

Extending Generic BPM with Computer Vision Capabilities

Adrian Mos^(✉), Adrien Gaidon, and Eleonora Vig

Xerox Research, 6 Chemin de Maupertuis, Meylan 38240, France
{adrian.mos, adrien.gaidon,
eleonora.vig}@xrce.xerox.com

Abstract. Leveraging Business Process Management (BPM) is key to enabling business agility in organizations. Video analysis is a nascent technology that allows for innovative sensing and understanding of a large family of tasks across many diverse application domains. This includes interactions between persons, objects, and the environment in domains such as Healthcare and Retail, as well as more general activities (e.g., in video-surveillance or Transportation). It can, therefore, enable better BPM by giving the opportunity to augment, complement, and improve the observation, description, monitoring, triggering, and execution of a broad array of tasks, including new ones that can only be described visually. This may be of particular interests in cyber-physical systems where interactions between human agents and artificial agents can be tracked and managed in the context of various business processes. This paper proposes a way to integrate video data and analysis into the control flow of business processes, in a way that enables the seamless augmentation of business process execution with information from the observable environment.

Keywords: BPM · Modelling · Computer vision · BPMN

1 Introduction and Overview

Cameras are now ubiquitous in public spaces, industrial sites, and workspaces (e.g., warehouses). However, their capabilities are currently exploited through human vision only. This results in an underuse of the information provided by cameras, as they cannot be easily integrated into existing BPM suites. We propose to leverage **Computer Vision (CV)** techniques in order to *automatically transform visual data into actionable data for BPM*.

Our approach allows modelling the vocabulary (visual concepts) and grammar (interactions) needed to **represent complex processes by composition**. This allows the re-use of component models (e.g., pedestrian detectors), composition rules (e.g., spatio-temporal relations), and training data across domains. This also enables business agility, as new business processes (BPs) can be easily modelled by reusing visual components of other processes.

We also propose a specific model management approach to bridge the gap between the yet unconnected worlds of video analysis and BPM. This allows business experts to leverage CV technology for BPM, without the need for a CV expert or laborious

manual description. We also investigate the impact of the **uncertainty associated to video-based observations** in BPM.

To the best of our knowledge, integrating automatic video analysis in BPM platforms is a novel concept with a broad interest. In today's state-of-the-art BPM environments, if one wanted to add computer vision capabilities, one would need to manually create all the connecting glue to integrate with some computer vision software and most importantly manually collect CV training sets for the *specific events* of interest in a *specific context*. For instance, a parking management business process that includes visual occupancy monitoring would require: (i) the collection of video data from each camera in each specific parking lot, (ii) a manual video labelling stage in order to build a per-camera (and per-parking lot) training dataset, (iii) a per-dataset learning phase to get per-camera individual visual models (*e.g.*, by relying on scene-specific background subtraction). This type of effort would then be repeated each time new computer vision support is needed (for instance to monitor different visual events), and in each context (*e.g.*, on-street vs. off-street parking). This approach is not scalable, neither in terms of cost (acquiring and labelling data is expensive and infeasible for each additional modelled event), nor in terms of usability (due to the re-designing from scratch of every new event model).

Our approach is different in that it entails a modular CV engine where various patterns are individually described and can be composed *on-the-fly for any (BP)*. To do so, we propose composition mechanisms accessible through APIs that are automatically employed by specific BPM elements. While some off-line pre-training still needs to be performed in order to enable the detection of *generic patterns*, the modularity of our method allows for the reuse and combination of any number of patterns to model arbitrary events. Furthermore, our approach significantly simplifies the usage in BPM, by reducing it to selecting CV elements and making simple parameter choices in their properties. In the previous example, our approach would rely on generic pre-trained car detectors (*e.g.*, on the KITTI dataset [1]), reusing them across cameras and parking lots (or other contexts), in order to meet different business process needs (*e.g.*, occupancy measurement, entry-exit monitoring, or even different applications such as lane enforcement). Similarly, this generic pattern detection approach could be applied to a healthcare cyber-physical system where we might have human agents such as patients, nurses and doctors in explicit and implicit interactions with artificial agents such as video cameras, personnel and patient badges or medical monitoring equipment. Computer vision elements could then be used to detect and track such agents and interaction patterns. Their detection could be based on pre-trained atomic elements such as person detectors and then parameterised as part of their usage with information such as uniform colour, badge detection and others. Such CV elements could make first class citizens in processes such as reactive optimization of patient treatment depending on personnel and equipment availability, patient arrivals and other observable properties of the environment. Other types of processes that could benefit from such elements include room preparation and allocation, equipment management (to ensure availability at each floor for instance) or tool preparation and regulatory compliance in operating rooms.

