UNIVERSITY OF CAPE TOWN

DEPARTMENT OF COMPUTER SCIENCE

CS/IT Honours Final Paper 2019

Title: Salsational: Implementation of an API for the Computerisation of Dance Choreography

Author: Micara Shashi Marajh

Project Abbreviation:

DeDance

Supervisor(s):

Professor Maria Keet

Category	*Min*	*Max*	Chosen
Requirement Analysis and Design	*0*	*20*	20
Theoretical Analysis	*0*	*25*	0
Experiment Design and Execution	*0*	*20*	0
System Development and Implementation	*0*	*20*	15
Results, Findings and Conclusion	*10*	*20*	15
Aim Formulation and Background Work	*10*	*15*	10
Quality of Paper Writing and Presentation	*10*		10
Quality of Deliverables	*10*		10
Overall General Project Evaluation (*this section allowed only with motivation letter from supervisor*)	*0*	*10*	
Total marks		**80**	80

Salsational: Implementation of an API for the Computerization of Dance Choreography

Micara Shashi Marajh
mrjmic001@myuct.ac.za
University of Cape Town
Cape Town, Western Cape

ABSTRACT

Dance is considered to be an intangible cultural heritage, therefore development of tools to aid instructors in teaching choreography is a necessary and important research topic. Dance notations exist for the purpose of recording dance movement through symbols. However, these notations have been described as too complicated for the novice dancer to understand and model. Through ongoing research, it was noted that little work has been done to transform paper-based dance notation into software. Thus, we have recognised that there is significant demand for the creation of a software tool to computerise dance movement. In particular, no software tool has been developed for the purpose computerizing Salsa dance.

In this paper, we address these shortfalls by developing a tool to computerise Salsa dance notation in the form of an API. The API provides developers with a platform to describe dance moves and create a dance sequence based on these moves. The API design is abstract and extensible as it allows implementation of various dance styles catering to the user's specific needs. The main goal is to aid developers in computerising various dance styles in attempt to revolutionise the way dance is taught. Focus was based on assessing the extensiveness, abstraction, learnability and re-usability of the software tool in order to be implemented by other developers. After conducting software tests, we concluded that our API successfully computerized a Salsa dance notation. Various dance styles were also tested with the system and found that our system can computerize other dance styles provided they are performed on a multiple of 4-beat counts. Software quality testing with developers proved our system to be understandable, learnable and re-usable.

KEYWORDS

Application Programming Interface, Notation, Extensibility, Usability, Salsa;

1. INTRODUCTION AND BACKGROUND

Dance is the most elementary of the art forms [7], that involves expressions through bodily actions. Alike, it is a great therapeutic activity and evolved as a form of medicine for psychological and physical illnesses [1]. Consequently, the preservation of dance is of the utmost importance. Salsa is believed to be one of the most practiced and most popular dances of the Latin dance styles in the world today. Since the early 1990's, Salsa has experienced added attention from American audiences who have put a large amount of money into learning Salsa dance [4].

The Evolution Dance Company (EDC) is an established Salsa Dance company in Cape Town which strives to unite and empower diversity through dance. Our client, Angus Prince, aims to provide a sociable and professional environment for students to captivate themselves in the art of dance. Their primary focus is on the development of dance, with the vision of making Salsa accessible to all the people of Cape Town. Such an establishment gives new impulsion to dance. In the interest of fulfilling this vision, the current methods of teaching dance need to be revised.

Dance now integrates technological aspects in teaching, performance and choreography. Due to rapid technological advancements, it is increasingly important for novice dancers to keep up their technological expertise and advances used for creating, producing and documenting their dance attempts. Remote education has also become an increasingly popular method of teaching dance [18]. These technological advancements will give students the tools for developing their own environment of exploring movements and reflecting upon them. By bringing in creative practical tools, it allows students and dance teachers to enrich their experiences using a technological platform for dance. From our external client's knowledge, the dance community relies heavily on videos to record dance moves which is problematic because videos lack formal clarification of moves into clear steps and positions which proves this method to be inefficient.

Our project goal is to develop a user-friendly, software tool, in the form of a desktop application, for teachers to plan lessons that will provide a revolutionary approach to the way Salsa dance is taught. This tool will allow novice dancers to develop their skills by enabling them to analyse specific dance moves and create dance sequences. Developing a tool to formalise dance steps and animate a Salsa dance form, proves challenging as there are currently limited resources available [26]. The existing paper-based notations have proven to be too complex for students to record and model dance choreography [8]. As these are paper-based systems, there has not been much uptake on computerising these notations.

We intend on leveraging an existing notation system to generate a more representative notation for dance. Our solution to the aforementioned problems is to create an Application Programming Interface (API) to computerise dance notation. Our API will provide a way to support several dance styles in a well-defined manner. This approach allows for re-usability as developers can use the underlying code structure created and modify it according to their needs. Developers will interface with our system through plugins, making it extensible in it is design.

Through extensive research, we have structured our study around a few main research questions. These are crucial to examine the extensibility, usability, level of abstraction and learnability of the API. These research questions are as follows:

1. Does the API allow for extensibility with other dance styles relating to Salsa?
2. Is the API designed well enough to allow for easy learning and understanding?
3. Does the API's abstraction level serve the purpose of re-usability?
4. What is the effort needed to interpret the API based on the class names and attributes?

The development of APIs is becoming a larger part of programming [22]. The use of APIs is advantageous as it improves programmers' productivity by enabling the reuse of more code instead of writing it from scratch. Another motivation for our decision is that an API provides a larger scope. With an API, an application can distribute information and services to other developers which can create custom user experiences. We hope we will assist dance students in enhancing their learning experience through the use of our tool. The remainder of this paper is structured as follows: related work pertaining to the project, the Object- Orientated Design and Implementation of the API and finally, our Results and Conclusions after system and software quality testing.

2. RELATED WORK

2.1 Dance Notation Systems

Dance notations are used to record dance choreography and enable them to be reproduced by dancers and choreographers. Dance notation has a symbolic representation that is similar to music. Various notation systems have been attempted for analysing human movement. However, human movement is complex to describe, therefore movement notation systems are inherently complicated and difficult to master. Benesh Movement Notation (BMN) is one of the oldest notation systems for analysing and documenting human movement by using symbols [5]. BMN is a 2D notation that records human movement in 3D of space which has been successfully implemented in the production of scores for a wide range of dances, excluding Salsa [19]. However, it lacked possibilities of complex movements inferences for dance such as turns and foot movement. Labanotation is a general body motion notation that is commonly used as it does not depend on any specific dance [5]. It has been described as complicated and only easily understood by those who study it [2]. It was also proven difficult for dance novices to understand since instruction manuals regarding it is encoding patterns are a countless number of pages in length [2]. In spite of this, it is still the most popular dance notation used today [1]. This dance notation is used for human movement but has not been optimised for a partnered dance. These notations focus on the more classical dance styles and does not allows users to experiment with different sequences and clearly see the movement of different parts of the body.

