Grazer Beiträge zur Ethnomusikologie

herausgegeben von Gerd Grupe

Band 22

Die *Grazer Beiträge zur Ethnomusikologie* sind die Fortsetzung
der Reihe *Musikethnologische Sammelbände* 1 – 21,
begründet von Wolfgang Suppan, zuletzt herausgegeben von Gerd Grupe

Institut für Musikethnologie
Universität für Musik und darstellende Kunst Graz

Graz Studies in Ethnomusicology

Series Editor: Gerd Grupe

Vol. 22

The *Graz Studies in Ethnomusicology* are the continuation
of the series *Musikethnologische Sammelbände vol. 1 – 21*,
founded by Wolfgang Suppan and edited by Gerd Grupe

Institute of Ethnomusicology
University of Music and Performing Arts Graz

GERD GRUPE (Ed.)

Virtual Gamelan Graz

Rules – Grammars – Modeling

Shaker Verlag
Aachen 2008

Gedruckt mit Unterstützung der Universität für
Musik und darstellende Kunst Graz

Bibliografische Information der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.ddb.de abrufbar.

Printed in Germany.

ISBN 978-3-8322-7637-9
ISSN 1867-4682

Cover-Illustration: Rainer Schütz

Shaker Verlag GmbH • Postfach 101818 • D-52018 Aachen
Telefon: 02407 / 9596-0 • Telefax: 02407 / 9596-9
Internet: www.shaker.de • eMail: info@shaker.de

Contents

Rainer Schütz, Julian Rohrhuber

Listening to Theory. An Introduction to the Virtual Gamelan Graz Framework[1]

Introduction

One of the results of the project Virtual Gamelan Graz is a framework for experimenting with rule-based models that aim at generating the sound of a basic central Javanese gamelan-ensemble playing *gendhing* (≈ 'gamelan-composition'[2]) by evaluating *balungan*-notation (lit. 'skeleton', ≈ 'core melody'). Here, we shall give an introduction to ideas and principles underlying this framework.[3] On the technical side, we try to be helpful to readers who are not at home in the world of computer music. Conversely, as the system intends to be a research environment for central Javanese gamelan music, we try to provide some background information on the music where it seems necessary to understand decisions made in the implementation. For the novice reader it might be helpful to seek complementary information in the glossary of Javanese musical terms.[4]

---

1 We would like to thank our colleagues from the VGG project and the participants of the symposium for their valuable input. Additional thanks go to Alberto de Campo for advice and help with the implementation and to Jesse Snyder for his hard work on the final draft.

2 Translations or short paraphrases are not given to do justice to the translated term but to give those unfamiliar with Javanese music a rough idea of the concept referred to. If a term appears untranslatable, a brief English explanation, preceded by '≈', is given that fits the context (e.g. irama (≈ 'speed/subdivision level')). If a term is not well understood or its meaning disputed, but the literal translation appears helpful, the translation is preceded by 'lit.' (E.g. kalimat lagu lit. 'tune sentence')

3 It should be noted that, being part of ongoing research, its current state (mid 2008) is still in flow. However, its architecture has proven to be reliable and extendible, so that we consider it a valuable prototype.

4 There are quite a few general introductions to Javanese gamelan-music available. Among them Sorrell (2000) is well accessible, Pickvance (2005) contains an encyclopedic wealth of information, and Brinner (2008) offers a course book with accompanying audio-CD that makes sure to present the music in its cultural embedding.

The VGG implementation is written in SuperCollider 3 (SC3)[5], an advanced audio-synthesis programming environment with a rich feature-set in the domain of algorithmic composition. Combining abstraction and reasonable simplicity with the efficiency of a real-time synthesis system, SC3 is a computer language that is becoming increasingly common in computer music and sound research. While the VGG implementation provides graphical user interfaces such as editor and player for sound synthesis, the main interface for user interaction remains textual. The programming environment is not a hidden layer that produces a finished application that the user is bound to, but rather an accessible and central element of the system. We try to provide both a number of small, simple and open user interface elements for specific tasks, as well as a programming environment that permits to read and modify the program itself.

Over the last four decades, a great number of approaches have been devised of how to handle the programming process, with the result that the act of programming itself has repeatedly become the subject of computer science. Maybe the most interesting in our present context are the paradigm of *literate programming*, and methods of *interactive programming*. Literate programming emphasizes the integration of program text and scientific documentation, with a strong emphasis on the readability of the code itself (Iverson 1979, Knuth 1992). While still formal, code is comparatively close to a human readable text, combining description with computation. Moreover, such an approach emphasizes the ability to extend the language in such a way that its vocabulary remains meaningful within the research domain. Here, it follows a strategy that has evolved within computer language design: instead of keeping two accounts, one being a description of a system, the other its technical implementation, one may integrate the modeling process in a single, sufficiently descriptive programming language (Meyer 2000). A computer language then serves both as a knowledge representation system and as an active experimental context that allows to reason about the implications of this knowledge. In the case of VGG, this allows us to unify a description of assumptions about musical performance and competence

---

5 SuperCollider was developed by James McCartney and is a GPL-licensed open source application, now actively maintained by its author and a lively community. More information on SC3 is available on James McCartney's site (http://www.audiosynth.com/) and the project-homepage on Sourceforge: http://supercollider.sourceforge.net/. The VGG implementation was developed on OS-X 10.4. Since SuperCollider-versions for Linux and Windows have come a long way towards cross-platform compatibility in 2008 it should not be difficult to adjust the VGG implementation to run on both platforms.

with a system that realizes these assumptions in a form which one can listen to. The same program text has two audiences: the human reader and the machine.

A system for *interactive programming*, on the other hand, is structured as to allow the rewriting and reassembly of any significant part of the program while it is running. Thus, the programming language is not just the technical means to finally implement an already existing model, but also a part of an experimental and iterative process. It extends the idea of the integration of knowledge and computation by the concept of interactive writing. Although interactive programming is a method in the sciences at least since the late 1960s (Klerer/Reinfelds 1968), its implications are still under active research. The SuperCollider language incorporates the outcome of these efforts to provide a suitable environment for computer experiments (Rohrhuber/De Campo 2008; Rohrhuber/De Campo/Wieser 2005).

These two paradigms, literate and interactive programming, underlie many basic design decisions in VGG. As a consequence, the development cycle is fairly short, and on-the-fly changes can generally be listened to immediately. Maybe the most obvious influence of literate programming in the VGG implementation is the extensive use of nested name spaces (using `dictionaries`, see p. 140), often including Javanese terminology. Although this makes the program text less accessible to non-expert readers, we have chosen to give preference to the richness and semantic descriptiveness of Javanese terms. Replacing *laras* by ‘tuning system’, *pathet* by ‘tonal mode’, *garap* by 'part generation' or *tafsiran* by ‘derivation’ would have made the text appear more understandable to most Western readers, but would have also veered away from the specifics of the music we try to describe. Furthermore a concept like *irama* (≈ ‘speed/subdivision-level’, see p. 154) does not have an equivalent in English, and it seems impossible to find a compact term that would reflect all the musical facets that contribute to its meaning. It should be noted though, that we don't want to give the impression that we claim this implementation could offer a general and rich description of the Javanese terms it employs. It can only do so within the specific context of the given system. As such, it may be part of a process of reasoning about Javanese music and may complement existing means of description.

We hope that the VGG implementation – written within an entirely open framework that gives access to all its layers – will prove a useful means of exchanging views on gamelan-music by experimenting with the model itself. Because within the culture of the ethnomusicological community it is not common

to express thoughts about music in an audio-synthesis programming language, we think that an introduction from a basic user perspective is warranted here.

In the following, we will give an overview of different aspects of the system, starting with notation and knowledge representation, and a brief introduction of essential concepts of the language SuperCollider. From there, we use a case study as a means of illustrating the components required to model a *gendhing,* and to give a basic account of the inner logic of the framework. Research questions arising from the modeling of basic aspects of gamelan music, such as *irama*, *pathet*, part generation, and interaction are considered in the light of a concrete implementation. Finally, we describe main features of the sound synthesis system that allows users to apply formal tonal patterns to different kinds of gamelan sounds.

Notation and knowledge representation

Musical notation

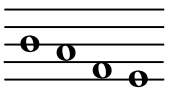
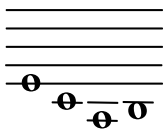
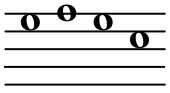
The VGG framework attempts to allow gamelan experts to model musical patterns using an already familiar notation: the Kepatihan-style cipher notation. Kepatihan has become the quasi-standard of notation of Javanese gamelan music.[6] This is not a trivial task, because in most systems executable programming code cannot contain non-standard symbols, which occur frequently in Kepatihan.[7] Although this required making some compromises, simple patterns will still look quite familiar to those familiar with Kepatihan, and it should not be too difficult to get accustomed to the special features of the VGG derivative:

---

6 Kunst (^{3}1973, vol. 1:346ff and illustrations in Appendix 3, vol. 2) gives an overview of early experiments with musical notation in Java, mainly at the courts of Yogyakarta and Surakarta in the early 20th century. In the second half of the 20th century the Kepatihan-style cipher-notation became the almost exclusive means of musical notation, broadly established both in musical practice and teaching. It was developed by musicians at the court of the prime minister (*patih*, thus *kepatihan* for the court) of Solo (Warsadiningrat 1987:165) and is strongly inspired by the notation used in the Galin-Paris-Chevé method (Bullen 1877) of teaching music.

7 It would be technically more appropriate to say that available means of reproducing Kepatihan-style notation in a computer-system are a) not standardized, b) use character code-points from the extended ASCII-set that easily conflict with editors in programming languages, are difficult to access from keyboards, and behave differently on different platforms, and c) have a display oriented logic of employing rhythm markers that can be difficult to interpret by an algorithm.

Table 1: Comparison of Kepatihan- and VGG-style notation for simple tone successions (middle, lower and higher octave-register, no rhythmical distinctions) and trans-notation with western musical symbols[8].

Kepatihan	6532	2756	2327
VGG	6532	27.5.6.	2'3'2'7
Staff-notation			

The variant developed for VGG is a superset of the diacritical notation introduced in Barry Drummond's edition of *gendhing*.[9] The notational conventions his system introduced make it possible to query *gendhing* phrases within in a web-browser, and therefore must restrict themselves to a basic set of symbols readily accessible from the keyboard in all browsers on all computer-systems. There are two systematic differences between standard Kepatihan notation and both the Drummond system and the VGG derivative:

---

8 Trans-notation to western staff notation is occasionally offered to allow a reader unfamiliar with Kepatihan-style cipher-notation to grasp a few of its basic properties at a glance, although many implications of western notation (e.g. pitch-representation) are highly misleading. In Kepatihan notation there are 7 ciphers designating tone steps (originally instrumental keys) within one of two tuning systems (*sléndro* and *pélog*). Rising ciphers refer to rising pitch and dots below and above ciphers refer to octave register. A cipher also stands for one basic rhythmic unit, which could be fast or slow, depending on performance context. There are horizontal rhythm-bars above ciphers for even rhythmic subdivision and a prolongation dot, the rhythmic value of which equals that of a tone-cipher in the same context. Tone-steps could be small (between half- and full tone), middle-sized (between major second and minor third) or large (between minor and major third), depending both on the respective tuning system and the step-position within the tuning system. Instrumental tunings may be considered as comparatively flexible, adding another layer to pitch-flexibility in Javanese music. In lack of melodic context the given examples are metrically indistinct, yet an isorhythmic group of four basic beats is usually understood as being end-weighted. End-weighted 4-beat groups (*gatra*) are often separated by whitespace to enhance legibility.

9 Gendhing Jawa - Javanese Gamelan Notation: http://muse.calarts.edu/~drummond/-gendhing.html, in particular Gendhing Search: http://muse.calarts.edu/~drummond/search2.html

- In standard Kepatihan-notation octave-dots and rhythm-bars display above or below their tone-cipher, in the VGG derivative all diacritics follow their tone-cipher in a fixed order.
- In order to express rhythmic values shorter than the basic beat, standard Kepatihan-notation groups symbols in (nestable) subdivision-pairs (tone ciphers and the prolongation dot *pin*). In the VGG derivative there is no such grouping. Instead, each tone-cipher (or *pin*) is followed by its own set of diacritical markers.

The diacritics themselves were chosen from the lower ASCII set:

Table 2 Diacritics for octave register and rhythmical values in VGG-style notation

Diacritic	Description	Meaning	Comment
'	single apostrophe	higher octave	
.	period	lower octave	
:	colon	middle octave (explicit)	facultative
-	minus	prolongation	called *pin*
_	underscore	single subdivision (half)	
=	equals	double subdivision (quarter)	
|	vertical line	no subdivision (explicit)	facultative

By convention, octave diacritics precede rhythmical diacritics. In both traditional Kepatihan notation and the VGG derivative tone-ciphers (1 to 7) and the prolongation-symbol *pin* implicitly carry the basic relative duration 1, sometimes called *sabet* (≈ 'beat', lit. 'whip') in Java. Tone-ciphers and *pin* together add up to the intended duration, while the subdivision diacritics modify the immediately preceding symbol. Subdivision takes precedence over duration totaling:

Table 3 Examples of rhythmical symbols in VGG notation

Example	Octave and tone	Relative duration
`1--`	middle 1	3
`1'=`	high 1	0.25
`1.-_`	low 1	1.5
`7.--_`	low 7	2.5

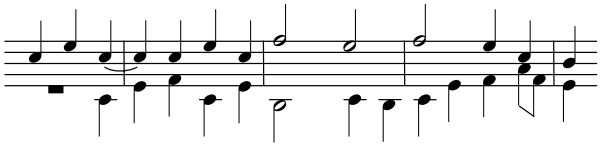
Examples in context:

Table 4 Examples of Kepatihan *balungan* notation with rhythmical diacritics, their equivalent in the VGG derivative and trans-notation of the rhythmical values.

Kepatihan	.i6i253.	.56i23i
VGG	`-_1'_6_1'_2_5_3-`	`-=5=6=1'=2'_3'_1'`
Rhythm in staff-notation		

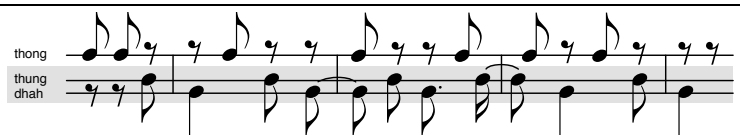
Kepatihan style notation is in general use for *balungan* and has also been adapted to other components of gamelan performance, mainly at arts education institutions in Java. Two-voice parts are usually written on two lines. Native to SC3 is the convention to write simultaneous intervals and chords in brackets. Thus the expression "[2.|2:|]" represents low and (explicit) middle 2 sounding together, each tone with the same duration 1 (explicit). Two-voice parts thus look substantially different in the VGG derivative:

Table 5 A basic version of a *céngkok* (≈ 'pattern', lit. 'kind, style') for *gendèr barung* leading to goal-tone 2, in Kepatihan notation, the VGG derivative and western staff notation. The VGG notation exemplifies the special notational convention of writing simultaneous tones in brackets.

