Applying aspect-oriented programming to music computing

Book Section

APPLYING ASPECT ORIENTED PROGRAMMING TO MUSIC COMPUTING

Patrick Hill
The Open University
Walton Hall
Milton Keynes.
MK7 6AA
PatrickHill@bcs.org.uk

Simon Holland
The Open University
Walton Hall
Milton Keynes.
MK7 6AA
s.holland@open.ac.uk

Robin C. Laney
The Open University
Walton Hall
Milton Keynes.
MK7 6AA
r.c.laney@open.ac.uk

ABSTRACT

Computer programs for the composition, performance and analysis of music generally involve the tangled interaction of many dimensions of musical and extra-musical concern. In this paper we introduce the concepts of Aspect-Oriented Programming (AOP) to Music Computing and argue that AOP and related techniques and technologies form an appropriate solution to the separation and composition of such concerns. We motivate our argument with simple examples from the musical domain, but argue that the underlying principles may be applied to a wide and expressive range of musical applications.

1. INTRODUCTION

Musical composition may be considered in terms of the construction of tangled hierarchies [22] in which various musical dimensions are 'woven' together to form a 'logical and coherent musical whole' [31]. This view of music is supported by the exposition of complex musical interrelationships, presented by writers such as Miranda[22], Rowe[30], Lerdahl & Jackendoff[18], Persichetti[26], Dannenberg et al.[9] and Piston[27]. These interrelationships exist at different levels of musical abstraction, relate to different kinds of musical activities, appear at different levels of detail and abstraction, and involve different kinds and dimensions of music. Despite the diversity of these authors' concerns, we will argue that the common issue of tangled hierarchies suggests that the techniques we will outline here are potentially capable of very wide application in Computer Music.

Many music software systems exist that aim to assist in particular elements and tasks of musical composition, analysis and performance. Each system effectively implements its own partial musical ontology that maps to its areas of interest. However a difficult and pervasive issue in developing computer music systems is that it is impossible to know, a priori, what dimensions of concern are required and what relationships may be established between them, and therefore what constitutes an appropriate musical representation. This is particularly true since the needs, methods, and approaches of different composers vary so widely. The situation is further complicated when these concerns and relationships may vary inter- and intra-opus, potentially requiring dynamic reconfiguration of the software system itself.

Aspect Oriented Programming [17] and related principles and technologies such as Multi-Dimensional Separation of Concerns (MDSoC) [24] are new emerging software engineering methods originating from meta-object research [37] and designed to enable software developers to manage the separation and subsequent recomposition of separately defined dimensions of concern. Moreover, one of the requirements of AOP is that it should enable non-invasive addition of concerns [3] and therefore the approach goes some way to supporting the evolution of software systems in which requirements are not, and indeed cannot, be known at the outset. To the best of our knowledge, the present project [15], is the first time in which the application of these technologies to musical concerns has ever been explored. In AOP, the term *concern* is not always clearly defined. Loosely speaking a *concern* relates to those parts of a software system, or application domain, that relate to a common purpose, goal or concept [41]. Examples of concerns in the domain of software engineering include security, persistence, and concurrency [16]. Examples of concerns in tonal music might include harmony, melody and rhythm, or the four principal decompositions of Lerdahl and Jackendoff [18].

We suggest that the AOP technique can be adapted and used to deal with crosscutting concerns in music, with an extremely wide range of uses and applications. In this paper we outline the principles of AOP and describe its application in addressing illustrative problems that exist in various domains of computer music. For simplicity and clarity of examples, we will mostly (though not exclusively) give examples from tonal music: However the principle and the technology apply much more widely to music representation and composition in general.

2. SEPARATION AND COMPOSITION OF CONCERNS IN MUSIC

Generally, music may be viewed as being composed of a finite set of musical elements that are structured and manipulated in various ways by the composer in order to form a logical and coherent whole.

While the output of a compositional process, in the tonal case, is often expressed as Common Practice Notation (CPN), composers typically do not think in terms of 'dots on pages' or detailed note-lists [23], but rather in terms of higher-level structures, such as rhythmic motives, melodies, tone rows, harmonic progressions, orchestration and so forth. The resulting score, CPN or otherwise, may be viewed as representing the results of the composer's detailed weaving together of these various dimensions of concern, and given the extreme diversity of compositional approaches, composers may wish to establish relationships of arbitrary complexity between any set of musical dimensions.

