Audio Engineering Society
Convention Paper

Presented at the 110th Convention
2001 May 12–15 Amsterdam, The Netherlands

Means of Integrating Audio Content Analysis Algorithms

Anssi Klapuri
Signal Processing Laboratory, Tampere University of Technology
P.O.Box 553, FIN-33101 Tampere, Finland

ABSTRACT

Two generic mechanisms are proposed that facilitate the efficient integration of audio content analysis algorithms. The first mechanism, priority-rule based interleaving of algorithms, allows the simultaneous interoperation of several bottom-up analysis modules by interleaving their atomic steps. It aims at increased accuracy through controlled manipulation of common data. The second mechanism, top-down routing of requests for data, allows high-level predictions to direct the bottom-up analysis towards verifying the predicted hypotheses by observations. Examples from automatic music transcription are presented to clarify the use of the proposed methods.

INTRODUCTION

Computational analysis of audio signals has several important applications. Managing the increasing amount of multimedia information calls for efficient content-based indexing and retrieval tools. Bringing computers to the real world requires that they are equipped with perceptual abilities, including machine hearing. Also, content information allows flexible and more advanced processing of audio signals.

Essentially, audio content analysis means modeling the functions of the auditory perception. This is a complicated and many-faceted process, involving the use of both acoustic data and stored knowledge [1]. *Data-driven* (bottom-up) analysis techniques are characterized by a bottom-up flow of information: observations from an acoustic waveform are combined into meaningful features and passed on to higher levels for further analysis. *Prediction-driven* (top-down) processing utilizes internal, high-level models of the content of the acoustic signals and prior knowledge of the properties and dependencies of the objects in it [2]. Together, these attempt to resolve and recognize distinct sound sources and events in complex input signals.

Meaningful integration and interoperation of the different processing principles has turned out to be one of the most difficult goals to achieve. Also, whereas the understanding in this area is constantly increasing, incremental design would be desirable, adding new analysis principles when they are discovered. Other requirements include:

- Analysis algorithms and data types of very different kinds can be integrated to the system.
- After being encapsulated to the architecture, the individual algorithms collaborate and compete without explicit reference to, or knowledge of, each other.
- The architecture should make it relatively easy to add and remove processing modules

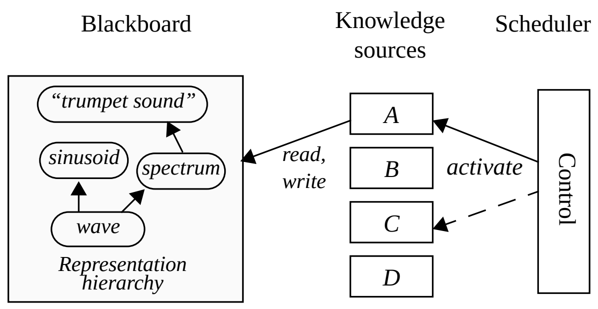

Fig. 1. Overview of the blackboard architecture.

- System should be able to handle uncertain data, and let several alternative explanations evolve side-by-side.

In this paper, a so-called blackboard architecture was selected to be the implementational starting point. Blackboard systems, originally developed in the field of artificial intelligence, meet several of the architectural needs that arise in audio content analysis. They are able to handle problems that require the integration of diverse types of information and knowledge sources [3,4,5]. However, the blackboard paradigm as such merely structurizes the problem, without specifying the exact data representations or the control flow.

In the following sections, an implementation of the blackboard architecture is first described. Then the mentioned two integration mechanisms are presented in this framework. Effort was taken to make the design as modular as possible, and to specify the standard classes and interfaces in such a flexible manner that they are able to accommodate other classical reasoning methods, too, particularly Bayesian belief networks [6], and the different blackboard control structures proposed in the literature [7].

BLACKBOARD ARCHITECTURES

The name blackboard refers to the metaphor of a group of experts working around a physical blackboard to solve a problem. Each expert can see the solution evolving and makes additions to the blackboard when requested to do so.