Renesse and Ecke [6] created a "Space of Salsa Dance" notation using mathematical equations in the form of a text-based diagram. This method is limited to arm movement and lacks notation for feet movement. In 2002, the "Salsa Dictionary" was created [23] as a way to learn Salsa dance patterns displayed in a table-based notation. To our knowledge, this method is not defined in any structured serialisation. A move is defined for a pair: the leader and follower using symbols to describe each move. There is ambiguity in the representation of the symbols used to describe the actions in the Salsa Dictionary. However, in the past 17 years, there has been no uptake to the use of this notation. The aforementioned notations are paper-based notations, which can be easily misinterpreted [8] and difficult to conceptualise [8].

Hand hold	* N *	* N *
Direction	M → ← L	L → ← M
Man		
Common action	XBL	
Lady		

Figure 1: Salsa Dictionary notation consisting of elements and symbols [18].

2.2 Computerising Notation

While BMN provides a theoretical approach for modelling dance, it is not part of regular language of all dance instructors. Web-based Movement Library (WML) was proposed to provide an interface for searching recordings and collect data on how the dance experts characterize various parts of the recordings pertaining to their movement positions [15]. Specifically, users can search the recordings by dance genre, and search by using keywords that are included as data or annotations of the recordings. The keywords refer to various areas of movement. XML (eXtensible Markup Language) has been designed to describe data. Files will be searchable, software independent and be able to share and interchange data with several systems. HumanML [12], an XML specification, attempts to codify human thought, emotion, gestures, attitude, intent and many other characteristics of the human condition. Nevertheless, it is complex for quick notation and unsuitable in terms of limited gesture description for the purposes of dance notation.

In 2006, Nakamura and Hachimura discussed an XML representation of Labanotation, called LabanXML [19]. This language allows a user to input and edit movement of dance in the human body as well as exhibit animation of a human body model in 3D graphics. This research was improved upon by Hatol et al [12] using MovementXML. Its purpose is to represent the semantics of Labanotation in XML. MovementXML is a useful XML preliminary plan to encode dance movement. MovementXML allows smaller movements to be joined together to form a higher- level movement. However, these have not been widely adopted to describe gesture for the Salsa dance style.

eXtensible Dance Scripting Notation (XDSN) is a type of dance notation which has a handwritten and machine-readable format [9]. It is structured in a way to encourage various approaches on how the notation should written and implemented. The XDSN system expands on existing notation systems [9] using word, abstract symbol and notation markings in numbers. To modify the handwritten notation to the machine-readable format, it has to be grouped into defined elements and values. Only the values are used in the handwritten score, so the elements can be derived and appended during conversion into the machine-readable format [9]. However, there has been no uptake in attempt to encode this notation. Even though there exist several software notations, attempts to convert notation into machine readable format proved complicated. Problems included abstract symbol transformation

[17], cross platform operability [18] and complexity of software development [2].

3. DESIGN

Our system was developed using the Iterative Waterfall Model approach [16]. The step-by-step nature of this approach allows the project to contain a more detailed, robust scope and design structure to allow for more extensive planning and documentation. This model separates the development process into phases which can be repeated several times [16]. A detailed description of the development processes carried out during this project is presented in Table 1. Focus was spent on the Implementation and evaluation phases as changes were made after testing the system.

Table 1: Iterative Waterfall Development Phases [16]

Development Phase	Description
Planning and Requirements	Project planning specification and software requirements established.
Design	Technical requirements gathered and programming logic established.
Implementation	Coding process of the project
Testing	Software testing and system quality testing carried out
Evaluation	Examine feedback from testing procedures

3.1 System Architecture

In this section we provide a high-level architecture of the API system with the purpose of visualising the interaction between the components of the system. The project comprises of a notation graphics pipeline component. The notation pipeline defines and analyses the dance notation to formalise it in a way that can be accurately computerised. The notation system will be governed by an underlying ruleset which specifies sequences that can and cannot be performed in succession in the form of a grammar. A parser will be plugged into our system to determine the legality of a dance sequence as seen in Figure 2 below. Once a dance sequence is validated, it is rendered in the graphics pipeline. The system is comprised of abstract classes and interfaces to provide interaction between components. System plug-ins were created to allow developers to extend our application and easily add new features. These components were split to ensure modularity in the design. The API plays a significant role in integrating the notation pipeline and graphics pipeline components in order to create the final product. It serves as a messenger that processes request and ensures seamless functioning of the systems.

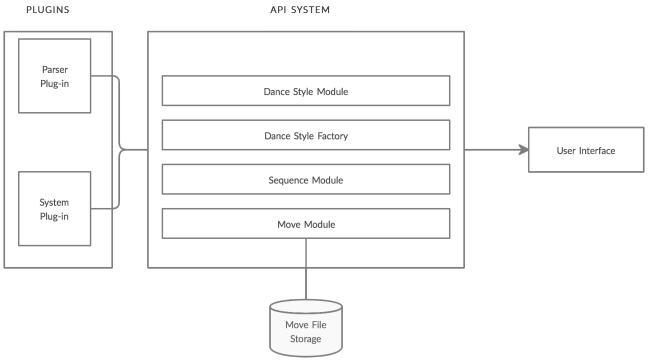

Figure 2: API System Architecture

3.2 Requirements Analysis

To ensure this project meets the requirements of our client, it was necessary to gather information relating to the final product expectations. This assisted us in ensuring that the design of our system correlated with the needs of client during the iterations of the design cycle.

The first step was to meet with our client from Evolution Dance Studio and determine his expected requirements for the system. During the first meeting, he provided us with a Salsa Dance syllabus which consisted of the various basic moves in Salsa. The syllabus covered the most important introduction to Salsa steps as well as the preferred nomenclature for each. This syllabus will be used in the development of the notation definition component, which will play an important role in the design of the API. We had to ensure we treat the notation with meticulous attention to detail as it provided the basis of our system. The second meeting with our client involved acquiring his expert knowledge on our naming conventions of the moves and constraints associated with moves. The third meetings took place to communicate our progress and ensure that the approach taken to computerize the dance made logical sense to our expert. In total we had 4 meetings with the client and outcomes of each meeting was documented to be referenced throughout the project. The knowledge gained from the expert's requirements was used to create the design of our final system as shown in Appendix A, Figure 3.

The requirements analysis focuses on the core tasks that the system should perform in order create the final product for our client. Below is a list of the main components of the system:

1. Define dance elements which are terms used to describe each step in the dance.
2. A dance line should be created consisting of the defined elements. Each line describes the main positions of dancers depending on the dance style.
3. A dance move consisting of all lines and the elements associated with each line.
4. Create a dance sequence comprising of a list of moves
5. System must provide plug-in support for developers
6. The system should be able to support the implementation of various dance styles.

