A Model Based Systems Engineering Approach to Automated Music Arrangement

Jalal Possik, Charles Yaacoub, Simon Gorecki, Grégory Zacharewicz, Andrea d'Ambrogio

A MODEL BASED SYSTEMS ENGINEERING APPROACH TO AUTOMATED MUSIC ARRANGEMENT

Jalal Possik

ADERSIM
School of Administrative Studies
York University
Toronto, Ontario, M3J 1P3, CANADA
jpossik@yorku.ca

Charles Yaacoub

School of Engineering

Holy Spirit University of Kaslik (USEK)
Kaslik, 1200 Jounieh, LEBANON
charlesyaacoub@usek.edu.lb

Simon Gorecki

GREThA
University of Bordeaux
16 avenue Léon Duguit, 33608 Pessac, FRANCE
simon.gorecki@u-bordeaux.fr

Gregory Zacharewicz

Laboratoire des Sciences des Risques (LSR)
Institut Mines-Telecom (IMT) Mines Ales
30319 Alès, FRANCE
gregory.zacharewicz@mines-ales.fr

Andrea D'Ambrogio

Dept. of Enterprise Engineering
University of Rome Tor Vergata
Via del Politecnico 1, I-00133 Roma, ITALY
dambro@uniroma2.it

ABSTRACT

MBSE (Model Based Systems Engineering) is considered a valuable and effective approach not only in engineering domains, although it appears to bring a potential assistance to artistic visions and initiatives. The cross-domain capitalization is the key to further innovation and advances in technology. This paper describes an MBSE approach, applied to the development of a complex multi-component arrangement system, to generate an automated music arrangement for any melody, with a harmonically correct result. The user introduces the melody in the form of a MIDI file. The system analyzes the melody, detects the scale, then follows algorithms based on thorough theoretical and harmonic studies, to finally generate the arrangement based on the selected music genre (classical, jazz, pop, etc.). Compared to other existing systems in the market, the developed system largely reduces the dissonance between the melody and the arrangement, which makes the final arrangement a good accompaniment to the melody.

Keywords: model-based systems engineering (MBSE), automated music arranger, music generation, BPMN, MIDI parsing.

1 INTRODUCTION

The digitalization of concepts, ideas, and propositions introduced to satisfy needs that may arise in any social or business domains, contributes to enhance creativity and provide added value. When moving to a digital world, a major problem frequently identified is the transformation of human-related vision and senses into digitalized systems. The contribution in Zacharewicz et al. (2017) clearly identifies the opportunities provided by modeling as an effective approach to achieve such an objective. A model may represent a system, a structure, a phenomenon, or a concept. Researchers and engineers are increasingly using the *model-based systems engineering* (*MBSE*) approach to describe and control complex systems. Moreover, MBSE and models simulation techniques are widely applied to examine "what if" scenarios that help managers in decision making. In this paper, the MBSE approach is employed as a guide to the development and implementation of a complex *automated music arranger system* that organizes the music according to the needs or requirements of the composer, the group of performers, or the singer.

To compose a piece of music, the melody is created first, then the musical arrangement is produced. Depending on the style, the music arranger makes the arrangement, i.e., the arrangement of classical music is different from the arrangement of jazz music. The instruments used, the music rules, and the rhythm differ. A musical arrangement can be played by a band or an orchestra, although, it can also be obtained by using synthesizers or Virtual Studio Technology (VSTi) instruments. Musical arrangements always requires a human intervention; no machine, to our knowledge, has been able to generate a correct chord progression to accompany a certain melody. The arrangement process of a three-minute melody takes hours, and based on the chosen style, it might take days sometimes. The goal of the project described in this paper is to simplify the work of music arrangers by developing a multicomponent system that parses the MIDI melody file and converts it to a fully arranged music within seconds.

In summary, this project aims at fully automating the music arrangement process using a model-based approach, while respecting the rules of music harmony. After providing the melody to the developed system, the user chooses a set of parameters (musical genre, tempo, etc.). Then, the system analyzes the melody, associates different chord progressions with it, and finally performs the simulation process to select the appropriate chord progression that accompanies the melody.

2 STATE OF THE ART

Research efforts have been made to combine artistic activities with the current digital world revolution. Bonnin and Jannach (2014) presented a comparative evaluation of playlist generation methods based on historical data. Morris and Wainer (2012) introduced a mapping tool for the automated creation of music using cellular models. Whorley and Conklin (2016) investigates an improved sampling technique for music generation from statistical models of harmony. Nevertheless, most of the music generation research involves both the melody and the arrangement generation. When the melody is imposed by the user, the generation of arrangement becomes more difficult and complicated requiring deep theoretical and harmonic studies of the melody to detect the appropriate chords (for each beat/bar) on which an arrangement is built, based on the selected music genre (classical, jazz, pop, etc.).