A blackboard architecture is composed of three components, which are illustrated in Figure 1. *Blackboard* is a hierarchical network of hypotheses, with the input data at the lowest level and analysis results on the higher levels. Hypotheses have relationships and dependencies on each other. Blackboard is often also viewed as a data representation hierarchy, since hypotheses encode data at varying abstraction levels. Intelligence of the system is coded into *knowledge sources* (KS), which are the processing algorithms that may manipulate the contents of the blackboard. A third component, *scheduler*, decides which knowledge source is in turn to take its actions. Since the state of analysis is completely encoded in the blackboard hypotheses, it is relatively easy to add new KSs to extend a system.

ENCAPSULATION AND INTERFACES OF THE DATA AND ALGORITHMS

In this section, the designed instance of the blackboard architecture is explained in more detail. Certain technical solutions were needed to fulfill the fundamental requirements for the architecture, and to realize an efficient control structure. Implementation took place in the Matlab environment.

Hypothesis network

Hypothesis hierarchy, the blackboard, can be understood as a *graph* where hypotheses are the nodes and dependencies between them form the links. Hypotheses have their probability ratings and they carry data at varying abstraction levels, a *representation hierarchy.* Other attributes include e.g. the time interval during which the hypothesis holds. Different types of directional and non-directional links can be specified to represent different kinds of dependencies and relations between the hypotheses. The architecture itself does not limit these types, or the topology of the graph.

Hypotheses

Co-operation between KSs happens mainly through the blackboard, where they place their analysis results. It is therefore necessary to define carefully the hypothesis and link classes to enable an efficient integration. Hypothesis class has the following attributes:

- *Hypothesis type*. Determines the type of the data representation that the hypothesis carries (waveform, frequency component, symbolic data, etc.), and implies the abstraction level at which the hypothesis belongs.
- *Unique identifier* enables reference to this hypothesis.
- *Real world location*. Time interval during which the hypothesis holds. Intervals of the other (spacial) dimensions may be added.
- *List of links* to other hypotheses. Defines the logical 'place' and dependencies of the hypothesis.
- *Rating*. Probability of the hypothesis to be true. The combined overall rating is followed by a list of knowledge sources and their given ratings, if there are several of them.
- *Remarks*. List of hypothesis related comment items that KSs may add, modify, or remove.
- *Representation data* that is carried by the hypothesis. Hypothesis type identifies also the representation type.

Many of these fields are used by the scheduler to *focus* processing. For example, the scheduler may constrain a KS to work on a certain hypothesis (unique identifier field) or in a specified time interval (time field) to force causal operation. Focusing takes place through an interface which has to be provided by each KS. Time is not quantized into frames or processing scopes. Although spacial dimensions are not used in the example application, they are likely to be needed e.g. in 3D-sound and in microphone arrays.

The *remarks*-field allows the specification of simplistic languages that KSs may use to communicate with each other, or to suggest changes to the scheduler's flow of control. For example, in a discrepancy directed processing, a KS may detect a discrepancy, such as a signal having an incomplete interpretation, and mark it to the appropriate hypothesis [8]. Also failed processing efforts may be logged, in order that they will not be repeated later on.

Rating of the probabilities of the hypotheses is necessary when dealing with uncertain knowledge. All KSs that process a hypothesis during its lifetime must leave their *rating*, and an estimate of the *accuracy* of the rating in this field. This gives the possibility to develop a dedicated KS whose task is to combine the 'opinions' of different KSs. In the case that a KS is not able to estimate its own accuracy, a default value is used. In the case that KS is not able to rate the probability of a hypotheses, a zero rating accuracy is used.

Representation data carried by the hypothesis may be numeric, such as a discrete signal, or symbolic, such as the parameters of a sound source. Calculation parameters are encapsulated along with the data itself, in order to enable recalculation with different parameters in the case that the signal analysis is not sufficient. This is one of the sustaining ideas in the IPUS architecture [8,9]. Inner structure of the data and parameters are specific for each representation, being interpreted by the KSs that are designed to manipulate them.

Links

Hypotheses are dependent on each other. The ones at the higher abstraction levels are often based on the belief in the lower level hypotheses. In the same manner, a lower level hypothesis may be dependent on the reliability of a higher level world model that has predicted it. Furthermore, a hypothesis may be supported by another that precedes it temporally. Together these form a kind of belief network, where hypotheses have their probabilities and affect each other according to conditional probability tables which are coded into the data and functionality of the knowledge sources that manipulate the blackboard.