3.2.1 Stakeholders

Considering this is a software engineering project, our next task was to identify and analyse the people involved that are impacted

by the quality of our API design. As an API designer, it is crucial to understand the needs of the stakeholders in order to ensure success of the project. For the scope of this project, we consider the following stakeholders: API users are the programmers who use will utilize our API to write their programs, therefore our API need to be suited to our target audience [25]. End users of the final product are also affected by our API, although they are affected indirectly.

3.2.2 Programming Language Requirements

In this section we will discuss the framework, libraries, development kit and languages that relate to the construction of the API. Our API is structured using the principles of object-oriented design and dynamically linked objects. Dynamically linked objects refer to the system plugins. The core architecture maintains a set of classes that it uses in it is user-interface. We have created a simple design for all classes in order to have a structured core that is easily understood by those that are extending it. This allows for portability and added functionality to be implemented within the plug-ins.

Language requirements play an integral role in the implementation of the project. Java was selected as the language of choice due to its robustness, ease of use and good cross-hardware-platform portability [3]. The ability to run our program on many systems is crucial and Java succeeds in doing so. We adopted the most current software development practices in object-oriented programming. Apache NetBeans 11.0 IDE was the framework of choice as it has dynamic language support and is an extensible platform. JDK 12 was used as our software development platform. Language binding was also used by the API. Language binding allows a library written in one language to be used when developing in another language. Considering that our graphics component was created in OpenGL in C++, we required language binding in order to execute the dance animations. LWJGL is a Java library that enables cross platform access to APIs.

3.3 Software Design Considerations

The API design must be structured in a way that it creates a large impact on at least one of our stakeholders to measure success. Below we identify the potential challenges faced when deciding on the design of our API:

1.Finding descriptive, non-ambiguous names for the API features to ensure that programmers understand what each class does.

2.Discovering relations between API classes require significant effort. A simple design is beneficial when you are unaware of the level of experience of each programmer.

3.Flexibility can have both a favourable and unfavourable impact on the API design: experienced programmers can take advantage of it, but it may confuse the novice programmers.

4. Complexity of the integration of both the notation and graphics pipeline

5. Modularity must be ensured in order to improve maintainability. Components will be tested rigorously before being integrated with the final system.

The two most important qualities of an API are it is usability and it is capabilities [21].

3.4 Algorithms and Data Structures

3.4.1 API Development

The software design is the most crucial consideration in developing the API as we need to ensure that it will support various dance styles as stated in the requirements analysis. Thorough research was performed in Section 2 to identify which paper-based notation to computerize that was most representative of all dance styles. This is to ensure that we eliminate current problems faced by the dance community making this project unique in its execution. The basis of our design follows the structure of the Salsa Dictionary [22]. To compensate for the ambiguous nature of the notation, we will adjust the notation, but we will still preserve it is fundamental principles.

The Salsa Dictionary provides a platform to for beginners to learn Salsa On 1 and Salsa on 2 moves by breaking them down into basic elements and lines that are always used when creating a Salsa dance move. These elements and lines are also present in several other dance styles allowing us to create a generalized dance notation structure. In order to capture the various Salsa steps, the concepts of “Salsa Lines” and “Salsa Elements” are described in the Dictionary for a pair of dancers. The five Salsa Lines describes the orientation of the main components pertaining to a specific dance style. For Salsa, these are Handholds, Direction, Leader, Follower and Common Action as seen in Figure 1. The four Salsa elements describe the step performed by the dancer unique to each line. Elements are made up of Handhold, Direction, Position and Actions pertaining to the pair of dancers.

The Handhold line indicates the specific handhold position for the pair of dancers during each bar. A bar consists of 4 beats. The Direction line shows where the dancers are facing in relation to the line of dance. The Leader line indicates the position and actions performed solely by the man, independent of the common actions performed by the pair. Common Action line indicates actions performed by both dancers. The Follower line is used to indicate actions performed by a lady following the man’s lead. The system is designed in a way that allows developers to define their own symbolic representation of an element since this differs for each dance as shown in the Element class in Figure 4. The element symbol is of type string which allows developers to define a symbol in any way they please as all dance styles will require different symbol representations. Therefore the system can support the addition of an arbitrary dance element names without forcing the client to adhere to the element requirements of Salsa. These elements are denoted as symbols that describe the movement of each line as shown in Figure 1. The complete matrix represents a Salsa dance move.

The Element class stores a unique name and symbol for a step. The Line class defines each row in the Salsa matrix in order to maintain complete control over a dance step. The Line class creates an array populated with Element objects associated with that particular Line illustrated in the class diagram in Figure 4. A move comprises of all elements stored in the Line class. An Array list data structure was chosen to store a move as it holds a dynamically sized collection of elements since there is no fixed size of elements shown in Figure 4. The API is being created with other dance styles in mind, so it is imperative

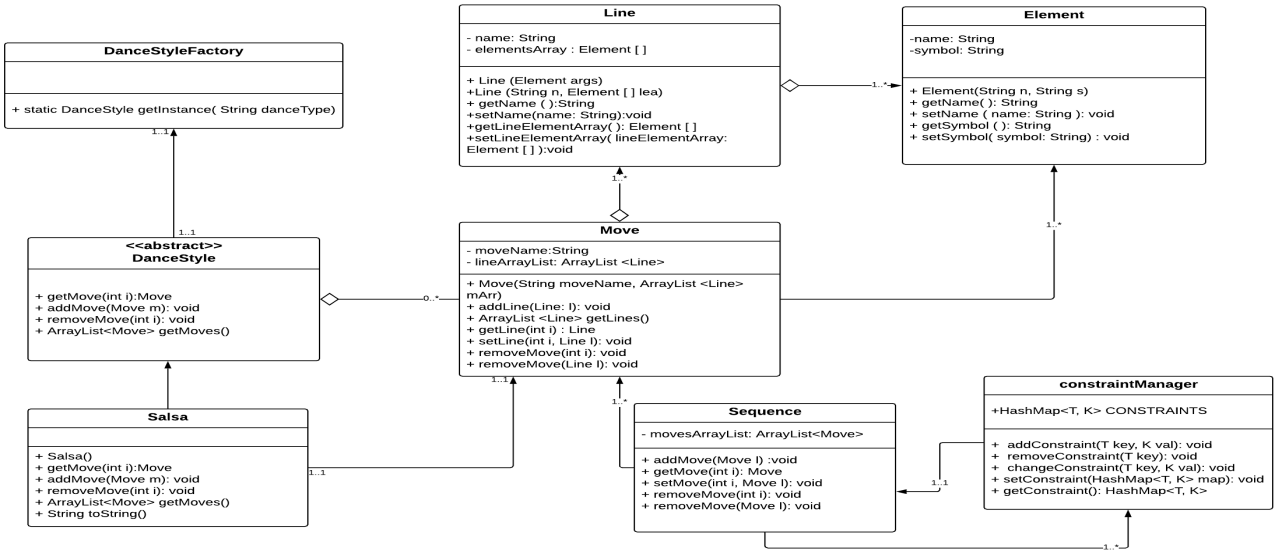

Figure 4: API Class Diagram

that the data structure chosen can support a large number of elements.