Most of the existing arranger tools are non-automatic arrangers. In this kind of software, there is no scale detection, and subsequently, no automatic generation of chords. The user provides the chords he sees fit, and the arrangement will be generated according to these chords. These software tools are based only on user chords and arrangement styles provided by the arranger, without any theoretical analysis performed by the machine. There are, to our knowledge, only two tools for automatic arrangement, in addition to machine learning-based tools, but not based on theoretical and harmonic studies. This leads to wrong scale detection as well as dissonant chords with the melody. This section presents some of the existing arranger software, highlighting their ineffectiveness from a professional's point of view, which motivates our idea of developing an automatic arranger. For non-automatic arrangement, several software tools exist. For instance, "Easy Music Composer" (Biglobe 2017) and "Chord Composer" (kvraudio 2021). In such tools,

arrangement is generally obtained after three steps: loading the melody in MIDI file format, choosing the arrangement style, and manually providing the suitable chords for the loaded melody. Such tools might be convenient whenever the user's own chords are desired. On the other hand, "Onyx Arranger" (Jasmine Music Technology 2004) and "e-Xpressor" (Maruta 2005) allow the automated music arrangement after loading the melody as a MIDI file and choosing the desired style. With both tools, in many cases the arranged music has a lot of harmonic errors, which most of the time cause dissonance. Moreover, it is frequently observed that silence is suddenly introduced, i.e. the arrangement suddenly disappears. Furthermore, frequent chord repetition is also observed. The "Onyx Arranger" also suffers from a poor scale detection, which leads to complete dissonance throughout the melody. Recently, "AWS DeepComposer" (Amazon 2020) has emerged as a new arranger tool based on machine learning approaches. The user provides the tool with a simple melody which is sent to a cloud service provided by Amazon Web Services, where generative artificial intelligence is exploited not only to arrange the melody, but also to complete the melody with machine-generated music. The synthesized music includes guitar pieces, bass, synths and drums. Generative models are provided for four different genres: rock, pop, jazz, and classical, in addition to the user's own models that can be built. The main disadvantage of DeepComposer is that it highly relies on training length and datasets; larger datasets and extensive trainings often provide better results, at the expense of increased computational resources and costs. In addition, theoretical correctness is highly dependent on training data that require hundreds of musical pieces for each genre, which might not be convenient for users.

The current project opposes the artistic view on the one hand, and the technical view on the other hand. Some research efforts have been produced to bridge the gap between both different worlds, for instance, model-based reconciliation can be considered one of the important solutions that can be used (D'Ambrogio and Bocciarelli 2007). Model driven engineering methods have emerged to facilitate the trans-domain interoperability and support productive modeling. The first and most used one is the Model Driven Architecture (MDA) proposed by the Object Management Group (OMG 2013). MDA is a system design approach that introduces a significant degree of automation in the development of software systems. It provides a set of guidelines for structuring and transforming specifications, which are expressed as models. It is considered as an important solution in diverse research domains (Gianni, D'Ambrogio, and Tolk 2014), especially for the prediction, analysis, and description of complex systems (Possik et al. 2019). Nevertheless, model-based approaches have not been used in the music domain yet. In this paper, we present a model-based approach to drive the development and implementation of a complex multi-component music arrangement system. The Business Process Model and Notation (BPMN) is used to define the existing system components and their interaction. Papyrus, an integrated environment for editing UML (Unified Modeling Language) models and executing UML-based processes, is used to accomplish the orchestration between the developed components.

3 MATERIALS AND METHODS

MIDI (Musical Instrument Digital Interface) was developed in 1983 as a software and hardware design that enables the communication between synthesizers and computers. The MIDI communication protocol allows the data exchange between electronic musical instruments. The MIDI digital interfaces are designed to transmit control information, called messages or events, between devices. Unlike audio recordings, which capture sound waves, MIDI recordings capture MIDI messages, which in turn broadcast the sound to an external device (sound generator), such as a synthesizer. Using MIDI format, it is possible to apply non-destructive transformations, such as transposition or tempo modifications. Such transformations only modify the MIDI messages and not the sound quality of the recording. Using MIDI files allows the users to change many factors that look hard and sometimes impossible with audio files. In our project, the melody is supplied in MIDI format, as in other existing software solutions on the market. The first component developed for the music arranger system is a MIDI parser.

3.1 MIDI Parsing

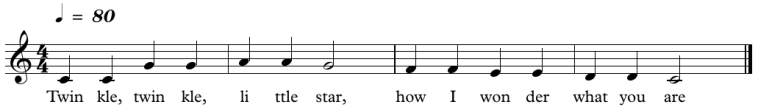
The MIDI file is a succession of blocks called chunks. Each chunk follows the pattern of the chunk header. A MIDI file contains several chunks that can be of different sizes. Figure 1 is an example of the music notation and the hexadecimal code resulted from the recorded melody of the popular nursery rhyme "twinkle twinkle little star" in MIDI format. The hexadecimal code starts with the MIDI track header (MThd), 0x4d 0x54 0x68 0x64, that marks the beginning of a MIDI file. The length in bytes of the rest of the chunk header, the MIDI type (0,1, or 2), the number of tracks, and the number of pulses per quarter note (PPQN) are found in the rest of the hexadecimal code of the first row (see Figure 1). 0x4d 0x54 0x72 0x6b of the MIDI track (MTrk) marks the beginning of the track. The tempo hexadecimal code starts with the byte 0xff that indicates the beginning of a meta message; the second byte (0x51) represents the tempo meta message. The third byte is the size of the tempo data (0x03). Hence, the following 3 bytes (0x0b71b0) represent the number of microseconds per beat. It is a 24-bit unsigned, big-endian integer that is equal to 750,000 microseconds per beat when converted to decimal. In all applications and synthesizers, the tempo is shown in Beats Per Minute (BPM). There are 60,000,000 microseconds in a minute. The formula used to calculate the tempo in BPM is: $\frac{60{,}000{,}000}{BPM} = microseconds\ by\ beat$

