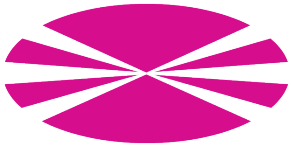


UNIVERSIDADE DA CORUÑA

Herramienta para armonización musical a través de Answer Set Programming

Tool for musical harmonization through Answer Set Programming

Rodrigo Martín Prieto

February 18, 2016

Director:
José Pedro Cabalar Fernández

Motivation

- Musical teaching is still very traditional nowadays.
- Self-teaching of music theory is hard.

Motivation

- Musical teaching is still very traditional nowadays.
- Self-teaching of music theory is hard.
- There are not many tools to aid and guide students and self-taught students.
- Composition tools seek results assuming that the user knows musical theory.
- There are intelligent composers: CHASP, Vox Populi, ANTON...

Example: Harmonization

- Harmony is a very important subject in music theory learning
- Choral music is the root of this subject

Example: Harmonization

- Harmony is a very important subject in music theory learning
- Choral music is the root of this subject
- Exercises consist in choosing chords sequences and completing musical pieces
- Already existing tools do not apply to this particular field

Goals

1. Harmonize and annotate chords over any musical score

Goals

1. Harmonize and annotate chords over any musical score
2. Given a certain harmonization, be able to complete on purpose blank sections of any incomplete voice of the score

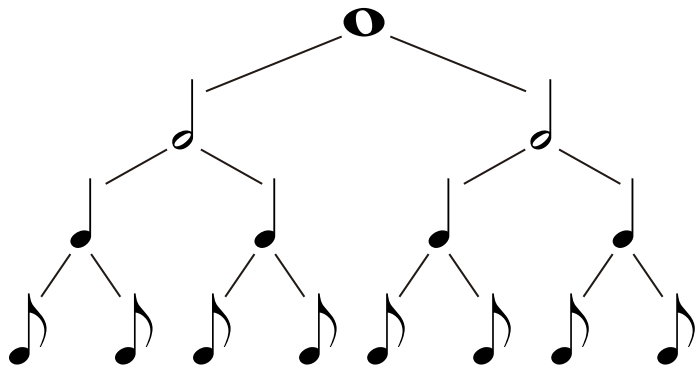
Goals

1. Harmonize and annotate chords over any musical score
2. Given a certain harmonization, be able to complete on purpose blank sections of any incomplete voice of the score
3. Add new voices that complement the voices already in the score

Overview

Figures and Rhythm

- Every note is represented by a figure that determines it's length
- Each figure can be subdivided in two shorter figures
- Rhythm is created by combining figures of different lengths with special symbols called silences



Melody

- Horizontal dimension of music
- Pitch is represented by the height at which the note is written, higher position means higher pitch

Play

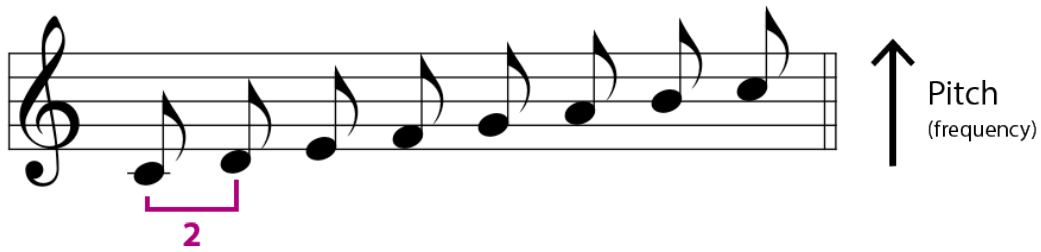
Melody

- Horizontal dimension of music
- Pitch is represented by the height at which the note is written, higher position means higher pitch
- Interval: Jump difference between two notes (including both endpoints)

▸ Play

Melody

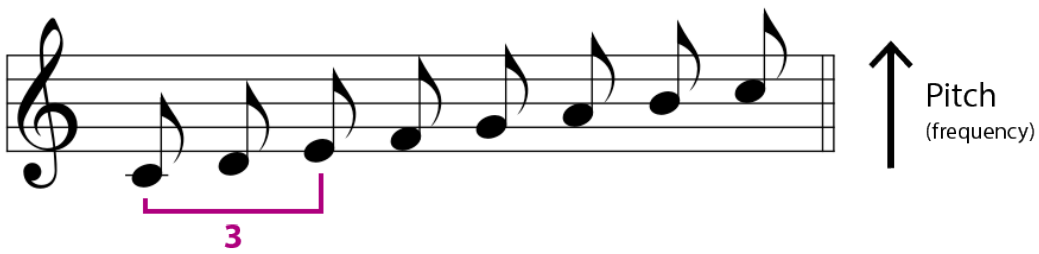
- Horizontal dimension of music
- Pitch is represented by the height at which the note is written, higher position means higher pitch
- Interval: Jump difference between two notes (including both endpoints)