Links between hypotheses represent dependencies. They are of different types, defining the type of the relationship. The architecture does not limit the range of types, since they are basically just labels that are being interpreted by the KSs. Two-party links, such as the support of one hypothesis to another, are marked to the link-field of the hypotheses at both ends. Multi-party links are implied by the link type (e.g. competiting hypotheses). They are coded as a two-way linked list, where each hypothesis knows the partners right before and after it in the list.

Links comprises the following attribute fields
- *Link type*.
- *Unique identifier* enables reference to the link.
- *Parties.* Unique identifiers of the hypotheses at the both ends.
- *Remarks*. A list of link related comment items.
- *Creator.* Identifier of the KS that created the link.

Knowledge sources

Knowledge sources must share a common interface in order to communicate with the scheduler that invokes them. The interface should be flexible enough to allow the encapsulation of any functional entity into a KS, and effective enough to facilitate meaningfull integration of KSs and their control by the scheduler. The internal structure and functionality of KSs is not specified.

Main contributions in designing the interface for KSs were made in standardizing the way in which the scheduler can query *metadata* about a KS and to *focus* its actions. The following metadata must be provided by a KS when queried:
- List of hypothesis types that the KS understands, accompanied with the type of the operation that may be performed for them. Currently in use: {*create*, *read*, *verify*, *modify*, *extend*, *delete*}.
- List of link types that that the KS understands, accompanied with the type of the operation that may be performed for the link. Currently used are {*create, read, modify, delete*}. Also the type of hypotheses between which the link may exists are defined for each item.
- List of methods that the KS supports.

A number of facts can be interpreted from this information. For example, a KS that extends a hypotheses into a higher level one does bottom-up processing, and vice versa. A KS that deletes competition links is specialized in resolving competitions between hypotheses. The metadata information is extensively used by the priority-rule based control, as will be shown.

A minimum set of generic *methods* that all KSs must support is:
- canOperate(*BB*, *focus*), which returns true or false depending on the ability of the KS to make operations in blackboard *BB* in the constrained area *focus*.
- writeOne(*BB*, *focus*), which causes the KS to perform one complete operation in *focus* area of blackboard *BB*, if possible. One complete operation means that the KS does not calculate everything it is able to do in the blackboard, but a single meaningful operation.

These methods allow the two fundamental operations with KSs: preconditioning with *canOperate*, and activation with *writeOne*. Both methods investigate the blackboard, and must be able to read the constraining focus parameter. *Focus* parameter constraints KSs to work on certain hypotheses or a part of them. Focus lists a subset of hypotheses's generic attributes, followed by the value constraints for these fields. As long as the focus hypotheses are worked with, the operations are allowed to cause read and write operations outside the focus, too.

Scheduler

Design of the control largely determines the successfulness of a blackboard system in integrating the knowledge of the functional entities that are encapsulated into the KS components. While the architecture by nature provides a great flexibility in structuring the problem solving, it also makes effective control difficult because this complicates the choice of the most prominent action among the potential ones.

Several different control structures have been proposed in the literature of blackboard systems. An excellent review can be found in [7]. The biggest paradigm difference can be seen between an opportunistic control that chooses the action that allows best progress with the given data, and a strategic control which tries to choose actions that most likely take towards solving the problem. In the following sections, two scheduler structures are proposed.

INTEGRATION OF BOTTOM-UP ALGORITHMS

In this section, we propose a mechanism that is particularly applicable to the integration of several bottom-up algorithms that perform a same task, yet are not redundant in the sense that they use different analysis principles. The mechanism, *priority-rule based interleaving of algorithms* achieves simultaneous interoperation of algorithms through interleaving their atomic steps.

Interleaving as a scheduling approach in general means that software components are not run simultaneously, but are temporally interleaved in order to avoid uncontrolled manipulation of common data. Construction of the proposed control consists of two steps. First, algorithms are split into the smallest atomic steps that can calculate meaningful intermediate results. Certain priority rules are then used to define the temporal order of activating these atomic entities. This allows running algorithms side-by-side in the sense that their steps are temporally interleaved and they share the data they manipulate. The size of the atomic KSs determines the level at which they may cooperate.

