is 0, we can say *A* is "at" *B*. We can define a "move" event as a change of state from *A* being at *B* to *A* being at some other object *C*.

It may be useful to have a further notion of "Move-Directly-Toward." Someone walking obliquely past a doorway will have a period in which they are moving closer to it, but we might want to draw different inferences in this case and the case where the person moves directly toward it.

It may also be useful to have a notion of one object being "near" another object. This is generally a functional notion—near enough *for some purpose*. In a particular domain, it may be possible to define this precisely in terms of distance.

Containers have Portals that can be open or closed. A Mobile Object A and a Container B can be in one of two relations—inside(A,B) and outside(A,B). This gives rise to two primitive events:

Enter(A,B)	A changes from outside B to inside B.
Exit(A,B)	A changes from inside B to outside B.

For either of these events to happen, the Portal must be open, and *A* must go through (be at) the portal during the state change.

The two possible relations between Mobile Objects and Portable Objects are "hold" and its negation. The primitive events are

Pick-Up(A,B)	There is a change from A not holding B to A holding B.
Put-Down(A,B)	There is a change from A holding B to A not holding B.

In terms of these, we can define a "carry" event as consisting of

Sequence(Pick-Up(A,B), AND(Hold(A,B), Move(A,C,D)), Put-Down(A,B))

When a Portable Object is held by a Mobile Object, they are constrained to move together. A "take-from" action is a change from a portable object being held by one object to its being held by another. An "exchange" consists of two reciprocal take-from actions.

Events are possible involving two Mobile Objects, but these can be decomposed into primitive events we have already presented. For example, when two mobile entities approach each other, they are each moving toward the other.

5. Domain Ontologies

During the workshop participants used VERL to develop ontologies in the domains of physical security and meeting videos domains to highlight commonalities and differences.

5.1. Physical Security Domain

Monitoring events for physical security is an important application of video analysis. This domain also offers a good test of the ontology framework because events of interest are defined almost directly in terms of observable activities.

The researchers participating in this group naturally divided up into six subgroups, each exploring a different sub-domain, such as bank monitoring, outdoor surveillance, and railroad crossing monitoring. Initially, each sub-group defined its own set of objects, relations, primitive events, and complex events, which were combined to form complex events of interest to the users. Later, common elements were found between the sub-domains and used to construct the beginnings of a common ontology.

The definitions include more than 100 primitive and composite events. Clearly, more events could be defined and more sub-domains included. In particular, more events that require coordination between multiple actors over extended periods of time (perhaps days) were not considered. Nonetheless, we feel that the defined events helped validate the power of the representation framework and provide a set of definitions that should be usable by other researchers.

5.2. Meeting Domain

The domain of meeting and conference videos is considerably more complex than the security domain in the sense that events of interest take place at many levels of granularity, plus audio and text play important roles. To limit complexity, we decided not to represent the linguistic content.

Meetings can be analyzed at four levels:

1. The who, what, where, and when of the meeting, the sort of information that appears on a seminar announcement
2. The type of meeting, such as presentations and roundtable discussions
3. The "everyday rules of order" – how the contributions of individual members cooperate to perform group actions, e.g., bring up and dispense with topics, make decisions, and assign tasks
4. The communicative actions of individuals, such as utterances and gestures, and communicative complexes of small sets of individuals, such as "F-formations"

Our ontology describes an initial approach to levels 1, 2, and 4. Level 1 is fairly straightforward to characterize. Level 2 is a matter of explicit convention that members of our culture are familiar with, and thus it is also fairly straightforward to formalize. Level 4 is a very complex domain and is the locus of a great deal of good research. We have only been able to sketch some elementary aspects of this level. Level 3 is a difficult and little-studied area, and we have had to leave this for future work.

6. Markup Language (VEML)

We have defined a Video Event Markup Language (VEML) for representing specific instances of objects and events detected in video streams so that they can be exchanged between research groups and analysis techniques. VEML encodes such things as the name of an event, its type, the beginning and ending times of it, and the actors participating in it. This information is encoded in XML and written to a file in a standard way for use by other computer programs.

Figure 1 shows the relationship of VEML to VERL. VERL is essentially a programming language for describing generic events, such as sneaking in a door behind someone or stealing something, as compositions of other events. VEML encodes instances of objects and events detected in video data in XML. In particular, VEML is designed to encode five items for a set of events that have been automatically extracted or interactively annotated in a set of stream data:

- the ontology used (i.e., a pointer or file reference to the VERL definition)
- the data streams involved
- the context, such as the geometric structure of the local scene
- the objects, such as people and suitcases, that participate in the events
- the events themselves, such as approach, grasp, and sneak-in

The next section contains an example of a VEML file that describes a set of objects and events associated with a person entering a locked facility by tailgating another perons.

