FROM MUSIC INFORMATION RETRIEVAL (MIR) TO INFORMATION RETRIEVAL FOR MUSIC (IRM)

Michaël G. M. Koenders

A thesis submitted in partial fulfilment of

the requirements for the degree of

Doctor of Philosophy

Sydney Conservatorium of Music

University of Sydney

June 2016

Supervisor: Dr. Ivan Zavada.

Auxiliary Supervisor Dr. Michael Smetanin.

Statement of originality

I declare that the research presented here is my own original work and has not been submitted to any other institution for the award of a degree.

Signed:

Date: 28th of June 2016.

Abstract

As an algorithmic composer I have been looking at a more abstract meta-level of musical composition: style and structure of a musical composition. To accomplish this I looked at certain techniques from the domain of (Music) Information Retrieval, in particular at some (common) general data mining algorithms.

Other well-known approaches, such as the use of Augmented Transition Networks (ATN) from the field of Music Information Retrieval are, to a certain extent, adequate as long as one keeps the underlying tonal constraints and rules as a guide to understanding the structure one is looking for. But since a large proportion of algorithmic music, including music composed by the author, is atonal, tonal constraints and rules are of little use.

Analysis methods from the field of Information Retrieval such as *k*-means and Expectation-Maximisation (EM), both Hierarchical Clustering Techniques (HCT), facilitate other approaches. HCT are Information Retrieval and general data mining tools that are better suited for finding (clustered) structures in large data sets. Other techniques as the ART2 Neural Networks (Adaptive Resonance Theory) can be used for analysing and categorising these data sets. And even more conventional statistical tools as histogram analysis, mean, variance as well as correlation calculations can tell us something about certain connections between members in a data set. Altogether a most promising palette of usable data analysis methods and strategies for creating algorithmic atonal music is now at our disposal. Now acting as (software) strategy tools, their use is determined by the quality of their output and usability in a musical context as I demonstrate when developed and programmed into my Computer Assisted Composition Environment: CACE4. We therefore turn Music Information Retrieval techniques the other way around and use their specific techniques and their associated methods of Information Retrieval and general data-mining to access the organisation and constraints of abstract (non-specific musical) data in order to use and transform it in a musical composition.

In this thesis I will review and discuss obtained results from the previously mentioned IR techniques and their specific adaptation(s) for use as building blocks in the CACE4 software application. By using them in this way as mathematical principles and methods, without the musical context, it is possible to use them as techniques in order to find structure and relationship in large(r) amounts of data. By using the data in this way we are now able to develop strategies to satisfy our musical goals: generating musical material with certain musical characteristics (i.e. style, structure, form and aesthetics).

This project consists of a thesis outlining the analytical methods as previously mentioned and implementing the methods in an application (CACE4) together with a portfolio of a number of compositions and their analysis.

Acknowledgements

Dr. Ivan Zavada.

Dr. Michael Smetanin.

MSc, MMus. Hans Timmermans.

Prof. Dr. Ludger Brümmer.

Simon Carter.

Pieter Suurmond.

Ted Szántó.

Willem Willemse.

Ensemble Camenae (Tizianna Pintus, Bart de Vrees, Antonius Pratsinakis, Daniël Kool, Laura Carmichael and Rozemarijn van Egeren).

Laurens de Boer.

Barbara Woof.

And many thanks to friends and family for their support:

Woof fam., A. Koenders-Seiler.

George Henderson Scholarship made available by the GH Foundation.

Index/Table of contents

List of Appendices

List of Equations

List of Figures

List of Tables

“But above all we shall have greater strength, for we shall feel we are participating, creators of ourselves, in the great work of creation which is the origin of all things and which goes on before our eyes.” (Bergson 1948, p. 105)

“The rule of science is the one posited by Bacon: obey in order to command. The philosopher neither obeys nor commands; he seeks to be at one with nature.” (Bergson 1948, p. 126)

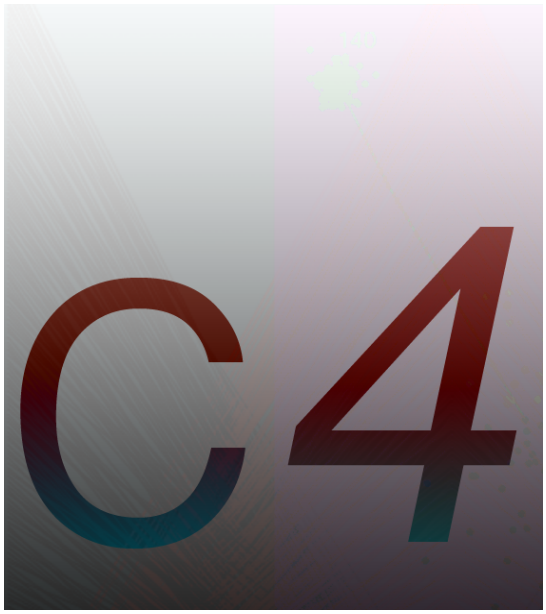


Figure 1 The CACE4 Application Icon.

Chapter 1
Introduction

As a composer I was always intrigued by structures as found in the aural domain of sound and music as well in the visual world around us. One of my earliest thoughts and desires was to combine these two more or less separate worlds into one, showing that there is a connection between them and that they share a common ground and specific structures.
By shifting and redirecting my compositional methodology of working to the domain of informatics and the use of computers, in the early eighties when cheap desktop computers, (e.g. Ataris and Commodores) came available for a larger audience, I was at once intrigued by this opening up of a whole new world of fascinating possibilities. It was now possible for me to start experimenting through developing software: small computer programs for synthesis and composition. Due to the fact that most of these early home computers had little ready-made software, I had to start with computer programming right away. Luckily there was always a compiler or an interpreter available, mostly BASIC[1] that in those days came standard with the machine itself. To accomplish new sounds and music I had to learn to program the computer as well. Fortunately not hindered by thoughts of how elaborate and time consuming it would be, I started right away.

From that moment on computer programming and composing became synonymous of each other. At last there was a machine, based on rules of logic, that gave me the possibility to combine the aural and visual world in one piece of software so I could start exploring there common ground. Now after more then thirty years of computer programming experience, developing many software applications in the domain of Digital Signal Processing[2] (DSP) and musical composition and creating numerous compositions with it, it is still as intriguing as it was from day one.

Fortunately the technology has evolved enormously and become much more sophisticated over time, greatly helping me in my quest for looking for structure in the visual and aural world around us. Concepts as musical structure and structured, statistical analyses of data and data mining[3] came together. If Music has structure and if found data has structure as well, would it be possible to connect

[1] BASIC: An Acronym for Beginner's All-purpose Symbolic Instruction Code. The original language design was done in 1964 by John Kemeny and Thomas Kurtz, Darthmouth College, New Hampshire USA. "It has a simple algebraic structure and is supported by a simple interpretive implementation. The language became extremely popular with the advent of personal and home computers." (Ghezzi and Jazayeri 1982, 1987, p. 354).
[2] The domain of discrete mathematics applied on signals: "Signals are represented mathematically as functions of one or more independent variables. For example, a speech signal would be represented mathematically by acoustic pressure as a function of time,…" (Oppenheim and Willsky 1983, p. 8). Digital Signal Processing (DSP) is the domain of signal engineering used for processing discrete (digital) signals. It is focused on the processing of digital streams (signals) with all its possibilities. It is a wide field where the laws of physics are used for either passively (measuring) or actively processing of a signal.
[3] The domain of data mining is defined as the process of analyzing data for finding relevant information as hidden patterns and structures. See for more information about the concept of data mining: https://docs.microsoft.com/en-us/sql/analysis-services/data-mining/data-mining-concepts

these two worlds? And would it be possible to use ideas from the domain of Information Retrieval[4] (IR) and general data mining and use them as part of the compositional process?

The possibility to look at a more abstract and higher meta[5] level for analyses and structure into large amounts of real-world data and to apply found structures for creating music gave me the idea for this PhD: from Music Information Retrieval[6] (MIR) to Information Retrieval for Music. Visualising by means of computer graphics and being able to listen to the results by creating MIDI[7] files (SMF) gave me a first connection between the worlds of hearing and seeing I was looking for.

Looking briefly at certain aspects of tonal music for an example of how to incorporate the idea of connections and constraints that can be translated to a specific grammar or set of rules and by handling the data in such a way to make it possible to look for the underlying connections, Augmented Transition Networks (ATN), derived from the earlier developed Petri nets, are well suited for this task[8].

The scientist and composer David Cope, makes extended use of Augmented Transition Networks techniques. As Cope states: "Augmented transition networks (ATNs) help vary output and generate extended examples of natural language processing. ATNs were crystallized and popularized by William Woods" (Cope 1991, p. 59). And on the next page: "Following roughly the same design as the language ATN algorithm, the musical version produces logical phrases (...) with style relevance (in this case Mozart)." (Cope 1991, p. 60) Cope copies the outlined idea of an ATN from Woods and applies it directly to the link between the ATN model for language and the use of an ATN model for Music. This approach for modelling information is adequate as long as these underlying tonal constraints and rules (melody, harmony and rhythm) are kept as a guide for understanding the structures it applies to. By focusing instead on the domain of Information Retrieval and general data mining for doing numerical analysis on data for creating artistic output, the use of the internal data is restricted to a one-dimensional stream for applying (mathematical) functions.

---

[4] As described in: Information Retrieval: "Information retrieval (IR) is concerned with representing, searching, and manipulating large collections of electronic text and other human-language data." (Büttcher, Clarke, and Cormack 2010, p. 2).

[5] A higher meta-level means on a higher, more abstract level, where groups of properties are brought together, and can be symbolized by different symbols and means. Rick Taube states in 'Notes from the metalevel': " If the score represents the composition then the metalevel represents the composition of the composition. A metalevel representation of music is concerned with representing the activity, or process, of musical composition as opposed to its artifact, the score. If the metalevel seems more ephemeral than the performance level it may be due to the fact that it is more closely related to the mysterious cognitive processes that occur within the composer." (Taube 2004, p. 3).

[6] Music Information Retrieval uses the same approaches and techniques found in the domain of IR only to apply it solely at the domain of music and sound.

[7] MIDI is an abbreviation of Musical Instrument Digital Interface, and was first introduced early 1980's. The created MIDI files are of a Standard MIDI File (SMF) format, type 1. The MIDI association watches over all standardization about the MIDI formats and MIDI file formats. See website: https://www.midi.org for details about MIDI and all of these different formats.

[8] "Petri nets are a special type of transition network that is used for the simulation of event-controlled processes and are represented by bipartite graphs. Nodes may consist of data, conditions and states (places) or actions (transitions)." (Nierhaus 2009, p. 127).

A construction of an ATN would therefore not be necessary. The fast developing field of Information Retrieval offers some other approaches: more extended statistical analysis techniques such as *k*-means[9] and Expectation-Maximisation[10] (EM) clustering. These two Hierarchical Clustering Techniques[11] (HCT) with unsupervised learning seem to be an adequate approach in classifying large(r) sets of data, without knowing a priori underlying relationship(s) or rules from these data sets.

These analysis methods, both from the group of Agglomerative Hierarchical Cluster[12] techniques (and there are many more), give other, more abstract all-round mathematical tools that are better suited and of more help to the more abstract domain of atonal music. Other advanced techniques such as Adaptive Resonance Theory neural networks (ART2[13]), are also promising for use as a data analysis method. In addition, even basic tools as histogram analysis, mean, variance and correlation calculations, product-moment correlation coefficient (Pearson) and rank order correlation (Spearman & Kendall τ) can be useful to understand the underlying structure and constraints being looked at.

more abstract, and most of the time non-musical approach, opens up the vast field of Information Retrieval and general data mining with all of its extensive and plural techniques.

Overall, it is an approach based on finding structures with no a priori musical knowledge whatsoever. Structure and organization in (input) data sets, as correlation or density (clustering) as spreading of the same set or other detectable statistical qualities, are only retrieved and learned by input (e.g. text files) and their specific analysis technique alone. This way of working liberates one from first needing to gain an in depth understanding of what type of musical structure and style can be obtained by incorporating these text files. For example, by using spectral analysis files based on Discrete Fast Fourier Transforms[14] derived from Spear[15], access to a huge domain of interesting data is provided.

---

[9] *k*-means is a clustering around centroids technique for detecting multiple clusters. For details see section: 5.5.1
[10] Expectation Maximization is another clustering technique. For details see section: 5.5.2
[11] Hierarchical Cluster techniques are a group of unsupervised cluster techniques used for detecting separate groups of clusters in unsorted data.
[12] There are two types: agglomerative, or 'bottom up' and divisive, a 'top down' approach. The divisive method is obtained by partitioning the observed single cluster to two least similar clusters. The agglomerative method is obtained by adding the most similar clusters. They can therefore be seen as complementary methods for detecting clusters.
[13] ART2 Neural Network, is a NN based on the use of Adaptive Resonance Theory originally designed and proposed by Carpenter & Grossberg (Carpenter and Grossberg 1987). Further explanation about ART2 and its implementation in CACE4, can be found at section 5.3.1 of this thesis.
[14] The Fast Fourier Transform is a technique based on the concept of the French mathematician Joseph Fourier. It is based on the concept of harmonic analysis of a sound and nowadays is in use in a special format as the Discrete Fourier Transform (DFT) and thus providing a method for doing an analysis on small segments of a sound (= band limited signal). Its use in a real-time DSP situation, for example in live performances, is therefore made possible. The Fast Fourier Transform has two stages; first an analysis of the sound is done, resulting in spectral components known as complex number pairs, where the real part of the complex number pair is the frequency component and the imaginary part is the amplitude component of the spectrum at a specific moment in time. While the sound has now been transformed to the frequency (or complex) domain, easy transformation of the spectrum is possible, because the two components, frequency and amplitude, are known.
At the second stage of the algorithm it has to be transformed back to the original time domain by using the inverse FFT, before one can hear the result. For more information about implementation and use of the FFT, see:

Various other types of data can be used in the search for underlying structures. Much scientific data for analysis is available on the Internet, saved in an Excel or other text based file formats and can be used for this purpose, with minor editing[16].

All the different approaches, as mentioned above, act as software tools for finding a strategy and are examined by the goal that can be achieved, based on the quality of the musical output and usability in a musical composition context.

Instead of using typical Music Information Retrieval techniques, it is turned the other way round and the approaches of Information Retrieval are looked at in order to obtain access to the organisation and constraints of abstract, mostly non-specific musical data. By adopting this more abstract approach, it is even possible to use certain techniques for looking at the structure of the data on a more meta-level[17]. This could be defined as being part of techniques for style and style recognition and also for organising the structure of a composition. All these separate ideas and different strategies have been translated into individual software modules that are integrated into a newly designed software environment I created: Computer Assisted Composition Environment (CACE4), programmed in LISP/CLOS[18]. These software modules will be tested in functionality and design for writing short and medium sized compositions. These compositions can either be score-based, performed by musicians, or they are MIDI based electronic compositions.

The CACE4 environment provides a toolkit of different techniques: solutions from the domain of Artificial Intelligence and Statistics. Its Graphical User Interface (GUI) makes playing around with the different components easily and quickly adaptable. This is necessary for doing experiments to find a well-suited strategy by reordering software modules (as represented by CACE4 objects) and connecting them in different ways. Accordingly, by doing so, the obtained results as generated by the software modules can vary strongly.

Most importantly, the composition as a conceptual idea should remain the starting point. Different compositions therefore, could demand different approaches in strategy.

---

http://en.dsplib.org/content/fft_introduction.html and
http://www.cmlab.csie.ntu.edu.tw/cml/dsp/training/coding/transform/fft.html

[15] Spear© created by Michael Klingbeil, is an Analysis/Synthesis program based on the principles of Fast Fourier Transforms. See http://www.klingbeil.com/spear/ for further details about Spear.

[16] A more detailed description of the data format used can be found at section 5.1

[17] Meta-level means in this context: on a higher level of abstraction.

[18] LISP is short for List Programming. An all-purpose, Computer Programming language based on lambda calculus (invented by John McCarthy in 1958). "In fact, the original LISP, introduced by John McCarthy in 1960, known as pure LISP, is completely functional."(Ghezzi and Jazayeri 1982, 1987, p. 274)
CLOS is short for Common Lisp Object System. An ANSI Standard, Object Oriented Programming language extension (1994, ANSI X3J13), as described in *Object-Oriented Programming in Common Lisp, A Programmer's Guide to CLOS*. (Keene 1989), and *Understanding CLOS, The Common Lisp Object System* (Lawless and Miller 1991)

Chapter 2

Music and Mathematics

2.1 Sharing the numbers.

Before looking in more detail at the relation between music and mathematics, it is useful to consider this from an historical perspective, in order to find common thoughts and ideas to explore in our research for a link between these two phenomena. Amongst others, Pythagoras (Ancient Greece, c. 570 – c. 495 BC) and his monochord is one of the first and most well known examples: "zu diesen Philosophen – vor Sokrates – gehören auch die Pythagoreer, eine Gruppe von Denkern, die sich um 500 v. Chr in Unteritalien um Phythagoras aus Samos geschart hatte. Die darstellung der Zahl als Urgrund des Seins in all seinen Erscheinungsformen - so auch der Musik – gehörte zu ihren zentralen Anliegen." (Morbach 2004, p. 26)[19] This idea placed Music in the same group as Geometry and Arithmetic, the two latter ones nowadays part of what we define as Mathematics. Sharing this common ground of Mathematics, from antiquity up to the late medieval period, was common practice. This practice was reflected in the (early) medieval ordering of the scientific world into the group of the seven so called free or liberal Arts (the word Arts, has its origin in the Latin word for knowledge and skills: Ars[20]).

Isidore of Seville (c. 560-636 AD) described in his *Etymologiae*[21] this system of division and ordering of the seven liberal arts into more detail. In accordance with older Hellenistic[22] views, Music has been put into the same group (as described in Book III), as Arithmetic, Geometry and Astronomy. Grammar (Book I), rhetoric and dialectic (or Logic), both from Book II belonged to the Trivium[23]. The first four were grouped into the Quadrivium (see Table 1, page 6). This ordering system of the Liberal Arts (Sciences) continued to exist during medieval times and only changed during the Renaissance[24] period.

---

[19] "Also this group of philosophers before Socrates, belongs the Pythagoreans: a group of thinkers which somewhere around 500 BC formed a group in Southern Italy around Pythagoras from Samos. The representation of numbers as a foundation of being – also for music – belongs to their central thoughts and beliefs." (Translation: the author.)
For further reading about Pythagoras and the Pythagoreans: http://plato.stanford.edu/entries/pythagoreanism/

[20] Isidor of Sevilla states: "Kunst heisst 'ars', weil ihre Ausübung nach festen (artus) Regeln und Vorschriften geschieht." (Morbach 2004, p. 36). Translation: "Art is called 'ars' because their practice is according to strict rules and regulations." (Translation: the author.)

[21] The *Etymologiae* (c. 600–625 AD) can be seen as a precursor of what later became the encyclopaedia. It is a bundling of twenty books describing all knowledge of the known World, and ordering it into specific topics (Sevilla 2008, 830).

[22] The Hellenistic Age is stated as roughly the period between the death of Alexander the Great and the establishment of the Roman Empire (323 BC – 31 BC) (Perry et al. 1989).

[23] The Trivium are the first three skills acquired at medieval Universities. The Quadrivium (four) skills were taught after the Trivium (Morbach 2004).

[24] The Renaissance is a cultural period between the 14th and 17th century in Western Europe. It involved a major shift in worldviews about Arts, Science and Society. (Perry et al. 1989)

Johannes Kepler (1571 – 1630)[25] succeeded in placing three of the four Liberal Arts more closely together:

	The seven Liberal Arts
Trivium:	1 Grammar
	2 Rhetoric
	3 Dialectic (also called Logic)
Quadrivium:	4 Arithmetic
	5 Music
	6 Geometry
	7 Astronomy

Table 1 The seven Liberal Arts grouped into Trivium and Quadrivium.

Geometry and Astronomy were placed under the leading role of Arithmetic (Mathematics). Music was more closely associated with Astrology (in those days not clearly separated from what we nowadays define as Astronomy). At that time Music became the 'Music of the Spheres' transformed and strongly influenced by the idea of 'Harmonices Mundi'[26] as stated by Kepler. These newly revealed harmonic proportions between celestial bodies (sun and planets) and their movement in the firmament were widely accepted as a fundamental law. Apart from everything being related in a harmonic way, there was no new fundamental insight into what Music as a natural phenomenon could really be. As a consequence, Music was no longer accepted as belonging to the same domain as Mathematics but 'drifted' into the group of the performing Arts. Renewed interest in the calculable part of Music came with Mozart's "Musikalisches Würfelspiel"[27] and the introduction of Music Machines (automata). The latter ones are not really to be taken seriously: they were built for entertaining purposes only.

Only with later scientists as Hermann von Helmholtz (1821 – 1894)[28] this shifted to a scientific (empirical) foundation of music and sound according to the laws of physics, as described in his Book: *On the sensations of Tone* (Helmholtz 1885). The early 20th century saw the development of a much more scientific foundation for the natural phenomena of sound, in particular through research into telecommunications as was done in the USA by Bell laboratories[29] and subsequently being explored for applications in music. After World War II, with the more widespread introduction of computers, a renewed interest in the calculable definition of music was revived. The previously introduced 12-tone composition techniques by Arnold Schönberg (1874 - 1951), as described in his book *Harmonie*

[25] More can be read about Johannes Kepler at the site of the Stanford Encyclopedia of Philosophy: http://plato.stanford.edu/entries/kepler/

[26] *Harmonices Mundi* is the title of a book published in 1619 by Johannes Kepler.

[27] A 'Musikalisches Wurfelspiel' was a well-known practice to compose music according existing rules and only 'randomize' the moment when it should be used in the composition.
"Perhaps the most famous historical example of algorithmic composition is W. A. Mozart's Musikalisches Würfelspiel – a dice game for assembling minuets out of a set of pre written measures of music. The sequence of measures was determined by a set of dice throws." (Roads 1996, p. 823)

[28] For further reading about Hermann Helmholtz and the importance of his book: On the sensations of Tone: http://plato.stanford.edu/entries/hermann-helmholtz/

[29] One of the well known' inventions of the Bell Telephone Laboratories Inc., is the transistor. The official website from the company: http://www.bell-labs.com/about/history-bell-labs/ with other historical inventions.

Lehre[30] (Schoenberg 1978, 1911), had already opened the field of parameterisation of music. Extension of this view with more strict serial composition techniques[31] and even by further formalizing of compositional techniques was done by the composer Iannis Xenakis (1922 – 2001), as demonstrated in his *ST/4* (= stochastic) composition for String quartet (1956 – 1962). This process of formalisation of the compositional process in an abstract, mostly mathematical process description, is described in his book: *Formalized Music* (Xenakis 1971). This total formalisation of the composition process based on Mathematics and logical rule based assumptions about what the nature of Music is, made a direct translation into a formal computer language possible. Xenakis did this by using Set theory, Stochastic Processes and Game theory as the core of the compositional process instead of relying on the older 'rules' of Tonal Music. (To what extent the calculated output is used directly and unaltered, or that it has been 'edited' by hand is not always clear.)

Other, more rule based approaches of formalisation have been done by Lejaren Hiller and Leonard Isaacson in their String Quartet #4: *Illiac Suite*[32] composed in 1957. It has been widely acknowledged to be the first algorithmic music composition composed on a digital computer: *Illiac I* (Roads 1996). From the sixties on, computers became more readily available and purposes apart from doing scientific calculations were researched. With the introduction of even smaller and cheaper Personal Computers (PC's) in the late seventies and early eighties of the 20th Century, the development of software for doing research or to create new sounds and music compositions extended in many new and different directions.

2.2 Adding a certain ratio: Pythagoras, Birkhoff and Max Bense.

One of the major misconceptions about Pythagoras is the assumption that he would have said that everything is derived from numbers. However, this does not seem to be the case:
"I have learned that many of the Greeks believe Pythagoras said all things are generated from number. The very assertion poses a difficulty: How can things which do not exist even be conceived to

[30] Also known by its English translation: *Theory of Harmony*.
[31] Serial Music or Serialism can be defined as a strict and rigid parameterization of Music. Parameters as pitch, duration, loudness and timbre are fixed series of values. Provided as fixed entities and ordered in time by other, mostly random processes. In-depth research on behalf of these processes and their implication for the aesthetics of Serialism are described in: *Serial Music, Serial Aesthetics* by M. J. Grant (Grant 2001).
[32] The *Illiac Suite* for String Quartet (1957) is composed in four movements, involving several composition techniques. Ranging from generating a 'Cantus Firmus' up to the processing of Markov chains. It is widely accepted to be the first electronically generated composition based on compositional rules programmed (in a computer language) on an electronic computer (The Illiac I). (Loy 2006) (Roads 1996)
Hiller and Isaacson state, (originally from: Hiller, Lejaren and Leonard Isaacson 1959 *Experimental Music*, New York): "The process of musical composition can be characterized as involving a series of choices of musical elements from an essentially limitless variety of musical raw materials. Therefore, because the act of composing can be thought of as the extraction of order out of a chaotic multitude of available possibilities, it can be studied at least semi-quantitatively by applying certain mathematical operations deriving from probability theory and certain general principles of analysis incorporated in a new theory of communication called information theory. It becomes possible, as a consequence, to apply computers to the study of those aspects of the process of composition which can be formalized in these terms." (Loy 2006, p. 360)

generate? But he did not say that all things come to be from number; rather, in accordance with number - on the grounds that order in the primary sense is in number and it is by participation in order that a first and a second and the rest sequentially are assigned to things which are counted." (Waithe 1987, p 12)

More evidence for this thesis can be found in work from the Greek philosopher Theano of Crotono[33]: "Theano is saying that when we ask what is the nature of an object, we can reply either by drawing an analogy between that object and something else, or we can define the object. According to her, Pythagoras meant to express an analogy between things and numbers. This is the concept of imitation: things are like numbers. By its participation in the universe of order and harmony, an object, whether corporeal or not, can be sequenced with all other objects and can be counted. Things can be counted in accordance with number, the primary sense of which is ordering." (Waithe 1987, p 13)

If the two previous citations are kept in mind, it is reasonable to presume that Pythagoras meant that it is possible to substitute integer numbers as abstract symbols for things. This can be seen as one of the first approaches of symbolic manipulation and acts as a core foundation for mathematics. By using this idea as an axiom[34] (postulate or assumption) he made it possible to set the symbols (integer numbers) in relation to each other, thus defining ratio as a proportional ratio symbolised by the mathematical operand of dividing (e.g. a:b or a/b)[35]. Pythagoras himself did introduce this thought in his famous idea about the Monochord: where every overtone and thus musical scales can be derived from the initial fundamental tone divided by a specific ratio[36].

By freeing up this imaginary 'ideal' world from its natural context and transforming it into an abstract, symbolic world, the next step of introducing this postulate into the world of sound and music is now much easier. Nowadays, this kind of substituting 'things' with numbers is no longer an abstract philosophical thought: every computer simulation[37] on a computer is based on this assumption[38]. This

---

[33] Theano of Crotono (6th-century B.C., Greece).
For further reading: http://plato.stanford.edu/entries/pythagoreanism/#women

[34] An Axiom is a fundamental starting point, or presumption for a theory. From *The Oxford Companion to the Mind*: "The Aristotelian definition of an axiom is that it is a principle common to all sciences, which is self evidently true (and thus knowable a priori) but incapable of proof." (Adrian et al. 1987, p. 67)
"A postulate on the other hand is a principle specific to a given science which is assumed without proof and whose truth may not be self-evident." (Adrian et al. 1987, p. 67)

[35] In more detail: both notational forms imply a different idea: ':' stands for more proportional opposites. And the '/' implies a strict divide, which taken strictly, has a different meaning.

[36] Pythagoras used the Monochord to illustrate the proportional properties, expressed in integer numbers, of a given fundamental (or keynote) in ratio with an overtone or partial. For example, for finding the octave the string needs to be halved in length (2:1). By using only a quarter of the original string length the second octave above the fundamental can be found. Further division can be used to find all of the tones used in the diatonic scale.

[37] Computer simulation is a collection of algorithms (and equations) in which a real-world process description is modeled. In order to obtain knowledge about the real life situation, running the model or simulation does calculations. Also, as defined in *The Wordsworth Dictionary of Science & Technology*: "Simulation (Comp.) Method of studying the behavior of a system by using a model of the system and processing it on the computer." (Heckl et al. 1988, 1995, p. 816)

[38] There is one catch here that needs to be mentioned: Computer simulations are not only about quantifying things into numbers but also about processing (algorithm) calculation as well.

is especially the case in the direction of physical modelling in computer music, where physical processes (e.g. string vibrations) and the properties of the material (e.g. wood, iron etc.) are used for simulating the physical behaviour of those components according to all physical processes involved. At this point one could ask if music is by nature a language or if there is another underlying - hidden - process or organisational structure. The utmost consequence of either approach is that it is now possible to see Music as a 'chain' of context sensitive symbols that can be replaced or represented by other symbols: numbers, notes, language etc. This direct form of representation of properties of the (visual and auditory) Arts is something that is also seen in the work of the German philosopher Max Bense (1910 – 1990)[39]. Although Bense's initial idea is based on aesthetics derived from linguistics and the visual arts, there are many parallel ideas for structure and organisation of musical material to be found. (In a later work he addresses music in more detail.) Two important views in his philosophy can be found. First is the idea about using entropy[40] (of the art work) as an important factor for the computability of aesthetics in art. Initially derived from George David Birkhoff[41] (1884 – 1944) an American mathematician, Max Bense adopted one of his ideas as expressed into the equation:

$$M = \frac{O}{C}$$ **, where *M* is the aesthetic measure, *O* is the amount of order and *C* is complexity.**

Equation 1 Birkhoff's equation of the aesthetic measure.

Originally proposed by Birkhoff, where M is the aesthetics measure which can be calculated from Order (O) divided (ratio) by Complexity (C). This formula (see Equation 1, page 9) makes a calculation of the measurement of aesthetics possible and has a direct connection with a physical entity from thermo dynamics: entropy. While CACE4 is a computer program, its behaviour and output are strict logical constructs. Therefore, all operations are inside the logical domain.

If Bishoff's equation (see Equation 1, page 9) is then applied as a measure of the desired artistic output and the way CACE4 operates, it can be seen that the Order (O) should be high and the Complexity (C) has to be low, in order to obtain a high value for the artistic measure (M).

This equation strictly applies to the main goal of CACE4: to find order, and therefore restricting chaos to achieve an artistic output with a high value of *M*: the Artistic Measure. If order (*O*) is measured

[39] On the official Stuttgarter-schule site of Max Bense: "Max Bense (1910-1990) studied mathematics, physics, geology and philosophy at the Bonn University where he gained his Ph.D. + Sc. in December 1937. Already as a student, he began to publish. " (Walther 2000). Max Bense's works focuses on topics as semiotic, aesthetics, cybernetics and art. The official Max Bense site: http://www.max-bense.de

[40] Entropy is defined as a measurement of order-disorder in a given system.
"In chemistry, entropy is a measure of the ways in which energy of a molecular system is distributed among the motions of its particles, its thermodynamic probability. In information theory, entropy is a measure of the ways in which the information of a signal system is distributed among its communications."(Loy 2006, p. 345)

[41] In 1933 Birkhoff published *Aesthetic Measure,* a mathematical theory about aesthetics.

with the equation Bishoff proposed, then more order can be achieved by the use of CACE4 objects, as they are tools for finding order (in data) with Information Retrieval and data mining techniques. They act otherwise as tools for enforcing order by applying mathematical functions, thus also increasing order in the (pre-) musical (artistic) result. This should result, all according Bishoff's equation, in a higher value of *M*, the Artistic Measure.

Bense's second idea about generative aesthetics: that the aesthetic measure of a work of art based on these principles (of generative aesthetics), should be calculable according this formula[42]. Bense gives a precise definition of his idea about generative aesthetics: "Generative aesthetics therefore implies a combination of all operations, rules and theorems which can be used deliberately to produce aesthetic states (both distributions and configurations) when applied to a set of material elements. Hence generative aesthetics is analogous to generative grammar, in so far as it helps to formulate the principles of a grammatical schema–realizations of an aesthetic structure.

Any generative aesthetics that leads to an aesthetic synthesis must be preceded by analytical aesthetics. This process is responsible for the preparation of aesthetic structures based on the aesthetic information found in given works of art. In order to be projected and realized in a concrete number of material elements, the prepared aesthetic information must be described in abstract (mathematical) terms. At the moment there are four different ways of making abstract descriptions of aesthetic states (distributions or configurations), which can be used to produce aesthetic structures–the semiotic (employing classifications) and the metrical, statistical and topological methods–the latter three are numerically or geometrically orientated " (Reichardt 1971, p. 4)[43]. To summarise Bense's philosophy: Any generative aesthetic model which leads to a new synthesised aesthetic product (a work of art) should be preceded by any type of aesthetic analyses based on structures and described in abstract, mathematical terms.

This provides one with an aesthetic manifest for use in the world of informatics and formalised music. The interesting conclusion is that in 2500 years of western philosophy, the fundamental idea as stated by Pythagoras, of substituting numbers for (real world) objects, is even more valid then ever. This is even more emphasised with the now widespread practice of using computers in the world of music.

2.3 How is this idea incorporated in the design of CACE4?

The question about how this idea is incorporated in the design of CACE4 can be answered by taking the previously described, rather strict mathematical approach of substituting the properties of

[42] The formula demonstrates that using low values for the amount of order in a piece of Art but also combined with a low value for complexity gives a rather high value for the aesthetic measure. (The formula was later adapted: the divider operand was replaced with a multiplication. This implies that with small numbers for the values of O (order) and C (complexity) one still has a small number for M (Aesthetic Measure).
[43] This citation is from: http://www.computer.org/Bense_manifest.pdf (page 4). The original is from Max Bense and was published in: "Cybernetics, art and ideas by Jasia Reichardt (Reichardt 1971).

musical notes by numbers as the fundamental idea of CACE4. By using this analogy, one is not limited to fundamental musical properties such as duration, volume/dynamics, start-time and timbre. This model is not fixed so that other properties as coordinates in space (Cartesian and polar), movement (direction and speed) and timbre (other parameters of timbre such as partials and f0[44]) can be added to this model. The other important software design idea was to create a direct translation of the data into a 1 or 2 dimensional numerical stream of numbers, without the use of an underlying musical language. This gives us the ability to approach every selected topic as such: as a single delimited process with specific methods, described by the mathematical description of the process in a formula and translated into an algorithm. These two basic ideas can be seen as the core strategy for designing CACE4 and all of its functionality into a computer program suitable for doing analysis on data and creating Musical content by found mathematical constraints.

The decision to qualify the phenomenon either as language or to perceive it as a numerical stream is initially an arbitrary one. By doing so however, certain pitfalls can be avoided about a qualified system for representation of Music as a (formal) language[45]. By taking a more mathematical approach for analysing data as the core strategy, as suggested by Max Bense in his thesis about generative aesthetics and by 'replacing' musical properties (e.g. pitch, dynamics, duration etc.) with (real) numbers numerical Mathematics can be used to alter these properties. Furthermore, in creating blocks of musical data and saving them in a file, the output can be worked with in musical compositions.

[44] Further parameterization of the timbre can be obtained by incorporating formal models of timbre description of the domain of DSP.

[45] On the other hand, the open design of the CACE4 environment allows one to create a separate module solely based on representing Music as a specific system of language representation.

Chapter 3
From MIR to IRM

As the title of this thesis "From Music Information Retrieval to Information Retrieval for Music." suggests, the goal is to find useful tools in the domain of mathematics and informatics for musical purposes. In this chapter some aspects of the development of algorithmic music composition are looked at and, in relation to this, how IR (and many more techniques) is used for creating musical tools.

3.1 Definitions.

In this thesis certain terminology will be taken into account. In the description of the several CACE4 objects the following terminology will be used:

CACE4 *name* Object	Indicates one of the CACE4 objects'. It is a direct representation of the process and presents itself (to the user) with a clear Graphical User Interface (= GUI). This is a CACE4 object and has a representation as a small colored box in the CACE4 Project window, and it's own specific GUI. It is also referred to as a CACE4 Modules.
Input list, input stream	The two can be seen as the same. (Although the use of the word stream is in the context of CACE4 a bit ambiguous. A stream will normally be seen as a more or less continuous flow of data.). The input list contains the original input as a 1 dimensional stream of numbers.
Output list, output stream	Are just like the input list synonyms and are used for saving the calculated output for further use by other CACE4 objects.
Generator	The generator objects are at the beginning of every strategy in a CACE4 process window. They are needed to get initial data for further processing done by other CACE4 modules. The input of these Generators do either consists files or calculations.
Manipulator	Manipulator modules are the core business of CACE4. They take care for analyzing and processing. Therefore transformation of the input data in the desired output data or generating data otherwise is possible by either changing the data
Algorithm	Is a formal description of a process, in order to obtain a specific result (Mostly programmed in a computer language).
<button-name>	Will be used for naming buttons as they show up as part of the GUI. This will mostly be accompanied by a figure number as reference.
CACE4 processing chain, processing and strategy	They all apply to the chain of connected objects visual in the processor window. The CACE4 objects are connected by lines (arrow-objects): visible in the GUI and indicating with their arrowheads the direction of flow of the processed data.
anylispfunctionname()	All other LISP functions, either from CL or introduced in CACE4, will be displayed in italics and ends with parentheses.
defgeneric()	A specific LISP macro[46] to define generic functions or methods. This is a very useful tool for bringing similar methods, though belonging to different Classes, together[47] (Steele 1990, p. 827).

[46] The function of a macro, as defined in Common Lisp, is to write functions and methods on a more general level of encoding. Macros are not functions in Common Lisp and should not rely on the execution of specific system variables. Their purpose therefore, is to write more general code before the process of compilation takes place. When executed at run time, different types of execution of the code are possible. For example, the use of a variable can be declared as one of many types: floats, shorts, lists, arrays, strings (etcetera) and can be recognized and accordingly dealt with at run time. This interchangeability of the different types makes it a very powerful software developer tool.

defmethod()	A LISP function for defining a method for an existing class (member) (Steele 1990, p. 838).
defclass()	A LISP function for defining a new class. In the LISP coding this pre-defined class will be used with (make-instance <class-name>) to create an instance of the original class. All functionality of the class and its Methods will be inherited (Steele 1990, p. 822).
[…, …]	Number ranges of indicated parameters. Can either be used for indicated input ranges for text-edit fields (GUI), or indicates a range of output (domain).
Musical functionality – pitch, dynamics, volume and duration amongst others.	These musical properties are used as well known musical terminology; no new terminology has been introduced.

Table 2 Used terminology and definitions.

3.2 Taking a look at algorithmic music composition.

In order to make use of these techniques from IR, statistics and data mining, one must first describe and define what these rules are, how to apply them and what use they have in music. Is it possible to create a formal description of musical functions that can be used to symbolise and thus could replace the underlying structure for creating music?

Most rules developed in particular to the constraints of harmonic progressions, voice leading and melodic structures in tonal music, cannot be adapted sufficiently to atonal music and music where parameters other than pitch are being used as a structural basis for a composition (for example, spectral music[48]).

If music is to be perceived as ordered (distributed) data over time, according to a mathematical description, and processes inside a system (music) are developed over time, can a field in the domain of mathematics be found which deals with these descriptions of these processes and their system, and apply it as a model for composing music? Probability as a mathematical description is one of the possibilities, as Temperley states in *Music and Probability*: " If music perception is largely probabilistic in nature (and I will argue that it is), this should not surprise us. Probability pervades almost every aspect of mental life-the environment that surrounds us, and the way we perceive, analyze, and manipulate that environment." (Temperley 2007, p. 2)

Xenakis offers a mathematical (probability calculus) solution, as stated in *Formalized Music*: "But everything in pure determinism or in less pure indeterminism is subjected to the fundamental

[47] It has been defined in 'Common LISP The Language – second edition. (Steele 1990) as: "The macro *defgeneric()* is used to define a generic function or to specify options and declarations that pertain to a generic function as a whole." (Steele 1990, p. 827). Grouping of functionality on the programming level contributes to functions and methods, which are capable of operating automatically on very different types of variables. As a computer programmer: it frees the process of software development from the painstaking checking of all variables (or in LISP terminology: arguments) of functions and methods, as this will be done by the Macro as defined.

[48] Spectral music emerged in the 1970's and is defined as music where pitch, tonality and harmony are no longer defined in the traditional, tonal sense. Instead it mostly focuses on the use of psychoacoustics and spectral changes applied in the domain of physics by using frequency, amplitude and timbre. See for more details on the wide variety of spectral music: URL: https://www.york.ac.uk/music/undergraduate/modules/2013-14/spectral-music/

operational laws of logic, which were disentangled by mathematical thought under the title of general algebra. These laws operate on isolated states or on sets of elements with the aid of operations, the most primitive of which are the union, notated ∪, the intersection, notated ∩, and the negation. Equivalence, implications, and quantifications are elementary relations from which all current science can be constructed. Music, then, may be defined as an organisation of these elementary operations and relations between sonic entities or between functions of sonic entities." (Xenakis 1971, p. 4)

As a consequence, Xenakis transforms the compositional process by using probability and statistics as rules for composing music, as stated in *Formalized Music*: "I originated in 1954 a music constructed from the principle of indeterminism; two years later I named it "Stochastic Music." The laws of the calculus of probability entered composition through musical necessity." (Xenakis 1971, p. 8) The last remark of Xenakis, of making probability enter the composition, has to be understood as using probability as a formal set of rules to be applied to the compositional process. It can only be understood as an artistic decision deeply rooted in science, based on the assumption that music is a physical phenomenon.

At the same time, other systems of formal composition were proposed by composers such as Pierre Barbaud in his book *Initiation a la composition musical automatique* (Barbaud 1965), based on matrix probability calculations.

As more composers gained access to the computers in the seventies and eighties, in particular due to the development of personal computers (PCs), the vision broadened of how music could also be defined as sets of data, algorithms and mathematical equations and using these as an alternative for more traditional, tonal and harmonic rules. Approaching the domain and the process of composing on a more abstract level led to many analogies between other areas of science and research, as compositional processes for creating music[49]. One of these quickly evolving other scientific research areas, is the domain of Information Retrieval and the science of 'Big Data'[50]: a vast and rapidly growing domain with an expanding community[51]. There is a common interest and, to a certain level,

[49] One of these techniques with a parallel in a different scientific domain is the direction of music genetic programming based on the associative translation of concepts from biology as DNA, and DNA sequencing as carriers of musical information. The information of the DNA can be altered over time, hence mimicking DNA damage and the processes of mutation and evolution. This model has been copied as the compositional process for evolving music over time (for more information see URL: http://elib.mi.sanu.ac.rs/files/journals/yjor/39/yujorn39p157-177.pdf).

[50] "Big data essentially means datasets that are too large for traditional data processing systems and therefore require new processing technologies. As with the traditional technologies, big data technologies are used for many tasks, including data engineering. Occasionally, big data technologies are actually used for implementing data mining techniques. However, much more often the well-known big data technologies are used for data processing in support of the data mining techniques and other data science activities,…" (Provost and Fawcett 2013, p. 8).

[51] In the domain of Music and Information retrieval: ISMIR (= International Society for Music Information retrieval) is the organization involved in coordinating this rapidly growing area of new music research. Mailing lists for information exchange and organizing a yearly Conference is one of the tasks of ISMIR. For more information, see http://www.ismir.net for details about the activities of the organization.

overlap between these areas in approaching the problem of the 'unknown' data. To have a priori no knowledge of the data makes it rather easy and legitimate as well, to use any kind of data to be analyzed by these techniques. Whether it is musical material, growing algae or the latest information of the global economy as the amount of data produced every day by the financial markets, it can all be approached as a single stream, where only the underlying and intrinsically hidden structures are in the data set itself.

Recapitulating, it can be concluded that by defining music as a physical phenomenon (as Xenakis and Temperley previously stated), these rules of probability do apply and are fully functional. Therefore, by making use of IR, statistical analysis and data mining as our set of rules[52] to be applied for creating music, CACE4, as a compositional tool, is defined within the limit/boundaries of this definition.

Focusing now on how to define a method for comparing several methods for their usability in a musical, compositional context, leads to the definition of a set of (four) questions, which first need to be answered.

3.2.1 Firstly, before coping with (large) amounts of data, are there any readymade tools and techniques from the domain of IR one can use?

3.2.2 Secondly, can we group the different mathematical approaches together and use them as a strategy for analysing their output in a compositionally useful way?[53]

3.2.3 Thirdly, is there a possibility of defining statistical algorithms in a more (traditional) musical approach: using them as musical entities? To be more specific: are there analogies between mathematical formulae and musical functionality?

3.2.4 Fourth and finally, is it possible to tell something more and directly derived from the organization of the data itself, about so called meta-events such as the structure and style of a musical composition? Can we therefore conclude that, to a certain extent, the way the data is organized results in specific characteristics directly noticeable on the meta-level of style and structure of the composition? The latter question will be without doubt the hardest one to answer, if at all possible.

In order to answer these questions the domain of applied Mathematics and Information Retrieval will be looked at.

For the first question (3.2.1), in the case of handling (large) amounts of data and whether there are any readymade tools for processing: mathematics and especially the field of statistics give a well defined set of possible solutions to gain knowledge about, mostly hidden, connections and correlations between different data sets. One technique for solving this problem can be found in the area of

[52] A user of the CACE4 has to develop a strategy in a CACE4 Processor object by using several CACE4 objects in a chain (see section 5.8.1, page 101, for further details), and as such, compositional rules have to be translated into a strategy in a CACE4 Processor window.

[53] All of this without intermingling too much with the original values of the data set, since otherwise it may well be possible that important underlying numerical relationships could be disturbed before they have been detected.

correlation calculation, used specifically for finding correlation between initially unrelated data sets. Other tools from the domain of statistics, such as linear regression and histogram analysis, have different approaches in finding useful information from the data (tendency and grouping). From the domain of cluster analysis, algorithms can be used for detecting multiple clusters and their distances in a set of data. Hierarchical Cluster Techniques as *k*-means and Expectation-Maximisation (EM) provide these tools. A tentative conclusion can therefore be made, that readymade tools are available for handling data sets.

For answering the second question (question 3.2.2), on whether it is possible to find functional groupings of methods from IR and to use them as an analogy with a specific musical term and functionality: these higher organisation levels in music need to be compared with the functionality of the IR methods provided. This could be hard to achieve since not every musical term defined as such, can be copied into an existing IR technique. CACE4 should therefore be restricted and focus on only a few of these music functions, such as pitch, dynamics, delta start time and duration, plus others on a more higher level of organisation as building blocks for larger sections. The structure of these larger sections as (pre-) musical output, should not be ordered around musical terms such as structure (form) in a strict classical tradition (for example sonata and fugue), but should reflect structures found in data files and create new possibilities for composing structures.

As such, CACE4 is not a music generator focused around the generation of output according to these musical terms. Instead, it can be seen as a variable music generator, which could calculate a single note or many thousands at once.

With regard to the third question (question 3.2.3), on whether a meaningful comparison is possible between these newly grouped processing algorithms and older, more traditional musical approaches: it becomes apparent that CACE4 is much better suited for developing a strategy as an analogy to existing techniques of aleatoric compositional processes. It can be seen as a 'Musikalisches Würfelspiel', based on extended use of statistics and data mining as a means for creating music. The processing of the data makes it well suited for serial composition techniques, since the development and progress in the music is replaced by calculated output of mathematical equations and functions.

Without, for now, the implementation of coded musical knowledge, these methods for analyzing the input data are focused on solely, before the last question (question 3.2.4) about style and structure of a music composition can be answered. This means that the complex puzzle of a musical style cannot be solved in a direct way. Smaller parts however, acting as smaller building blocks of musical style, can be isolated and recognised[54]. This gives them a direct practical implication in the processes of

[54] Certain properties as density, give information about distribution of an amount over time. If reflected in the domain of Music; if the density of pitched notes or sounds are high, the total perception of the material could be perceived with the following qualifications: dense, quick or with a high tempo marking. This is in contrast if the same material would be presented at a much lesser density: sparser, slow or with a low tempo marking.

composing. Atonal music based on formal (= mathematical) process description and not defined music in other terms, is better suited for finding these mathematical building blocks.

Taking these limitations of CACE4 into account, one could say that as long as a part of style is defined as a 'gathering' of discrete characteristics then certain aspects can be detected, isolated and analyzed by the previously described IR techniques. By using several techniques and by putting them in a suitable sequential order, a more complicated compositional process description can be achieved. As such, by connecting the musical usability to creativity, it is possible that by using these ideas as another way of composing music, a different style of music could emerge.

3.3 Creating musical tools from common Information Retrieval techniques.

Before starting to describe (see Chapters 4 and 5 for more detail regarding analysis, design and development of CACE4 objects) the details of the modules developed, one needs to take into account whether it is possible to create musical tools from Information Retrieval techniques, statistics and data mining practice. What first comes to mind when considering this is the legality of the implied analogy between the two domains in respect to handling the data. Can there be a sensible translation of music and its representation to a representation usable by IR techniques (and vice versa)? Furthermore, does the result from this approach calculate material usable for creating interesting musical material?
By using SMF as input, the data can be worked with as long as the MIDI functionality of the original data in the SMF is respected. In CACE4 this has been accomplished by creating an internal representation of the (related) data. The MIDI representation of Timing (MIDI Clock Ticks, Tempo) and note representation (keys/pitch, velocity/dynamics) must be taken into account. If this translation can be made without altering their representation in the MIDI domain, a translation to an internal representation is possible. The problem has now been isolated and thus the Information Techniques implemented in CACE4 can be worked with. The same problem, but in retrograde, occurs by translating the data back, after working with the selected Information Techniques. By grouping data with the same MIDI representation and to maintain their internal CACE4 representation, it is now possible to use the IR Techniques only on the functional groups (e.g. pitch, velocity etc.) as desired. In order to keep track of these functional groups, sequential labelling should be done to keep the context (= internal representation) at all times. If this method of translation (of the SMF) is adhered to, then a positive answer can be given to the question.
What occurs however, if other (text) files are used as input? In the case of using Spear partials text file, the result of a DFT (Discrete Fourier Transform) of an audio file can be read in a particular format. (This opens up the possibility doing some analysis in the frequency domain)

Analysis of the audio file: Piano.wav done by Spear generates a Piano partials.txt file in the following format:

par-text-partials-format
point-type time frequency amplitude
partials-count 24
partials-data
0 100 0.001301 1.228546
0.001301 1518.669800 0.001688 0.020790 1544.828003 0.002815 0.028885 1545.946289 0.004013 0.038296 1552.878174 0.004992 0.050129 1556.715454 …

The above text (in italics) is the first few lines of a SPEAR partials text file. After point-type (second line of the text), time, frequency and amplitude are found. This is the order of reading the file data. After partials-data starts the read in: time (in milliseconds, the exact time interval depends on the precise Sample Rate per second), frequency (in hertz) and the amplitude (domain: [0.0,…,1.0]). This SPEAR partials text file will be automatically read in and converted by the CACE4 program.
For all other text files some work has to be done in order to prepare a file. In the case of files in .csv or Excel files being used: get rid of all tabs and ',' and other extra characters. The numbers need to be in pairs of numbers (x,y) orientation, and saved in a text (<filename>.txt) file format.
Although the SPEAR text files are based on the analysis of a signal with the aid of FFT's and as such belong to the domain of Digital Signal Processing (DSP), the same type of consistency in processing as previously realized with the Standard MIDI files (SMF), can be achieved by using the same principles of sequential labelling[55]. By using a strict working method as described above, it is now possible to incorporate more elaborate ideas about musical functionality. If abstract ideas such as chord-builder, melody-creator or melody-imitator are taken into account, then musical imagination is the only limitation. This is demonstrated by the development of the CLUS (Clusterer) object (see section 5.5.1, page 54) and with the implementation of the Correlator (see section 5.5.5, page 67), where certain musical functionality can indeed be simulated (e.g. The Clusterer can be used as a kind of chord generator). In the case of STAPS (see section 5.5.4, page 64) and STAM objects (see section 5.5.2, page 56) however, abstract mathematical processing has no direct analogy in the world of Music and as such, can only be used in a statistical way, not related to a real musical process.
By using these two approaches, creating abstract musical tools from common Information Retrieval techniques is indeed possible and can be used as one of the CACE4 objects needed for creating the correct strategy for obtaining the desired musical result.

[55] CACE4 in the version: 0.5.x has no possibility for using a context. For now the user must remember which entry (dimension) should stand for which musical parameter. Taking these limitations into account a future version of CACE4 will have this option. See for more details Chapter: 7.2 (page 129), Future development plans.

Chapter 4
CACE4: design and analysis

This chapter discusses the design of CACE4 as a music composition computer program, focused on the use of statistics, IR and data mining as compositional tools. It incorporates many ideas about software functionality and software design. The main goal for the design should be the original idea about the functionality of the program: its intended purpose and use as it is reflected in the original design of the software package. This implies a certain context in which one has to operate. By not using a (music) language-based approach, one must decide what else to use as a fundamental core-process of information exchange. CACE4 uses numerical streams, instead of other techniques such as an Augmented Transition Network.

Although ATNs are constructs that can also be used to create atonal music, they are, in the case of CACE4, not applicable while the definition (core engine) and functionality is not based on a Music language construction. The internal data structure, as communication between independent software components (CACE4 objects), is restricted to a one-dimensional stream for applying (mathematical) functions, therefore a construction of an ATN would not be necessary. These different approaches in design will lead to different outputs and are thus an important factor in our design.