An important rule governing the order in which the KSs are activated is to build interpretations and extensions only on hypotheses that are *stable*. A hypothesis is unstable if there are KSs that can modify or refine the hypothesis and they have not done it yet, or if the hypothesis has unresolved competition links to other hypotheses. Since the set of KSs does not change during the analysis, the scheduler can start the computations by quering each KS the hypotheses they are able to read, modify, and write. During the actual analysis, when a new hypothesis is written to the blackboard, the scheduler does not allow the KSs to use it as input data or as a basis for further interpretations until all KSs that may modify the hypothesis have done that.

When an algorithm reaches the point where it shares its mid-level representation with another algorithm, the scheduler makes the other algorithm hit in between the calculations of the first one, in order to refine the mid-level representation before it is developed further. This is clarified by an example.

Case study

As an example, we consider a system for finding the pitches and separating the spectra of concurrent musical sounds. The details of the algorithms are beyond the scope of this article, and can be found in [10,11]. Algorithm 1 performs the task by using the *harmonic concordance* of simultaneous frequency components as a cue for indicating a common sound source. This algorithms consists of two main parts that are applied in an iterative succession The first part, predominant pitch estimation, finds the pitch of the most prominent sound in the interference of other harmonic and noisy sounds. As an output, it gives the fundamental frequency F, inharmonicity factor β, and the precise frequencies and amplitudes of the harmonic partials of the sound. In the second part, the spectrum of the detected sound is linearly subtracted from the mixture. These two steps are then repeated for the residual signal.

It was soon found out that the spectral estimates produced by Algorithm 1 are not accurate enough to remove them correctly from the mixture. More appropriate estimates can be produces by an Algorithm 2 which separates sounds by a *spectral smoothness* principle, i.e., by assuming that the spectral envelopes of real sounds tend to be continuous.

These two algorithms can be integrated using the interleaving mechanism. The calculations proceed as follows. The blackboard is initialized with a time-domain acoustic signal *wave.* A frequency transformer $\mathrm{KS}_{\mathrm{FFT}}$ calculates a short-time Fourier transform $\mathit{spectrum}_1$ which is linked to the input signal. Predominant pitch estimator $\mathrm{KS}_{\mathrm{PRE}}$ is activated. It finds the most prominent pitch and links *harmonic sound*$_1$ to $\mathit{spectrum}_1$. In this stage, the scheduler makes the spectrum estimator $\mathrm{KS}_{\mathrm{SMOOTH}}$ of Algorithm 2 to hit in between the calculations of Algorithms 1, in order to make its modifications to *harmonic sound*$_1$. After this, *harmonic sound*$_1$ is a stable hypothesis, since no other KSs are able to modify it, and the hypothesis can be used as an input data to a residual formation $\mathrm{KS}_{\mathrm{SUB}}$ which subtracts the detected sound from $\mathit{spectrum}_1$. The analysis continues for the residual.

As described in [11], integrating the spectral smoothness principle to the baseline iterative system makes a significant improvement in the performance of the system. The average pitch detection error rate in musical four-voice mixtures reduces from 25 % to 12 %. This is the kind of effect that is desired to result from a successful integration of several bottom-up algorithms.

It is quite obvious that the presented extension to the system could have been realized using basically any means. However, the advantages of a suitable architecture become more evident in the course of further extensions and in making experiments with different algorithm configurations and priorities. For the presented system, the next step would be to integrate the system with a functional entity that performs exactly the same operation, musical sound separation, but using different knowledge, synchronous spectral changes of simultaneous spectral components [12]. This situation, where two algorithms perform the same operation, yet are not redundant, is an example of an integration case, where simple input-output relations between modules do not work at all.

Discussion

A critical point in an extendable architecture is to define such data representations that are expressive and generic enough to serve several different algorithms. Definition of them is most difficult in the middle levels of the abstraction hierarchy. Some fundamental types, such as time-frequency spectrograms and sinusoidal components are rather obvious, but others are not. The problem has been addressed e.g. in [13,9]. At the very high and low abstraction levels this is easier.

Another challenge in the presented integration scheme is that the algorithms should be able, not only to produce hypotheses themselves, but also to refine and evaluate the probability of already calculated data. Two different algorithms may produce competing explanations, but they should also be able to evaluate the validity of each other's interpretations before one of them is selected.