2 Architectural Details

We propose a practical system composed of several components and interface mechanisms for bridging BPM and CV listed below and illustrated in Fig. 1.

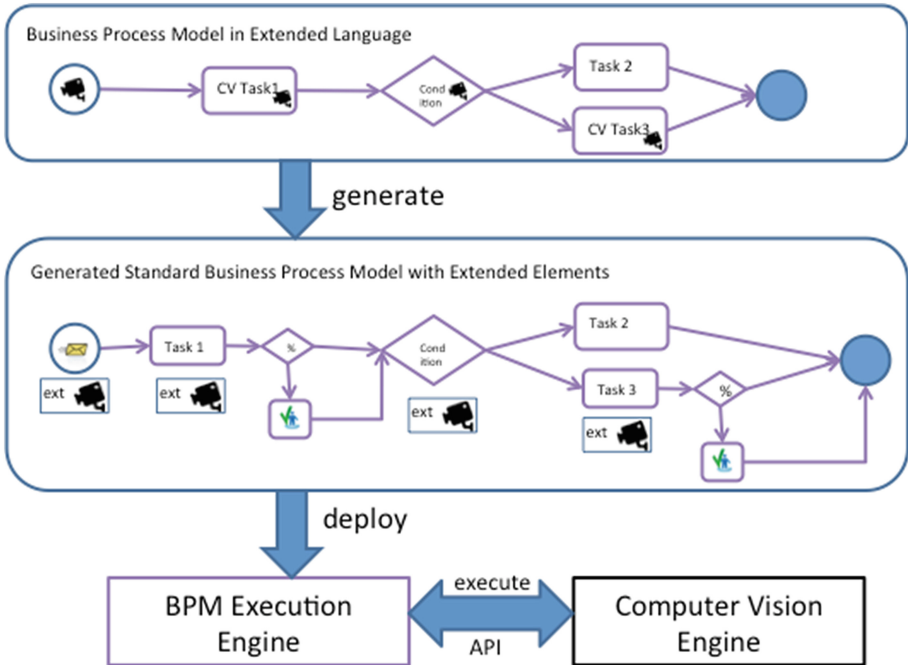


Fig. 1. Overall architecture

- A modelling environment for designing business processes that natively interface with CV capabilities (**VEPE: Vision Enabled Process Environment**). This comprises specific language extensions and BPM-type modelling support for bringing CV capabilities into business process design.
- A **generation component (GEM)** that takes process designs created in the VEPE and creates plain executable business process design models in a language understood by commercial off the shelf (COTS) BPM Suites. We target BPMN 2.0 here, as it is the most common standard BP design language. These generated models may include extensions added to the plain language elements using extensibility mechanisms of the standard process languages (BPMN 2 provides extension points for its elements).
- An enhanced **BPM engine (BPME)** for interfacing with CV capabilities at runtime when BPs execute. This engine will be an extension of COTS BPM engines that supports any extensions to the plain BP design models generated by the GEM.

- A **CV Engine (CVE)** using a modular approach that employs the expressivity and modularity required for interfacing with BPM elements. The CVE provides an API that the BPME uses to leverage the CV capabilities.

The process designer would create a BP diagram in the VEPE, which will have palettes of standard BP elements as well as CV-enabled elements. In Fig. 1 the starting element is a vision-event-based start, with some of the tasks as well as the gateway also vision-based. Task 2 is the only one not vision-based in this example, to illustrate the mix of plain BP elements with CV elements. The system would use the GEM to first translate this language into standard BPMN that is enriched with extensions that pertain to the appropriate CV extensions. These extensions are markers for API usage in the engine, with properties attached that correspond to API parameters. This generated model would also contain specific patterns automatically added in order to accommodate the uncertainty of the various CV events (in this example, a simple gateway checking for the confidence level and deciding whether to involve a human validation step or not). Lastly the generated process is deployed onto the BPME engine, which would employ the CVE for all tasks that require CV functionality by interpreting the markers and thus translating process semantics into CV operations.