The two design criteria however, must first be defined. These are both artistic (Can we create musical interesting material with it?) and technical (Which IT techniques are involved?). The technical criteria provide the framework and determine the possibilities of translating the musical ideas into software. For reasons of technical feasibility and to be able to develop the ideas for the program, different existing development techniques have to be taken into consideration. Computer Programming Language, Object Oriented Programming (OOP) techniques and Object Oriented Wrapper Class design are a few to name. All these different criteria, with all their different approaches, have to be taken into account in the early stages of the design and development of the software program. The original goals of the software have to be translated into the design of the program, by means of the technical (IT) design techniques available. This all distils into a design and a working method for developing the CACE4 software package.

4.1 Design and development criteria of CACE4.

Although roughly based on an older composition program (CAC1 – 1996) and making use of some previously programmed functions, the design of this new version has been from scratch. A new LISP IDE with new tools and libraries for the GUI has been used. These tools offer to adapt different approaches in program development and programming style. Keeping all these criteria in mind, they now can be combined with the goal and create a workspace for design and development.

The four major design criteria:

Firstly: open-ended software design. As music is a ‘never ending story’, there should always be room for new ideas about music, compositions and informatics (section 4.1.1, page 20).

The second criterion is: easily extendable and modular design. The most suitable approach is an Object Oriented and uniform class design (section 4.1.2, page 21).

Our third design criterion focuses on educational purposes, in particular focusing on a graphical display (GUI) so that students can get acquainted with (basic) concepts of statistics, mathematics, informatics and music, where the focus is on software design and application programming (section 4.1.3, page 21).

The fourth and last criterion is to make it suitable for creating contemporary (acoustical and electronic) compositions (section 4.1.4, page 22). Aesthetic criteria are involved and mostly these criteria are difficult to qualify. They can therefore only be tested subjectively. In chapter 6, three compositions actually created with the aid of CACE4 are analyzed and its artistic implications discussed.

4.1.1 Open-ended software design.

As one of the first and major software design criteria, the software package should be an open-ended computer program. The design of the object system and the GUI of the software is a so-called framework application. This specific software development technique makes it possible to have a freer design, based on the design principles of creating an (object-oriented) application framework[56] first. The engine and underlying core of this 'open' framework approach is based on the technique of Object Oriented Programming. This makes further development and adaptation possible, quick and rather easy. Because CACE4 is programmed in the computer language Common LISP, it can make use of the CLOS (= Common LISP Object System) extension for this particular LISP dialect. Therefore, the boxes (or cases) as they are shown in the program, do represent to a certain extent, object oriented classes as well. Using this programming technique, the design of CAE4 is brought back to two major classes, each with their particular GUI: one class for text displaying and another class for more elaborated graphics and views. Both prepared as template documents, they now can be quickly adapted to specific needs for newly added classes and their specific associated processes (methods and functions).

[56] A framework application can be defined as an application build out of reusable software components (The application framework). ‘Wrapped’ in these frames (mostly GUI based), objects can easily be embedded into the application environment as such. See for more details about the topic:
http://www1.cse.wustl.edu/~schmidt/CACM-frameworks.html

4.1.2 Easily extendable and modular design.

CACE4 is not based on a musical language description. The major design-criteria are based on a 'plain', isolated and mostly mathematical description of a specific process. Every object should have just one (or in one special case, more then one) input-stream represented as a list that contains numbers. These numerical series - mostly in a one or two dimensional (x,y) number pairs format - are combined into one newly created output-list after altering by processing and calculation. By doing so, we have a single, uniform design based on isolated objects, represented by boxes (or cases), with one input and one output connector[57].

Figure 2 A Single CACE4 object box, in this case: a DATA Manipulator object with an Input (in) and an Output (out).

Each box, as shown in Figure 2, can thus be seen as a separate process with its specific GUI suited best for visualizing and working with the specific process. This framework approach does not a priori exclude the use of a musical language description, but, if implemented, it should be programmed instead as a separate process in one of these boxes together with their own unique set of tools. This software development technique creates the possibility of fairly easily extending CACE4 and incorporating other software libraries when necessary. Further incorporation of existing software modules originally in C/C+ can be achieved with the use of the FLI (= Foreign Language Interface), a module already available in the LISP environment (LispWorks - IDE), and as such, will be used for incorporating existing C/C++ DSP libraries (e.g. AudioLAB2[58]).

4.1.3 Educational purposes.

[57] The only exception, for now, is the Merger object. The Merger object can handle up to 12 inputs (for further explanation see Chapter 5.3.2).

[58] AudioLAB2 is a C/C++ Digital Signal Programming (DSP, see footnote 58, for a further explanation of DSP), command line computer program, developed by the author, which was originally used for composing the electronic 43-loudspeaker composition *Ploutôn*: a commissioned work (NFPK: 2010, for further information: http://www.nfpk.nl) of 25 minutes duration. It is a fixed media composition (24-tracks), about Minerals and their chemical composition, and our perception of wealth (Ploutôn or Pluto are the Latin names of Hades: a Greek God). This composition was specially made for the ZKM 'Zirkonium' loudspeaker setup (see for details about the Zirkonium: https://www.zkm.de/zirkonium). AudioLAb2 has also been used in lectures about developing DSP application software, for undergraduates of the Music Technology department of the University of the Arts Utrecht, the Netherlands.

Another, but rather important criteria for the design and development of the CACE4 program is for making use of it as an educational tool. Therefore the program should be well equipped and suited for educational purposes. Whilst CACE4 has a strong visual component as intended by design, it makes use of transforming and plotting the data in its own separated output in several ways, mostly represented by graphics (different kinds of plotting techniques are used), or, in some cases, by text fields. This isolated solution of representation of input and output data, best fitted for the process it represents, makes it possible to make use of these processes as separated, boxed objects. Therefore it should be useful, not only as a musical composition environment, but also to act as an educational tool. As a lecturer at the University of the Arts, Utrecht, The Netherlands, I teach undergraduate students in computer software development, mainly in the area of Music Processing and Music Software Development and to a lesser extend DSP. This program, the way it is now, gives me the opportunity to explain certain, sometimes rather complicated processes, initially isolated from other areas. For example an algorithm as *k*-means (a Hierarchical Cluster detection algorithm), a common and rather 'simple' technique from the field of Information Retrieval, will be initially presented as a so-called 'black box' process. This way its use in a more musical context can be more quickly focused upon without having to first gain the in-depth mathematical knowledge needed to understand how it actually works and operates (this can/will be explained at a later stage). Other techniques necessary for constructing musical output for listening, such as how to construct SMFs, is also, in this context, another isolated problem. (The more technical details about the MIDI protocol and SMFs can be explained later to the students.)

With the aid of a few extra modules such as DATA and Mathematical Transformer objects, it is now possible to 'redirect' the obtained output to a more musical format. At the moment this is mainly done in the SMF[59] format, although music notation in MusicXML (MXML) and LilyPond are, to a certain extent, prepared and in the near future will be made available as new modules. The students will be able to hear the output of the process by playing the SMF generated.

4.1.4 Suitability for creating contemporary (acoustical and electronic) compositions.

This is the last but not the least of the four major design criteria for CACE4. As a professional composer, I need a good package of tools for composing algorithmic music. Although many interesting solutions are offered in different software packages (e.g. OpenMusic[60], ACToolBox[61],

[59] Standard MIDI files (SMF for short) is a specific file format for storing MIDI data according to the specifications defined by the MIDI association. URL: https://www.midi.org/specifications/item/standard-midi-files-smf

[60] OpenMusic is developed by IRCAM (Institut de Recherche et Coordination Acoustique/Musique, Paris France). Further information can be found at: http://repmus.ircam.fr/openmusic/home

athenaCL[62] and Grace[63]), I prefer to write my own (more than thirty years of computer programming experience in the domain of Computer Music makes the real difference). Another even more important reason is to be able to work in an all-purpose programming language for expressing my ideas about music software design in more detail. CACE4 is much more restricted to mathematical analysis of numerical data in order to find certain numerical properties. Information retrieval and machine learning algorithms are centred in the way CACE4 operates. In this way CACE4 is significantly different than the other previously mentioned composition programs.

4.2 Technical Development criteria.

Other major development criteria and work concerns, besides the previously stated four design criteria, the development criteria. These criteria are more incorporated into the specific development platform called the Integrated Development Environment (IDE). This can be seen as the programming language embedded into a software program development program, with all the development tools it has to offer as such.

None the less, these other criteria are important and, throughout the development of the software package, they are gaining in importance. For example, the development of a software package based on Object Oriented Programming (OOP) techniques, without the aid of a so-called class browser, is, without the necessary class overview, painstakingly difficult. Furthermore, as the program continues to enlarge it is, in the long term, not really manageable.

4.2.1 Computer programming language.

Taking into account our criteria for extendability, multi-platform and ease of development, the choice of computer language was limited to an object oriented one (OOP). Our initial choice therefore, would be C++, C#, Smalltalk, or LISP. I chose LISP because it is known for its 'rapid prototyping[64]' possibility. It has an OOP extension (CLOS, which stands for Common LISP Object System), which gives it easy extensibility and re-usability of the software code.

Also Object Oriented Wrapper Class Design: for easy extendibility of our composition program. This is a software design issue and should be defined by Class design and embedded in software code.

The choice of LISP as the programming language will be explained in further detail in section 4.2.6

---

[61] ACToolBox has been designed and programmed by Paul Berg, and has been developed in Lisp (latest version: 4.5.6 (2014). Additional information and downloads: http://www.actoolbox.net

[62] A composition software application designed and programmed by Christopher Ariza and published in *An Open Design for Computer-Aided Algorithmic Music Composition* athenaCL (Ariza 2005).

[63] Grace has been designed and programmed by Rick Taube. The implementation is described in his book *Notes from the Metalevel* (Taube 2004).

[64] This could be a point of discussion: how is rapid prototyping defined? In this case, Lisp offers the possibility of testing software algorithms in the Listener, which acts as the interpreter. The interpreter acts as a continual kind of compilation, and as such speedups the development of new software ideas and implementations.

4.2.2 IDE, the Integrated Development Environment.

During the selecting of the right development environment, public domain LISP software packages[65] were taken into account. They were rejected however, on the basis that a commercial package would be a better candidate, as there is more guarantee that, in the near future, it would be maintained and further developed for new developments in specific Operating Systems[66]. As well as this, quick responses to specific technical questions and a good backup system, with the aid of a rather large user group with a good functioning newsgroup with its own mailing-list and its use by major manufactures of the aeronautical industries such as Boeing and NASA, gave some security in making it the right choice of platform.

The choice of LispWorks with its extended IDE, there being no real other comparable competitor on the market, makes it pleasant and easy to work with. Truly preferable in working with this IDE is that it is fairly simple to extend and adapt the working environment. CACE4 has, from its early initial stage of development, its own place in the Menu of the IDE. So it is rather easy to switch back and forth between the LispWorks IDE (the Listener) and CACE4. There is always a fully functional LISP Listener in the LispWorks environment, which speeds up the design and development of difficult algorithms. Also the necessity of compiling each time before running and testing the algorithm is, in an interpreted language as LISP, no issue. This results in a much shorter cycle of development compared to a compiled language (e.g. C/C++ etc.).

The output of error message can be, especially in an Object Oriented computer language, cryptic and initially hard to understand; this is also a well-known issue with other object oriented languages such as C++ and C#. Another important issue is that the LispWorks IDE comes in different 'flavours'. Also it is truly a multi-platform IDE, as there are versions available for MS-Windows, Linux, Unix-BSD and MacOSX). In order to get acquainted with the IDE, there is an entry-level version of LispWorks Personnel Version, free from any charge[67] and fully functional except for two things: the total working time in the IDE has been restricted up to 4 hours in a row, (then the work has to be saved and IDE quitted before getting back to the work) and the amount of active RAM allocation of the IDE has been restricted. Besides these two restrictions it is a fully functional, with all the CAPI (GUI libraries) and other LISP libraries and program features, software development environment.

---

[65] As there are: CLISP (URL: http://www.clisp.org), Closure CL (URL: http://ccl.clozure.com), Quicklisp: (URL: https://www.quicklisp.org/beta/) and CMUCL (URL: https://www.cons.org/cmucl/).

[66] The multi-platform approach is also very well organized in LispWorks. By using the ANSI standard for Lisp programming, the application can run under MacOS, Windows and Linux. Depending on the License obtained from Lispworks.

[67] This is an important consideration for using it for educational purposes as well, students have easy access to this free entry-level version.

4.2.3 MVC, the Model View Controller paradigm.

Most of the GUI coding has been done based on the Model - View – Controller (MVC) principle (paradigm) and design[68]. This means that there is a strict division of functionality for processing and mathematical purposes (= The Model), the View (how to represent the data) and the necessary

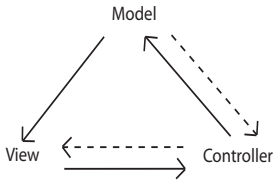


Figure 3 The MVC programming paradigm.

Controllers such as: editable text fields, buttons, sliders and menu's etc., to interact and control the program. These three corner stones of the MVC paradigm are combined in a GUI (= Graphical User Interface), to make the program overall easier to handle by a user. To integrate this software programming technique and the use of CLOS means that *defclasses()*, and their associated *defmethods()* must be used, sometimes defined by *defgenerics()*. The Controller interacts with the Model. The Model in turn, interacts with the View.
The reverse direction is also possible between Controller and View. The MVC paradigm can easily implemented directly in OOP techniques, by making use of superclasses and subclasses. For creating the controllers and the view, as part of the GUI, the use of CAPI[69] was essential. The Model makes more use of LISP functions but is also defined in a combination of superclasses and subclasses of its own. The View has its representation by slots[70] and panes of the CACE4 Object, which are used for creating the GUI. And the Controllers, as part of the GUI are also, separate subclasses as provided by the LispWorks CAPI Library[71].

4.2.4 GUI, the Graphical User Interface.

The whole design of the Graphical User Interface (GUI), apart from its incorporated MVC model, is based on the principle of flexibility of data representation. Two important types of data representation can therefore be distinguished: text based, which is mostly used as a representation for numbers or elongated lists of numbers and graphical representation, which provides the user with

[68] Look for further details on the MVC programming technique: https://msdn.microsoft.com/en-us/library/ff649643.aspx
[69] CAPI is an extended library for GUI implementation. All classes are equal to classes used in CLOS. For details about the use of CAPI and other libraries visit: https://wwwlispworks.com/
[70] A slot of an Object can be seen as an exclusive storage space for literally anything in Lisp. "Object-oriented Programming in Common Lisp" by Sonja E. Keene (Keene 1989) offers a detailed description about slots and all the other aspects of CLOS programming.
[71] For further details see Chapter 4.2.7 where an UML diagram shows class design and class dependencies.

pictures and diagrams. This is mostly done by plotting in two dimensions and grouping the data into xy number pairs. To be consistent throughout the CACE4 program, using the same colouring has visually supported the grouping of certain classes in objects of the same functionality.

Superclasses:	Subclasses:	CACE4 Box object colour representation:
CACE-MATH-generator-GUI ()	All fractals and attractors. Random & Tendency.	:*blue*
CACE-FILE-generator-GUI ()	*spear-partials-text-file ()* *text-file ()* *midi-file ()* *mxml-file ()*	:*lightblue*
CACE-AI-manipulator-GUI ()	*art2 ()*	:*orange*
CACE-DATA-manipulator-GUI ()	*pruning ()* *merging ()* *sorting ()* *splitting ()*	:*palevioletred*
CACE-MATH-clusterer-GUI ()	*math-clusterer ()*	:*orangered2*
CACE-MATH-manipulator-GUI ()	*correlator ()* *scaling ()* *disturbance ()*	:*orangered3*
CACE-MATH-sieve-GUI ()	*math-sieve ()*	:*pink*
CACE-MATH-property-sieve-GUI ()	*math-property-sieve ()*	:*orangered1*
CACE-ML_MIR-manipulator-GUI ()	*expectation-maximization ()* *k-means ()*	:*red*
CACE-score-GUI ()	No subClasses	:*grey*
CACE-processor-GUI ()	No subClasses	:*black*
CACE-project-GUI ()	No subClasses	No Box
CACE-translator-GUI ()	*miditranslator ()*	:*purple*
CACE-informer-GUI ()	*informer ()*	:*green*

Table 3 Table of all CACE4 superclasses and subclasses. The first column are superclasses defined by: *capi:define-interface()*. The second column are defined as CLOS classes: *defclass()*.

4.2.5 OOP, the Object Oriented Programming software development technique.

Common LISP Object System (CLOS) as the Object Oriented Programming extension for use with Common LISP makes it easy (with an initialize-instance method call) to extend basic classes, mostly superclasses with extra or altered views. This can be done rather easily, by writing extra methods for these subclasses. Dummy slots and panes (= defined view area in the CAPI library) in the initial superclass are replaced by newly defined extra interfaces to suit best for this particular subclass.

As previous stated, working with CLOS means that there is a specific approach in organizing the software on the coding level. The operation of the software and the GUI are fixed in accordance with the rules of CLOS to the use of classes and their methods. Most interaction between GUI using the buttons and editable text-fields, is done by *defgeneric()* methods[72] (Keene 1989), with their behaviour defined in LISP coding. The user interacts with instances of classes, defining the GUI and the methods belonging to these classes by using these controllers, thus clustering the same functionality of interaction of all the different subclasses grouped together into a single *defgeneric()* method, each with their own, specific class-dependable algorithm (mostly checking a few details as checking ranges of arguments). This last method is also part of an Object Oriented programming technique. This OOP technique of encapsulating data and functionality is a core Object Oriented programming paradigm, providing a framework for flexibility and speed during the development stage of the computer application. As such, it is of major importance for developing software in a less rigid, more flexible way.

4.2.6 The choice of a programming language; why LISP?

There are many reasons for the choice of LISP as the underlying computer language and not for example, modern industrial languages as C/C++ or C#, especially when using Apple computer's Objective-C as the preferred development platform. The major advantages of flexibility and extensibility as found in a general-purpose computer language, is also present in LISP. Symbolic manipulation (for example note names as strings and their numerical representation) can be combined in any way applicable for the software, without worrying at first too much about more low-level technical aspects such as typecasting and memory allocation. In LISP this is the standard way of working. The way it is incorporated into the design of the computer language makes it an ideal language for describing processes as functions and, in our case, the use of associated musical symbols, systems and structures. The plentiful availability of large libraries for mathematics and many more topics is also of importance. The LispWorks libraries for Graphics and GUI design (CAPI) and a strong ANSI standard[73] of this particular LISP dialect (Common LISP/Harlequin LISP) made it the right choice. Other options would have been to design and program CACE4 straight away into the Max/MSP environment or even MatLab[74], but these were not found to be applicable for this job[75]. In

[72] The book by Sonja E. Keene is still regarded as one of the important books about CLOS (Keene 1989). And also the book by Lawless and Miller: *Understanding CLOS, The Common Lisp Object System*. (Lawless and Miller 1991) Offers detailed description of all features implemented in the CLOS standard.
[73] The ANSI standard: X3J13.
[74] MatLab: the software-tool for doing calculations in the domain of discrete mathematics. Its software application is widely used in the world of science and engineering.
The official web-site: http://www.mathworks.com/products/matlab/index.html?s_tid=gn_loc_drop
[75] For future development purposes the LISP language possibilities as the use of association lists are needed for implementing certain feature of a Music language and context. This is explained in more detail in Chapter 7.2 (page 129), Future development plans.

the case of Max/MSP, real-time was not an important option and all of the algorithms had to be written as external objects (in the computer language C). This could have limitations for the algorithms. For example in the case of 2^n-algorithms[76], it would block the whole (real-time) functionality of the Max program. These specific types of calculations can consume a lot of computing power and time, depending on the amount of data to be analyzed - 2 to the power of n. Also in most cases the calculations can only be applied on the data set as a whole, which is a further limitation if the concept of CACE4 has to be programmed into this real-time environment.
From the developer's point of view, the option should be kept open for future extension of the software package with new specific modules that would make use of certain LISP functionalities or a specific technique of programming as, for example in the case of 'frames': a small A.I. program for demonstrating a specific technique of frame using as designed by P.H. Winston, and described in his book *Artificial Intelligence* (Winston 1984).

A general-purpose functional[77] computer language, with all the (extended) libraries as designed in LISP, was the only option for having this kind of flexibility and extensibility I needed for programming CACE4 and all of the designed CACE4 objects. As previously stated, that LISP is an interpreted language and therefore the whole development cycle of editing/debugging and testing is much faster than with a compiled language, makes it, in my opinion, the only right computer language to use for this kind of software package.

At the moment the focus of CACE4 is more on the use of Artificial Intelligence processes: ART2 (Adaptive Resonance Theory – by Carpenter and Grossberg (Carpenter and Grossberg 1987) et al.) and Information Retrieval processes such as *k*-means, EM (= Expectation Maximization), statistical analysis and a few other processes for sorting, deleting and scaling of the data to make it scalable for MIDI use. It also has a rather large Generator group of Attractors and Fractals. This group has been incorporated due to historical reasons: it was a group of fundamental generative algorithms in the older versions of the CACE programs and is still interesting to use for generating data. The design of CACE4, as it is for now, makes it possible to easily extend it with other software modules from other, interesting areas. These can cover totally different concepts. Even ATN's, as described by Cope[78] (Cope 1991) (Cope 2001) (Cope 2005), or other ways of using a Music language context, are possible, as long as the developed Wrapper Classes[79] are used and one single 'stream' for both input and output is sufficient for the process.

One other important consideration (for the choice of LISP) is that although many different computer languages have been used, LISP has a strong tradition as a computer language well suited for creating

[76] K-means is such a type of algorithm.
[77] As opposed to a Procedural languages as there are: C/C++, C# or Objective-C.
[78] All of the computer programs designed and developed by Cope are written in LISP.
[79] Wrapper classes are provided as a template for an Application Program Interface (API). By putting shared functionality in the wrapper class, development of the software as such, speeds up.

algorithmic compositions, as well for the development of computer assisted composition environments. From the mid 1980's, several music composition software packages were developed: FORMES (1984) is amongst them. Developed by X. Rodet & & P. Cointe at IRCAM, Paris. It focuses on composition and scheduling of processes. One of the earliest commercially available software packages was Symbolic Composer (SCOM): a software package existing for almost 30 years (release in 1991).

In the following 10 years many different packages were developed, all using LISP as their core programming language. Some notable examples are: Common Music (CM) of Heinrich Taube; an object-oriented music composition environment introduced in 1989. In 1990, CLM (Common Lisp Music) was created by William Schottstaedt as a sound synthesis package for CM. In combination with his CMN (Common Music Notation), the use of music notation and score printing became available. Paul Berg released the first version of the AC Toolbox in 1992, a LISP based computer composition environment. The AC Toolbox main focus is on the use of different types of generators for creating notes, note structures, masks and sections.

David Cope took a somewhat different approach with his EMI (Experiments in Musical Intelligence), originally released in 1996. EMI is AI software (pattern recognition) for music analysis and the creation of compositions based on these analyses.

In the same year (1996) IRCAM released Patchwork a computer composition environment originally created by Mikael Laurson, Jaques Duthen and Camilo Rueda. Patchwork has a strong focus on the GUI and the use of graphics. In 1997 OpenMusic, also from IRCAM (Gérard Assayag, Carlos Agon and Olivier Delerue), was released. OpenMusic was developed even further into a graphical/visual style of composing computer music, with a strong focus on the use of a GUI. Furthermore, in 2002 Mikael Laurson, Mika Kuuskankare and Vesa Norilo developed the idea of Patchwork into PWGL (PatchWorkGraphicsLibrary).

All these Computer Assisted Composition environments (see Table 4, page 30), were programmed with LISP (or in the case of Grace: Scheme) as their core computer language.

Name CACE (Alphabetical order).	Author(s).	Year of release.	Remarks and URL.
AC Toolbox	Paul Berg	1992	A Computer Assisted Composition Environment. http://kc.koncon.nl/downloads/ACToolbox/
AthenaCL	Christopher Ariza	2005	An Open Design for Computer-Aided Algorithmic Composition. (Nowadays rewritten for python). http://www.flexatone.org/athena.html
CLM (Common Lisp Music)	William Schottstaedt	1990	Sound synthesis package. https://ccrma.stanford.edu/software/clm/
CMN (Common Music	William Schottstaedt	1994	Music notation package. https://ccrma.stanford.edu/software/cmn/cmn/cmn.html

Notation)			
CM (Common Music)	Heinrich Taube	1989	Object-oriented algorithmic music composition environment. http://commonmusic.sourceforge.net At a later stage Grace (Graphical Realtime Algorithmic Composition Environment), programmed in JUCE (C+) and S7 Scheme, was added to CM.
EMI (Experiments in Musical Intelligence)	David Cope	1996	AI software for music analysis and composition. http://artsites.ucsc.edu/faculty/cope/experiments.htm
Flavors Band	Christopher Fry	1984	Originally a LISP-based music language. http://www.algorithmic.net/system/flavors_band
FORMES	X.Rodet & & P. Cointe	1983/4	IRCAM: Software for Composition and Scheduling of Processes. Based on a Object-oriented language for synthesis and music composition (in VLISP). See for more information: http://articles.ircam.fr/textes/Rodet85a/
Nyquist.	1997	Roger Dannenberg	Music and synthesis programming environment. https://www.cs.cmu.edu/~music/nyquist/
OpenMusic (OM)	Gérard Assayag and Carlos Agon with Olivier Delerue.	1997	IRCAM: MIDI, audio, symbolic notation. https://www.ircam.fr/transmission/formations-professionnelles/openmusic/
Patchwork	Mikael Laurson, Jaques Duthen and Camilo Rueda.	1988	IRCAM: MIDI and symbolic notation http://recherche.ircam.fr/equipes/repmus/RMPapers/CMJ98/
PWGL (PatchWork-openGL)	Mikael Laurson, Mika Kuuskankare and Vesa Norilo.	2002	Visual music programming environment http://www2.siba.fi/PWGL/
SCOM (Symbolic Composer)	Peter Stone	1991	http://www.symboliccomposer.com/page_main.shtml

Table 4 Overview computer assisted composition environments. Used sources: Tim Thomson: http://nosuch.com/tjt/plum.html and Paul Doornbush: http://www.doornbusch.net

Thus after careful consideration, LISP as the computer language of choice was self-evident.

4.3 Design Analysis of CACE4.

The two major groups of objects in CACE4 are Generators and Manipulators. Certain aspects of their design and functionality in the CAE4 program will be analyzed in more detail. The two groups directly reflect the idea of first generating and secondly, manipulating the (input) data, as two separate entities. Their only connection is by exchanging data as a numerical stream (output).

4.3.1 The CACE4 Generators.

The Generator is a group of CACE4 objects used in the beginning of every CACE4 chain of objects. They are necessary in order to generate initial material as a single stream of numbers. There are two major CACE4 Generators groups to select from, as can be seen in Figure 4:

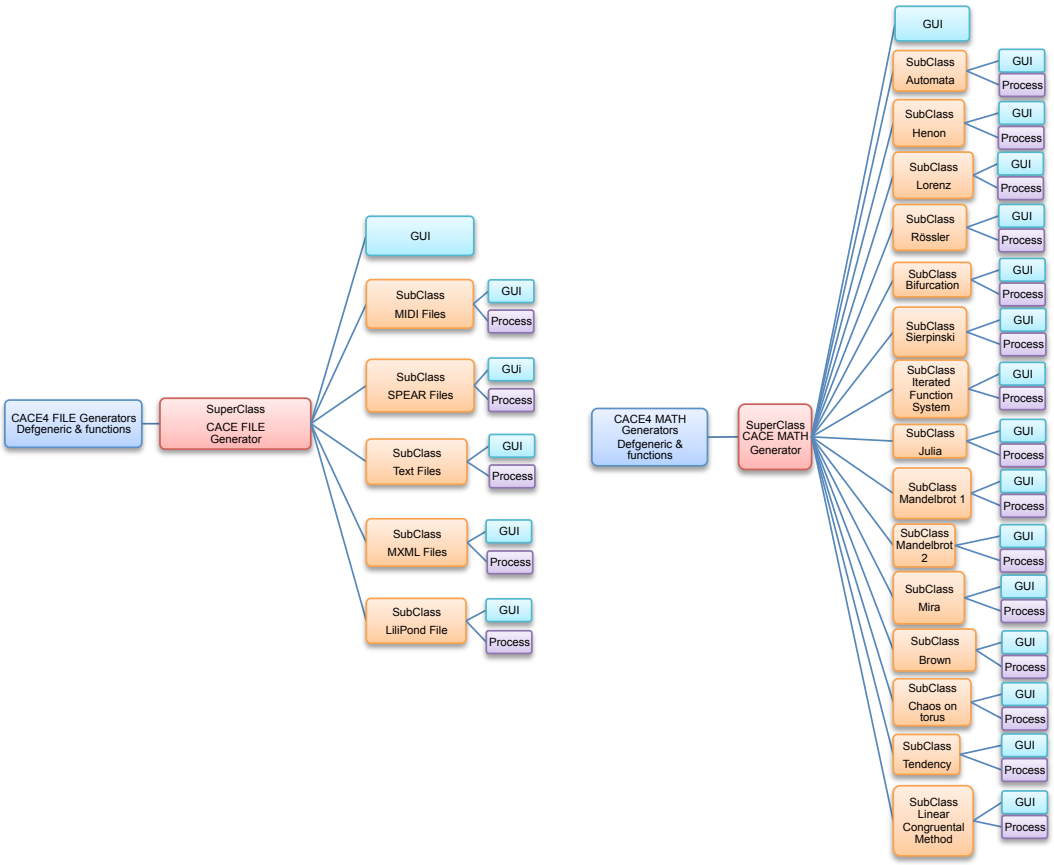


Figure 4 The CACE4 Generators and their CLOS Class dependencies.
Superclasses are in red. All processes (purple) are methods and functions. GUIs (light-blue) are subclasses.

All of these Generator objects act as a single initial source of data for other Manipulator objects (see section 4.3.2, page 32). In the case of selecting a CACE4 FILE Generator as our initial Object, a number sequence from a file in ASCII format (text-based) is used as a Generator of data. This can either be a plain text file: in most cases numbers in (x, y) pairs, or a SPEAR partials text file. Also the use of a SMF (Binary) format 1, as initial data source is possible[80]. In all three cases the file will be read in and displayed in a graphic plot format. At the right hand of Figure 4, a large group of fractal and attractor calculations can be seen: they can be chosen as well and used as initial data generators. This shared functionality is reflected in the design of the underlying class dependencies and shared behaviour (by *defmethod()* and method combination). The two superclasses (in red, see fig. 3) define

[80] For now Music XML (MXML) and Lilypond file format are not yet available.

for a large part the GUI and functionality of plotting aspects of the displayed data. By using the computer programming technique of defining 'defgeneric' combination of methods and functionality for 'low-level' plotting aspects elements such as pixel-size, zoom factor and positioning of the plot are shared. All subclasses (in orange, see Figure 4) inherit these more global GUI parameters, by being a subclass of either one of these two superclasses. Every subclass does have its own set of methods for dealing with parameters values on the GUI level (in blue). For executing a calculation (also called Process: in purple), other more subclass specific methods and functions are used. By making use of these object oriented programming techniques of sharing large blocks of code by inheritance of methods of use, more compact computer coding is achieved.

4.3.2 The CACE4 Manipulators.

The group of Manipulators is the largest group of CACE4 objects. They act as manipulating objects of the Generator objects (original data stream) or they act (in a chain) on a Manipulator object as well.

There are four superclasses (in red, see Figure 5, page 33): AI Manipulator, ML & (M)IR Manipulator, MATH Manipulator and DATA Manipulator. Just like the Generator objects, they share functionality for plotting by making use of defgeneric method combination. Subclasses (in orange) inherit from either one of these four superclasses and are in this implementation of CACE4 closely related in definition and behaviour.

By grouping certain approaches into these design groups, software design and thus computer programming is made much easier. From a mathematical point of view, the organisation of the software by grouping certain classes around mathematical processes and calculations, as is in the case of the Generator objects (fractals, attractors and etc.), does make sense. In the case of the CACE4 Manipulator objects, the four Manipulator groups consist of entries grouped around a certain design and functionality. The latter property can be derived from a domain of Informatics (AI: CACE4 AI Manipulators, MIR and IR: *k*-means and Expectation-Maximization), or, in case of the CACE4 DATA Manipulators, more freely according to their functionality; examples being Pruner (a deleter), Merger, Splitter and Sorter CACE4 objects.

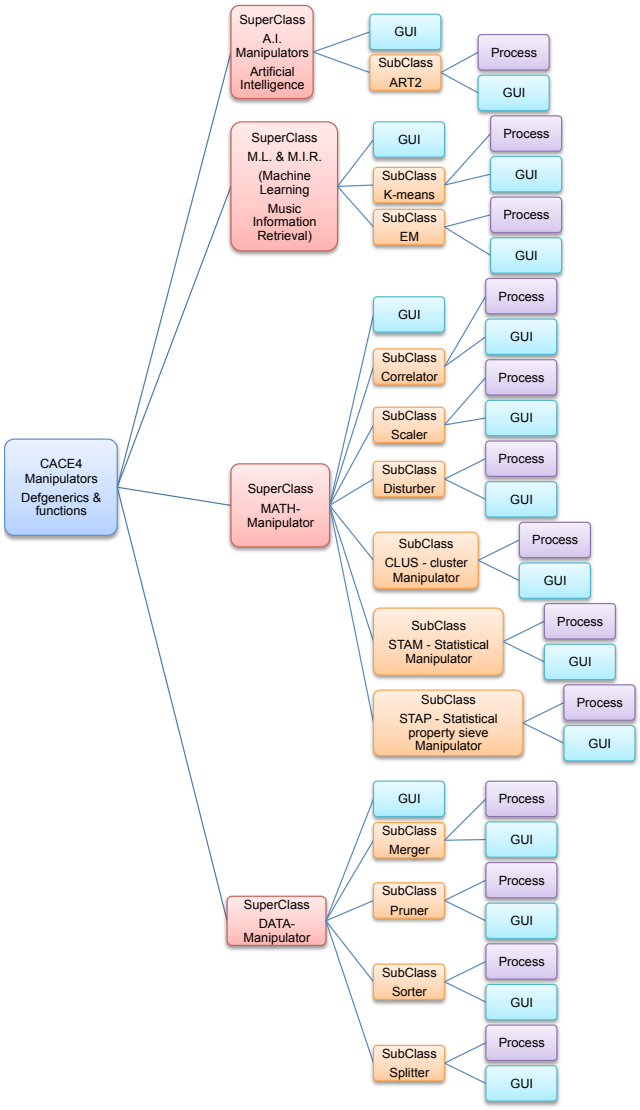


Figure 5 The CACE4 Manipulators and their CLOS class dependencies.
Superclasses are in red. All processes (purple) are methods and functions. GUIs (light-blue) are subclasses.

Extension and alteration however, are always possible. All subclasses have their own process, shown in purple (see Figure 5) and specific GUI-elements are shown in light blue. Specific mathematical algorithms are defined as a *function()*, or as a *defgeneric()* method, in order to encapsulate the data type (e.g. multiple sources defined as different types: strings, numbers, arrays and lists). For example the hypotenuse can be calculated for just two sides, or for a whole list of sides at the same time. The consequence of applying this programming technique is to achieve compacter and much easier adaptability of the coding[81].

[81] In future versions of CACE4, further use of this implementation technique will be extended into larger parts of the CACE4 (function) Libraries.

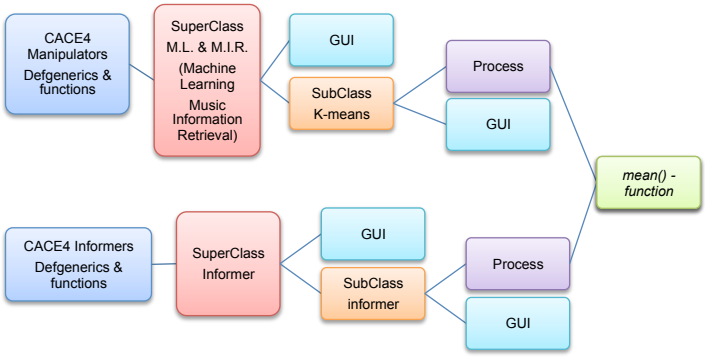


Figure 6 Example of class dependencies and methods with shared functions.
The following colour coding has been applied: superclasses in red, subclasses in orange, GUI in light-blue, methods in purple and (shared) functions in green.

Figure 6 shows an example of class dependencies and shared functions. *Mean()* as a LISP function, is used by a CACE4 Manipulator Object[82]. But the same function *mean()* is also used in a CACE4 Informer Object[83].

4.3.3 List of all objects and functionality in the software package.

Table 5, page 35 shows a list of all 36 objects (and process algorithms) implemented so far (December 2015). In the first column (Group) we find 6 groups. The first four Group members (CACE4 Generators, CACE4 Manipulators, CACE4 Translators and CACE4 Informers) can be used in a Processor window as part of a strategy chain as can be seen in a processor window (e.g. Figure 9, page 37, shows a possible strategy chain).

Group	Sub-group	CACE4 box-object
CACE4 Generators	Fractals	Automaton
		Bifurcation diagram
		Brownian movements.
		Chaos on Torus
		Iterated Function System (IFS)
		Julia
		Linear Congruential method
		Mandelbrot 1
		Mandelbrot 2
		Gumowski-Mira
		Random Cloud
		Tendency masks
	Attractors	Henon type 1
		Henon type 2
		Lorenz attractor
		Rössler attractor

[82] The order of *class()* dependency; all the objects are directed to their *superclass():* CACE4 Manipulator <- ML/MIR (Machine Learning/Music Information Retrieval) <- k-means <- mean.
[83] The order of *class()* dependency; all the objects are directed to their *superclass():* CACE4 Informer <- Informer <- mean.

	Files (input)	Text files
		Spear partials text file
		Standard MIDI file (SMF)
CACE4 Manipulators	A.I.	ART2 (Adaptive Resonance Theory 2, Neural Network)
	ML/MIR (Machine Learning/Music Information Retrieval)	*k*-Means. (Hierarchical Cluster Techniques)
		Expectation Maximization. (HCT)
	Data Manipulators	Pruner
		Merger
		Sorter
		Splitter
	MATH(ematical) Manipulators	Correlator
		CLUS (Clusterer)
		Disturber (Disturbance).
		Scaler (Scaling).
		STAM (Statistical Manipulation)
		STAPs (Statistical Property Sieve)
CACE4 Translators	Translators	Translator (MIDI Translator)
CACE4 Informers	Informers	Informer/Viewer

Table 5 List of all 34 CACE4 Processor objects for using in the Processor window (CACE 0.56 - December 2015).

Table 6 shows the two remaining CACE4 objects belonging to the CACE4 Project object with a separated display. Initially when a new project is started, at least one CACE4 Project object needs to be added to the Project window. Later, more CACE4 Processor objects and Project Score objects can be added. For now, only one CACE4 Score object is necessary in order for the program CACE4 to function properly.

Group	Sub-group	CACE4 box-object
CACE4 Processors	Processors	Processor
CACE4 Scores	Scores	Score

Table 6 List of all CACE4 Project objects (2) for using in the Project window in version CACE4 0.56 (December 2015).

4.3.4 The CACE4 Project object and display.

This is the first window-display when a new project is selected in the CACE4 menu. After opening, the user must select a new CACE4 Processor object from the left <Add Processor box>. After creating a strategy with multiple CACE4 objects inside this Project Processor object, the output of the strategy can be translated with a CACE4 Translator object.

For displaying and saving the calculated output to a file, a CACE4 Score object must be created in the CACE4 Processor window. By doing so, the user is now able to save the output to a SMF. As a main technical design feature, the purpose for this approach of grouping together is to keep several CACE4

Processor objects visually tied with their (associated) Score object. This is also directly reflected in the GUI of the Project Window (see Figure 7).

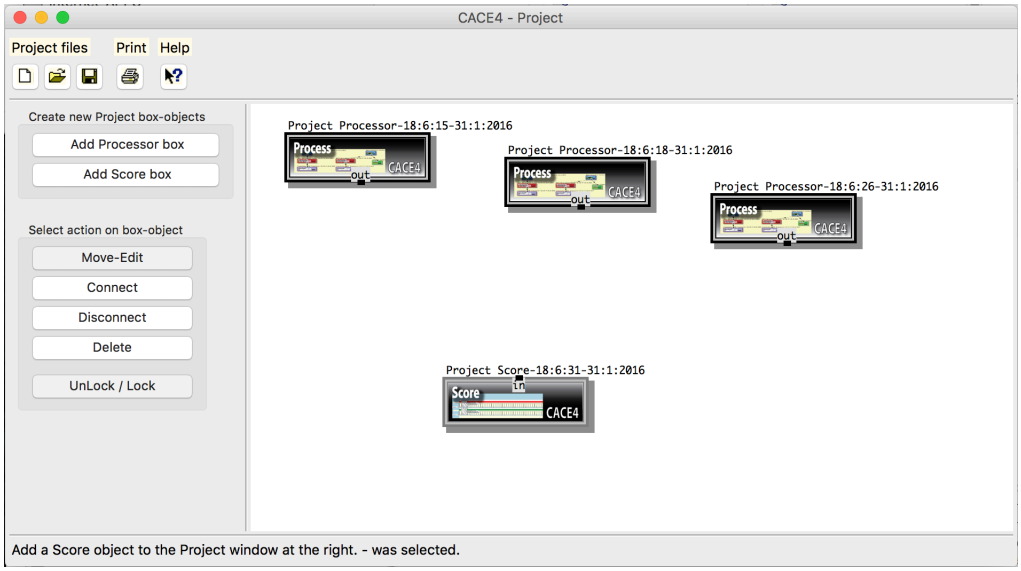


Figure 7 An example of a CACE4 Project window with three Processor objects and one Score object.

The differences in functionality, but also their functional dependencies, are reflected in their colour coding. Black is used as the colour of choice for the outside of the Project object and its background. The Score object uses grey as the colour of choice (see Figure 8).

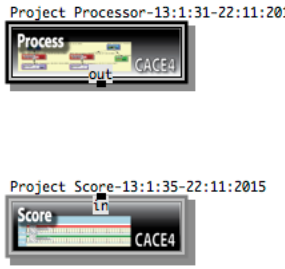


Figure 8 Colour coding of the CACE4 Project objects.
The two main groups of Project objects: the Processor object in black and the Score object in grey.

The purpose and functionality of the CACE4 Processor is restricted to a GUI one, in order to act as a container (or box) for different strategies. It gives the user of CACE4 the right metaphor: an object for accessing by double clicking to start using the several data analysing and transforming objects[84] available in CACE4. By 'chaining' them together they are connected on a GUI level to form

[84] See Table 5 for a detailed overview of all CACE4 objects available. All CACE4-Generators, CACE4-Manipulators, CACE4-Translators and CACE4-Informer objects can be used.

sequential visual process descriptions for ordering and re-ordering of different ideas. These chained lines are a metaphor of how the data flow is organized and are displayed as lines with arrowheads (to indicate the direction of the data flow) in the GUI.

4.3.5 The CACE4 Processor object and display.

The Processor object acts as a design window, or workspace, for laying out a strategy by the user. This is directly reflected in the way it behaves and looks (GUI) (see Figure 9, page 37): boxes can be moved around, they can also be linked, new ones can be added and unnecessary ones can be deleted. This all creates a more intuitive way of working and considerably speeds up the process of designing a strategy. The layout and the way the objects are linked together by means of using an arrow object, are also designed into the process to keep track of how the strategy is put together.

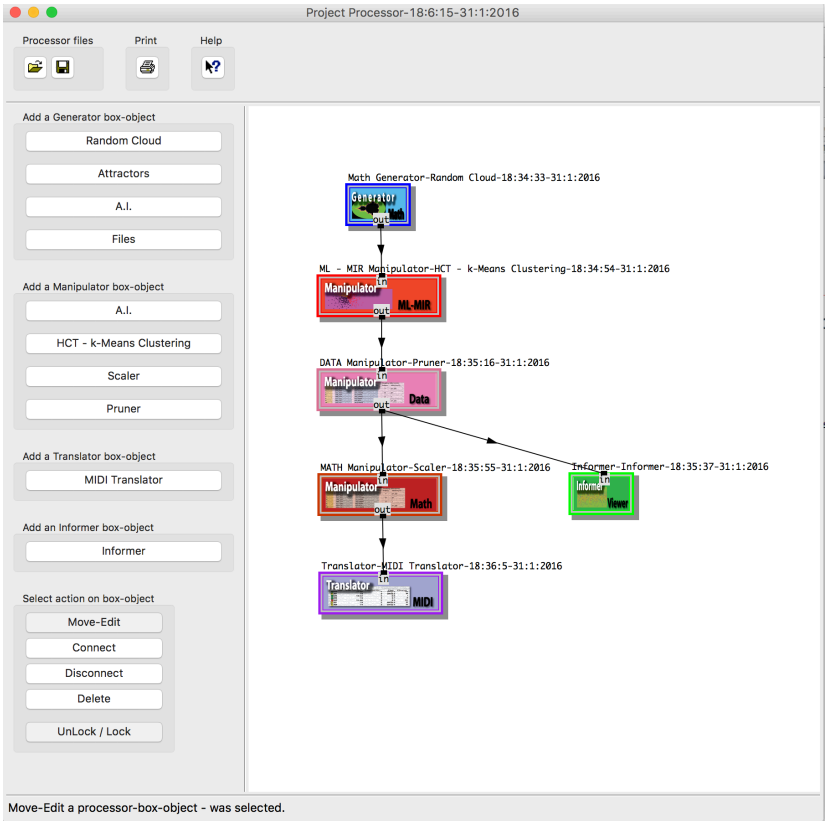


Figure 9 An example of a CACE4 Processor window display with a simple strategy for using *k*-means.

The differences in functionality of each of the four groups, but also their functional dependencies, are, just like the previously discussed Project objects, reflected in their colour coding (see Figure 10, page 38). The colour blue is used for the Generator objects. All different shades of red are used for the Manipulator object group. The Translator object uses purple and the Informer is displayed in the colour green.

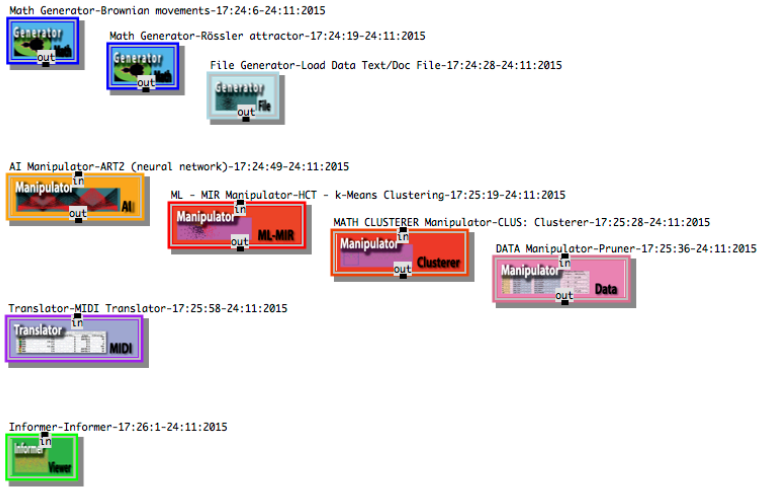


Figure 10 A selection of the coloured CACE4 Processor objects.

It is a direct implementation of the functional design of CACE4 into these four main groups. The choice of colour is bound only to these four groups[85] and thus is also reflected in the GUI-design of the CACE4 application (see Figure 10).

4.3.6 The CACE4 Object System & Modelling Organizing Shell, COS&MOS2: connecting everything together.

As described previously in the analysis of the CACE4 program, due to extendibility and modularity, there was a necessity for first creating a specific object oriented system for handling the control and flow of the data and the organisation of the CACE4 object system. Its design therefore, would guarantee a specific type of communication[86] between the connected objects and a correct flow of the data in CACE4. After creating the first version of COS (= CACE4 Object System) and later extending it with MOS (= Modelling and Organising Shell), rewriting parts of the system was necessary in order to make it as simple, clear and flexible as possible. In addition, the 'lack' of an underlying language and therefore no context availability to give it a structure to keep the data flow 'together', caused COS&MOS2 to be designed the way it is. Based on the principle of linked objects, superficially comparable to the well-known technique of linked lists, the objects are only 'connected' by their CLOS object-slots that contain a specific OBJ-ID to which it is connected. This specific,

[85] The initial choice of colour for each group: red was obvious for something to process the numbers and thereby drastically alters the output or generate, according the process, totally new output. Blue is used for all generators (three different type of files and several fractals and attractors). Purple for Translation of the stream. And finally green for an informer object, which only reflects certain statistical qualities of the stream and gives the user a 2D view of the stream. Also as it is the only exception for all CACE4 objects: it doesn't interfere with the stream, as defined.

[86] A more elaborated system of connections for exchanging data is not necessary in this version of CACE4. However, future versions, with some kind of knowledge about context, will make use of a system for exchanging other forms of data (information).

absolutely unique number is generated with the special LISP function: *gensym()*, which stands for generate symbol, on creation of every instance of an object. This, guaranteed by the underlying LISP system, unique OBJ-ID number, is stored in a slot of the instance with the same name and is used as an identifier when connections are made or data is retrieved (See Figure 11, page 39). It shows a detailed overview of this system of object linking. By avoiding the copying of whole instants in slots and the burden of keeping track of updating the other slots of the connected instance[87], a chain of linked instances (or objects) with a random order can be created. This gives one the possibility to experiment and to alter the sequential order to get the desired results.

Figure 11 CACE4 COS&MOS2 the chain of communication by OBJ-ID and data transfer between the CACE4 GUI objects.

Figure 11, page 39, shows an overview of all possible object slots, involved in the COS&MOS2 Linking system. A software feature build in every CACE4 GUI object Class.
Upon linking two objects together, this OBJ-ID is stored in the LINK-INPUT2 slot or LINK-OUTPUT2 slot. Which slot is used depends on whether the object is linked to, when its LINK-OUTPUT2 slot is used, or whether it is linked from, when its LINK-INPUT2 slot is used to store the

[87] This is now only done by the instance itself, thus avoiding extra coding and keeping the functionality of COS&MOS2 as clear as possible.

OBJ-ID. By using this approach a bi-directional link has been created between the two connected objects.

For all of the objects, all flexibility of changing values in other slots of the instance is only done by the instance itself, after been created by the LISP function *make-instance()*, and stored in the *CACE4POOL*. The global variable *CACE4POOL*, acts as a 'pool' of type *list()* where every instance, after creation, is stored. In creating a chain of objects, several objects will be created and stored, ready to retrieve from the *CACE4POOL* to save or load data from slots necessary for performing a specific task.

Looking at the CACE4 program, it can be seen that a connection is drawn when two objects are connected together. This is only a GUI and has no further purpose in the process. With a simple set of slot-readers and writers, it is now possible to keep track of all the connections made. By opening a CACE4 object, the slot-reader actively reads the LINK-INPUT2 slot and gets the specific output data of the object to which it is connected (slot: OBJ-OUTPUT). In this way all output obtained by calculation or processing of the data is stored only in the OBJ-OUTPUT slot of the object itself, thus avoiding extra overload in storing copies of this data (as LISP lists in RAM[88]) to slots of other connected objects. Strictly speaking, these extra copies are unnecessary and should be avoided in order to keep the RAM occupation by the program to a minimum[89]. The OBJ-output slot is filled by the object itself and will only be read by the other, connected object, if it needs the data for rerunning a specific task or calculation.

Except for each other's OBJ-ID therefore, there is no other type of information storage necessary to maintain the chain of connections and the associated order of the objects. Also by using a copy of the list of data on which to perform calculations, there will be no interference with the original output of the attached (input) object. COS (= CACE4 Object System), stands for the process of connecting CACE4 objects and lets them have access to all object slots necessary for processing their input (data).

MOS (= Modelling and Organising Shell) is reflecting the Class dependencies of the OOP Model as it is designed and used to create CACE4. The organisation of all these classes is reflected in the UML Class Diagrams of CACE4 (see Appendix 2.1 to 2.6). MOS acts as a shell, with the use of template files for wrapping (new) CACE4 objects in predefined classes for the GUI.

[88] LISP lists occupy more memory space then arrays due to the fact that they store more then only the number itself. Mostly extra information as type description is stored as well.
[89] Although for certain types of calculation the lisp function *copy-list()* is used before destructive lisp functions as *delete()* and *sort()* are called. But this is all done on a local binding level in a *let()* or *let*()* block. After it has been used, the LISP Garbage Collector (GC) does have access to those blocks and thus it can clean up no longer needed RAM space.

4.3.7 UML 2.5 diagrams.

The use of Unified Modelling Language (UML)[90] diagrams[91] provides the tools for displaying the class dependencies, as created in CACE4, in more detail. Normally used for imperative computer languages, UML can also be used for analysing LISP programs as CACE4. It offers several types of analyses, but in case of CACE4, Class Diagrams are used. These show Class dependencies by displaying a Generalization: the line with the white arrowhead and the relations between classes by showing an Association: a line without an arrowhead. Another possibility is to display all packages used, in a Packet Diagram as shown in Figure 12.