PREDICTION-DRIVEN PROCESSING

Prediction-driven processing, or, top-down processing, utilizes internal high-level models that encode prior knowledge of the properties and dependencies of the objects in the acoustic input signals. In this approach, information in the representation hierarchy flows top-down, too. A sensing system collects evidence that would either justify or throw away the predictions of the internal model. The foundation of signal analysis is still in low-level observations, but top-down techniques can help to solve otherwise ambiguous situations: recommend an interpretation and cancel out others. Also, high-level knowledge can be used to guide the attention of the low-level analysis.

Several prediction-driven phenomena take place in the human audition (see e.g. [14,1,2]). To pick an example, *auditory restoration* can be demonstrated by cutting away a short segment of a stationary sound, and by replacing that with a wide-band noise burst. In this case, the auditory system automatically compensates for the noise burst and the broken sound is perceived as continuous under a simultaneously occurring noise burst.

The problem

Top-down predictions are not valid until they are justified or at least accepted by bottom-up observations. The core question is: *how can we efficiently direct the bottom-up analysis towards verifying a predicted hypothesis by observations?*

Often top-down reasoning is particularly needed in cases, where bottom-up analysis is difficult. An example case from the analysis of drum and percussive sounds is illustrated in Figure 2. Based on a tem-

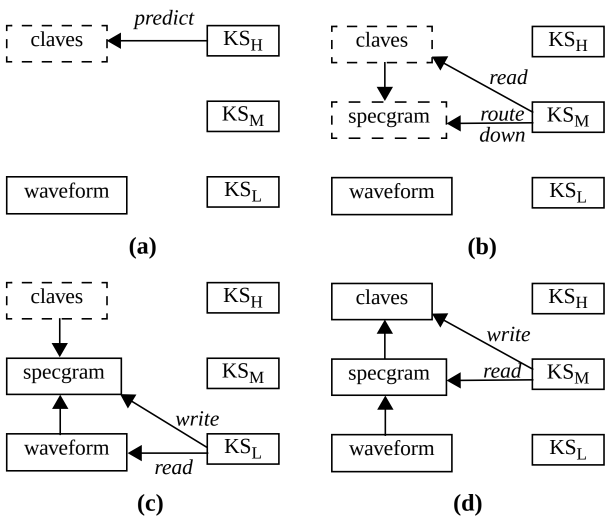

Fig. 2. Top-down routing of requests for data.

porally repeating pattern, a *claves* sound has been predicted by a KS_H to occur at a certain point. A concurrent snare drum hit masks the sound to the extent that it is only faintly audible. In order that the prediction would turn into a valid interpretation, the system must wait for bottom-up observations to give their rating to its probability. However, the sound is so badly masked, that bottom-up analysis will never come up with a probability rating for that particular hypothesis. To do that, bottup-up analysis should produce an excessive amount of all possible explanations in ambiguous situations and then just hope that the predicted explanations are included. This strategy fills the blackboard with useless data which blocks the analysis process.

The bottom-up analysis should be directed towards double-checking the predicted hypothesis. However, the high-level modules must not communicate directly with the low-level analysis modules in order to maintain the flexible addition and removal of knowledge sources. Instead, all communication has to take place via the data at the blackboard. Here a mechanism is needed to repair the break in the communication between high and low levels, a break which has been caused by two things: The mid-level modules that recognize the predicted data type cannot verify it, since their input data has not been made available by the low-level algorithms. Low-level algorithms, in turn, do not recognize the predicted high-level data type, and thus do not realize that they should start processing towards it.

Top-down routing of requests for data

The described situation is solved by a *top-down routing of requests for data*. The mechanism is illustrated in the panels (a)–(d) of Figure 2. (a) A high-level module predicts the claves sound, based on an internal model and on a temporally preceding pattern of claves sounds in the blackboard. Since the prediction has not been confirmed by observations, the module labels it as a *request* for “claves”. (b) The data type is recognized by a mid-level knowledge source, which – when not able to calculate a bottom-up probability for the sound, “routes” the request downwards by requesting its own input data that it would need to verify the predicted sound. (c) This is recognized by a low-level module, which is able to calculate input data for the mid-level algorithm. (d) In the last stage, the bridge of requests can finally be replaced with actual analysis data calculated step-by-step by several bottom-up algorithms.