Kepatihan	7 2̇ 7 . 7 2̇ 7 3̇ . 2̇ . 3̇ . 2̇ 7 6 . . 7̣ 2 3 7̣ 2 6̣ . 7̣ 6̣ 7̣ 2 3 5̄32
VGG	"72'[77.]2[37][7.2'][27][6.-3'-] [7.2'-]6.[7.3'-]2[32'][5_7]3_[26]"
Staff-notation	

Basic drumming notation is also fairly standardized in Solo. As we are not dealing with tone-succession, but with sounds resulting from strokes on drum-heads in different positions and with different technique, the Kepatihan-style ciphers cannot be used. Drumming sounds are represented with a different set of symbols. VGG uses a variant of these symbols, with capitals for the large drum *kendhang gendhing* and lower case letters for the small *ketipung* and middle sized *ciblon*.

Table 6: Short fragment of basic drumming (*kendhangan*) for the genre *ladrang*, illustrating differences and similarities between drumming notation common in Solo and VGG. The trans-notation indicates a separation between core-sounds (*thung* ᑭ and *dhah* ᑲ) and the filler (*thong* ∘).

Solonese	∘∘ᑭᑲ ∘ᑭᑲ∘ ᑭᑲ∘ᑭ̄∘ ᑲ∘ᑭᑲ
VGG	oopB opBo pBo_p_o BopB
Staff-notation	

From notes to events: situating assumptions on musical knowledge

VGG converts character sequences (`Strings`[10]) written in this notation to the program-internal representation of event lists. Hereby, each note is represented by an event, which, as we will see, is a more general, explicit and extendible internal representation than strings. For the person writing or reading a musical score, the notation systems described above are frequently more concise and expressive. However, in some cases this method of representation can become limiting, because masking off the complexity of the internal system also hides its richness and reduces the accessibility of potentially useful features.

In a way, this trade-off is closely related to the frequently discussed weakness of any notational representation compared to the complexity and variability of real world music. By aiming at incorporating Kepatihan-style notation into VGG we also inherit its constraints. Let us say first that users are not bound to the use of the VGG notation, the internal representation can always be used at any place were Kepatihan-notation can be used. However, all static notational representations of musical events share intrinsic limitations which cannot be overcome by increasing the complexity of the system. Perhaps more interestingly, we found that some of these shortcomings are alleviated if notation is used within the dynamic framework of an audio-synthesis application. Kepatihan-style notation is not explicit when it comes to tuning, loudness and agogic flexibility. Expressing pattern variability in written form is cumbersome at best, and communication processes during performance are necessarily ignored by the logic inherent in a static visual representation of sound-structures. Yet this does not necessarily mean that the musical structures represented in musical notation lack all musical significance; they might still aptly represent a vocabulary used by communicating musicians *in context*.

As we will see once we have gained an overview of the framework, VGG embeds notational information in a much more situation-dependent system of rules. The internal representation of notes, the *event*, acts as a dynamic medium in which knowledge can be captured incrementally. As a result, we are able to represent musical knowledge both within a specific set of competences and in the context of a given moment.

---

[10] SC3 class-names or code-examples are typographically marked by using a fixed-width font.

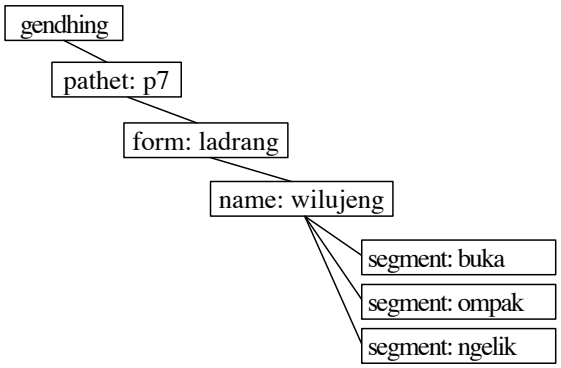
Generative principles in VGG aim to be as generic as possible. The goal is not to reproduce a specific performance statically with maximum precision. Instead, we attempt to model the general principles common to many performances while leading to different results in individual performances. We don't consider essential aspects like variability, agogic fluctuation and interaction as obstructions to using notation, but rather as complements, operating on different levels that jointly contribute to the manifestation of music as a product of culturally bound human activity. An audio-synthesis application like SC3 provides means both to model generic operational principles like variability and interaction and to store static patterns in data structures of theoretically any required degree of complexity. So rather than avoiding the use of notation we anticipate a process in which a symbolic system for the representation of static patterns continuously develops to complement the dynamic processes in which they are embedded in a more adequate manner. Requirements of the VGG implementation have already led to modifications to the initial set of symbols. As a starting point for a symbolic system used to store static data within the VGG framework we consider Kepatihan-style notation, which has evolved within the music-culture we are exploring as the most appropriate option.

SuperCollider lingo

Dictionaries and event-streams

Resembling a taxonomy tree, a `Dictionary` is a data structure that allows us to organize contextual information flexibly and to use meaningful category names in the system implementation itself. By means of nested key-value pairs, we may represent data in a way that maintains their specific frames of reference. For instance, the levels of a dictionary containing the *balungan* of *gendhing* could be *pathet* (≈ 'tonal mode'), form, name and segments:

Figure 1: Structure of a dictionary containing *balungan* notation of *gendhing*, exemplified for *Ladrang* "*Wilujeng*", *pélog pathet barang*



This dictionary, together with values for the lowest-level keys would look like this in SC3 code:

Figure 2: Dictionary entry for *Ladrang* "*Wilujeng*" *pélog pathet barang*. Subordination is encoded by nested parentheses, coordinated entries are separated by commas

```
(
  gendhing: (
     p7: (
        ladrang: (
           wilujeng: (
              buka: "7.32 6.7.23 7.7.32 - 7.5.6.",
              ompak: "27.23 27.5.6. 33-- 6532 5653 27.5.6.
                      27.23 27.5.6",
              ngelik: "--6- 7576 3567 6532 66-- 7576 7732
                       -7.5.6."
                     )
                  )
              )
           )
)
```

The segments could be further subdivided into subsegments, or generally speaking, there is no limit to the branching depth of dictionaries, the hierarchical structure may be freely adjusted to descriptive requirements.

The internal representation of a sound-event in SC3 is also structured as dictionary. For example, the notation-parser could convert tone 6. to the following representation:

Figure 3: Tone 6. in a very basic event-representation

```
(
degree: 6,
  dur: 1,
  octave: 4
);
```

In dictionaries, the order of items within a hierarchy-level is not fixed. Querying a dictionary, we cannot rely on the order of items, but instead, the key is used to retrieve a specific value. In situations where a fixed order is essential, we can use arrays, which are enclosed in brackets. The tone-sequence 27.5.6. would look like this:

Figure 4: Basic representation for the tone-sequence 27.5.6. The event-dictionaries have a fixed order and are thus enclosed in brackets.

```
[
(degree: 2, dur: 1, octave: 5),
(degree: 7, dur: 1, octave: 4),
(degree: 5, dur: 1, octave: 4),
(degree: 6, dur: 1, octave: 4)
]
```

If we use this array of dictionaries for sound synthesis (which would assume a lot of additional data not shown here) it would usually result in a sequence of four tones. If played with gamelan sounds, at a reasonable speed and with a *pélog* tuning, a competent listener might already interpret it as a meaningful musical segment: one (end-weighted) *gatra* of *balungan* in *pélog barang* leading to the goal tone low 6. The listener might associate pieces where this pattern occurs and other parts that paraphrase the tone-sequence. Conversely, a listener not

accustomed to this music might construct a beat at the beginning, consider the tuning as awkward and have none of the associations mentioned above. The computer system knows nothing of the meaning this sound sequence obtains in the mind of listeners. As our aim is to model musical competence within central Javanese gamelan music, we have to add such information to the physical sound-information in order to be able to reason about and operate with all the associations sound creates in the competent listener. To make it available for evaluation and experiment, we have to make such implicit knowledge explicit within the system. Event-dictionaries in SC3 are not limited to physical sound parameters, in fact they are not limited to anything given by the system, they are freely extendable by user-defined data, provided it can be represented in the form 'key: value'. This could be numerical data as well as lexical data and data-type names, enabling semantic criteria to be translated into actions in the processing of event-streams.

Classes and objects, patterns and streams

The system is specified in the form of so-called *classes*, which are general descriptions of *objects;* little specialized and encapsulated programs that conduct certain tasks required in a specific context. A class acts as a template, defining the characteristics and behavior of objects derived from it. For instance, each of the above event-dictionaries is an object. The class `Event` defines their common behavior (for example that they contain key-value pairs). Each class can be used to create multiple independent objects. The VGG implementation comes with many classes tailored to specific aspects of central Javanese gamelan music. Their names were chosen to be recognizable: either English terms, or – if the musical concept encapsulated therein appeared specific to Javanese gamelan-music – in Javanese terms.

A *stream* is a specific kind of object for generating or modifying other objects dynamically. So in order to create a sequence of event objects, we do not need to write them out individually, but may instead provide a rule that generates them. This also means that the sequence need not be known in advance, and that it may be defined as never-ending. The internal dynamic behavior of the VGG system is determined by numerous such streams which incrementally generate and modify sequences of events. Because one often needs multiple independent variants of the same stream, there is a way to describe streams in the

abstract: this is what is called a `Pattern`[11]. Thus, from a single pattern multiple event streams may be created, and thus multiple melodic lines. The system remains open to parallelization at all times. Additionally, since one pattern may derive from another pattern, this strategy is very useful to express how one stream modifies or depends on the output of another stream. In particular, one may think of an event pattern as an abstract description of a dynamic process that generates a sequence of event-dictionaries.

Since one of the aims of VGG is to give explicit access to assumptions about competences deemed necessary for successful music performance, patterns are an appropriate way to structure such a system. The pattern `Pgongan` e.g. keeps track of the current position of a performance relative to the main formal and implicitly metrical unit of central Javanese gamelan music, the *gongan*-period. It adds orientational properties to each event within an event-stream. After being processed by `Pgongan`, the event-stream from above could look like this:

Figure 5: Event-stream notation for 27.5.6. after being processed by a stream defined by a `Pgongan`.

```
(degree: 2, dur: 1, octave: 5, gonganDur: 32,
            gonganPhase: 4, sabet: 5)
(degree: 7, dur: 1, octave: 4, gonganDur: 32,
            gonganPhase: 5, sabet: 6)
(degree: 5, dur: 1, octave: 4, gonganDur: 32,
            gonganPhase: 6, sabet: 7)
(degree: 6, dur: 1, octave: 4, gonganDur: 32,
            gonganPhase: 7, sabet: 8)
```

These orientational properties express elements of contextual knowledge. For example, they are required in order to define positions where activities like speed changes should start or end. By serving as disambiguating properties or constraints, they also become meaningful in the generation of parts.

---

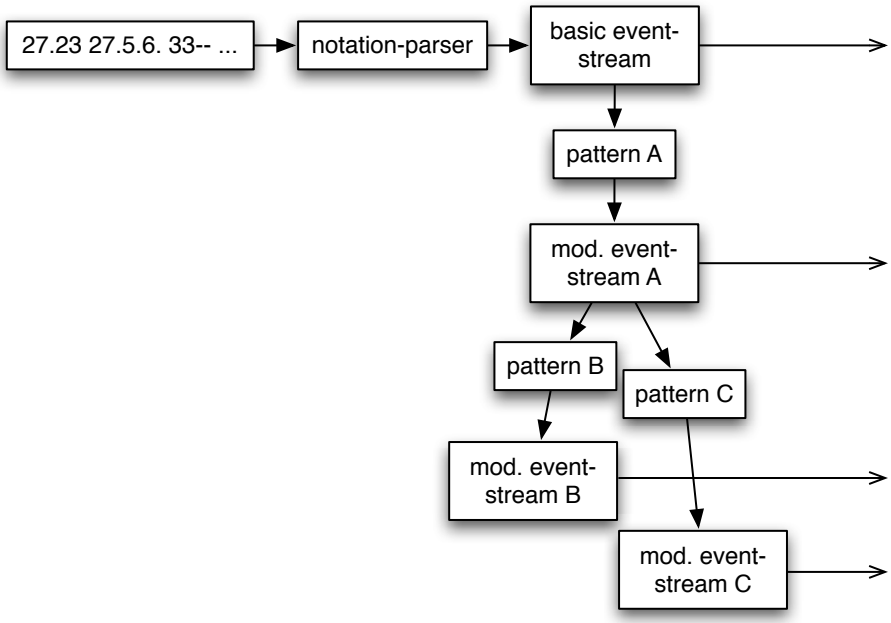
[11] Unfortunately the dynamic, operational understanding of the word 'pattern' within SC3 collides with the musicological use of pattern as a static tone- or sound-sequence. A pattern within SC3 is static only insofar as it is a description of a generator for a sequence (a stream); this stream itself may vary, while its description, the pattern itself, remains outside of time. We have to ask the reader to rely on the context to identify the current use-mode of the word.

Step by step, on their path through the system, the event-dictionaries passed between streams accumulate specific additional information, such as their *pathet*, the current *irama* or the instrument they are meant to be played on. From a conceptual point of view, each of them represents a sound event, including temporal parameters such as the sound's duration and the inter-onset time (`dur`). This is the reason why such dictionaries are called events, despite the fact that they may represent events that never happen, or events that are combined to later form one composite event. The VGG implementation is strongly based on the event-stream model of SC3. We thus may consider the basic entity of the system to be series of events, representing temporally ordered sets of sounds, or more generally, the musical knowledge about such sounds. This has two essential implications: a) we are in full control of the properties we want to add to a sound-event because we are free to define custom properties using any terminological paradigm required, and b) each event is autonomous, it contains its own complete set of properties that describe its sound.

In short, playing a piece in such a framework basically means letting the notation-parser create an initial basic event-stream from *balungan* and, by applying patterns that may add or modify properties, creating further streams. Additional patterns can define operations relative to the modified patterns or to the same input pattern (branching, parallelization). The streams derived from these patterns will be modified accordingly. The concurrent event-streams resulting from chaining and branching can subsequently be processed independently. This allows the larger system to maintain independent bundles of event-properties to be localized – and situated – in different streams of events (see fig. 6).

Streams represent a lazy computation scheme, meaning they may operate incrementally – a stream need not wait until its input stream finishes processing its entire sequence. Instead of this, streams receive one event at a time and pass it on, as soon as it has been processed, which usually is much faster than the duration of the sound event itself. Therefore many streams can operate within the timeframe given by the effective duration of an event. In some cases, this basic one-in and one-out logic is complicated by the fact that streams have to evaluate more than a single event in order to process the current event. Then, one of the stream nodes has to accumulate a number of events by computing a little bit ahead of time.

Figure 6: Schematic illustration of the branching of event-streams caused by applying patterns to them. An initial basic stream is created from Kepatihan notation. Patterns describe streams that modify the input stream and create a new one. A single pattern can serve as basis for several derived patterns (pattern B and C from pattern A). The resulting streams B and C share properties of the basic stream and stream A, but contain separate properties described by patterns B and C respectively. All 4 streams remain available to the system.



Generally, realtime processing is mandatory for two features of the system: to allow a user to interfere with the ongoing sound – which therefore must not be predetermined too far ahead – and to model interaction among musicians, which can also interfere with the default progression of a performance at any moment. This can require tight timeframes between recognition and reaction. Given the importance of anticipation in music praxis, it can be hardly surprising to encounter contradictions between different levels of adaptive behavior. A stream may need to look ahead to what comes in the future of its input stream, while a choice predicated upon such an 'expectation' may preclude a reaction to a sudden change from the outside. This tension is not merely an artifact of the tech-

nology; it is partially a reflection of a similar tension within the realm of gamelan performance practice. As such, it is an interesting area for realtime exploration and experimentation, even if such experimentation demonstrates that "realtime" is not as real as one would maybe like it to be.