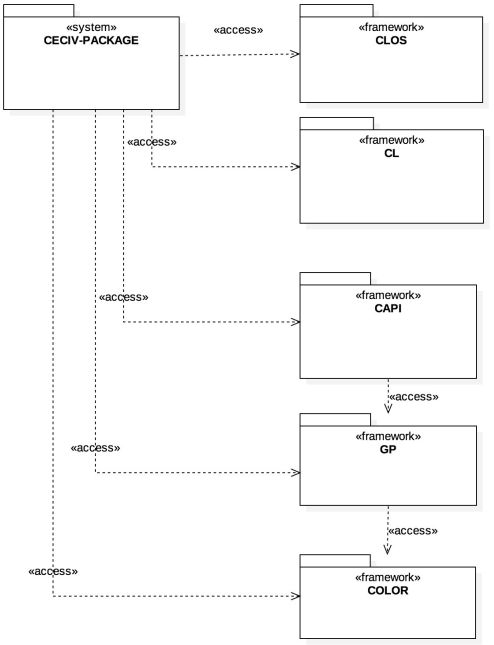


Figure 12 The CACE4 UML Packet Diagram. This shows the packets and their mutual relationship. Access is granted by package design.

CACE4 makes use of several libraries provided by LispWorks (part of the IDE). The GUI is embedded in these Libraries in order to be able to create an application with a GUI. Common Application Programmer's Interface (CAPI) is mostly used together with GRAPHICS-PORTS (GP) and COLOR (all extended libraries). Combined they provide all classes and functions necessary for building GUI's. The two other packages: Common LISP (CL) and Common LISP Object System (CLOS) are the language packages; they provide all necessary functions, macros, generics, methods and classes for creating an application.

[90] UML is widely used by IT professionals for creating diagrams for different situations and to be able to build Application Frameworks (mostly consisting of GUI) and to be able to generate programming code (mostly in C++, python, java) It is outside the scope of this thesis to explain UML in depth. See for further information: http://www.uml.org (this is the site of the official UML organization).
[91] All UML diagrams shown have been made with: StarUML 2.7.0

Appendix 2.1 shows a UML Class Diagram of a few of the CACE4 Generator objects[92]. They show the relation between CACE4 Generator objects and CAPI objects and functions for creating objects with a GUI[93]. The numbers left and right from an Association line (the line without an arrowhead), shows the multiplicity of the connection (mostly there is just a single connection). The generalization lines show the class dependencies. They all point to a macro: (*CAPI::define-interface()*[94]) and provides CAC4 with an instance of a GUI superclass. All further functions etc., necessary for creating a GUI, are inherited.

Every Class is represented by a rectangle with three compartments. The upper compartment is for displaying Packages and Class names and stereotypes[95]. The middle one holds all the arguments (or in this LISP case, all the slots) of a CACE4 Object. They all have a minus sign for their name, standing for 'private scope'. The lowest compartment is for displaying all important methods and functions used by the CACE4 Object, together with all of their arguments.

The next two, Appendix 2.2 & 2.3, are two halves of one larger UML Class Diagram. Together they show the Class Diagram of All CACE4 Manipulator objects. Most CACE4 Manipulator GUI objects point to their superclass GUI: *CAPI::define-interface()* for inheriting all necessary functions, classes and methods.

Just like the CACE4 Generator objects, the CACE4 Manipulator objects have a simple Class relationship. A CACE4 GUI Object, with all additional GUI Classes and functions, acts as a superclass template (strictly speaking, it is a subclass) for all calculating (or processing) objects, such as fractal, automata and file input objects etc. All CACE4 objects are built with this OOP concept, providing every Model with its necessary View(s) and Controllers.

Appendix 2.4, shows an UML Class Diagram of the two remaining CACE4 objects: A CACE4 Informer object and a CACE4 (MIDI-) Translator object. They show the same UML dependencies and relations as there are in the previous UML diagrams.

In the best tradition of UML, although the diagrams are detailed, they are not complete. They do however provide another, more convenient way of describing relations between software modules, as there are packages and their classes.

---

[92] An extended view of all available Objects can be found at Appendix 2.5 and 2.6.
[93] Although CACE4 objects (for viewing and controlling) are embedded in the CAPI classes, the Model (from the MVC paradigm) is separated by a CACE4 GUI class (CACE-Generator-GUI object and CACE-Manipulator-GUI object). It only makes use of the CLOS classes, so the Model code (and the software engine) is still transportable to other IDE's and other OS platforms.
[94] CAPI:: indicates the package where the macro or other functions, classes and methods are first defined.
[95] Stereotypes is UML terminology used for specifying a certain type to an item. This can be anything: from strings and integers to methods and classes.

Chapter 5

CACE4: taking a closer look: GUI and software functionality

This chapter takes a closer look at the design of the GUI as well as the underlying functionality of CACE4 as a Computer Aided Composition Environment application. The major group of CACE4 Manipulators is split into four smaller sub-groups, representing certain types of processes that cover specific topics from the domain of A.I., Machine Learning and Music Information Retrieval, Mathematical manipulation and Data manipulation. Due to the fact that CACE4 is still a work in progress, with newer versions being created on a weekly basis, it is difficult to give it a final version number. Therefore for this thesis all output has been created with this version: CACE4 v00d.56.19.479 // created on 10-12-2015[96].

5.1 The CACE4 Generator objects.

Although the CACE4 Manipulators objects are the main focus of this thesis, without numerical input from one of the two different CACE4 Generators, analysis and manipulation of data by the CACE4 Manipulator objects could not be done. Therefore a brief explanation is necessary as their major function is to take care of the Input Output (IO) of the CACE4 program. This is done by either generating from mathematical equations (fractals and attractors) or by reading a file and provides the CACE4 Manipulator objects of proper data for analysis.

The two main groups of Generators:
1 CACE4 MATH Generator objects.
2 CACE4 FILE Generator objects.

The CACE4 MATH Generator object group consists of different Fractals, Attractors and different types of random number generators. From the large group of fractals a Bifurcation diagram, an Iterated Function System (IFS) a Julia set fractal, a Gumowski-Mira fractal, two different types of Mandelbrot calculations and an Automaton calculation have all been implemented as separate CACE4 MATH Generator objects. There is a group of random number generators: Brownian movements, Linear Congruential method, a Chaos on Torus calculation, a Random Cloud generator and Tendency masks have all been implemented as separate CACE4 MATH Generator objects. In addition, a few different Attractors have been implemented in CACE4. A Henon type 1 and 2, a Lorenz and a Rössler Attractor are all implemented as CACE4 MATH Generator objects.

[96] By now: 11th of May 2016 there is a newer version available: v00d.57.08.490.

The CACE4 FILE Generator object is available in three different types. First as a SMF format 1 reader, second as a SPEAR partials text file reader, and third as a plain text (.txt) file reader. When using this last CACE4 FILE Generator object, the numerical data in a plain .txt file needs to be in a (x,y) paired orientations as the data read in from the file will be used for direct plotting (2 dimensional x/y axis) as well. The choice of representing data in a one or a two- dimensional list really depends on how one wants to view the data. If a one-dimensional view is selected, an extra x will be generated and used for plotting (on the x-axis). Whatever is chosen, the stream itself will always consist of the unaltered original data as a one-dimensional list.

5.2 The CACE4 Manipulator objects.

As is previously explained in section 4.2, the objects from the Manipulator group are the main core of analysing and manipulating of the data streams. The split into four main groups is reflected in the software design by the selectable menu items on the left side of the CACE4 Project-window (see Figure 9, page 37).

The main four groups of CACE4 Manipulators are:

1 AI Manipulator.

2 Machine Learning / Music Information Retrieval Manipulator (ML-MIR Manipulator).

3 Mathematical Manipulator (MATH Manipulator).

4 DATA Manipulator.

Although completely different in processing the input data, these four groups have one thing in common: they all share the ability of changing the data with their limited techniques according to their algorithm design. Sharing certain characteristics is reflected in their naming: the MATH Manipulator reflects the use of mathematics as the core process, by using correlation, scaling and statistical properties. The DATA Manipulator uses techniques from the domain of Informatics and mostly involves a more elaborate use of techniques for processing data (e.g. merging, sorting, deleting or splitting). The two remaining groups, AI Manipulator and ML-MIR Manipulator, have been created as two separate CACE4 objects although they share many characteristics[97]. The reason for this is that the ART2 neural network belongs to the domain of AI. Originally designed as a model for how neural based networks could be programmed on a computer, ART2 is vector based. Other (M)IR techniques use (x,y) number pairs and are more related to the domain of Statistics and data mining. The

[97] This is also reflected in the class dependencies: All four subclasses share certain characteristics by their superclass(): CACE4 Manipulator.

Hierarchical Cluster detection techniques, *k*-means and Expectation-Maximization (EM), are grouped into ML-MIR Manipulator objects.

5.3 The CACE4 AI Manipulator object group.

This is a separate group of objects based on techniques from the domain of Artificial Intelligence (AI). For the moment ART2 is the only working A.I. object[98] in CACE4.

5.3.1 Adaptive Resonance Theory Neural Network (ART2).

The core engine of the A.I. Manipulator ART2 Neural Network is the Adaptive Resonance Theory Neural Network: ART2 as designed and proposed by Carpenter & Grossberg (Carpenter and Grossberg 1987). Originally designed for recognition of optical patterns in pictures, ART2 is vector based. The (input) vector length can be set to anything between 2 and 20 entries. (To be most effective the vector size should be between 5 and 10 entries, also called nodes.) ART2 NN is constructed as a 3 layer neural network, and obtains its input from the CACE4 object it is attached to as a list of vectors. This input vector will first be normalized in the first layer and, after weighting, transferred to the hidden layer and finally to the output layer, where the weighting of the nodes to group the vectors into different categories takes place.

The Adaptive Resonance Theory NN makes it possible to work with a neural network that has the property of being adaptive to 'strange' input. If there is a new vector to qualify and it does not fit with the previous vectors in the existing categories, ART2 will create a new category for this vector. From now on, this newly created output category and ART2's vector based grouping of data, results in different categories with a certain overlap at their adjacent boundaries. As a final result, these grouped vectors share the same characteristics in a looser context than grouping by strict values (for example < or >) would allow.

The ART2 Neural Network works without supervision, as it doesn't have to be trained by using specially created vectors as an example or template vector in order for it to become effective. ART2 works according Adaptive Resonance principles: a self-regulating feedback mechanism. Amongst several parameters that can be used are the Learning Cycle Counter (see Figure 13, page 46: <number of learning cycles>) and the vigilance parameter of the NN, which represents the overall alertness of the NN. The LCC is used in the process as a variable counter for the number of times the ART2 NN will evaluate the vector. This is necessary in order to classify (categorize) the vector. It is a

[98] In future versions other techniques as Frames (Winston 1984), Inference Rulers (Forward and backward chainer) and A-Search (Heuristic search) will be added to the group.

fundamental property of the ART2 Neural Network, as opposed to other types of neural networks[99], which need some training sessions in order to function properly.

Alpha and Theta are two (adaptable) constants of the ART2 NN model. Alpha is used to scale the input vector and therefore has the initial value of 1.0 (and should be used with caution). Theta is calculated according to: $theta = \frac{1}{\sqrt{nInputs}}$, where *nInputs* stands for the size (number of nodes) of the input vector, and should therefore be adapted if the input vector size changes (for example, increasing if the number of nodes are decreasing and vice versa). The reset threshold can be used to change the NN's overall sensibility for resetting itself in case the ART2 NN is categorizing the input vector in a wrong category.

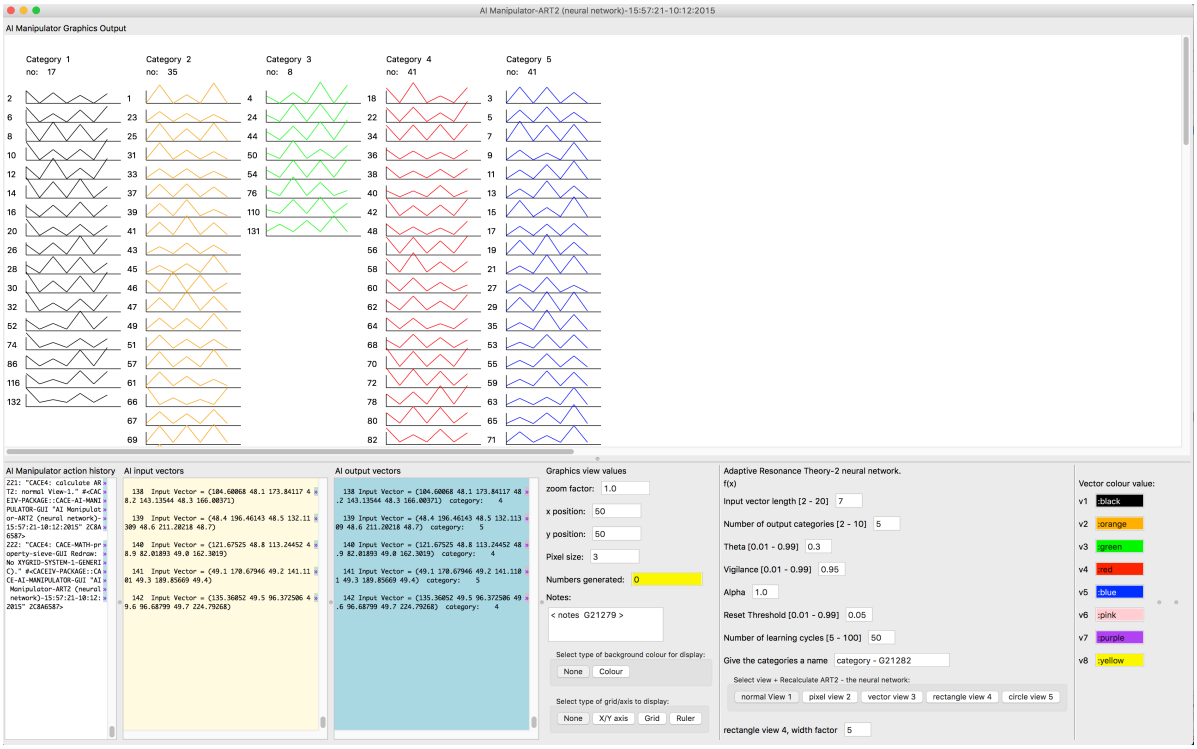


Figure 13 The A.I. Manipulator ART2 (Neural Network) GUI.
This is view-1 the 'normal' view.

By fine-tuning certain input parameters of the ART2 Neural Network such as vigilance, which is used as a limit for the magnitude of the allowed mismatch inside one category and the amount of Learning Cycles (LC), we are able to influence the overall performance of the Neural Network. The amount of Learning Cycles should be in the range of at least 15 to 20 to be effective (50 is recommended, but more time consuming). What is 'uncontrolled' in the ART2 algorithm is the use of a random in the

[99] For example the Perceptron, the first model of an artificial neural network and ART2's predecessor, needed supervised learning (Wasserman 1989). Other models as Hopfields nets and Bidirectional Associative Memory (BAM) networks need training sessions as well (Wasserman 1989, p. 122).

initialization phase of the network calculation. It puts a tiny value[100], mostly in the range of: 0.001 – 0.1, in the array where the values of the nodes from the network will be stored. This small random number will guarantee that a non-local minimum will be avoided if the algorithm gets 'stuck' in one of these local minima. This has a side effect to the process itself: a specific category can change from number depending on the initial random value. The initially processed vectors (input) however, are still grouped together in the same categories.[101] ART2 uses these newly categorised vectors in order to create a new output stream (added per category, in a block structure, one after the other).

$$p_i = u_i + \sum_j g(y_j) * wDOWN_{ji}$$

$$q_i = \frac{p_i}{L2NORM(p)}$$

$$u_i = \frac{v_i}{L2NORM(v)}$$

$$v_i = f(x_i) + b * f(q_i)$$

$$w_i = I_i + a * u_i$$

$$x_i = \frac{w_i}{L2NORM(w)}$$

$$f(x) = 0, \text{ if } 0 \le x < theta \text{ or } f(x) = x, \text{ if } x \ge theta$$

$$y_i = \sum_j p_i * wUP_{ij}$$

$$wDOWN_{ji} = \sum_i \sum_j \text{learning rate } * d * p_i * wDOWN_{ji}$$

$$wUP_{ij} = \sum_i \sum_j \text{learning rate } * d * p_i * wUP_{ij}$$

$$L2NORM(v_i) = \sqrt{\sum_i v_i} + 0.001$$

Equation 2 The Carpenter and Grossberg ART2 Model equations (Watson 1991, p. 83).

If ART2 is resetting itself as a neural network, it does so in order to be able to categorize the vector correctly according this principle: as stated in Rojas, Neural networks: "The purpose of the reset signal is to inhibit all units that do not resonate with the input. A unit in layer F2, which is still unused, can be selected for the new cluster containing x. In this way, a single presentation of an example sufficiently different from previous data, can lead to a new cluster." (Rojas 1996, p. 417). This last principle of the design of the ART2 algorithm will enforce a categorisation of the vector. Equation 2 (page 47) demonstrates the several stages of the algorithm. The three layers of the network are

[100] In Equation 2, page 26 this is shown for the normalization phase of the vector: L2NORM(v) + 0.001

[101] This phenomenon can be observed in the examples of different type of views in fig. 12, 13, 14 and 15. The colouring of the detected categories changed (the number of the category changed), but not the categorized vectors. They are all correctly detected and sorted into the same previous groups. In future versions, control of the fixation of the categories (random number control) will be done by the user.

represented by: the vectors: x_i, p_i, q_i, u_i, v_i, w_i and x_i. The output layer of neurons is stored in y_i. And I_i is the vector to be evaluated. The variables: a b d, f, theta, alpha and vigilance all belong to the model constants according Watson[102] (Watson 1991). As a consequence of its architecture and model[103], ART2 uses a so called hidden layer: when a vector (or neuron) is neither part of the Input (I_i) nor Output (y_i) layer it is called hidden (Nierhaus 2009).

5.3.2 ART2 GUI design topics.

As previously stated, looking at both the input and output data in different ways can help in understanding why the data has been sorted into the different categories. Not only by using the 'normal' view (see Figure 13, page 46), but also by extending it with four other, different graphical representations of the same data, this idea has as such been implemented in CACE4. These five separated output categories, as detected by the ART2 neural network, can be viewed with specific techniques of plotting applied. Each category is associated with a different colour. Not all entries are located at clearly separated positions and therefore the use of colour coding is necessary to make an easier distinction between the detected categories. As an example, a Brown (*random(x)* – white-noise, n=500) fractal calculation was used as input. Vector $v_{i=1}$ has a length of 7 nodes and consists of a sequential copy of 7 items (rational numbers in the range: 0.0 - 300.0) from the output of the Brown fractal. For the next vector $v_{i=2}$, copies of the next 7 numbers are used until all Brown fractal output

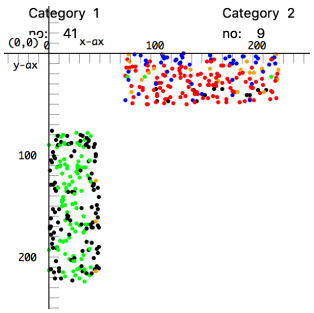


Figure 14 View 2: the pixel view.

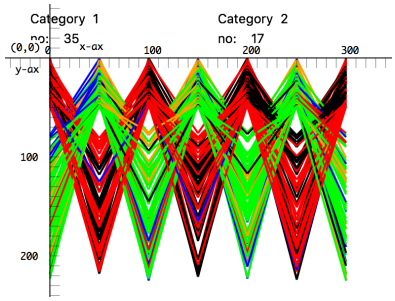


Figure 15 View-3: the vector view.

is executed: $v_{i=500}$. The first view represents all vectors to be plotted in columns, each column representing a different category[104] (see Figure 13, page 46).

The second view shows the same vectors, but now as single nodes plotted as (x,y) number pairs on top

[102] Mostly hidden for the user except for alpha, theta and vigilance see Figure 13, page 26 for other input parameters for the user.

[103] Floreano and Mattiussi show that the resonance in the name of ART2 relates to the fore- and backward going, for weighting of the vector. Until either the vector is categorized (either in a new or already existing one), or the network reset itself to new initial values. (Floreano and Mattiussi 2008)

[104] This view, as of December 2015: CACE4 version 0.54.19, will be replaced by a new view altogether in one of the upcoming updates (autumn 2016).

of each other[105]. They can be now only distinguished by the unique colour associated with the specific category (See Figure 14, page 48 view-2: The pixel view). The third view involves the same plotting techniques as the view-1 this time used for plotting the nodes connected by a line. (See Figure 15, page 48, view-3: vector view). By plotting the vectors this way as super-positioned on top of each other, the similarity (of the categories) of the vectors can be seen directly.

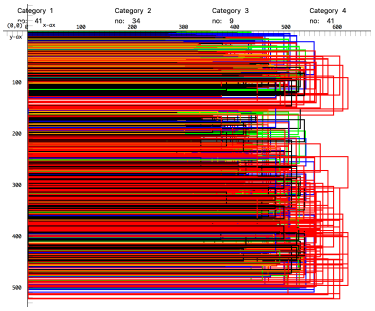


Figure 16 View-4: the rectangular view.

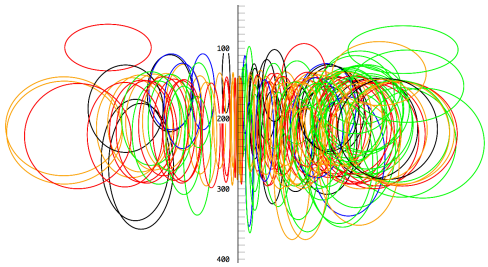


Figure 17 View-5: the circle view.

A possible fourth view is to plot the vector elements as (scalable) rectangles (see Figure 16, page 49 view-4, the rectangular view). All vectors are also plotted on top of each other and can be scaled for plotting only; this does not alter the original input. A fifth and final technique for plotting the vector entries involves plotting the nodes of the vector as circles (see Figure 17, page 49 view-5, the circle view). For scaling (size of the ovals) purposes only, correlation calculations have been applied as well[106].

5.4 The CACE DATA Manipulator object group.

The DATA Manipulator group shares several characteristics: a simple dialog layout, together with an input and output numerical display column. They are based on well-defined algorithms from the domain of informatics and are available for use in many computer languages. They are therefore not strictly based on mathematical formulas (like the STAT/STAPS group) but depending more on the implementation of the functions (in the case of CACE4: LISP). Sorting – *sort(), stable-sort()* - and replace – *replace()* - algorithms implemented in the LipsWorks programming environment and, as such, defined by the Common LISP language standard, are a few examples.

[105] While view-2 is a plot of (x,y) number pairs and the number of vector nodes is uneven (in this example 7 nodes) are used, there is a swap between x and y values. This explains why two blocks are observed, instead of one. To avoid this way of plotting the result, it is possible to eliminate the x-value with a pruner object or by changing the number of input nodes to an even number.

[106] By executing the ART2 calculation first, then calculating the spearman correlation of the pattern and scale the result with linear scaling and then multiply the plotting coordinates with this result, it is possible to spread the vector plot. This is a more experimental algorithm design and will probably change in the near future.

One other criterion for grouping together the four MATH DATA manipulators is their ability to split the input into separated streams by means of using the index check-box <start-index>. By making use of a jump size, one single stream can be treated as multiple – independent – streams and therefore more complex processing (e.g. deleting) is possible.

5.4.1 Pruner.

The Pruner object (see Figure 18, page 50) acts as an all-round pruning tool for deleting items from the input stream. The algorithm of the Pruner object makes use of the replace[107] function of Common LISP. It can be used in either of two ways: in the simple mode to delete a specific chunk of the input stream (a-start to a-end), or, in a more complicated manner and together with the possibility of creating ones own jump size, as a delete function, which can jump thru the input and delete specific members of the input. The thus created new output will be accessible by the next object in the strategy chain, attached to its output.

Figure 18 The CACE4 Pruner.

[107] LISP function (destructive): (*replace sequence-1 sequence-2 &key start1 end1 start2 end2*) => sequence-1 (Steele 1990, p. 408)

5.4.2 Merger.

The CACE4 merger object is a special one in this group of DATA manipulators. It can handle up to 12 objects as input and merges them in five different ways. The names given to these five choices are the same as those that have been used on the buttons of the GUI (see Figure 19, page 51).

1 - <Adding>: By placing al members of the separate input streams as a single block in a consecutive order, one after the other: [all input 1, all input 2, … all input 12].

$X_{new} = [A_i, B_i, ..., L_i]$, **where** X_{new} **contains all newly added members** ***i. The ordering of the blocks is shown between closed brackets.*** **And** L_i **is the maximum numbers of blocks used (*i* = 12).**

Equation 3 Adding algorithm.

2 - <Reverse>: By doing the same process as adding but now in reverse sequential order [all input (max) 12, … all input 1].

$X_{new} = [L_i, K_i, ..., A_i]$, **where** L_i **and** X_{new} **are the same as in equation 3.**

Equation 4 Reverse algorithm.

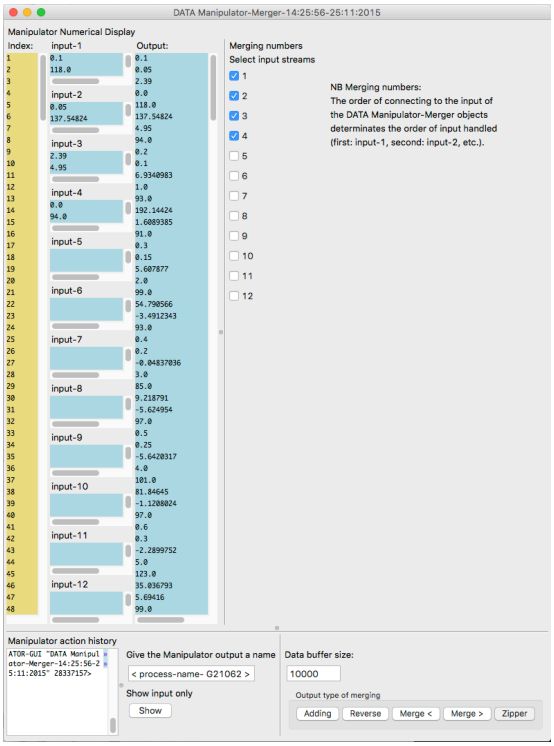


Figure 19 The Merger GUI.

3 + 4 - <Merge < > and <Merge > >: This is done by first merging all entries in one single list and then sorting them in ascending (<) order [i-minimum, … i-maximum], or sorting them in descending (>) order [i-maximum, … i-minimum]. First the Adding algorithm from Equation 3 is used and then it is sorted according Equation 5.

$X_{new} = sorted(X_i)$, **where *sorted* can be: < or >. And** X_{new} **contains all newly sorted members *i*.**

Equation 5 Sorting algorithm. See section 5.3.3 for further details for the used Common LISP sort functions.

5 - <Zipper> : By merging all entries in a 'zipping' (or interleaved) way of ordering. Take one entry from every input stream place them consecutive, one after the other in the list. Then proceed to the next entry, until all members of all input lists have been placed.

$[A_{i1}, B_{i1}, L_{i1}, A_{i2}, B_{i2}, L_{i3}, ..., A_{in}, B_{in}, L_{in}]$, **where *L* stands for the maximum number of input blocks used.**

Equation 6 Zipper algorithm.

This can be achieved even when the length of the entries (A, B and L) are not equal. It is important to note that all merger actions can be performed with numerical input lists of unequal length. Each calculation will be done until all input lists have been exhausted.

5.4.3 Sorter.

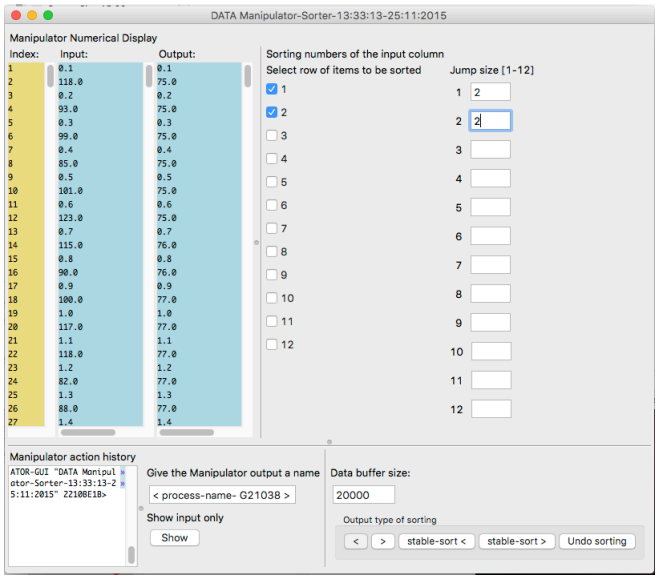


Figure 20 The Sorter GUI.

The Sorter is a CACE4 object for sorting. Two basic types of sorting are available: sort and stable-sort. Sort makes use of the Common LISP function *sort()* (see Figure 20, page 52). Stable-sort makes use of the Common LISP *stable-sort()* function[108]. Both types of sorting can be used with the operand: < and >. Together with the use of the jump size it is possible to create a unique selection of the input list for processing by the sort algorithm. This results, as seen in the example shown in Figure 20, in two sorted (x,y paired) outputs combined in one output stream. It is sharing its sorting algorithms with the previous described CACE4 Merger object (see section 5.4.2, page 51).

5.4.4 Splitter.

The Splitter is the last in the series of four DATA Manipulator objects. It takes one input stream and splits it into several others according to the user's specification. Its purpose is to generate one single output stream, but sorted in blocks (also called chunks), one after the other. Every chunk contains a specific part of the input stream according to the index number.

Figure 21 The Splitter GUI.

[108] The two LISP functions are both destructive on the original sequence, therefore a copy of the input is used (*sort sequence predicate &key :key*) => sequence, (*stable-sort sequence &key :key*) => sequence (Steele 1990, p. 408).

Together with the associated jump size, one stream will be created containing up to twelve chunks. (output-1, … output-n). The newly generated chunks will also be shown in the output views (with the maximum of twelve separate views). (see Figure 21, page 53).
To obtain a single block of the input stream it is either possible to generate just one block, or in case more than one is created, the pruner object is used to remove extra, unwanted blocks.

5.5 The CACE MATH Manipulator object group.

The MATH(ematical) Manipulator group is, in contrast to the previously discussed DATA manipulator group, based on a much more complicated mathematical description. It can either process the data according known mathematical processes (mostly statistics), or use processes that are based on other, more complex techniques. Information Retrieval offers tools for more elaborate techniques for a different kind of analysis. In addition, the programming techniques for making GUI's capable of changing the background colour quickly, are of great help in making a decision about the generated output (for more see section 5.5.3, page 58).

5.5.1 Clusterer (CLUS).

The 'Clusterer' can be regarded as an object with only artistic process output in mind. The 'Clusterer' acts as a 'cluster creating' algorithm. Up to ten clusters can be independently positioned in the window, with separated values for x-centre, y-centre, width, height, the gravitational constant and massa1 and massa2.

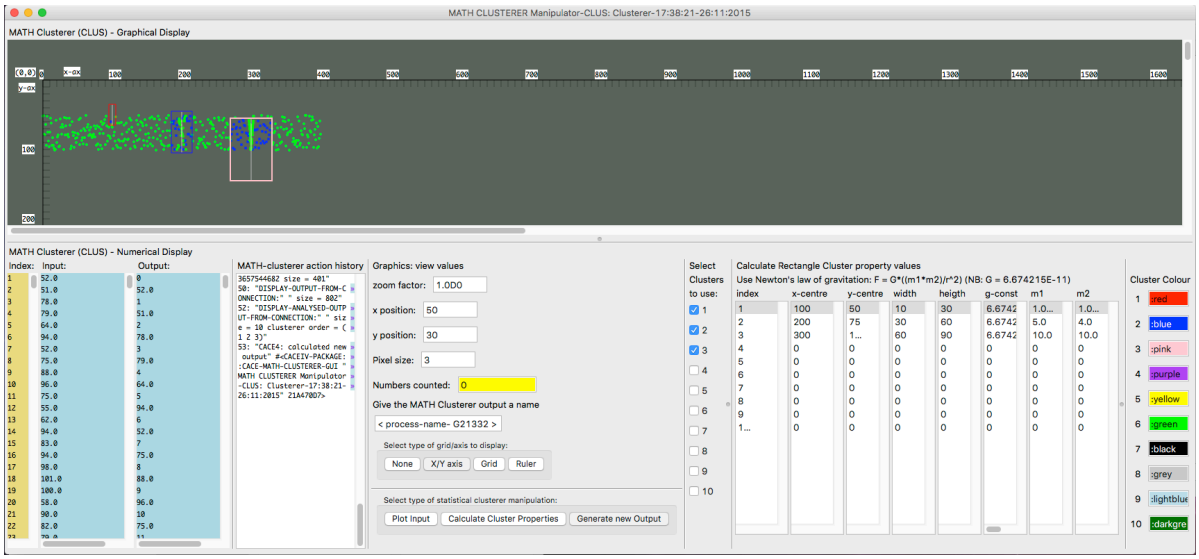


Figure 22 The Clusterer GUI.

All clusters use the basic equation of Newton's Universal Law of Gravitation (see Equation 7). All parameters of the equation can be used for alteration of the Cluster output. According to Newton's equation, there are two bodies with mass *m*1 and *m*2.

$$F = G\left(\frac{m1 * m2}{r^2}\right)$$ **, where m1, m2 are masses, r is the distance and G is the universal gravitational constant.**

(G = 6.674E-11 N m2/kg2).

Equation 7 Newtons Universal Law of Gravity.

Larger masses result in a larger gravitational force. Accordingly, the clustering effect will be larger. The size of the squares can be altered as well: using larger cluster areas will also result in larger clusters. Figure 22, page 54, shows us the GUI of the CACE4 CLUS Object: all coloured rectangular boxes are acting as independent areas of gravitation, with their mass centred onto the line in the middle of the box[109].

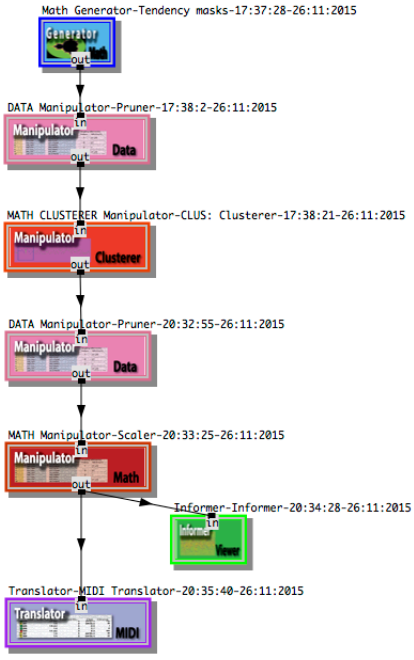


Figure 23 An example of a simple setup for working with the CLUS object in the CACE4 Processor window.

[109] A coloured box in the GUI is a graphical representation of m1. Inside its boundaries is an area where the cluster algorithm is active. It can be freely positioned in the x, y space. Changing the values for width and height can alter the size of the box. Inside this box a small line, from top to bottom, is used as a representation of the centre of mass (m1). According Newton's Universal Law of gravity, all mass is centered into a single point in space. By skipping the y value and only using the x value of the calculation, this centre of mass coincides with this drawn line inside the box. By representing the original input as the second mass (m2) as pixels, if having enough mass, they can be drawn to this centre-line. The values for m1 and m2 can be altered in the GUI.

When low mass is entered for both m1 and m2, the pull of gravity is low and according Newton's Law of Gravity the entries inside the box will 'fly' away and be dispersed over the graphical display. If a higher mass is used for both m1 and m2 however, the entries (shown as blue dots) will pull towards the centre of gravity (the vertical line in the centre of the box), and displayed at their new position as light-green dots.

Figure 23, page 55 shows a typical, simple setup to work with the CACE4 CLUS Object. It shows how to generate chords from a previously generated data stream (Tendency masks are used in this case as data source). With a CACE4 Translator object at the end of the strategy chain, the newly calculated output (showing up as light-green dots in the GUI, see Figure 22, page 54) can be sent to the CACE4 Score object (in the Project window)110

5.5.2 The Statistical Manipulator (STAM).

This is also one of the more complicated objects in the CACE4 program. It gives access to a whole range of statistical functions used for doing manipulations on the input list.

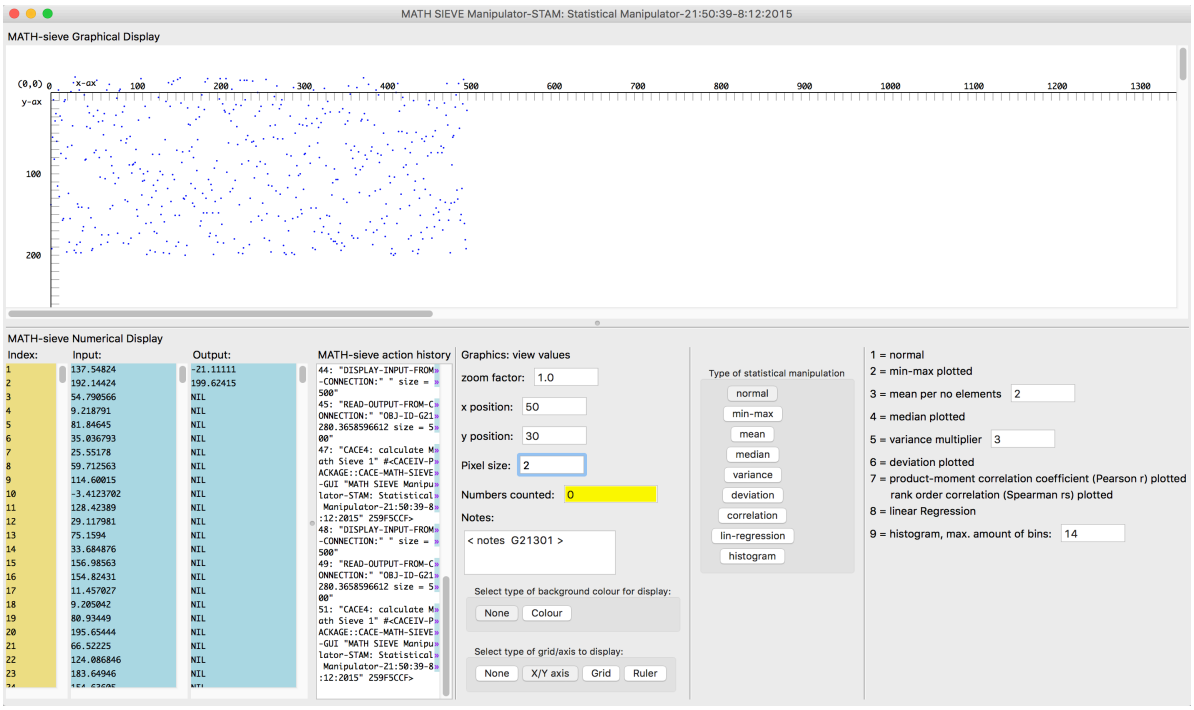


Figure 24 The STAM GUI.

[110] Due to the fact that the cluster algorithm adds single x values before any y-value, the data output stream will be doubled in size. Also when using the Translator object, Absolute timing instead of delta timing should be used.

Besides the basic tools as Minimum-Maximum, Mean, Median, Variance and Standard Deviation, it gives access to two correlation calculations, Pearson (product moment correlation coefficient) and Spearman (rank order correlation), as well as Linear Regression and Histogram analysis.
All statistical procedures create a specific output that can be used by the next object in the CACE4 strategy chain of connections. The output calculated by this object is as diverse as all the statistical procedures used: from only two <max-min> values, up to the maximum number of values, which equals the number of entries in the input <normal>.

The output created by using one of the following procedures (<button-names>):
<normal>: just copies the input, and displays it.
<min-max>: creates just two numbers: the minimum and the maximum value of the input list.
<mean>: The exact number depends on the amount of elements which should be grouped before taking the mean. (See Figure 24, page 56: text-edit field: "3-mean per no elements"). When using the maximum amount (= number of entries input) the output created is just a single number: the mean. In the case of Figure 27 (page 59), three are used. Therefore the maximum amount / 3 output elements is created, which is equal to a third of the size of the original input.
<median>: Here there is just one number: the calculated median is the output.
<variance>: The variance (per element) can be altered (multiplied) by: text-edit field variance multiplier <deviation>: Creates three numbers: The Standard Deviation, the Absolute Deviation and the Squared Deviation.
<correlation>: There are two different correlation calculations: Pearson and Spearman as the calculated output.
<lin-regression>: According to a linear regression algorithm,[111] only four numbers in strict order are calculated. The first number (*n*) is the number of entries in the input. Followed by *m*: the slope of the least-square best fit line, then *b*: the y-intercept of the least-square, best fit line. Then as a fourth number, *rs*: the squared correlation coefficient.
<histogram>: The output created is the number of items per bin of the Histogram analysis. The exact number is variable, but mostly depends on the number of bins (see Figure 24, page 56), and ranges somewhere between 5 and 14 bins. Due to the character of the algorithm used however, it looks for the optimum number of bins: the results are variable. See at the end of section 5.5.3 (page 63), for further explanation of the Histogram algorithm used.

[111] The functioning of the algorithm is explained in more detail in section 5.4.2.7.

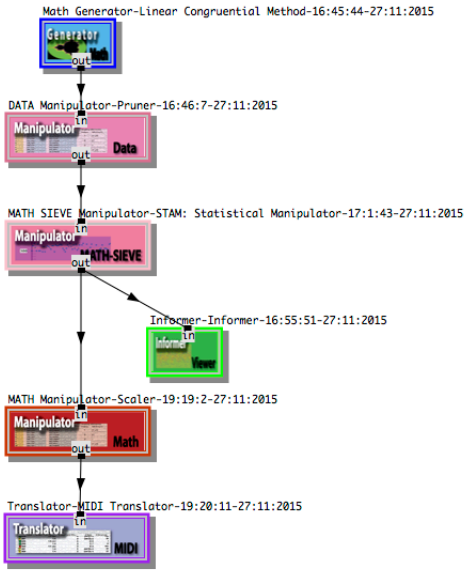


Figure 25 Example of a setup centered round a STAM object in the Processor window.

Figure 25 above, shows a simple strategy setup for working with a STAM object in the CACE4 Processor Window. In this example, a linear congruential method, a special algorithm for generating a pseudo random sequence[112], acts as a data generator. The Pruner is used to remove all x-values as generated by the MATH Generator object, thus guaranteeing a stream of maximum random numbers (the initially generated y-values). A MATH Manipulator Scaler object, in combination with a Translator MIDI object, takes care of the final steps in the strategy chain, by transforming the data into pre-MIDI data and 'sending' it to the CACE4 Score object (in the Project window).

5.5.3 The different STAM processes.

<min-max>.

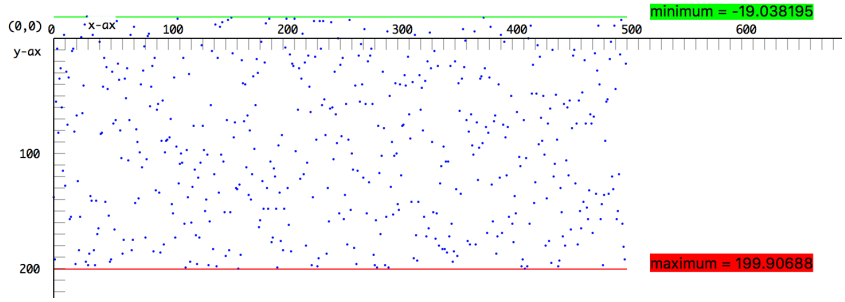


Figure 26 Minimum and maximum display in the STAM GUI.

[112] According: $X_{n+1} = (aX_n + c) \bmod m \quad n \geq 0$, see Moore (Moore 1990, p. 409) for more detail of the algorithm.

These two are basic quantitative qualifiers: minimum and maximum are really operating as a pair together. They cannot be omitted in the statistical approach of CACE4 as they are often used for finding ranges in the data that is being processed (see Fig. 30). Its effective range; $< -\infty, \infty >$, depends on the implementation of the LISP system and Operating System of the computer used. (-NAND and NAND are also used as an error flag for the OS, though mostly not in LISP, but more in imperative languages as C and C++.)

<mean>.

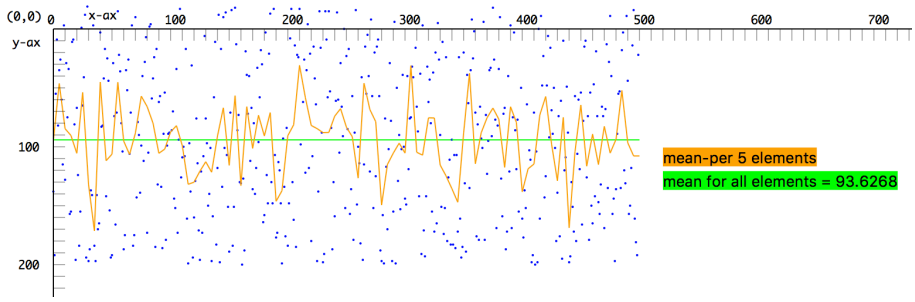


Figure 27 Display of mean.
In this case the orange coloured line plots the mean of 5 consecutive elements of the input list. The green line indicates the mean for all elements in the input list: $\bar{x} = 89.799...$

With the use of mean as a selected statistical calculation, there is the possibility for creating a kind of line bender (keep in mind that it can be used as a melody creator/bender as well). The data is first analyzed, which will produce a mean of the data group, with a variable mean per number of elements in the range of: $x \in [2, \ldots, n_{\max}]$ (see equation 9 for the formal description). This gives smoother numerical patterns with fewer jumps in the line. The minimum and maximum value (of the orange coloured line) will be more flattened if more numbers are used[113]. Equation 8 shows a more generalised description of mean used in other statistical processing in CACE4:

$$\bar{x} = \frac{x_1 + x_2 + \ldots + x_n}{n}$$

Equation 8 Mean formula.

$X_{output} = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)$ **, Where X output is a list of the all calculated mean values.**

Equation 9 A variable segmented Mean.

[113] For now, no interpolation takes place between adjacent values. Therefore the (orange) line as viewed in Figure 27 will be outputted as a compressed version. In a future version of CACE4 extra options will be available for interpolation purposes.

<median>.

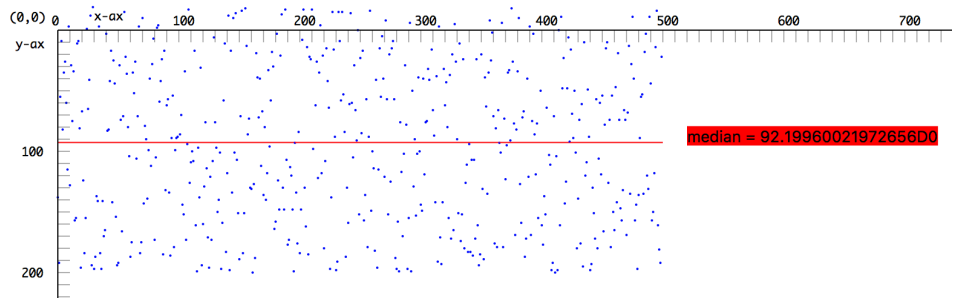


Figure 28 Display of the Median.

NB. In this case n=500 (= even number of members in the input list) therefore the Median = 92.199600....

The Median can best be described in an algorithm. First create a sorted list by using the operand: < on the input list. In case of there being an uneven number of members in the input list, take the middle member of the sorted row. When the amount of members in the input list is an even number, take the middle 2 numbers of the sorted list (<) and divide them by 2. The resultant number is the median.

When the total of the members is uneven Equation 10 is used:

$$M = \frac{(i_{n/2} + i_{n/2+1})}{2}, \text{(}i\text{ = sorted list: <)}$$

Equation 10 Median for *i* = uneven.

And when total of the members is even Equation 11 is used:

$$M = i_{n/2}, \text{(}i\text{ = sorted list: <)}$$

Equation 11 Median for *i* = even.

<variance>.

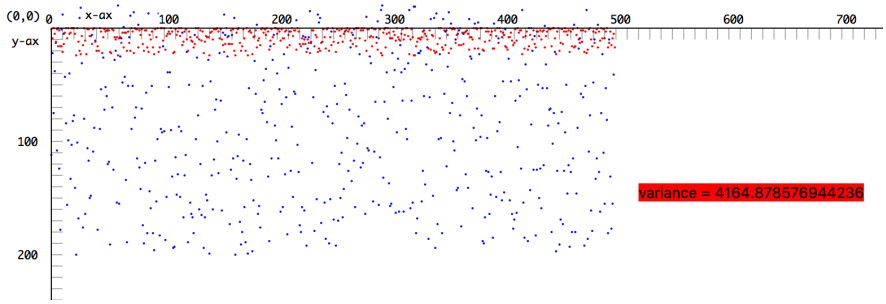


Figure 29 Display of the Variance.

The variance can be described as the measure of the spread of the numbers. As such, it calculates the expectation of the squared standard deviation per entry from its mean in the input stream[114]. The process results in generating new output (plotted as red dots, see Figure 29, page 60) with a much smaller range than the input stream (plotted in blue). Because the result is stored in the output, the variance calculation can be used to generate new musical material. Melodies, but also rhythmic patterns can be obtained with the calculated output stream. The mean-variance (the variance of all members), is plotted in the GUI of CACE4.

$$s^2 = \frac{\sum_{i=1}^{n}(x_i - K)^2 - (\sum_{i=1}^{n}(x_i - K))^2 / n}{n-1}$$ **where K is a constant (in this case: K=mean).** x_i **are the members of our input data. And** s^2 **is the variance.**

Equation 12 The Shifted data Variance.

<deviation>.

The Standard Deviation, together with the absolute and squared deviation (as plotted in Figure 30) of the input list, tells us something about the deviation of the members from the input stream. The output will consist of the three number values as plotted in the GUI (see Figure 30, page 61).

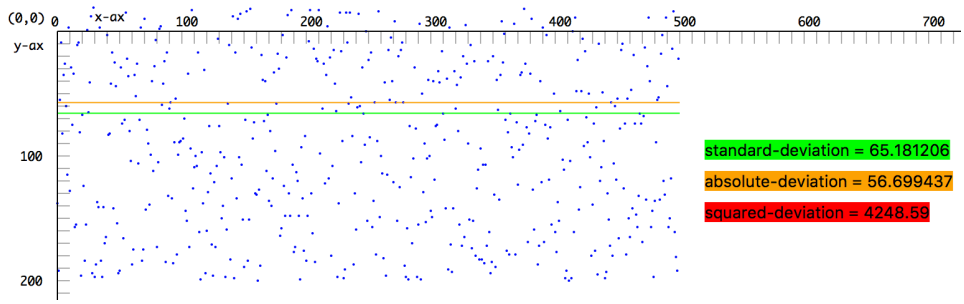


Figure 30 The display of the Standard Deviation, the Absolute Deviation and the Squared Deviation of the input stream.

$$s_n = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$ **Where** x_i **are the observed sample values and** $\bar{x}$ **stands for their mean. n is the number of entries, and** s_n **is the standard deviation.**

Equation 13 Standard Deviation formula.

[114] CACE4 uses the shifted data variance algorithm. The red pixels represent a weighted, individually calculated deviation from the variance.

<correlation>.

In general, correlation, as a mathematical/statistical procedure, is used to show a correlation between members of a data set. This is interesting for finding similarities between unrelated separate input streams, as is the case with the Pearson product moment coefficient, mostly used in CACE4. The Spearman correlation (or rank order correlation) is a special case of this Pearson correlation function. It is used for describing the relationship between two variables using a monotonic[115] function, and is as such less sensitive to members who are further away from the normal distribution.

The different correlation approaches will be discussed in more detail in section 5.5.5 (page 67): Correlator, together with the third correlation calculation in CACE4: Kendall **τ**.

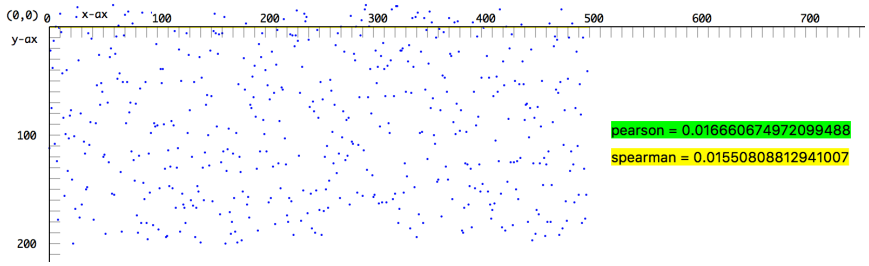


Figure 31 Display of the 2 different Correlation calculations.

The two numbers displayed in the GUI (correlation coefficients) are available as an output with 2 entries for further use.

<lin-regression>.

Linear Regression is a somewhat different approach than the processes previously described. It is used to find the line of best estimate and to find the regression pattern in the data (shown as the red line in Figure 32, page 63). As such it is also a more complicated process, but can be used in finding a linear direction of the data in the input stream.

$y = X\beta + \varepsilon$ **,where** X **denotes the transpose,** β **is the intercept and** ε **is an error variable (a non-observed random variable)[116].**

Equation 14 Linear Regression formula (generalized form).

[115] A monotonic function is a function where all numbers are entirely increasing or decreasing. For all x and y: $f(x) \leq f(y)$ or $f(x) \geq f(y)$.