When a piece of music is realised, interpretive practice introduces yet more dimensions of concern. For example, phrases may be articulated through changes in tempo and dynamic that are not notated. When a piece of music is to be performed by computer, these performance details must also be defined, procedurally or declaratively.

One of the key goals of AOP is to help in avoiding *tangling* and *scattering* [17] by enabling concerns to be separately specified and subsequently composed in a coordinated way. Broadly speaking, scattering refers to situations in which the realisation of a given concern is distributed throughout other concerns. The related concept of tangling refers to situations in which concerns are not properly separated, but instead an artefact implements multiple concerns.

It is important to note that our proposals do not seek to impose any particular philosophy or working methods on composers, analysts or performers. Indeed a key aim is to avoid any such impositions, as should become clear below.

We now present two simple examples that are illustrative of two important, and complementary, approaches to the separation and composition of concerns in software. These approaches are embodied by Aspect Oriented Programming systems, such as AspectJ [40] and Multi-Dimensional Separation of Concerns (MDSoC) [24] supported by Hyper/J [25]. We briefly outline these approaches and indicate how they might be used to manage the issues arising from our examples. We have implemented [15] several musical examples using AspectJ, but for clarity and simplicity of exposition, hypothetical examples are used throughout this paper. In order to make the rudiments of AOP techniques as clear as possible, our first example does *not* deal with any tangling of musical dimensions at all: instead it presents a version of a well-studied AOP example that deals with the tangling of purely *programming* concerns. Having used this example to present the rudiments of AOP unambiguously, our two subsequent examples illustrate (albeit in very rudimentary fashion) ways in which AOP may be used to manage tangled *musical* concerns.

Example 1 – Crosscutting

Consider the design of a Common Practice Notation editor. The editor enables the user to enter and arrange CPN via a graphical user interface (GUI). The system also incorporates a sequencer that enables the user to play the CPN as a MIDI sequence. The system permits the user to edit the CPN while the sequencer is playing the piece.

However it is observed that so doing causes the sequencer's timing to become distorted. One of the possible causes of this variation in timing is thought to be the routines that update the screen while editing takes place. The GUI has been designed using object-oriented principles, in which each graphical symbol is derived from the class `Glyph` and is responsible for rendering itself on the screen through its own implementation of the polymorphic method `draw()`.We therefore wish to be able to trace calls to the `draw()` methods of our various `Glyph`-derived classes.

Since the `draw()` methods share no common implementation, this *tracing concern* cannot be encapsulated using an object-oriented decomposition. Therefore an implementation of this concern, using only object-orientation, must necessarily tangle the `draw()` methods with the tracing concern, and scatter the tracing concern across all of the `Glyph`-derived classes. The tracing concern is said to *crosscut* the class graph. A similar argument applies *a fortiori* when the methods to be modified are scattered across classes with no relevant common root.

AOP [17] enables crosscutting concerns [17] to be modularised into *aspects* [17] such that tangling and scattering is avoided. Aspects are like classes, and can be specified separately, but contain additional information (outlined below) that specifies exactly the diverse loci (potentially scattered around the main program) where their behaviour is to be deployed. Aspects therefore represent dimensions of concern that cannot be encapsulated within a single dominant decomposition, such as an object-oriented class structure.

Having separated and modularised concerns, we need to consider how we might recompose them. In the AspectJ [40] implementation of AOP, aspects are composed at well-defined points in a programs execution, termed *joinpoints* [40]. Joinpoints are typically method calls and member variable accesses. Each concern implementation is associated with a set of one or more joinpoints, termed a *pointcut* [40], describing the points in the programs execution at which the concern is to be invoked. The concern implementation itself, termed *advice*, describes both the procedural elements of the concern implementation and its execution position relative to the joinpoint. *Before advice* executes before the code invoked by its joinpoint, *after advice* runs after the code invoked by its joinpoint. *Around advice* runs before the code invoked by its joinpoint, but exerts control over whether the

joinpoint is subsequently executed. In this way, around advice may run *instead* of a joinpoint.

Thus, a possible AOP implementation of the tracing concern might be to encapsulate tracing as an aspect that defines a pointcut consisting of all of the `draw()` methods that we wish to trace, and an advice that defines our preferred tracing implementation, such as writing to a text file.