Tags

Let's consider the basic units for adding to the flow of events in the system. Events and dictionaries, which here mainly differ in usage and not in functionality, are composed of simple associations between keys and values. What we call *tags* are quasi-lexical units represented as a key-value pair 'symbol: Boolean' (e.g. `goToNgelik: true`) which can be fed into an event-stream. This means that they become a property of each event in their stream. Tags can be set and removed in realtime and become localized in time by the events they are bound to. As tags are encapsulated in their respective event-stream, concurrent streams (like different parts playing together) can contain different sets of tags, but may also share them. Setting tags in realtime typically happens in one of two ways: either as a result of external intervention or by actions specified by musical patterns, for example to trigger change in the program flow conditionally. As a convention, we therefore distinguish external- and internal tags.

External tags operate indirectly. Rather than triggering a 'switch to *ngelik*', or a set of activities necessary to 'end a piece' directly, they only trigger the first musical signal that initiates the larger process. The next step is triggered by the recognition of the musical signal in a pattern called `Preact`, which in turn could trigger more signals by the means of internal tags. The external tag thus resembles the decision of a musician responsible to initialize a change, leading to a musical signal that propagates to other musicians. External tags are preceded by the prefix `mc`[12] (Figure 7). The most common external tags can also be set in a simple graphical user interface for playback .

---

12 Abbreviation of 'master of ceremonies', inspired by the MC [emsi] at Javanese weddings. External tags could also be seen as representing outside requests. Bp Emsi ('Mr. MC') has gained some notoriety among musicians for making difficult requests, and for overriding their decisions in order to adjust the performance to unexpected ceremonial developments, or to expose his capabilities as singer, comedian or the like.

Figure 7: Examples of the syntax used to set and unset tags manually

```
event-stream.tagable.tag = (mcGoToNgelik: true);
event-stream.tagable.tag = (mcDoSuwuk: true);
event-stream.tagable.tag = (mcGoToNgelik: false);
event-stream.tagable.tag = (mcDoSuwuk: false);
```

Internal tags are the result either of pattern-recognition or of another type of evaluation of the flow of a performance, e.g. orientational information (which beat in which *gongan*) or speed-change information. While internal tags can be set and unset manually in almost the same manner as external tags, this is more risky, because internal tags are often part of a complex chain of events triggered by, and depending on one another. The internal tag `goToNgelik`, triggered by the recognition of a *ngelik*-signal simply switches to another segment, but the internal tag `suwuk` (≈ 'end the piece') for a *gendhing* like *ladrang* Wilujeng triggers a chain of events stretching over more than one gong-period, and including several speed changes, a context-dependent segment switch, changes in drum-patterns and dedicated behavior towards and at the final *gong* (Figure 10). More importantly though, internal tags are not intended to initialize activities modeled to be results of interaction. They are just the messengers of decisions resulting from the playing and recognition of musical signals.

Modeling a *gendhing*

In the following, we will cover the basic components of the framework by example of modeling a *gendhing*. This overview explains what components are required to achieve this and how these components are combined. VGG being an open framework, both these components may be replaced by different, possibly more refined versions, and may be combined in entirely different ways. Giving an account for one possible implementation, we hope to make it reasonably easy to develop new models, as well as to use and complement the given one.

Performance flow

Before starting playback, we have to set some general initialization parameters like speed, volume and instrumentation (or accept defaults) and select a piece from the mentioned dictionary of *gendhing*.

Figure 8: Some initialization parameters to be set before starting playback of a piece

```
g.gendhing.pathet = \p7;
g.gendhing.form = \ladrang;
g.gendhing.name = \wilujeng;
g.gendhing.gonganDuration =
g.gongan.[g.gendhing.form].gong[0];
g.gendhing.bukaDuration =
g.gendhing.notation.buka.eventDuration;
g.gendhing.segments = g.gendhing.notation.keys.asArray;
g.gendhing.initialTempo = 1.8;
...
```

Balungan-notation is usually split into the segments a *gendhing* consists of (e.g. *buka* 'opening', *umpak* ≈ 'transitional part', *ngelik* ≈ 'vocal part') which can also be nested in substructures (e.g. a *ngelik* containing several *gongan*). We need a functionality that brings segments into the required order and follows each substructure. As neither the amount of repetitions nor segment succession is predetermined, a method is required to dynamically decide how to continue at each segment border (most often at a *gong*). While there is always a default order at each position of choice, musicians can choose an alternative path, e.g. in order to adjust to an unexpected occurrence in the context of the performance. Even if the default order is maintained, segment succession is reinforced by musical signals played by musicians whose musical roles include this responsibility.

A common case is a branch-signal, played to indicate a switch to a segment called *ngelik*, in which metrically bound singing of classical poetry steps to the foreground.[13] The signal is played by the bow instrument *rebab*, and in its ab-

[13] The importance of musical signals for organizing performances in Javanese gamelan-music has been pointed out frequently. Heins (1970) puts a focus on *wayang* accompaniment and the

sence by the gong-chime *bonang barung*. Other parts[14] can also reflect or anticipate the shift to the high register, from which *ngelik*-singing usually starts. According to Javanese musicians it is primarily the responsibility of the *rebab* to indicate the switch. In actual practice we can observe that the *rebab* generally is the first to rise, while the *bonang* is the most widely audible part among those that can reflect the register-switch in their playing style. It is noteworthy that the *ngelik*-signal itself is not a dedicated pattern used exclusively as branch-signal, but a pattern reflecting high register in a melodic context, where unmarked playing would imply a low register. This is interesting in itself, but also poses a challenge to pattern-recognition, because the object to be identified is not just a fixed tone-sequence, but a defined difference in *garap* (lit. 'treat', ≈ 'part-generation') that could manifest itself in many different surface-forms. Realtime pattern-recognition (on a symbolic level) is one of two important mechanisms that trigger setting a *tag,* which in turn may determine the segment played next. In figure 9 we see how the *tag* '`goToNgelik`' controls segment succession. The ruleset[15] contains two blocks containing succession rules (associations denoted by an *arrow* between two objects) for all segments of the performed piece. If the '`goToNgelik`'-*tag* is set while a segment switch is approaching, the block headed by '`goToNgelik`: true' will be evaluated, if not, the default rules apply (headed by an empty event-dictionary '()').

interaction between *dhalang* ('puppeteer') and musicians, and Brinner (1995) describes them within a general framework of musical interaction.

14 *bonang panerus*, *gendèr barung*, *gambang*, *siter*, *gendèr panerus*, *suling*, *sindhénan*, if present *gérongan* and sometimes also *kenong* and *kempul*

15 As opposed to the *gendhing* `Dictionary`, rules are stored as `Array`, which is indicated by their enclosure in brackets [] rather than braces (). As opposed to a dictionary, the elements in an array have a fixed order. This makes sure that every condition is applied in a deterministic order.

Figure 9: Succession-rules for a simple form containing the three segments *buka* ('opening' – played only once), the unmarked *gongan umpak* (≈ 'transitional section') and the vocals dominated *gongan ngelik* (lit. 'make small' i.e. 'become high').

```
g.gendhing.segmentSwitch = [
   (goToNgelik: true) -> [
                  \umpak  -> \ngelik,
                  \ngelik -> \ngelik,
                  \buka   -> \umpak
                 ],
    () -> [
                  \umpak  -> \umpak,
                  \ngelik -> \umpak,
                  \buka   -> \umpak
          ]
    ];
```

Ending a *gendhing* requires a decision – otherwise the performance would go on forever by repeating one or several *gongan* over and over. The decision to come to an end could come from outside or from a musician, usually the drummer. In a composition like *Ladrang* "*Wilujeng*", performed with *gérongan* in the *ngelik*, a default *suwuk* (≈ 'process of ending a piece') would be initiated by the drummer well before the start of the last *ngelik* and stretch over almost 1.5 *gongan* (Figure 10). In this case a slight speedup (*ngampat*) at a defined position becomes the signal for all musicians that the piece is going to end at the next default opportunity. This results in an obligation to switch to the *ngelik* at the next opportunity because the piece should end there. Therefore the *rebab* has to signal the *ngelik* before the *gong* (it usually has the choice to signal *ngelik*) and other *garap*-parts reflect the signal by switching to high register as well. This behavior is made obligatory by the initialization of *suwuk* by the drums, but it doesn't outwardly differ from a normal switch to *ngelik,* except that it is performed at a slightly increased speed. While the *garap*-instruments switch to high register the drummer plays a special pattern[16] to the gong, which is only used towards the penultimate *gong*, thereby reaffirming the impending end at the next gong. Another modification of standard drumming to reinforce *suwuk* can be

[16] The drum-pattern towards the penultimate gong in *ladrang kendhang kalih* .ꞵ.ꟾ.ꟾ.ꞵ.ꟾ.ⓑ is taught at ISI-Surakarta but not always used in other communities.

used towards the second *kenong* in the *ngelik*.[17] The final phase of the *suwuk* is initiated towards the third *kenong* by continuously slowing down until the final gong. The slowing down is controlled by the drummer who uses a dedicated pattern. During the slowdown musicians with subdividing instruments do not do an *irama*-switch, as speed would require in a default situation, and all instrumentalists wait with the last tone for the strongly retarded gong to be hit first.[18]

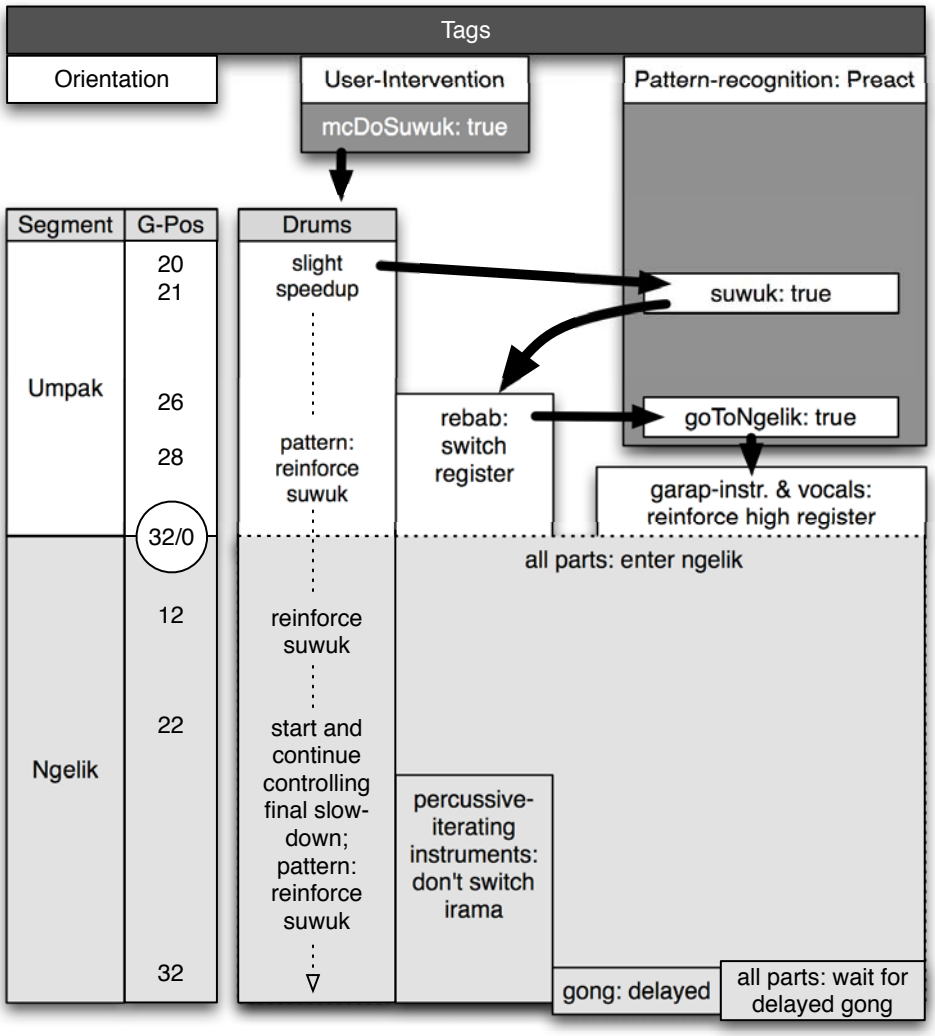
This description of the way to end *Ladrang* "*Wilujeng*" can be used to end all pieces of comparable build (form and performance mode) if implemented in a generic way, i.e. not bound to a static reproduction of *Ladrang* "*Wilujeng"*, but formulated in terms that can also be applied to other pieces (Figure 10). The actual steps to be taken (use dedicated *kendhang*-patterns, speed up or down at certain positions relative to the *gongan*, branch because of impending *suwuk*, etc.) are formulated programmatically in a way that can be reused and modified centrally if so wished. Most importantly the process is not modeled as a fixed sequence of occurrences, triggered once by an external tag, but as a process which at two points depends on a mechanism that a) creates a musical signal and b) recognizes this signal by evaluating its symbolic representation, without relying on the initial trigger.

---

17 Adding a .ʔ.ɓ before the default .ʔ.ɓ.k͡tʔ.ɓ̂

18 Vocals, especially the *pesindhèn* ('female solo vocalist'), play a central role in the coordination of the timing during the last tones and for the final gong. It is outside of the scope of this description to follow this up, and outside of the possibilities of the VGG implementation to model such interdependencies, as vocal parts are not implemented yet. *Suwuk* without vocals can be observed in ensembles that don't employ vocalists (e.g. *bonangan*).

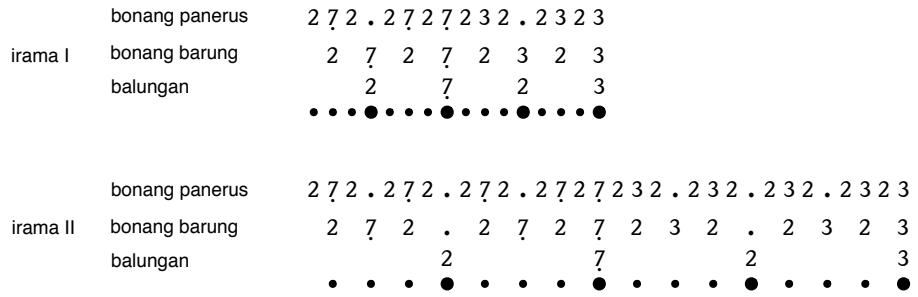
Figure 10: Slightly simplified representation of the most common, unforced way of ending a piece like *Ladrang* "*Wilujeng*" with vocals of the poetical form *salisir* in the *ngelik*. The process is initiated by the external tag `mcDoSuwuk` and continued by signals triggering two internal tags (`suwuk` and `goToNgelik`). Timing is controlled by evaluating orientational tags (current segment and current position in gong-period in beats [*sabet*]), continuously supplied by the class `Pgongan`.



Irama and speed

The interrelation of speed and *irama* (≈ 'speed/subdivision-level') are handled by the pattern class `Pirama`. The stratification of the musical texture into layers of different beat subdivision levels is an eye-catching feature in central Javanese gamelan music.