[116] In linear algebra the transpose is defined as an operator, which is a matrix, flipping over its diagonal. The intercept is defined as a point on the y-ax where the function intersects with the y-ax (Also called the y-intercept or vertical intercept).

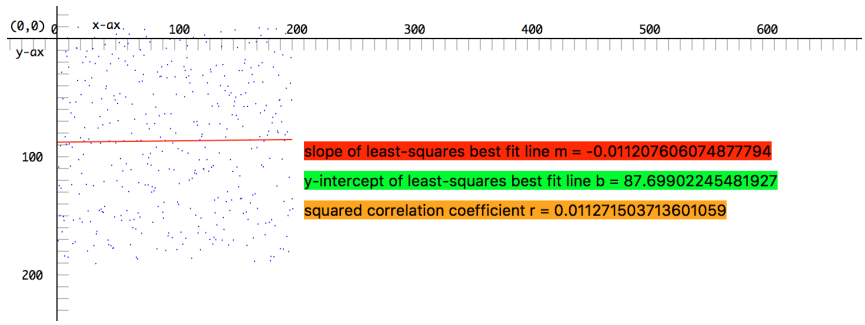


Figure 32 Display of the Linear Regression calculation.

As can be observed in Figure 32 the generated x-values[117], previously plotted, are omitted in the calculation as well as in the plotting of the data (GUI). The line plotted shows the slope and direction of the linear regression of the input list. As previously described in section 5.4.2, the output stream consists only of four numbers[118]. For now there is no direct relation of these four numbers with a musical process[119]. However the slope of the least squares (m) can be used as a selection criterion for sieving the generated newly calculated input (see section 5.4.4 for further detail about this use).

<histogram>.

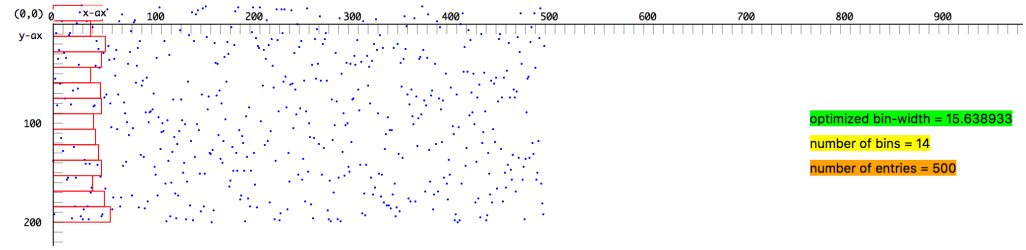


Figure 33 Display of Histogram calculation of the input list.

A Histogram has been implemented as an alternative way of grouping the input data. The algorithm splits all entries according to their y-value into discrete bins (the x-value of the plotted xy-pair is omitted).

With the aid of a special histogram bandwidth optimisation technique, the data can be put into flexible sized bins. The algorithm is implemented with this flexible bin size approach in the manner as has been proposed by H. Shimazaki and S. Shinomoto (Shimazaki and Shinomoto 2007). The number of bins can be entered as an initial maximum value for the algorithm. The algorithm will find an

[117] For plotting purposes in all other statistical plots, generated x-values are the index numbers of the member in the data and are the actual index position of the member itself in the data (list). In linear regression this generated x-value is omitted and only the original input values are plotted.

[118] The first number is the number of entries in the input (*n*). The second *m*: the slope of the least-square, best fit line. Third *b*: the y-intercept of the least-square, best fit line. Fourth *r*: the squared correlation coefficient.

[119] In future versions of CACE4 a 'prediction' based object will be added. This will make use of the linear regression algorithm in order to calculate new data for the output stream.

optimum bin-width value (see Figure 33, page 63: printed in light-green: optimised bin-width) Therefore the exact number of bins (Figure 33, in yellow) detected can vary from the one entered. The output (mostly 10 to 20 entries) will be stored as a stream consisting of the number of entries per bin, in sequential order from low to high.

5.5.4 STAPS: a STAtistical Property Sieve.

Although sharing the same statistical calculations with the previously described CACE4 STAM object (see section 5.5.3 for a detailed description), the calculation methods are used in a totally different way for generating a totally different kind of output. The ***STA***tistical ***P***roperty ***S***ieve or STAPS acts as a statistical procedure sieve. The STAPS object works in 2 separate stages: first the analysis of the data according selected statistical procedures (see Figure 34). Then these found statistical values are used as parameters for constructing a new value. This 'new' value is obtained by generating a random number and checked if this new value is inside specified boundaries. If found true; use this number otherwise, omit value and generate a new one until a number suits these checked statistical properties. This procedure is according to the way a sieve algorithm functions.

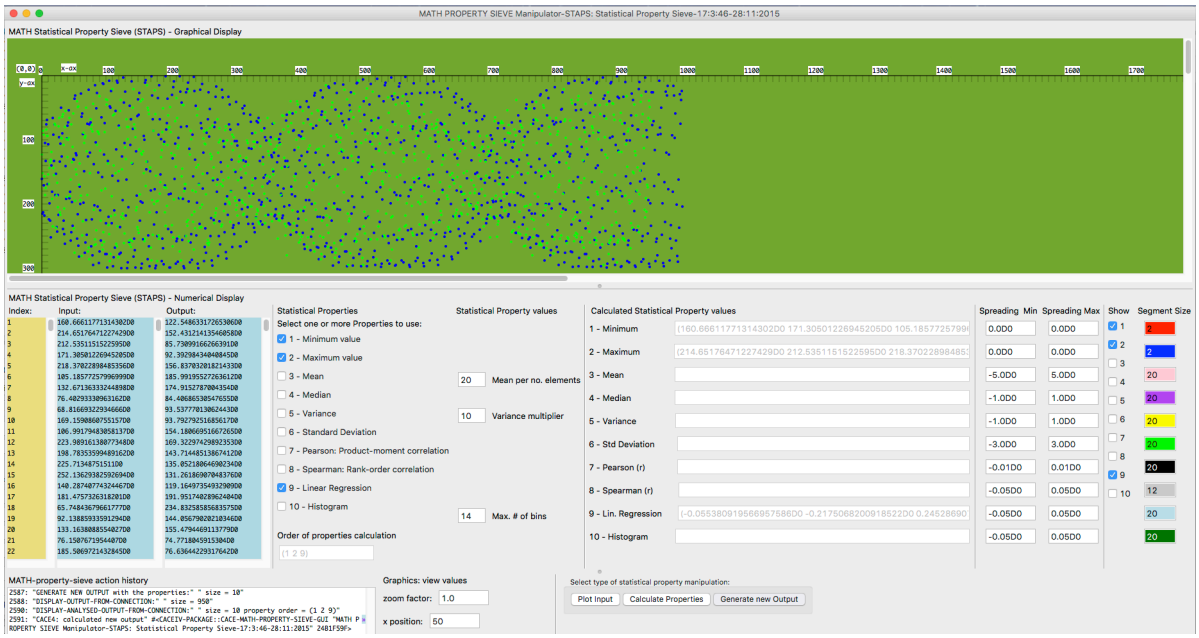


Figure 34 Display of the STAtistical Property Sieve or STAPS GUI.
The original input list is plotted in blue and the newly generated output in light-green. NB Notice use of coloured background GUI option.

The CACE4 STAPS object has a variable analysis window size for every statistical function. Therefore an independent optimal analysis window size for every selected calculation is used. Because the algorithm has to makes multiple calculations, the possible order of selecting a statistical procedure

influences the output: $R_{calc} = [calc_1, calc_2, ..calc_{10}]$, where R_{calc} represents the order of all chosen calculations. The output of *calc1* is used as input for *calc2*, up to the last calculation selected. Therefore, unselecting and selecting in the appropriate order can alter the rank order of the calculation sequence. The created order will be shown in the 'Order of properties calculation' text-display-field. Select <Calculate Properties> will obtain the initial lists with calculated values. The results, as lists of numbers, will be displayed in the GUI in the column: 'Calculated Statistical Property values' (see Figure 34, page 64).

After the initial analysis, it is possible to rerun the analysis after alterations are made for the variables. This procedure can be re-ran until the results look satisfying. The final result is obtained by executing the calculations with the generated sieve properties: select <Generate new Output> and wait for the result. As a feature of the GUI, alteration of the background colour is possible for all CACE4 objects. By pressing the mouse and moving left <-> right over the object output display area, the background colour can be altered. By slowly changing the background colour from blue to dark-green, the two different coloured pixel groups blend with their particular background colour. This helps in focusing from original input to generated output and vice versa, making a comparison between the original input and the output result much easier. The underlying algorithm design makes it possible to obtain a wide variety of either close imitations of the input list. e.g. by using small analysis windows (Analysis size <= 2), or less related output (analysis size => 20). In addition to this, a particular order in processing influences the outcome of the process as well. To obtain a satisfying result, several attempts must be made together with some experimentation with the values of the independent sliding analysis window.

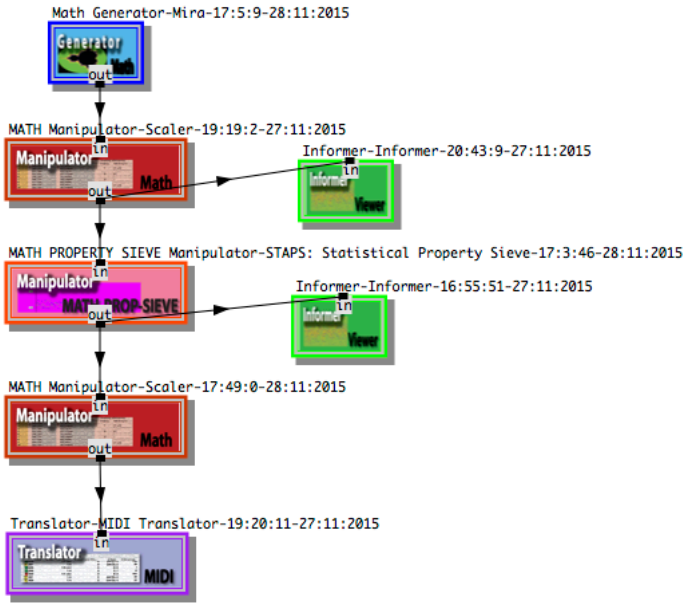


Figure 35 Example of a small strategy setup for using the STAPS object in the CACE Processor window.

In a CACE4 strategy chain a STAPS object can be used to create sets of numbers with calculated imitations of the original input stream. As an example strategy setup see Figure 35 (page 65), where a CACE4 MATH Generator a Gumowski-Mira fractal calculation is used (see Figure 36, page 66).

$$F(x) = ax + \left(\frac{2(1-a)x^2}{1+x^2} \right)$$

$$x_{n+1} = y_n - F(x_n)$$

$$y_{n+1} = -x_n + F(x_{n+1})$$

Equation 15 Gumowski-Mira fractal set of equations.

NB Equations are according Hans Lauwerier (Lauwerier 1987, p. 108 - 109).

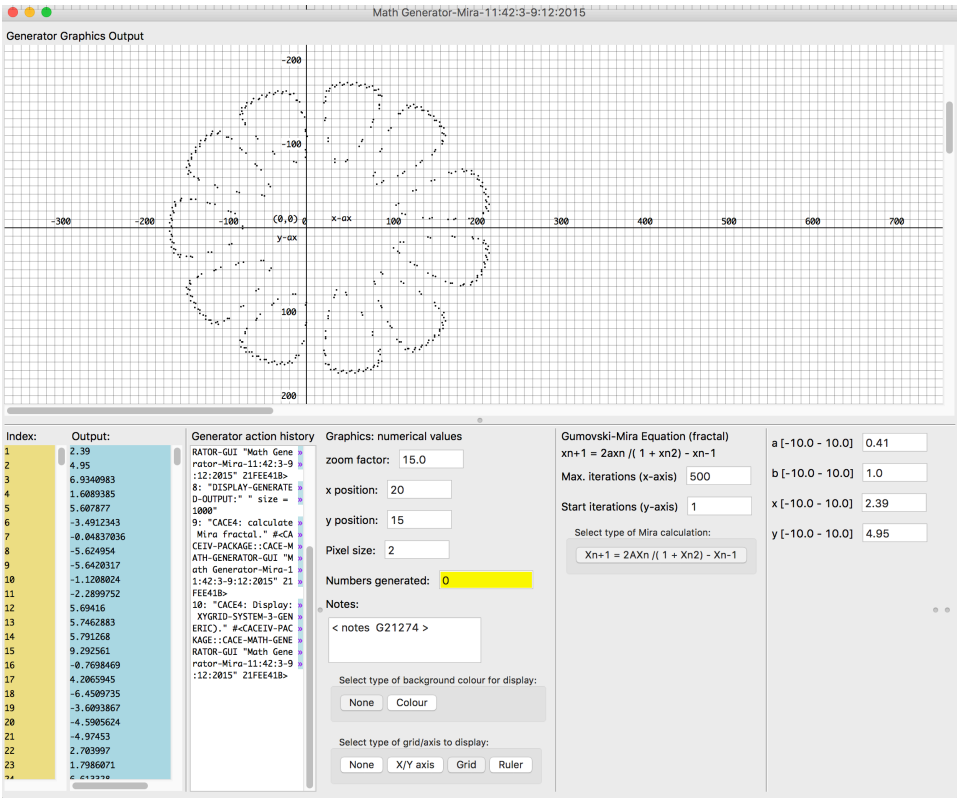


Figure 36 Display of the original GUI of a MATH Generator window.
In this case we used a Gumowski-Mira fractal as input for the STAPS Object.

After generating 500 numbers with, in this case a fractal f(x) with output according to equation 16, an informer object can be attached directly at its output, sin order to view the generated data in a different way (see Figure 35, page 65, for a possible CACE4 strategy chain). The difference in the plotting of the data is caused by instead of plotting them as number pairs (x,y), a new x-value is generated and the original (x,y) number pairs are both plotted as independent y-values. In plotting the Gumowski-Mira fractal this way we can observe a link between the original x and y value (see Figure 37, page 67).

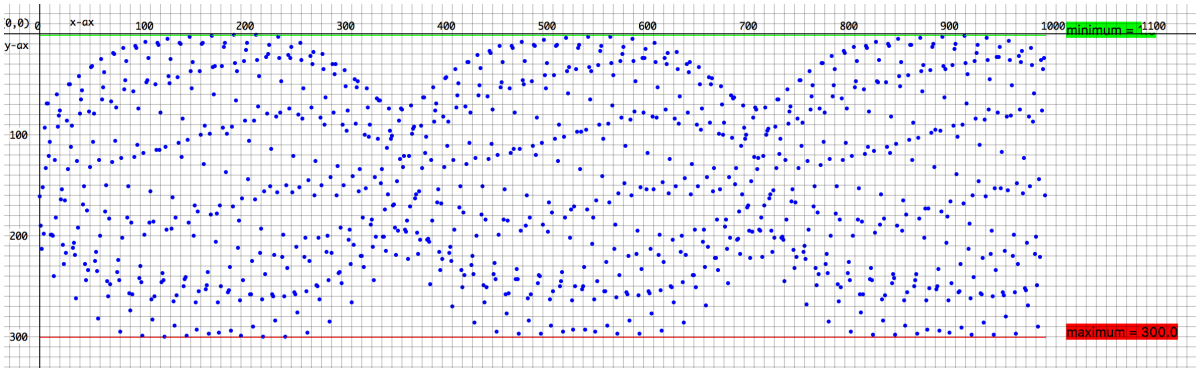


Figure 37 The output of the Gumowski-Mira fractal displayed in the Informer object.

Differences can now be seen between the original Gumowski-Mira fractal displayed as in Figure 37 (page 67) and the data displayed in a second Informer object (see Figure 38), attached to the output of the CACE4 STAPS object and displaying the newly generated data.

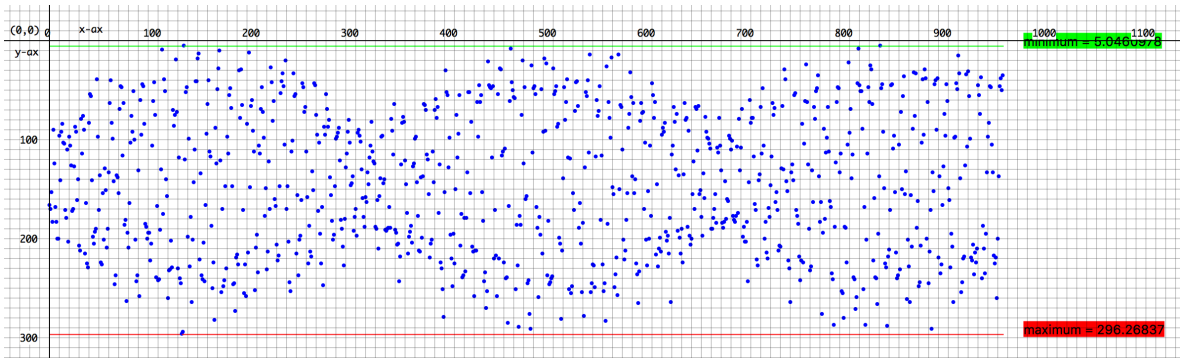


Figure 38 The output displayed in an Informer object after STAPS has been used.
NB The output of STAPS changes every time it is used. There is a random involved in the algorithm.

The outside shape of the original Gumowski-Mira fractal output is still recognizable, although all of its displayed members are newly generated numbers. All is done according to the parameter settings in the CACE4 STAPS object window (GUI display see Figure 34, page 64). In this case it uses only three properties: minimum, maximum and the linear regression of the input stream to create a new series of numbers as an output stream.

Because the first step in the algorithm number ‘creation’ is done with the aid of a random number, rerunning the process in the CACE4 STAPS object will always result in different numerical output, even if no parameters are altered.

5.5.5 Correlator.

This section focuses on the use of the ‘Correlator', a CACE4 Manipulator object based on three kinds of different correlation calculations. It is checking the input by a correlation algorithm for finding

correlated[120] content in one, or between more, input patterns. In this particular case the two input lists (as stream 1 and stream 2) have both their own input streams with independent correlation type and correlation value. The selected input streams are then mixed before processing takes place, according to choice: added, sorted or 'zipped' (interleaved). The Correlator calculates two new output streams either of which can also be added, sorted or interleaved according to choice. Figure 39 (page 68), shows us the design of the algorithm for the CACE4 Correlator object. The order of attaching an output of a CACE4 object to the input of the CACE4 Correlator object is also the order in which the input streams will be used for doing the correlation calculation[121].

Figure 41 (page 72), shows as an example, a possible setup of a strategy in a CACE4 Processor, involving the CACE4 Correlator.

Since correlation has been defined as a mathematical equation for finding a correlation between two or more points (mostly taken from a larger data set), it can be used for finding a certain mathematical relation between these detached datasets. This relation can be expressed in the output r (= correlation coefficient) with a certain range: $r \in [-1.0, 1.0]$ for Kendall τ (tau) and $r \in [0.0, 1.0]$ for Pearson and Spearman. For Kendall τ this means if r is -1.0 there is perfect disagreement between two rankings.

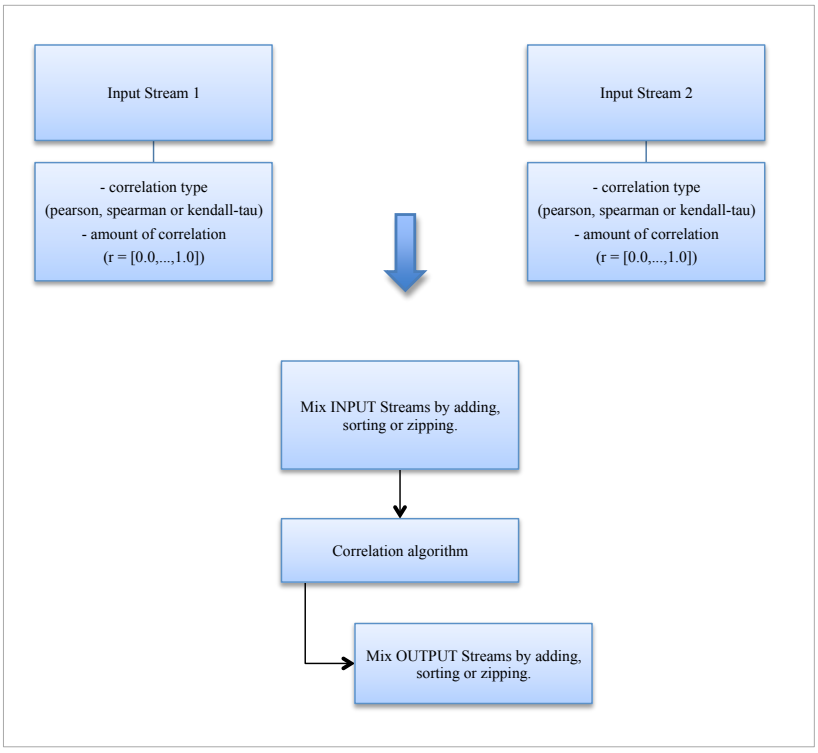


Figure 39 The Correlator algorithm design.
In this particular case two independent input lists (Stream 1 & 2) Both input streams with its independent correlation type and amount value. Then mixed according choice: added, sorted or 'zipped'. Then the Correlator with its entered parameter values makes the calculation. And the output can either be added, sorted or 'zipped', according to choice.

[120] Correlation is a measure for linear dependency between 2 numbers.
[121] In future versions this will be made available as a feature in the GUI itself. This enhances further flexibility in experimentation with the algorithms and their correlation values.

With $r = 0.0$ there is an independency of the two elements and with $r = 1.0$ there is a perfect correlation in ranking between the two elements. Correspondingly, when the values for Pearson and Spearman are $r <= 0.0$, there is negative correlation between the two sets of data. At the other end of the scale, with a maximum value of $r = 1.0$ it will provide a strong correlation of the used dataset. Figure 39, page 68 shows the description of the algorithm used, in this case with two independent input streams (as a LISP type: list). Both input streams with their own independent correlation type and amount value, are mixed according to choice. The generated output can then be recombined into new, ordered streams according to the selection criteria set.

The correlation coefficient, needed as selection parameter for the calculation, is displayed on the right hand side of the GUI, as a set of red drawbars (Figure 40, page 69), this enables the coefficient (r) to be altered as one chooses, in order to use it as a boundary checker for the stream input, by means of changing the value of the drawbars[122].

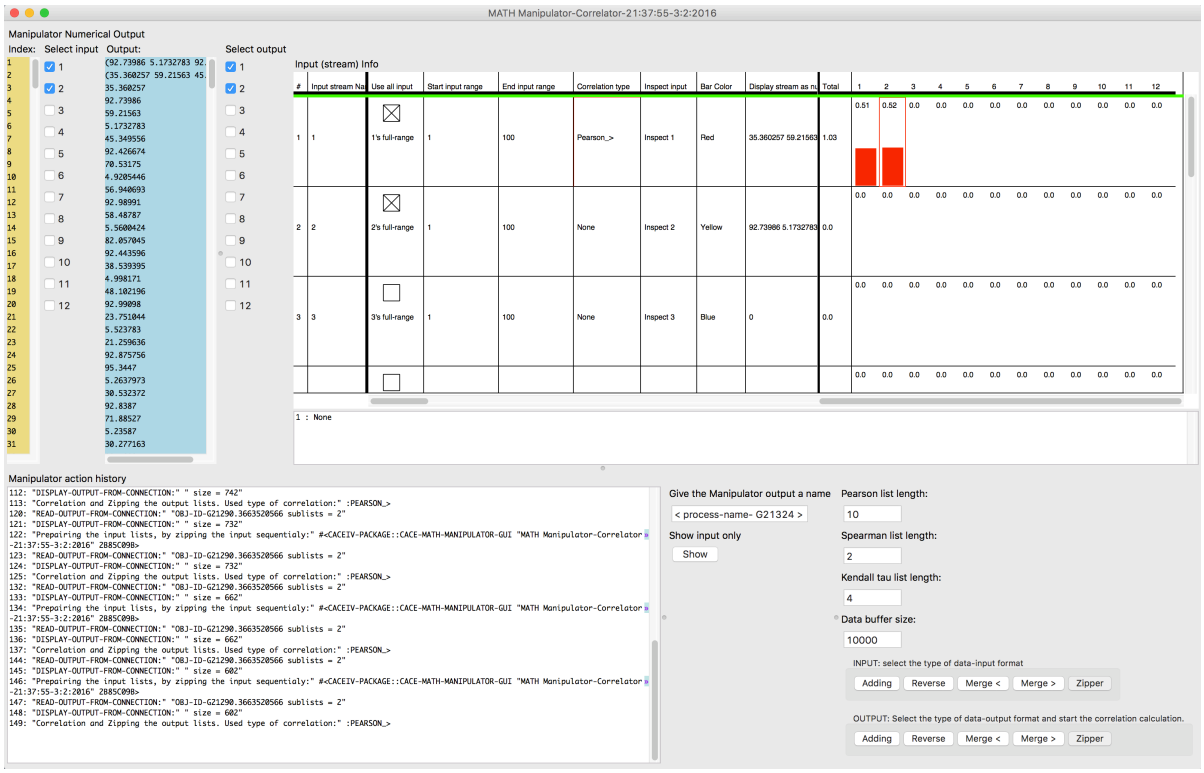


Figure 40 CACE4 Correlator GUI.

[122] The CACE4 object Correlator makes it possible to make a selection above the value indicated by the coloured sliders at the right hand side or use data below the correlation value for the analyzed stream.

Further to the left, the menu option can be found where <, <=, > or >= can be selected, together with the three types of correlation[123].

As an example: if *r*=0.5 is used, the obtained output stream of selected members has a value correlation coefficient (*r*) of a minimum of 0.5 up to a maximum of 1.0. But it is also possible to make a selection the other way around: in that case everything below the minimum limit (< and <=) is selected, resulting in an output stream consisting of entries less or equal then the selected value. This will result in an output with a much less strict correlation between the output elements.
The algorithm was designed to work also on specific selections of a single (one-dimensional) data stream. A start-index and end-index value can be used to create a separate, smaller chunk of the data stream (In the GUI this is shown as start input range and end input range, see Figure 40, page 69).

The Pearson correlation coefficient (PCC), or product moment correlation coefficient, acts as a measure for calculating the linear dependency between 2 variables in separated datasets (X and Y) and was originally proposed by Pearson (1895). This linear dependency is written as r, the correlation coefficient (see Equation 16). And is a real number between: $r \in [-1.0,\cdots,1.0]$.

$$r = r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$ **where** $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$, $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ **(is the sample mean), n is the number of entries and r is the correlation coefficent.**

Equation 16 Pearson equation formula.

Spearman is a rank order correlation calculation originally proposed by Spearman (1904).

$$\rho = 1 - \frac{6\sum d_i^2}{n(n^2-1)}$$ **, where** $d_i = rg(X_i) - rg(Y_i)$, rg **is the difference between the two ranks of each observation, n is the number of entries (observations) and** ρ **(rho) is the correlation coefficent.**

Equation 17 Spearman equation formula.

Spearman is defined as a special case of the Pearson product moment correlation. It calculates the correlation of the rank order of the entries in the input, instead of the correlation of the entries themselves.

[123] The initial key-bindings of LispWorks are used in the GUI of CACE4. Therefore changes of the menu selection need to be confirmed by using the enter key (and not the return key). The issue of key-binding will be addressed in future versions of CACE4.

The third correlation calculation available in the CACE4 Correlator is the Kendall rank correlation coefficient or Kendall **τ** correlation coefficient.

$$\tau = \frac{(ncp) - (ndp)}{\frac{1}{2}n(n-1)}$$ **NB. ncp = number of concordant pairs, ndp = number of discordant pairs. N is the number of entries. And τ is our correlation coefficient.**

Equation 18 The Kendall τ formula.

Maurice Kendall originally proposed this rank order correlation calculation in 1938. It is the third correlation calculation in CACE4 and is also known as Kendall's tau coefficient. It is a statistical technique, just as Spearman is for measuring the rank order between two adjacent members of any given data set. It is therefore also a rank order correlation but based on a tau test: a non-parametric hypothesis test, specially developed for measuring the dependencies based on the tau coefficient principles. The Kendall **τ** algorithm works with two lists of concordant and discordant pairs of numbers. A concordant pair is a number pair of observations, expressed as: $\{X_1,Y_1\}$ and $\{X_2,Y_2\}$, with the property: $\text{sgn}(X_2 - X_1) = \text{sgn}(Y_2 - Y_1)$, where *sgn* is either negative, zero or positive: -1, 0 or 1. This holds as long both members of the number pair have either a higher, lower or equal value as the corresponding other number pair.

A discordant pair is also a number pair of observations, expressed as: $\{X_1,Y_1\}$ and $\{X_2,Y_2\}$, but now with the property: $\text{sgn}(X_2 - X_1) = -\text{sgn}(Y_2 - Y_1)$, where *sgn* is also either negative, zero or positive: -1, 0 or 1. This must be true for the discordant number pairs $\{X_1,Y_1\}$, which have a higher value of *X* as the other corresponding $\{X_2,Y_2\}$ number pair, which should also have a higher value of *Y*.

First sorted according these principles, now entries from both lists are used for calculating the Kendall **τ** correlation coefficient.

All three correlation algorithms described can be used in this way as a measurement of linear dependencies or independencies of the dataset, thus, as previously stated, giving a kind of indicator for the amount of correlation between the points of measurements. The data grouped in sequential sections as datasets gives an indication of a rank order correlation calculation or as a product moment correlation calculation.

For all three correlation calculations in the CACE4 Correlator, the output *r* could act as a gradual linear dependency controller between two, or more input streams. Used in this specific way, it is possible to use correlation calculations as a tool for finding (dis-)similarities between two originally separated input streams (see Figure 41, page 72 for an example of a setup of a CACE4 strategy).

Extrapolating the strategy and result of the correlation algorithm onto the musical domain, it could be used, for example, when writing a composition for several instruments. To be more specific: looking for similarity or dissimilarity between several instrumental parts provides another way of approaching the well-known practice of counterpoint. Thus not based on sets of well-described sets of tonal rules, but on a mathematical base: executed by a correlation calculation. This is based on the idea that if there is less correlation between two streams: there is also less connection between the two musical patterns as well.

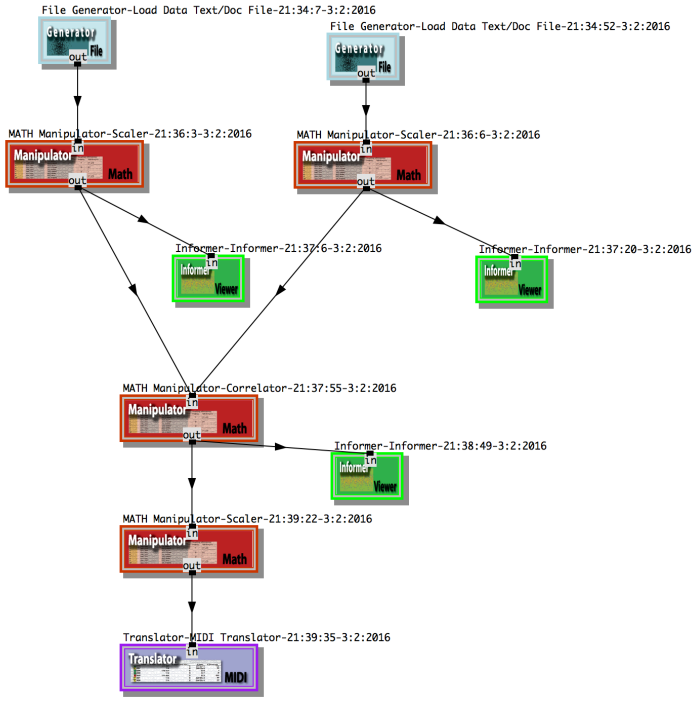


Figure 41 A CACE4 Strategy setup for using a CACE4 Correlator.

This gives the instruments more independence in their voicing. With this design, it is possible to create single lines or short motives (defined as a melody and/or as a rhythmic pattern[124]), extending from simple repeated entries resulting in monotonic lines, to, at the other end of the scale, wild and non-repeated chaotic melody lines[125].

[124] Although in all cases of using the CACE4 environment, it is only with the use of the Translator Object that the stream gets its real musical context in the form of the MIDI protocol: delta start time – key –velocity – duration, and this context output is fixed by using the send it to the score button. By using the scaling object earlier in the chain of linked objects however, it is possible to use its output as a scaled version of the calculated output adapted to the MIDI protocol, without actual translation to the MIDI domain: this still has to be performed by a Translator Object in the end of the chain of linked objects.

[125] Some further experimentation needs to be done in order to find out if there are any other musical possibilities with the CACE4 Correlator object.

5.5.6 Scaler.

The Scaler acts as an all-round linear or logarithmic scaling object.

In the case of Figure 42 (page 73), which acts as an example of scaling the data to the MIDI range, it can be observed that index 1 is used for linear scaling all elements in the input list according to the values for Minimum = 62.5[126] and Maximum = 1500.

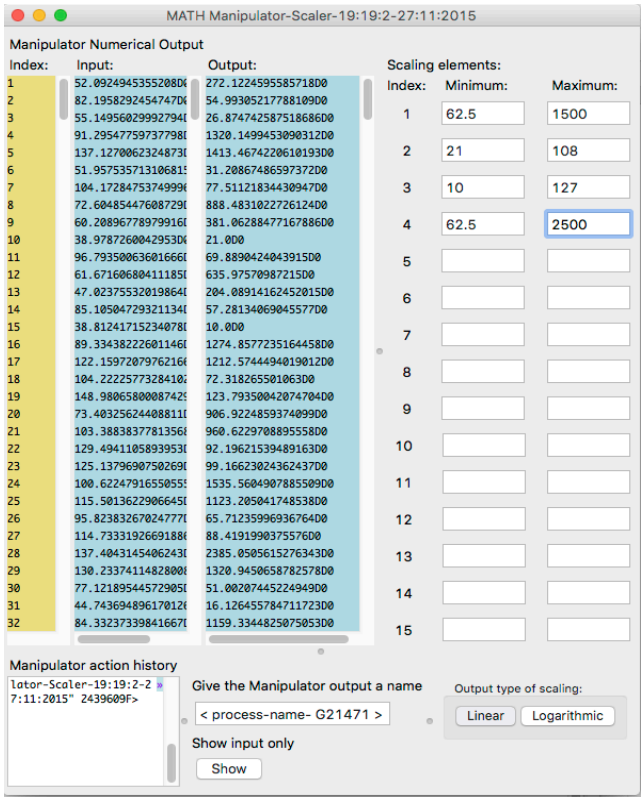


Figure 42 The Scaler object with 4 indexed Minimum and Maximum values.
It creates a new output list with values in range according MIDI protocol for delta-time (in miliseconds), pitch/key [Piano range = 21-108], velocity [10-127] and duration (also in miliseconds).

This corresponds with the MIDI Clock Ticks setting (with tempo marking MM=120), for 62.5 -> 1/32 note and 1500 -> ½ note. The 'hop-size' for the index is calculated according to the amount of rows of entered minimum and maximum values used. As an example: the first value will be used to calculate the (CACE4 Translator object) MIDI delta start-time. This is the inter onset time (as described in Dessain and Honing 1992, p. 46), in milliseconds (tempo 120) between the start time of the notes. MIDI (Byte): value 21 = A0 (= piano key). The maximum is the highest key in MIDI (Byte) with a value of: 108 = C8 (= piano key).

The index 3 Minimum entry is the MIDI Velocity range: 10 = approx. pppp - ppppp[127]. Here it is not

[126] Both numbers (Minimum and Maximum) represent milliseconds.
[127] Music dynamic notation: they can be perceived as extremely soft dynamic registers. In the case of the forte signs it is the opposite, as extreme loud dynamic registers.

0, but 0 or 1 as a Minimum (MIDI velocity) value can be used as well. The maximum value used is 127 = fffff.

The last index, number 4 is the duration of the note. In this case (see Figure 42, page 73) the Minimum value 62.5 (in milliseconds, when used tempo 120) stands for a 1/32 note, the same as the first element: delta-time. The Scaler object acts as a linear-scaler for the input list and works according to its description in Equation 19 (page 74).

$$(x_i - a_{in\,\min})\frac{(d_{out\,\max} - c_{out\,\min})}{(b_{in\,\max} - a_{in\,\min})} \quad \text{where}$$

$a_{in\,\min}$ = minimum of input list.

$b_{in\,\max}$ = maximum of input list.

$c_{out\,\min}$ = minimum value range for the output.

$c_{out\,\min}$ = maximum value range for the output.

Where $[a_{in\,\min}, b_{in\,\max}]$ represents the range of the original input list.

And $[c_{out\,\min}, d_{out\,\max}]$ is the range of the newly calculated output.

Equation 19 Algorithm description of the CACE4 linear scaler.

By adding more minimum and maximum values in the GUI of the CACE4 Scaler object (see Figure 42, page 73), the index is also altered by adding 1. After setting the range accorded to values for every index member, select linear. This will result in creating new values according to the Minimum and Maximum values entered[128]. In the case of Figure 42: a separate vector of 4 members scaled to the values entered acts as a pre-MIDI scaling for a CACE4 MIDI Translator object, attached to its output. Using this CACE4 Scaler object connected between a CACE4 FILE Generator object and a CACE4 ML/MIR object is also good practice. In order to obtain a good spreading before doing any calculus, with the points read in from a plain text file, the Scaler object is used to scale the new entries to the appropriate new values inside the minimum and maximum range ($x \in [\min, \max]$).

5.5.7 Disturber.

The Disturber (see Figure 43, page 75) acts as a randomiser on the elements of the input list. It is also an algorithm with an adaptable index just like the CACE4 Scaler object. Each member accorded to the index will be randomly altered by an indicated percentage. With separate percentages for altering the output randomly, the Disturber is an interesting tool for artistic purposes. Up to fifteen

[128] In case the Logarithmic calculation is used: the maximum output value will be exceeded. This is inevitable with the use of the logarithm as scaling calculation.

separate indexed input data streams can be created, all with an independent variable percentage of randomness.

There is a slight difference in generated output between the use of <Disturbance 1> and the use of <Disturbance 2>. The first one has a range of [minimum, maximum] and the output is centred round *fc* (*fc* = maximum - minimum / 2). And <Disturbance 2> has a range of [0, maximum], and all output is above *fc*, as can be seen in Figure 45 and Figure 46 (page 76).

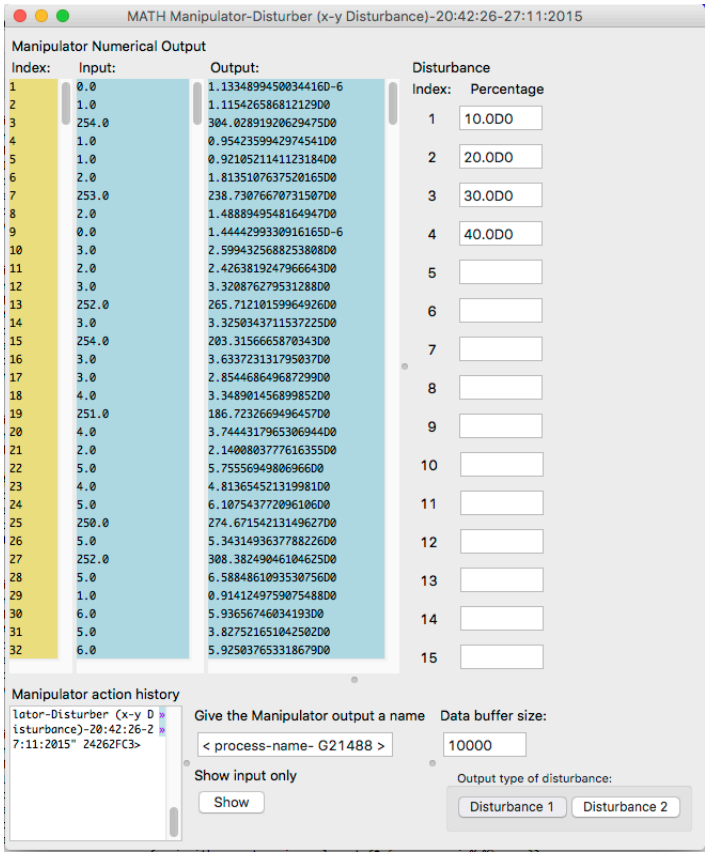


Figure 43 Display of the Disturber object with 4 indexed percentage values.

$$R_{range} = (\frac{abs(n_{\max} - n_{\min})}{100.0}) \times percentage$$

$$j_n = i_n + random(R_{range})$$

, where j_n is the calculated output and i_n is the input. $n_{\min}, n_{\max}$ are the observed minimum and maximum value of our input. R is the range used as seed for the random.

Equation 20 Disturber algorithm description.

The next example shows these differences. It uses a Brown fractal as a CACE4 Generator object.

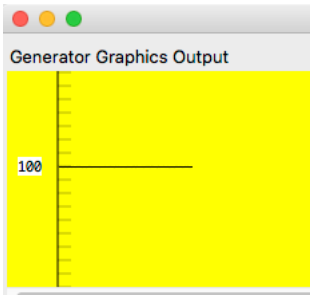
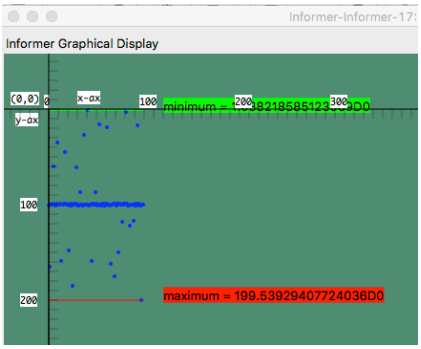


Figure 44 The display of the output of the Brown fractal (as a straight line).

The Brown fractal generates 100 numbers (with a value of 100.0) in a straight line, as shown in Figure 44, page 76.

The next two figures show the difference in output of the bandwidth and distribution of the generated output. The disturbance percentage is for index i1=1.0%, i2=100.0%, i3=1.0% and i4=1.0%, and are in both cases identical.



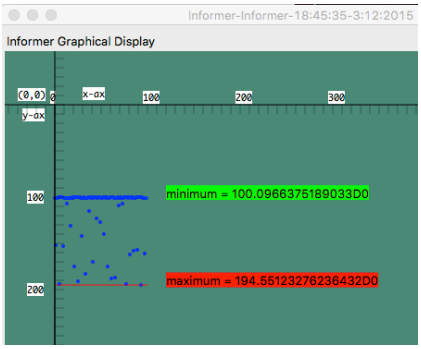


Figure 45 The output of <Disturbance 1>. NB Displayed by an Informer object. (see set up: Figure 47).

Figure 46 The output of <Disturbance 2>.

Figure 45 shows the newly calculated output both up and down the initial value of 100. Here the new values can now be found in the range [0.0,200.0]. Figure 46, shows the same procedure but now all entries are in the range: [100.0, 200.0]. The two different approaches result in different output. As an example of use in a strategy chain in the CACE4 Process window, a setup was created as shown in Figure 47 (page 77). The setup starts with a Generator object as a first member in the chain of processing (Keep in mind that one always has to put a Generator object at the first position of the processor-chain). In this case a Cellular Automaton fractal (Sierpinski sieve) calculation is used, which acts as the chosen source of data. Other CACE4 Generator objects can be used (Text files and SPEAR partials text file are the 2 other CACE4 Generator objects).

$$a = 2$$
$$i = j - 1$$
$$k = j + 1$$
$$R_j = \mathrm{mod}(Q_i + Q_k, a)$$

, where a = modulo divisor, j = number of iterations, i and k are index numbers for Q.

Equation 21 Sierpinski triangle (also called Sierpinski gasket or sieve) Automaton Algorithm description.

After entering the Cellular Automaton fractal (see equation 22), calculation with a = 2, x-maximum = 256 and y-maximum = 50 values, a total of 832 new output list members is generated.

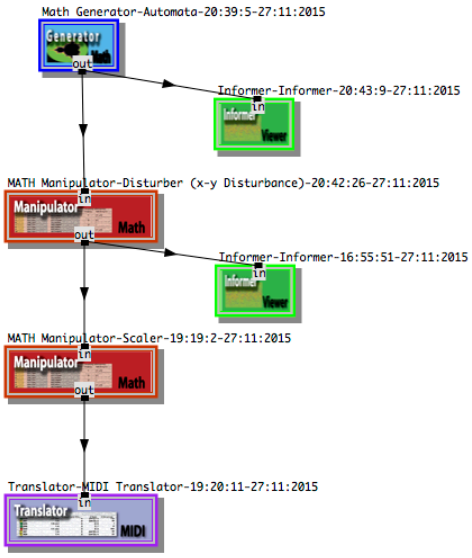


Figure 47 Example of a small strategy setup for the Disturber object in the CACE4 Process window.

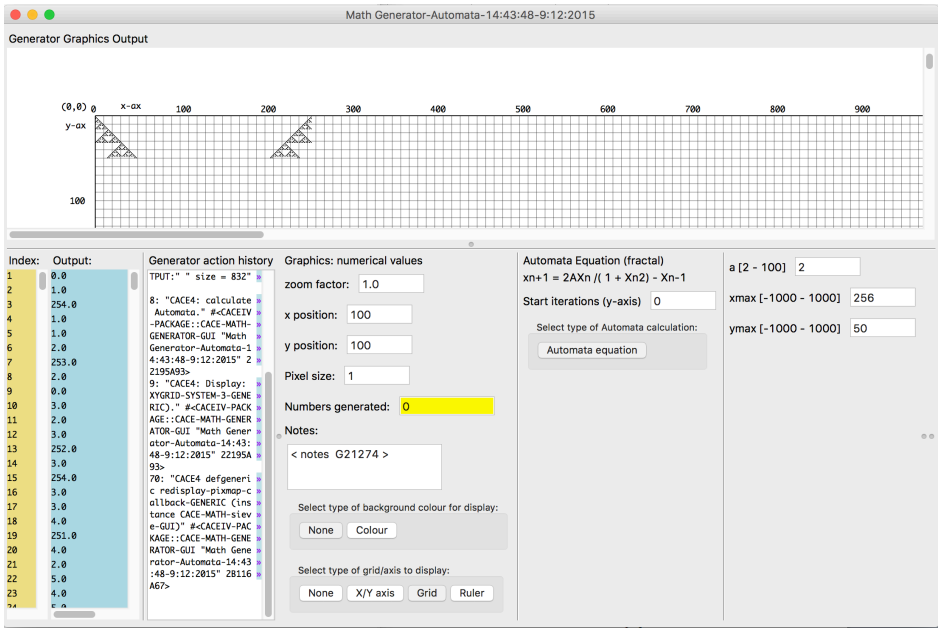


Figure 48 Display of the GUI of a MATH Generator (Cellular Automaton fractal, n=832).

(See Figure 48, page 77). The numerical output is always printed in the blue (scrollable) column (see Figure 48, bottom left)[129].

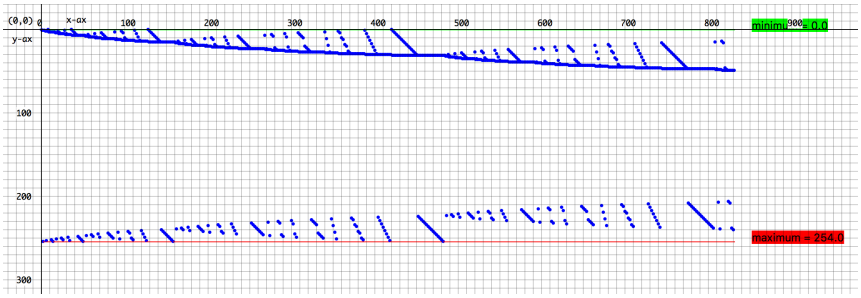


Figure 49 The Informer object shows the same output as the Generator object except the display has been altered. For every member a new (x,y) number pair has been created and shown in a (x,y) scatter plot.

For a different type of output plotting, an Informer object is attached to display the data in a (x,y) pair orientated scatter plot (see Figure 49). Every member of the input list acts as the y value of a newly created (x,y) pair. The original x value is replaced by $x_{new} = i_{i+1}$. This results in a different display of the output and now statistics can be applied to the input list as well.

Next step in line is the Disturber object. According to the entered settings (see Figure 43, page 75), and after selecting <disturbance 1> new output has been generated. The output of the random process is stored in the output slot of the object (see Figure 50, page 78).

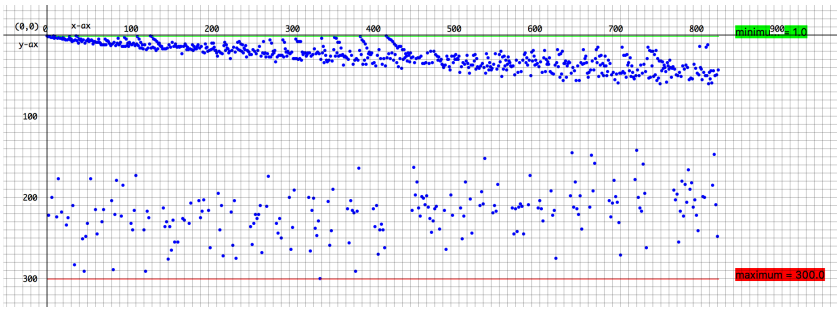


Figure 50 Display of another CACE Informer objects, attached to the Disturber object. It is showing the members as (x,y) pairs with newly created x-values and the original x-values : as y-values.

If an Informer object is attached to the newly, now disturbed output, and this then is plotted, as seen in Figure 50, the y value of the plotted pixels in the graph represents their true value. A difference in disturbance between the upper ($y < 100.0$) and the lower ($y > 150.0$) band of pixels can be noticed directly[130]. This has been achieved by setting the second entries (with a jump size of 4) to 100 %

[129] Both types of display of the data obtained by the fractal calculation can give the user an adequate perception of the data generated.
[130] NB The original disturbance percentage is for index i1=1.0%, i2=100.0%, i3=1.0% and i4=1.0%.

disturbance. The first, third and fourth entries are altered with only a 1% setting for disturbance, in the CACE4 Disturber object.

5.6 The CACE ML_MIR Manipulator object group.

This Special Group of CACE4 Manipulator objects can be seen as the most important group of objects of CACE4. It offers other tools, more advanced than those discussed earlier.

5.6.1 *k*-Means.

k-Means, also called Lloyd's algorithm, together with the Expectation Maximisation (see section 5.6.2) algorithm, belongs to the group of Hierarchical Clusters. Data is grouped in (x,y) pairs. The weighting of the input will be done strictly on a numerical basis and is unsupervised. Only the number of detectable clusters is an editable parameter of the model.

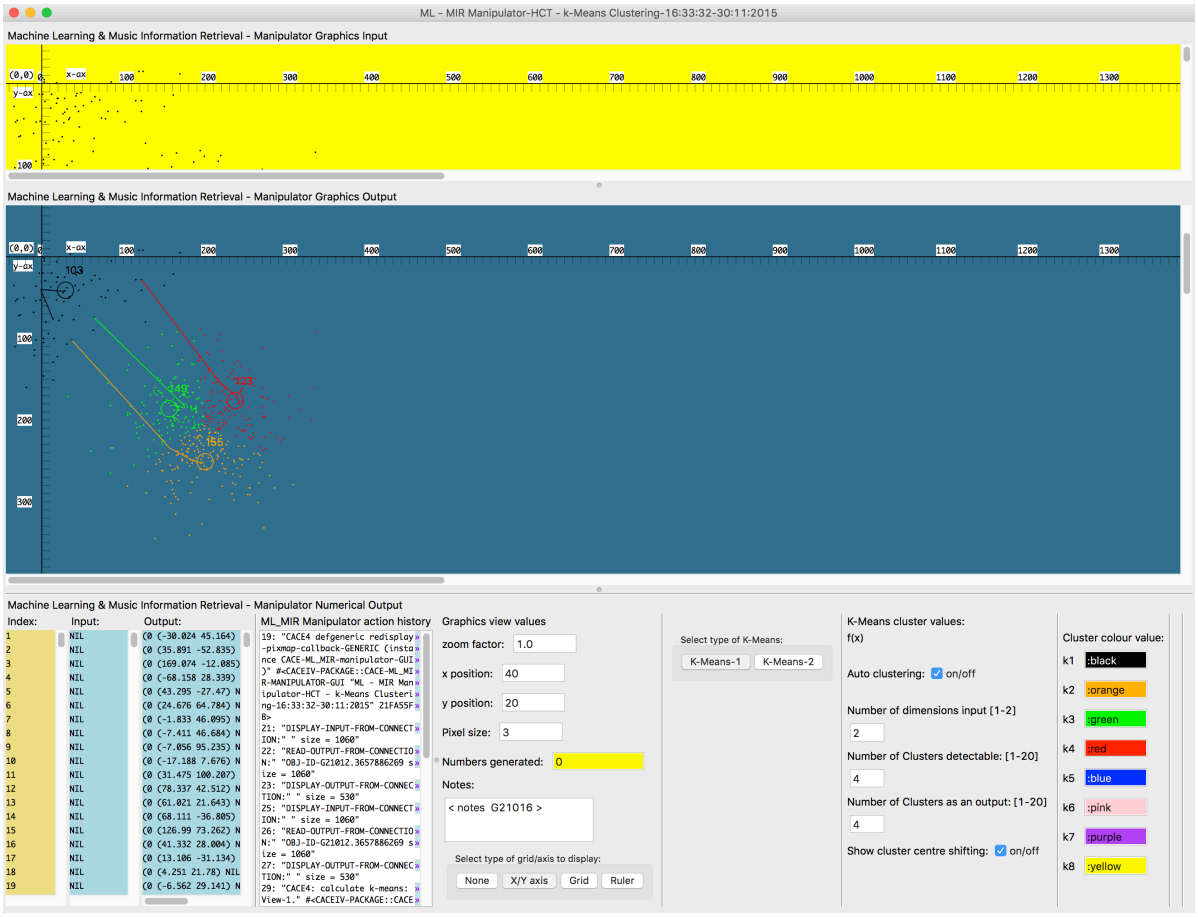


Figure 51 *k*-Means GUI.
The original input displayed in top, yellow background, and the grouped data by *k*-means in the middle part. Each group is coloured according the Cluster colour value (right hand side).