Since pointcuts are declared within aspects, the 'base code' at which they are invoked is unaware of the aspect's existence. Consequently aspects may be applied and removed without invasive modification of the base code. AOP is typically applied to object oriented systems, but the underlying principles apply to software in general [12].

Example 2 – Multidimensional Separation of Concern

For this second example, we will consider higher-level concerns that might be addressed using aspects. For purposes of exposition and for brevity the example is presented in a highly simplified form.

Some concerns naturally extend across multiple, unrelated dimensions. For example consider a system, such as that described by Zimmerman [39], in which a fixed set of musical materials is required to be automatically composed and played in a variety styles, to convey various ambiences according to some script or storyboard. Changes in ambience or mood may require changes across many musical dimensions.

For the purposes of this example, we will consider a subset of the dimensions suggested in [39], namely Tempo, Rhythm, and Harmony, and three sample ambiences, A, B and C. MDSoC [24] avoids any dominant decomposition. Instead, MDSoC considers systems to be described as abstract slices, *hyperslices*, of functionality encompassing any number of dimensions of concern. Figure 1 illustrates conceptual hyperslices across tempo, rhythm and harmony dimensions, that satisfy three of the ambiences described in [39].

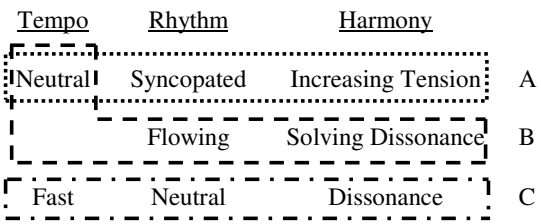

Figure 1

A key feature of the MDSoC approach, and its support through the Hyper/J™ tool, is its ability to achieve a 'clean' separation of concerns, and thus helps to reduce complexity, facilitate evolution and non-invasive adaptation and customisation, and promote reuse, in part by simplifying component integration. For example, we may wish to enable the composition system to produce a new ambience D by configuring a different slice through the existing implementations of the three dimensions, e.g. Tempo=Fast, Rhythm=Flowing, Harmony=Increasing Tension, or we may also wish to replace the Harmony implementations, say, to account for cultural differences and so forth. Using MDSoC, these changes may be effected largely through the description of the desired hyperslices rather than invasive software modification.

The implication is not that a composer should be constrained to use any particular simplified system of representation or control: quite the reverse. The use of aspects or MDSoC allows 'what-if' experiments to be made with diverse evolving approaches of arbitrary complexity, without invasive modifications of settled code. AOSD facilitates diverse experiments by facilitating altering any number of concerns independently.

3. ISSUES OF MUSICAL REPRESENTATION

Musical representations allow the user, composer, analyst or performer, to represent musical knowledge at an appropriate level of abstraction. However, as Smaill et al. point out [34], there are an enormous number of ways of thinking about music, and this leads to a diverse set of representations of which there is no single, all encompassing, representation.

Common Practice Notation and MIDI tend to focus on the principal perceptual dimensions of tonal music; pitch, rhythm, timbre and volume [20]. However such representations have little to say about musical structure. Conversely, structural representations, such as Structured Music Pieces [4] and CHARM [34] permit the composer to express hierarchic temporal and transformational relationships that exist between musical elements, but have no representation of, for example, harmonic progressions or orchestration. Other approaches to representation include declarative constraint-based systems [11][39], Grammars [6][5][18], Patterns [32][8][7] and Processes [35][2]. A particular composer's compositional process may involve any mix of these, and other, representations. Cope's EMI system [8], for example, is based on pattern matching and grammars, while Cybernetic Composer [1] utilises style rules and stochastic grammars. However, combining such approaches becomes problematic if the external representation of a process does not expose symbolic information that is subsequently required in other areas of the process.

Example 3 – Intrinsic Crosscutting

To illustrate this, with a very simple tonal example, consider a system that is required to generate arpeggio figures based on a sequence of symbolic chord representations, such as C, C7, C6. One way to implement this system is to iterate through the chord symbols and simply transform each chord into a set of MIDI pitch values from which the required arpeggio figure may be algorithmically generated and the pitches played sequentially. Such a process is illustrated in Figure 2 below.