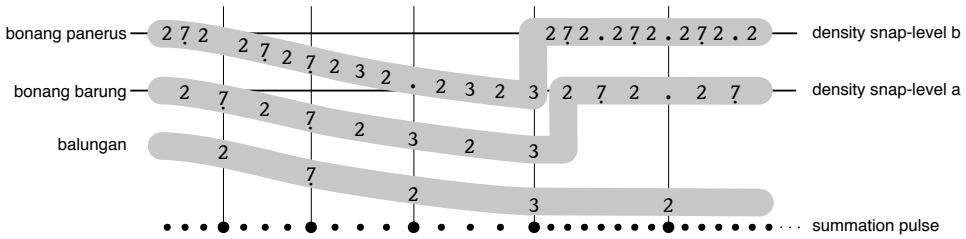
Figure 11: Two *irama*-levels with *balungan*, exemplified by schematic realization of the gong chimes *bonang barung* and *bonang panerus*. Both instruments play with the same beat density on both *irama*-levels, yet with twice as many tones per *balungan*-tone in *irama* II. At some point of an (usually continuous) *irama*-transition the subdividing instruments will snap back to their idiomatic beat density.



While most of the time a gamelan plays at one of various static speed-levels, there are phases of transition, where the entire ensemble either slows down or speeds up until it reaches a new static speed level, which usually is close to half or double of the previous speed (as measured by *balungan*-beats). This process is often called *irama* transition. While some parts, e.g. the *balungan*, simply go along this change, others switch their subdivision-level to half or double to compensate for the speed change. Because of this behavior it is sometimes difficult to tell whether a gamelan plays faster or slower after a transition. If the new static speed-level is less than half the previous speed, all instruments that 'snap back' could be described as playing faster than before while the *balungan* (and interpunctuating instruments) have slowed down considerably. The same paradox holds for the opposite speed change direction. In fact the slowest speed level from the perspective of *balungan*, the *rangkep* (lit. 'doubling') of *irama wilet* (often just called *irama rangkep* or *irama IV*) is often seen as the fastest playing

style, because the percussive *garap*-instruments play with a higher beat-density than on other *irama*-levels. So if talking about speed in Javanese gamelan music it is necessary to make explicit which *irama* and part we are referring to.

Figure 12: Schematic illustration of an *irama*-transition from `\ir1` to `\ir2`. The gong-chimes *bonang barung* and *panerus* snap back to their idiomatic stroke-density once the *balungan*-speed has stabilized on a level close enough to half of the initial speed, such that snapping back can result in an idiomatic stroke density. Consequently the pulse resulting from the summation of all parts also returns to its initial speed or density. Practice isn't necessarily as systematic as indicated here – players do not always switch at exactly the same moment, nor necessarily at the end of a 4-tone-group (*gatra*). The duration of the transitional phase, as controlled by the drummer (*pengendhang*), may differ substantially. The actual timing of this interactively controlled transition isn't well understood yet. We hope that, in the future, the VGG implementation can help to gather some more insights.



VGG distinguishes five *irama*-levels by defining different speed-bands relating to one beat of *balungan mlaku*[19]. The fastest (`\ir0`) and slowest level (`\rangkep3`) only require the definition of one threshold to their inner neighbor – the three inner levels have an upper and a lower threshold. In order to reduce the risk of instability (jitter) during transitions, or when micro-rhythmic variations occur around thresholds, the system distinguishes between thresholds while

[19] *Balungan mlaku* is generally considered the default *balungan* density. It is usually notated as a succession of ciphers without any rhythmic diacritics (e.g. `27.23`) while its sibling, the sparser *balungan nibani*, is notated with interleaving prolongation-dots (e.g. `2.3.`). One way of defining *balungan mlaku* without reference to notation is by relating it to the gong period of a specified form: a gong-period of the form *ladrang* consists of 32 *balungan mlaku* or 16 *balungan nibani* beats. The definition is clear when bound to either *irama* 1 or 2 and can - e.g. because of conflicting notational traditions (which in turn might reflect perception shifts) - become blurred in *irama* 3. This discussion lies outside of our current scope.

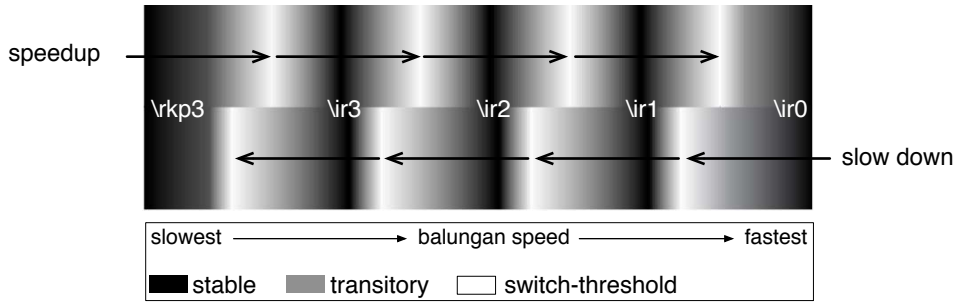
speeding up (higher) and slowing down (lower). This seems to agree with actual musical practice: in a downwards transition, musicians tend to switch rather late in a relaxed manner, reinforcing the new *irama* only once it has almost been reached. This prevents both an impression of haste and does not push the drummer during slow down. Therefore the downwards-threshold is close to the target speed (see Figure 13, the lower band. The threshold values are represented in white, the target speed in black color). In an upwards-transition players tend to switch earlier (relative to the target speed). The accelerated pulse quickly becomes too fast to be played, but musicians tend to switch even earlier than their technical reserves would allow, both to prevent the impression of haste and to ease and reinforce the speedup (see Figure 13 upper band). The speed at the threshold will nevertheless be higher than during downwards transitions. Switching late also allows distinguishing a proper *irama*-transition from a slight speedup to, e.g. to initiate the ending of a piece, or to adjust to contextual requirements in a dance or puppet-play (see fig. 13).

Speed is not the only factor determining the moment of actual switch in an *irama*-transition. An interfering factor is the principle of pattern-integrity. A threshold might be passed in the middle of a pattern that has a melodic conclusion. Switching exactly when the threshold is passed could break the melodic conclusiveness. The current VGG implementation always concludes a pattern before a part switches *irama*. As a consequence, parts sometimes switch at different moments – an occurrence that can also be noticed in actual practice. Yet the implications are deeper, because a) patterns might have an internal breakpoint due to sub-segmentation and b) because we can also observe some players trying to find creative solutions for the wish to switch close to a felt threshold, obviously in order to reinforce the new *irama* from an as appropriate as possible moment onwards.

Within general usage, automatic *irama* switching just works in the background. Dedicated experiments would probably start with fine tuning thresholds and refactoring the model. It is not very probable that thresholds are the same among different forms, genres and performance-contexts of gamelan-music. Thresholds might also depend on the starting speed of a transition. Another point of interest not mentioned so far is the profile of transition curves, which will be discussed in the context of a separate model of interrelating clocks (see p. 176). There are many open questions as to the adequate parameters of speed gradients.

The current state may be seen as a starting point from which to explore these, based on evaluation by competent musicians.

Figure 13: Schematic illustration of the model underlying switch-thresholds in irama-transitions in the VGG implementation. Each irama-level (\rkp3 to \ir0) has a stable speed-bandwidth (deepest black). The upper band shows thresholds (white) during upwards transitions, the lower during downwards transitions. The grey-shaded areas between stable bands are only traversed during transitions.[20]



[20] The very senior Solonese musician Bp. Mujiono, who likes to entertain his co-musicians by leading the gamelan on the drums through very uncommon and unexpected transitions and speeds, once had a spectacular success at a *klenèngan* at the home of Bp. Rahayu Supanggah (Benawa, 2003), when drumming *ladrang* "*Pangkur*" in a most unexpected manner. While he is known for entering *irama rangkep* unusually often and in the most unexpected places, in this performance he surprised everybody by doing the opposite in a piece where a drummer has the opportunity and is expected to demonstrate his originality, amongst others by using *rangkep* a lot. Bp. Mujiono's first surprise was that he only started the first transition into *rangkep* at the most common place towards the first *kenong* in the *ngelik.* While this would be normal for most drummers it is impossibly conformist for Bp. Mujiono. Yet he made up for this "shortcoming" instantly by never slowing down far enough to conclude the transition and enter *rangkep,* while it had already become too slow for *irama wilet*. Bp. Mujiono kept his co-musicians in this unstable state between *irama III* (*wilet*) and *rangkep*, and stayed there for almost two *gongan*, without ever - and that is very unusual for *ladrang* "*Pangkur*" too - stopping to allow for a solo of one of the female singers (*andhegan* 'stopping'). The performance was accompanied by frequent exclamations of amused irritation and ironic protest by the musicians. On a transitory speed-level, musicians continuously expect the conclusion of the transition (which Bp. Mujiono never allowed to happen) and become unsure at which subdivision-level to play. Next to being a priceless example of extravagance in *klenèngan* it helps to demonstrate that unstable speed-bands between *irama*-levels do exist.

Pathet and tonality

Many *gendhing*, especially more popular ones, can be played in several *pathet*. Accordingly, many *céngkok* can be transposed a step up or down as well as be used in both tuning systems *sléndro* and *pélog*. This circumstance, which poses quite a challenge to a Western notion of melodic identity, is better understood in its general outline than in its details. We can't always easily tell why certain pieces or *céngkok* can be transposed and others not, whether small changes in transposed *céngkok* should be considered a part of a general fuzzy variability, part of an instrumental idiom, a result of general tonal dynamics, or simply part of – not necessarily homogenous – traditional convention. Additional phenomena that add to complexity are temporary shifts of the tonal centre or tonal ambiguity, *minir* in *sléndro*, where only *rebab* and voice use shifted tones, and momentary alterations in *pélog*, which can sometimes cause a scale-split between pentatonic *garap*-instruments and the rest of the ensemble.

On the implementation side, we want to avoid writing rules and patterns in several *pathet* if they can be derived from one another by transformations. This keeps data-sets smaller, simplifies their maintenance and is a significant step towards modeling tonality in Javanese gamelan music. On the other hand there is a risk of over-generalization: if some forms of *céngkok* are bound to a certain *pathet* or context, we cannot transform them together with other, more generic ones. Therefore we need to be able to specify the tonal scope of a rule.

Such specifications are also a prerequisite for the possibility of playing back pieces in a different *pathet* than their source notation. Principally there are two options: either the *balungan* is transformed first and parts are generated from that new base, or transformation is done during part generation. The first strategy seems more straightforward, it basically only requires a few tables defining how tones are mapped in a transformation. Cases though, in which plain tone-per-tone mapping does not lead to the required results because shift of mode and/or tuning system also causes a change of tone successions in the *balungan*, require additional means of transformation. Such changes can be handled by the rewrite system used for part-generation described in the next chapter. As said above the rewrite-system aims at reducing the amount of necessary rules by doing *pathet* normalization where applicable and therefore also offers the option to transform between tuning-systems and/or transpose between modes. As the use of a rewrite-logic is required anyway to cover cases in which *balungan* changes

melodically in transformations, it might thus be more efficient to do such transformations during part generation.

Transforming a piece between the heptatonic tuning system *pélog* and the pentatonic *sléndro* is straightforward only if we know that the *pélog* version consistently uses a pentatonic subset of the heptatonic scale. In this case the notational conventions of Kepatihan notation are such that the tones mapped in a plain transform between the two tuning systems are designated by the same ciphers, except of course for *pélog* 4 and 7, which do not exist in *sléndro*. Yet for consistently pentatonic *pélog* pieces the transform from *pélog* to *sléndro* is nevertheless straightforward, as 7/1 and 4/3 never concur in such pieces. 7 thus maps to *sléndro* 1 and 4 to *sléndro* 3. When transforming from *sléndro* to *pélog* the traditional *pathet* designation often, but not always allows identifying the *pélog* tone to be chosen among the two alternatives. The situation is further complicated by the fact that many if not the majority, especially of larger *pélog gendhing* use more than 5 tones. The most common case is an alteration of high 1 to the lower 7 at the apex of a melodic contour. Alterations between 4 and 3 are less predictable, and sometimes, though rarely, a high 4 should be mapped to 5 rather than 3 in a pentatonic reduction. Except for this last case transformation from *pélog* to *sléndro* should be straightforward, yet the result need not necessarily be pleasing for competent musicians. As said above, the factors determining whether certain transforms are possible or not are not well understood yet. We have a traditional body of pieces existing in various *pathet*, experimentally generating transformed pieces not documented by tradition could help to understand better, which factors prevent transformations.

An additional challenge for refinements of the treatment of *pathet* by the system is the handling of traditional *pathet*-assignment, which does not conclusively contain the tonal information required for part generation. Some pieces in *pélog pathet nem* follow the *manyura* logic of tonal relations among *céngkok* and parts, while others follow the *sanga*-logic. This means that traditional *pathet* assignment can be ambiguous in a feature essential to pattern-generation in *garap* parts. Additionally, quite a few *gendhing* have momentary shifts of their tonal center, or have tonally ambiguous segments, i.e. segments that are treated differently in tonal respect in different performances, or by different players. While it might be possible to disambiguate such cases in the rewrite system by evaluating long contexts and metrical constraints, it might as well turn out to be necessary to subcategorize *pathet* with secondary tonality specifications. The

latter approach is more straightforward because it does not assume that tonal disambiguation can always be done by context-evaluation, but requires piece-specific additional information to be added either to the source-notation or to dictionaries complementing traditional notation.

Part generation and the rewrite system

Parts are generated in realtime by means of VGG's rewrite system implemented in the class `Tafsiran` (lit. 'interpretation'). Rewrite-rules are formulated as associations which are evaluated in a fixed order. An association binds a match-key to a rewrite-value. A match-key in its simplest form is represented as a sequence of tones in VGG notation. In order to produce valid results, the duration of match-key and rewrite-value have to be identical[21]. To derive a pattern for *bonang barung* from *balungan* we would write:

```
"27.5.6." -> "27.5.5.5.7.--5.7.5.-6.6.7.6."
```

Order in matching events

Both match-key and rewrite-value may have various forms, and may be recursively nested. The rules are listed in arrays and thus are evaluated in a fixed order. The first key to match returns the rewrite-value (see fig. 14).

The relevance of match-order becomes apparent by comparing associations 1, 2 and 3 in Figure 14. In absence of the first rule the *bonangan* for *balungan* `"27.5.6."` would be generated by matching `"2.7."` first, and `"5.6."` in a second run (associations 2 and 3). Yet it is preferable to use the pattern returned by association 1. If `"2.7."` or `"5.6."` occur in different melodic context, e.g. `"2.7.23"` or `"5.6.7.6."`, the output of the associations 2 and 3 is valid. Thus more specific associations are positioned in front of more general ones, to make sure they are matched first.

---

[21] The rewrite-system assumes that the duration of match-key (more precise: replacement window, see below) and rewrite string are identical. It will scale any duration of the rewrite-string to the duration of the replacement window within the match-key. This prevents loss of synchrony caused by errors in rewrite strings, but more importantly adds some notational flexibility: rewrite-strings, consistently written at double- or half-time, will also be rendered correctly.