{Note that we allow multiple synchronized video data sets and audio tracks to be examined and annotated in parallel, although the majority of our initial examples involve only a single video stream.}

7. An Example of VERL and VEML

In this section, we present a sample set of VERL definitions that leads up to a description of a tailgating event, and then present a portion of a VEML file that encodes occurrences of these events detected in a video.

7.1. Sample VERL Definitions

```
// subtypes of entities     {a partial taxonomy of ents for
// this example}
SUBTYPE(person, ent)
SUBTYPE(facility, ent)
SUBTYPE(portal, ent)
SUBTYPE(door, portal)
SUBTYPE(window, portal)

// primitive properties of ents
// {Note:    if you were to specify mobile and container
// as subtypes of ent, you could not have mobile
// containers.}
PRIMITIVE(container(ent x))
PRIMITIVE(mobile(ent x))
PRIMITIVE(open(portal x))
PRIMITIVE(closed(portal x))
PRIMITIVE(locked(portal x))
PRIMITIVE(unlocked(portal x))
PRIMITIVE(portal-of(portal p, container c))
PRIMITIVE(inside-of(ent x, ent y))      // x is on the inside
    // of the container y
PRIMITIVE(near(ent x, ent y)      // (= close) x is within
    // some distance of y, where the distance is context
    // dependent.
PRIMITIVE(behind(ent x, ent y))  // x is on the same side
    // of y as the back of y.
    // Defined only when y has a "front" //and "back",
    // such as a person or vehicle.
PRIMITIVE(behind(ent x, ent y, point observer-position))
    // x is on the opposite side of y as observer-position
    // ("behind" the tree means that //the person is on the
    // other side of the tree rom the observer).

 // rules associated with ents and evs
RULE(IMPLY(person(x), mobile(x))    // people are
    // mobile
RULE(IMPLY(facility(x), container(x))      // all facilities
    // are containers
RULE(IMPLY(portal(p), AND(container(c),
                                portal-of(p, c))))
RULE(IMPLY(AND(portal-of(p, c), open(p)), open(c)))
// portal open => container

// processes describing relationships, events, etc.
PROCESS(far(ent x, ent y),    NOT(near(x, y)))
PROCESS(outside-of (ent x, ent y),
            NOT(inside-of (x, y)))

PROCESS(approach(ent x, ent y),
            cause(x, change(far(x,y), near(x, y))))
PROCESS(leave(ent x, ent y),
            cause(x, change(near(x, y), far(x, y))))
PROCESS(exit(ent x, ent y), change(inside-of(x,y)),
                    outside-of(x,y)))
PROCESS(enter(ent x, ent y), change(outside-of(x,y)),
                    inside-of(x,y)))
PROCESS(unlock(portal p), change(locked(p)),
                    unlocked(p)))
PROCESS(open(portal p), change(closed(p)), open(p)))
```

```
// definition of a TAILGATING event x is near y when
// they get access to a facility, & then x enters behind y
SINGLE-THREAD(tailgate(ent x, ent y, facility f),
                              AND (portal-of(door, f))
        Sequence(
                approach(y, door),
                unlock(y, door),
                open(y, door),
                AND(enter(y, f), near(x, y)),
                NOT(unlock(x, door)),
                enter(x, f)))
```

7.2. Portion of a VEML File Encoding Events Detected in a Specific Video

A VEML file, such as the one below, encodes the global information about the data and scene, and then describes the objects and events detected in the data. In this example, there are four objects of interest:

1. Person1, who unlocks the door and enters the facility legitimately
2. Person2, who enters by following Person1 through the door
3. Facility1, which is the locked container that the two people enter
4. Door1, which is the door through which the two people enter the facility

There may be several events detected and encoded in the file, but some of the key ones for this example are