Therefore, the tempo used in the MIDI melody of Figure 1 is equivalent to: $\frac{60{,}000{,}000}{750{,}000} = 80\ BPM$

If no tempo meta messages exist in the MIDI file, 500,000 microseconds per beat (120 BPM) are considered as the default tempo of the song. The hexadecimal code 0xff marks the beginning of a meta message. The hexadecimal 0x58 marks the beginning of the time signature related section. This byte is followed by 5 bytes. The second byte represents the numerator of the time signature. In this case, the numerator is 4. The third byte represents the denominator as a negative power of 2; i.e., 2 signifies a quarter note, 3 signifies an eighth note, etc.). In this case (0x02), it represents a quarter note. The code 0xff 0x59 marks the beginning of the key signature section. This code is followed by 3 bytes. The second byte specifies the number of flats or sharps that identify the key signature. In the case of Figure 1, it is equal to 0x00, indicating that no flats or sharps exist in the key signature. The third byte specifies a major (0x00) or minor (0x01) key.

After the hexadecimal meta messages, the synthesizer starts reading the notes. 0x90 means that a note is turned on. 0x80 means that a note is turned off. Both hexadecimal bytes (0x90 and 0x80) are followed by two bytes. The first byte specifies the note and the second one specifies the velocity of the note, i.e., 0x90 0x3c 0x4c means that the note number 60 or C4 (0x3c) is turned on with a velocity equal to 76 (0x4c). This code is followed by 0x80 0x3c 0x00, which means that the same note (0x3c) is turned off with a velocity 0. Turned on notes should be turned off. Check Figure 1 for the detailed description of the hexadecimal bytes related to the notation process. The hexadecimal sequence 0xff 0x2f 0x00 marks the end of the track.

BPMN is a business process-modeling standard that offers a graphical notation based on a flowcharting technique. BPMN represents the end-to-end flow of a process. The Business Process Management Initiative (BPMI) developed the Business Process Modeling standard. In 2005, this group merged with the OMG (Object Management Group). In 2011, OMG released BPMN version 2.0 and changed the name of the standard to Business Process Model and Notation. The modeling language became more detailed by using a richer set of symbols and notations for business process diagrams. The main goal of BPMN is to deliver a standard notation easily readable by non-expert users. In the present work, we used BPMN to specify the proposed methodology and simplify the understanding of the integration and collaboration between discrete event simulators.



MThd	4d 54 68 64 00 00 00 06 00 00 00 01 00 80
MTrk	4d 54 72 6b 00 00 00 8c
Tempo	00 ff 51 03 0b 71 b0
Time Signature	00 ff 58 04 04 02 18 08
Key Signature	00 ff 59 02 00 00
Notes	00 **90 3c 4c** --------- Note ON (0x90): C4 (0x3c), velocity 76 (0x4c) 87 40 **80 3c 00** ----- Note OFF: C4, velocity 0 00 **90 3c 4c** --------- Note ON (0x90): C4 (0x3c), velocity 76 (0x4c) 87 40 **80 3c 00** ----- Note OFF: C4, velocity 0 00 **90 43 50** --------- Note ON (0x90): G4 (0x43), velocity 80 (0x50) 87 40 **80 43 00** ----- Note OFF: G4, velocity 0 00 **90 43 4e** --------- Note ON (0x90): G4 (0x43), velocity 78 (0x4e) 87 40 **80 43 00** ----- Note OFF: G4, velocity 0 00 **90 45 4e** --------- Note ON (0x90): A4 (0x45), velocity 78 (0x4e) 87 40 **80 45 00** ----- Note OFF: A4, velocity 0 00 **90 45 4d** --------- Note ON (0x90): A4 (0x45), velocity 77 (0x4d) 87 40 **80 45 00** ----- Note OFF: A4, velocity 0 00 **90 43 4b** --------- Note ON (0x90): G4 (0x43), velocity 75 (0x4b) 8F 00 **80 43 00** ----- Note OFF: G4, velocity 0 00 **90 41 4c** --------- Note ON (0x90): F4 (0x41), velocity 76 (0x4c) 87 40 **80 41 00** ----- Note OFF: F4, velocity 0 00 **90 41 4c** --------- Note ON (0x90): F4 (0x41), velocity 76 (0x4c) 87 40 **80 41 00** ----- Note OFF: F4, velocity 0 00 **90 40 4c** --------- Note ON (0x90): E4 (0x40), velocity 76 (0x4c) 87 40 **80 40 00** ----- Note OFF: E4, velocity 0 00 **90 40 4c** --------- Note ON (0x90): E4 (0x40), velocity 76 (0x4c) 87 40 **80 40 00** ----- Note OFF: E4, velocity 0 00 **90 3e 4b** --------- Note ON (0x90): D4 (0x3e), velocity 75 (0x4b) 87 40 **80 3e 00** ----- Note OFF: D4, velocity 0 00 **90 3e 4c** --------- Note ON (0x90): D4 (0x3e), velocity 76 (0x4c) 87 40 **80 3e 00** ----- Note OFF: D4, velocity 0 00 **90 3c 4c** --------- Note ON (0x90): C4 (0x3c), velocity 76 (0x4c) 8F 00 **80 3c 00** ----- Note OFF: C4, velocity 0
End of Track	ff 2f 00

Figure 1: The music notation and the MIDI hexadecimal code of the melody “twinkle twinkle little star”.