Figure 14: A small array of rewrite-rules to demonstrate the relevance of match-order

```
(irama: \ir2, pathet: \p7) -> [
   "27.5.6." -> "27.5.5.5.7.--5.7.5.-6.6.7.6."
   "27."     -> "27.2- 27.27.",
   "5.6."    -> "5.6.5.- 5.6.5.6.",
   "33--"    -> "3.3.[3.-3-] 3.[3.-3-]3. [3.-3-]3.[3.-3-] 3.[3.-3-]",
   "33"      -> "3.3.[3.-3-] 3.[3.--3--]"
   ]
```

Broadening scope with variables

Comparing associations 2 and 3 reveals some redundancy. While the keys differ, both share the same derivation-principle: the tone-pair of the match-key is repeated four times, the 4th tone of the resulting string is left out, and the duration of the third tone duration is doubled. Such a derivation could also be generalized by using a single `Function` that processes any tone-pair.[22] VGG offers another way to define such derivations in a general manner which keeps close to the syntax used to formulate associations. It uses letters as variables for tones, or technically speaking, as variables for a single `Event` of arbitrary tone degree and duration, both in key and value. The following association will cover above associations – and any tone-pair – with a single rule:

```
"ab" -> "aba-abab"
```

Associations of this degree of generality can be expected to be strongly overgeneralized, so care must be taken to position rules in front of this one for any case where a tone-pair is handled in a different manner. Tone-repetitions, for instance, should be treated differently from tone-steps. Therefore the above rule must be preceded by a rule that makes sure that all tone-repetitions are matched first and only the remaining tone-steps are matched by the more general rule:

```
"aa" -> "a.a.[a.-a-]a.[a.--a--]"
"ab" -> "aba-abab"
```

---

[22] In SC3 e.g.: `~pipilanIrII = { |a| a.dup(4).flat.put(3, "-").join }`

As we see, VGG variables can take octave and duration diacritics like normal tone-ciphers. A mixture of tone-ciphers and variables is also possible. Variables in rewrite-associations allow formulating very general rules with a simple and transparent syntax. Yet this also introduces a risk of overgeneralization. One of the challenges for adequate part-generation will be to prevent these by finding the correct match-order and by constraining the applicability of rules with adequate conditions.

While letter variables are bound to single events, the asterisk `"*"` makes it possible to match several events at once. This wildcard `"*"` takes rhythmical diacritics appended to it (e.g. `"*--"`) and sums up the durations of an arbitrary amount of events until the duration indicated notationally has been reached. This allows the formulation of match-keys that look for a goal-tone (*sèlèh*) at a rhythmically defined position, regardless of the tone-degrees and amount of tones/events before the goal-tone. As opposed to `"abc3"` the key `"*---3"` will not only match `"5653"` and `"1'653"` but also `"1_2_3_5_6_5_3"`.

It is helpful to understand the logic behind this functionality. Both keys (match-strings) and values (rewrite-strings) of associations are entered in notation as `Strings`. This is a convenience method that allows representing musical patterns in a relatively familiar form. These strings are converted internally to arrays of events by the notation-parser of VGG, and it is through this notation-parser that certain characters gain special functionality. For greater flexibility it is possible to use event-notation directly in the rewrite system as well. It is even possible to mix event-notation and string-representation. The parser only converts notation enclosed in quotation marks (i.e. `Strings`) and passes everything else to the system unchanged.

An important feature of a rewrite system is the fact that there need not be any intrinsic relationship between input and output values – they are just defined as being associated. Parts like *pekingan* and *bonangan* in basic playing modes can well be understood as being derived from *balungan* by processing its tones (doubling, grouping, iterating...), and one could argue that such relations between in- and output should be reflected in the generative model. In fact earlier versions of the VGG implementation contained functions to derive *pekingan* and *bonangan mipil* from *balungan*. Functional representation of parts-derivation can easily get very complicated or even impossible. As the event-variables allow for a concise and efficient representation of such derivation-relationships, the

functional notation was given up for the benefit of a unified syntax of defining part-generation.

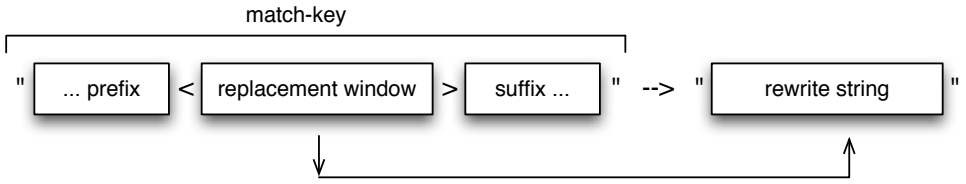
Disambiguation by melodic context

During the rewrite process, an input melody – typically but not necessarily the *balungan* – is sequentially compared to keys from the rules-array. Each time a match is made, a rewrite-string is returned and the matched string from the input melody is discarded, or – as we will see below – moved to the context. Subsequently the process starts over. Provided every tone-sequence of a melody finds a match, the concatenation of all return-strings creates a complete new part. As an implicit requirement of this operation, the rhythmical durations of key and value must be identical, otherwise input and generated part will not stay synchronous. The system exploits this implication by scaling all rewrite-strings proportionally to a duration identical to that of the key. This allows us to notate a value on any rhythmical level. Provided the rhythmic proportions are correct, the rewrite-system will deliver the right durations, thus both `"21"` and `"2_1_"` will return the same duration values.

While the above principles are useful, they do impose a serious restriction on the rewrite system introduced so far: they imply that a rewrite-string can be determined by internal features of the input pattern to be rewritten only. Yet many rewrite rules require the consideration of melodic context of the actual rewrite-string.[23] Therefore the rewrite-system offers the option of distinguishing three segments within a match-key: prefix, replacement window and suffix.

[23] All – at least recent – students of *garap* instruments have been taught by use of expression like: "to reach goal-tone z use *céngkok* n if starting after tone x. Use *céngkok* o if starting after tone y". Such rules consider what we call the prefix within the match key.

Figure 15: The three segments of a match-key. Only the replacement window is rewritten in the current run. In the subsequent run the content of the replacement window is added to the prefix.



The prefix of a match-key is compared to that part of the input melody that has been rewritten in the previous runs. The suffix is compared to that part of the input string that will be rewritten in the next runs. Prefix and suffix allow adding melodic context as another constraint to the rewrite rules. They are separated from the replacement window by angle brackets. This allows formulating a more specific variant of rule 1 in 0:

```
"3<27.5.6.>33" -> "27.5.5.5.7.--5.7.5.-6.6.7.2"
```

And as a continuation:

```
"27.5.6.<33--" -> "[3.--3--][3.--3--][3.--3--][3.--3--][3.---3---]"
```

The distinction of replacement-window and context opens up another option which might seem counter-intuitive at first: it allows breaking up a long rewrite string into several components by moving parts of it into the suffix first and then consecutively matching all components while moving the adjacent segments to the context as needed. This approach is less efficient with respect to the amount of match-runs required, but it can be advantageous with respect to realtime-behavior and variability. The longer a rewrite string gets, the more difficult it becomes to make sure that no conflicting change occurs after the match has already been made. A simple example is a segment change: if a match is made across a segment border before a switch tag has been set, a conflict will arise when the switch occurs. Either matches across segment borders must be prevented, or replacement windows must be kept short enough to guarantee that matches never hurry ahead so far as to obstruct an interactively initiated change. Breaking up a

long rewrite-string into components – which often can be reused in different contexts as well – has the additional advantage making it easier to recombine components with a high degree of variability (see below).

Complementing notation by tags

While the notational extensions made available in the VGG rewrite-system allow for many ways of addressing features of musical patterns selectively, these are necessarily bound to the feature-set available in the event-chain, which in turn mirrors the features made available by the musical notation in use. This restriction can quickly become limiting. For example, there is no established way to express *irama*, *pathet*, form or metrical position, although we know that these factors are significant for the disambiguation of keys. We could easily imagine other, less obvious factors (e.g. tradition/style, personality, mood, performance context), which could also influence pattern-choice, but which cannot readily be expressed in notation.

The VGG rewrite system tries to offer a place for all these aspects in a unified, flexible tagging system. It allows dividing rule-sets into classes distinguished by tags or bundles thereof and thus can distinguish contextually determined outputs of the same key, as well as dramatically reduce the number of rules to be tested in a single iteration. It is a free tagging system, yet it also allows the nesting of rules, providing an efficient way to express hierarchies. The tagging system extends the representational potential of musical notation into the semantic sphere. Anything that can be expressed lexically or by relations among lexemes can be made a tag and thus become part of the selection criteria in the rewrite system.

We have already mentioned the most obvious criteria for the selection among rules: form and *irama*. These two are special in that any rule needs to explicitly define at least one of them, and most rules require both. The current implementation, next to 'form', also adds the category 'segment' to distinguish forms of the same interpunctuating structure but differing performance modes, e.g. *ladrang* or *ketawang* vs. *ompak* and *ngelik*. We would assume *pathet,* and therefore *laras,* are also obligatory attributes, yet this is less straightforward than one might expect – some rules for some parts can be formulated in a *pathet*-independent manner. It will be interesting to see to which extent the traditional cate-

gorization (6 *pathet*, three in each *laras*) provide sufficient and unequivocal criteria for rule selection in pattern-matching.

Some aspects of musical performance cannot have any impact on pattern matching in the current implementation. The most notable of these are loudness and micro-rhythmical aspects which become effective after parts have been generated in the rewrite-system. While the exchange of musical signals, *irama* and significant base-speed changes can become part of pattern-matching, volume and micro-rhythmical speed variations cannot take effect in the present design of the rewrite rules. This might be considered a shortcoming if one assumes that any distinct feature of musical performance has the potential to become significant in part generation. However we know of no case within the current scope of VGG where loudness or micro-rhythmical variations are not directly connected to other phenomena already explicit on the symbolic level. Therefore they do not introduce any new information significant in determining part generation. Should this change, similar classes like that of `Pirama` would have to be introduced together with appropriate matching rules.

Variability, disambiguation and randomness

Another important aspect of part generation is variability. Variability is an interesting musicological problem because the degree of variability we observe largely depends on our own sophistication as observers. Specifically, it depends on our ability to disambiguate patterns of comparable distribution. Within the scope of VGG, the problem of how to handle variants ultimately boils down to deciding what degree of disambiguation to integrate into the system. Both on the level of notation and tags we have to choose the degree of detail appropriate to our requirements and knowledge. If we increase notational precision without additional distributional criteria, the amount of variants will rise.

Logically “disambiguated variants” are not variants any more, and if disambiguating factors are supplied they are not treated as variants by the system. It is well known that the perceptive filters that identify musical equivalence are learned to a substantial degree. This doesn’t mean that the – by whichever means – observable differences between two entities conceived as identical are not perceivable at all, but perception might be less sensitive to the difference – and if the difference is perceived, it is not considered distinctive. Such non-distinguishing differences could be either ignored or presented as essentially random.

An ideal treatment of variability would require a consistent degree of descriptive precision with respect to data representation (which requires a theoretical base from which to define this consistency) and a reasonable account of how musical distinctions are drawn within the respective music-culture. Variants would then be cases where entities distinguished by data-representation have identical distribution.

In the study of central Javanese gamelan music, and perhaps the cultural sciences in general, it would be naïve to expect studies to achieve both the consistent descriptive precision of a symbolic system, and a recognition of distinctiveness as defined by the culture, and – to add another dimension – the individuals making up that culture. We don't know of any approach to objectify or quantify musical difference, and culturally defined distinctiveness in music must be seen as highly multifaceted and dynamic. Research should be seen as an attempt at moving in this direction. Our hope for the VGG implementation is that it will become a tool to support this process.

A framework for modeling difference on both levels should be detailed down to the physical level. Advanced audio-synthesis programs are probably some of the best available tools for this purpose. The question will be how to make use of the potential both epistemologically and syntactically within a system ultimately aimed at the description of culture. The notation used cannot claim conclusiveness in that respect, its benefits derive from the fact that it is developed within – and links to – a rich tradition of discourse about the culture under study. The freely extendable event system was chosen in order to avoid unnecessary constraints on potential differentiations and allow literal language to complement notation. It comes at the price of potentially complex data structures, which can become difficult to maintain and potentially long evaluation cascades, difficult to process in realtime.

In most scenarios, our treatment of variability will be more modest. We will come across variants with identical distribution relative to the degree of differentiation required for our central purpose. There may be distinctive factors, but we feel legitimized (or forced) to ignore them. For these cases the rewrite system offers an efficient notation to store the variants side by side, and to have the system choose randomly among them on each instantiation of an association. Such variants can be written as `Sets`:

Figure 16: Example of an association using a set in the rewrite string to allow for random choice among equivalent variants.

```
"33--" -> Set["3.3.[3.-3-]3.[3.--3--]3.3.[3.-3-]3.[3.--3--]",
              "3.3.[3.-3-]3.[3.-3-]3.[3.-3-]3.[3.-3-]3.[3.-3-]",
              "3.3.[3.--3--][3.--3--][3.--3--][3.--3--][3.-3-]",
              "[3.--3--][3.--3--][3.--3--][3.--3--][3.---3---]"
             ]
```

Sets can also be used on the key-side of associations, where a match is valid for *any* of its members. This can be an efficient way to broaden the scope of a rule and is another way to formulate many to many relations efficiently:

Figure 17: Example of an association using a set in the match-key to efficiently add several keys to one rewrite rule.

```
Set["33--", "3<-33-", "3<--3-"]
     -> Set["3.3.[3.-3-]3.[3.--3--]3.3.[3.-3-]3.[3.--3--]",
            "3.3.[3.-3-]3.[3.-3-]3.[3.-3-]3.[3.-3-]3.[3.-3-]",
            "3.3.[3.--3--][3.--3--][3.--3--][3.--3--][3.-3-]",
            "[3.--3--][3.--3--][3.--3--][3.--3--][3.---3---]"
           ]
```

Pattern-components, variables and data-maintenance

By now it is probably clear that data maintenance within the rewrite system can be challenging: adding variants alongside definitions of their distribution and refining descriptive granularity of notation[24] can quickly create large data-bodies with a complex structure. At the same time many rewrite strings consist of concatenations of subsegments or components which can be highly repetitive and/or occur in many different contexts.

---

[24] Most of the transcriptions of patterns in publications are deliberate simplifications representing sets of real-world patterns with slight melodic variations. The knowing reader can usually derive real world patterns from such a representation without difficulty. As long as variability and the specific shape of such variants is not within the focus of the author, this shouldn't be a problem. As the notation used to generate parts in the VGG implementation ultimately forms the base from which sound is generated, the notational granularity will have to be finer than in common practice.

Variables can be very useful in controlling some of the complexity and notational redundancy. We can store musical notation or functions[25] under a meaningful variable-name and call them by that name anywhere at any time. If at some point we find that our data-representation requires some refinement or correction, we just do that once in the variable assignment. As a consequence the change will automatically propagate to all places where the variable is called. Using variable names rather than notation for the representation of musical patterns is far less error-prone and significantly eases data maintenance. Additionally, and maybe most interestingly from an epistemological point of view, it allows us to employ existing, or introduce new suggestive terms to address musical patterns. This means we can use semantically motivated terminology, and query the content of each term by calling the current value of the variable at any time.

Figure 16 presents a transparent example to demonstrate some ways to reduce notational redundancy. Reformulating that association in the way outlined above could take the following steps (in slightly simplified SC3 syntax).