▸ Play

Melody

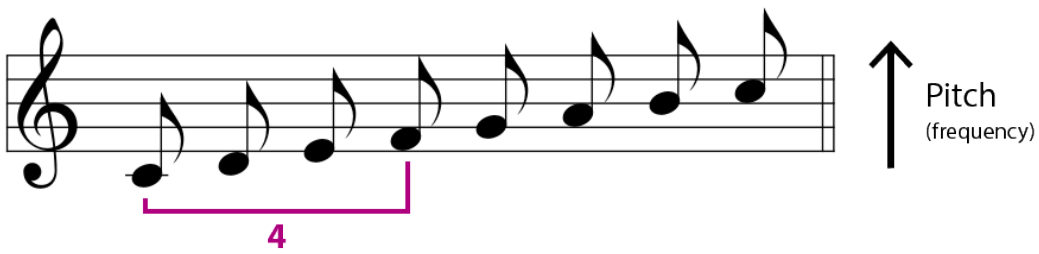
- Horizontal dimension of music
- Pitch is represented by the height at which the note is written, higher position means higher pitch
- Interval: Jump difference between two notes (including both endpoints)



▸ Play

Melody

- Horizontal dimension of music
- Pitch is represented by the height at which the note is written, higher position means higher pitch
- Interval: Jump difference between two notes (including both endpoints)

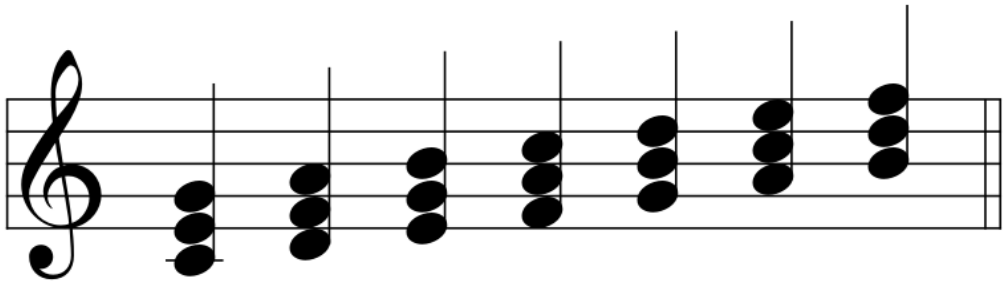


▸ Play

Tonality

Harmony

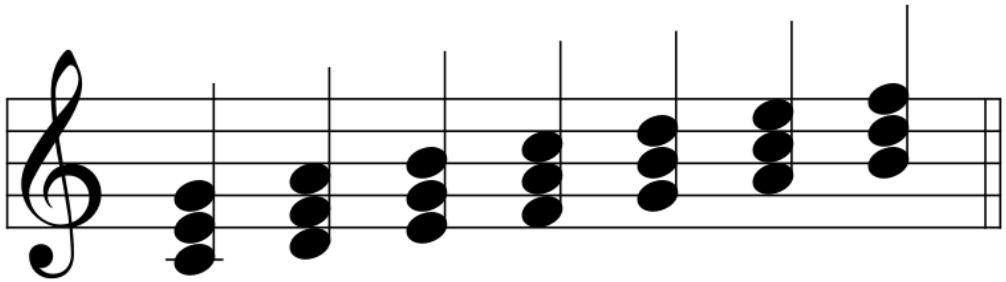
- Vertical dimension of music
- Only present in polyphonic pieces or pieces with polyphonic instruments
- Two notes or more of different voices that play at the same time form a chord



Play

Harmony

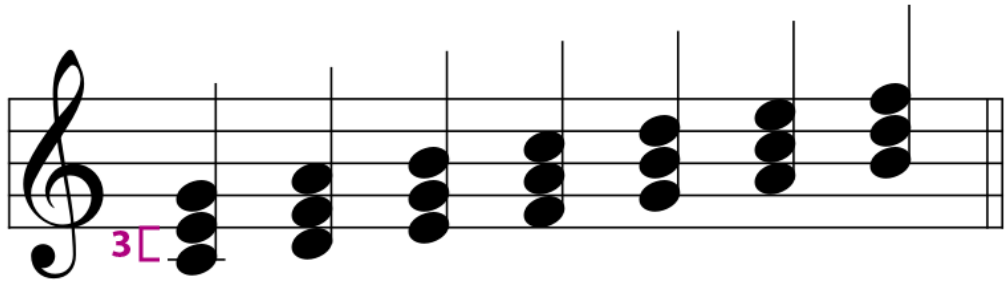
- Vertical dimension of music
- Only present in polyphonic pieces or pieces with polyphonic instruments
- Two notes or more of different voices that play at the same time form a chord
- Fundamental chords of the scale are built adding the third and fifth notes of the root