An important concept in dance is the time progression for each step. Every dance style is defined by the number of beats. Salsa is comprised of two bars, where each bar is made up of 4 beats. In order to factor in the time progression for dance, the first symbol entered as an element will indicate a step for the first 4 beats and the second for the latter 4 beats etc, separated by a vertical line as shown in Figure 1. This allows for extensibility as several dance styles relating to Salsa. Bachata, Kizomba and Cha-Cha-Cha are related dances performed on a 4-beat count or multiples of 4 beats. There are a number of constraints that restrict the order in which certain moves are performed. Based on the constraints defined by our expert, the parser component, developed by Alka Baijnath, was responsible for verifying whether moves can be performed in the order it is inputted to ensure accuracy in the system functions.

3.4.2 Plugin Development

In order for third party developers to extend our application, a plugin software component was created in order to enable customisation suited to the developer's needs. When a plugin is added to the system, it looks through the classes and interfaces to determine what functions are provided and required by the component, and how it can be connected to the system. Our API provides services for the plugin to use in order to exchange data. The plugin depends solely on our host application and cannot work on it is own. A plugin interface was created in order for the host application to load the plugin.

The host application calls a reference to it itself to allow the plugin to record call-backs which then extends the operations of the host application. The host application depends only on the plugin interface. The host application will use the Java class loader that loaded the application, to load the plugin class. In order for this to occur, the plugin must be provided in the Java class path when initiating the host application. The Class Diagram in Figure 5 demonstrates an example of Plugin System operations for the Line class. The plugin implementation for the rest of the system is not shown.

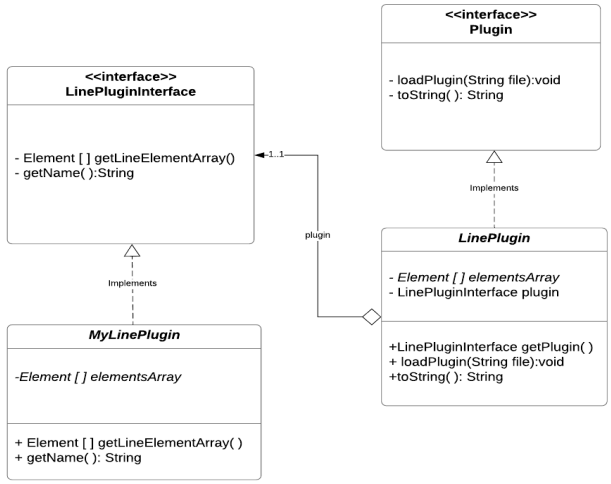

Figure 5: Plugin System overview for Line Plugin

3.5 Design Patterns

Considering that the principle goal of our API is extensibility to other dances, design patterns play an important role in ensuring that our software design is problem-free. Design patterns are a common approach in API design, making it popular among developers [14]. Software developers extending our system can refer the name of the pattern they want to use and immediately understand how to implement it. Creational patterns were used as they provide object creation mechanisms that increase flexibility and reuse of existing code. Structural patterns explain how to assemble objects and classes into larger structures, while keeping the structures flexible and efficient.

3.5.1 Factory Design Pattern

The Factory Method separates product construction code from the code that actually uses the product. Therefore, it is easier to extend

the product construction code independently from the rest of the code. This method provides low coupling and high cohesion of elements which improves maintainability of the system. Factory pattern provides abstraction between implementation and client classes through inheritance. A Dance Style factory method was created to reduce the code needed to construct components across the framework into a single factory method. This allows methods to be overridden in addition to extending the component itself. The Dance Style class declares the factory method and returns new product objects. The factory method is abstract to ensure all subclasses implement their own version of the method. In this case the Dance Style class allows users to create dance moves and implement them according to the dance style chosen. In our case, the Salsa class is a concrete class that overrides the factory method to return a different type of dance. The DanceStyleFactory class gets an instance of the dance style the user wants to implement and returns the selected dance. The Dance Style and DanceStyleFactory class can be seen in Figure 4 above.

3.5.2 Façade Design Pattern

Facade is a structural design pattern that provides a simplified interface to a library, a framework, or any other complex set of classes. Facade defines a simplified interface to a subsystem of objects, but it doesn't introduce any new functionality. The subsystem itself is unaware of the facade. Objects within the subsystem can communicate directly.

The constraints manager class is a facade class as it provides functionality that is hidden from the rest of the code. It was used as a way to isolate the code from the system in order to reduce the complexity and improve readability of the subsystems by masking interaction with more complex components. This class is responsible for validating a sequence of moves against a set of constraints to ensure that the particular sequence can be performed in the chosen order. Once this operation is performed, the facade class will direct the client code to the associated subsystem object that will require this functionality. In this case, once a sequence is validated, it will link back to the Sequence class subsystem to perform the remaining operations. This process is illustrated in Figure 4.

4. SYSTEMS DEVELOPMENT AND IMPLEMENTATION

The implementation of the system follows the system architecture structure created in the Design section in Figure 2. We ensured that it adhered to the requirements analysis and stakeholder requirements developed in the earlier stages of our life cycle.

4.1 Methodology

Following the design, the structure of the API was evaluated in order to measure optimal usability with a variety of user-centred methods. Our API design structure followed heuristic evaluation guidelines [13] to ensure good design principles were met. The names of variables, methods and classes were made as general as possible to ensure easy interpretation by users. Additionally, we ensured that all parts of the design were consistent throughout the development of the API. An aesthetic and minimal design was adopted to ensure successful usability and learnability. To ensure that the API answers all research questions thoroughly and meets all requirements, we divided the evaluation into two parts. First, we evaluated the extensibility of the API by testing it with other dance styles. Secondly, we conducted rigorous in-house testing with developers to evaluate the learnability and understandability of the API.

4.2. System Testing Method

To evaluate our tool, we will follow an evaluation criterion to ensure that the system components are efficient and effective. In order to prove that the software built was suitable to release to other developers, it was crucial to ensure that it was tested thoroughly to prove that the required functionality was produced. Several software tests were performed to determine whether our system met the initial requirements and if it is successful in its approach. The evaluation method process is described in detail below.

We first tested whether our API could successfully carry out its core purpose of computerising dance notation for the Salsa On1 and SalsaOn2 dance styles based on the Salsa Dictionary. We tested this using Figure 1 which represents a Cross Body Lead move in the Dictionary. To test the API's functionality, we input the data described in the table into our system and determine whether the output of the API matches the data in the table. The test is discussed in more detail in section 5.

In order to prove extensibility of our API, software testing occurred to determine whether the system can accommodate the plugin of other dance styles. Tests were conducted against two dance styles relating to Salsa in the sense of an 8 beat count and partner work. Bachata is a dance style similar to Salsa, as shown in Table 2, as it consists of a 2 bar 4-beat count. It is also a partnered dance with open, closed and semi-closed positions, mapping directly to the design of our Salsa dictionary notation. Bachata differs from Salsa in terms of hip movement and a slower pace in rhythm which does not have an effect on the computerization of this dance as hip movement has a standard motion. In order to computerize Bachata dance notation, the coder would follow an approach similar to the SalsaOn1 and SalsaOn2 implementation. The number of elements and lines of dance would remain the same as Salsa, with obvious changes in the nomenclature.