When working with a *k*-means algorithm, visual output, as is shown in Figure 51, page 79 is

imperative. Using colour coding for the different categories, after plotting, facilitates their distinction. *k*-Means is a Hierarchical Cluster Technique (HCT) and is defined as a method for quantifying vectors: in this case presented as (x,y) data pairs. *k*-Means is used for the partitioning of *n* observations (in this case *n* (x,y) data pairs) into a number (*k*) clusters. *k*-Means partitions these into more or less blocks of the same size.

Looking at Figure 51 (page 79), it is immediately observed that just two categories would be sufficient. Therefore in the second attempt to find the clusters, the number of detectable clusters is changed to two. The algorithm is based on two steps: first there is an initial assignment phase, followed by step two, an update phase. Both phases of the algorithm are reflected in the two formulas: Equation 22 and Equation 23 (both at page 80). The first step (see Equation 22) is done for assigning every member of the data set $S_i^{(t)}$ to a unique cluster centre. Initially a set of randomly chosen means (as the centres of the clusters) values: $m_1, \ldots m_k$, where *k* is the desired number of clusters to be detected. In step 2 (see Equation 23) the value is now recalculated as the mean of Euclidian distances[131] between the members of each cluster (*k*). This second step is used for calculating new values for the means, which are the centroids[132] of the clusters.

The two steps are an alternating process between assigning (step 1) and updating (step 2), until $m_1, \ldots, m_k$ no longer converges to new values. The cluster centres (means) have reached their final value and the *k*-means algorithm stops.

$$S_i^{(t)} = \{x_p : \left\| x_p - m_i^{(t)} \right\|^2 \le \left\| x_p - m_j^{(t)} \right\|^2 \forall_j, 1 \le j \le k\}, \text{ where } \mathrm{X_p} \text{is assigned to } S^{(t)}$$

Equation 22 *k*-Means algorithm: the assignment step (step 1).

$$m_i^{(t+1)} = \frac{1}{\left| S_i^{(t)} \right|} \sum_{x_j \in S_i^{(t)}} x_j$$

Equation 23 *k*-Means algorithm: the update step (step 2).

As it is a heuristic algorithm, there is no guarantee that the algorithm will find an optimum and a solution for this particular (cluster detection) problem. Therefore, trial and error, together with a CACE4 Scaler object (at its input), are the tools used in finding a solution for a cluster problem.

The GUI offers the possibility to separate the detected clusters and the (same) clusters saved as an output stream. The detected clusters will be block aligned and copied to the output stream of the

[131] The Euclidian distance between 2 points is defined as the hypotenuse between two points in Euclidian space: $d(p,q) = d(q,p) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \ldots + (q_n - p_n)^2}$.

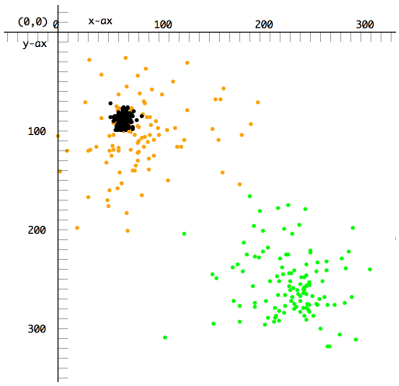
[132] Centroids are the arithmetic mean of our clusters.

CACE4 *k*-means object for further use in the CACE4 strategy chain. *k*-means, as a Hierarchical Cluster Technique of processing data, is not well suited for finding clusters of different size, while the separation of the output clusters are based on equal sized partitions of the clusters. Expectation-Maximisation (EM, see next section 5.6.2) provides a better solution for this problem. The output of both (*k*-means and EM) MIR Manipulator objects are the original data sets (x,y) but newly ordered according found categories.

5.6.2 Expectation-Maximisation (EM).

As previously stated, Expectation Maximisation (EM) as a HCT is better suited for separating overlapping clusters as displayed in Figure 52 (page 81). The algorithm[133] of EM is also composed of two steps and is mostly seen as a generalized version of the previous discussed *k*-means algorithm. First the Expectation (see Equation 25: E-step, page 82) has to be found. This is done according to principles of learning with hidden variables (Russell and Norvig 2012). In addition, EM as a HCT is an unsupervised process based on probability. In the case of CACE4, either a Poisson distribution (button 1 and 2) or a Gauss distribution[134] can be used. EM is based on the principles of finding the maximum likelihood[135] (as parameters) in a statistical model.

Figure 52 (page 81), shows clearly the possibilities with an EM cluster detection algorithm. It is capable of detecting clusters with a considerable amount of overlap; even cluster detection inside other



clusters is possible.

Figure 52 EM: Overlapping cluster detection.

[133] CACE4 makes use of coding published by Barry Fishman at: http://compgroups.net/comp.lang.lisp/lisp-and-symbolic-integration/705310.

[134] Although implemented, the Gaussian version does not work properly all the time. The second Poisson distribution version is, for now (May 2016), the same as the first one. In a future version this one will be adopted to a more specific different approach.

[135] The maximum likelihood estimation is defined as a method for finding the estimation by observations of the parameters of our model.

Just like *k*-means, the fine-tuning of the parameters (three different Poisson distribution variables are possible), need some attention and will mostly need several runs before the results are satisfactory. Equation 24 shows how the marginal likelihood is calculated according to observed data and estimated data.

$L(\theta;X) = p(X \mid \theta) = \sum_z p(X,Z \mid \theta)$ **, where** $L(\theta;X)$ **= observed data with the Maximum Likelihood Estimate. And** $\sum_z p(X,Z \mid \theta)$ **is the Maximum Likelihood Estimate (MLE).**

Equation 24 The Maximum Likelihood Estimate (MLE).

Equation 25 (page 82) shows the first step or Expectation step (or E step) of the EM algorithm. In the Expectation step, the expected value of the (log) likelihood function is calculated, and stored with the observed data set.

$Q(\theta|\theta^{(t)}) = E_{Z|X,\theta^{(t)}}\left[\log L(\theta;X,Z)\right]$ **,where X = observed data, Z = unobserved (missing values) data.**

θ **= unknown parameters.**

Equation 25 Expectation-Maximisation (EM) algorithm, the expectation phase (E step).

In the second phase of the algorithm, the Maximization step (or M step: see Equation 26) is used for finding the parameter that maximizes the expected value.

$\theta^{(t+1)} = \arg_\theta \max Q(\theta|\theta^{(t)})$ **, where** θ **= unknown parameters.**

Equation 26 Expectation-Maximisation (EM) algorithm, the maximisation phase (M step).

Although the EM algorithm uses, in its initial state, *k*-means for an estimation of an initial value for the cluster centre, it differs from *k*-means by using two further steps. The expectation or E-step is needed in order to estimate the newly calculated labels (as there are moving cluster centres, calculated according Euclidian distance calculating). In the second step or M step, the maximization of this likelihood (expectation) will be calculated by iteration, in order to find the maximum expectation. These two steps iterate through a given vector of a certain size. In the case of CACE4, a vector size of 2 - 200 entries is used for executing the separation of the cluster (x,y) number pairs into different sets. All are centred differently according to the calculation. These are also called the hidden variables involved in the process. According to *Machine Learning* (Alpaydin 2010), "In the case of mixtures,

the hidden variables are the sources of observations, namely, which observation belongs to which component." (Alpaydin 2010, p. 150).

Figure 53 page 83, shows the GUI of the algorithm. Colour coding of the detected separated clusters can be adapted to serve needs. The p1, p2 and p3 variables are used for the value of the Poisson mixture. Scaling of the input (with linear scaling) is also possible for optimizing results.

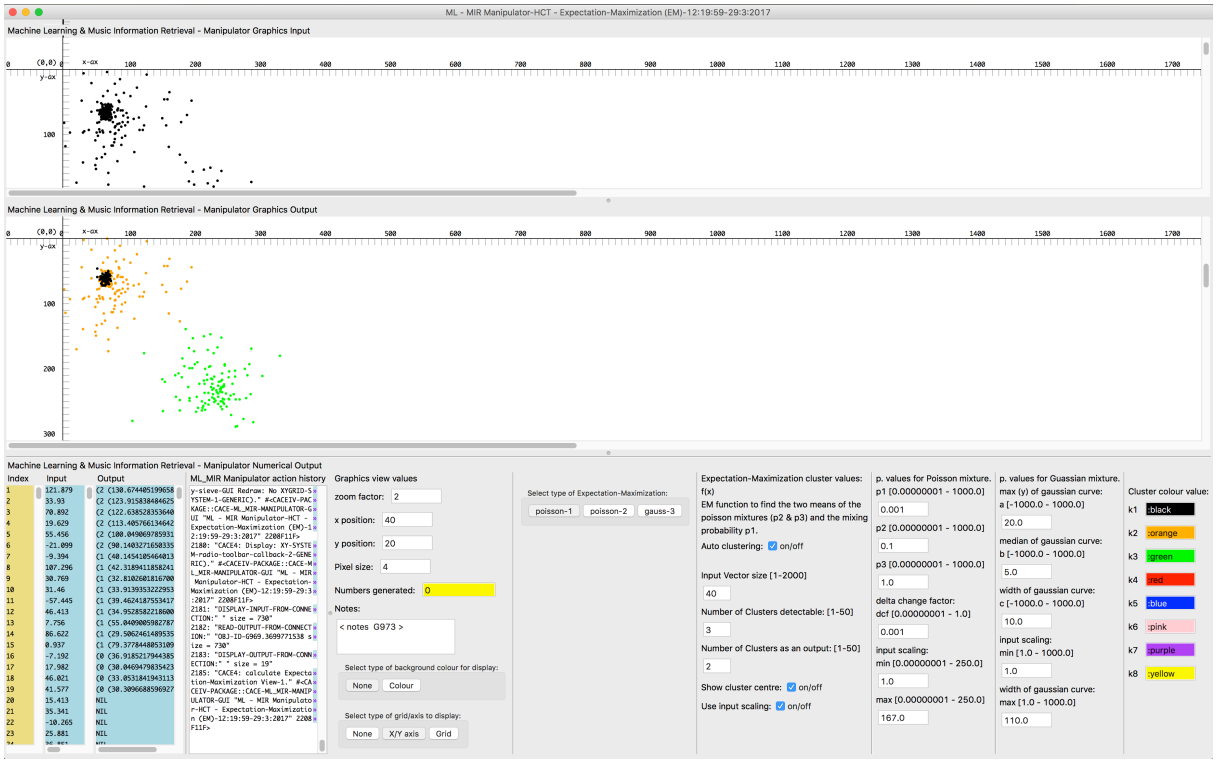


Figure 53 GUI CACE4 ML-MIR EM object.

A delta change factor is available and can be altered by the user. For good results several trials need to be calculated. Values for a Gaussian mixture can be changed as well. A few specific parameters as width and median of the Gaussian curve (bell-shape) can be changed with caution.

5.7 Other objects of CACE4.

There are two more CACE4 Processor objects to mention briefly: The Informer object and the Translator object. Both have very different functions in the CACE4 program. The task of the Informer object is to have a data-view possibility in different ways with the aid of statistical tools, without altering the input data. The Translator object is different. It acts as a translator of pre-MIDI ordered data. It displays the input stream both as numerical and MIDI values. It alters the input stream (into pre-MIDI values) when the translated stream is sent to the CACE4 Score object.

5.7.1 Informer object.

The CACE4 Informer object acts as the ‘stethoscope’ of the CACE4 Processor window. It can be attached, as can be seen at the numerous examples previous showed, to any CACE4 object in the Processor window. The only exception is the CACE4 Translator object, in this version of CACE4 (December 2015). It has the same set of statistical tools as the STAM (see section: 5.5.2, page 56) and STAPS (see section 5.5.4, page 64) CACE4 Manipulators objects: minimum - maximum, mean, median, variance, deviation, correlation, linear-regression and histogram analysis.

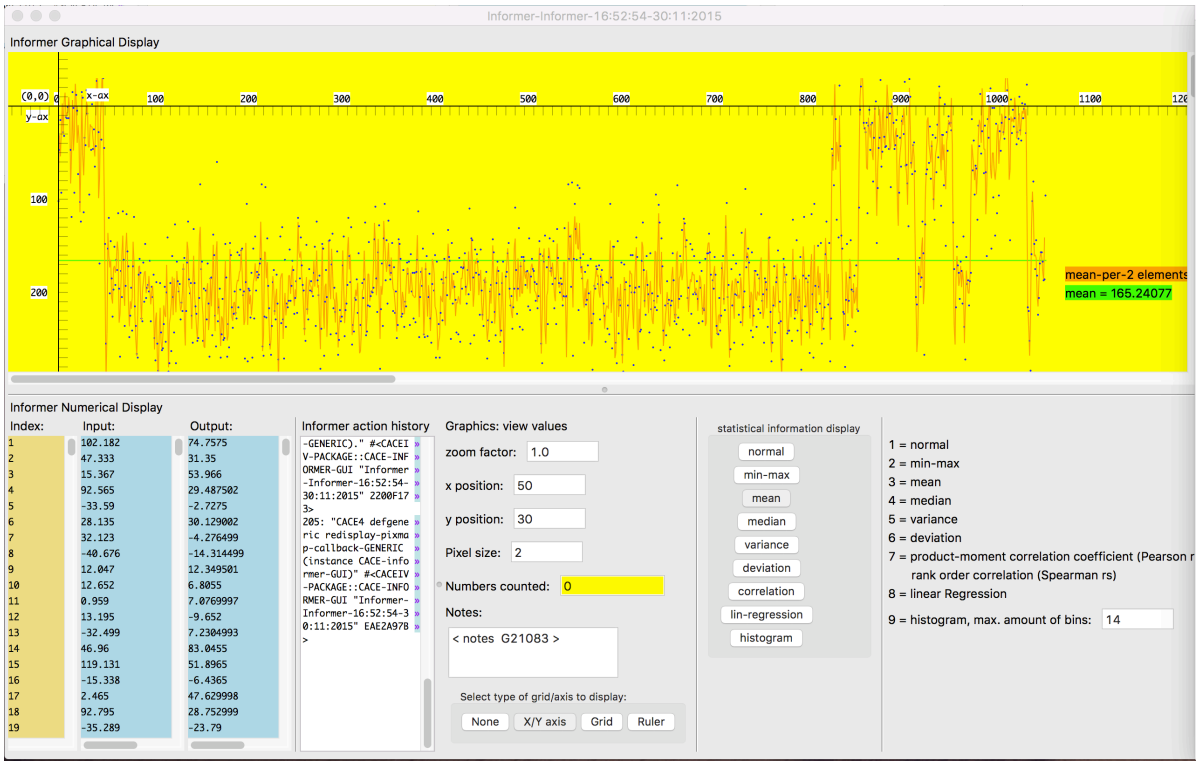


Figure 54 the CACE4 Informer object GUI.

All these statistical processes as discussed previously (see section: 5.5.3, page 58), are available for analysing the data together with a few graphical tools for altering the display appearances of the output. By design the Informer object cannot change its input and, due to its task of just showing the input list in several possible ways, does not produce any output. The only output will be displayed in its GUI. Figure 54 shows that changing the pixel size, zooming and positioning of the plotting grid and xy-axis is also possible. The position of the intersection (0, 0) of both the xy-axis and the grid display can be altered.

5.7.2 Translator object.

The CACE4 Translator object was originally designed with three different file formats in mind. Not only for transforming data to the Score object in a pre-MIDI file format, but also as Music XML (MXML) or as a file ready for LilyPond translation into a score file (pdf format).
For now the focus has been solely on the creation of SMFs (of type 1).

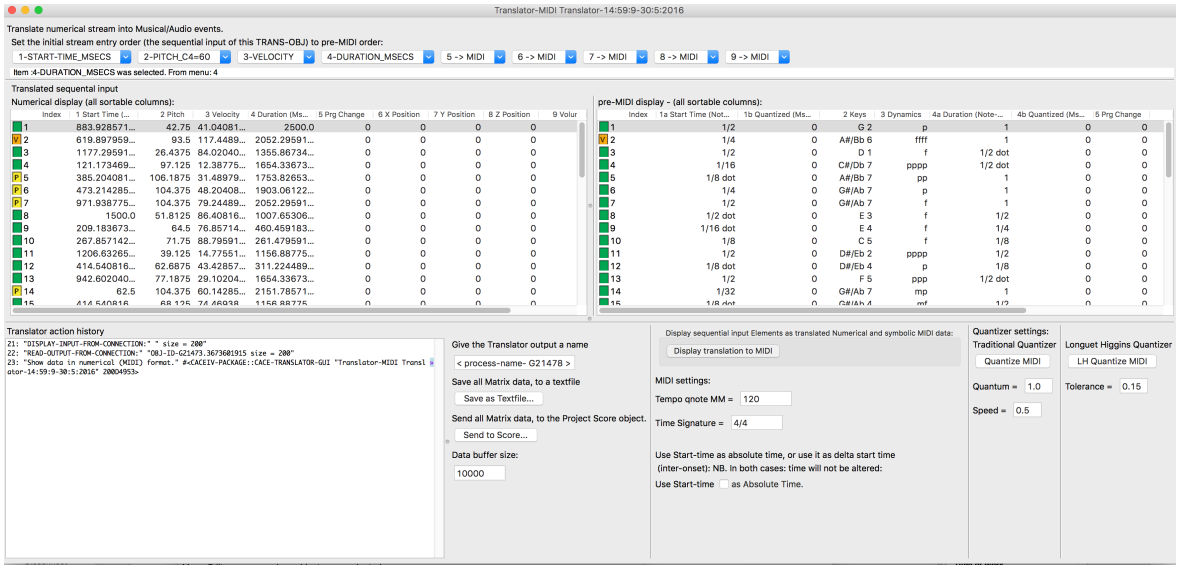


Figure 55 The CACE4 Translator GUI.

At the top of Figure 55 four selected MIDI parameters can be seen. Selectable from a drop down menu, they represent not only the MIDI value as keys, velocity, delta start time[136] and duration, but they are also used to split the input list into several (4) streams. Selecting another menu-entry from the drop down menu can alter the order in which the stream will be selected. If an already selected menu item is used, the previous one will automatically flip to neutral (0), and has to be given a new value in order to function properly[137]. This linking of the separated input-streams to a pre-MIDI format is necessary so that the CACE4 Score object will know which numbers will need a specific MIDI format in order to be able to write the stream to a SMF if requested[138]. Therefore the Translator object should always be the last CACE4 object in a chain of other CACE4 objects.
The tempo and time signature can be altered: values entered will be used to write to the SMF (header) done in the CACE4 Score object. If no changes are applied, the initial values shown in Figure 55 will

[136] Dessain and Honing in: *The Quantization Problem: traditional and Connectionist Approaches*, are referring to inter onset time as the delta start-time and it is just like the duration expressed in milliseconds (The contrary is expressed in offset time: a delta time adder). (Minsky et al. 1992, p. 449)
[137] And thus mimicking a kind of matrix in its functionality; every dimension of the input-stream can be attached to a specific pre-midi parameter.
[138] Together with the CACE4 Generator Objects the Translator Objects are the begin object and end object of a Strategy in a CACE4 Processor Object.

be used (tempo quarter note is 120 MM and time-signature is common time: 4/4).
Two more views, besides the 'neutral' (button: <Display translation to MIDI>) view, are available, as the input values can also be quantized in two different ways: by using the Micro Traditional Quantizer or the Longuet Higgens Quantizer[139]. Figure 56 (page 86), shows the same input after the Micro Traditional Quantizer has been applied (button: <Quantize MIDI>).

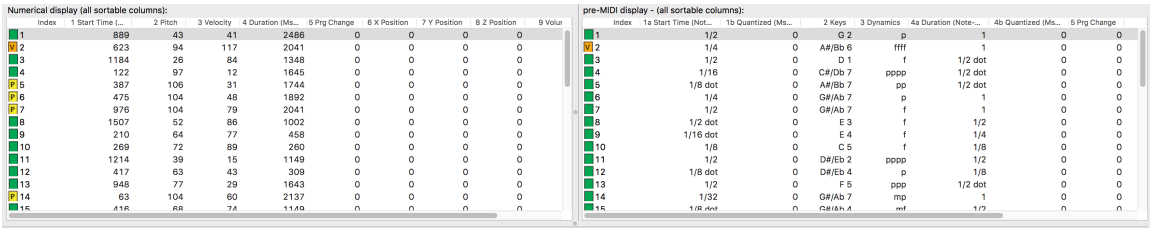


Figure 56 Output from the Micro Traditional Quantization.

This quantization algorithm has two parameters for alteration: the quantum and speed [0.1 – 1.0] can both be used. It uses this Inter-onset quantization as explained by Dessain and Honing: " This simple method rounds the inter-onset intervals of the notes to the nearest note duration on a scale containing all multiples of a smallest duration (time-grid unit or quantum)" (Minsky et al. 1992, p. 450). Therefore this quantization approach can be seen as a rounding off process, and is therefore sensitive to rounding off errors.

Figure 57 (page 86), shows the output after a quantization according the simplified version[140] of the Longuet Higgens Quantizer by Dessain and Honing (Minsky et al. 1992, p. 455), has been used (button: <LH Quantize MIDI>).

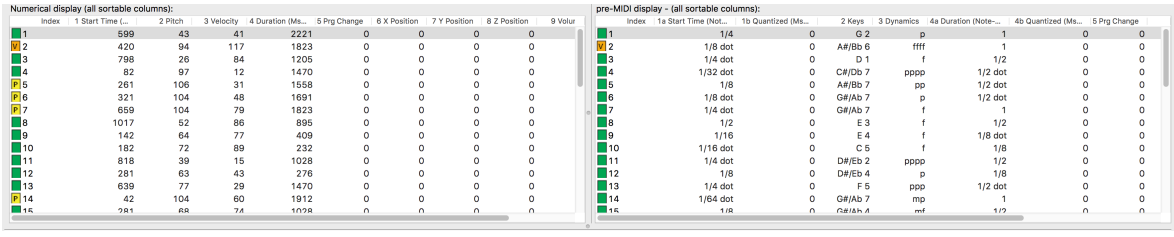


Figure 57 Output from the Longuet Higgens Quantization.

[139] The implementation in CACE4 of both models makes use of Common Lisp source code published in: *Understanding Music With AI*, Chapter 19, by Dessain and Honing (Minsky et al. 1992), The traditional Algorithm, a Micro Traditional Quantizer by Dessain and Honing: (Minsky et al. 1992, p. 454) and a 'stripped' version of the Longuet Higgins Algorithm by Dessain: (Minsky et al. 1992, p. 455).
[140] Simplified means: no tempo tracking, no metrical structure tracking, and no articulation analysis takes place. Only beat tracking takes place (Minsky et al. 1992, p. 455).

Although this version is a 'stripped down' version from the original one, it is much more complex, in its behaviour and coding as the Micro Traditional Quantizer.

Both quantization algorithms have specific approaches but also their strong limitations. Therefore Dessain and Honing have proposed a third model: the Micro Connectionist model[141] (Minsky et al. 1992, p. 459).

After all processing has been done, the user presses the button: <Send to Score…> to send the transformed output to the Score object or use <Save as Textfile…>, for saving the output in a text file format[142], which can be used for viewing the output as a text based reference (see Figure 58).

time:11:59:37 date:31:5:2016 block-size:4 math-index:800 notes:200

Index	-STime	-Pitch	-Veloc	-Durat	-Timbre	-Xpos	-Ypos	-Zpos	-Volume
1	62.5	44.082	80.000	446.4	NIL	NIL	NIL	NIL	NIL
2	64.7	38.755	80.236	223.2	NIL	NIL	NIL	NIL	NIL
3	66.9	86.694	80.472	169.6	NIL	NIL	NIL	NIL	NIL
4	69.1	31.653	80.709	330.4	NIL	NIL	NIL	NIL	NIL
5	71.3	77.816	80.945	80.4	NIL	NIL	NIL	NIL	NIL
6	73.5	63.612	81.181	71.4	NIL	NIL	NIL	NIL	NIL
7	75.7	40.531	81.417	241.1	NIL	NIL	NIL	NIL	NIL
8	77.9	108.000	81.653	71.4	NIL	NIL	NIL	NIL	NIL
9	80.1	65.388	81.889	169.6	NIL	NIL	NIL	NIL	NIL
10	82.3	60.061	82.126	169.6	NIL	NIL	NIL	NIL	NIL
11	84.5	90.245	82.362	98.2	NIL	NIL	NIL	NIL	NIL
12	86.7	47.633	82.598	401.8	NIL	NIL	NIL	NIL	NIL
13	88.9	95.571	82.834	303.6	NIL	NIL	NIL	NIL	NIL
14	91.1	42.306	83.070	482.1	NIL	NIL	NIL	NIL	NIL
15	93.3	67.163	83.307	62.5	NIL	NIL	NIL	NIL	NIL
16	95.5	29.878	83.543	464.3	NIL	NIL	NIL	NIL	NIL
17	97.7	22.776	83.779	169.6	NIL	NIL	NIL	NIL	NIL
18	99.9	86.694	84.015	116.1	NIL	NIL	NIL	NIL	NIL
19	102.1	24.551	84.251	312.5	NIL	NIL	NIL	NIL	NIL
20	104.3	86.694	84.487	321.4	NIL	NIL	NIL	NIL	NIL

Figure 58 The output of the Translator object as a text file.

All three different output formats: as raw MIDI data and twice as quantized data, can be written to a text file.

5.7.3 The CACE4 Score object.

The CACE4 Score object is literally the last object in the line of a CACE4 strategy. It acts as a collector of all basic (pre-) MIDI Music-blocks, generated by a (MIDI-) Translator object. The original design idea was to create an object with a track display, as often used by commercially available sequencer/editor Audio/MIDI programs[143].

[141] This connectionist model brings context into the quantization algorithm. It is not yet available (December 2015), but will be implemented into a future version of CACE4 as a third quantization option.

[142] As a .txt or .doc file.

[143] For example, Nuendo (Steinberg), GarageBand (by Apple), Digital Performer (by Marc of the Unicorn: MOTU) and Logic Pro X (by Apple) are just a few examples of these commercially available Audio and MIDI Sequencer applications.

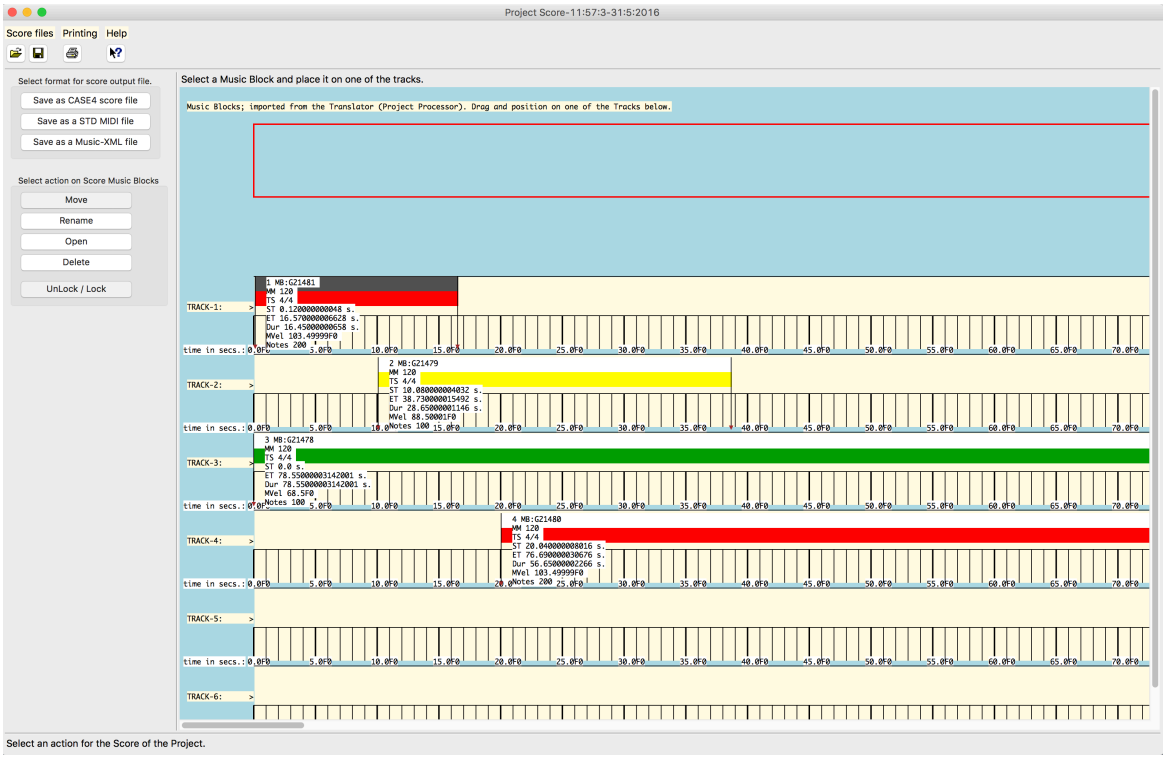


Figure 59 A CACE4 Score object.

In this version of CACE4 (December 2015), the Score object is a very simple object[144]. Its only purpose is to translate a pre-MIDI block (see Figure 59) put by the user on the first track, into a SMF. The idea behind this Score object was to create a workspace for placing the Music-blocks in a (random) sequential order, and for future use[145]: to be able to consolidate them into a single or multiple Tracks.

By selecting button <Save as a STD MIDI file>, the user saves the output of this (a single music-block) to a SMF of format 1. The previously ordered pre-MIDI stream will now be typecast to a byte-stream and saved to a file (including writing a header with all – according to SMF format – necessary track-, including track headers, and all other file information). When the user uses the <Send to Score> button in the CACE4 Translator object, the pre-MIDI data will appear as a Music-block in the top corner of the Score object Window. The Music-blocks show some additional information as can be seen in Figure 59. A gray-scale colour provides information about density (equals spreading of the total amount of notes over the time of the Music-block). The darker the gray colour appears, the denser the Music-block will be. The lower part of a Music-block displays a band of colour. It represents a scale (from green to red) of the mean velocity of all entries (in the Music-block). The redder the colour appears the higher the velocity values will be.

[144] Due to time constraints this object acts now only as a (visual) storing place for the MIDI data before transforming them into a SMF. Future development will focus on this object to give it more features for handling these MIDI-blocks.

[145] This will include a second file format: SMF format 2, will multiple Tracks and separated Tempo Tracks.

Extra text, at the left of every Music-block, gives information about Track-number, unique ID – number, Tempo, Time-signature, Start-time, End-time, Duration, the mean Velocity calculated of all entries and the number of Notes of this block. For now (December 2015) they have no other function besides displaying this information[146], and are all omitted for creating a SMF.

Although the functionality of the Score object is comparable to a Sequencer program, this is not its main objective. All functionality provided by an off the shelf, commercially available Sequencer program, is unavailable in this version of CACE4. This was also not the original intension: CACE4 should act as an intermediary processor between data files as input and, for now, MIDI files as output. In this way it is possible to use it in a Sequencer or a Notation Program, for extra editing.

Future development will focus on monitoring the output generated by listening an easy 'glueing' functions for consolidating Tracks and implementing a real Multi-track version with a SMF format 2 as output.

5.8 Building a strategy with CACE4 objects.

Setting up a strategy for reaching a certain predefined goal is the core application of CACE4. It really depends on the results one is looking for and of the complexity of the problem itself, what the calculated outcome of the process is. Taking time to experiment with the order of the objects in the Process-window and scaling it to the right proportions can help as well. In addition, deleting certain items in the data stream, (e.g. x=x+1 values) can clean up the results. The results will differ also if one experiments when entering parameter values from the Generator objects. Displaying and plotting the results, doing some (only visual output) statistics with the Informer object will help in understanding the results. It must be remembered that not all the Information Retrieval (*k*-means and EM) or A.I. (ART2) objects can solve the problems, because they are not well suited to the specific data problem[147]. So it is best to initially devise small experimental setups and streamline the strategy process according to the results obtained.

5.8.1 About the Art of designing a CACE4 strategy.

This paragraph illustrates a typical working session with the CACE4 program. After initial ideas and considerations about the kind of composition one would like to work on (e.g. instrumentation,

[146] A future version should not only be capable of using the SMF format 2 (= Multiple Tracks format), but also provide the user with an adequate listening possibility of the tracks before creating a SMF.
[147] *K*-means is used for more widely spaced cluster detection. The EM algorithm can be used in case the clusters are more narrowly spaced. An ART2 NN uses vectors for qualifying however and is as such, not directly used for detecting clusters but for grouping vectors (with a scalable size) into distinct categories.

duration, and other compositional ideas[148]), a new project window is created in the CACE4 application. After adding a CACE4 Processor object, and opening it, objects are selected and arranged in order to be used in the chosen process. See Figure 60, page 90 as an example of a lay-out and connections of objects.

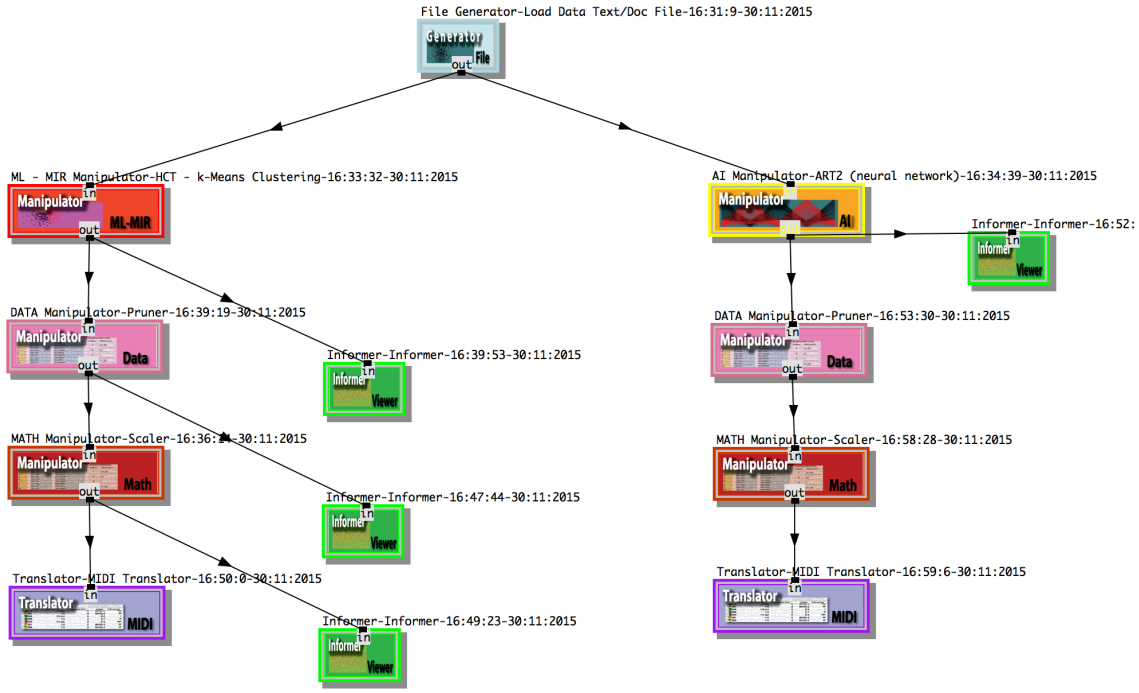


Figure 60 Example of a working session with several CACE4 objects in the Processor window.

The first object in the process-chain should be a Generator object. Therefore a CACE4 FILE Generator object of a text file type is selected and a choice is made (by using a dialog window) for the desired file with data. In this case the data is grouped in (x,y) pairs: ready for scatter plotting.

Figure 60 shows two Manipulator objects attached for transforming the data, in this case: a *k*-means object and an ART2 object. Both are attached to the CACE4 FILE Generator object. The ART2 object is used in a second strategy-chain. The CACE4 objects are not connected and therefore the outputs are not interfering[149] with each other and generate two separated (pre-) MIDI streams of the split processes involved. All four green informer objects are used in order to have a visual/statistical check of the output if necessary and will have no direct influence in the final output.

[148] The list of compositional ideas can be rather long and depends on how the user likes to achieve their compositional goal. In this example the goal was to achieve a strategy capable of comparing the output generated by the two separated streams. (In this case a *k*-means cluster Object versus an ART2 Neural Network Object)
[149] If one likes to intermingle or add the results, a Merger Object must be used and the two streams connected somewhere before a Translator Object.

Next the Pruner object (see Figure 60) is used in both strategy processes. It takes care to remove unnecessary or unwanted numbers in our stream. In this case only the first cluster of k-means (n=50) and 1 large output category of the ART2 object (n=700) are used (see Appendix 1.1 and 1.2).

The last object used in the Strategy chain is the Scaler object, this being necessary before the Translator objects can be used (see Figure 61, page 91). It should always be in the chain were MIDI output is desired, in order to be able to scale the output to the range of MIDI (Note numbers and Velocity) values. In this case the following (MIDI) ranges are used: 32.5 – 1500, 21 -108, 10 -127, 32.5 – 1500 1/64th note. For scaling timing it is necessary to scale in order to create MIDI time scaled values (Delta-start-time and duration, both in milliseconds). Values are typically somewhere between 32.5 milliseconds (= 1/64 th note) for a minimum and 1500 as a maximum. For entering duration values, comparable ranges can be used. After entering the desired data ranges, press <linear> for scaling.

Appendix 1.1 and 1.2 shows a typical working session involving several open windows for displaying and altering the data stream. By using the Informer objects and 'connecting' them at different object outputs in the chain of strategy, the intermittent and final result can be looked at in different ways. The final CACE4 object in the process chain is always a Translator object; in this case MIDI translation is used.

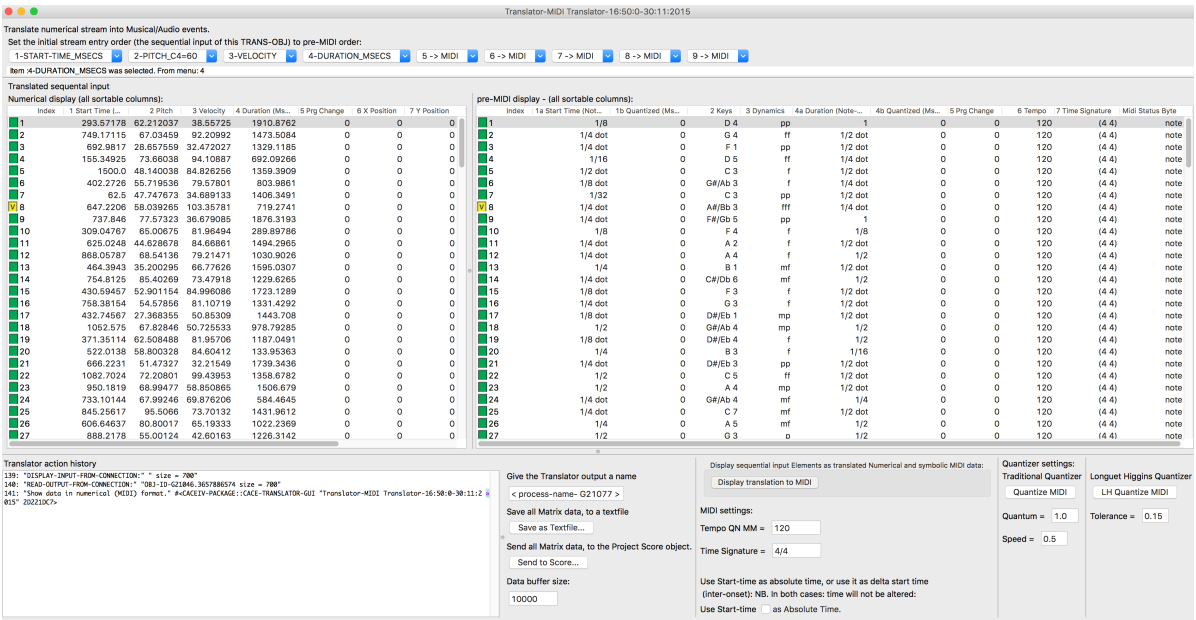


Figure 61 Shows the view of the Translator object with the first 27 entries of the translated (to pre-MIDI) output.

After selecting the appropriate properties: delta start time, key, velocity and duration (all of these items can be found in the top of the Translator window as pull down menus), the values now have to be matched to the according index numbers. The first element in the input list will be used as delta

start time in milliseconds. The second one will be used as a MIDI key value. The third one is used as a value for MIDI Velocity. The fourth and last one is the Duration in milliseconds.

By attaching a Translator object to the end of this process chain, everything can be translated to the Score object where the final results can be saved as a SMF. When saved in this file format it can be used in most other Music Editor Software (e.g. Sequencers) for further use/editing.

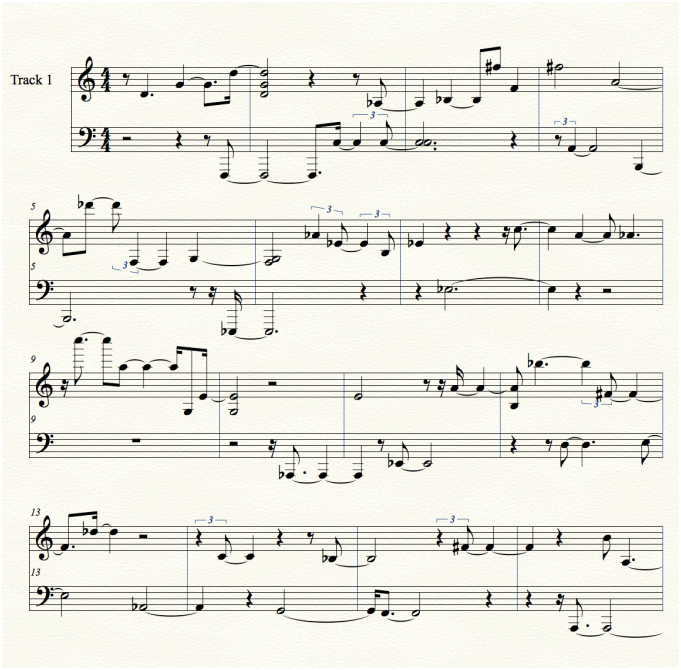


Figure 62 Screen shot of Finale™ display of the SMF generated in CACE4.
This is the output of the ART2 NN object. See Figure 61 for its CACE4 Translator object display. No additional editing has been done.

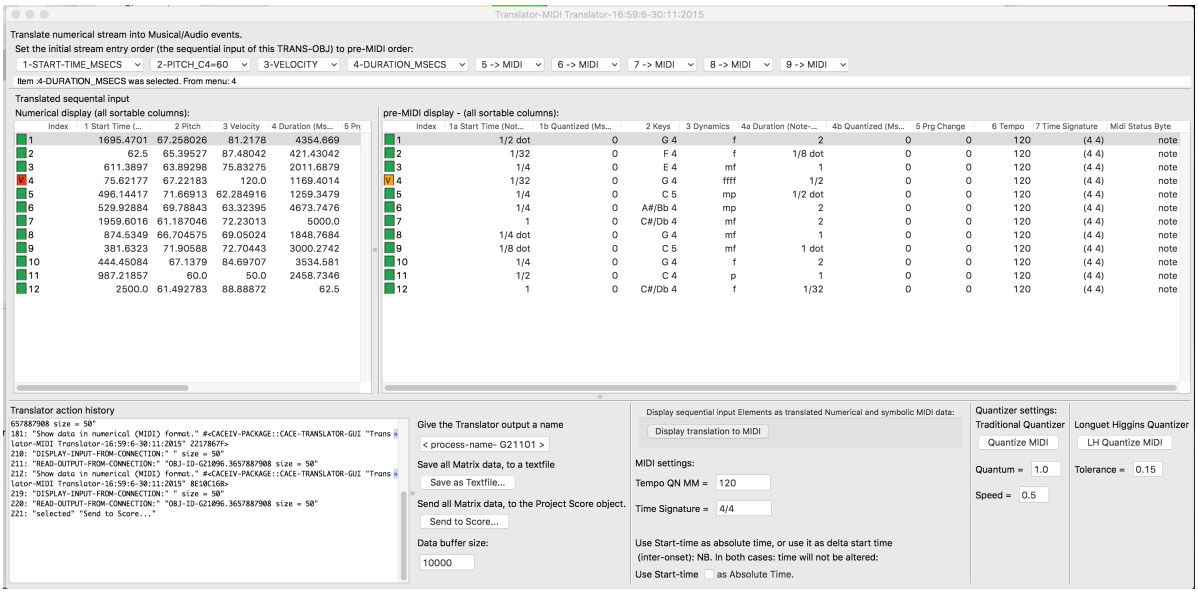


Figure 63 Shows the view of the Translator object with only 12 entries used for translation to pre-MIDI data.

The output displayed in Figure 62 (page 92) and Figure 64 (page 93), is a view of the same output but now in the Finale™ application. Only quantization 1/32 notes have been applied when the generated SMF was first opened and read. This is the same quantization as has been applied in the Scaler object as the smallest possible delta start-time and duration: 62.5 milliseconds (with a MM=120 tempo and 4/4 time signature, see both Appendix 1.3 and 1.4.

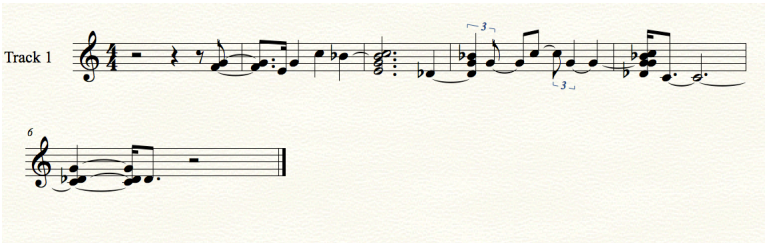


Figure 64 Screen shot of Finale™ display of the SMF generated in CACE4.
This is the output of the k-means object output. See Figure 63 for its CACE4 Translator object display. No additional editing has been done.

5.8.2 How does CACE4 compare with other computer composition environments?

In order to make a valid comparison between CACE4 and other Computer Composition Programs it is necessary to find shared criteria rooted at a higher or so called meta-level.
These higher-level abstractions are necessary for finding shared definitions and matching ways of processing of the data [150]. Certain program design issues must be taken into account as well. For example: is the computer composition environment command line based, or is it an application with, to a certain extent a GUI for interaction by a user? Also the major software design choice: is it based on a Language Model, or does it use another approach as Model? In order to be able to make any kind of comparison possible the list of criteria needs to be reduced to a few essential points:

- Usability, a very subjective one: command line provides more flexibility then a GUI, but this comes with a cost as the user needs more time to become acquainted with all the possibilities offered.
- Extendability: on which level (for example, the programmer or the user) is the program extendable?
- How is the GUI defined? Does this involve the use of graphics, terminal or the use of both?
- Core foundation: is it based on Music as a language[151] or as a 'neutral' data stream? Or is there

[150] Otherwise too many details have to be taken into consideration and comparisons made far beyond the scope of this thesis.
[151] Although defining Music as a Language has its advantages, it also put a restriction on certain details of the program. If it is not defined (inside the music language involved) it cannot be observed or otherwise used for doing (music) calculus. These restrictions can be easily avoided if there is a possibility in the application involved for doing mathematical calculus. It is still outside the domain of music as a language, but for now it is knowable and therefore can be used.

another Model used in the design of the application?
The following algorithmic composition programs place differing emphasis on these criteria.

AthenaCL[152] is an open design for computer-aided algorithmic music composition (Ariza 2005). This music composition program by Christopher Ariza is command-line based, although certain parameters can be displayed in separate graphical displays windows (as Event-sequences).

The athenaCL Sytem offers an open-source set of objects written in python. It is command-line based and therefore is scriptable and can be embedded into other platforms. It comes with advanced Libraries for all sorts of Musical Modelling E.g. Microtonal tools are amongst them. The musical material obtained can be outputted in several formats: CSound, SuperCollider, Pure Data, MIDI, audio files, XML and text formats. The program is easily extendible, also on the level of the user. Working with a command line[153] offers greater ease in letting the user work with pre-defined template documents. This speeds up the process significantly. The use of a GUI is rather restricted to a few 'output' objects for plotting purposes.
As a software design principle, music is treated as a language and therefore restricts itself to pattern creation. Music can therefore only be defined on the level of different kinds of patterns, all related to each other in a hierarchical relationship.

ACToolbox[154] is defined by Paul Berg as a collection of (software) tools for doing algorithmic composition. It is based on the idea of defining all processes as objects. This offers the advantage that every process can be treated independently from each other[155] thus offering the user ease of extendibility and flexibility in the process of defining new objects (and their processes).

Generators here, as also in CACE4, are doing the job of creating musical material. Transformer[156] objects then take over in order to transform the generated input. Input and output can also be used from several file formats available: MIDI, OSC, FOMUS, CSound and even output for use with the Kyma Capybara can be obtained. The integration of a LISP interpreter (from LispWorks, version 6.1.0) and therefore being able to use a Listener, offers tremendous flexibility. Users can create their own objects on source code level. This is the maximum flexibility any program can provide and therefore has no restrictions on extendibility. It comes with a small cost, as users need a little time to get acquainted with the software. Good examples do exist, however.

[152] AthenaCl can be found at: https://pypi.python.org/pypi/athenaCL/2.0.0a15
[153] From the perspective of the user: there is great flexibility in using this command line, although a certain 'learning-curve' has to be taken into account.
[154] The ACToolBox application can be downloaded from URL: http://www.actoolbox.net/download/
[155] This is comparable the way CACE4 was designed.
[156] This is different from CACE4: Transformer Objects are equivalent with the functionality of the Manipulator Objects, although Manipulator Objects do not always transform the input. Mostly the original input is sorted according a certain process without altering the original input values (e.g k-means, EM, ART2 etc.)

Musical material can be grouped in musical chunks, as notes or as groups of notes. Then methods can be used for altering the output, all according to OOP techniques[157]. With this embedment in a LISP Programming Environment, there are no real restrictions and limitations to the way in which it can be used.

Graphical Realtime Algorithmic Composition Environment (Grace)[158] & Common Music (3.9.0) has a long development History. Originally started in the 1980's, its core software foundation consists of the Common Music[159] packages and has been further developed into Grace by Heinrich Taube (et al.). As Taube states in his book, *Notes from the Meta-level* : "It is primarily intended for student composers interested in learning how computation can provide them with a new paradigm for musical composition." (Taube 2004, p. IX) It offers all the flexibility of using an interpreted language (Listener). Also GUI objects exist in order to help the user to obtain a graphical representation of certain output. Common Music is, just like ACToolBox and AthenaCL, embedded in a LISP environment. This offers the use of an interpreter and a Listener for the user. As Taube states in *Notes from the Metalevel*: "The system was originally written in Common LISP but now includes both Scheme and Common LISP bindings …" (Taube 2004, p. 10). This gives it the maximum extendability for the user; own source code can be applied, even in several LISP dialects[160]. This intermingling of the LISP programming language environment and composition tools is a strong concept.

CACE4 presents itself as a computer composition environment with a strong focus on the use of statistics and IR as single blocks. This approach is directly reflected in the overall design of the GUI. The user is encouraged to play around with the specific parameters for generating the desired output. There is no underlying music-language construction available at present (May 2016). Just by changing the order of connections in the processor window of the current project, the output will have different output. The focus of CACE4 is the use of statistical tools and techniques, together with processes from the field of Artificial Intelligence and Information Retrieval. All tools are focused on visualising data in two different manners: numerical and graphical (mostly in scatter plot format). The techniques involved in the process will be treated as a closed box where the only way of influencing the process is by using the given parameters. Alterations on the stream can be made, by using the process parameters involved. By attaching the Translator object to the end of the process chain, everything can be

[157] Even a 'Kill' menu option can be found, once in a while necessary for stopping infinite loops.
[158] Version: Grace 2016 and Common Music 3.9.0. May 2016. And can be freely downloaded from Apple's AppStore.
[159] Grace – consists also of several other packages: JUCE v3.0.3 (c) 2016 Julian Storer, S7 Scheme 3.5 (17-Feb-14), Sndlib 23 (c) 2016 William Schottstaedt and oscpack 1.1.0 (c) 2016 Ross Bencina.
[160] An overview of all implementations can be found in 'Notes from the metalevel': (Taube 2004, p. 11).

translated to the Score object. The final results can be saved as a SMF. When saved in this file format it can be used in other Music Software for further use/editing.