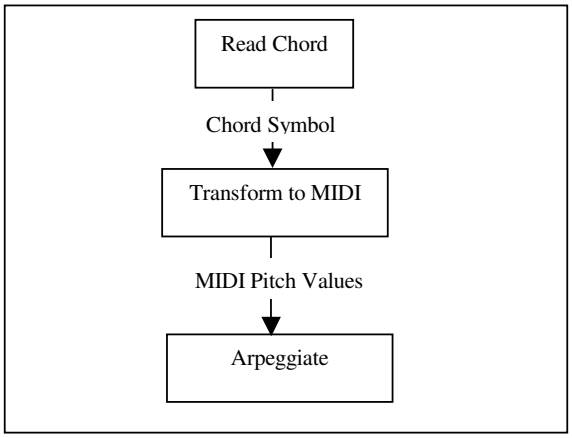

Figure 2

However, if we extend the system such that the exact arpeggio figure depends upon the chord type, then MIDI ceases to be a useable representation for the interface between the modules, since it does not convey the chord type directly nor is it generally possible to infer the chord type from the pitch values themselves. Using aspects however, we are able to non-invasively intercept the chord symbol as it is read, and make this information available to the arpeggiator. Thus when the arpeggiator receives the MIDI events, it is aware of the chord type and can produce the appropriate arpeggio figure. This is illustrated in Figure 3; the broken lines denote the aspect

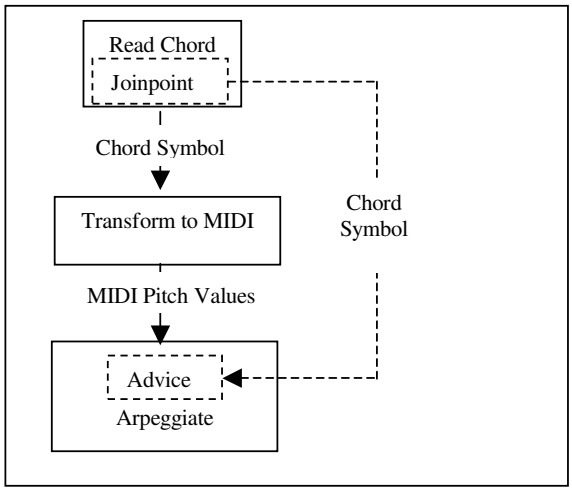

Figure 3

Clearly, it would be possible to design the software to fulfil the new requirement without using AOP. However, this example is illustrative of the general application of AOP in supporting the non-invasive evolution of software, where legacy code is required to work in new and unforeseen ways. This is, we believe, particularly appropriate to creative domains, such as music composition. More specifically however, we believe that aspects can play a useful role in the integration of musical applications in which symbolic state information is available within computational processes, but is encapsulated and not externalised by the available representation. While some representations, such as Structured Music Pieces [4] and MODE [28], permit arbitrary attribute/value pairs to be associated within musical objects, these items do not extend across representational boundaries, neither do they possess any inherent semantic value. Aspects, conversely, have the potential to permit symbolic information to remain in its native location, do not require arbitrary extension of representations and convey meaning through the use of pointcuts and advice.

4. MUSIC COMPOSITION PROCESSES AND DYNAMIC AOP

While composers such as Schoenberg [31] and Hindemith [33] speak of having a 'vision' of an entire work, other composers appear to work in an iterative fashion in which musical ideas across various musical dimensions are sketched out and elaborated, often incompletely, before the work is finally pieced together and completed [36][33].

Thus, not only are the musical dimensions themselves tangled, but so also are the cognitive processes involved in their composition. Computer systems that support musical composition should therefore allow the composer to work in an iterative,

incremental and interactive fashion with possibly incomplete musical material. We believe that aspects can help in achieving these goals since they help to reduce and manage complexity, can be applied without invasive modification, and have the ability to cross representational boundaries.

However, a key feature of music composition is that the relationships between musical dimensions do not necessarily persist for the entire duration of a piece. Rather they are added, modified, replaced or removed over time. Consequently, it would be desirable to be able to dynamically insert, withdraw and modify aspects either interactively, or based upon musical context. For example, a 'crescendo' aspect may be defined that simply modifies the dynamic dimension. This could be replaced with a 'crescendo' that is achieved by changes in orchestration or harmony. Further, the choice of 'crescendo' may depend upon context. For example, a solo piano part may required a 'dynamic crescendo' while an orchestral interlude may require an 'orchestral crescendo'.