The Cha-Cha-Cha is an Afro-Latin dance style performed as a partnered dance with both open and closed embrace similar to Salsa. It is counted with a 4/4 time as well and is characterised by quick spins and strong torso movement. In order to implement this variation of a partnered dance, we need to store the data associated with a particular move and test it in our system. Results of this test is discussed in Section 5.

Table 2: Comparison of attributes in partner-based dances

	Salsa On1	**Bachata**	**Cha-Cha-Cha**
Music	4/4 timing	4/4 timing	4/4 timing
Basic Footwork	1,2,3; 5,6,7;	1,2,3; tap;5,6,7; tap	1,2,3;cha-cha;4,5,6;cha-cha;
Style	Quick, Quick, Slow. Partner dance.	Relaxed and slow partner work.	Three quick steps with two slower beats done on the one beat and the two beats.

4.3 Software Quality Testing

Rigorous implementation procedures testing was conducted in phases in order to evaluate our software usability and learnability. We have opted to conduct usability tests for qualitative research purposes. We consulted with fellow engineers through an iterative process, requiring them to examine, assess and test our code in order to answer the research questions proposed in Section 1.

Phase 1: A Heuristic evaluation will take place as it can possibly reveal problems in the API. We selected 16 heuristics based on API design guidelines identified by Zabran [23] and used them to categorise the problem areas occurring when evaluating our API. Table 3 gives a short explanation of each heuristic based on [23] we will be testing.

Table 3: Explanation of Heuristic Methods based on [23]

NAME	EXPLANATION
Complexity	Abstraction should be used. API must not be too complex.
Naming Conventions	Names should be used in a consistent manner and must be self-explanatory.
Correctness of Concept	Ensure that elements are modelled correctly to assist programmers in using the API correctly.
Factory Pattern	Factory Pattern should only be used when necessary.
Data Types	Use correct data types suited to the use of that data.
Implementation vs Interface Dependency	Interface dependencies are favoured as they are flexible in their design.
Use of Attributes	Ensure clear interaction among attributes in order to achieve specific functionality.
Method Parameters and Return Type	The use of many parameters should be avoided. Return type must return the value of the given method.
Consistency	API design must be consistent and obey conventions.
Readable Code	API should be designed with the programmer in mind to ensure usability. Programmers must understand what the parameters mean in order to use it.

Phase 2: Once the developers had examined and were familiar with the API code structure, we will then request a practical evaluation to be done. The participants will be presented with 5 tasks involving creating dance moves and sequences using the API features. An outline of the tasks are as follows:

Task 1: Input dance elements for the SalsaOn1 dance style (8 beats) and produce a dance line consisting of these elements.

Task 2: Create a dance move through the factory pattern method.

Task 3: Remove a dance move using the factory pattern method.

Task 4: Create a dance sequence based on the moves in the previous task.

Phase 4: Once the tasks are completed, developers will answer a series of questions based on their findings. Following guidelines on measuring API usability, ten questions were asked based on the assessments. The questions are as follows and classified by the research question they target:

Questions regarding Understandability in Research Question 4:

1) Do you feel you had to keep track of information not presented by the API to complete tasks?
2) Do you feel you had to learn several classes and methods in order to complete a task?

Questions regarding Abstraction and Re-usability in Research Question 3:

3) Do you identify the API abstraction level to be suitable to solve the tasks?
4) Did you often need to adapt the API to meet your needs? e.g. inheriting and overriding
5) In order to use the API, did you feel you had to understand the underlying implementation?
6) Was it easy to assess your own progress/results while solving the tasks?
7) Do you feel you had to choose only one way to solve the required task?

Questions regarding Learnability in Research Question 2:

8) Was it easier to perform the remaining tasks once the first two tasks were performed?
9) Does the code match your expectations to solve the required tasks?

Phase 5: Discussion. We hope to gather quality feedback from the evaluation in order to identify usability issues, if any. The outcome of the methodology will potentially provide us with recommendations for improving the API. Feedback is essential to ensure that we answer all the research questions formulated at the interim of the project.

5. RESULTS AND DISCUSSION

In this section, we will present and discuss the results of the in-depth analysis of the extensibility of our API and the implementation procedures testing conducted. Each phase of the testing is presented in subsections corresponding to the research questions in Section 1. For each question, we will critically

examine, summarize and discuss the results of the Systems Testing and Software Quality Testing in this section.

5.1 API Extensibility

Following the implementation from our system testing in the previous section, results from the tests are discussed in this section. In order to computerize the move, we first input the elements to create a line. We specified the name and symbol associated with that element in order to prevent ambiguity in the understanding the symbols. Each line is associated with a name which defines the main gestures in Salsa. Depending on the dance move, a line will not necessarily have an element associated with it. In order to create a Salsa, move, an arraylist of line objects was constructed based on the input. The adjustable size of the arraylist caters for the different dance style elements as shown in the Line class in Figure 4. Once a move is defined, a sequence of moves is then constructed from an array list of move objects. The sequence is first validated in the constraint's manager class and thereafter a sequence is constructed. Once we have input the elements and created lines, we were able to successfully store a Cross Body Lead move as shown in Figure 6 that mapped directly to the table in the Salsa Dictionary. Test cases were used to test that all requirements specified at the start of the project was met. A test was executed with every class in the system to verify compliance for each requirement (Appendix C). All test case resulted in a pass. Hence, we can conclude that we have successfully computerized a paper-based dance notation for the SalsaOn1and SalsaOn2 dance style.

Handhold	N	N
Direction	ML	LM
Leader		
Common Action	XBL	
Follower		

Figure 6: Cross Body Lead Move representation

In order to prove our claim of extensibility, we attempted to computerise an intermediate dance move in Bachata called the Hammerlock based on our Salsa Dictionary approach in Figure 1. The dancers hold each other with both hands with the leader or follower having one arm bent behind them, while holding the hand of the partner. Due to the congruency of Bachata and Salsa, we were able to maintain the same structure in our coding technique. Data was input in the same fashion as with Salsa however element symbols and names changed according to Bachata. For the hammerlock move in Bachata, the leader and follower perform moves independent of each other with no common action performed. Hence the common action line will have no elements associated with it. Test cases were presented (Appendix C) to shows the successful implementation of the Bachata dance style based on the Salsa dictionary. Figure 7 shows the YAML configuration file that stores the Bachata moves that we implemented. Appendix B describes the YAML file specification for storing a move.

```
## YAML Template for Bachata Hammerlock move
lineArrayList:
   -name: "Handhold"
   elementstArray:
      -name: "Normal Hold"
      symbol: "N"
      -name: "Normal Hold"
      symbol: "N"
   -name: "Direction"
   elementsArray:
      -name: "Man facing Lady"
      symbol: "ML"
      -name: "Man facing Lady"
      symbol: "ML"
   -name: "Leader"
   elementsArray:
      -name: "Hammerlock"
      symbol: "HL"
   -name: "Follower"
   elementsArray:
      -name: "Hammerlock"
      symbol: "HL"
```