This solution has several nice properties

- Bottom-up analysis can be directed towards the predicted data
- There may be several layers of abstraction between the predicted high-level data and the acoustic input data
- Each knowledge source needs to know only its own inputs and outputs, and the relation between them
- Knowledge sources do not need to know about each other

In this scheme, bottom-up algorithms produce only a limited number of rather reliable interpretations from the acoustic input. Requests for data, in turn, ensure that the results of bottom-up analysis include the observations that are required to verify the predictions at a high-level. Thus the overall analysis is driven *both* by the bottom-up observations in the acoustic input *and* by the top-down predictions based on internal models. The amount of extraneous data stays small.

The presented mechanism does not require that the knowledge sources are able to calculate their input data for a given output. Instead, they should at least request the *type* of input data which is needed to calculate the requested output data. However, some parameter fields in the requested data may have been filled to specify the request more accurately. In this case, it would be desirable to keep the requests as specific as possible in the course of routing it down. This would require that a KS is able to transform the parameters from output to input. A global integration problem has been turned into a local problem in the design of sophisticated knowledge sources.

CONCLUSIONS

A blackboard architecture for the purpose of performing content analysis in audio signals was designed and implemented. Two scheduling approaches were proposed in this framework. Although a system architecture in itself cannot perform the analysis, it may greatly facilitate the integration of, and experimentation with, different combinations of analysis algorithms. We know that the human auditory perception is “redundant” in its mechanisms in the sense that the analysis results from the collaboration of several concurrent processes. Trying to imitate this with a large collection of even simple algorithms is an almost unexplored area of research.

The presented architecture fulfills the requirements presented in Introduction to a certain extent. A critical point from the point of view of an extendable architecture is to define such mid-level representations that are expressive and generic enough to be shared by several different algorithms. Also, the design of the scheduler is likely to get more complicated when the number of knowledge sources increases.

REFERENCES

[1] Bregman. “Auditory Scene Analysis”. MIT Press, 1990.
[2] Ellis. “Prediction-driven computational auditory scene analysis”. PhD thesis, Massachusetts Institute of Technology, 1996.
[3] Russell, Norvig. “Artificial intelligence — a modern approach”. Prentice-Hall Inc., 1995.
[4] Nii. “The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures”. The AI Magazine, vol. 7, no. 2, 38-53, 1986.

[5] Godsmark, Brown. “A blackboard architecture for computational auditory scene analysis”. Speech Communication 27, 1999.
[6] Kashino, Nakadai, Kinoshita, Tanaka. “Application of Bayesian probability network to music scene analysis”. IJCAI workshop on CASA 1995.
[7] Carver, Lesser. “The evolution of blackboard control architectures”. Massachusetts Amherst University technical report #92-71, 1992.
[8] Lesser, Nawab, Klassner. “IPUS: An architecture for the integrated processing and understanding of signals”. Artificial Intelligence Journal, vol. 77, no. 1, pp. 129-171, Aug. 1995.
[9] Klassner, Lesser, Nawab. “The IPUS blackboard architecture as a framework for computational auditory scene analysis”. Proc. of the CASA workshop; IJCAI, Montreal, Quebec, 1995.
[10] Klapuri, Virtanen, Holm. “Robust multipitch estimation for the analysis and manipulation of polyphonic musical signals”. Proc. COST-G6 Conference on Digital Audio Effects, DAFx-00, Verona, Italy, 2000.
[11] Klapuri. “Multipitch estimation and sound separation by the spectral smoothness principle”. Proc. IEEE International Conference on Acoustics, Speech and Signal Processing, 2001.
[12] Virtanen, Klapuri. “Separation of Harmonic Sound Sources Using Sinusoidal Modeling”. Proc. IEEE International Conference on Acoustics, Speech and Signal Processing, 2000.
[13] Ellis. “Mid-level representations for Computational Auditory Scene Analysis”. Proc. International Joint Conference on AI, Workshop on Computational Auditory Scene Analysis, 1995.
[14] Slaney. “A critique of pure audition”. Proc. International Joint Conference on AI, Workshop on Computational Auditory Scene Analysis, 1995.