The BPMN model of Figure 2 describes the MIDI parsing system developed to extract the key information required to build the automated music arranger framework. This system is composed of a MIDI parser and a header parser (each of them having a different role), in addition to Papyrus system that orchestrates the components. The parsing system loads the MIDI file from the Papyrus engine (F2. A), then reads the first four bytes to verify that the file loaded is a MIDI file. A MIDI file should start with the following sequence of hexadecimal bytes: 0x4d 0x54 0x68 0x64. If the file loaded is a MIDI file, the parsing process starts. Otherwise, a message is sent to the Papyrus engine to terminate the parsing process (F2. B).

The MIDI parser begins by loading the chunk header hexadecimal bytes and calling the function that gets the MIDI type, the number of tracks, and the PPQN of the loaded file (F2. C). All those information are sent to the Papyrus modeler, which in turn saves them to an external database. If the number of tracks is higher than one, a message is sent to Papyrus to terminate the process, since the developed system is expecting a one-track melody. After receiving the chunk header data, the system begins reading the next bytes. If the next byte is equal to 0xff, the system recognizes that the following sequence of bytes might be the information associated with the tempo, the time signature, or the key signature. If the next byte is equal to 0x51, the system loads the next four bytes, calculates the tempo, then sends the tempo of the song to

Papyrus. If the next byte is 0x58, the system loads the next five bytes, analyzes them, then sends the time signature to Papyrus (F2. E).

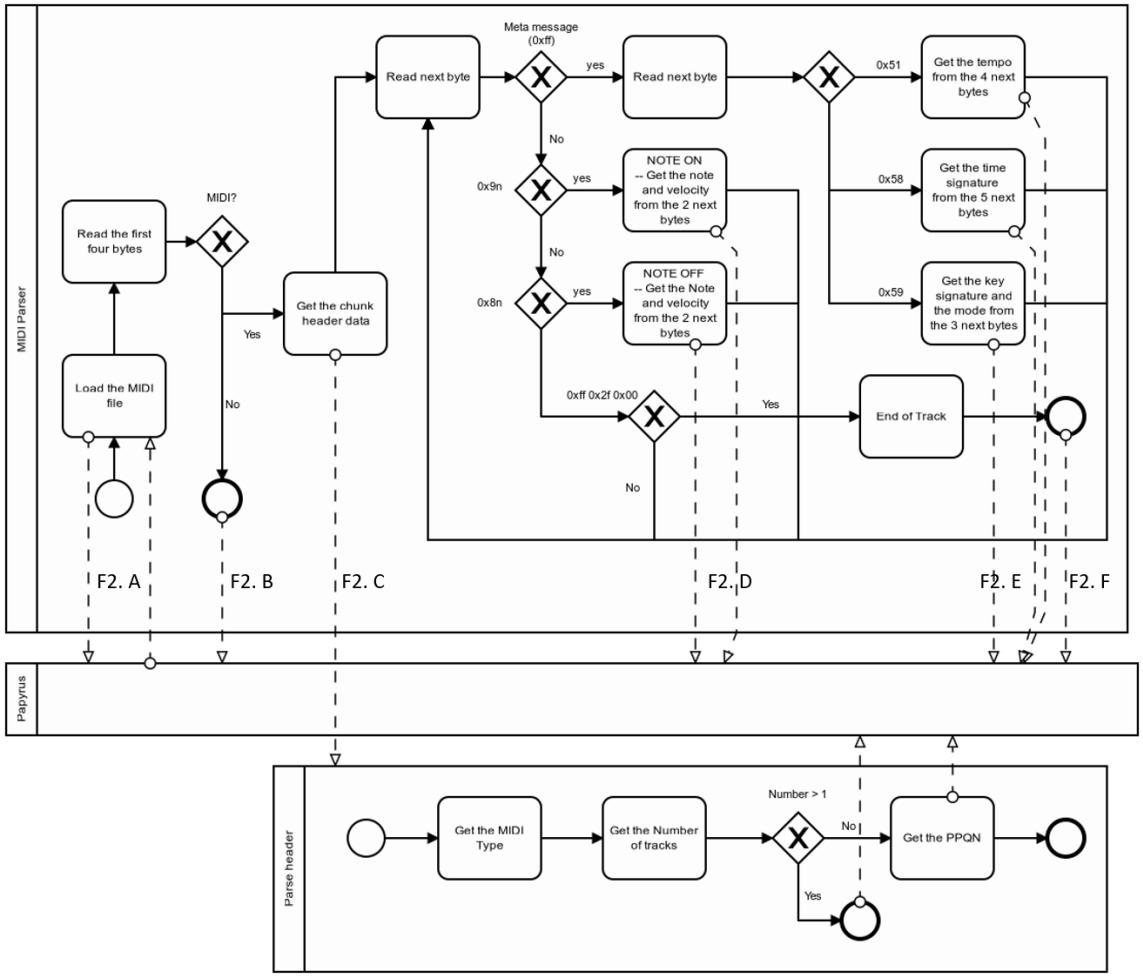


Figure 2: BPMN model of the MIDI parser.