Figure 7: YAML configuration file to store Bachata Hammerlock move

To prove further extensibility, attempt was made to implement a full bronze move in Cha-Cha-Cha. Fan is an open position, whereby leader and follower are positioned facing each other starting with the man on the left and the lady on the right as shown in the Direction line in Figure 8. To perform the Fan position, the man leads the lady to his left side during the second half of the bar in a closed basic movement as seen in the latter 4 beats in Figure 8. He leads this by turning the lady to her left, releasing his right hand from her back and then extending the arm to the side. The computerization of Cha-Cha-Cha in this case differs from Salsa in terms of the move performed. Leader and follow perform steps independent of each other. Figure 9 represents the YAML file storing the data for the Fan move.

Handhold	C	C
Direction	ML	LM
Leader		F
Follower	F	

Figure 8: Fan move representation for Cha-Cha-Cha

```
## YAML Template for Cha-Cha-Cha Fan move
lineArrayList:
   -name: "Handhold"
   elementstArray:
      -name: "Closed Hold"
      symbol: "C"
      -name: "Closed Hold"
      symbol: "C"
   -name: "Direction"
   elementsArray:
      -name: "Man facing Lady"
      symbol: "ML"
      -name: "Lady facing Man"
      symbol: "LM"
   -name: "Leader"
   elementsArray:
      -name: "Fan"
      symbol: "F"
   -name: "Follower"
   elementsArray:
      -name: "Fan"
      symbol: "F"
```

Figure 9: YAML configuration file storing Cha-Cha-Cha Fan move

5.2 Software Usability Testing

5.2.1 Heuristic Evaluation

The developers identified a total of 24 usability problem instances over the 10 different types of heuristic methods. The most significant problems were regarding complexity of the API and the method parameters. Unnecessary complexity was discovered, and suggestions included removing some methods to reduce the complexity of the API such as the remove sequence method which was not needed. Unnecessary parameters were used in some methods particularly the Element class such as Element style and Element ID. These were advised to be removed as it caused confusion and was not used in the creation of a move. The Heuristic evaluation aimed at exploring all parts of API uniformly. Due to inspection-like methodology, the developers could only evaluate definition of classes, methods, interfaces etc, but were not able to analyse run-time behaviour. Hence further evaluation was conducted in Phase 2.

5.2.2 API Effectiveness and Accuracy

The collected material provided us with insights into understanding problems of the presentation and the tutorial as well as usability problems of the API. During the whole workshop 17 usability relevant issues with the API have been identified. Based on these tasks we found that the effort required to understand the semantics of API features was considered overall moderate, but a few method names were found confusing and potentially misleading. The method get Lines is used to get a dance move. Developers labelled this as misleading as get Move would have been the more appropriate name. It was also discovered that 5 out of the 12 participants were unfamiliar with the use of the factory pattern method. Since it is still possible to instantiate criteria without using the corresponding factory, all participants could successfully complete Tasks 2 and 3 even if they had problems with using factories. Figure 11 compares the results of the Heuristic evaluation to the results found when developers carried out tasks.

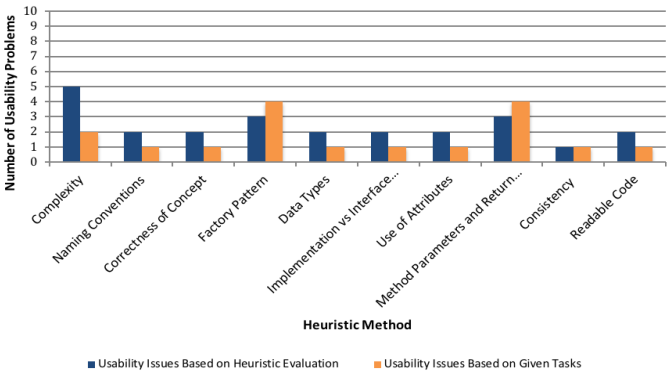

Figure 10: Comparison of Usability issues based on Heuristic evaluation versus usability findings on given tasks

5.2.3 System usability questionnaire

All the developers, who participated in the task assignment, took part in the interviews. The answers to these questions were critical in understanding whether we had successfully answered our research questions. Each question discussed participants findings in previous sections. Figure 11 classifies answers to each interview question into "yes", "no", and "sometimes", giving the number of responses in the form of a percentage. This provided a way for us to determine the success of the quality of our API developed.

5.2.3.1 Understandability

The 42% of the developers that voted that they had to learn several methods before completing a task were those that were unfamiliar with the factory design pattern in question 2. However, task completion was still possible without that knowledge. The remainder found the classes to be acceptable for the required goal.

5.2.3.2 Abstraction

The abstraction level of the API was largely considered appropriate and the functionalities were found suitable to solve the tasks. However, 33% of developers occasionally found it useful to peek at some implementation details in order to more readily understand relations between classes in question 5. This was due to them not having background knowledge on dance. The majority also found the API abstraction level appropriate to the tasks.

5.2.3.3. Reusability

The developers agreed that they managed to write concise code in an incremental fashion, and that their solutions were reusable to solve variants of the problems. They also agreed that the API offers different ways of implementing certain functionalities. Over 80% of developers positively answered question 6.

5.2.3.4. Learnability

The learning curve for the API is initially steep, as it requires to become familiar with a few non-trivial abstractions. After the initial learning phase, however, solving more advanced tasks becomes relatively simpler, as the developers got comfortable with the code. Over 90% of the participants agreed that they became more efficient after completing the first two "exploratory" tasks. This

rigorous in-house testing approach allowed us to gain valuable insights into the biggest problems that the developers struggled with. These problems are the most crucial to be corrected as they represent obstacles, which the developer has to overcome during the very first use. Such obstacles can then easily discourage a novice developer from using our framework. The evaluation revealed a total of 17 unique usability problems in our API.

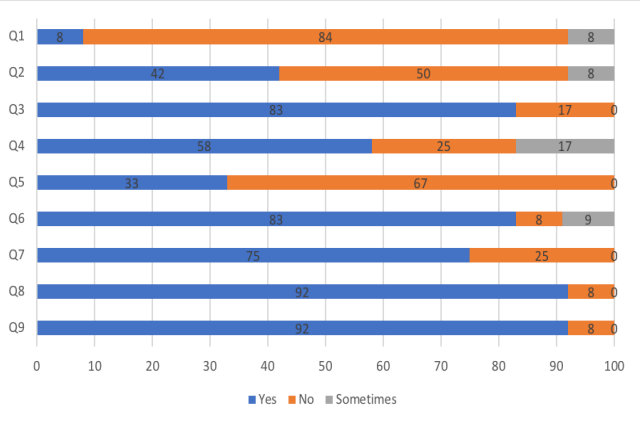

Figure 11: Summary of Interview Answers based on Research Questions

The approach and their findings differed for each part of the methodology. The findings from heuristic evaluation and findings from the tasks focus on usability problems regarding concept and structure of the API are detailed descriptions of problems in the API shown in Figure 10. During the task phase we could reveal more run-time problems that are not obvious and thus cannot be found in the heuristic evaluation. Interviews not only provided deeper understanding of these problems, but also revealed their attitude towards our final product as shown in Figure 11.