1. Person1 approaches Door1
2. Person2 follows Person2
3. Person1 unlocks Door1
4. Person2 enters Door1 by tailgating Person1

```
<scene>

  <ontology>
  <source>…/ontologies/physicalSecurity
  .verl</source>
  </ontology>

  <streams>
    <video id="sneak02">
      <offset unit="frames">0</offset>
      <duration unit="frames">450
      </duration>
      <samplingRate>30</samplingRate>
      <source>/home/dvtt2/IU/video/
      data/sneak02/sneak02.sriv
      </source>
    </video>
  </streams>

  <context>
  <!-- To Be Determined -->
  </context>

  <objects>
    <object type="PERSON" id="OBJECT1">
      <property name="name" value=
      "Person1"/>
      <tracks></tracks>
    </object>
    <object type="PERSON" id="OBJECT2">
      <property name="name" value=
      "Person2"/>
      <tracks></tracks>
    </object>
    <object type="FACILITY" id=
      "OBJECT3">
      <property name="name" value=
      "Facility1"/>
      <tracks></tracks>
    </object>
    <object type="ENTRANCE" id=
      "OBJECT4">
      <property name="name" value=
      "Door1"/>
      <tracks></tracks>
    </object>
  </objects>

  <events>
    <event type="APPROACH" id="EVENT1">
      <begin unit="frames">136</begin>
      <end unit="frames">247</end>
      <property name="name" value=
      "Approach1"/>
      <argument argNum="1" value=
      "Person1"/>
      <argument argNum="2" value=
      "Door1"/>
    </event>
    <event type="FOLLOW" id="EVENT2">
      <begin unit="frames">177</begin>
      <end unit="frames">247</end>
      <property name="name" value=
      "Follow1"/>
      <argument argNum="1" value=
      "Person2"/>
      <argument argNum="2" value=
      "Person1"/>
    </event>
    <event type="UNLOCK" id="EVENT3">
      <begin unit="frames">260</begin>
      <end unit="frames">332</end>
      <property name="name" value=
      "Unlock1"/>
      <argument argNum="1" value=
      "Person1"/>
      <argument argNum="2" value=
      "Door1"/>
```

```
    </event>
    …
    <event type="TAILGATE" id=
      "EVENT12">
      <begin unit="frames">177</begin>
      <end unit="frames">508</end>
      <property name="name" value=
      "Tailgate1"/>
      <argument argNum="1" value=
      "Person1"/>
      <argument argNum="2" value=
      "Person2"/>
      <argument argNum="3" value=
      "Facility1"/>
    </event>
  </events>

</scene>
```

8. Summary

We believe that we have made significant progress in defining an event ontology framework, a formal representation language, and specific ontologies for the security and meeting domains. We believe that use of such ontologies will greatly help in advancing research in event recognition from videos by allowing researchers to share their results in a unified framework. In addition, VEML can help in exchanging annotated videos and in evaluating the results of automated analysis as outlined in our evaluation document. Another important benefit of the whole challenge project has been in bringing large segments of the research community together and forming a consensus on foundational issues.

In spite of the progress made, much remains to be done. For example, the framework needs to be exercised on much more complex events and the ontologies for specific domains need to be expanded significantly. We have not addressed ontologies for domains such as broadcast news video where the content tends to be largely unrestricted. For videos containing speech or text, integration with linguistic ontologies also needs to be explored.

The event ontologies need to be embedded in a standard knowledge representation framework, such as OWL or Semantic Web Rule Language (SWRL), to exploit the inference engines available in that community. Plus, tools need to be developed to efficiently annotate videos and check their consistency with an underlying ontology. We hope that the broader research community will take interest in exploring these issues.

Acknowledgments

This research was funded by the Advanced Research and Development Activity (ARDA) of the U.S. Government under a contract to the Pacific National Northwest Laboratory of the Department of Energy.

The paper describes contributions of many participants. Rich Quadrel provided the project management and Mark Maybury provided general project guidance. Penny Lehtola and John Prange were the principal sponsors.

Jerry Hobbs, Brian Burns, Sadiye Guler, Francis Quek, Isaac Cohen, Chris Connolly, and Alex Francois were group leaders or co-leaders.

Other participants included: Terry Adams, Dan Aldridge, Umut Atdemir, Aaron Bobick, Rachel Bowers, Francois Bremond, Rama Chellappa, Larry Davis, Peter Doucette, Jon Fiscus, , John Garofolo, Wayne Greiff, Ismail Haritaoglu, Alex Hauptmann, Ryan Hohimer, Ramesh Jain, Ranga Kasturi, Chris Laprun, Steve Long, Inderjeet Mani, Alvin Martin, Paul Matthews, Andy Merlino, Martial Michel, Joe Mundy, Srini Narayanan, David Palmer, Randy Paul, Bruce Porter, Mubarak Shah, Tracy Standafer and Monique Thonnat

References

[1] Allen, James F., and George Ferguson, 1997. "Actions and Events in Interval Temporal Logic," in O. Stock (ed.), {\it Spatial and Temporal Reasoning}, Kluwer Academic Publishers, Dordrecht, Netherlands, pp. 205-245.

[2] Ankolekar, A., Burstein, M., Hobbs, J., Lassila, O., Martin, D., McIllraith, S., Narayanan, S., Paolucci, M., Payne, T., Sycara, K., Zeng, H., 2002. DAML-S: Web Service Description for the Semantic Web, International Semantic Web Conference (ISWC) Sardinia, June 2002.

[3] Bettini, Claudio, X. Sean Wang, and Sushil Jajodia, 2002. "Solving Multi-granularity Temporal Constraint Networks," *Artificial Intelligence*, Vol. 140, pp. 107-152.