- create a dictionary
  `g = ();`
- make a branch for atomic components, headed by a meaningful name. The term *gantungan* (lit. 'hanging') is used to refer to repetitive patterns remaining on the same tone
  `g.gant = ();`

[25] The dynamically typed object-oriented design of SuperCollider allows to meaningfully store anything (any "object") in a variable. If we store a function rather than the result of a function-evaluation, we can trigger the evaluation of that function each time the variable is called. If the function constitutes a variable generation-principle (if it contains some kind of randomness), it can create different results at each evaluation. If we store a `Set`, we can ask for a randomly selected member of the set each time we call the variable:
Assign a set to a variable: `~mipilIr1Sl6 = Set["27.5.- 6.6.7.6.", "27.5.-6.7.-6.", "27.5.- 6.7.--"];`
Retrieve any of the three set-members: ~mipilIr1Sl6.choose;

- add the components for later concatenation

```
g.gant.compA = "3.3.[3.-3-]";
g.gant.compB = "3.3.[3.--3--]";
g.gant.compC = "3.[3.--3--]";
g.gant.compD = "3.[3.-3-]";
g.gant.compE = "[3.-3-]";
g.gant.compF = "[3.--3--]";
g.gant.compG = "[3.---3---]";
```

- concatenate components in the various known ways to fill a rewrite string of required duration. Express repetition by the method `dup(n)` 'duplicate(number of duplications)'.

```
g.gant.versA = g.gant.compA ++ g.gant.compC.dup(2).join
g.gant.versB = g.gant.compA ++ g.gant.compD.dup(4).join
g.gant.versC = g.gant.compB ++ g.gant.compF.dup(3)
               ++ g.gant.compE.join
g.gant.versD = g.gant.compF.dup(4) ++ g.gant.compG.join
```

- create a `Set` of variants:

```
g.gant.variants = Set[g.gant.versA,
                      g.gant.versB,
                      g.gant.versC,
                      g.gant.versD
                     ];
```

- use the `Set` in an `Association`. For each match one of the variants is randomly chosen:

```
"33--" -> g.gant.variants;
```

This might not look very attractive at first sight. Yet with rising complexity of data structures and an increase of the frequency with which atomic components can be reused in different contexts, it can increase clarity. It could also be used as formal base for the development of terminological paradigms to address pattern-segments.

Pattern matching and interaction

Parts generation by means of VGG's rewrite system is so far based only on *balungan*. This approach draws on a long tradition of theoretical attempts to assess or model the internal build of the rich texture present in gamelan music. Centering a description of this texture around *balungan* is a strategy employed from the earliest to the most recent descriptions of this music. This despite the

fact that the approach has come under heavy attack in the post-Kunst era, in which both scholarly and practical study of gamelan music became very popular, especially in the United States, and with important contributions of Javanese both in the US and in Java.[26] The clearest reflection of the role of *balungan* in the conceptualization of this music is the Javanese notation system, which represents a *gendhing* by writing down its *balungan* only, accompanied by some additional information like *pathet* ('tonal mode'), form and name. In view of the strong opposition against a "*balungan*-centric" perspective in parts of the more recent discourse about gamelan music, it might appear surprising that this *balungan*-centric discourse is quite common in Java as well, both in writing and teaching. In Java, this approach appears to be less of an issue, probably because of a more pragmatic perspective, in which the model is used as a tool, e.g. to teach students principles of parts generation in practical classes. Use of *balungan* does not necessarily imply its function as a starting point of a musical derivation or variation process, or that it constitutes the mental representation of a composition in the musical mind. *Balungan* could simply be seen as a compact symbolic representation of a richer musical experience taught by means of practical examples and verbal explanation. If seen as a symbolic system rather than a musical agent, one might approach the question whether the entire texture of a performing gamelan group could be derived solely from *balungan* in a less charged manner. While the criticism of a *balungan*-centric perspective has greatly enriched our discourse on gamelan music, the question of how much of the musical texture could be explained by evaluating *balungan* only is still legitimate, and the choice of *balungan* as the most prevalent basis for part generation appears to be justifiable. Attempting to advance the descriptive power of a *balungan*-based rewrite system, and pushing it to its limitations should be a highly revealing way of obtaining insights into both Javanese gamelan music and the power of a rewrite system as a descriptive tool.

This said, VGG's rewrite system is by no means limited to *balungan* as input pattern. In fact, any part can serve as input and be evaluated meaningfully, provided a descriptive set of rewrite-rules is given. Part-derivation from parts other than *balungan* is very compelling, because it could constitute a highly revealing model of musical interaction. If somewhat unspecific, it is nevertheless a gener-

---

[26] Sumarsam (1975), who introduced the term *inner melody*, inspired a lasting discussion problematizing the central role of *balungan*. For an overview and discussion see Perlman (2004).

ally accepted view that *sindhénan* (female solo singing) is heavily guided by *rebaban*, that *gendèr* players sometimes choose patterns inspired by *sindhénan*, that *ciblon*, *bonang imbal* and *gendèr* with *laku wolu* (lit. 'step eight', ≈ 'double density') *céngkok* have tight interconnections and are likely to inspire one another in a spirited performance.

It is also generally acknowledged that *bonangan* plays an important role as guide of the *balungan* when the playing style *mipil* is used. This melodically highly redundant playing style allows *balungan*-players to derive much of their part in realtime just by listening to the anticipating patterns of the *bonang*. The *bonang* also plays an important role in fine tuning and reinforcing agogic fluctuations in some forms and playing modes. So turning around the current rewrite logic and deriving *balungan* from *bonangan* appears a very appealing model to construct, which – as opposed to the examples given above – would not be particularly difficult to implement. In fact, this would be a very interesting experiment to make, because it has not yet been made sufficiently explicit to what extent *balungan* is derivable from *bonangan mipil*. Most of the time, derivation is very simple, following a logic analogous to `Set[aba-abab, aba--ba-]->ab`. Yet occasionally the *bonang* uses patterns where derivation is ambiguous: `"a.a.[a.-a-]a.[a.--a--]".dup(2)` could be played alongside `"aa–" or "--a-"`. In other cases it may even appear impossible to derive *balungan*, because *bonangan* uses tones not present in *balungan*: `21.5.5.5.1--5.15.-6.6.1.6. -> 2126`. In practice though, musicians can usually derive their *balungan* safely in the latter cases too. Of course it is hard to say whether – at any particular moment – musicians derive their part, or whether they simply play from memory. Yet anecdotal evidence indicates that there is an intuitive capability to derive *balungan* in realtime even in the latter cases. This is in fact generalizable, because *balungan* – like other parts – has idiomatic *céngkok* and could be described as an informationally redundant, prototypically functional bridge between goal-tones. If we look at the above *bonang*-pattern `215.5.5.1`… it is important to know after only the third tone (5.), it is safe to say that the entire pattern will end on 6. Therefore the derivation of the appropriate *balungan* boils down to the question of how to reach 6. in a way idiomatic for *balungan*, and whether – in a specific context – there is only one answer to this question. Given such a procedure, it would be possible to formulate a rewrite-rule, specifying the appropriate context in prefix and/or suffix that allows deriving `2126.`

from the discussed *bonangan* pattern. Formulating this rule as a sort of hypothesis, and then applying it to a large pool of real musical contexts should provide a fair test of whether the result describes musical practice adequately.

Of course the real situation is more complicated than this brief discussion indicates. Real world *bonangan* is more flexible than the examples show, and there are quite a few cases of unusual *balungan*. Yet the redundancy both in *bonangan* and *balungan* is worth considering, and a careful study of the relation between *bonangan mipil* and *balungan* should reveal that quite a lot of *balungan* can be derived from *bonangan mipil* even in cases where the simple `aba-abab -> ab` logic does not apply. The implication is that the apparent redundancy in Javanese gamelan music makes it possible to actually learn repertoire while participating in a performance.

The main purpose of this digression was to demonstrate that implementing the derivation of *balungan* from *bonangan mipil* by means of a rewrite-system, which in turn is part of an audio-synthesis system, yields a very promising tool with which to test a scholarly hypothesis. The act of implementation presents a series of specific, detailed puzzles. The solution (not the workaround) to each of these puzzles could be seen as a formal encapsulation of a little music-theoretical statement. Because the result can be played back, we are able to then ask competent musicians for an evaluation based on listening only.

While appreciating the potential benefit of such a system, we should also have a look at the difficulties it presents. Solutions that take the more conventional approach of deriving parts from *balungan* – rather than the opposite – initially appear as the only viable option. *Balungan* is a comparatively compact part,[27] where many hours of music can be notated with a small amount of data. It is readily available in many editions, including digital formats, and could be transcribed quite easily from recordings when necessary. While there are *balungan* variants for quite a few pieces, it is undisputed that the version agreed upon in a performance is well defined and reproducible. Simultaneous use of different *balungan* versions in a performance is never intended.

---

[27] While balungan is very compact it could still be seen as quite redundant. It would be an interesting experiment to try to identify the minimum amount of information required to generate everything that can be generated from *balungan*. While this "part" doesn't exist, it should be possible to derive *balungan* from it.

This cannot be said about other parts. While there are a few transcriptions of *garap* parts, most of them from ISI Surakarta, their coverage – except for the *rebaban* books by Djumadi[28] – is not comprehensive. Most studies don't simply write down a part, but instead present the playing style as a general collection of *céngkok.* The task of applying the material to specific musical contexts is a task left to the reader. A description of the principles of their application is usually not part of the publication. A collection of *céngkok* with undisclosed application principles could thus only be used as a source from which to build the rewrite-system if the rules are added. This brings us back to the conventional approach of deriving parts from *balungan* by combining rules and *céngkok*.

There is a tradition of presenting a part as a collection of *céngkok*, and there are good reasons to do so, even if it carries some strong limitations. Some *céngkok* appear very frequently, making writing them out explicitly on each occurrence very cumbersome. The verbosity involved in working this way might be a requirement for certain categories of study. For example, a project that focused on variability or on minor adjustments made to *céngkok* at joints might depend on detailed descriptive transcriptions. Conversely, the studies we have mentioned generally tend to have a more prescriptive stance[29] and attempt to present a part as concisely as possible. This is not only a scholarly challenge; it is also preferable in a discursive context where readers often already have mature conceptions of how *céngkok* are applied. In this context, writing out entire parts by reproducing the *céngkok* notation each time would only obscure the text and discourage the reader. A viable alternative is to present musical compositions as chains of *céngkok* represented either by names or other short symbolic representations. This approach is quite common in informal gatherings and classes, yet to our knowledge has never been published in comprehensive studies covering larger portions of repertoire. There probably are several reasons for this: the approach could be seen as too flat, or as exposing aspects of this music that should better be studied in actual practice or in face to face communication between teacher and student. Last but not least, experts can reach such a high degree of fluency in interpreting *balungan* that the *balungan* itself suggests *ga-*

---

28 Djumadi (1976-83, 1982)

29 A notable exception is the *siter* study by Sigit Astono (1990), in which the *siteran* of three well reputed *siter* players of contrasting musical background is transcribed for five popular *gendhing*

rap as succinctly as the chains of *céngkok* names proposed above. This expertise is rarely made explicit. More often it is taught through a process where over-generalized statements[30] about the applicability of *céngkok* to *balungan* – or the principles of concurrence of *balungan* and *céngkok* in other parts – are gradually revised and complemented by more specific ones. In this process, the scope of rules gradually gets blurred – it is usually left open whether a more specific rule is bound to the individual occurrence where it was introduced, or whether it should be applied to all instances of comparable[31] *balungan*. This is a shortcoming for the theoretically minded, but the question is often of little relevance in the classroom, which comprises only a small part of a student's exposure to *garap.* Their pedagogical function in this context could be viewed primarily as a means of encouraging careful attention to interesting features of *gendhing*, rather than providing conclusive explanations of *garap*. The more advanced *garap* questions become, the less adequately they can be answered in terms of wrong or right, and the more they shift into the realm of individual preference. Likewise, musicians differ considerably in the degree of standardization they consider appropriate – some have strong views how things should be played, while others are considerably more permissive or appreciative of variety.

Material derived from parts other than *balungan* is difficult to retrieve, far more variable than *balungan* and often highly redundant. This makes them substantially more difficult to use as a starting point of part generation. Choosing the conventional *balungan*-centric approach has the added benefit of integrating well with scholarly discourse from ISI Surakarta. *Balungan* centricity might be questionable as a model of the mental representation of gamelan music, but it is

---

30 This is a deliberate simplification. In the beginning, students are taught without any reference to scope, and the genericness of céngkok is something that only gradually reveals itself in the learning process. Yet on an advanced level, the process resembles the way described above: students can easily create a tentative part by applying a few general rules and are taught more specific or sophisticated realizations on that base.

31 Comparing instances of *balungan* makes no sense without considering contextual factors, among the most prominent *pathet* and melodic context (preceding and succeeding *balungan*). Additional factors include also the position within gong-phrase or metrical weight and form. The impact of the latter two factors is less well rationalized, likewise the term *kalimat lagu* (lit. 'tune sentence') which is occasionally used in Javanese discourse on such matters. As factors contributing to the identity of a certain *balungan* phrase can be quite manifold, it can be difficult to ascertain whether two *balungan* instances with identical tone-successions are identical with respect to *garap*.

an attractive foundation for studies of *garap*, and well represented in Javanese discourse on the matter. Following this course will reveal as much about *balungan* itself as it will about *garap* parts.

Using parts other than *balungan* as input for the rewrite system is also relevant on the level introduced at the beginning of this chapter – some musicians having an impact on the performance of others in realtime. Musical agency inspires variation and controls the course of a performance. Melodic interaction – a *pesindhèn* following a *rebab*, a *bonang* associating with a *ciblon* or vice versa – is fundamental to advanced performance. Modeling this interaction with the current VGG framework is not trivial: realtime constraints are even more rigid in parts which have basically been derived in the very moment. The time span allowed for recognition-based reaction must be very short. As *balungan* is preexistent, we can make the key-size as large as required.[32] However, if pattern recognition is used to model interaction, information can only be evaluated once it has become audible. This means there is a clearly defined and sometimes very short time span between the recognition e.g. of a melodic signal and a potential reaction.

In order to derive results from pattern-matching beyond basic rewrite-strings, we use the class `Preact`. It also uses `Tafsiran` as pattern-matching component, which means that it can recognize patterns based on the same symbolic representation of match-keys, and it can evaluate tags. Yet while the rewrite system returns strings, `Preact` specifies functions. Therefore `Preact` can act as an external agent to the system, doing things like setting or removing a tag (to be evaluated in the rewrite-system), or triggering actions like switching to another segment or speeding up and down. A sophisticated model of interactivity should also allow for shifting attention, modeled as a switch between different input melodies, or even divided attention, monitoring several input-melodies in rapid alternation and choosing the most relevant one in each instant. More differentiated interactivity in part generation thus remains a challenge to be explored in future experiments with the system.

---

32 There is a pitfall though: if a key matches cross segment-borders and a segment-switch is initiated after a match was made problems will occur.