6. CONCLUSIONS

The API tool that we have developed in this study is driven by a novel approach to computerizing Salsa and various partner-based dance routines. Previous literature provided evidence that there was no uptake to computerizing paper-based notations for Salsa. The API was designed in an abstract manner to cater for extensibility to different dance styles. To assess the extensibility of the API we tested it across various dance styles concluding that our API can be used to extend other dances provided that the dance follows a 4-beat pattern. To ensure that meticulous attention to detail was paid to the design of our API, we conducted internal testing with developers. We evaluated the learnability and understandability of the API by comparing the expectations of programmers for our API to their actual performance using programming tasks. Through rigorous evaluation and testing we have proven our API framework to be abstract, learnable and reusable. Thus, creating a solid foundation for programmers to extend upon in the future.

7. FUTURE WORK

Due to the lack of tools developed to implement paper-based notations to teach dance, a greater scope is provided for future extensions to our work. Time restrictions in this project prevented us from implementing certain features, making future work more evident.

There are several other design patterns commonly used in APIs that we were unable to implement in the given time. These can be researched to improve flexibility and communication between objects in the API. Further studies can examine the best design patterns suited to implement to improve usability of the API. Ten Heuristic techniques were performed in the evaluation of our software quality. In the future, the remaining heuristics, can be used to obtain more valuable feedback on the API design. API's that are flagged as highly usable are more often chosen to be used by developers.

Since the API is flexible and abstract in its design, it provides a basis for developers to extend upon by allowing them to implement other dances. If time permitted, minor modifications would have been made to the code to implement dances that contain any number of beats and not being restricted to multiples of a 4-beat count.

Our dance notation can also be extended from modelling dance gestures to modelling human body movement in other cultural activities. The underlying structure of our code allows for easy modification into other forms of human movement. Activities such as sport can be represented with our notation by defining the core parts of the body used as a line and the elements relate to the action performed in a line. For example, in boxing, a line would be hand position of a boxer and elements would be the orientation of the hand depending on the type of boxing move performed. This could be a potential tool to provide a different approach to the way sport is taught.

Implementation of the suggestions mentioned above would play a significant role in encouraging the preservation of not only dance, but other intangible cultural activities in the future whilst also benefitting the members in these industries.

ACKNOWLEDGMENTS

An undertaking of this sort would not have been achievable without the support of those around me.

Foremost, I would like to convey my sincere gratitude to my supervisor Maria Keet for her ongoing support and patience throughout this project, and Angus Prince for his valuable contribution towards this project. My sincere thanks also go to my colleagues Alka Baijnath and Jordy Chetty, both of whom have supported and motivated me, making this an enjoyable experience. Finally, I would like to offer my heartfelt appreciation to my family: My parents, Shashi and Anupa Marajh who have been my continuous source of support, advice and encouragement.

REFERENCES

[1] Alpert, P.T. 2011.The Health Benefits of Dance. Home Health Care Management & Practice.DOI= https://doi.org/ 10.1177/1084822310384689.

[2] Bernstein, N. (1967) The co-ordination and regulation of movement. London: Pergamon Press

[3] Bloch, J. Effective Java Programming Language Guide, Addison-Wesley, Boston, MA, 2001.

[4] Bosse, J. 2008. Salsa Dance and the Transformation of Style: An Ethnographic Study of Movement and Meaning in a Cross-Cultural Context. Dance Research Journal, 40(1), 45-64. DOI=10.1017/S0149767700001364.

[5] Choensawat, W., Nakamura, M. and Hachimura, K. (2014). GenLaban: A tool for generating Labanotation from motion capture data. Multimedia Tools and Applications, 74(23), pp. 10823-10846.

[6] Christine von Renesse and Volker Ecke. 2011. Mathematics and Salsa dancing. https://doi.org/10.1080/17513472.2010.491781

[7] Colla J. Mac Donald. (1991). Creative Dance in Elementary Schools: A Theoretical and Practical Justification. Canadian Journal of Education / Revue Canadienne De L'éducation, 16(4), 434-441. doi:10.2307/1495255
[8] DonHerbison-Evans. 1988.Dance, Video, NotationandComputers.Leonardo21, 1 (1988), 45–50. http://www.jstor.org/stable/1578415.
[9] Gough, Matthew. (2004). Notation Reloaded: eXtensible Dance Scripting Notation. Body, Space & Technology.
[10] Gough, Matthew. (2005). Towards Computer Generated Choreography: Epikinetic Composition.
[11] Hutchinson Guest, A (1989) Choreographics: A Comparison of Dance Notation Systems from the Fifteenth Century to the Present. Routledge London and New York.
[12] Hatol, J. (2006). MOVEMENTXML: A representation of semantics of human movement based on Labanotation (Doctoral dissertation, School of Interactive Arts and Technology-Simon Fraser University.
[13] Jakob Nielsen and Rolf Molich. (1990). Heuristic evaluation of user interfaces. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.
[14] Joshua Bloch. (2006). How to design a good API and why it matters. In Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications (OOPSLA '06). ACM, New York, NY, USA, 506-507. DOI: https://doi.org/10.1145/1176617.1176622
[15] Katerina El Raheb, Aristotelis Kasomoulis, Akrivi Katifori, Marianna Rezkalla, and Yannis Ioannidis. (2018). A Web- based system for annotation of dance multimodal recordings by dance practitioners and experts.
[16] Kelmendi, T. (2017). Analiza Dhe Krahasimi I Modelit Waterfall Me Modele Të Tjera Për Zhvillimin E Sistemeve (Analysis and Comparison of Waterfall Model with Other Models for Software/System Development). SSRN Electronic Journal.
[17] LarsWilke, ThomasW.Calvert, RhondaRyman, andIleneFox. (2005).Fromdance notation to human animation: The LabanDancer project. Journal of Visualization and Computer Animation 16 (2005), 201–211.
[18] Leijen, Ä., Lam, I., Wildschut, L., Simons, P.R.J. (2009). Difficulties teachers report about students' reflection: Lessons learned from dance education. Teaching in Higher Education, 14(3), 315 - 326.
[19] Nakamura, Minako. (2019). An XML Representation of Labanotation, LabanXML, and its Implementation on the Notation Editor LabanEditor2.
[20] Nakata, T (2002) Generation of whole-body expressive movement based on somatical theories: Proceedings of the second international workshop on Epigemetic Robotics, pp.105-114.
[21] R. B. Watson, "Improving software API usability through text analysis: A case study," (2009) IEEE International Professional Communication Conference, Waikiki, HI, 2009, pp. 1-7. doi: 10.1109/IPCC.2009.5208679
[22] SalsaIsGood. (2001). A Dictionary for Salsa and Mambo moves. Retrieved April 24, 2019 from http://salsaisgood.com/dictionary/main_Salsa_Method.html
[23] Scaffidi, C. (2006). Why are APIs difficult to learn and use? Crossroads, 12(4), pp.4-4.
[24] Singh, B. Beatty, J. and Ryman, R. (1983). A graphics editor for benesh movement notation. Computer Graphics, 17(3), pp.51-62.
[25] Stylos, J., Clarke, S. and Myers, B. (2006) Comparing API Design Choices with Usability Studies
[26] Sweet, J. and Royce, A. (1978). The Anthropology of Dance. Dance Research Journal
[27] Victoria Watts (2015) Benesh Movement Notation and Labanotation: From Inception to Establishment (1919–1977), Dance Chronicle, 38:3, 275-304, DOI: 10.1080/01472526.2015.1085227