If the next byte is 0x59, the system gets the key signature from the next three bytes, then sends the key signature used in the MIDI file. After retrieving the essential information, the system starts reading the notes and sending them sequentially to Papyrus to be saved (F2. D). Each note has a timestamp called delta time that changes according to the assigned tempo. A turned-on note (0x90) keeps sounding until receiving a Note OFF message (0x80). Both bytes (0x90 and 0x80) are followed by two bytes that indicate the note and its velocity. At the end of the track (0xff 0x2f 0x00), the parsing system sends a message to the Papyrus engine indicating that the parsing process has been successfully completed (F2. F).

3.2 Bars (or Measures) Decomposition Process

In order to split the music into equal measures, from rhythmic perspectives, one should know the numerator and denominator of the time signature for the sequence of consecutive measures. As shown in Figure 4, the duration of a whole note is equivalent to the duration of two half notes, the duration of two half notes is equivalent to the duration of four quarter notes, etc. The time signature is a fraction representing the quantity and the value of the beat. For example, a $\frac{4}{4}$ (or $4 \times \frac{1}{4}$) time signature (Figure 3) indicates that we have 4 beats per bar, each one of them having the value of one quarter note (1/4 of a whole note). A $\frac{2}{2}$ (or $2 \times \frac{1}{2}$) time signature indicates that we have 2 beats per bar, each one of them having the value of one-half note (1/2 of a whole note).

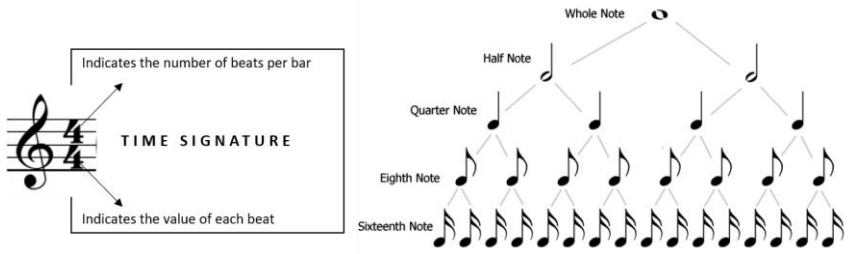


Figure 3: Time signature, note duration, and divisions.

To facilitate the development of the bars decomposition system, we have normalized the calculation so that the number of quarter note N per bar is obtained, whatever the time signature is. The decomposition process places the notes into numbered bars. It will not affect the musical composition, the notes, and the arrangement process. It only assembles the notes according to their values and the time signature. Considering that n is the numerator and d is the denominator of the time signature, the number of quarter notes per measure is calculated as follows: $N = 4 \times \frac{n}{d}$

For example, in a $\frac{6}{8}$ time signature, there are: $4 \times \frac{6}{8} = 3\ quarter\ notes\ per\ bar$

The total number of bars is equal to the total number of quarter notes divided by the number of quarter notes per bar.

The BPMN of Figure 4 describes the bar decomposition process. This process is launched by the model orchestrator Papyrus.

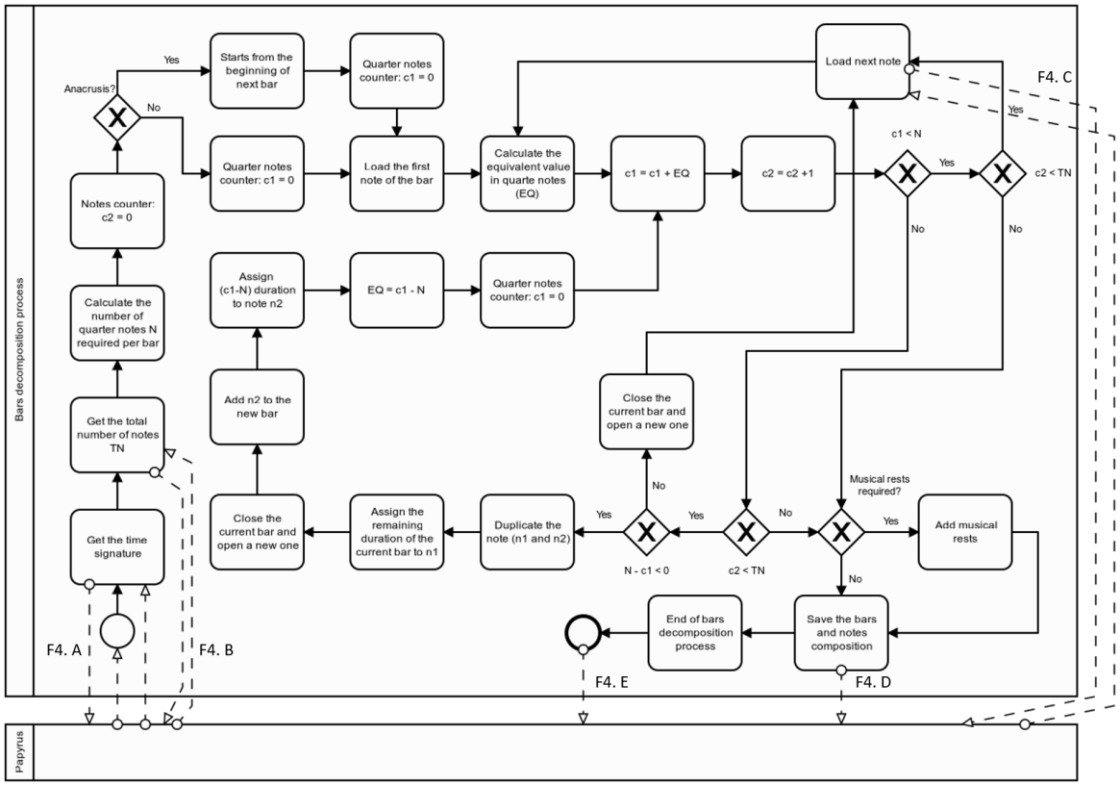


Figure 4: BPMN model of the bars decomposition process.