But how does CACE4 function, compared to the other three Composition programs?
CACE4 is focused on the user by offering a GUI. Menus, Windows and other GUI Controllers offer their advantages: no steep learning curves are involved. They also have their limitations: a lack of extendibility for the user is one of the major differences compared with the other three composition programs[161]. Although this was originally not part of the design, it has to be taken into consideration as a future feature of the CACE4 program. A real-time playback/listening option is the other major consideration that has to be taken into account with future versions of CACE4.
The other large conceptual difference is that CACE4 is not designed for being an 'all-round' algorithmic music composition program, for now[162]. Its initial goal was to be able perform data analysis in the area of AI and (M)IR and to apply these tools in the domain of music. Therefore the functionality was restricted and strongly relies on the Graphical aspects of a GUI (pixel plotting and the use of colour). By incorporating a LISP interpreter into the CACE4 application however, it would make it much more suitable for use by students, not only for exploring (M)IR techniques, but also through programming (in the Listener) LISP code as well.

[161] This lack of extendibility will be addressed in the final conclusion as well. See Chapter 7 for more details. And has to be taken into consideration for future versions of CACE4.
[162] Although the usability for creating interesting musical material is still relevant: future versions of CACE4 will focus on this topic, by developing new CACE4 Objects. They will focus on different, more music related aspects.

Chapter 6

Analysis of four Compositions created with the aid of CACE4

6.1 Introduction.

During the process of working on CACE4, the possibility for creating a composition for the first time was with a preliminary version[163] of the software in 2013. This resulted in *Argos Pansonos* a composition for Piano and Computer (Max/MSP).

A year later (September-December 2014, at the Zentrum für Kunst und Medientechnologie (ZKM), Karlsruhe, Germany) with a more evolved version of CACE4, I started working on the second composition *Zwicky's Box* for ensemble and computer. The last composition is *Scope* for ceramic (or metal) tiles, 2 string drums and computer, and has been made with one of the latest versions of CACE4[164].

During this period I also worked on *MMM Transformations in pink*. This is part of a larger on-going project with the Dutch artist Willem Willemse and CACE4 has been used here to generate STD MIDI files for the four compositions.

6.1.1 Artistic reflections.

As stated in section 2.2 (page 10), Bense's philosophy of generative aesthetics gives a clear description for describing a synthesized aesthetic product and the terms required to meet it. He states that every new piece of Art should be preceded by an aesthetic analysis based on structures described in mathematical terms. According to Bense[165], there are four different aesthetic structures: semiotic classifications, metrical, statistical and topological, which we apply to reflect onto the design, workings and artistic output of CACE4.

Bense originally categorized music in the group of semiotics, but by replacing certain musical composition processes by applying mathematics, music can be ordered into the other three categories as well.

According to the first of the four aesthetic structures, the semiotic classification for both CACE4, the program (written in an artificial programming language) and the artistic output, Music (as a language

[163] This was not a standalone version of the application CACE4, but rather a development version embedded in the LispWorks IDE as a separate menu item.

[164] CACE4: version 0.56.07.467 – august 2015.

[165] "At the moment there are four different ways of making abstract descriptions of aesthetic states (distributions or configurations), which can be used to produce aesthetic structures–the semiotic (employing classifications) and the metrical, statistical and topological methods–the latter three are numerically or geometrically orientated." (Reichardt, 1971, p. 4)

of symbols and signs), are directly reflecting the definition of semiotics[166] as the science of symbols and their intrinsic meaning. By not using an artificial music language as the core of the design of CACE4 however, the other three artistic structures mentioned apply as well. The definitions of metrical, statistical and topological artistic structures are reflected in the mathematics involved. These domains of mathematics: statistics and topology, are applied as functions in CACE4 objects and their direct application in a CACE4 strategy. The strategy, as created in a CACE4 Processor object, needs to reflect artistic ideas about the creation of an artistic product (in this case a music composition), symbolises the artistic process with boxes (of smaller, modular processes) and arrow-lines for connection and direction of the flow of the, in this case, numerical output, as part of the artistic process.

The process of creating musical material used for all of the compositions is a reflection of the modular strategy elaborated in a CACE4 Processor object window. As a prerequisite: structuring and characterising the musical composition into separate parts and hence defining the structure for the composition as well. Each part has to be well defined with specific musical constraints and characteristics. As can be observed in the three compositions for musician(s) and computer[167], each of them is divided into sections that have a distinct musical impression and an artistic description of a process. These sections are mostly constructed using smaller parts containing contrasting musical content and serve to bind these larger sections creating the notion of starting a new direction and flow in the music, in a way that could loosely be compared to a cadenza. Some examples of this can be seen in the following excerpts from the scores: *Argos Pansonos* and *Zwicky's Box* (see Appendix 5: portfolio compositions, for the scores).

For example, in the composition *Argos Pansonos* for piano and computer, Appendix 5 (portfolio compositions) rehearsal mark B, bars 28 – 31 (see score page 4); rehearsal mark D, bars 46 – 49 (see score page 5) and F, bars 77 – 79 (see score page 6); show larger sections with these smaller parts in between. They can be seen as cadenzas connecting the larger sections with smaller, contrasting themes.

Other techniques involving smaller parts, but now used as building blocks for larger parts, can be seen in the composition for sextet and computer: *Zwicky's box* Part 3, rehearsal mark H, Bars 101 – 103 (page 19); I, Bars 105-109 (page 20), J, Bars 111-115 (page 21) and K, Bars 117-123 (page 22). Here chords are used and the number of bars increases in time, from 3 bars to 4 and ending with 7 bars. This creates a slowing effect on the music.

[166] Semiotics (or Semiology) is defined as the study of signs and their meaning, The American scientist Charles Sanders Pierce is widely recognized as founding scientist (see https://www.britannica.com/science/semiotics).
[167] See Appendix 5: the following compositions: *Argos Pansonos* (piano and computer), *Zwicky's box* (ensemble and computer) and *Scope* (percussion and computer).

The use of smaller parts is not restricted to vertical use, with a strong vertical connection between the instruments such as the as chords as be seen in the previous examples, but also more as independent, horizontal layers, as can be observed in *Zwicky's box*: Part 1, D, bar 55 – 69 (pages 12 - 13) and Part 5, M, bars 164 – 191 (pages 28 – 31).

Other uses of these blocks where to use the output of a calculation for different staves. Several instruments were calculated at once, for example in Part E, bar 71 to Part G, bar 96 (pages 14 – 16). All four compositional techniques were used in order to create a more dynamic development of the musical material as opposed to a more static music.

Since one of the ideas in this thesis (see chapter 1, page 1), was to find structure and connections between sets of data and apply them in the domain of music, CACE4 was developed in the search for these properties. Found data (as .cvs files on the internet, see section 6.2.1, page 102), was used for analysing and transforming data to map them onto musical properties such as pitch and dynamics, delta start time and duration. In particular, the application of the same function for obtaining values for pitch, dynamics, delta start time, rhythm and duration demonstrates a link to the ideas and practice of serialism in music. Since the intrinsic value of the data is given by the applied function on the found data sets, it is, according function definition, (mostly) gradually, changing over time. This whole process of transforming data into music parts with distinct musical characteristics should reflect these properties of the analyzed data.

This is not the only artistic consideration and technique involved by creating the compositions. By implementing the CACE4 STAM object (see section 5.5.2, page 56) and the CACE4 STAPS object (see section 5.5.4, page 64), this processing is used for creating several, more closely related, variations of output. The different outputs of these generated blocks of data can be obtained by fine-tuning parameters inside the objects used in the CACE4 strategy. This method of working is comparable with the compositional technique of thematic development (variations) and is, as such, well known in music.

Defining the compositional process as a feedback or looped process for obtaining output, the plasticity of the material generated (or found as data on the internet), mimics the methods of the Visual Arts (sculpting and moulding) by exploring and changing strategies, in combining different CACE4 objects. This approach can be used to create series of different but closely related sequences and mimics the creation of series in Visual Arts. Chisels, hammers and palette knives have been replaced with mathematical equations for 'sculpting' (music) data. This preoccupation with structures as lines

and surfaces is a typical Dutch[168] subject in the visual Arts. Mostly known by works (paintings) from Jan van Doesburg, Piet Mondriaan[169] and Jan Schoonhoven[170] (Paintings and Plastic Art). They researched harmony in the visual arts by means of logic. Geometrical patterns as lines and surfaces combined with the use of primary colours was their starting point of expression. This use of geometry can also be observed in the computer graphics of Willem Willemse, who was inspired by the works of Jan Schoonhoven.

By classifying CACE4 to the metrical, statistical and topological methods proposed by Bense, (pre-) musical output[171] created by CACE4 can be seen as output in strict (logical) mathematical constructs.

Although different tools are used in the process of creation in both the visual arts as music, their intrinsic function: to add more order and eliminating chaos and reducing complexity by applying tools, artistic rules and restrictions in order to reach an artistic goal, is comparable. In CACE4, all is done within a computer program and this therefore restricts the artistic rules applied to logical constructs, based on algorithms. By processing the data with the aid of mathematics, CACE4 and its artistic output reflects Bense's philosophy on the artistic measurement and is subject to Birkhoff's equation (see equation 1, page 9).

6.2.1 'Argos Pansonus' (or the meaning of *k*-means).

A composition for piano, computer and a 2D/3D Sound System, duration 12 minutes – 2013/15, *Argos Pansonos*[172] is the first (algorithmic) composition I have created with the newly written composition program CACE4 -Computer Assisted Composition Environment 4[173]. As I often do in my compositions, I take a mathematical approach for solving a specific problem and use it as a certain kind of idea and 'theme' in my compositions. *Argos* is an algorithmic composition where the central algorithm is the k-means statistical algorithm, from the family of Hierarchical Cluster Techniques. By making use of several sources of data and selecting and slightly altering (scaling) them, I was able to transcribe the result of the mathematical output into a score, playable by a human being.

---

[168] The author has Dutch nationality (The Netherlands).
[169] These are just two names of the well-known Art movement of Neo-plasticsm (or "De Stijl", in Dutch), of the late 1910's and 1920's. More details can be found at: http://www.tate.org.uk/learn/online-resources/glossary/n/neo-plasticism
[170] Jan Schoonhoven does not belong to the group of Neoplasticism, but can be seen as elaborating on this idea of minimalism in the Visual Arts. More about Jan Schoonhoven and his works of Art can be found at the URL: http://www.tate.org.uk/art/artists/jan-schoonhoven-1907
[171] Created as a SMF, which can be seen as a contextual sequence (stream) of symbolic signs in only 0's and 1's.
[172] The title is derived from the old Greek saga of Argos[172] Panoptus, the mythical figure with many eyes, as an icon of vigilance and wakefulness. I transformed his nickname: the Greek word panoptus (= pan means surrounding and 'optus' means seeing) as an analogy into 'pansonus' ('surround' hearing), which has a strong link to the 3D surround sound system which will be used for the live performance of the composition (a 3D Ambisonic loudspeaker dome).
[173] CACE4 is the fourth version of a composition program I wrote in LISP, some parts of which date originally from 1993 as the CAC I program.

As a piano composition accompanied with live use of a computer, *Argos Pansonos* moves between passages based more on harmonics and timbre alterations by the computer, which acts as an invisible second 'player' and more rhythmical 'streams', as a single movement. By using close microphone techniques (and by making use of a PA-system[174] in a live-performance situation), we can utilize 'tiny' and soft sounds played inside the piano. This, in combination with DSP, alters and enriches the sound palette I use as a composer. The inside of the piano is not the only sound source used. As a contrast and musical counterpoint, more rhythmically developed patterns of notes are used as melodic lines, acting as 'cadenza's' and interludes in the composition.

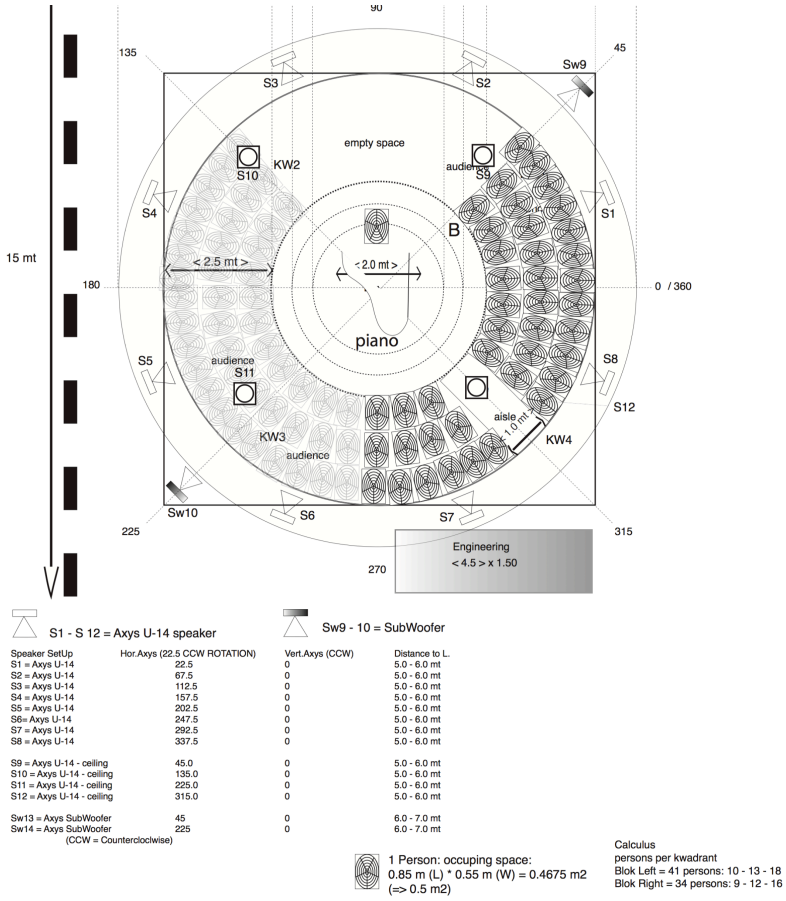


Figure 65 A Top view of a 3D Audiospace design for *Argos Pansonos*.

The starting point was to create a composition which would use the inside of the piano as a rich sound source, which makes it a good source for the DSP Max/MSP patches as well and alternate it with normal use of the keyboard which would act as a counterpart in this piano composition of fifteen minutes. The composition can be played in a 2D (horizontal circle) or 3D (as a dome) speaker setting

[174] A PA-system is short for a Public Address system. It is a general term used for a sound amplification system consisting of loudspeakers together with their amplification and generally incorporating a sound-mixing console (sound mixer for short).

(see Figure 65, page 101).

After finishing the implementation of the k-means algorithm in the CACE4 program, I looked at the internet for data in a (x,y) - 2 dimensional format and found some useable text based Excel data sheets at the site of the British Geological Survey. It concerned data from gravitational measurements from all over the UK[175]. I used two columns in the Excel data sheet: free_air_an and Bouguer_an, representing both different measurements (as can be seen at the right hand side of Figure 66, below).

SURVEY_AREA	STATION_ID	STATION_CODE	LATITUDE	LONGITUDE	GRID_EAST	GRID_NORTH	STATION_ELEV	ELEVATION_UNIT	BOUGUER_DENS	BASE_CODE	OBSERVED_GRAV	FREE_AIR_AN	ITOT_TC	ITOT_TCZ	TOT_TC	TOT_TCZ	BOUGUER_AN	SOURCE_COI
BRISTOL CHANNEL	___00_09	0	51.411339	-2.92667	335560	168490	-5	0	2.7	-41	1186.41	-9.91	0	Q	0.01	H	-9.33	-80
BRISTOL CHANNEL	___00103	0	51.216999	-3.0835	324330	147030	-5	0	2.7	-41	1178.34	-0.8	0	Q	0.01	H	-0.22	-80
BRISTOL CHANNEL	___00_79	0	51.566502	-2.68667	352410	185560	-5	0	2.7	-41	1201.31	-8.71	0	Q	0.01	H	-8.13	-80
BRISTOL CHANNEL	___00102	0	51.227501	-3.03533	327710	148150	-5	0	2.7	-41	1177.52	-2.55	0	Q	0.01	H	-1.97	-80
BRISTOL CHANNEL	___00_77	0	51.567169	-2.70917	350850	185650	-5	0	2.7	-41	1201.45	-8.63	0	Q	0.01	H	-8.05	-80
BRISTOL CHANNEL	___00_49	0	51.49567	-2.8705	339580	177820	-5	0	2.7	-41	1193.09	-10.68	0	Q	0.01	H	-10.1	-80
BRISTOL CHANNEL	___00_11	0	51.472672	-2.88783	338340	175280	-5	0	2.7	-41	1191.6	-10.14	0	Q	0.01	H	-9.56	-80
BRISTOL CHANNEL	___00_26	0	51.425671	-3.12683	321670	170280	-5	0	2.7	-41	1191.44	-6.15	0	Q	0.01	H	-5.57	-80
BRISTOL CHANNEL	___00_66	0	51.516499	-2.998	330760	180250	-5	0	2.7	-41	1196.95	-8.66	0	Q	0.01	H	-8.08	-80
BRISTOL CHANNEL	___00_04	0	51.4165	-2.9095	336760	169050	-5	0	2.7	-41	1187.52	-9.26	0	Q	0.01	H	-8.68	-80
BRISTOL CHANNEL	___00_42	0	51.522499	-2.875	339300	180810	-5	0	2.7	-41	1194.71	-11.43	0	Q	0.01	H	-10.85	-80
BRISTOL CHANNEL	___00_02	0	51.514	-2.73383	349080	179760	-5	0	2.7	-41	1197.32	-8.07	0	Q	0.01	H	-7.49	-80
BRISTOL CHANNEL	___00_96	0	51.219669	-3.09367	323630	147340	-5	0	2.7	-41	1179.45	0.07	0	Q	0.01	H	0.65	-80
BRISTOL CHANNEL	___00_55	0	51.485668	-2.96233	333190	176790	-5	0	2.7	-41	1192.71	-10.18	0	Q	0.01	H	-9.6	-80
BRISTOL CHANNEL	___00_97	0	51.223499	-3.08183	324460	147750	-5	0	2.7	-41	1179.35	-0.37	0	Q	0.01	H	0.21	-80
BRISTOL CHANNEL	___00_28	0	51.414669	-3.0995	323550	169030	-5	0	2.7	-41	1191.62	-5	0	Q	0.01	H	-4.42	-80
BRISTOL CHANNEL	___00_12	0	51.491669	-2.884	338630	177390	-5	0	2.7	-41	1193.2	-10.22	0	Q	0.01	H	-9.64	-80
BRISTOL CHANNEL	___00_69	0	51.506168	-2.9025	337370	179020	-5	0	2.7	-41	1194.28	-10.42	0	Q	0.01	H	-9.84	-80
BRISTOL CHANNEL	___00_44	0	51.520329	-2.85383	340770	180550	-5	0	2.7	-41	1194.49	-11.46	0	Q	0.01	H	-10.88	-80
BRISTOL CHANNEL	___00_76	0	51.561001	-2.746	348290	184990	-5	0	2.7	-41	1199.62	-9.92	0	Q	0.01	H	-9.34	-80
BRISTOL CHANNEL	___00_54	0	51.47467	-2.96933	332680	175580	-5	0	2.7	-41	1191.61	-10.31	0	Q	0.01	H	-9.73	-80
BRISTOL CHANNEL	___00_73	0	51.540829	-2.801	344450	182790	-5	0	2.7	-41	1196.18	-11.58	0	Q	0.01	H	-11	-80
BRISTOL CHANNEL	___00_34	0	51.589329	-2.69417	351910	188110	-5	0	2.7	-41	1203.03	-9.01	0	Q	0.01	H	-8.43	-80
BRISTOL CHANNEL	___00_52	0	51.478168	-2.90417	337220	175910	-5	0	2.7	-41	1191.32	-10.91	0	Q	0.01	H	-10.33	-80

Figure 66 A screen shot of the .csv used for creating musical material for *Argos Pansonos*.

Figure 66 shows the first entries of the original .csv file[176] with data used for creating the musical material. While this original data has a certain repetitive pattern, more contrasting material with a different (numerical) output was needed. A Gumowski-Mira fractal calculation[177], also displaying a degree of repetitiveness, was chosen (see Figure 36, page 66). This would result in distinct musical output when the data has been translated (CACE4 MIDI Translator object) to the musical domain. Both input streams are regrouped xy-paired output sets ordered into different categories (see Figure 67, page 103), as they are analyzed with a (separate) k-means cluster analysis algorithm.

[175] The data were retrieved from the site (http://www.bgs.ac.uk/home.html) of the British Geological Survey (in the Standard BGS Land Gravity data delivery format) in .csv format. It is a file consisting of gravitational measurements of the UK continental plate (off-shore).
[176] The screen shot used is the output of the program File Spy. This is a simple utility program for looking 'into' files and displaying additional header information. It can be downloaded from the Apple Appstore.
[177] The Gumowski-Mira fractal calculation is one of many fractal calculations available in CACE4 as a MATH Generator object.

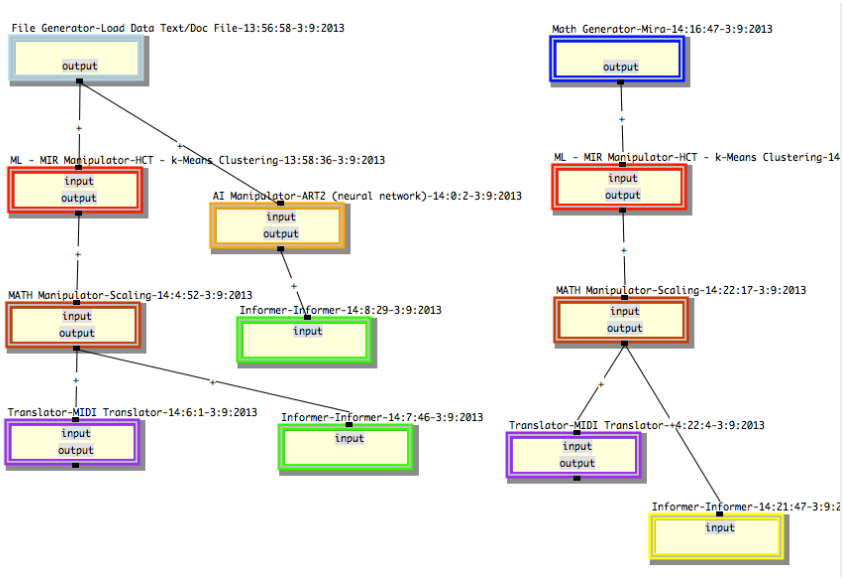


Figure 67 The CACE4 Processor strategy for *Argos Pansonos*.

As a third source for obtaining material, I used a short sound file of a piano sound (made in the inside of a piano) and made a spectral analysis with the SPEAR program (see Figure 68, page 103). After filtering the spectrum -30dB[178] and creating a SPEAR analysis file with lesser partials, it was now possible to import this file into the CACE4 program and to make use of this program with all its possibilities.

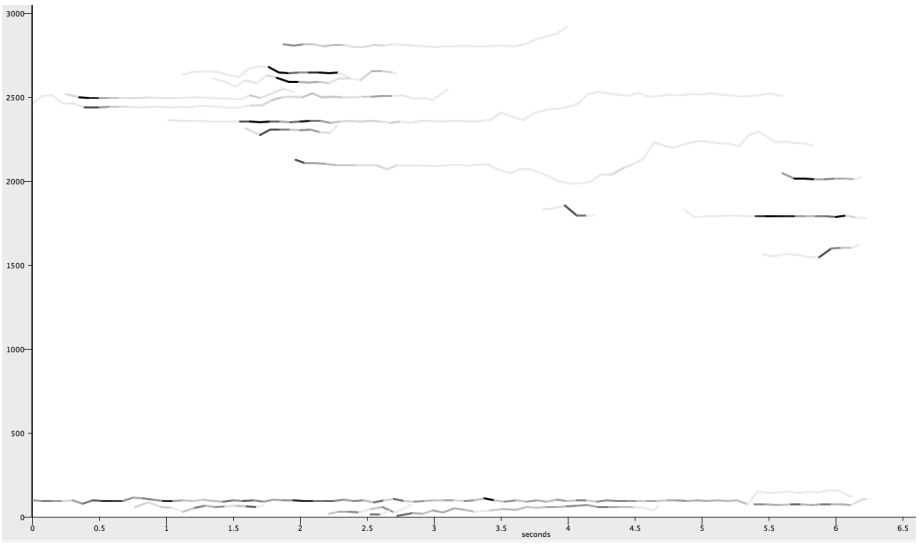


Figure 68 A screen shot of the spectral analysis of the used piano sound as seen in SPEAR.

[178] A first attempt was at -40dB but this was not useful for creating note material for the composition due to the fact that too many partials were calculated, resulting in too much data, not well defined and widely spread, being generated. By using -30dB spectral amplitude filtering, and drop all values below this range, the important spectral information was left over and thus less data points were created.

The analysis obtained in SPEAR was used to create a direct translation from the spectral points (time, frequency and amplitude) into a spear partials text file and as such has been used in the strategy and then finally translated by the CACE4 Translator object for creating a SMF as showed in Figure 67, page 103.

6.2.2 Musical analysis.

Section A, Bars 1-12[179]

As previously explained, the text-based SPEAR analysis file was the basis for this part A, that acts as an introduction to the composition. Several of these files were used: one major file together with several smaller ones. After reading these files into the CACE4 program, they were grouped using k-means and then linear scaling it into the MIDI domain[180], thus creating a SMF. It was now possible to bring this material into the score of the composition. Using the well-known music notation program Finale, some alterations and minor edits of the material were done by hand in this program.

Section B, Bars 13-31.

In the second part of the composition, some Gumowski-Mira fractal-generated material was used and analyzed with the *k*-means algorithm. With this procedure, it was possible to regroup the material thus generated and to make a selection based on these groupings. In section A and B, Max/MSP is used for transforming the sound of the piano with multiple adaptable delay lines. Delay times are changed in real-time (after an analysis of the input signal).

Section C, Bars 32-45.

A sound file input was used for analysis by the program SPEAR (see Figure 68, page 103). The material for creating the score for the inside of the piano was retrieved from the spectral analysis, done again using SPEAR. A selection of a smaller part of a sound file which consisted of a recording made earlier (2005) from a sound from the inside of a piano (a wire-brush hit the strings with the aid of a full sustaining pedal to obtain a sound with a rich spectrum and long duration over time). Only the rhythmical patterns created by this analysis (made out of partials) for duration and delta start-time of the notes was used. The spreading of the pitch information over the full range of the piano was done by hand, thus creating the separate pitches for a register form of notation (6 separate registers were used). It was the intension, in this part of the composition to switch to register notation, which is more suitable for notating sounds created in the inside of a piano.

The main function for this part was to act as a bridge between the 'normal' sounds of a keyboard,

[179] The musical analysis makes use of the rehearsal marks as can be found in the score of *Argos Pansonos*.
[180] See for a more detailed explanation of this process at Chapter 3, with an extended description of how the CACE4 program works.

although the sound has been altered by multiple delay-lines and sounds from the inside of the piano. This was achieved by focusing on the use of piano chords, also derived from spectral analysis of the piano sound and constructed into chords. The actual duration and delta start-time of the material was, in this part, ignored. The chords have been modified and altered by the use of multiple real-time harmonizers, where delay-times and the amount of harmonization will be altered by real-time analysis of the piano chords.

Section D, Bars 46-49.
This rather short section acts as a bridge (or very small interlude) to the next one.

Section E, Bars 50-67.
Section E returns to register notation as this time small chains are used for playing the inside the piano. As the main DSP treatment of the sound real-time time stretching (dilatation) of the signal done by FFT (= Fast Fourier Transforms) was chosen. This results in the smearing of spectral components in real-time, while movement in the 3D audio space is added after the signal has been changed. Modifications have been made in the duration: 400% disturbance (= one of the Manipulator processes available in CACE4) has been used to change the original material that was derived from the previously analyzed piano sound file. At bar 57 a change takes place. Instead of making use of small chains wire brushes are now used for playing on the strings of the piano and this makes rhythmical playing more precise. The change back to using harmonizing as the major DSP treatment of the signal is due to its spectral enhancement and is musically connected to the previous part.

Section F, Bars 68-79.
This represents a major break with the previous sections. Halfway through the composition we go back to the keyboard. Although multiple interactive, real-time delay lines are used to alter the signal, the main focus is still on playing 'normal' melodic lines on the keyboard together with strongly accented chords in the last measure. Together with tempo MM = 64 this creates a ‘back to earth’ feeling in the middle of the composition. In the last measures of this section, the melody line gets transformed into the last two chords: providing a gradual change into the next section: G.

Section G, Bars 80-84.
This short section of just 4 measures acts mainly as an interlude to transform the composition from more melodic lines and chords to more sound orientated (from the inside of the piano). Chords are broken up in higher and lower parts, both notated on transposed staffs (the right hand: up one octave and the left hand one octave down). Thus chromatic, slowly plaid chords as a prelude to section H are created.

Section H, Bars 85-96.
This part is, for the last time, played inside the piano. The tempo suddenly drops to MM = 60. Now a metal guitar slide is used, slowly rolling along the strings (with much use of the blocked sustain pedal). Sometimes it is used to produce very short and quick-slides in order to produce more squeaky sounds. The time stretching of the FFT's, as also used in the previous section E, are prolonged until the end of this section. It all ends with a bang: the slide is 'thrown' on the bass strings and mirrors the beginning of this section, which started with the same sound.

Section I, Bars 97-100.
These four bars are acting as an introduction for the final part J of *Argos Pansonos*. Material for these four bars was separately created with a Brown fractal in order to create material with a large spread and as random as possible. Musically, a more pointillist character is obtained and is used in this isolated way as an upbeat to section J.

Section J, Bars 101-111.
This is the final part of the composition. DSP alters the sound in real-time and slowly the composition comes to a halt.
Tempering the dynamics (going from f to ppp), part J starts with chords altered by the DSP. Harmonising and FFT stretching is used to alter the chords and their spectra. Glimpses of the previous chord-based section (G), prepares the composition for its final bars. The dynamics are broad: from subito p to fff is used. It ends with a small dissonant cluster: d-e flat-e, slowly decaying from fff to p.

6.3 'MMM_Transforms in pink': four pieces with computer animation.

MMM_Transforms is an ongoing (for the past ten years) collaboration project with the Dutch graphical artist Willem Willemse. *'MMM_Transforms in pink'* is the second DVD-video project and will, just like the previous series (*'MMM_transforms in black and white'*), consist of a total number of twelve animations with music. The first four new animations with music of this second DVD-video project, *'MMM_transforms in pink'*, are presented here.

Willem Willemse, commenting on his animations:
"Our experiences don't end for us in the visible world. We perceive more layers as we look into the universe or feel under our skin. It's not static, it moves and makes sounds. In order to move within these realms, science devises models, religion uses rituals. As an artist I try to express this connection in my animations. The animations create a feeling of constant change. They conjure up different

atmospheres: now the feeling of space, then that of a microstructure or of growth.
It is concerned with our meso- or middle-entity in relation to the micro- and macro-worlds.
The pictures seem to occupy the border between the material and immaterial.
Using 3-D programmes, elementary forms are animated into arbitrary organic bodies. Visual elements such as transparency, layering and light give a distinctive expression to the animations. These elements have evolved out of my earlier paintings with movement being the most important addition to this basis. The animations are three to four minutes in length. Static forms alternate with slow and more fast moving formations. The forms appear from a point or out of a matrix, move within this and progress, to an arbitrary order. By matrix I mean origin, an order without hierarchy.
In *MMM_transforms*, both the layering and transparency are maintained; the matrix or grid, as an equal distribution of points, is present as a visual element in addition to the randomly formed surfaces. The future is determined by the polarity between these two visual facts. The matrix stands for precision, determinism or the ordered; the random surface represents uncertainty, the malleable or indeterminate. This seems like an abstract fact, but on reflection it extends far into our thinking and acting."[181]

6.3.1 Musical Analysis and the use of sound.

On the sound for *MMM_Transforms in pink* :
After finishing the previous series (*'MMM_Transforms in black and white'*), I had to rethink the concept of how to approach this new series with for now, four new compositions. This time I didn't want to make use of electronic sounds as before, but felt it should have a more 'acoustical' approach. Plain instrument sounds, although generated by samples (used software: MachFive – MOTU), were the initial sound source. All short compositions (3'40) should have as an initial starting point a rather 'normal' acoustical feel. In time, alterations would change this 'normal' behaviour to create an atmosphere of 'verfremdung': the instrumental sound is starting to loosen itself from the original acoustical instrumental foreground. Speeding up of the notes over time, plus the use of artificial layering combined with artificial movement in a 2D acoustical space, adds to this concept as well. Each short piece was made separately for each animation and the whole should be regarded as an animated installation, where movement of the graphics combine, or collide, with the music in the (exposition) space[182]. Four instruments: piano, cello, celesta and timpani, were chosen as the initial sound sources. Although rather uncomplicated in sound and form and each only 3:40" in duration, it is always a challenge to make the right decisions at the right moment.

[181] Comments made by Willem Willemse at his exhibition at theP-Arts Gallery 27th of June 2011, Zeist The Netherlands.
[182] And as such should be played back with a modest sound level, just enough to 'fill' the room.

Here the music not only supports the animations without sacrificing too much autonomy, but also at the end of each piece envelopes the visual forms as well, carrying the graphics with the sound into the acoustical space.

This has, at present, resulted in four etudes for spatial composition, instrumental sounds and computer graphics: *'MMM_Transforms in pink.'*

Compositional remarks:
As mentioned before, SMFs were generated in the CACE4 composition environment. K-means cluster analysis techniques were combined with fractal generated material, scaled in the right pitch domains. Sometimes left over material was used, generated for the piano composition *Argos Pansonos*. Unplayable by a human being, it was still usable as musical material interesting enough to be slightly altered and transformed for use in these four compositions.

While all the graphics are computer-generated images (executed in Maya[183]), it would have been straightforward to also use it for synchronization of the music with the graphics. This would result however, in a rather static image/music relation. To avoid this perfect synchronization, the artistic idea of creating dense layers (of sound) and to position them, more freely in time, has been chosen. This results in a more freely artistic translation of the movements in the Computer Graphics. This can be observed in the four compositions (see Appendix 5, DVD-video). The choreographed movements in the Computer Graphics are translated into more independent, abrasive layers of dense clouds of sound. The use of movements however, is not only restricted to these sound layers but is also translated to movement of the sounds in a 2D (Dolby Surround 5.1) Audio space.

The four compositions will be part of a larger DVD concept, which will be realised in the latter half of 2017. All musical material has been generated in the CACE4 composition environment, using only the MachFive2 (MOTU) sample library. Mastering and additional editing was all done in Digital Performer (MOTU). It was used for layering of the generated MIDI material and controlling their movement in space.

There is one more important artistic consideration: the final, larger DVD-video installation project is to be presented as a sound and vision installation at a gallery, and not as a movie (or concert). The final music will be played on a 5.1 home theatre Dolby surround system at a rather low sound level, just filling the acoustical listening space. Since being presented in a gallery, the audience will mostly not listen to the whole of all the pieces, but only excerpts, as they move along and visit other parts of the exhibition.

[183] Maya® is commercially available computer animation software from the software company Autodesk. See for more information on Maya: http://www.autodesk.com/products/maya/overview

6.3.2 Musical analysis.

orgamatrixflf(1-12)vert.

The first of four compositions for computer animations: *orgamatrixflf(1-12)vert,* makes use of normal piano samples as a single source of sound.

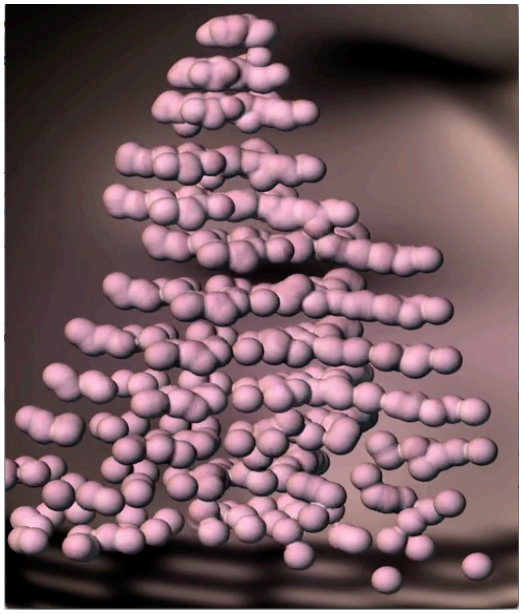


Figure 69 ***orgamatrixflf(1-12)vert*****, computer animation: Willem Willemse.**

In this composition left over material, generated for the piano composition *Argos Pansonos*, was used. By shortening the duration and the inter onset time of the MIDI material in a sequencer, clouds of notes were created: unplayable by a human being, but still interesting as musical material with a certain sound quality. By altering these parameters it is possible to use the same material for creating separated single notes, or as a vast and dense sound cloud, thus providing contrasting sound fields for this composition.

It is a continuous process of deconstruction and rebuilding of the material, thus mimicking the flow of the computer graphics. After approximately two minutes in the composition, trills are used as new musical material. By constantly switching between accelerando and ritardando, combined with sudden changes in tempo, the ongoing process of construction and deconstruction is musically emphasized. Small changes in the attack curve achieved by altering the slope of the curve create new sounds, perceived as bowed string sounds. These are used as small contrasting fragments between the larger and louder movement of the trills being played. Strong movements of the sound in the acoustical (2D) space, together with these deconstructed trills, create swirls of sounds culminating in the final movement which is perceived as a 'lifting up' of the sounds before they disappear, one after the other.

sdspheres(10-13).

The second composition, *sdspheres(10-13)*, uses two different violoncelli samples: bowed and plucked.

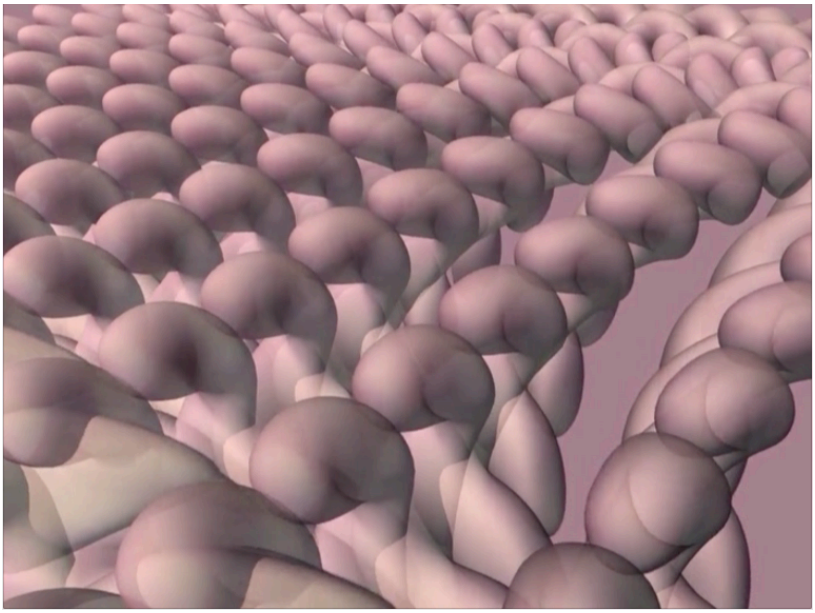


Figure 70 *sdspheres(10-13)*, computer animation: Willem Willemse.

By slightly altering pitch and movement of the sample sounds, dense layers of shifting pitches can be heard, if listened in a surround audio setup. The dense layers of the bowed string sounds create a phenomena which can be perceived as an 'acoustic bath' of dense harmonics. Small changes in pitch fluctuation can be perceived as very small frequency shifts.
On a few occasions this completely surrounds the listener. The ongoing tension it creates, reflects the mechanical movement of the grid pattern in the computer animation. This results in a strong tie between the computer animation and the music, not built on synchronic events, but on movement as displayed in the computer animation. In the final part of the composition, the slow movement in the computer animation does contrast strongly however with the tremolo patterns played by the celli samples thus creating a strong tension between sound and vision.

sc-planes(6vert)zzz.

The third composition *sc-planes(6vert)zzz,* uses celesta samples as its only sound source.

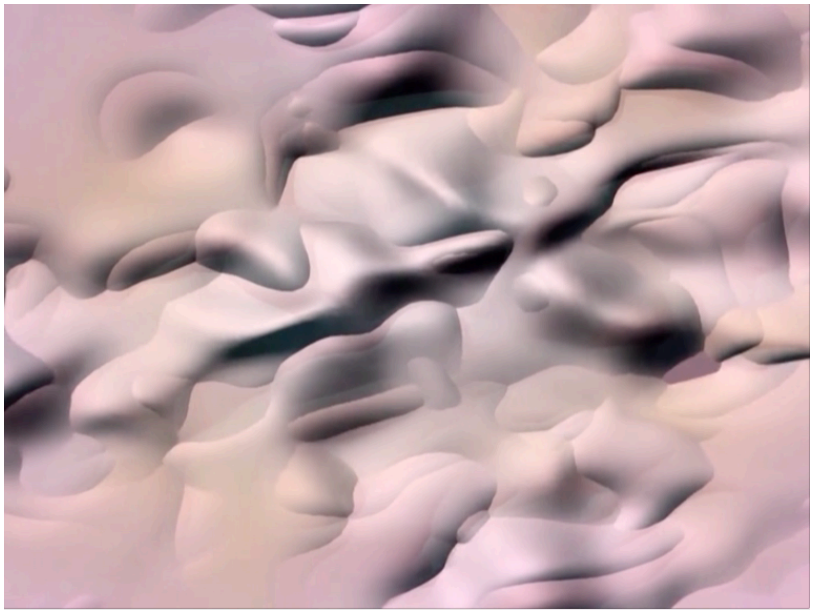


Figure 71 ***sc-planes(6vert)zzz*****, computer animation: Willem Willemse.**

Just like the movements of the computer animation, every sound should be floating around in space. Sometimes it has well-defined borders and sometimes it blends into the background and completely disappears. For this composition this has been achieved by altering (prolonging) attack times of the sample and by movement of the sound in the acoustical space. This process creates diffuse sounds, strongly blending into each other, but with an overall static musical character. In this animation the music is not 'pulling' or 'pushing' at the visuals. Instead it creates an ongoing flow and mimics the process of the slowly evolving computer animation.

dbl-rotormatrixzz.

The fourth composition, *dbl-rotormatrixzz*, uses timpani samples played rolled and as single strokes as the sound source. The computer animation calls strong associations to mind: that of a futuristic virtual machine or the picture of galley slaves rowing by the beat of a drum are just two of these. The 'machine' look-alike and the continuous movement of a 'rotor' as can be seen in the computer-stills of the animation (see Figure 72, page 112), finds its direct reflection in the mechanical rolled timpani sound.

After the computer animation fade-in, the timpani rolls are shifting in pitch and slowly transform into single short bursts of timpani rolls. They play a continuous game of hide and seek, symbolizing the individual, single (sound) component as a part of the whole (computer animation). Alterations in the pitch of the rolls coincide with the more sudden movement of the sound in space.

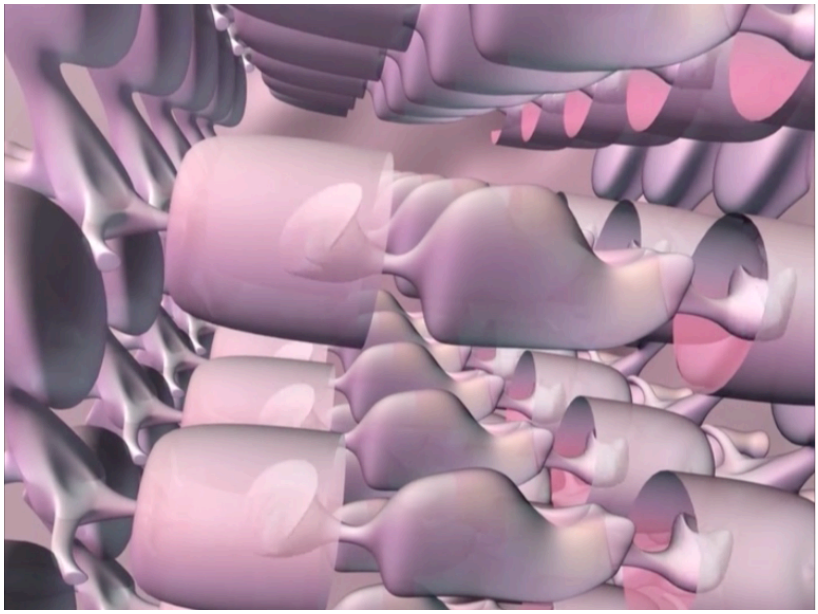


Figure 72 ***dbl-rotormatrixzz*****, computer animation: Willem Willemse.**

Slowly a diminishing of sound density takes place. A continuous thinning of the sound leaves results in a remaining single roll sample. It acts as a point of rest just before a renewed build up of the sound density is used for the finale of this composition as the computer animation slowly fades away.

6.4 'Scope'.

Scope is a ten minute solo percussion composition for six ceramic tiles and two spring drums, all set in a 2D/3D surround audio sound system, together with real-time DSP done by a computer with dedicated Max/MSP patches. The original compositional idea was to create some blocks of data, with certain musical properties. The second step was to create, with the aid of CACE4 STAPS Manipulators, new output with the same numerical properties. The numerical properties of the data are analyzed in order to use them as selection criteria for newly generated numerical material. The newly generated output should have the same numerical properties as its example data. No other rules or restrictions are applied: only the mathematical algorithm and the order of processing have been taken into account for creating the musical material for this composition.

6.4.1 Musical Analysis and the use of CACE4.

As a stable first version of CACE4 (v00.55.07) had just been finished, it was intended to use this composition as an opportunity for testing this first version of CACE4 as a stand-alone application as well. A roadmap was therefore required in order to design the right strategy for this composition as I intended to calculate the output for this composition in just one run. The overall form (A - B - A' - B' - C – D) allowed me to include two statistical property sieve modules (CACE4 STAPS object). Both are

used for analysing a specific part of the composition (A & B will be imitated by A' & B'), and calculate an 'imitation' of the original data, based on certain statistical rules: minimum-maximum range and standard deviation. Due to the fact that random based algorithms have been used for generating the initial data (Part A & B), the effective use of other analysis procedures as available in the STAPS object is limited. Certain characteristics of more random based algorithms are not that easily detectable. The use of a minimum-maximum is important: this gives the initial range, and the use of the standard deviation, which shows us the spreading of the numerical values of the data. The initial size of the floating analysis window is small (2 samples[184]) if this is compared to executing the standard deviation calculation, where a substantially larger analysis window size of at least 20 samples is required in order to be used effectively.

The overall procedure was: first analyze the (input) data and then, with a second calculation, generate new (numerical) material with the specific characteristics. This was repeated several times before satisfying numerical material was generated. The process window, with the chain of data flow (strategy) of the composition *Scope,* shows the way the processing was designed for the composition (see Appendix 1.3).

Part A[185] and Part B act as independent Generator blocks, each with its own processing chain. Part C and Part D are independent Manipulator blocks, where each also has its own processing chain. It uses the output from the original Generator objects for analysis and re-synthesis of new output.

Finally, Part E and Part F act again as independent Generator blocks, each with its own chain of processing. All 6 streams, as seen in Appendix 1.3, have their own independent processing chain.

Each stream also corresponds with a separate block, or part in *Scope*. The chain has been built with a left to right orientation, for visual purposes only. The first, most left branch of the chain consists of a Brownian fractal calculation acting as a Generator object. This is followed by a Pruner object, to remove the x-values of the calculation and a Scaler object, used to scale it to MIDI range values (in this case: into MIDI keys and MIDI velocity values). The Scaler object is also used to scale the inter onset timing and the duration to the appropriate timing values (all values are in milliseconds). At the end of the chain, a CACE4 Merger object adds every output (6 streams) together as separate blocks of data, one after the other. Appendix 1.5, shows the result after the CACE4 Merger object has done its job. The first branch of the chain of strategy has its output plotted on the x-axis between 1-180.

The second branch starts with a random cloud fractal (based on *sin(random)* and *cosine(random)* functions). To further alter the output of the fractal cloud calculation by disturbance, a CACE4 object with the same name is used: Disturber. This was done in order to have a greater, non-related spread of the material generated by the fractal calculation. This is followed by a Scaler object, as is the case in

[184] When presented in xy number pairs a single sample consists 2 members (numbers).
[185] This corresponds with the rehearsal markers in the score.

the previous stream. The output is Part B in the composition (Appendix 1.5, output on x-axis between: 180-400). The third chain (part C in the composition), output on x-axis between: 400-800) is the first imitating branch of our strategy: it uses the output of the first chain (after the Pruner object) to analyse and create a newly generated block of data with the same numerical properties. Appendix 1.4 shows the GUI of one of the STAPS GUI objects with the original input plotted with blue pixels and the newly calculated output is plotted with light-green pixels. The fourth branch (D) of the strategy is also 'imitating' a previously generated data stream: in this case it is the second branch. After the CACE4 STAPS object has been used, a newly generated data block with the same numerical properties is added to the output stream.

The last two branches make use of fractal calculations in order to generate numerical data. The fifth branch, part E in the score, is generated by a bifurcation fractal (or Feigenbaum diagram). Attached to its output is a CACE4 Disturber object, in order to disturb the generated output. After scaling has been applied, it is added and can be seen in Appendix 1.5 as output on the x-axis between 800-1480. The sixth (F) and last branch of our strategy chain echoes the first one. It also makes use of a brown fractal calculation of type: 1/x^5, generating dense numerical output (plotted close to the x-axis) with some elements spread out along the y-axis. By using a different scaling factor, a different data block is created, but with shared numerical properties (with the first block). By using a CACE4 Translator object, the data can be translated into a stream with approximately equal musical properties as the initially generated data block.

Appendix 1.5 shows the GUI display of a CACE4 Informer object attached to the output of a Merger object. All merged (or in this case: added) data blocks are sequentially displayed. The six separate blocks can easily be visually distinguished from each other. After merging all data, a single CACE4 Translator object 'sends' its output to a CACE4 Score object where it can be saved to file as a single SMF. Further alterations and editing of the material has been done in Finale, in order to create a score of the composition. The previously generated SMFs are handled as a block of 'rough' musical material. Notes are deleted or translated to the notational forms necessary for writing the score. In more detail, this means that certain notes were completely transformed into a different notation and others, when no longer needed, are deleted. There were no strict rules involved in this process, just a compositional idea.

6.4.2 General remarks on the process of composing 'Scope'.

Before putting all of the calculated output in the score, certain notational aspects had to be decided upon before the process of creating the score could be started. After some research, it was concluded that there is no standard, nor even a beginning of standardization, of the notation of something as trivial as a spring drum. This gave the opportunity for inventing one's own notation system. I decided to use two staves per spring drum: the upper one is used for 'normal notation': e.g tapping on the top

and side. The second stave is only used for actions with the spring of the spring drum (see Figure 73, page 115).

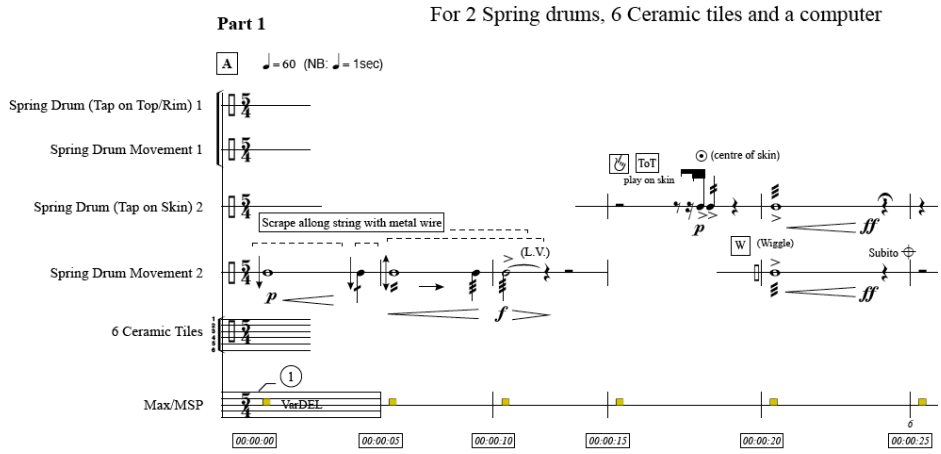


Figure 73 The first 25 seconds of the score of the composition *Scope*.

Also a notation symbol for the hole, on top or the side of the instrument (covering and uncovering of the hole) needed to be chosen (see Figure 74). Before deciding on which symbol to use, the check the literature[186] was always checked and a decision was based on what is more common to use in such a situation.

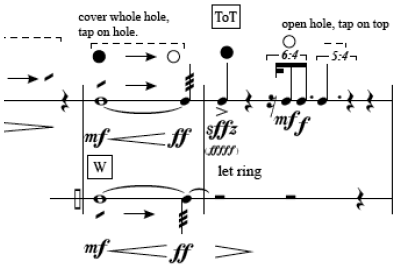


Figure 74 A 10 seconds excerpt of *Scope*, showing different notation styles.

When *Scope* is performed live, it interacts with an ART2 Neural Network Max/MSP object[187]. Parameters are extracted from timbral features in real-time. Detection of f0 and partials are packed together with other parameters to create a vector. This vector: #(DST pitch1 amplitude1, … , pitch5

[186] In this case two older books were used. Risatti: New Music Vocabulary, A Guide to Notational Signs for Contemporary Music(Risatti 1975) and Kurt Stone: Music Notation in the Twentieth Century, A practical Guidebook (Stone 1980).

[187] For the recording found on the DVD-video accompany this thesis (Appendix 7) a version with hand control is used instead of the ART2 Max patch. With future performances this will be replaced by this ART2 object and Max patch as discussed in this thesis.

amplitude5 duration instrument) is then sent to the ART2 NN for further classification (see Figure 76, page 116, for details about this vector).