2.1 VEPE and Language Extensions

In typical BPM solutions, once BPs are designed and configured with enterprise resources (people, roles, services etc.) they are executed and monitored by a BPM engine. The VEPE modelling environment could be a stand-alone process editor or it could be built on top of existing modelling environments such as the open source Eclipse BPMN 2.0 Modeller [2] or indeed any other graphical process editor based on Eclipse Modelling Framework (EMF) [3]. This includes a large variety of industry leading business process studios from IBM, Tibco, SAP, Oracle and others. If VEPE were designed as a stand-alone editor, it would have just enough basic process design functionality to cover the structural elements of normal process design and the needs of CV-based processes. The assumption with this approach would be that most of the business functionality that is not CV centric would then be enriched in a standard BPM editor at a later stage, after the GEM generation of the BPMN.

If VEPE is designed as an extra layer on top of existing process environments, it can add specific support for designing the CV processes in the form of additional dedicated tool palettes containing the vision elements, property sheets and configuration panels for the specification of the various parameters required by the CV elements, as well as any other graphical support necessary to highlight and differentiate such elements from standard BPM elements. Additionally, a specific decorator for CV could be enabled which, when applied to any supported BP elements would transform it into a CV element (e.g. by dragging and dropping a camera icon onto a BP task). In this case, the GEM would either run in the background, constantly translating the vision elements into BPM elements or would run when the model is saved for instance or any

other chosen moments (similar to how a spell-checker could either be used as you type or when you specifically choose to execute it).

Language extensions are required in order to support the definition of BPs that use CV capabilities. The GEM would use them in the generation phase. Creating such language extensions may imply definition of new elements or extending and customizing existing elements. We consider BPMN 2.0 our main target in terms of extensions, as it is by far the most widely used BP design language, supported by most industry-leading solutions. It has basic extension capabilities that allow the enrichment of standard elements with a variety of options. Where such extension capabilities do not suffice, new elements can be introduced. The additional elements and the extensions to the existing elements need to be supported by the BPME.

2.2 BPME

The contribution presented in this paper does not aim to replace existing BPM engines but rather to leverage them through strategic extensions. For open source engines such as Stardust [4], Bonita [5] and others, extensions are relatively straightforward to implement as the code is readily available. For proprietary solutions, extensions can be added through special agreements with the vendors or indeed through specific APIs. In fact it is possible that any extensions required for CV be implemented as specific types of BPMN Service Tasks prefilled with appropriate data.

Adding the extensions would involve adding connectivity to the CV Engine using its APIs in one of two ways: The first option involves natively supporting them, meaning they would be first class elements of the engine (this would only be necessary for extensions that the GEM cannot map to standard BPMN). A simple option for most of the extensions would involve the intermediate transformation operation in which the BPs expressed with language extensions would first be converted by the GEM to a more classical form of BP descriptions before being executed by the engine. For instance in this case the CV Task would first be converted into a normal BPMN Task with prefilled specific parameters and the engine would execute it as a normal Task, by simply calling a web service that would be provided by the CV Engine. This can be achieved through a BPMN Service Task prefilled with web service information that points to a specific web service that acts as a façade to the CVE. More likely, a combination of the two options would be used, where a GEM transformation will generate an almost-classical BP process and the BPM engine will have minimal extensions to support very specific CV operations.

2.3 CVE and Uncertainty

The Computer Vision Engine provides the execution support for CV-enabled BPs. Its functionality can be divided in three main parts: native support for a video domain-specific language (VDSL) by allowing the specification of composite actions, patterns and associated queries using the elements specified in the VDSL definition; call-back

functionality for triggering actions in a BP when certain events are detected; allowing the specification of rules in the observed scenes (to be then used for detecting conformance, compliance, and raising alerts).

A specific challenge of integrating CV capabilities in BPM relates to the inherent uncertainty that CV algorithms entail. Detecting an event in a video stream comes with an associated confidence level so the BPs need to account for that. For instance if the system is 99 % certain that a certain event was detected, we can consider that the risks that this is wrong are minimal and therefore the process can assume it's true. For a lower value (e.g. 80 %), the BP might need to have additional logic in order to deal with the lack of certainty, for instance by executing additional tasks by a human supervisor to double-check the data. Such handling may even depend on the nature of the task (for instance critical tasks may need a higher degree of certainty than others and in some cases a human may need to be briefly involved all the time for targeted double-checking of certain visual sequences). For the purpose of this contribution we take a simplified approach and provide a threshold-based pattern where humans only need to be involved in cases of lower confidence of video detection operations. Other more complex patterns can be used and added to the system as plugins to the GEM that will drive the specific generation of the confidence-handling elements.