The main role of this component is to load the notes, analyze them, and place them into the music bars based on the provided time signature. This component loads the time signature from Papyrus (F4. A) and

the total number of existing notes (TN) (F4. B), then calculates the number of quarter notes (N) required for each bar. Afterwards, the system initiates two variables c1 and c2. While scanning the notes of the melody sequentially and based on the time signature, the variable c1 counts the number of equivalent quarter notes required to fill the bars. The counter c1 is reset to 0 each time it reaches the value N. The variable c2 is in charge of counting the total number of scanned notes. During the scanning process, if the counter c2 is lower than TN and the counter c1 reaches N, a new bar is generated, and then filled in again. This process is repeated until the end of the track (F4. C, F4. D, F4. E).

Three main challenges that should be taken into consideration are the following ones: (1) the first bar of a melody starts in anacrusis (known as "levare" in Italian): in this case, the first bar in anacrusis must be eliminated and the arrangement process starts from the beginning of the second bar. The anacrusis simply entails of the unaccented note/notes preceding the first accent of the rhythmic division in a music composition; (2) the duration of the last bar notes is less than the expected duration: in this case, musical rests must be added according to the remaining unfilled duration; (3) pair tied notes exceeding the remaining rhythmic value available in the bar: in this case, the note is duplicated, then assigned the corresponding duration in the current bar and the next new established bar.

3.3 Scale Detection

The scale detection stage is considered as the most delicate phase in the automated music arranger system, since incorrect scale detection leads to incorrect chord generation, and subsequently to a bad arrangement result. A scale detection component is developed to detect the six scale possibilities based on the last played notes in the MIDI file. Knowing the last note (n) of the melody, this component gets the last second, third, fourth, fifth, sixth, and seventh intervals in relation to the note (n), checks if the fourth and fifth are perfect intervals or augmented, then checks the other intervals if they are minor or major. Based on the results of the aforementioned intervals, the system assigns the right scale to the uploaded melody (see Figure 5, that describes the BPMN model of the scale detection component).

Based on the last note of the melody and the scanned intervals, six possibilities exist; i.e., if the last note of the melody is C, the melody can be on: (1) C major if the melody ends with the tonic; (2) C minor if the melody ends with the tonic; (3) A minor if the melody ends with the minor third of the tonic; (4) F major if the melody ends with the right fifth of the tonic; (5) F minor if the melody ends with the right fifth of the tonic; or (6) Ab major: if the melody ends with the major third of the tonic.

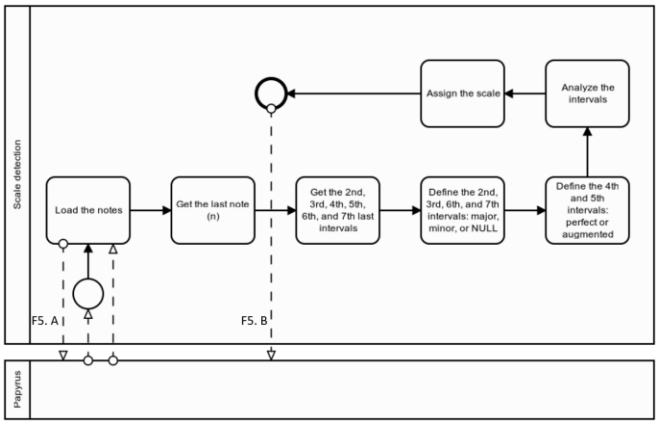


Figure 5: BPMN model of the scale detection component.

3.4 The Simulation Process for the Chord Progression Selection

After choosing the scale of the melody, different chord progressions are simulated, analyzed, and then selected. In this section, we define the system component in charge of simulating all possible chord progressions to finally select the best chord progressions that accompany the melody. Some chord

progressions are built for major melodies and others for minor melodies. The main advantage of an arrangement based on well-defined chord progressions is that it does not produce a list of successive chords that leads to poor music arrangement results. As the number of chord progressions increases, the repetition of chords decreases. This component assigns to each bar one or more chords. The melody usually starts with an accompaniment of I_Maj or I_min chord, depending on the scale. “I” means the tonic note, “Maj” for major chords, and “min” for minor chords. The system then proceeds to the simulation and comparison of major chord progressions for major scales, and minor progressions for minor scales. Chord progressions are of different sizes. They can be composed of 2 chords up to 64 consecutive chords. Hundreds of possible progressions for minor and major melodies exists. The chord progressions are saved in an external database linked to Papyrus. All progressions are chosen based on thorough theoretical and harmonic studies. The system scans the beats/bars of the melody successively and assigns one or multiple chords to each beat/bar. The simulated first beat/bar of the melody is compared with the first chord of each progression. Next beat/bar is compared with the second chord of each progression, and so on. For each simulated chord, a level of reliability (LRC) is assigned to it. The chord that has a LRC lower than the required level is rejected with the progression that contains it. A chord progression is considered acceptable when all its chords have an LRC higher than the required level. All selected progressions (not previously rejected) are considered eligible to accompany the melody. The same set of bars can have more than one accepted progression. Therefore, another analysis takes place to choose the best chord progression between the selected ones. By calculating the mean value of LRCs, one can have the level of acceptance for each chord progression (LAS). Based on the LAS, the system chooses the most appropriate chord progression that accompanies a part of the melody. When a progression is chosen, the whole process starts again until the end of the track is reached (see BPMN model in Figure 6). Furthermore, different factors and parameters affect the selection of chords/progressions: (1) the length of chord progressions: longer chord progressions (having a greater number of chords) are preferred, i.e., a 6-chord progression having the same LAS as a 2-chord progression will be selected; (2) when common notes are found between the notes of the chord and those of the corresponding beat/bar, the LRC is increased; (3) the LRC is decreased when there are notes in the chord dissonant with the notes of the beat/bar. For example, minor, and sometimes major, second notes; and (4) a chord repetition can occur between consecutive beats. However, the repetition should be avoided between two consecutive bars. Once the chord progressions are chosen successively, and the style of music is determined on Papyrus, the system starts calling the respective MIDI arrangements; e.g., if the user chooses the "Samba" style, the system retrieves all required Samba chord-based arrangements saved in advance, offline. The arranged chords are loaded and concatenated in a sequential order, based on the chord progressions simulation result.