[4] Hobbs, Jerry, 2002. "A DAML Ontology of Time," http://www.cs.rochester.edu/~ferguson/daml/daml-time-29jul02.txt.

[5] Y.A. Ivanov, A.F. Bobick, "Recognition of Visual Activities and Interactions by Stochastic Parsing," *IEEE Trans. on PAMI*, no. 8, pp. 852-872, August 2000.

[6] S. Narayanan (1997) KARMA: Knowledge-based Action Representations for Metaphor and Aspect, Ph.D. dissertation, University of California, Berkeley, California.

[7] R. Nevatia, T. Zhao and S. Hongeng, "Hierarchical Language-based Representation of Events in Video Streams," Proceedings of the Workshop on Event Mining (in conjuction with IEEE CVPR), Madison, WI, June 2003.

[8] T. Vu, F. Brémond and M. Thonnat, "Automatic Video Interpretation: A Novel Algorithm for Temporal Scenario Recognition", The Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, 9-15 August 2003.

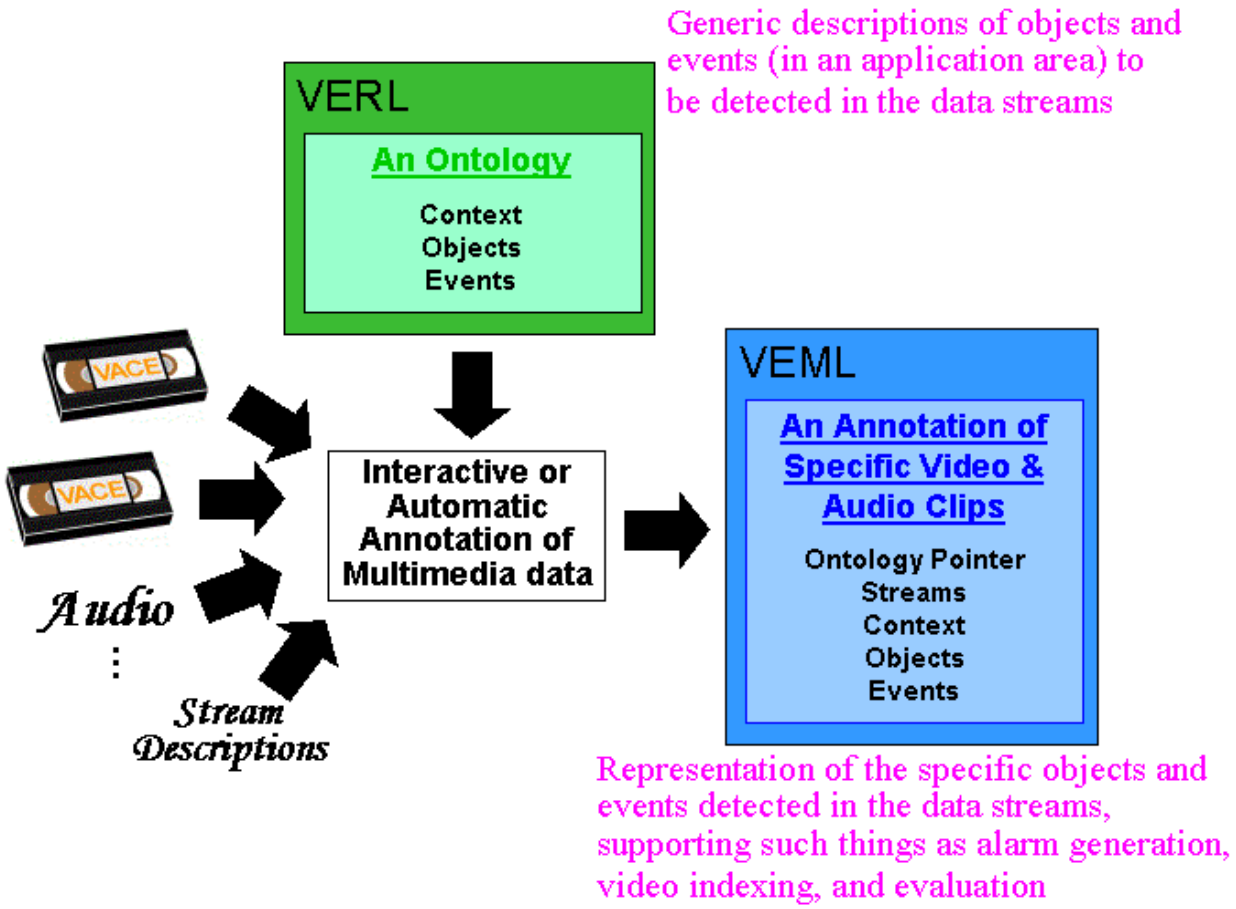

Figure 1. *A diagram of the relationship between VERL and VEML.*