3 Language Extensions

As described in Sect. 2.1, a number of new elements specific to CV need to be made available to process designers using the VEPE.

These elements are either available in a palette or they can be generated by adding the CV decorator (camera icon) onto existing standard BPMN elements. Regardless of how they are created, these elements have embedded CV semantics that will ensure their correct execution by the BPME using the CVE. Figure 2 lists a number of such elements showing the CV element, its icon, the BPMN counterpart and API usage indicator, pointing to the possible API that could be leveraged to achieve the required functionality. More elements could naturally be made available however in this contribution we do not aim to provide an exhaustive list of all possible implementations, rather to show that this approach can be achieved using existing technologies and standards. For most of the elements, the BPMN counterpart is an element as well, however the CV Task mapping to a complex pattern illustrates the variety of mappings that can be used. Once the mappings are decided, they need to be stored in a way that can be leveraged by the GEM, as the GEM will ultimately use this information when generating BPMN from CV process designs.

It is important to note that the GEM functionality can be achieved using existing technologies such as EMF approaches for model to model transformations. Even elements that are not specifically created as extensions for CV process design can still benefit from CV input. For instance, role assignment to a process or a task could benefit from compliance checking if the CVE has the capability to discern between various subjects in a scene.



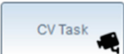
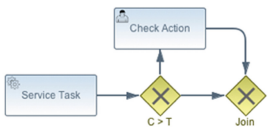






| CV Element | BPMN Counterpart | CVE API Usage |
|---|---|---|
| CV Start  CV Start | Start Event  Start Event | <pre>createEvent(Context ctx) addEventComponent(...) startEventMonitor(Event e)</pre> |
| CV Task  CV Task | Complex Pattern  Service Task, Check Action, C > T, Join | <pre>createEvent(Context ctx) addEventComponent(...) startEventMonitor(Event e)</pre> |
| CV Gateway  CV Gateway | Event Gateway  Event Based Gateway | Callback from CVE |
| CV Catch Event  CV Catch Event | Catch Event  Intermediate Catch Event | Callback from CVE |
| CV Data Object  CV Data Object | Data Object  Data Object | <pre>getPatterns(Context ctx, PatternFilter[] pfs)</pre> |

Fig. 2. Language extensions

4 Computer Vision Engine (CVE)

After generating a vision-extended business process model, a BPM execution engine can leverage the visual information available from cameras by querying a Computer Vision Engine (CVE). We first describe the internal components of the CVE, based on a Video Domain-Specific Language. Consequently, we provide the external API of the CVE used to formulate queries from the BPM execution engine.

4.1 Video Domain-Specific Language (VDSL) Elements

The diagram in Fig. 3 illustrates the different elements of our proposed VDSL. It organizes visual concepts in different entity (the *vocabulary*) and relation categories (the *grammar*) depending on their functional roles, their semantic, and their dependency on data. The first category pertains to **relations** applicable to visual concepts. *Geometrical transforms* (such as translation, scaling, rotation...) can be measured via image matching and optical flow estimation techniques. They are needed to describe the motion of objects and to reason about the dynamics of a scene. *Pairwise spatio-temporal relations* describe spatio-temporal arrangements between visual entities. They are required to reason about interactions, but also to represent the relative evolution of the different components of a process. Note that these generic rules (geometrical and spatio-temporal) are formalized *a priori*. Finally, *similarities* are measures of visual relatedness according to different predefined visual features (*e.g.*, colours).