Micro timing: interaction and jitter

Interaction is also an important factor contributing to finer aspects of musicians' timing. While it is generally acknowledged that the drummer is in control of speed and speed changes, our model should not depend on absolute control. In a performance musicians don't know in advance when a transition or tempo-adjustment is going to occur, nor is the profile (duration, gradient, and curve) of speed changes precisely predetermined. Even though in many performances, conventions are followed about when and how a transition should occur, all players will wait for the drummer to take the first step. While a drummer may give signals to indicate the ending of a piece or to anticipate the switch to a different drumming mode, speed changes are initialized just by carrying them out, and are not signaled in advance. Control that permeates as change in sound requires a bit of time to become effective and not every player will necessarily react in exactly the same way. The musicians' reaction to an initialization of speed changes also has some influence over how a particular transition will ultimately be executed. The feedback of rhythmically salient instruments like *peking* (highest metallophone with a very constant stroke-density) and *bonang* probably plays a more important role than that of others.[33] At the start of a piece it is usually not the drummer who sets the initial speed, but the player of the melodic introduction *buka*. The drummer may adjust that speed if desired, but the change should be smooth and everybody should be able to follow.

The VGG implementation attempts to capture timing as a result of mutual adjustment by allowing each stream of independent events (which usually represents the sequence of actions of one player) to have its own temporal frame of reference. Timing negotiations between players can be very intricate, because the judgment of the others' tempo (and one's own) depends on many different factors which are not easy to formalize. To experiment with a system where

---

[33] One of the most distinguished musicians of Solo, Bp. Suyadi, the former leader of the RRI ensemble (Radio Republik Indonesia Surakarta), claimed in a lesson that (even) the *rebab* could guide the long initial transition at the beginning of a *gendhing* and advised to play *rebab* there in a rhythmically plain, less ornamented and delayed manner. *Rebaban* is one of the most prominent parts in a gamelan, but one wouldn't count it among the rhythmically salient instruments because it is often not heard well by all players. While the musician's statement would probably be disputed by others, it shows that sound characteristics (rhythmic salience) and social factors (role as leader) contribute jointly to the dynamics of playing together.

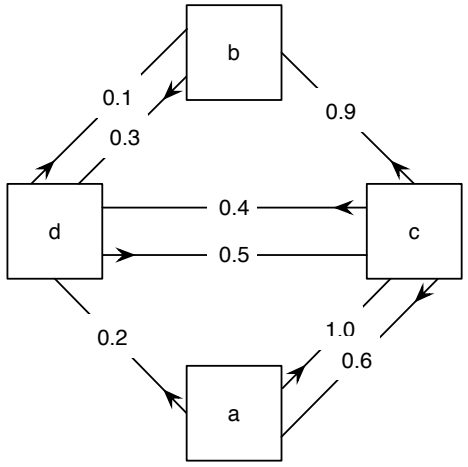
timing is *negotiated* between the players, the class `ListeningClock` abstracts from individual tempo judgment and instead makes it easier to reason about a network of interacting frames of time reference. The intention is to allow experimentation with tempo adjustment without assuming a single master clock, and with the option to distinguish between different listening and response behaviors. In such a network, every player may take into account any number of other players to any degree. The degree to which they listen is defined as a list of weights between 0 and 1, where 0 means not listening to the respective player and 1 means only listening to this player[34] (see Figure 18). Thus, a listening clock can temporarily loosen the synchrony between the clocks of a couple of instruments during speed changes. When a transition returns to a stable speed the clocks will be resynchronized as well. The instruments participating in such mediated timing relations can be chosen freely, and relations can be chained. For each relation the parameters can be set independently. Rather than a mature model of interactive timing in central Javanese gamelan music, this implementation should be seen as a framework for experiments and development; an initial attempt to address the issue (see fig. 18).

In the current implementation, we simplify the situation by assuming that every musician knows every other musician's *tempo* and *phase* (the relative timing offset) at any moment. Every player may 'listen' to every other player and adjust his own playing according to: (1) a weighted average of the others' behaviors, (2) a playing attitude, denoted by the two parameters `empathy` and `confidence`. The algorithm takes into account two possibly contradicting exigencies: (1) the tension between one's own consistent speed and the other players' speeds, (2) the tension between achieving contiguous playing and remaining in time with the rest of the ensemble. The first is related to tempo, the second to timing offset, to phase. In figure 18, we give a more detailed account of the simple algorithm that runs at a given rate in each clock.

---

[34] These values may be changed at any time during the performance. Future extensions of this clock network could take the current context into account and vary these values accordingly.

Figure 18: A small network of clocks listening to each other. The arrows depict weights of mutual influence. An external change to one of the clocks results in the proliferation of this change in the ensemble.



First, each clock calculates $\Delta\theta_{avg}$, the (weighted) average of the phase differences between its own phase θ_{ref} and the phases of the clocks it is listening to. It also calculates the average tempo φ_{avg} of those other clocks. In a next step, the clock derives its new tempo from these values. The clock's parameters `confidence` and `empathy` determine how the ensemble's average phase difference and tempo influences the clock's new tempo:

Equation 1 $\varphi_{new} = \varphi_{avg} \cdot (1 - confidence) + \varphi_{old} \cdot confidence + \Delta\theta_{avg} \cdot empathy$

where φ_{avg} is the average tempo of the rest of the ensemble, and $\Delta\theta_{avg}$ the average phase difference. φ_{old} and φ_{new} are the previous and the new tempo of the currently adjusting clock.

The parameter `confidence` is intended to denote the confidence in the relevance of one's own tempo, so that high confidence entails a tendency to insist on one's own tempo. The parameter `empathy` models the willingness to adjust one's phase, "to be with the others". It is easy to understand the influence of these parameters (which both range between 0.0 and 1.0), if we consider extreme cases. For instance, a fully confident player without empathy would simply remain unaffected by, and thus out of sync with, all the other players. Such extreme cases provide a good starting point, since the tempo calculation formula

above becomes greatly simplified. Let's first consider this case of the player where `empathy = 0.0`. As a confident player without empathy, one may be thought to remain in one's own tempo despite changes in the ensemble (c = 1.0, e = 0.0):

Equation 2 $\varphi_{new} = \varphi_{old} \cdot 1.0$

Conversely, when both one's empathy and one's confidence are minimal, one would not remain with one's old tempo, but immediately jump to the average tempo of the rest of the ensemble (c = 0.0, e = 0.0):

Equation 3 $\varphi_{new} = \varphi_{avg} \cdot (1 - 0.0)$

Such a player will always follow tempo changes, but will be indifferent to whether they play in sync with the others; there is no adjustment for phase differences, but only to tempo differences.

Now, the second parameter, `empathy`, is the degree to which a player reacts to phase differences. In other words, while confidence is tempo-oriented, `empathy` is synchronicity-oriented, so that high empathy will cause us to follow other players very closely, low empathy will cause us to ignore phase differences, even if we follow their speed meticulously.

For instance, imagine a player guided mainly by an awareness for where in the piece the others are now, and less by his own tempo. In this case, the contiguity of his own playing will be reduced, and adjustments to the ensemble quick. This situation is extreme when confidence is minimal (= 0.0) and empathy maximal (= 1.0). In the above formula, the influence of the player's own tempo can be left out, so that only the average ensemble tempo and the average phase difference have an effect (if the ensemble is faster, but behind – so that $\Delta\theta avg$ is negative, the new tempo is not necessarily higher than the old one) (c = 0.0, e = 1.0):

Equation 4 $\varphi_{new} = \varphi_{avg} \cdot 1.0 + \Delta\theta_{avg}$

If, however, we decide to play both maximally confidently and emphatically, our own tempo will not adjust very much. This is not because we note the ensemble's tempo changes, but because we try to play on the others' beats. This

situation shows a maximum tension, and when all players of the ensemble listen to each other in this way, the ensemble may diverge, oscillate, or become chaotic. The simplified formula then looks like this (c = 1.0, e = 1.0):

Equation 5 $\varphi_{new} = \varphi_{old} + \Delta\theta_{avg}$

We can easily imagine that transitions can work with reasonable grace when the players have a well adjusted balance between confidence (considering each others' tempos, and their own), and empathy (neither jumping on the others' beats, nor ignoring their current phases). The way an ensemble slowly adjusts to tempo changes depends on the details of all these behaviors and to what degree each player listens to other players.

On the level of micro-timing, VGG provides a separate simple algorithm that may be applied to every stream: the class `Ptiming` allows control of systematic delay and micro-rhythmic variation (parameters that normally do not compromise part synchrony) by means of several parameters. This pattern is a starting point where further research may be done in order to refine the relation between musical context and micro-timing. In the current implementation, `Ptiming` takes into account four parameters of micro-variation: *offset*, *jitter*, *drift* and *driftratio*. Generating numerical values, it can be applied to any event stream's `\lag` or `\offset` field, which serves as a phase offset, independent of both the events' inter-onset time, and the clocks' tempo variations. All variations are generated on the basis of SC's pseudo-random number generator, which is fast and reliable, with a very long period length.

The parameter offset (in `Ptiming`) is a constant timing offset added to each value to account for a player consistently playing slightly before (negative value) or after the beat (positive value). This may be specified as dependent on specific roles for certain instruments, or on a player's personal style, in order to model the specific roles of certain instruments in a given piece, or simply to adjust for slower attack characteristics of some instruments (if not corrected by the player).

The second parameter, *jitter*, allows us to specify the micro-variations in timing that are independent of past events and speed: it accounts for a constant, physical or deliberate inaccuracy of playing.

Finally, *drift* and *driftratio* define a level of timing that may be described as standing somewhere between the previous two: while *offset* is a constant difference, and *jitter* a new random aberration from the beat given by the system, *drift*

is an incremental change of offset. Each new value differs a little from its previous value. In other words, being a random walk, consecutive values depend on each other. Similar to the tempo movements of `ListeningClock`, its effect can be taken as a constantly corrective and adaptive movement (yet much more simplified and on a micro level). Here, the parameter *drift* determines the maximum total drift from the offset points, and the *driftratio* the speed at which this drift changes. In difference to the continuous co-adjustment of the clocks, this algorithm accounts for an errant, independent aspect of alteration that is not captured by the former. Together, these four parameters frame a simple dynamics of micro-temporal variation that supplements the dynamics of tempo changes and rhythmical variation, for each stream independently. Further research efforts into this aspect of timing can be easily formalized and integrated into the system (and thus into the entirety of the experimental setup).

Audio-synthesis

Actual sound is finally created by sending the information derived from various resulting event-streams to either a sample player or an audio synthesis unit. This is preceded by applying a few necessary basic physical parameters like volume/balance and spatialization, which can partially be controlled from a simple graphical mixer.

The sample player currently uses a basic set of samples made from the gamelan of the Arts University Graz as part of the VGG project. Using samples has various advantages: we use tuning and sound recordings of a given ensemble and achieve a relatively realistic sound without putting a high burden onto the system. This is sufficient for many applications of the system: testing parts generation, experimenting with speed, *irama*, transitions and the like. Yet a sample-player does not allow detailed control over the physical parameters of each sound event. Without loss of precision, recorded sounds can only be manipulated as atomic units. To give a simple example, direct frequency modifications would be applied to all sound components alike and therefore change durations proportionally.

Once the sound itself moves into the center of attention, we will want to access its individual components independently and thus use the audio-synthesis module. Currently, the available measured data contain information about frequency, amplitude, phase and temporal behavior (attack and ring time) of the

partials in all samples in the KUG gamelan.[35] We have structured this information in a nested and *relative* manner: the timbre of each key or gong of an instrument is represented as a collection of these properties. These collections in turn are combined into one collection of tones (keys or gongs) per instrument. Frequencies are stored not as independent values but as ratios. Starting from the lowest pitch and nested according to the structure described above, all frequencies are stored in a relative manner. As a consequence, we can determine the scope of experimental detuning by choosing the appropriate nesting-level. The frequency of a partial ($f_{partial}$), for instance, can be derived directly from the product of the ensemble's root frequency (f_0), the ratio of instrument frequency to this root, (c_{instr}) the key's ratio to the instrument (c_{key}), and the ratio of the partial, relative to the key ($c_{partial}$):

Equation 6 $$f_{partial} = f_0 \, c_{instr} \, c_{key} \, c_{partial}$$

This is only the internal representation – the parameters can either be edited directly, or by means of their respective values in absolute frequency, relative or absolute cents. Since they are constantly updated,[36] they may be changed in real-time without interrupting playback, so that many experiments with various systematic modifications are possible: For example one may detune or dampen the whole ensemble, apply a different scale to an instrument, and choose whether to change the overtone pattern or not. Or one may dampen or detune all partials in a given range of frequency for a group of instruments. What gives this feature a new quality is the fact that such experiments can be made within a playing gamelan. Changes thus are not just applied to an abstract collection of sound (like a scale), but to a changing musical context in which tonal dynamics are active.

---

35 The values have been extracted from the recordings used in the sample player with MatLab scripts programmed by Franz Zotter as part of the VGG project. For simplicity of explanation we reduce the partial-properties to three here: frequency, amplitude and ring time.

36 All parameters are read with a rate of 750 Hz, and are linearly interpolated to avoid discontinuities. They can thus be either manually adjusted during playback, or programmatically modulated. In some cases, for instance, one may want to detune an instrument slowly over a long period of time, or one may want to introduce a vibrato on a number of partials.

Recapitulation: a walkthrough

We have touched upon very different aspects and components of the framework. The system as a whole combines these components (and their respective assumptions) in a fairly uniform manner, relying mainly on the pattern classes, which are extended by some additional classes like the synthesis module and the rule object. On the whole, we may see VGG as a prototype for a connection between different assumptions about necessary competences and interactions. This prototype can be restructured and extended easily, and its parts may be refined without affecting the rest of the implementation. In the following section, we will give an overview of how its parts are connected.

Starting from *balungan*-notation and the notation-parser, a basic event-stream is created, which is subsequently enriched with required information. Some of this information has to be set manually (e.g. *pathet* and initial speed), some is retrieved from dictionaries containing generic information. Streams are processed in realtime by other streams that derive from classes generically called `Pattern` in SuperCollider. One pattern may create multiple streams, each of which can operate at the same time, either chained or in parallel. If they operate within a chain of streams, one stream receives the output of the other without having to wait for the preceding stream to finish processing the entire stream. If a pattern only processes a single tone on each iteration, the next pattern can operate as soon as this tone is modified.

The first patterns to operate on the *balungan*-stream just add meta-information required later for parts-derivation: metrical information by `Pgongan`, the current subdivision-level by `Pirama`, required tags, `pathet`, form and others. Feeding this information into the stream means that each event contains the respective information independently. Processes depending on this information don't have to look it up somewhere else and a change is immediately present as part of the event definition. Operations applied subsequently may modify the *balungan* in its notated form in order to ease part generation. The output of such operations may either replace the current *balungan*-stream, be added to the stream, or be established as a parallel stream used for the generation of different parts.

Subsequently parallel streams are created for each part by invoking the pattern `Ptafsiran` for each instrument independently. This pattern generates each part by evaluating instrument-specific rules stored statically in the dictionary of

the respective instrument. While some instruments wouldn't really require the invocation of the rewrite engine to be generated, the current implementation has chosen this path to allow for a unified logic of formulating rules for all parts.

Splitting up streams not only creates different parts playing together, but also provides a way to localize agents that operate on self-contained information sets. These agents can exchange information with other agents, but they do not see everything present in the system. By controlling consciously at which place in the tree of streams information is fed into the system, we can control the information available to an agent. If we feed information into the stream before individual parts have branched off, all parts will share this information, if we feed information into one of the branches, only parts derived from this branch will contain it. Therefore in principle it is possible to model chains of information exchange or communication going on between individual agents or a subgroup of the ensemble only.