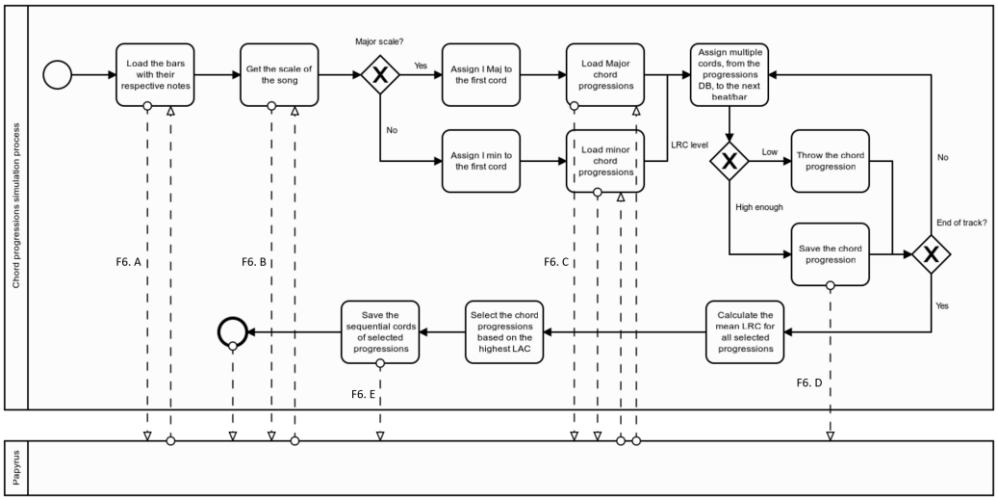


Figure 6: Chord progressions simulation, analysis, and selection process.

3.5 Papyrus Modeler Orchestration Process

The BPMN modeling language has been used to build the different components that compose the complex automated music arrangement system. However, the implementation of this system needs an interconnection interface to ensure the communication between several types of processes and technologies presented in the aforementioned sections. For this purpose, we use Papyrus (Guermazi et al. 2015) , an open-source project, providing an integrated environment for editing and executing processes through Unified Modeling Language (UML) and Foundational Subset for Executable UML Models (fUML). The tool offers many interconnection capabilities with complex and heterogeneous systems (Gorecki et al. 2020; 2021). Papyrus supports UML profiling mechanisms allowing one to extend metaclasses of existing UML metamodels by using stereotypes, so to create a so-called UML profile, and to define a custom behavior at the simulation time using the fUML execution engine (Moka). This feature allows users to define a context-dedicated modeling language, and an adapted execution process. For the orchestration process of the music system components, UML profiles have been defined to store the exchanged data between the Papyrus model, considered as the master component, and each of the developed components responsible for the arrangement process. Papyrus adds new parameters to the executed tasks. Several stereotypes are applied to the UML actions of the main UML metamodel. Thus, the created profiles can be applied on any task to define a communication interface with the components. Once a task is identified, the exchanged data are stored, as parameters, inside the task itself. The execution engine receives those parameters and sends them back to another process. Once UML profiles are defined, it becomes necessary to develop new extensions for Moka engine to achieve the required interaction with the parameters added through the UML profiles. These extensions are simulated each time an extended task is launched to interconnect with the music developed components.

4 RESULTS AND DISCUSSION

In this project, we took into consideration the six scale possibilities that can be obtained from the last played note in the melody, unlike the existing automated arrangers that consider only two possibilities from the last played note. The existing applications usually check the last note (n) of the melody, consider it as the tonic note of the melody, then scan the third interval. If it is a minor interval, the automated arranger executes the arrangement as if the melody is on (n) minor. If it is a major interval, the arrangement is executed on (n) major. In the scale detection system component, we consider the six scale possibilities from the last scanned note of the melody, i.e., a melody ending with a note C can be on C major, F major, Ab major, C minor, A minor, and F minor scales even though the highest possibilities are the two scales C major and C minor. After the scale detection process, the system chooses the best chord progressions to use, based on a grading system that gets the highest LRC from the chords, then calculates the best sequences having the highest LAS. Each chord progression has a sequential set of chords used to call the respective MIDI arrangements, which are loaded based on the musical genre selected by the user. After a comparison with existing automated music arrangement tools, the developed system shows multiple improvements starting from the scale detection to the arrangement result. Even with a simple melody, the existing tools, most of the time, showed a misdetection of the scale leading to a dissonant arrangement with the supplied melody, chords repetition that may be possibly repeated up to four successive bars, dissonant chords generation, and bad chord progressions. A detailed comparison between the developed system and the existing tools can be found in Possik (2010).