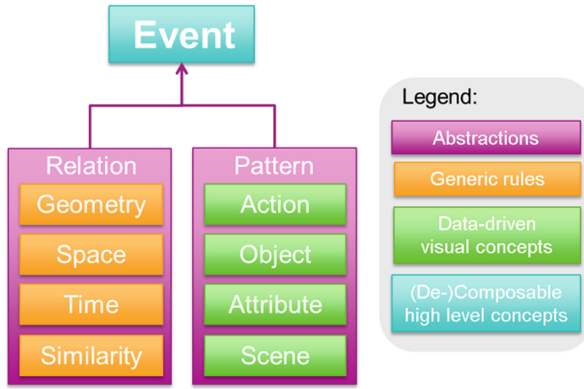


Fig. 3. Video domain specific language elements

The second set of VDSL elements we define are **patterns**. This abstraction unifies low-level visual concepts such as actions, objects, attributes, and scenes, which are *visually detectable*. *Actions* are the verbs in our VDSL, and are modelled using motion features. *Objects* are nouns modelled by appearance (*e.g.*, via edges, shape, parts...). Note that although this includes “persons”, this important category of objects requires separate treatment in the internals of the CVE. *Attributes* are adjectives corresponding to visual properties (such as colour or texture). *Scenes* are semantic location units (*e.g.*, sky, road...). Patterns are data-driven concepts: their underlying visual detectors need to be trained on visual examples. Note that some patterns are generic (*e.g.*, persons, vehicles, colours, atomic motions...), and can therefore be pre-trained in a generic fashion in order to allow for immediate re-use across a variety of domains. Domain-specific patterns need, however, to be defined *by user-provided examples*. The number of such examples can be fairly low in practice, depending on the specificity of the pattern (*e.g.*, as low as one for near-duplicate detection via template matching) and on the availability of weakly or unlabelled data (*e.g.*, when using semi-supervised learning approaches). The third

element in our VDSL is **events**. They formalize the high-level visual phenomena that the CVE is asked to observe and report about according to a query from the BPME. In contrast to patterns, models of events cannot be cost-effectively learned from user-provided examples. Events are indeed defined *at runtime* by particular *query* from the BPME. Therefore, we propose to *handle both the specificity and complexity of queried events by composition*. We detail in the following our approach to *modelling at query time an event* from the aforementioned elements of our VDSL.

4.2 CVE API

The BPME formulates queries to the CVE using the API summarised below.

A **Context** object holds a specific configuration of a set of cameras and their related information. It also incorporates constraints via a set of constraints from the BPME (*e.g.*, to interact with signals from other parts of the BP). The API allows filtering of the video streams processed (for instance to limit cameras to certain locations).

```
Context = {CameraInfo[] cis, BPConstraints[] bpls}
Context getContext(CameraFilter[] cfs, BPConstraints[]
bpcs)
```

Pattern objects are *detectable* visual entities. They are accessible via:

```
PatternType = Enum{Action, Object, Attribute, Scene}
Pattern = {PatternType pt, SpatioTemporalExtent ste, Confi-
dence c}
Pattern[] getPatterns(Context ctx, PatternFilter[] pfs)
```

The available patterns (actions, objects, attributes, and scenes) are the ones for which the CVE has a pre-trained detector. Note that these detectors might be coupled in practice (multi-label detectors) in order to mutualize the cost of search for related patterns (*e.g.*, objects that rely on the same underlying visual features). The pattern filter and context arguments allow searches for patterns verifying certain conditions.

Relations describe the interactions between two patterns:

```
RelationType = Enum{Geometry, Space, Time, Similarity}
Relation = {RelationType rt, RelationParameter[] rps, Con-
fidence c}
Relation[] getRelations(Pattern p1, Pattern p2)
```

The Geometry, Space, Time, and Similarity relation types correspond respectively to a list of predetermined geometrical transformations (*e.g.*, translation, rotation, affine, homography), spatial relations (above, below, next to, left to...), temporal relations (as defined in Allen's temporal logic), and visual similarities (*e.g.*, according to different pre-determined features) [6]. Note that these relations are defined a priori with fixed parametric forms. Their parameters can be estimated directly from the information of two patterns.

Events allow to hierarchically compose patterns and relations in order to create arbitrarily complex models of visual phenomena, *e.g.*, groups of patterns with complex

time evolving structures. Events are internally represented as directed multi-graphs where nodes are patterns and directed edges are relations. Two nodes can have multiple edges, for instance both a temporal and a spatial relation. Events are initialized from a context, and built incrementally by adding relations between patterns.

```

Event createEvent (Context ctx)
Event addEventComponent (
    Event e, Pattern p1, int id1, Pattern p2, int id2, Relation[]
rs)
    (add two pattern nodes with specified IDs and relations to the
event)
    CallbackStatus startEventMonitor(Event e)
    (instruct the CVE to send notifications each time an event
happens)
    stopEventMonitor(Event e)

```

5 Related Work

To the best of our knowledge, while there is a plethora of work around data-centric BPM [7] and context-aware processes (e.g. [8]), there has been no systematic approach to integrating BPM with CV. This is very likely due to the major challenges in BPM itself that have steadily been tackled by the community over the past few years and the typical approach taken in CV by tackling each new problem with one application.