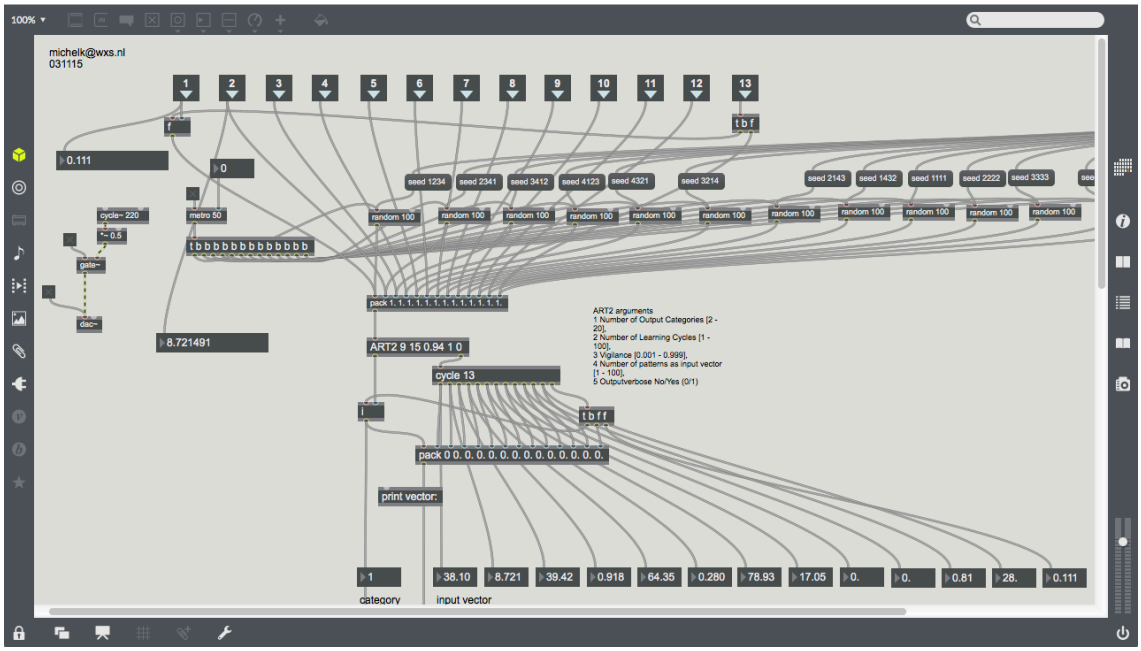


Figure 75 View of the ART2 NN Max object and patch in edit mode.

A LEAP motion infrared sensor provides extra sensory information of the position of the hands in space. This will be used for extracting information on which ceramic tile has been played by the performer.

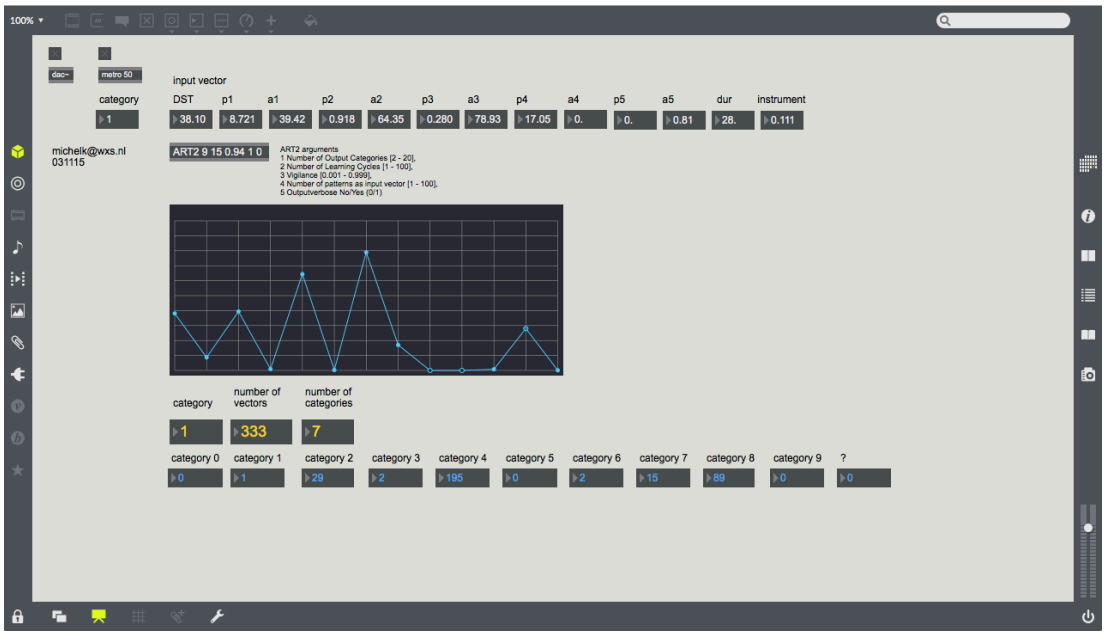


Figure 76 View of the ART2 NN Max object in a max patch in presentation mode.

All this generated information is added to the vector in order to make a classification by ART2 possible. After the classification has taken place, parameters will accordingly handle controllers for changing sound parameters.

The whole idea of creating an automatic controller is a very practical one. Most of the time during concert performances, there are far too many parameters to handle by one person in real-time. Therefore the need for a certain kind of intelligent decision maker is obvious. By using this approach early on in my compositions I am able to add extra 'hands' for controlling patches in live-performances.

6.5 'Zwicky's Box'.

A composition for chamber ensemble consisting of flute (doubling alto flute), bass clarinet, piano, percussion, violin and violoncello, together with a computer with Max/MSP patches for carrying out DSP and surround sound. Duration is approximately 20 minutes.

6.5.1 Compositional Process.

The original idea for this composition for ensemble came from an idea of Fritz Zwicky[188]. His General Morphological Analysis model, later named Zwicky's box after him, is a metaphor for handling complex problems[189]. It is represented as a 3 dimensional cabinet drawer with many drawers. This 3D drawer stands for the complex problem as a whole. Each smaller drawer represents a tiny sub-problem of the whole. Instead of trying to solve the big problem all at once, one takes care to focus on every smaller, but solvable, sub-problem of this box. Therefore complexity can be handled and is therefore solvable[190].

To use this approach for composing, the analogy between solving a complex process and creating a composition needs to be understood. In order to solve such a complex problem - and it certainly can be stated that any composition, composed of x number of instruments, together with the use of computer programs etcetera, is a rather complex process – means that a way has to be found to cut it up into

---

[188] Fritz Zwicky is famous for his ideas on missing matter in the Universe. He introduced the concept of Dark Matter and Dark Energy as a model for this missing matter. It is regarded nowadays as an important idea in the standard model of elementary physics and in the world of astronomy. More information about Fritz Zwicky and thefacts about Dark Matter and Dark Energy can be found at:
https://www.learner.org/courses/physics/unit/text.html?unit=10&secNum=2

[189] Fritz Zwicky describes the model as a new method for structuring and doing an investigation of the total set of all possible relationships contained in a multi-dimensional, non-quantifiable, problem space.

[190] Fritz Zwicky extended the idea of Morphological Analysis into a more generalized version: "I have proposed to generalize and systematize the concept of morphological research and include not only the study of the shapes of geometrical, geological, biological, and generally material structures, but also to study the more abstract structural interrelations among phenomena, concepts, and ideas, whatever their character might be." (Zwicky 1969, p. 34)
Much more information about Fritz Zwicky and this concept of General Morphological Analysis can be found at: http://www.swemorph.com/ma.html

much smaller problems[191].

As a loose analogy, and certainly not according to all the rules of General Morphological Analysis, I used it as a model in order to tackle the problem (hence our composition). I decided to work separately on certain aspects of the compositional process. Timbral aspects were left out of the initial calculation of the note material and were done at a later stage in a separate CACE4 project, thus providing tables with timbral information for the instruments. Figure 77, page 118, shows the timbre table for the percussion instruments.

06/26/16 21:56 Zwicky's Box version: 1
Timbre inventarisatie

Instrument	IG #	Timbre No #	Timbre-group	Sound/Timbre
Percussion	***B-1***		22	22 (27-5) (3^3, 3*9, 9*3)
Chimes	2	pitched- 41&42	P1 B,L+D	Beating, LV + Damp
Clockenspiel	2	pitched- 43&44	P2 B,L+D	Beating, LV + Damp
Cymbals (2)	5	45&46&47&48&49	P3 BO,CH,B,L+D	BOwing, CHains, Beating, LV + Damp
WoodBlock (1)	2	50&51	P4 B,L+D	Beating, LV + Damp
Vibraslap	2	52&53	P5 L+D	LV + Damp
Bassdrum	3	54&55&56	P6 R,B,L+D	Rubbing, Beating, LV + Damp
Snaredrum	1	57	P7 B	Beating
Rainstick	2	58&59	P8 SL,S+L	SLiding, Short - Long
Shaker	2	60&61	P9 SH,S+L	SHaking, Short - Long
	1	62	P9	Silence
	22			

Figure 77 Showing timbre numbers related to Sound descriptions.

These numbers can also be plotted in a xy-axis plot (see Figure 78, page 119) and by giving them a colour coding (percussion is the blue column) and by putting them into separate Excel columns, these events could be spread over time as well (see left excel column, Figure 78).

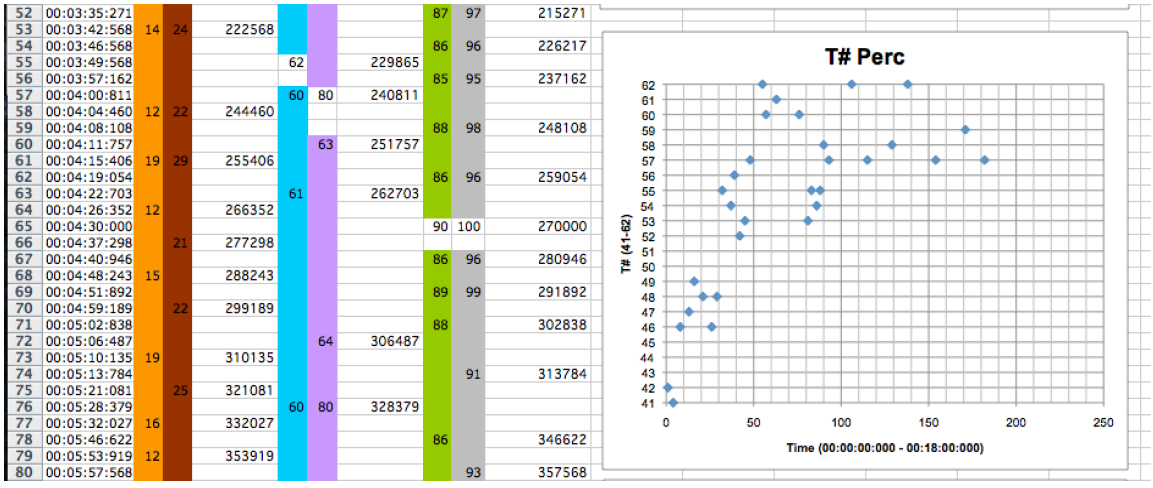


Figure 78 The timbre numbers displayed in a xy-axis plot (time - timbre number).

[191] In my opinion, this is a very interesting approach, and also very well adapted to the world of computer programming and algorithmically created music, where a computer program can be seen as a description of the problem. In order to solve this problem it has to be cut up in smaller chunks with clear stated goals. These smaller chunks represent a much smaller problem where a direct a translation to functions and procedures is now possible. This strategy is needed in order to make a functional description of the algorithms needed for writing the software application and solving the problem.

Initially of great help for investigating all different sorts of timbral options, it is not used in detail (see Figure 78). The leftmost column represents exact timing (this option was not used for the creation of the composition) and was used more as an advisory table as to how to evolve instrumental timbre lines over time. The timbre numbers were calculated by a fractal calculation (dragon curve) and mapped over small ranges (timbre number range) thus providing the number sequence with the right sequential order of timbre numbers.

Other problems involving the use of microphones, 2D/3D sound systems and the use of DSP in an ensemble setting were set aside (in a separate box) to be solved at a later stage in the process.

After initially finding four interesting and also related data sheets about global warming and climate change[192], it was decided to use all four, as this would provide more than enough material to work with. The first glimpses of the data showed that the spreading and also the semi-periodicity, was interesting to use for creating musical patterns (see Figure 79).

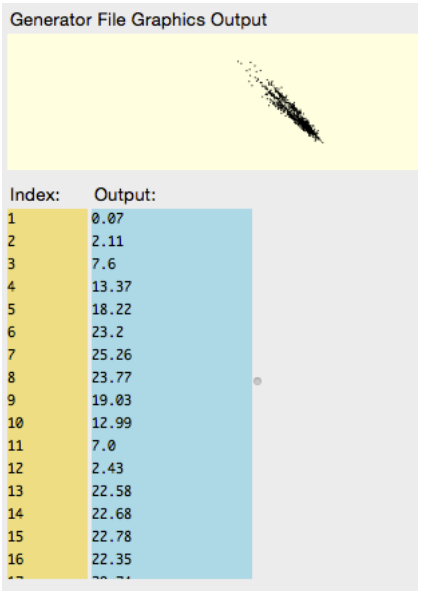


Figure 79 CACE4 Informer object xy-plot of one of the .csv data sheets used for *Zwicky's box*.

A transcription of the original MS Excel sheet into a comma separated data sheet was the first action to be undertaken[193]. After taking special care not to modify any data or otherwise change the original

[192] After some research on the Internet I found the URL: www.data.worldbank.org with some interesting easy accessible databases. This gave me the opportunity to download a CVS/MS Excel sheet at the Climate Change knowledge Portal: It's Historical data about the change of global temperatures. Used data can be found at: http://data.worldbank.org/data-catalog/cckp_historical_data

[193] The original procedure was to eliminate the original header text and replace all comma's (find and replace them with the IDE of LispWorks) by a tab (character, and save it for now as four plain text files. (Tab spaced).

order of the sequence, each file could now be read into a CACE4 FILE Generator object and work started upon them.

The next step in the compositional process was to design a strategy, to transform the data into the desired patterns for our composition. The four data sheets provided material suitable for generating four blocks of approximately five minutes musical material each. This altogether gave more then twenty minutes of material for the composition.

By adding an extra CACE4 CLUS object I, the strategy was further developed by clustering the data into different groups, resulting in a series of chord clusters generated for the piano. Furthermore, Merger, Pruner and a Translator object were added to the strategy chain, for further processing before creating a SMF of the transformed output (see Figure 80).

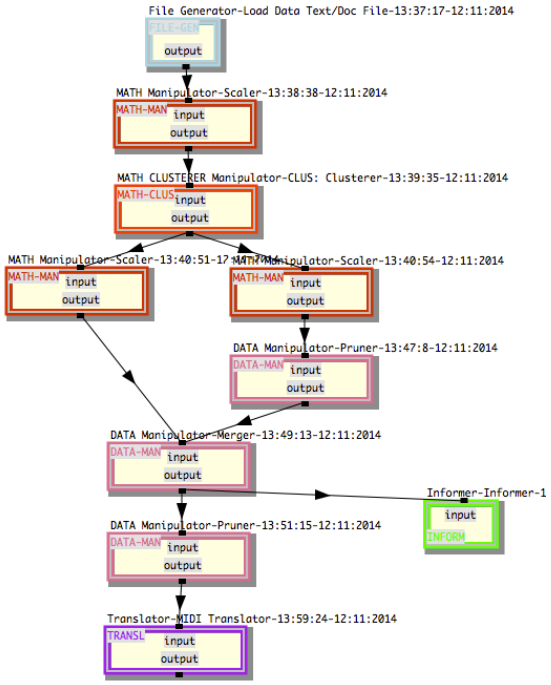


Figure 80 One of CACE4 Project strategy setups used for *Zwicky's box*.

The output of the first strategy developed was very promising, especially for the piano, with broad but slightly clustered chords with a nice initial rhythmic flow (as can be heard in the composition at Bar 20 - 24, Part A). Further on in the composition this method of working was developed into a strategy for orchestrating the composition. This was mostly achieved by 'spreading' the calculated notes by hand to the other instruments. This approach gave the advantage that, although different instruments play the notes of the calculated chords, the overall character (density and harmonic context) of the

chords is preserved[194]. This was, however, not the only strategy developed for creating different parts for the instrument. As a contrasting approach, for creating 'melody' lines with a slight solo character, fractal calculations were used.

For the percussion part, especially those parts involving glockenspiel and chimes, the algorithm used for creating musical material was based on a Sierpinski (dragon-curve) fractal calculation, where took a specific part with some overlapping notes was taken and fitted it into the score (section F, bar 63 – 90). Even single bars (bar 154) were obtained this way.

This same method of working was used for the melodic lines of the flute (part O, bar 212 – 213). This approach, as previously stated, gave the line a more independent, solo character.

After initially working with the previously generated musical material, it was soon realized that a slight change in strategy was necessary. It became rather obvious that too much material was being generated. The major spreadsheets with the 4 tabs generated too much material, although musically interesting. Also the overall musical character of the generated material pointed too much in a note biased direction. This was something to be avoided, due to the fact that the original idea about the composition also involved a more 'sound-based' or timbre approach. Together with the interactive DSP, which added extra timbre alterations, it was decided to drastically cut back on the use of this material in the score. The notes should be more separated from each other, with much more - silent - space between them, in order to create room for the DSP. The use of the DSP means not only the alteration of the timbral qualities of the sound by signal processing, but also to move the sound through the listening space. This movement costs 'processing' time and is perceived by the listener as an independent moving sound source.
Concluding that 'less is more', this was turned into a new directive for obtaining new material and as a guideline for further development of our deployed strategy.

The final step in the development of our strategy and work-flow, was to think about how to use the aspect of timbre and sound combinations of the instruments (see Figure 77, page 118) in a coherent way, together with the DSP.

Modifying the sound and timbre of the instruments with the aid of variable delay lines, harmonizers, granular synthesis and the use of FFT time stretching, came to mind as an initial starting point for doing DSP. Why all these different possibilities? Firstly they provide more DSP possibilities of doing processing for transforming the sounds involved. Secondly they share a single DSP property in that they all make use of some sort of windowed signals and therefore are not totally alienated from each other. There is an overlap in perceiving these processing phenomena and how they 'sound' together. Several MAX/MSP patches involved with the 'live' transformation of the sound combine this

[194] This method of working can be seen as a complementary process approach of a piano reduction of a composition. It is the other way around with respect to a more regular composition process, which is more working from the inside out.

approach in a single processing option. Gradual transformations from one DSP process to another DSP process are possible. Furthermore, by creating a matrix mixing option, one is able to transform the instrumental lines separately according these matrix settings. This matrix based approach makes it possible to link timbral transformations to the matrix of instrumental sounds (see Figure 77 and Figure 78, both on page 118). Not strictly applied, as has been previously stated, the tables are used as advisory tables and as a starting point for bringing DSP and instrumental timbre together.

By now all the drawers of Zwicky's box are filled with smaller problems, by breaking down the 'big' compositional process into much smaller, separately solvable chunks, thus creating a solution for the big problem, *Zwicky's box*, the composition.

6.5.2 Musical analysis.

Section A, introduction, bars 1 – 26.

Zwicky's box starts with a few chords based on Brownian movement (1/x2) calculation, played by the piano. It creates a harmonic layer where individual instrumental sounds can easily be introduced by the other instruments. The overall character is that of an introduction: a slow movement (tempo MM=60) where the DSP is used to add a certain rhythmic character to the more static sound layers created by the piano chords and the use of percussion (from bar 5: bowed cymbal).

Sections B, bars 27 – 41, C bars 42 – 54 and D bars 55 – 70.

These three smaller parts all belong to Part I of the composition. By using the same file as CACE4 input[195] for all three parts, the group is given a coherent musical character with shared musical characteristics. Together they create one large movement of open, more silent spaces where the DSP easily blends with the instrumental sounds (bar 27 – 69). Accents in instrumentation slowly evolve over time, from more piano and percussion orientated, to flute and bass clarinet at the end of Part I.

Part II:

Sections E, bars 71 - 82, F bars 83 – 89 and G bars 90 -97.

Part II consists, like Part I, of three smaller parts combined into one larger movement. The first bar of this part places the piano directly into the centre of attention. Played more vividly, it gives Part II its contrasting character to the first part[196]. It is much more vertically orientated, which also contributes to the more vivid character of Part II.

DSP: The main DSP for Part II is the use of FFT for stretching the signal (this can be heard as

[195] The file: Basin_temperatureCRU-1.txt (our first worksheet of the spreadsheet with Global warming data) was initially used for generating musical material.

[196] For creating note material in CACE4, I used the third tab of the Global Temperature Data .csv file: Country_temperatureCRU-3.txt. In this case Absolute Time was used instead of Inter onset time (for calculating the start time of every note). As a second step in the composing process, a melody line, originally intended for piano was taken and adapted for bass clarinet as the solo voice.

‘hanging’ notes and chords), slowly transforming into a more grain like character. This gives it its rather harsh’ character and clashes, once in a while, with the instrumental sounds.

Part III:

Sections H, bars 98 – 104, I bars 105 – 110, J bars 111 – 116 and K bars 117 – 125.
These four smaller movements are combined to form part III. They are all rather short phrases with a more sudden, explosive character. Section H starts as a single movement played by the piano but suddenly spreading to the other instruments as well. This adds to the expressive, explosive, sudden burst character of the phrase. Section I is a more prolonged and deferred version of the previous explosion character. Section J repeats this pattern but also combines them. Up to the end of this part the sudden character is also prolonged and deferred as a memorizing mirror of the previous two parts. The musical intention of section K is to finish Part III by repeating the pattern of explosion and prolonging as shown’ in the previous three sections (H, I and J). It is even more emphasized by introducing, at the end of this part, a vaguely recognizable IV-V-I harmonic movement (but without the I). This all contributes to the overall musical idea of finishing Part III. For an overall timbral effect, which nicely combines with the used DSP (12 independent variable, interpolate delay lines), a glockenspiel was used. This has a distinguishable timbre and therefore adds a shining, shimmering effect to the chords. Based on continually changing delay times, parameters of detected discrete input events as amplitude and frequency of the analyzed signal are used to change these delay times and volume in real-time.

Part IV:

Section L, bars 126 - 163.
Part IV inherits its musical character from Part III, but creates a (slow) turning point in the composition. It is conceived as one, much longer movement on a much slower time scale (MM Tempo = 32). Therefore melodic and rhythmical patterns unfold over a much longer period of time. The DSP, as continually changing Harmonizers, is used to alter the chords on a more microtonal base. There is no table used with presets of detuning the harmonizer in cents. All necessary parameter changes are calculated on the spot. Therefore a change in selected instruments for generating these input parameters can take place. First the piano is one of the sources, this shifts gradually when the bass clarinet starts to play. Together with the flute it is the major source for input parameter data. In total this creates a kind of strange chord progression effect: a slight feeling of ‘estrangement’ takes place.

Part V:

Section M, bars 164 -191.

The piano part is played inside the instrument with a blocked sustain pedal and the player uses metal wire brushes to play on the piano strings. For this reason the use of register notation with 2 staves, each consisting of 3 staff lines, adding up to a total of 6 lines was chosen. For making use of this technique and to do the necessary calculations in CACE4, it was decided to 'shrink' the output[197] of the pitch calculation by restricting it to only 6 different key/pitch possibilities. By means of linear scaling of the output in only these 6 possible MIDI key values, it was possible to transform the output of the CACE4 calculation into notes for these staves, after adjusting the durations of the notes by 200% in Finale and moving double notes to other instruments (bass clarinet, flute and violin). This longer movement, with less note material, focuses more on sound creation inside the piano and manipulation by the DSP, in comparison to the previous Parts. Multiphonics and muffled sounds played by the bass clarinet, flutterzunge, trumpet tones and pitch-bending from the flute, together with long glissandi by t cello and violin all contribute to this more sound-based part. Processing added by DSP as variable and fixed delay lines, supports this idea even further.

Section N [bars 192 – 204]. This second part of Part V is mostly centred on just a few notes, ranging from b to e-flat. A gradual feeling of slowing down and hence creating a point of tranquillity and renewal is its musical intention.

Part VI:

Section O, bars 205 – 214 starts with glimpses of previously introduced musical material. This quasi-repeat enhances the feeling of conclusion to this composition. Pitch-bending and multiphonics played by the bass clarinet, together with flute patterns created by a Sierpinski fractal calculation, are a few of the repeating musical elements. The use of the DSP repeats as well: variable delay lines are the main processing that takes place, effectively picking up the multiphonics played by the bass clarinet and creating an instrumental bridge to the swirly melodic lines of the flute. By using the bass drum as an extra sound layer of low rumbling noises, the whole creates a feeling of subtle tension.

Section P [bars 215 – 224] is the last part of the composition. It starts with a louder passage consisting of piano chords, bass drum rolls and fortissimo bowed cello notes. The whole has a musical character of slowly vanishing into nothing. Piano chords together with very high-pitched notes (a and c), played arco by the violin, are the very last notes played. The DSP is further enhanced with the use of the FFT Max/MSP patch by stretching the sounds. This creates a rather grainy sound field and leaves on with the impression of a final 'grinding' of the sounds.

[197] The file: Country_precipitationCRU-4.txt (our fourth worksheet of the spreadsheet with Global warming data) was used for generating musical material.

6.5.3 General remarks on the process of composing Zwicky's box.

The whole process of composing music in this manner, from the outside to the inside, is the opposite to the normal procedure of composing, which most often makes use of the process from the inside (composing note by note) and works slowly to the outside.
It is, to a certain extent, more comparable to the process of sculpting: to getting rid of what is too much and not desirable for the composition process. This means that besides experimenting with the data in order to obtain desired results from the CACE4 program, developing a strict compositional idea is essential for making the right decisions at those 'experimental' moments[198].

Unfortunately there are some pitfalls inherent to this way of composing music.
Firstly, there is the possibility of generating too much material (Ouch!), which makes the process of choosing and selecting far more difficult. It is easy to make the wrong decisions by deleting the wrong notes in this process of data reduction.
Secondly, there is a real chance of making errors in the process of transcribing newly generated material to the other instruments. Errors in range and transposition are easily introduced in this process. This can be, to a certain extent, avoided by doing all the transcription processing of the material to the other instruments, in the CACE4 program itself. This idea needs further development, but it could ease this process.

[198] Although I have initially fixed ideas, I always do some experimentation with the generated material purposely in order to obtain more different material and to discover to a certain extent, new possibilities with the data. (To pick up the unthinkable and to be open and especially be surprised by the generated material obtained, which is not a bad approach in the world of Big Data, MIR and creativity).

Chapter 7 Conclusion.

In this last chapter design and functionality of CACE4 will be evaluated. The initially stated goals of functionality and design will be used for this critical analysis.
The four Major design criteria, as stated in section 4.1.1, 4.1.2, 4.1.3 and 4.1.4 (pages 20 - 22) are:
1/ The application should be a fully operational version but also 'open-ended' for future developments.
2/ The modular design of the CACE4 program should allow for easy adaptability. Therefore the process of developing a new CACE4 object (software) module should be rather easy and inside certain time limits.
3/ The application can be used for educational purposes.
4/ The use as a computer music composition environment.

7.1 Final conclusion.

Although the application has proven its usefulness, as originally conceived and designed, as an open strategy building tool for analysing (non-musical) data and for using the output for creating musical output, it has its limitations and shortcomings. As previously stated in section 5.8.2, the object's ease of extendibility is not comparable with, for example, Max/MSP where Java, as an interpreted language, can be used to quickly create new Max or MSP objects. The more detailed and elaborate possibility to create external objects programmed in C, however, is comparable to the way CACE4 is extendable. It operates at the same level of coding; source code which needs to be compiled into a new external object (MAX/MSP) before being fully operational and can be used in the software environment[199]. Also due to the fact that it is a stand-alone application, a newly delivered[200] application has to be created first in order to be able to work with the new CACE4 object. Using a template[201] (a file with example program code) however, for creating a new object, is comparable with the way Max/MSP operates. One of the advantages of CACE4 is that it is not a DSP program and does not need a real-time DSP-engine. This makes it possible to fully exploit the possibilities of the desired algorithm in combination with its specific GUI. No real-time software-engine is necessary and

[199] Before this feature can be build into a newer version of CACE4, a few more additional changes in coding has to be made, so that the user can explore CACE4 by adding extra Objects in the Listener. These direct changes involve more program structure (on the level of coding) by using *defgeneric()* methods.
[200] LISP terminology for the process of compiling and linking.
[201] The design and programming of a template file (with all the necessary coding for creating a new CACE4 Object - fully functional with all slots for I/O (STREAM Input/Output and COS&MOS2 Link) has already been done in the early stages of developing CACE4. By making use of these templates the development time for fully creating a functional - new - object is now brought back to a few hours for the easy ones, what could extend up to 2 or 3 days development time for the more complex and elaborated one's (This is including the time for developing the algorithm and the corresponding functions in order to have a working, first version of the process).

therefore a process such as k-means can be applied to the input as a whole[202].

One of the major goals was to have a functional and running software package as a stand-alone application, but also with a maximum of adaptability. This target has been fully accomplished by taking this goal and making it a core program design rule of CACE4. Object Oriented programming[203] techniques, as CLOS provides, together with the use of *defgeneric()* method combinations, have been successfully deployed and used for the development of CACE4, thus speeding up the process of implementation. As such, OOP techniques were used for implementing CACE4, thus combining criteria one and two. Without the aid of these techniques, this would not be achievable in such a short period. There are still portions of the code however, that need some redefinition to be more CLOS compatible, with the aid of further deployment of *defgeneric()* functions. Some refinement in implementing Controllers in order to 'hang' them in the object, for the interface (the way CAPI implements the MVC paradigm), needs extra attention. This will also result in extended GUI's with new possibilities.

The third goal, of creating an application useful for education has still to be proven in a live situation, although my ten years of teaching undergraduates at the University of the Arts, Utrecht, in the domain of Music and Informatics indicates to me that the way it is now, with the GUI and the use of SMF output in order to playback the result (as yet, unfortunately not real-time) in any available MIDI sequencer, will be much more appealing to the students, than an application without a strong graphical representation. It shows them the graphical and also numerical output in several ways, and by being able to compare the original input with the calculated and plotted output. This is a very useful feature in order for students to get acquainted with the domain of statistics applied on data, music and sound. As has been previously stated in section 5.8.2, CACE4 needs a LISP interpreter and a Listener incorporated in the environment to extend the usability for educational purposes. With this feature it would be possible to teach LISP in the same computer composition environment as well.

The way CACE4 presents itself is strongly relying on its GUI for operation. Although speeding up the process of designing a strategy, it comes at a certain cost: it keeps the inner operations of the software package hidden from the user. The GUI of each CACE4 object is dedicated to the process it represents.

The implementation of extra explanation features by showing the source code of the program together with additional diagrams and linking it to the specific mathematics necessary for full comprehension of the process would increase its educational value.

---

[202] *K*-means is applied in the domain of DSP as well. In a real-time situation the signal size (chunk size: mostly 1024 or 2048 samples) is the band-limited signal we use for operating on.

[203] Design criterion one (a fully functional program) and criterion two (the modularity of the design) have a certain connection. Both criteria can be combined into a single list of OOP-design features.

The fourth criterion: is CACE4 as a computer composition program focused on the use of statistical tools suitable for creating Acoustical and Electronic Compositions, requires a more detailed answer. At present, three major instrumental compositions and four compositions for computer animations have been created with the aid of the CACE4 program. They all have in common that an initial idea as a kind of meta-level concept, was used in order to set up a strategy. This initial idea needs some refinement and more detail. This would be reflected in the CACE4 program by adding some extra objects, for example, a Context object, which makes this strategy approach clearer for the user.

CACE4 has proven to be very useful in creating SMFs for use in Electronic/MIDI based compositions such as the four composed for the Computer Animations of Willem Willemse. Smaller chunks of data with interesting musical properties are easily generated and manipulated in the CACE4 program[204]. CACE4 proved to be a program for quickly generating SMFs suitable for further editing and manipulation in a MIDI sequencer program.

The first instrumental composition, *Argos Pansonos,* was composed in 2013 and the last one, *Scope,* was composed in 2016, a time span of 2.5 years. The four shorter MIDI based compositions for Computer Animation have a comparable time span. Not all CACE objects were available in the earlier version, and therefore it is difficult to compare them. They all had relative ease and speed of creating interesting musical material in common. With the flexibility offered by CACE4, smaller experiments could be easily done and discarded if not sufficient. What is missing that would bring the program as such to another level altogether, would be the implementation of context (as a separated object). This would make the program much more valuable for use in the domain of music style recognition[205]. Although originally designed for use as quantitative analysis on (unknown) data sets and without the aid of a priori knowledge, the software tactics, together with the CACE4 objects, can certainly say something about the underlying mathematical constraints. A direct link to particular characteristics of music style, without using any qualifying context, is therefore not possible. With the aid of a few other CACE4 objects such as the Splitter, Merger and Scaler however, a certain pseudo-context can be added. By using the order of the Stream, giving every member of the sequence a certain range (min-max), 'forced' (hidden[206]) constrains can be applied.

The objects most useful in searching for certain statistical characteristics and therefore can be used to work with these pseudo-constrains, are the CACE4 STAM object (section 5.5.2, page 56) and the CACE4 STAPS object (section 5.5.4, page 64). They offer the GUI for entering parameters for forcing new output to have those mathematical properties as desired by the user. Serial and atonal music can use sets of tools based on both simple and advanced mathematical statistics and data analysis as input for a musical composition.

---

[204] The implementation of MIDI controllers would enhance its use in an electronic Music composition even further.
[205] As previously described in more detail in chapter 3.2 & 3.3.
[206] These constrains can be seen as pseudo or false constrains while they are unknown to the system.

Other similarities in musical functionality can be found. For example the Clusterer, as a cluster (chord) generator, has the same functionality as a chord generator in tonal music. They both share this same musical functionality although their results can be perceived differently.
Analysis with processes as EM (Expectation Maximization, see section 5.6.2, page 81 for details) can be applied in the domain of counterpoint, to find certain characteristics closely related, but also still with a certain independence and cluster them accordingly, thus creating new groups of sorted (clustered) musical material.

Although outside the original scope of this thesis, but available in CACE4, fractal calculations can be applied as melody or rhythm generators. One of the well-known characteristics of a fractal calculation is self-similarity and it is thus capable of creating streams of data with a very strong internal relationship. When calculating the correlation coefficient (r) from several fractals and attractors, one notices that the r is often between $[0.5,\ldots,1.0]$[207]. This suggests a stronger correlation in product moment and rank-order. This strong internal relationship can be exploited for generating melody lines. Fractals as the random cloud and the dragon curve (Sierpinski) with output generated according $f(x)=sin(x)$ or $f(x)=cos(x)$ provide quasi periodical repetitiveness and can be well deployed for generating rhythmical patterns by using their output as delta-time and duration of the notes.

These are just a few comparisons in musical functionality that can be found in CACE4. Especially the use of 'extracting' certain qualitative characteristics and applying them to a newly created stream in accordance with these qualitative properties opens up a whole new field for experimentation and future development.

Overall, the speed of working and the possibility for doing experiments in CACE4 has contributed greatly to the musicality of the compositions.

7.2 Future development plans.

When creating a software package such as CACE4 with such a number of different domains and topics involved, there always remain issues for further improvement and future development. The most important ones are listed below. Some ideas exist already from the beginning, having been developed during the first design stage of CACE4. Others have a more practical background and came into being during the further development of the program.
A music-language construction is, at the time of writing, not available, but will be provided as a separate CACE4 object module. This idea was originally omitted since CACE4 should not be music-

[207] Some of the fractals with their correlation calculations (all numbers are rounded to 2 digits): Julia fractal: Pearson: 0.73, Spearman: 0.66, Kendall-tau: -0.62, Mandelbrot1: P: 0.0, S: 0.01 K-tau: 0.77, Bifurcation: P: 0.76, S: 0.82, K-tau: 0.85, Automata: P: 0.0, S: 0.01, K-tau: 0.99 and Gumowski-Mira: P: 0.52, S: 0.51, K-tau: 0.14
And some of the Attractors (All as 2 Dimensional) attractors: Henon1 (2D): P: 0.01, S: 0.38, K-tau: -0.50, Lorenz (2D): P: 0.80, S: 0.79, K-tau: 0.94 and Rössler (2D): P: 0.19, S: 0.16, K-tau: 0.99

language based. As a separate object embedded in the CACE4 Environment however, it brings a new, strong concept to CACE4: that of (musical) context.
The Score object, where everything is translated to a SMF can only provide a MIDI format 1 file. This is a severe limitation and preparations are being made to extend the possibility of saving in another SMF format of type 2. The other, rather practical issue, is the lack of a real-time (MIDI) playback option as previously mentioned. All compositions could have been created more easily and more directly if this option had been available.

The development of a new CACE4 Translator object for creating Lily Pond files has also to be taken into consideration. Although the solution used - the importing of the CACE4 SMF into a program as Finale – is, for now, workable. The other Translator object that could be interesting to develop would be a CACE4 SPEAR Translator object for creating SPEAR (partial text format) files. Results, especially from larger data sets, could be directly written in this format for further editing and use in SPEAR.

This last object opens up the possibilities of bringing (some) DSP techniques and knowledge into the CACE4 environment. This idea needs further development: a small DSP Library for composing the electronic composition *Ploutôn* (2010) has been developed. It is a C++ program and its DSP Library is easily turned into an external DSP Library (by using the FLI library).

The GUI of a few objects (Correlator, Splitter and Merger) needs some redesign. Although the underlying algorithms (Model) are working well, the GUI needs some attention. More detail in specific aspects of the GUI require further development in fine-tuning the GUI to the process (adapting the View and the Controller to the Model). This will result in more precise results and more possibilities.

The GUI should be extended with a 'true' three-dimensional view. This can be accomplished by adding a third dimension (or z-axis) to the display. This creates the possibility of zooming and rotating clusters in a scatter plot. Also an implementation of Oculus rift goggles, as an extra way of projection of the data, would be an excellent option to have in this three-dimensional view[208]. Although the CAPI Library of LispWorks offers many classes, methods and functions for use, further 3D geometry algorithms need to be developed for this future implementation[209].
In this version of CACE4 there is still no possibility for saving - the state - of the program. Creating persistent code with all states of the objects preserved is therefore not possible yet, although an

[208] Data points can hide behind other data points: especially in larger data sets this kind of visual blocking can occur rather easily. This makes it much harder - without a certain form of graphical rotation - to detect certain constraints between these points.
[209] There are certain pseudo 3 Dim techniques of graphical projection: Glyph projection. A pair of goggles with a red and a green glass is used in order to create some visual (illusion) of depth in a still 2 dimensional plotting window.

attempt is on its way[210].

One last point of attention should be applied to the precision used in CACE4: the internal LISP float-type, used for doing all the arithmetic is in double-float precision, which gives a floating point format with fifteen digits behind the point. On several occasions round-off error will be introduced. Future plans to eliminate these round-off errors can be achieved by implementing a simulation of a new upcoming standard as suggested by John Gustafson: UNUM, which stands for Universal Numbers. The benefits are considerable in that it could mean the end of all sorts of round-off errors in, not only, floating point computation[211]. The disadvantage is that it needs to be coded in hardware to be really effective, and therefore it will take some time before being implemented and widely available for use.

[210] For now (May 2016) the version of CACE4 can save the objects to a (text) file. For now, only the Objects and their positions can be retrieved, thus further work needs to be done.

[211] *The End of Error Unum Computing* by John L. Gustafson is the book to read on the topic and implementation of UNUM (Gustafson 2015).

Bibliography.

Adrian, Richard H., et al. 1987. *The Oxford companion to the mind*. Oxford, UK New York, USA: The Oxford University Press. Encyclopedia.

Alpaydin, Ethem. 2010. *Introduction to Machine Learning Second Edition*. Edited by Thomas Dietterich. 2 ed. 16 vols. Vol. 16, *Adaptive Computation and Machine Learning*. Cambridge, Massachusetts, London, England: The MIT Press. Machine Learning, Informatics.

Ariza, Christopher. 2005. *An Open Design for Computer-Aided Algorithmic Music Composition: athenaCL*. Boca Raton, Florida USA.: Dissertation.com. Music & Informatics.

Barbaud, Pierre. 1965. *Initiation a la composition musicale automatique*: Dunod, Paris 1966. Music & Informatics.

Bergson, Henry. 1948. *The creative mind. An introduction to Metaphysics*. Translated by Mabelle L. Andison, *First Carol Publishing Group edition*. New York, USA: Carol Publishing Group. Philosophy.

Büttcher, Stefan, et al. 2010. *Information Retrieval Implementing and Evaluating Search Engines*. Cambridge, Massachusetts, London, England: The MIT Press. Information Retrieval, Informatics.

Carpenter, Gila, and Stephen Grossberg. 1987. "ART 2: self-organization of stable category recognition codes for analog input patterns." *Applied Optics* 26 (23).

Cope, David. 1991. *Computers and Musical Style*. Oxford, UK: Oxford University Press. Music & Informatics.

Cope, David. 2001. *Virtual Music Computer Synthesis of Musical Style*: MIT-press. Music & Informatics.

Cope, David. 2005. *Computer Models of Musical Creativity*. Music & Informatics.

Dessain, Peter, and Henkjan Honing. 1992. *Music, Mind and Machine. Studies in Computer Music, Music Cognition and Artificial Intelligence*. Amsterdam: Thesis Publishers. Informatics and Music.

Floreano, Dario, and Claudio Mattiussi. 2008. *Bio-Inspired Artificial Intelligence Theories, Methods, and Technologies*. Edited by Ronald C. Arkin. 16 vols, *Intelligent Robotics and Autonomous Agents*. Cambridge, Massachusetts, London, England: The MIT Press. AI, Informatics and Biology.

Ghezzi, Carlo, and Mehdi Jazayeri. 1982, 1987. *Programming Language Concepts*. 1 vols. New York, Chichester, Bribane, Toronto, Singapore: John Wiley & Sons. Informatics.

Grant, M. J. 2001. *Serial Music, Serial Aesthetics. Compositional Theory in Post-War Europe*. Edited by Arnold Whittall. 12 vols, *Music in the Twentieth Century*. Cambridge, U.K.: Cambridge University Press. Music and Aesthetics.

Gustafson, John L. . 2015. *The end of error. UNUM computing*. Edited by Horst Simon, *Chapman & Hall/CRC Computational Science Series*. Boca Raton, London & New York: CRC Press Taylor & Francis Group. Informatics.

Heckl, Maria, et al. 1988, 1995. *Dictionary of Science & Technology, The Wordsworth*. Dictionary of Science & Technology.

Helmholtz, Hermann 1885. *On the sensations of Tone, as a Physiological basis for the Theory of Music*. Translated by Alexander J. Ellis. New York, USA: Dover Publications, INC. Music and Physics. Original edition, 1885. Reprint, 1954 by Dover.

Keene, Sonya E. 1989. *Object-Oriented Programming in Common Lisp, A Programmer's Guide to CLOS*. USA: Addison-Wesley. Informatics.

Lauwerier, Hans. 1987. *Fractals, Meetkundige figuren in eindeloze herhaling*. Amsterdam, The Netherlands: Aramith Uitgevers Amsterdam. Informatics.

Lawless, Jo A., and Molly M. Miller. 1991. *Understanding CLOS, The Common Lisp Object System*: Digital Press. Computer Science.

Loy, Gareth. 2006. *Musimathics, the mathematical foundations of music, volume 1*. 2 vols. Vol. 1. Cambridge, Massachusetts, London, Engeland: The MIT Press. Music, Mathematics and Informatics.

Minsky, Marvin, et al. 1992. *Understanding Music with AI: Perspectives on Music cognition*. Cambridge, Menlo Park, London: The AAAI Press/The MIT Press. AI, Cognition, Informatics and Music.

Moore, F. Richard. 1990. *Elements of computer music*. London: Prentice-Hall. Music and Informatics.

Morbach, Bernhard. 2004. *Die Musikwelt des Mittelalters*. 2 vols. Vol. 1. Kassel, Basel, London, New York, Prag: Bärenreiter-Verlag Karl Vötterle Gmbh & Co. KG, Kassel. Music History.

Nierhaus, Gerhard. 2009. *Algorithmic Composition, Paradigms of Automated Music Generation*. Wien (Viena): Springer-Verlag. Algorithmic Composition, Music and Informatics.

Oppenheim, Alan V., and Alan S Willsky. 1983. *Signals and Systems*. Edited by Alan V. Oppenheim. 19 vols, *Prentice-Hall Signal Processing series*. London: Prentice-Hall International Inc. Systems & Digital Signal Processing.

Perry, Marvin, et al. 1989. *Western Civilization. Ideas, Politics & Society - Third edition*. Boston, USA: Houghton Mifflin Company. History.

Provost, Foster, and Tom Fawcett. 2013. *Data Science for Business*. Informatics, Big Data.

Reichardt, Jasia. 1971. *Cybernetics, Art and Ideas*. London: New York Graphic Society; First edition (1971). Informatics, Cybernetics.

Risatti, Howard. 1975. *New Music Vocabulary, A Guide to Notational Signs for Contemporary Music*. Urbana, Chiga, London: University of Illinois Press. Music Notation Theory.

Roads, Curtis. 1996. *The computer music tutorial*: The MIT Press. Computer Music, Music, Mathematics and Informatics.

Rojas, Raúl. 1996. *Neural Networks a Systematic Introduction*. Berlin Heidelberg New York: Springer-Verlag. AI, Informatics.

Russell, Stuart, and Peter Norvig. 2012. *Künstlicher Intelligenz. Ein moderner Ansatz, 3., aktualisierte Auflage*. München, Germany: Pearson Deutschland GmbH. AI, Informatics.

Schoenberg, Arnold 1978, 1911. *Theory of Harmony*. 1 vols: Belmont Music Publishers, University of California Press. Music Theory.

Sevilla, Isidor von. 2008, 830. *Die Enzyklodädie des Isidor von Sevilla*. Translated by Dr. Lenelotte Möller: Marixverlag. Encyclopedia. Original edition, Oxford Classical Library 1911.

Shimazaki, H., and S. Shinomoto. 2007. "A method for selecting the bin size of a time histogram." *Neural Computing* 19 (6):1503 - 1527.

Steele, Guy L. 1990. *Common Lisp The Language (second edition)*. International edition: Digital Press. Informatics.

Stone, Kurt. 1980. *Music Notation in the Twentieth Century. A practical Guidebook*. New York, London: W.W. Norton & company. Music Notation Theory.

Taube, Heinrich K. 2004. *Notes from the Metalevel, Introduction to Algorithmic Music Composition*. Edited by Marc Leman. 6 vols, *Studies on New Music Research*. London UK.: Taylor & Francis Group. Music & Informatics.

Temperley, David. 2007. *Music and Probability*. Cambridge, Massachusets London, England: The MIT Press. Music & Informatics.

Waithe, Mary Ellen 1987. *Ancient women philosophers*. 4 vols. Vol. 1: Martinus Nijhoff Publishers, Dordrecht The Netherlands. Philosophy.

Walther, Elisabeth. 2000. "Max Bense's Informational and Semiotical Aesthetics." http://www.stuttgarter-schule.de/bense.html.

Wasserman, Philip D. . 1989. *Neural Computing, Theory and Practice*. New York: VNR - Van Nostrand Reinhold. Informatics.

Watson, Mark. 1991. *Common Lisp Modules. Artificial Intelligence in the Era of Neural Networks and Chaos Theory*. New York, Berlin, Heidelberg, London, Paris, Tokyo, Hong Kong and Barcelona: Springer-Verlag. Informatics.

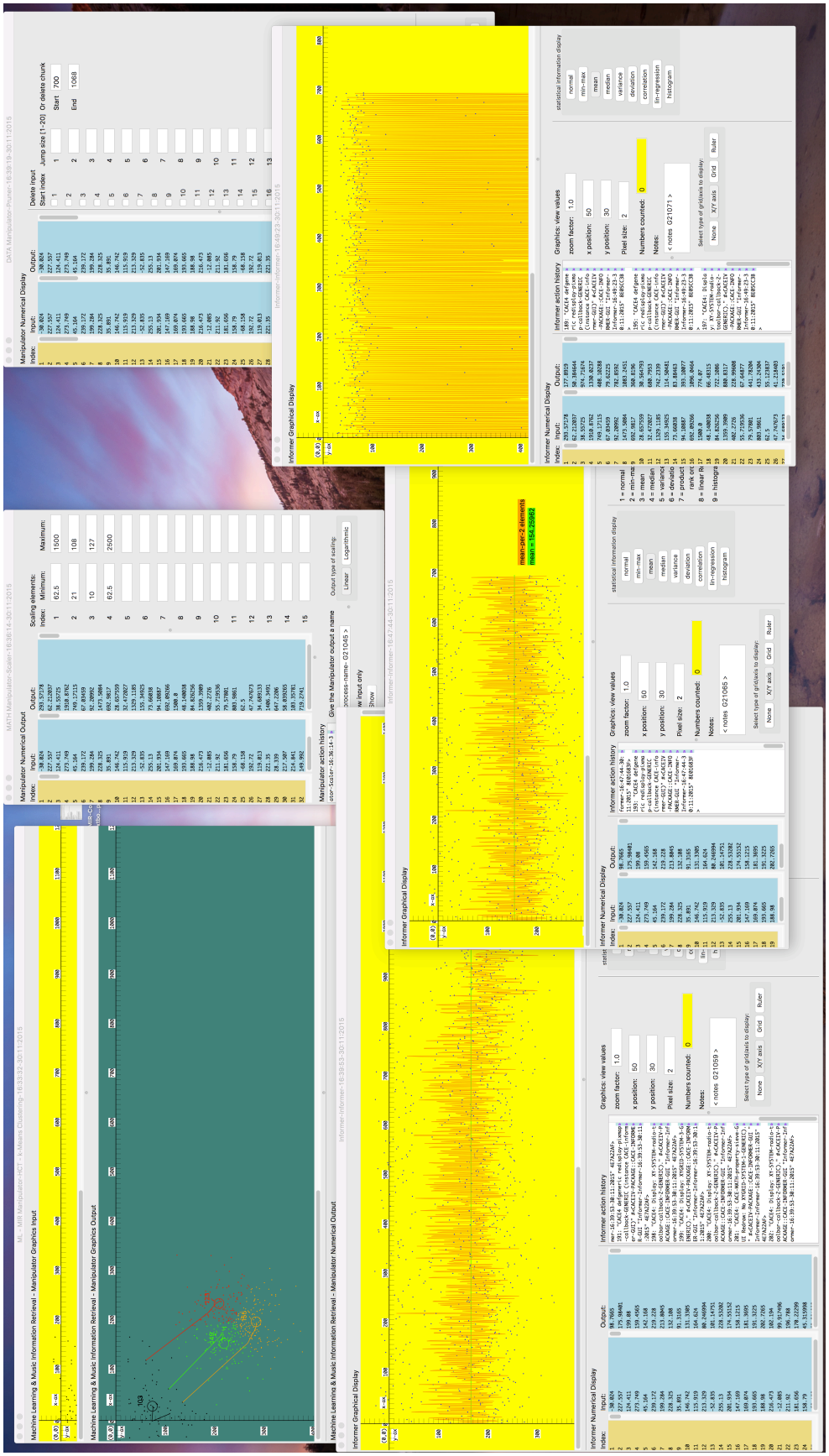
Winston, Patrick Henry. 1984. *Artificial Intelligence, second edition*.: Addison-Wesley Publising Company. AI + Informatics.

Xenakis, Iannis. 1971. *Formalized Music*. Bloomington, London: Indiana University Press. Music & Informatics.

Zwicky, Fritz. 1969. *Discovery, Invention, research -Through the Morphological Approach*.: The Macmillian Company. Philosophy.