Systems such as AspectJ and Hyper/J both implement aspects statically at compile time. Consequently, aspects do not 'exist', and therefore cannot be modified, at runtime. The joinpoint model of AspectJ, in particular, ties aspects to the particular application in which they are defined and makes it difficult, though not impossible [14], to define reusable aspects.

Nonetheless, emerging Dynamic AOP (DAOP) technologies do possess some of the features that are required to support the dynamism and interactivity requirements outlined above. In particular the techniques of Aspectual Components [19] and their realisation in systems such as JAsCo [38] enable generic aspects to be defined and loaded at runtime. The Caesar system [21] supports aspectual polymorphism in which the choice of aspect implementation may be determined at runtime. Event-based approaches to DAOP, such as Axon [3], PROSE [29] and EAOP [10], exhibit a synergy with the event-based nature of music and musical representations such as MIDI. We can imagine systems that generate musical events of arbitrary granularity, such as start of note, chord, start of phrase etc, and which are loosely coupled through an event mechanism to crosscutting implementations. We could then specify, for example, if the current phrase is phrase 1 or phrase 2 then apply orchestration A but if phrase 1 is in the context of section 2 then apply orchestration B. Logic metaprogramming approaches such as Andrew [13] enable declarative and dynamic expression of pointcuts, which in this particular implementation enjoys a symbiotic relationship with its Smalltalk environment.

5. CONCLUSIONS

In this paper we have outlined the basis of Aspect Oriented Programming and argued that AOP offers a technique allowing us to manage the tangling and scattering of various dimensions of concern that are inherent in many aspects of computer music.

We have identified two types of crosscutting that exist in music. Firstly, we have outlined that situations in which musical concerns are scattered over multiple unrelated classes may be represented and encapsulated as hyperslices using the MDSoC approach. We have shown that it is possible to compose new concerns by generating new hyperslices through existing functionality. Secondly, we have illustrated the application of aspects to situations in which two or more complex, normally encapsulated representations may need to mutually affect each other. In these situations, aspects obviate the requirement for a common representation by permitting non-invasive access to state information held in any of the representations.

Finally we have outlined the ad-hoc and dynamic nature of many interactions between dimensions of concern that arise in musical applications. We believe that Dynamic AOP may be fruitfully applied to musical systems as a means to solving some of the issues of uncertain requirements and system evolution that are evident within creative domains. Moreover, we believe that AOP enables the development of powerful new musical applications that enable the end-user to directly interact with and exploit dynamic, crosscutting musical concerns.

6. REFERENCES

[1] Ames. C., Domino. M. "Cybernetic Composer: An Overview". *Understanding Music with AI*. MIT Press, 1992

[2] Anderson, D.P., Kuivila, R. "Formula: A Programming Language for Expressive Computer Music". *IEEE Computer* Vol 24. No 7. 1991

[3] Aussmann, S., Haupt, M. "Axon – Dynamic AOP through Runtime Inspection and Monitoring." *ASARTI Workshop* 2003.

[4] Balaban., M. "Music Structures: Interleaving the Temporal and Hierarchical Aspects in Music." *Understanding Music with AI*. MIT Press. 1992.

[5] Beilharz, K.A. "Observing Musical Composition as a Design Grammar." Key Centre of Design Computing and Cognition.Department of Architectural and Design Science. University of Sydney. 2001

[6] Bel., B. "Symbolic and Sonic Representations of Sound-Object Structures". *Understanding Music with AI*. MIT Press. 1992

[7] Conklin, D., Anagnostopoulou, C. “Representation and Discovery of Multiple Viewpoint Patterns”. *Proceedings of the International Computer Music Conference*. 2001.

[8] Cope, D. “A Computer Model of Music Composition.” *Machine Models of Music*. MIT Press. 1993.

[9] Dannenberg, R. B., Desain, P., Honing, H. “Programming Language Design for Music”. *Musical Signal Processing* G. De Poli, A. Picialli, S. T. Pope, & C. Roads (eds.), 271-315. Lisse: Swets & Zeitlinger. 1997.

[10] Douence, R., Sudholt, M. “A model and a tool for Event-based Aspect-Oriented Programming (EAOP)”. TR 02/l 1/INFO, lecole des Mines de Nantes, french version accepted at LMO'03, 2nd edition, Dec. 2002

[11] Ebcioglu., K. “An Expert System for Harmonizing Chorales in the Style of J.S. Bach”. *Understanding Music with AI*. MIT Press. 1992

[12] Filman, R.E., Friedman. D.P. "Aspect-Oriented Programming is Quantification and Obliviousness." *Workshop on Advanced Separation of Concerns*, OOPSLA 2000, October 2000, Minneapolis.