In CV, event detection has been investigated extensively and multiple research directions to event modelling emerged [9]. Traditionally, events have been represented and recognized **holistically**. This involved the global classification of (pre-segmented) video clips [10]. However, understanding events that involve complex activities (e.g., human interactions, sports, street scenes, etc.) requires reasoning about the spatio-temporal “structure” of the event, including interaction between agents and objects. To enable such reasoning, **part-based methods** were proposed. These methods typically define an event in terms of *event primitives* and their space-time *relations*. Two different mechanisms to discover these primitives are common in the literature. On the one hand, primitives are often pre-defined by hand and their space-time relationships are either learned or manually configured [11, 12]. Other approaches go further and learn the event primitives from data [13, 14]. The third and most recent line of research represents events using **attributes** by making use of libraries of reusable high-level semantic concepts [15]. Continuing this line of research, current trends are towards leveraging knowledge bases for high-level reasoning [16]. Existing methods to event detection learn *event-specific representations* from a vast pool of training data. Hence, the availability of training data is critical for these approaches and is often the major bottleneck to their generalization capability. In contrast, here we *reuse generic visual elements* from our VDSL across events by interfacing with a BPM execution engine.

6 Summary and Conclusion

This paper presents an approach to integrate computer vision capabilities in BPM in a modular, composable and generic way. It proposes a Vision Enabled Process Environment that allows new CV elements to be added to the BPM standard languages as well as graphical modelling tools for designing with CV elements. The approach also entails a model generation system for attaching these CV entities into BPMN 2 or other process description and execution languages. Importantly, the user *does not need to program the connection between the BPMS and CV*, instead, they simply select to use CV-enabled elements in their BPs, the connections being generated. The execution support is provided by an extended BPM engine that executes the generated processes. This support can be readily applied to any BPMS that uses an extensible standard such as BPMN, or to any engine that has extension points. It can also be added to BPM engines through partnerships with the respective vendors. Lastly, the CV engine supports the VDSL and provides the API used by the engine to leverage the visual information available from cameras *in a modular way*.

We have started the implementation of this approach using a mix of domain-specific modelling environments, open source BPMS and computer vision libraries developed in house. We believe that this work has great potential to bring added value to enterprise business processes in a cost-effective way by bridging process design and execution with visual information through modular computer vision capabilities.

References

1. Geiger, A., Lenz, P., Urtasun, R.: Are we ready for autonomous driving? KITTI Vision Benchmark Suite. In: Conference on Computer Vision and Pattern Recognition (CVPR) (2012)
2. BPMN2 Modeler. <https://www.eclipse.org/bpmn2-modeler/>
3. Budinsky, F., Brodsky, S.A., Merks, E.: Eclipse Modeling Framework. Pearson, London (2003)
4. Stardust BPMS. <http://www.eclipse.org/stardust/>
5. Bonita BPM. <http://www.bonitasoft.com/>
6. Szeliski, S.: Computer Vision: Algorithms and Applications. Pearson, New Jersey (2010)
7. Cohn, D., Hull, R.: Business artifacts: a data-centric approach to modeling business operations and processes. Bull. IEEE CS TC Data Eng. **32**(3), 3–9 (2009)
8. Moon, J., Kim, D.: Context-aware business process management for personalized healthcare services. In: IEEE SCC (2013)
9. Weinland, D., Ronfard, R., Boyer, E.: A survey of vision-based methods for action representation, segmentation and recognition. CVIU **115**, 224–241 (2011)
10. Wang, H., Kläser, A., Schmid, C., Liu, C.: Dense trajectories and motion boundary descriptors for action recognition. IJCV **103**, 60–79 (2013)
11. Gaidon, A., Harchaoui, Z., Schmid, C.: Activity representation with motion hierarchies. IJCV **107**, 219–238 (2014)
12. Sadanand, S., Corso, J.: A high-level representation of activity in video. In: CVPR (2012)
13. Pirsiavash, H., et al.: Parsing videos of actions with segmental grammars. In: CVPR (2014)

14. Tang, K., Fei-Fei, L., Koller, D.: Learning latent temporal structure for complex event detection. In: CVPR (2012)
15. Liu, J., Kuipers, B., Savarese, S.: Recognizing human actions by attributes. In: CVPR (2011)
16. Deng, J., et al.: Large-scale object classification using label relation graphs. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) ECCV 2014, Part I. LNCS, vol. 8689, pp. 48–64. Springer, Heidelberg (2014)