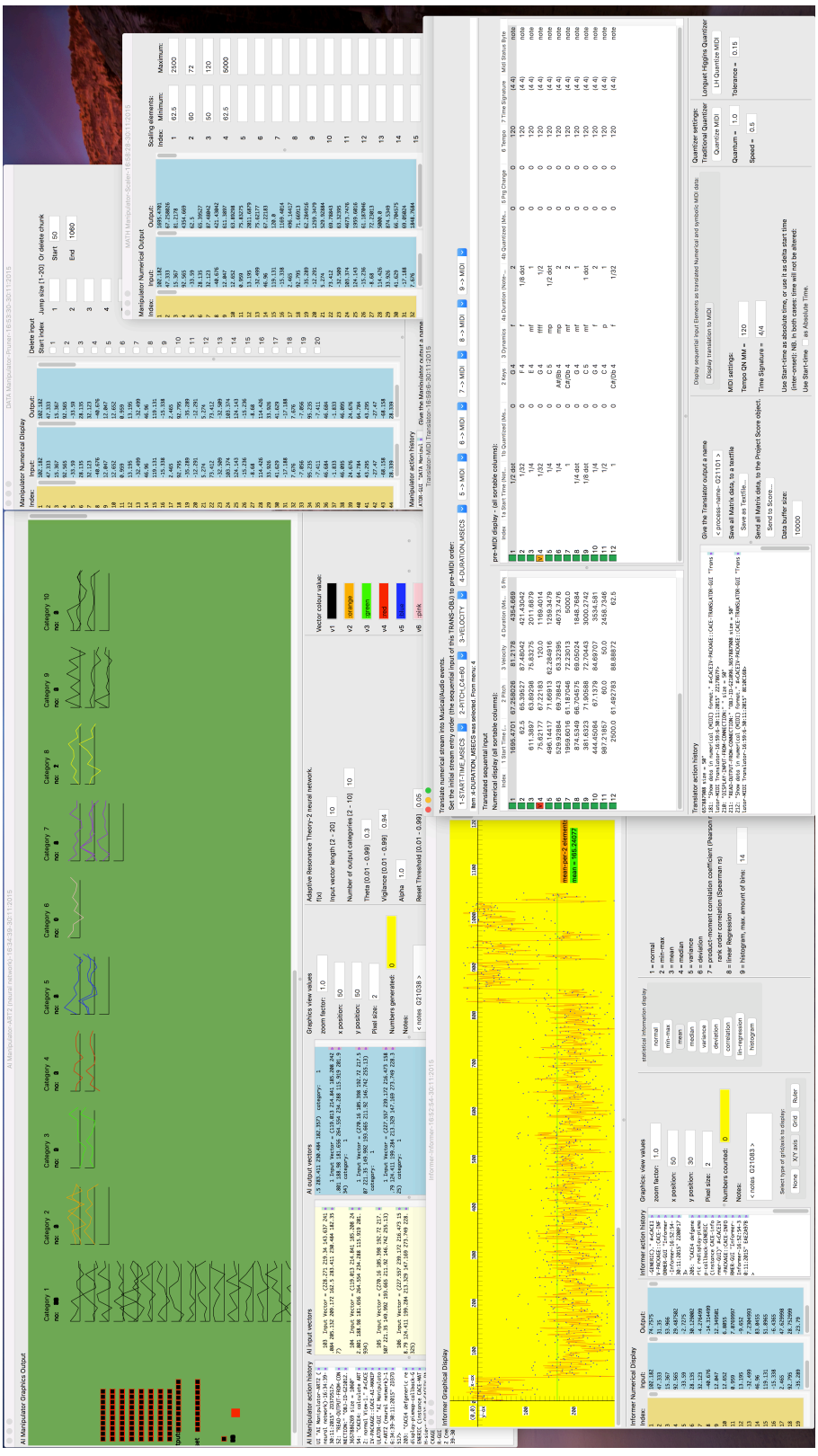
Appendix 1.1 CACE4 work session example 1.

A work session with the CACE4 *k*-Means object (left top position – dark green).



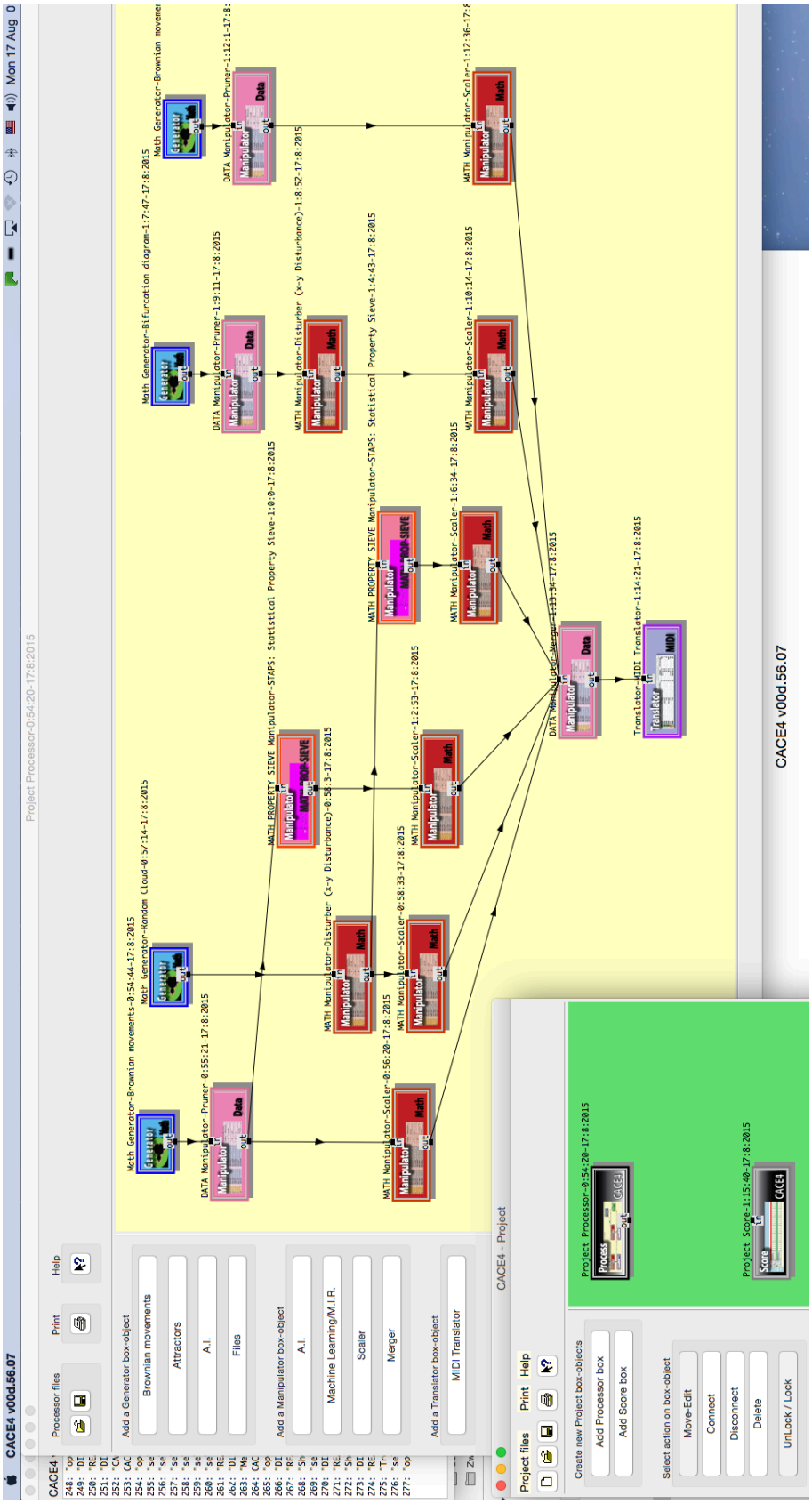
Appendix 1.2 CACE4 work session example 2.

Example of a work session with the CACE4 ART2 Neural Network object (top left). The CACE4 Informer object shows the output of the ART Neural Network (yellow view).



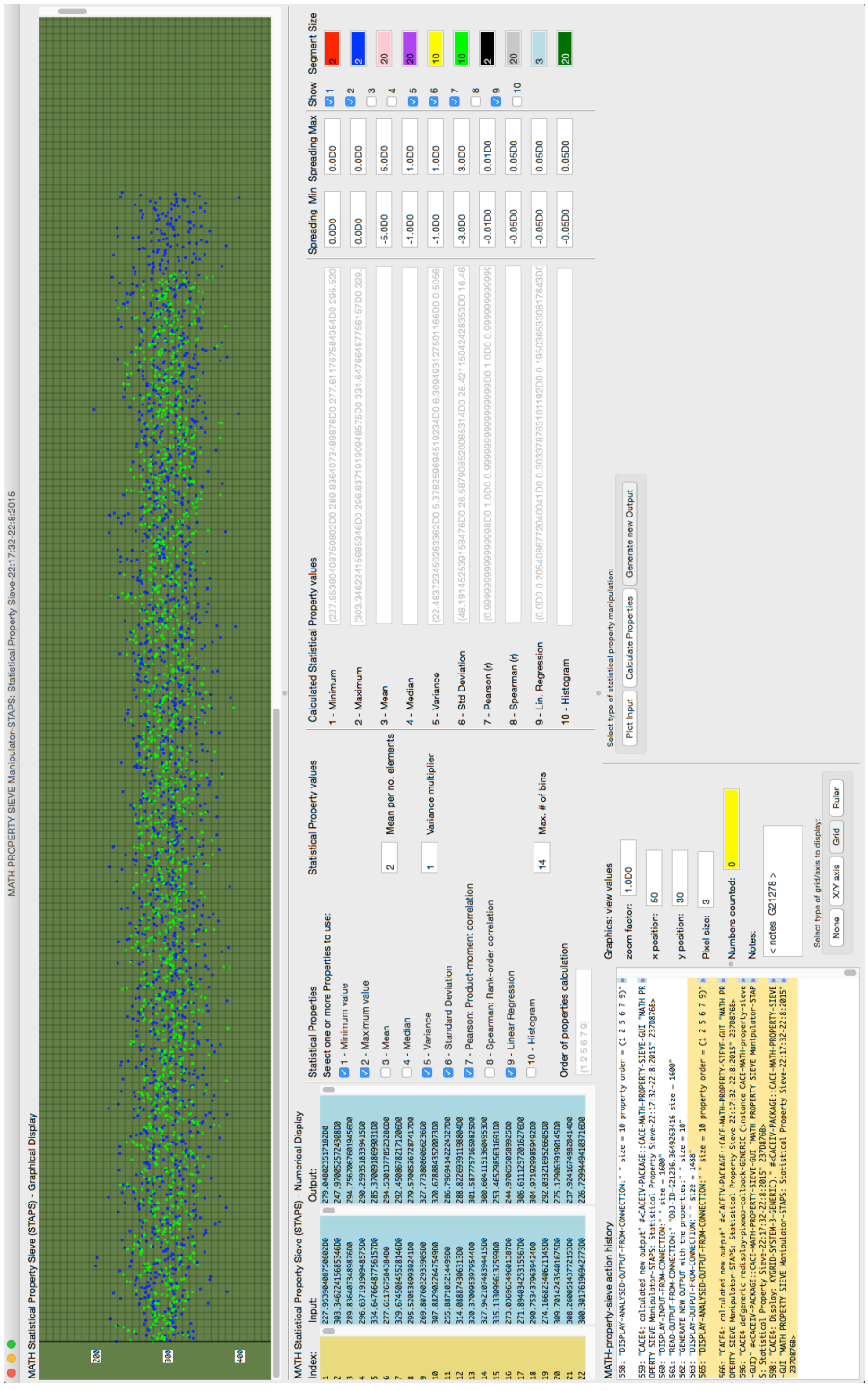
Appendix 1.3 Strategy *Scope*.

'Scope', showing both the Project and the Process window with the strategy displayed as a chain of connected CACE4 objects.



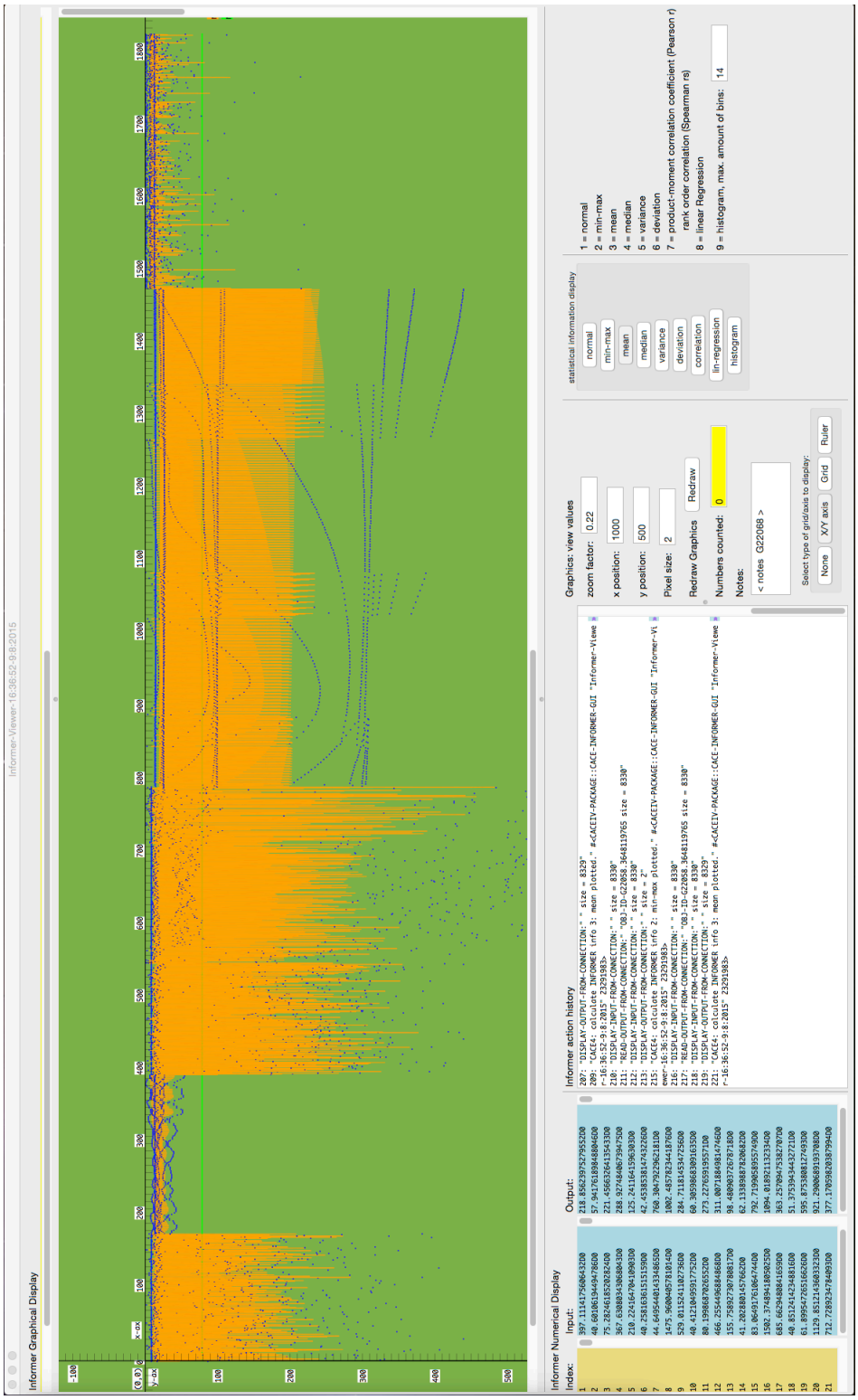
Appendix 1.4 STAPS GUI.

One of the used CACE4 STAPS Manipulator object GUI. It Shows the original input in blue, and the newly calculated output in light-green.

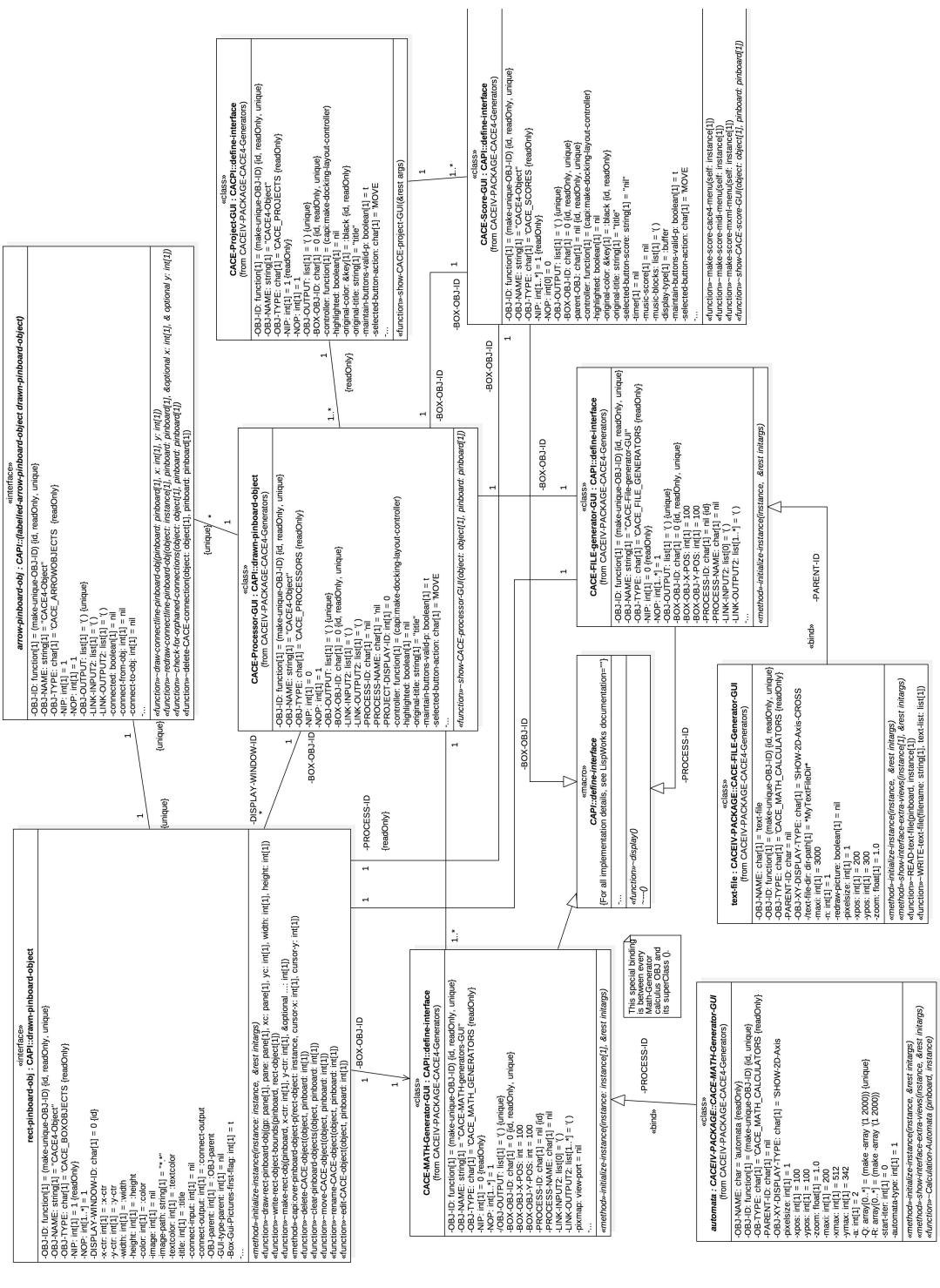


Appendix 1.5 Informer GUI.

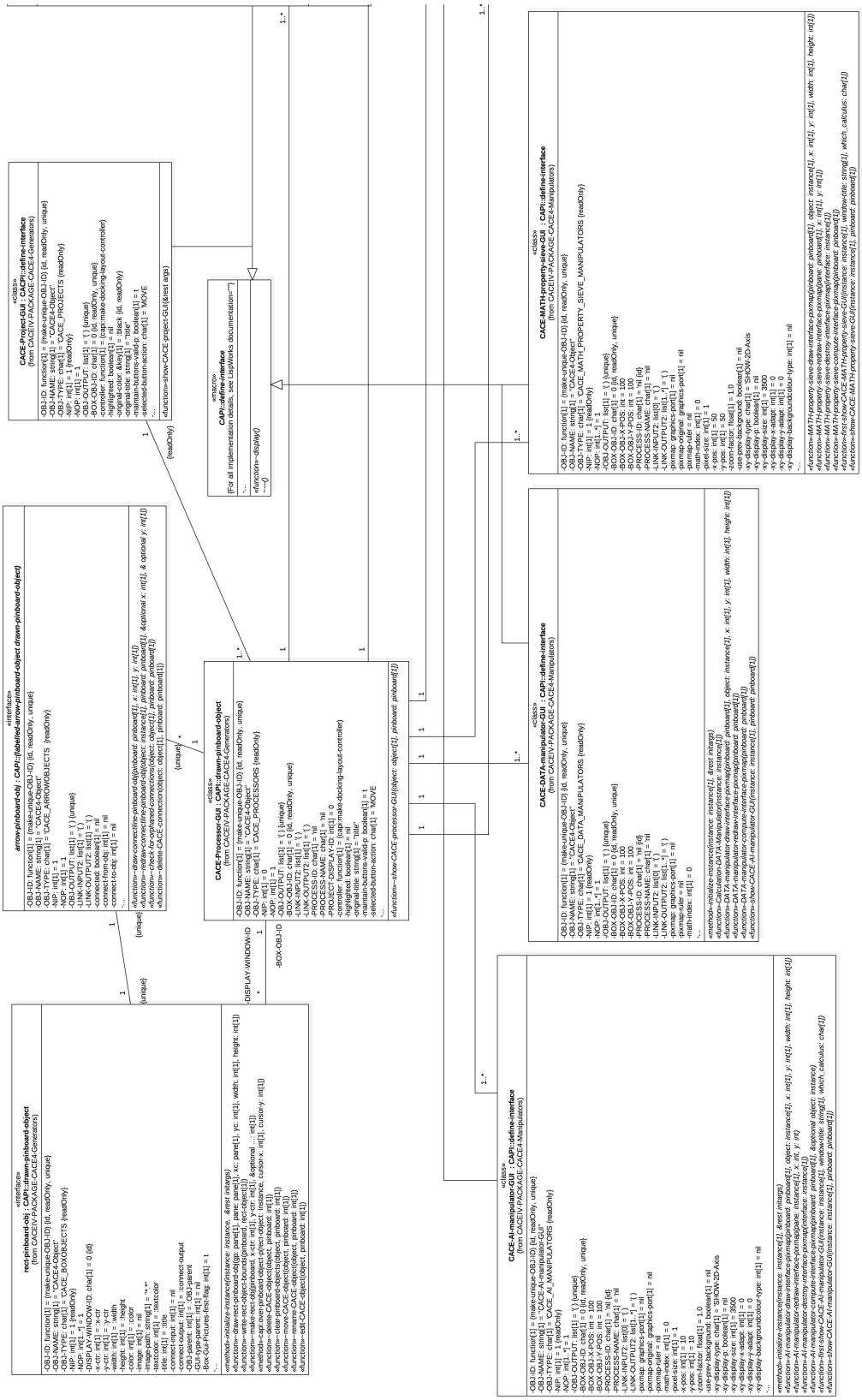
The display of the Informer GUI attached to the Merge object.



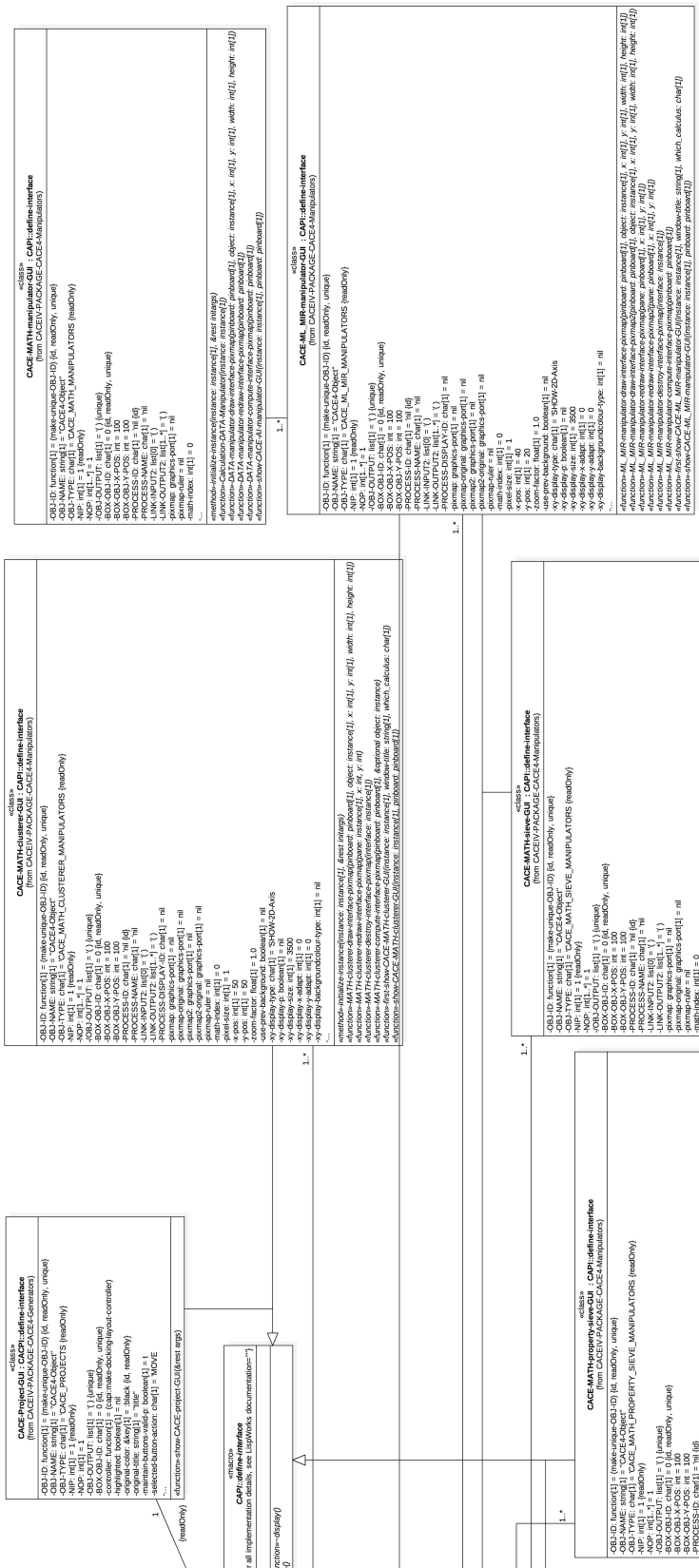
Appendix 2.1 CACE4 Generators: UML (2.x) Class Diagram.



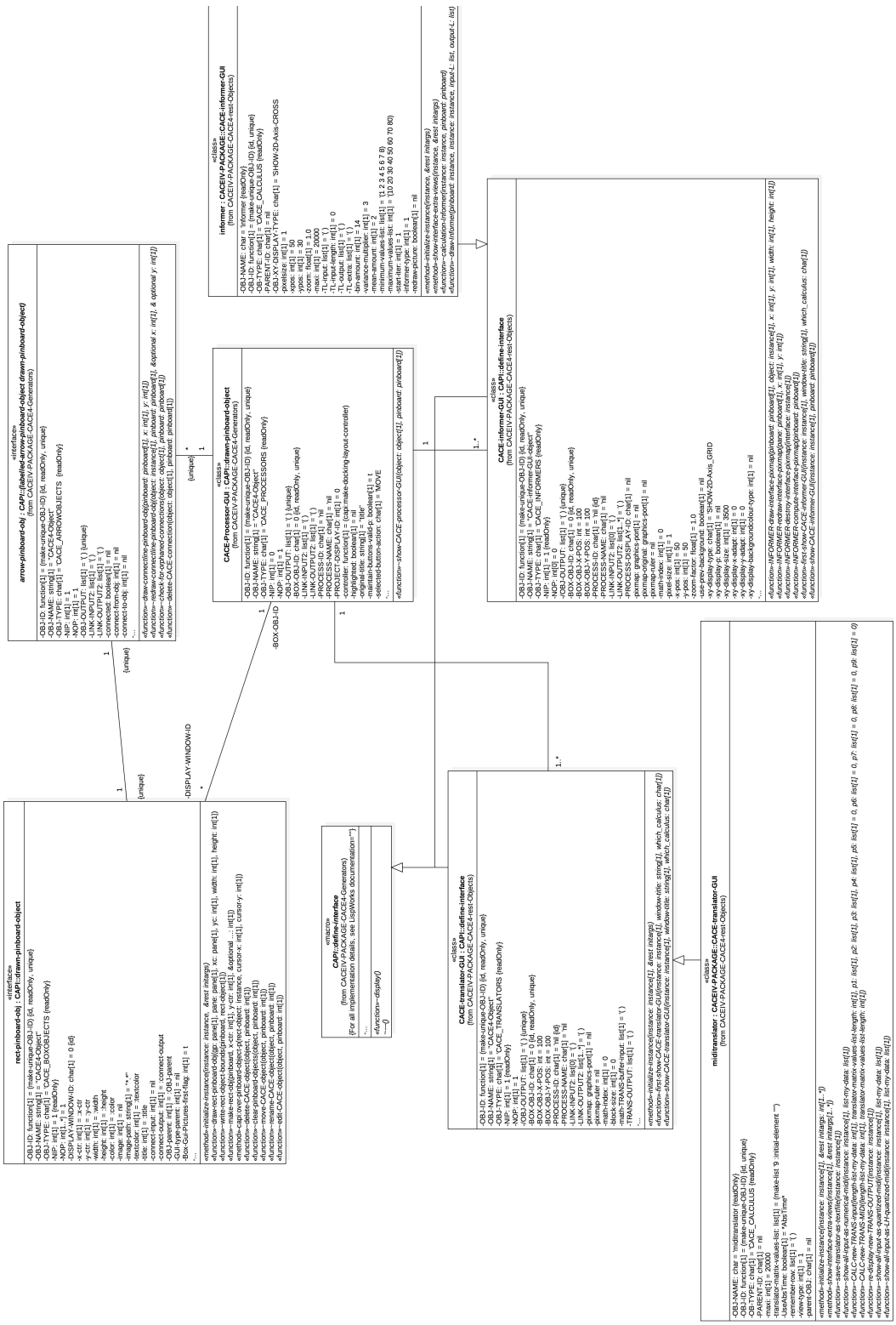
Appendix 2.2 CACE4 Manipulators, #1 of a UML (2.x) Class Diagram



Appendix 2.3 CACE4 Manipulators, #2 of a UML (2.x) Class Diagram



Appendix 2.4 CACE4 Miscellaneous objects in a UML (2.x) Class Diagram



Appendix 2.5 Class diagram of all CACE4-Generators objects #1

https://cloudstor.aarnet.edu.au/plus/index.php/apps/files?dir=%2FCACE4%20UML%20diagrams

Appendix 2.6 Class diagram of all CACE4-Generators objects #2

https://cloudstor.aarnet.edu.au/plus/index.php/apps/files?dir=%2FCACE4%20UML%20diagrams

Appendix 3: CACE4 Reference Manual

```
"-------------------------------------------------------------------------------------
       CACE4 REFERENCE MANUAL, Version 0.0.35 - July 7th, 2008 - June 28th, 2016.


     Reference Manual text: Copyright (c) 2016, Author: Mich├ƒl Koenders.
     CACE4 program design: Copyright (c) 1992 - 2016, (MK).
     Parts of the original coding was done in MCL 2.0 - 4.x And later adapted for LispWork.
        Copyright (c) 1987 - 2005, (MK).
     Lisp programming code: Copyright (c) 1992 - 2016, (MK).
     Lisp programming code in LispWork Personal Edition 5.x/6.x: Copyright (c) 2007 - 2016, (MK).
     After 01-11-2012: LispWorks 6.1 Professional Edition.
           01-06-2015: LispWorks 7.x Professional Edition.
     Additional code: Copyright (c) 1987--2008/2016 LispWorks Ltd. All rights reserved.
                      Copyright (c) 1990 This program contains software written by Mark Watson (ART2-
                                         module)
                                    2008 EMALGO Barry Fishman & Pascal Bourguignon

     Email: michelk@wxs.nl - URL: http://home.planet.nl/~michelk


     Change History: 070708 - started this reference.
               0.0.01              080708 - started the chapters.
               0.0.02              100708 - added text chapters 1, 2 & 3.
               0.0.03              110708 - put all the text in a separate .txt file,
                                            and changed the overall Layout.
                                            + several corrections in the text.
               0.0.04              170708 - added text chapter 2 & 3 and started the Index.
               0.0.05              060808 - added text chapter 2. + some indexing.
               0.0.06              130808 - added text chapter 2. + lots of indexing.
               0.0.07              150808 - added text chapter 1. + some indexing.
               0.0.08              220808 - added text chapter 1. + - HISTORY OF CAC.
               0.0.09              160309 - changed the header.
               0.0.10              170309 - changed the name of the app into CACE4.
               0.0.11              210309 - added text chapter 3 & 4 & the Index.
               0.0.12              270709 - added text chapter 3 & 4 & the Index.
               0.0.13              120412 - updated all dates and corrected some spelling mismatch.
               0.0.14              140612 - added text chapter 3.
               0.0.15              040812 - added text chapter 5.
               0.0.16              270912 - added text chapter 6 & changed some text in chapter 1 & 3.
               0.0.17              090113 - updated dates to 2013.
               0.0.18              170113 - added License text.
               0.0.19              170513 - changed text chapter 1 and added text chapter 3 & 5 & the
                                            Index.
               0.0.20              260513 - added text chapter 3.
               0.0.21              250214 - changed text.
               0.0.22              230715 - updated dates to 2015.
               0.0.23              270715 - adding text.
               0.0.24              300715 - changing text: CACE4 COS2 (v01b.02.14.18  )
               0.0.25              060815 - changing text.
               0.0.26              070815 - added text chapter 6. (and changed chapter 6 to chapter 7.)
               0.0.27              171115 - added copyright text.
               0.0.28              191115 - adding and changing text.
               0.0.28              231115 - adding and changing text (chapter 2 & 3).
               0.0.29              081215 - adding and changing text (chapter 2 & 3).
               0.0.30              091215 - adding and changing text (chapter 2 & 3).
               0.0.31              260416 - updated dates to 2016.
               0.0.32              020516 - changing text.
               0.0.33              020616 - changing text.
               0.0.34              060616 - changing text.
               0.0.35              280616 - changing text.



     -------------------------------------------------------------------------------------
     REFERENCE MANUAL:


     Chapter 1  - INTRODUCTION TO CACE4.

               Computer Aided Composition Environment written in ANSI Common
               Lisp, (version: LispWorks 7.0.0 (32-bit intel) - Personal edition) + CLOS.

               It is a frame work application based on an Object System called:
               CACE4 Object System & Modelling Organising Shell (COSMOS2 (v2.0))
               So it's an extendable music composition environment where generative processes as fractal,
               attractor and chaotic calculations, but also file input
               (text, SPEAR and Standard MIDI files)
```

Tendency masks.
Mandelbrot 1.
Mandelbrot 2.
Random Cloud.

3.1.2 - Attractor Generators.
Attractors:
Henon attractor type 1.
Henon attractor type 2.
R├╢ssler attractor.
Lorenz attractor.

3.1.3 - A.I. Generators.
For now not available (08-12-15, Michel Koenders).

3.1.4 - File Input Generators
There are for now, 3 types of file who can be used: Text files and Spear partials text files (NB take care to use the right output file format from Spear┬ -> Partials.

3.1.4.1 - Textfiles.
In the textiles the data news to be order in x.y pairs
(as a type of coordinates),
so they can be plotted in a x-y (2 dimensional) grid.

3.1.4.2 - Spear partials text file.
Be sure to export the data in Spear in the right format as a partials text file.
This can be found by export-type.
The Spear data can be viewed in 3 ways: as a spectrum, as a MIDI (point) translation of the spectrum (partials), and as a MIDI (line) translation + possibility of changing
the duration by multiplying with a factor as well.

3.1.4.3 - Standard MIDI file.

3.2 - The Manipulators objects [box-object]
A.I. Manipulator - Machine learning - Mathematics Manipulator - Sorting Manipulator.

- Manipulators:

A.I.:	ART2 (Adaptive Resonance Theory 2, Neural Network).
Machine Learning/MIR:	k-Means. (Hierarchical Cluster Techniques) Expectation Maximization. (HCT)
Mathematical manipulators:	Correlator. CLUS: Clusterer. Disturber (x-y Disturbance). Scaler. STAM: STAtistical Manipulation. STAPS: STAtistical Property Sieve.
Data manipulators:	Merger. Pruner. Sorter. Splitter.

3.3 - The Translator object [box-object]
Add a MIDI-Translator.
This translation unit translates any input to MIDI note format.

- Quantizing
Quantize the Translator output.

3.4 - The Informer/Viewer object [box-object]
Add Informer/Viewer and attach it to any other CACE4 Generator, Manipulator and Informer Object.

3.5 - Designing a strategy for connecting the objects.
Setting up a strategy to reach your goal is the core of the use of CACE4. It really depends on the results looking for, and of the complexity of the problem you want to tackle.
Take your time and experiment with the order of the objects. Scaling it to the right proportions, can help as well. Also deleting certain items in the data stream (e.g. x=x+1 values) can clean up the results as well. This process is called - data massaging,

are available and can be used to generate a sequence of numerical output.
Manipulation in the area of statistics, A.I. (ART2) and several (M)IR modules
(EM and k-means)
are available for processing the sequences.
And at the end of all processing the sequences are mapped to obtain
musical material which output can be directed to a Standard MIDI File.

- HISTORY OF CAC(E).
CAC(E) started somewhere in the beginning of 1992.
At the end of the summer of 1994 an early version of CAC II was ready for use.
Spear & Shield (piano + computer, Max+ISPW - 1994) was the first
composition (created during my stay at the Centrum fur Kunst und Medientechnologie
(ZKM, Centre for Art and Mediatechnology, Karlsruhe Germany) which was composed in the
early version of the CAC(E) II environment.

- Scope (2016): percussion and computer is the latest in a long row of algorithmical
compositions

Chapter 2 - STARTING A CACE PROJECT.

After starting the application a blank CAE4: Project window and a history window will present itself.
In the project window we can see, on the left side of the window a list of
available buttons:

- <Add Processor box> [button]
  Adding a Processor object at the right side of the CACE4 - Project window and
  start working with it.
- <Add Score box> [button]
  Adding a Score object at the Project window. (NB This option should only be used
  - for the moment - after a Translator box has send the output to the score object.
  (for details take a look at the score box object). So this should be done at the
  final stages of the project.

- <Move-Edit> [button] select for editing (by double clicking) or moving around (click
  and drag) of the selected Project Processor box or the Score box.
- <Connect> [button] can be used to connect several object boxes.
  (NB for now do not connect the Project Processor to the Score box).
- <Disconnect> [button] select this one and youΓÇÖre able to disconnect the selected
  box-object.
  from the other box-objects.
- <Delete> [button] select it for deleting the selected box object.

- <UnLock/Lock> [button] Lock or Unlock the window box display and the action on
  the box-objects.

IMPORTANT: First select an action button (left hand side window) then a (Black) Processor
box-object (left down side window).
You need at least one project object and (later) one score object in your project window.
You can do so by using <Add Processor box> [button].
(And use later: <add Score box> [button].

By making all the connections in the processor box-object the created data after
translation by a Translator box is available to the connected Score-box-object.
You can make use of several Processor-objects connected to one Score-box-object.

=> NB. Only one Score-box-object is allowed in the project - for now - (020616 mk).
-> although it is possible to have several Project Processor boxes in your CACE4
  Project, itΓÇÖs for now only possible to use one Score box-object).
Now you can also edit the object, by selecting the - <Move-Edit> [button] -
on the left side of the project-window and click on the Processor-object [box-object].

Chapter 3 - WORKING WITH A CACE4 PROCESSOR OBJECT.

Keywords:
  SELECTING and CONNECTING the different CACE4 Processor objects.
After adding one of the 4 basic building blocks:
GENERATORS, MANIPULATOR(S), TRANSLATOR(S) or INFORMER/Viewer(s), start

working with them.

After creating an empty Process Object (How? See Chapter 2),
double click and it will open.

In the newly created Process window we can see, on the left side of the window (bottom) a list of available buttons (NB they share the same functionality with the project window):

- <Move-Edit> [button] Select for editing (by double clicking) or moving around (click and drag)of the selected CACE4 Processor Object-box.

- <Connect> [button] Select can be used to connect several object boxes. Always connect the last one with the next to create a connection. When a connection has been established a black arrow appears between the two now connected objects. Indicating the direction (of data) of the CACE4 Object chain.

- <Disconnect> [button] Select this one and youΓÇÖre able to disconnect the selected box-object from the other box-objects.

- <Delete> [button] Select for deleting one of the selected CACE4 Processor Objects.

- <UnLock/Lock> [button] Lock or Unlock the window box display and the action on the box-objects.

Now make a selection from one of the following Generator box-objects:
- <Fractals> [drop-down menu] -> 3.1.1 - Fractal Generators.

- <Attractors> [drop-down menu] -> 3.1.2 - Attractor Generators.

- <A.I.> [drop-down menu] -> 3.1.3 - for now no entries (MK, 09-12-15).

- <Files> [drop-down menu] -> 3.1.4 - File Input Generators

Now we select one of the Manipulator box-objects and attach it to the previous selected Generator box-object.
Add a Manipulator box-object:
- <A.I.> [drop-down menu]

- <Machine Learning/M.I.R> [drop-down menu]

- <Mathematical Manipulator> [drop-down menu]

- <Data Manipulator> [drop-down menu]

After adding as a last object in the chain a CACE4 Scaler Object for scaling the data to MIDI ranges
Add a Translator box-object:
- <(MIDI) Translator.> [drop-down menu]

Add a Informer box-object:
- <informer-viewer.> [drop-down menu]

3.1 - The Generator objects [box-object]

Fractal Generators - Attractor Generators - A.I. Generators - File Input Generators.

3.1.1 - Fractal Generators.

Fractals:
Automaton.
Bifurcation diagram.
Mira.
Julia.
Iterated Function System (IFS).
Brownian movements.
Linear Congruential method.
Chaos on Torus.

and has to be done with great care otherwise we will influence the results as calculated.
Take also time when entering parameter values from the objects as well. The results will differ. For showing/plotting
result and to do some (only visual output) statistics with it. This really helps in understanding the results. Remember that not all our Information Retrieval (k-means and EM) or A.I. (ART2) objects can solve the problems, because they are not well suited to the problem.

Chapter 4 - WORKING WITH THE CACE4 SCORE OBJECT.

- Collecting data in the Score object.
  After sending data from the Translator to the Score it appears in the top, inside a red box. Select it and Drag it downward to the Tracks and drop it on Track #1.

- Save score to Standard MIDI file.

- (Save score to MusicXML file).

-

Chapter 5 - IMPLEMENTED CACE4 OBJECTS AND ALGORITHMS.

List of all 36 objects (and algorithms) implemented so far; date: 27th of July 2015.

CACE4 PROCESSOR OBJECTS:

- Generators:

Fractals:	Automaton. Bifurcation diagram. Mira. Julia. Iterated Function System (IFS). Brownian movements. Linear Congruential method. Chaos on Torus. Tendency masks. Mandelbrot 1. Mandelbrot 2. (Random) Cloud.
Attractors:	Henon type 1. Henon type 2. R╪ssler attractor. Lorenz attractor.
A.I.:	-
Files (input):	Text files. Spear partials text file. Standard MIDI file.

- Manipulators:

A.I.:	ART2 (Adaptive Resonance Theory 2, Neural Network).
Machine Learning/MIR:	k-Means. (Hierarchical Cluster Techniques) EM (Expectation Maximization). (HCT)
Mathematical manipulators:	Correlator. CLUS: Clusterer. Disturber (x-y Disturbance). Scaler. STAM: STAtistical Manipulation. STAPS: STAtistical Property Sieve.
Data manipulators:	Merger. Pruner. Sorter. Splitter.

- Translators: MIDI Translator.

- Informers: Informer/Viewer.

CACE4 PROJECT OBJECTS.
- Processor Object.
The Processor objects holds all CACE4 Processor Objects.
See above objects for details about using thoseCACE4 Processor Objects.

- Score Object.
Save single track as a std MIDI file.

Chapter 6 - Preference Panel - THE CACE4 APPLICATION.

- Use of the CACE4 preferences panel.

- CACE processes:
CACE4 COS&MOS2 (v2.0) verbose.
CACE4 INPUT verbose.
CACE4 OUTPUT verbose.
CACE4 PROC verbose.
CACE4 verbose.

- Numerical Input/Output columns display range.
The main purpose for this preference is to be used to speedup initial calculations.

Chapter 7 - ABOUT COS - THE CACE4 OBJECT SYSTEM.

In the preference panel of CACE4 you can turn COS verbose on/off.

CACE4 COS2 (v01b.02.14.18) : set-input-link

if object type == GENERATOR_OBJ then 0 connections are possible. (Generators have no inputs!).
if object type == FILE_GENERATOR_OBJ then 0 connections are possible. (File Generators have no inputs!).
if object type == MATH_MANIPULATOR_OBJ then 1 input connection is possible.
if object type == AI_MANIPULATOR_OBJ then 1 input connection is possible.
if object type == DATA_MANIPULATOR_OBJ then 1 or more input connections are possible.
if object type == ML_MIR_MANIPULATOR_OBJ then 1 input connection is possible.
if object type == TRANSLATOR_OBJ then 1 input connection is possible.
if object type == INFORMER_OBJ then 1 input connection is possible.
NB. 0,1 or multiple input connections can exist.

CACE4 COS2 (v01b.02.14.18): set-output-link

if object type == GENERATOR_OBJ then 1 or more output connections are possible.
if object type == FILE_GENERATOR_OBJ then 1 or more output connections are possible.
if object type == MATH_MANIPULATOR_OBJ then 1 or more output connection are possible.
if object type == AI_MANIPULATOR_OBJ then 1 or more output connections are possible.
if object type == DATA_MANIPULATOR_OBJ then 1 or more output connections are possible.
if object type == ML_MIR_MANIPULATOR_OBJ then 1 or more output connections are possible.
if object type == TRANSLATOR_OBJ then no connection is possible. (Output goes to ScoreObj!).
if object type == INFORMER_OBJ then no connection is possible. (Informers have no outputs!).
NB. All outputs are multiple connections.

Index Chapters

LICENSE

CACE4 - Copyright (c) Michèl Koenders, 2013 - 2016 - michelk@wxs.nl

---

"

Appendix 4: CACE4 MIDI and Audio Reference Table

All CACE4 timing MIDI Notes calculations are based on: tempo = (quarter note) MM 120,
Time-signature: 4/4. All can be changed in the TRANSLATOR-OBJ.

Note name	MIDI Number	frequency Hz.	period ms.	remarks
C-1	0	8.176	122.32	
C#-1 / Db-1	1	8.662	115.44	
D-1	2	9.177	108.96	
D#-1 / Eb-1	3	9.72275	102.84	
E-1	4	10.30075	97.08	
F-1	5	10.9135	91.64	
F#-1 / Gb-1	6	11.56225	86.48	
G-1	7	12.24975	81.64	
G#-1 / Ab-1	8	12.97825	77.04	
A-1	9	13.75	72.72	
A#-1 / Bb-1	10	14.5675	68.64	
B-1	11	15.434	64.8	
C0	12	16.352	61.16	
C#0 / Db0	13	17.324	57.72	
D0	14	18.354	54. 42	
E0	16	20.6015	48.54	
F0	17	21.827	45.82	
F#0 / Gb0	18	23.1245	43.24	
G0	19	24.4995	40.82	
G#0 / Ab0	20	25.9565	38.52	
A0	21	27.5	36.36	
A#0 / Bb0	22	29.135	34.32	
B0	23	30.868	32.4	
C1	24	32.703	30.58	
C#1 / Db1	25	34.648	28.86	
D1	26	36.708	27.24	
D#1 / Eb1	27	38.891	25.71	
E1	28	41.203	24.27	
F1	29	43.654	22.91	
F#1 / Gb1	30	46.249	21.62	
G1	31	48.999	20.41	
G#1 / Ab1	32	51.913	19.26	
A1	33	55	18.18	
A#1 / Bb1	34	58.27	17.16	
B1	35	61.735	16.2	
C2	36	65.406	15.29	
C#2 / Db2	37	69.296	14.29	
D2	38	73.416	13.62	
D#2 / Eb2	39	77.782	12.86	
E2	40	82.407	12.13	
F2	41	87.307	11.45	
F#2 / Gb2	42	92.499	10.81	
G2	43	97.999	10.2	
G#2 / Ab2	44	103.83	9.631	
A2	45	110	9.091	
A#2 / Bb2	46	116.54	8.581	
B2	47	123.47	8.099	
C3	48	130.81	7.645	
C#3 / Db3	49	138.59	7.216	
D3	50	146.83	6.811	
D#3 / Eb3	51	155.56	6.428	
E3	52	164.81	6.068	
F3	53	174.61	5.727	
F#3 / Gb3	54	185	5.405	
G3	55	196	5.102	
G#3 / Ab3	56	207.65	4.816	
A3	57	220	4.545	
A#3 / Bb3	58	233.08	4.29	
B3	59	246.94	4.05	
C4	60	261.63	3.822	***Central C***
C#4 / Db4	61	277.18	3.608	
D4	62	293.67	3.405	
D#4 / Eb4	63	311.13	3.214	
E4	64	329.63	3.034	
F4	65	349.23	2.863	
F#4 / Gb4	66	369.99	2.703	
G4	67	392	2.551	
G#4 / Ab4	68	415.3	2.408	
A4	69	440	2.273	***a=440***

Note durations	in 1/1000 sec	msecs	
1/128	15.625	15.6	
	19.53125	19.5	
1/128 dot	23.4375	23.4	
	27.34375	27.3	
1/64	31.25	31.3	
	39.0625	39.1	
1/64 dot	46.875	46.9	
	54.6875	54.7	
1/32	62.5	62.5	
	78.125	78.1	
1/32 dot	93.75	93.8	
	109.375	109.4	
	156.25	156.3	
1/16 dot 1	87.5	187.5	
	218.75	218.8	
1/8	250	250	
	312.5	312.5	
1/8 dot	375	375	
	437.5	437.5	
1/4	500	500	
	625	625	
1/4 dot	750	750	
	875	875	
1/2	1000	1000	
	1250	1250	
1/2 dot	1500	1500	
	1750	1750	
1	2000	2000	
	2500	2500	
1 dot	3000	3000	
	3500	3500	
2	4000	4000	
	5000	5000	
3	6000	6000	
4	8000	8000	
5	10000	10000	***Maximum!***

CACE4 dB's, Amplitudes to MIDI Volumes/Velocity's.

0.0 dB	= 1.0	= 127
-6.0 dB	= 0.5	= 121
-12.0 dB	= 0.25	= 115
-18.0 dB	= 0.125	= 109
-24.0 dB	= 0.625	= 103
-30.0 dB	= 0.03125	= 97
-36.0 dB	= 0.015625	= 91
-42.0 dB	= 0.0078125	= 85
-48.0 dB	= 0.00390625	= 79
-54.0 dB	= 0.001953125	= 73
-60.0 dB	= 0.0009765625	= 67
-66.0 dB	= 0.00048828125	= 61
-72.0 dB	= 0.000244140625	= 55
-78.0 dB	= 0.0001220703125	= 49
-84.0 dB	= 0.00006103515625	= 43
-90.0 dB	= 0.00003051757813	= 37
-96.0 dB	= 0.00001525878907	= 31
-102.0 dB	= 0.00000762939454	= 25
-108.0 dB	= 0.00000381469727	= 19
-114.0 dB	= 0.00000190734864	= 13
-120.0 dB	= 0.00000095367432	= 7
-126.0 dB	= 0.00000047683716	= 1
-132.0 dB	= 0.00000023841858	= 0
-138.0 dB	= 0.00000011920929	= 0
-144.0 dB	= 0.00000005960465	= 0

Note name	MIDI Number	frequency Hz.	period ms.	remarks
A#4 / Bb4	70	466.16	2.145	
B4	71	493.88	2.025	
C5	72	523.25	1.91	
C#5 / Db5	73	554.37	1.804	
D5	74	587.33	1.703	
D#5 / Eb5	75	622.25	1.607	
E5	76	659.26	1.517	
F5	77	698.46	1.432	
F#5 / Gb5	78	739.99	1.351	
G5	79	783.99	1.276	
G#5 / Ab5	80	830.61	1.204	
A5	81	880	1.136	
A#5 / Bb5	82	932.33	1.073	
B5	83	987.77	1.012	
C6	84	1046.5	0.9556	
C#6 / Db6	85	1108.7	0.902	
D6	86	1174.7	0.8513	
D#6 / Eb6	87	1244.5	0.8034	
E6	88	1318.5	0.7584	
F6	89	1396.9	0.7159	
F#6 / Gb6	90	1480	0.6757	
G6	91	1568	0.6378	
G#6 / Ab6	92	1661.2	0.602	
A6	93	1760	0.5682	
A#6 / Bb6	94	1864.7	0.5363	
B6	95	1975.5	0.5062	
C7	96	2093	0.4778	
C#7 / Db7	97	2217.5	0.451	
D7	98	2349.3	0.4257	
D#7 / Eb7	99	2489	0.4018	
E7	100	2637	0.3792	
F7	101	2793	0.358	
F#7 / Gb7	102	2960	0.3378	
G7	103	3136	0.3189	
G#7 / Ab7	104	3322.4	0.301	
A7	105	3520	0.2841	
A#7 / Bb7	106	3729.3	0.2681	
B7	107	3951.1	0.2531	
C8	108	4186	0.2389	
C#8 / Db8	109	4435	0.2255	
D8	110	4698.6	0.21285	
D#8 / Eb8	111	4978	0.2009	
E8	112	5274	0.1896	
F8	113	5586	0.179	
F#8 / Gb8	114	5920	0.1689	
G8	115	6272	0.15945	
G#8 / Ab8	116	6644.8	0.1505	
A8	117	7040	0.14205	
A#8 / Bb8	118	7458.6	0.13405	
B8	119	7902.2	0.12655	
C9	120	8372	0.11945	
C#9 / Db9	121	8870	0.11275	
D9	122	9397.2	0.106425	
D#9 / Eb9	123	9956	0.10045	
E9	124	10548	0.0948	
F9	125	11172	0.0895	
F#9 / Gb9	126	11840	0.08445	
G9	127	12544	0.079725	
G#9/ Ab9	-	13289.6	0.07525	
A9	-	14080	0.071025	

300513 mk