[13] Gybels, K. “Aspect-Oriented Programming using a Logic Meta Programming Language to express cross-cutting through a dynamic joinpoint structure”. Ph.D. Thesis 2001

[14] Hannenberg, S., Unland, R. “Using and Reusing Aspects in AspectJ”. OOPSLA 2001

[15] Hill, P., Holland, S., Laney, R. C. “Using Aspects to Help Composers”. Technical Report TR 2003/21. Open University Dept of Computing 2003.

[16] Hürsch, W.L., Lopes, C.V., “Separation of Concerns.”
Technical Report by the College of Computer Science.
Northeastern University. 1995

[17] Kiczales C., Lamping J, et al., “Aspect-Oriented Programming”. *Proceedings ECOOP* 1997.

[18] Lerdahl, F., Jackendoff, R. *A Generative Theory of Tonal Music*, MIT Press, 1983.

[19] Lieberherr, K., Lorenz, D. and Mezini, M. “Programming with Aspectual Components”. Technical Report, NU-CCS-99-01, March 1999

[20] Loy, G., Abbott, C. “Programming Languages for Computer Music Synthesis, Performance and Composition”. *ACM Computing Surveys*, Vol.17, No. 2. June 1985.

[21] Mezini, M., Ostermann. K. Conquering Aspects with Caesar. *Proceedings AOSD,* 2003.

[22] Miranda., E.R. *Composing Music with Computers*. Focal Press. 2001

[23] Oppenheim, D.V. “Towards a Better Software-Design for Supporting Creative Musical Activity”. ICMC 1991.

[24] Ossher H., Tarr P., “Multi-Dimensional Separation of Concerns in Hyperspace”. Research Report, IBM T.J.Watson Research Center, 1999

[25] Ossher, H., Tarr. P. “Hyper/J™ User and Installation Manual”, IBM Corporation. 2000

[26] Persichetti, V. *Twentieth-Century Harmony Creative Aspects and Practice*. Norton. 1961.

[27] Piston. W. *Orchestration*. Gollancz. 1955.

[28] Pope, S. “Introduction to MODE: The Musical Object Development Environment”. *The Well-Tempered Object: Musical Applications of Object-Oriented Software Technology,* S. T. Pope, ed. MIT Press. 1991

[29] Popovici, A., Gross, T., Alonso, G. “Dynamic weaving for aspect oriented programming”. *Proceedings of the 1st International Conference on Aspect-Oriented Software Development,* April 2002

[30] Rowe, R. *Interactive Music Systems Machine Listening and Composing*. MIT Press. 1993

[31] Shoenberg, A. (Strang F, Stein L. eds). *Fundamentals of Music composition*, Faber and Faber. 1967.

[32] Simon., H.A., Sumner., R.K., “Pattern in Music”. 1968. *Machine Models of Music*. MIT Press. 1993.

[33] Sloboda, J.A., *The Musical Mind. The Cognitive Psychology of Music*. Oxford Science Publications. Oxford University Press. 1985.

[34] Smaill, A., Wiggins, G., Harris, M. “Hierarchical Music Representation for Composition and Analysis”. 1993

[35] Smoliar., S.W., “Process Structuring and Music Theory”. 1974. *Machine Models of Music*. MIT Press. 1993.

[36] Speigel, L. “Old Fashioned Composing from the Inside Out: On Sounding Un-Digital on the Compositional Level”. *Proceedings of the 8th Symposium on Small Computers in the Arts*, Nov. 1988.

[37] Sullivan, G., T. “Aspect-Oriented Programming using Reflection”. *Proceedings OOPSLA* 2001

[38] Suvée, D., Vanderperren., W., Jonckers., V. “JAsCo: an Aspect-Oriented approach tailored for Component Based Software Development”. *Proceedings AOSD* 2003.

[39] Zimmerman, D. “Modelling Musical Structures”. *Constraints Vol 6, pg 53-83*. Kluwer Academic Publishers. 2001.

[40] *The AspectJ Programming Guide*, Xerox Corporation. 1998-2002

[41] http://www.research.ibm.com/hyperspace