Once generation for all parts is active, the streams responsible for identifying musical signals (derived from `Preact`) receive each stream. Identifying a musical signal means recognizing musical patterns in the event stream. The signal-identification uses the same algorithm as the rewrite engine. Any time a signal is recognized (a key matches), a function is evaluated which can change the default flow of the program. As opposed to the rewrite-system, where a match will always cause a rewrite-string to be issued, the function triggered in `Preact` could cause any programmatic activity. Common examples are setting a tag or controlling speed-changes. Therefore this pattern defines one type of the information sending agents in the system.

On the next layer two fine-adjustments to timing are made. The `Listening-Clocks` try to account for some of the interplay between musicians when speed-changes occur. We allow for the possibility that there is some reaction delay and that different parts have a different degree of rigidness adjusting to other parts. Setting the parameters for `empathy` and `confidence`, and setting up cascades of clocks, should reflect known roles and rhythmical salience of parts.[37] The resulting stream is processed by a pattern (`Ptiming`) that allows to control lag in

[37] The aforementioned pattern `Pirama` creates streams that derive tempo information from each respective clock.

a generalized manner and that adds some temporal randomness to the events reducing the over-precision of machine timing.

Before the resulting streams are sent to the synthesis-unit, a few basic physical parameters like volume and spatialization are set. As synthesis unit, either a simple sample player or an algorithmic sound-synthesis-unit can be chosen. The sample player uses a set of samples made from the gamelan of the KUG, the sound model used for synthesis works based on data retrieved from an analysis of the mentioned samples. When using the synthesis-unit it is possible to control the physical properties of each partial to experiment with sound.

Conclusion and methodological reflections

Over the course of developing the implementation, we have been aiming at a good balance between the different, but intertwined requirements of this project. While on the one hand, we had to realize and explain a working prototype; the framework should be more than a machine for automatic music rendering. To learn how to reason within such a system, and simultaneously how to construct it, requires both sensitivity to detail and limitation as well as decisions of how to structure a general discourse that does not get lost in the immediate needs of realization. As our discussion of the modeling of a *gendhing* demonstrated, basic problems are solved and the framework is suitable for further exploration beyond elementary questions. At this point, it may be helpful to reconsider some of the basic design decisions in VGG, and how they fit in a more general methodological view.

VGG has two sides: on the one hand, it serves as a generative scheme to play back pieces that are part of a repertoire. On the other hand, and more importantly, it is a framework for musicological research. While *experimental systems* (Rheinberger 1997) have been in focus mostly within natural sciences, there is no reason to preclude their application from other sciences, such as anthropology.[38] We must keep in mind though that such systems do not simply exist in isolation from research practice – rather they are born of the exigencies of explicit constraints and actual research questions, and they unfold through the

[38] The chapter *Algorithms in Anthropology* makes an attempt to provide a broader context for this subject matter.

process of resolving their ambiguities and problems. Despite a few exceptions, computer experiments are not yet an established method in ethnomusicology. In systematic musicology, and especially within cognitive musicology, this is more common, not least because of the historical importance of computation representing a model for cognition. On the side of art, the experimental application and development of computer languages has also become increasingly advanced, so that musical knowledge has become more clearly intertwined with computer languages. It is only a matter of time before different music cultures begin to influence computer music practice.

So what do we actually investigate when doing computer experiments in gamelan music? Our work with VGG and Javanese gamelan music has involved a continual reflection upon the relation between several discourses: the experienced knowledge of musical practice and competence, the emic reflection on the same, the ethnomusicological discourse, and a system that is capable to serve as a framework. A constant awareness was needed for the relation between vocabularies inherited from the different terminological cultures that have grown from their communities of practice.

We have tried to make clear that such an experimental system should not be seen as a model of central Javanese gamelan music. Rather, it should be considered a model of a given understanding of this music that allows us to explicate its underlying assumptions as well as experiment with their consequences. Maybe most importantly, this helps us to better understand the possibly unexpected dissonance between various suppositions. It cannot and need not be a replacement of any kind for the practice of performing this music, and obviously does not serve the same purpose as established means of organizing knowledge about it. This does not mean though that such as system is disconnected from its subject matter, since a computer model of our understanding of something may well turn out to show characteristic constraints and thus to shed light on possible reconfigurations of knowledge, be it by differentiation or abstraction. Following Rheinberger, it is relevant to consider that experimental systems necessarily combine what he calls *technical objects* (those parts of the system that form the canon of established techniques) and *epistemic things* (those which are related to the shifting, ambiguous concepts under investigation). In VGG, for instance, one could consider the sample playback system an established technical object, but not each of the rules that serve to generate the parts. Similarly, the way to represent knowledge of the current *sabet* may be considered mature, but the various

conditions under which it becomes relevant are subject to investigation. We should not preclude any reconfiguration of these roles though: under certain circumstances, the fixed points of yesterday shift in the focus of today's research questions. We have tried to balance 'black-boxing' with openness, a feeling of a perspicuous presentation with access on all levels of representation – yet only experimental practice will show where the most promising constellations are to be found.

Let us in the following briefly repeat and summarize a few orientating concepts that we based our decisions on. *Literate and interactive programming paradigms* have been mentioned in the introduction already – we wish to be able to combine knowledge representation and algorithmic model into one single system that is reconfigurable at runtime. The decision to include both terminology and notation established in ongoing practice is born from the necessity to provide a valid and practical extension to the methods of ethnomusicological work, and to really 'make sense'.

Apart from these general frames of reference, there are more specific design decisions. Perhaps most important among these is our focus on the high-level representation of interconnected *streams of events*. These events are the sites for inscription of knowledge (or, to be precise, they are the site of the inscription of an assumption about knowledge). This knowledge is produced in stream nodes, between which the events flow and incrementally differentiate into parallel worlds – knowledge is always situated[39] both in time and structure. The scheme that structures these situating processes is what we see in the chain of patterns in a given system, it is the static representation of a given program.

In other words, we may treat musical knowledge as situated both within a specific set of competences and in the context of a given moment. The first may be thought as an incremental differentiation of individual situated knowledge from collective knowledge. The second maintains the inherent temporal structure of musical context. By appropriately interlinking different levels of differentiation, and by injecting various degrees of retention, interaction may happen both between those levels and different moments in time. Hence, in order to

---

[39] There is a broad discussion of questions on situated knowledge and distributed cognition over the last decades. From the point of view of theory of science, see for instance Haraway (1988). On situated cognition Clark/Dave (1998) is a relevant introduction. Concerning computational systems, one may want to confer Clancey (1997) or Petric et al. (2001).

situate knowledge, one does not have to start with the assumption of an individual subject who is the bearer of knowledge. Neither do we have to assume one global archive that is accessible by such subjects. One may imagine the eventual sound synthesis to correspond to the bodily actions of each player, but the agency involved in the decisions required to perform these actions is distributed over both various degrees of collectiveness and temporal depth.

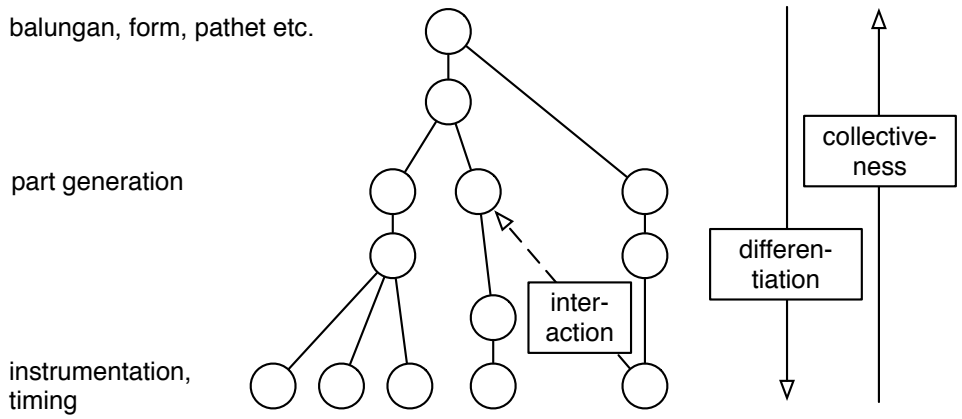
Thus, reasoning about the flow of causation in the system, we do not localize the agents of decisions in a preconceived group of entities that are to represent each individual player in the ensemble. Rather, the agents (which may be thought to coincide with competences) may have their place at any point in the field between individual and collective. This takes into account a view that conceptualizes cognition and action to be intertwined not only in each individual, but also in a collective situation: in this way, we assume various degrees of situated and distributed cognition, as to allow the researcher model subjective situations without binding action and competence to a subjectivist view of musical agency (see fig. 19).[40]

We would like to emphasize that such a system exposes only an extremely simplified silhouette of how a piece of music might develop in actual performance. On that account, notation is drawn between the extremes of writing out sequences of known variations, and narrowing the conditions under which they might occur, and the other extreme of attempting to find general underlying principles of their generation. Accordingly, the coarse rhythmical framework of binary subdivision might still underlie the musical processes leading to collective agogic fluctuations so characteristic of Javanese gamelan performances. Variability might still be considered as choice among given patterns, the principles of which we aim to understand. This last point may be considered particularly controversial. For one we would like to admit that implementing a model of musical inventiveness within the idiom of Javanese gamelan music appeared too ambitious a goal to aspire to. Yet it also seems that musical creativity in Java tends to express itself in an active command of many variants learned from different persons and traditions rather than frequent extemporization of previously unheard patterns. And maybe most importantly, while VGG aims to develop into

[40] It would, for instance, be possible to modify the current implementation as to introduce computational agents that make goal-driven decisions. But one would not necessarily have to assume that these goals are primarily individual and their collectiveness only secondary.

a generative model of Javanese gamelan music, the goal is nevertheless primarily descriptive. The model is a mode of expressing our understanding, and the resulting sound we see primarily as a tool to test our assumptions.

Figure 19: Degrees of differentiation and collectiveness in situating assumptions about knowledge and competence in an example system. While all streams are independent, they may share generative principles. Other constellations may reorder this structure according to different assumptions.



This all said, it remains one of the most essential aspects of a system like this that all reasoning with and about it is contextualized in the situation of listening to a complete ensemble in which all acoustic aspects form a complex experience; interacting with the system, modifying the system and listening to the system move close to each other, and the musicological knowledge laid down in the process is kept active and integrated in context. For instance, while the example implementation discussed here is still elementary, already all parts are connected by interaction and general rules. Consequently, listening to the system, one learns to relate what one knows about its inner logic to what one knows about gamelan performance. This process will always be productive especially by its inherent difference; it is not mainly similarity of a model that leads to insight. Simulation therefore need not be in the center of the epistemic process, but rather this multifaceted involvement in the act of listening, construction, and thought. It is precisely such a practice of experimental listening that makes posing questions to and by the virtual gamelan a captivating and fruitful activity.

References Cited

BRINNER, Benjamin E.

1995 *Knowing Music, Making Music. Javanese Gamelan and the Theory of Musical Competence and Interaction*. Chicago: University of Chicago Press. (Chicago studies in ethnomusicology).

2008 *Music in Central Java. Experiencing Music, Expressing Culture*. New York: Oxford University Press. (Global music series)

BULLEN, Geroge W.

1877 "The galin-paris-cheve method of teaching considered as a basis of musical education". *Proceedings of the Musical Association*, 4: 68–93.

CLANCEY, William J.

1997 *Situated Cognition: On Human Knowledge and Computer Representations*. Cambridge, U.K. and New York, USA: Cambridge University Press.

CLARK, Andy / CHALMERS David J.

1998 "The extended mind". *Analysis*, 58(1): 7–19.

DJUMADI

1976-83 *Titi Laras Rebaban* ('rebaban notation'). 3 vols., Surakarta: Akademi Seni Karawitan Indonesia.

1982 *Tuntunan belajar rebab* ('a guide to studying *rebab*'). Surakarta: Sekolah Menengah Indonesia.

L'ECUYER, Pierre

1996 "Maximally equidistributed combined tausworthe generators". *Mathematics of Computation*, (65): 203–213.

HARAWAY, Donna

1988 "Situated knowledges: The science question in feminism and the privilege of partial perspective". *Feminist Studies*: 14(3): 575–599.

HEINS, Ernst L.

1970 "Cueing the gamelan in Javanese wayang performance". *Indonesia*, 9: 101–127.

IVERSON, Kenneth E.

1979 *Notation as a tool of thought* (ACM Turing award lecture). Detroit: *Communications of the ACM*, 23(8).

KNUTH, Donald E.
1992 *Literate Programming*. Stanford (California): Center for the Study of Language and Information. (CSLI Lecture Notes, 27).

MEYER, Bertrand
2000 *Object-oriented Software Construction*. New York: Prentice-Hall.

KLERER, Melvin / REINFELDS, Juris
1968 "Interactive Systems for Experimental Applied Mathematics". *Proceedings of the ACM Symposium held in Washington DC August 1967*. New York: Academic Press.

KUNST, Jaap
[3]1973 *Music in Java. Its History, its Theory, and its Technique*. 2 vols. Edited by E. L. Heins. The Hague: M. Nijhoff.

PERLMAN, Marc
2004 *Unplayed Melodies. Javanese Gamelan and the Genesis of Music Theory.* Berkley/Los Angeles/London: University of California Press.

PETRIC, Mirko / TOMIC-KOLUDROVIC, Inga / MITROVIC, Ivica
2001 "A missing link: The role of semiotics in multiagent environments". *Proceedings Cosign 2001. 1st Conference on Computational Semiotics for Games and New Media*. Amsterdam: Stichting Centrum voor Wiskunde en Informatica: 108–113. http://www.cosignconference.org/downloads/papers/petric_et_al_cosign_2001.pdf (retrieved June 30th, 2008).

PICKVANCE, Richard
2005 *A Gamelan Manual: A Player's Guide to the Central Javanese Gamelan*. London: Jaman Mas Books, 2005.

ROHRHUBER, Julian / DE CAMPO, Alberto
2008 "Just in time programming". In: Collins, N., Wilson, S., and Cottle, D. (eds.): *The SuperCollider Book*. Cambridge (Massachusetts): MIT Press. (forthcoming).

ROHRHUBER, Julian / DE CAMPO, Alberto / WIESER, Renate
2005 "Algorithms today - notes on language design for just in time programming". In: *Proceedings of the 2005 International Computer Music Conference*: 455–458. Barcelona: ICMC.

RHEINBERGER, Hans-Jörg
1997 *Toward a History of Epistemic Things: Synthesizing Proteins in the Test Tube*. Stanford (California): Stanford University Press. (Writing Science).

SIGIT ASTONO
1990 *Pengenalan terhadap cengkok céngkok siteran* ('Introduction to *siter céngkok*'). Surakarta: Sekolah Tinggi Seni Indonesia Surakarta.

SORRELL, Neil
[2]2000 *A Guide to the Gamelan*. Ithaca (New York): Cornell University. (Society for Asian Music).

SUMARSAM
1975 "Inner Melody in Javanese Gamelan Music", *Asian Music*, 7(1):3–13.

WARSADININGRAT
1987 "Wédha pradangga" ('Sacred knowledge about Gamelan music'). In: Becker. J., Feinstein, A. (eds.): *Karawitan. Source Readings in Javanese Gamelan and Vocal Music*, vol. 1: 1–170. Ann Arbor (Michigan). (Translated from Javanese by S.P. Walton).