5 CONCLUSION AND PERSPECTIVES

The proposed automated arranger has been developed on a model-based collaborative platform integrating UML modeling and execution. The definition of Moka extensions and UML profiles using the Papyrus environment allow the interconnection and orchestration of complex processes. The developed automated music arrangement framework shows various improvements compared to some existing automated music

arrangement tools in the market. This is an ongoing project, work is in progress to introduce new approaches and features, such as: (1) adding a reinforcement learning feature for a better configuration of chords and chord progressions; (2) adding a new component that converts the audio format to MIDI, which allows the user to play the melody on an instrument, or sing it on a microphone, instead of providing it as a MIDI file; (3) adding a new component that converts the resulting arrangement from MIDI to audio format, while allowing the user to choose between local synthesizers, external synthesizers, or virtual studio technology instruments known as VSTi; and (4) allowing the user to modify the simulated chords and progressions through a graphical interface.

REFERENCES

Amazon. 2020. *AWS DeepComposer*. https://aws.amazon.com/deepcomposer/. Accessed Feb. 08, 2021.

Biglobe. 2017. *Easy Music Composer* (version 9.97). http://www5f.biglobe.ne.jp/~mcs/emc.html. Accessed Jan. 15, 2021.

Bonnin, G., and D. Jannach. 2014. "Automated Generation of Music Playlists: Survey and Experiments." *ACM Comput. Surv.* vol. 47 no. 2. https://doi.org/10.1145/2652481.

D'Ambrogio, A., and P. Bocciarelli. 2007. "A Model-Driven Approach to Describe and Predict the Performance of Composite Services." In *Proceedings of the 6th International Workshop On Software and Performance (WOSP'07)*, pp. 78–89. Buenos Aires, Argentina.

Gianni, D., A. D'Ambrogio, and A. Tolk. 2014. *Modeling and Simulation-Based Systems Engineering Handbook*. CRC Press.

Gorecki, S., J. Possik, G. Zacharewicz, Y. Ducq, and N. Perry. 2020. "A Multicomponent Distributed Framework for Smart Production System Modeling and Simulation." *Sustainability* vol. 12 no. 17. https://doi.org/10.3390/su12176969.

Gorecki, S., J. Possik, G. Zacharewicz, Y. Ducq, and N. Perry. 2021. "Business Models for Distributed-Simulation Orchestration and Risk Management." *Information* vol. 12 no. 2.

Guermazi, S., J. Tatibouet, A. Cuccuru, S. Dhouib, S. Gérard, and E. Seidewitz. 2015. "Executable Modeling with FUML and Alf in Papyrus: Tooling and Experiments." In *1st International Workshop on Executable Modeling, EXE 2015*, edited by Gray J, Langer P, Seidewitz E, and Mayerhofer T, 1560:3–8. Ottawa, Canada: CEUR-WS. https://hal-cea.archives-ouvertes.fr/cea-01844057.

Jasmine Music Technology. 2004. *Onyx Arranger* (version 2.1). http://www.jasminemusic.com/onyx.htm. Accessed Jan. 21, 2021.

kvraudio. 2021. *Chord Composer* (version 1.5). https://www.kvraudio.com/product/chord-composer-by-intuitive-audio. Accessed Jan. 10, 2021.

Maruta, S. 2005. *E-Xpressor* (version 1.31). http://www.allworldsoft.com/software/0-403-e-xpressor-real-midi-accompaniment-maker.htm. Accessed Jan. 10, 2021.

Morris, H., and G. A. Wainer. 2012. "Music Generation Using Cellular Models." In *Proceedings of the 2012 Symposium on Theory of Modeling and Simulation - DEVS Integrative M&S Symposium*. TMS/DEVS '12. San Diego, CA, USA: Society for Computer Simulation International.

OMG. 2013. "Architecture-Driven Modernization Task Force." http://adm.omg.org/. Accessed Feb. 01, 2021.

Possik, J. 2010. "Arrangement Musical Automatique." http://dx.doi.org/10.13140/RG.2.2.24777.67682.

Possik, J., A. D'Ambrogio, G. Zacharewicz, A. Amrani, and B. Vallespir. 2019. "A BPMN/HLA-Based Methodology for Collaborative Distributed DES." In *IEEE 28th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pp. 118–23. Capri, Italy. https://doi.org/10.1109/WETICE.2019.00033.

Whorley, R. P., and D. Conklin. 2016. "Music Generation from Statistical Models of Harmony." *Journal of New Music Research* vol. 45 no. 2. https://doi.org/10.1080/09298215.2016.1173708.

Zacharewicz, G., S. Diallo, Y. Ducq, C. Agostinho, R. Jardim-Goncalves, H. Bazoun, Z. Wang, and G. Doumeingts. 2017. "Model-Based Approaches for Interoperability of next Generation Enterprise

Information Systems: State of the Art and Future Challenges." *Information Systems and E-Business Management* vol. 15 no. 2. https://doi.org/10.1007/s10257-016-0317-8.