APPENDIX

A SYSTEM USE CASE DIAGRAM

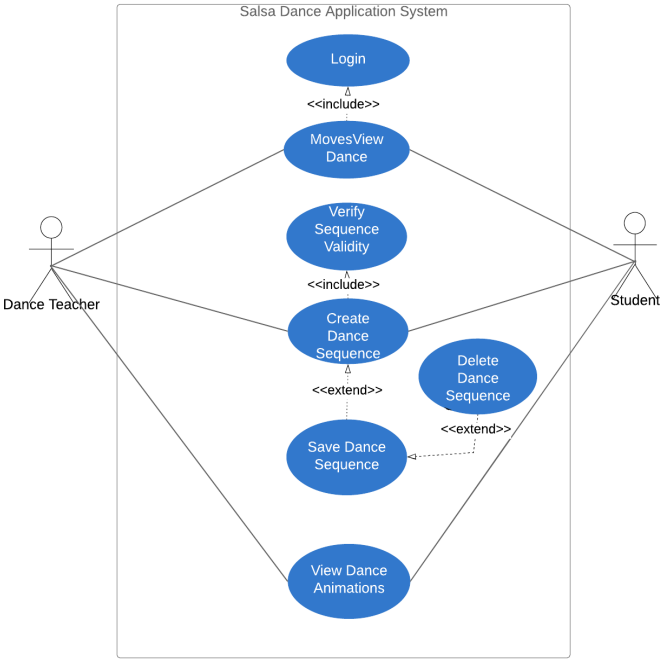

Figure 3: Use Case Diagram representing the final Salsa dance system.

B CONFIGURATION FILE

A YAML file is a human-readable data serialisation file used for configuration purposes. It is a description file used to store data from the system. It is used as an expressive, extensive and portable description file between programming languages. It defines key-value variables for every item. For example, the Move file is specified as follows:

- lineArrayList:
  -name: specifies the name of the Line
  elementstArray: an array of Element objects
  -name: specifies the name of the Element
  symbol: the symbol used to denote the dance step

These are defined for an Element object. The scope is defined with indentation within the YAML file. This allows the user to specify values that she wishes to derive from the file. When all values are defined, the system retries the elements defined above. When a file is loaded, the system searches for the related file and retrieves the values defined.

C TEST CASES

Test Scenario	Test Case	Test Step	Test Data	Expected Result	Actual Result	Pass/Fail
Verify element input	Check correct elements entered	1. Enter element name 2. Enter element symbol	Name: Normal Hold Symbol: N	Element must be entered correctly	Element correctly entered	Pass
Check Line creation functionality	Verify that line class consists of elements	1. Enter line name 2. Add element objects to a line	Name: Leader Name: Normal Hold Symbol: N	Line must consist of elements previously defined	Line displays elements with corresponding name	Pass
Test Move creation functionality	Verify that a move contains lines populate with elements	1. Return method to produce a line	Name: Leader Name: Normal Hold Symbol: N Name: Normal Hold Symbol: N	Move must include Lines with elements associated	Move created correctly	Pass
Test Sequence Functionality	Verify that a sequence contains a list of moves	1. Return method to produce moves	All data stored in YAML file for moves	Sequence must consists of a list of moves	Sequence created correctly	Pass

Figure 12: Test Case for Salsa Cross Body Move

Test Scenario	Test Case	Test Step	Test Data	Expected Result	Actual Result	Pass/Fail
Verify element input	Check correct elements entered	1. Enter element name 2. Enter element symbol	Name: Normal Hold Symbol: N	Element must be entered correctly	Element correctly entered	Pass
Check Line creation functionality	Verify that line class consists of elements	1. Enter line name 2. Add element objects to a line	Name: Leader Name: Normal Hold Symbol: N	Line must consist of elements previously defined	Line displays elements with corresponding name	Pass
Test Move creation functionality	Verify that a move contains lines populate with elements	1. method to produce a line	Name: Handhold Element Name: "Normal Hold" Symbol: "N" Element Name: "Normal Hold" Symbol: "N" Name: Direction Element Name: Man facing Lady Symbol: ML Element Name: Man facing Lady Symbol: ML Name: Leader Element Name: Hammerlock Symbol: HL Name: Follower Element Name: Hammerlock Symbol: HL	Move must include Lines with elements associated	Move created correctly	Pass
Test Sequence Functionality	Verify that a sequence contains a list of moves	1. Return method to produce moves	All data stored in YAML file in Figure 7	Sequence must consists of a list of moves	Sequence created correctly	Pass

Figure 13: Test Case Bachata Hammerlock Move

Test Scenario	Test Case	Test Step	Test Data	Expected Result	Actual Result	Pass/Fail
Verify element input	Check correct elements entered	1. Enter element name 2. Enter element symbol	Name: Normal Hold Symbol: N	Element must be entered correctly	Element correctly entered	Pass
Check Line creation functionality	Verify that line class consists of elements	1. Enter line name 2. Add element objects to a line	Name: Leader Name: Normal Hold Symbol: N	Line must consist of elements previously defined	Line displays elements with corresponding name	Pass
Test Move creation functionality	Verify that a move contains lines populate with elements	1. method to produce a line	Name: Handhold Element Name: Closed Hold Symbol: C Element Name: Closed Hold Symbol: C Name: Direction Element Name: Man facing Lady Symbol: ML Element Name: Lady facing Man Symbol: LM Name: Leader Element Name: Fan Symbol: F Name: Follower Element Name: Fan Symbol: F	Move must include Lines with elements associated	Move created correctly	Pass
Test Sequence Functionality	Verify that a sequence contains a list of moves	1. Return method to produce moves	All data stored in YAML file in Figure 9	Sequence must consists of a list of moves	Sequence created correctly	Pass

Figure 14: Test Case for Cha-Cha-Cha Fan Move