▸ Play

Harmony

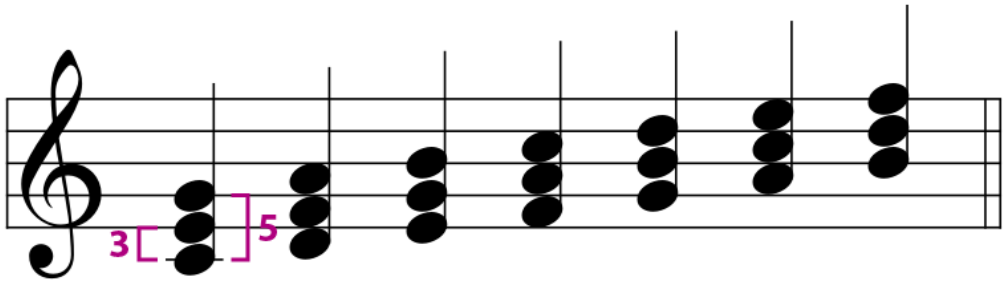
- Vertical dimension of music
- Only present in polyphonic pieces or pieces with polyphonic instruments
- Two notes or more of different voices that play at the same time form a chord
- Fundamental chords of the scale are built adding the third and fifth notes of the root



▸ Play

Harmony

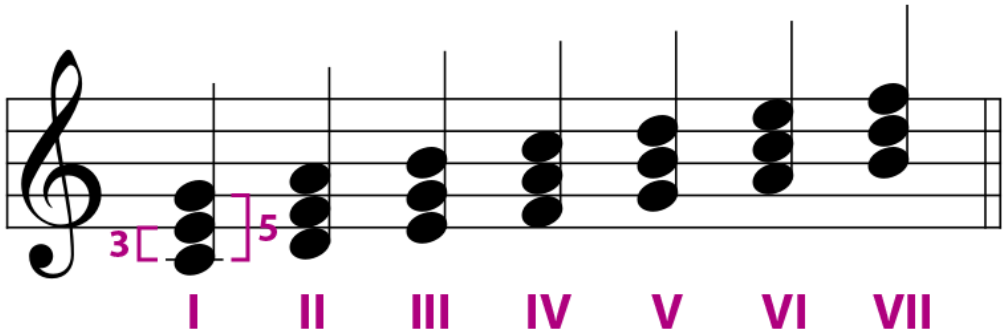
- Vertical dimension of music
- Only present in polyphonic pieces or pieces with polyphonic instruments
- Two notes or more of different voices that play at the same time form a chord
- Fundamental chords of the scale are built adding the third and fifth notes of the root



▸ Play

Harmony

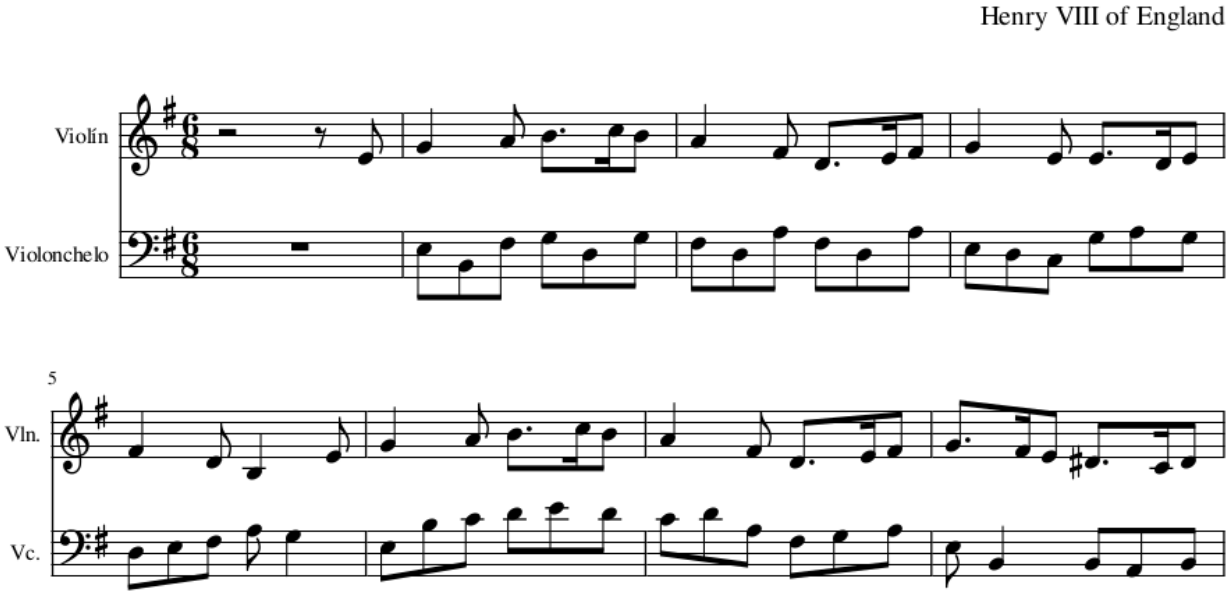
- Vertical dimension of music
- Only present in polyphonic pieces or pieces with polyphonic instruments
- Two notes or more of different voices that play at the same time form a chord
- Fundamental chords of the scale are built adding the third and fifth notes of the root



Play

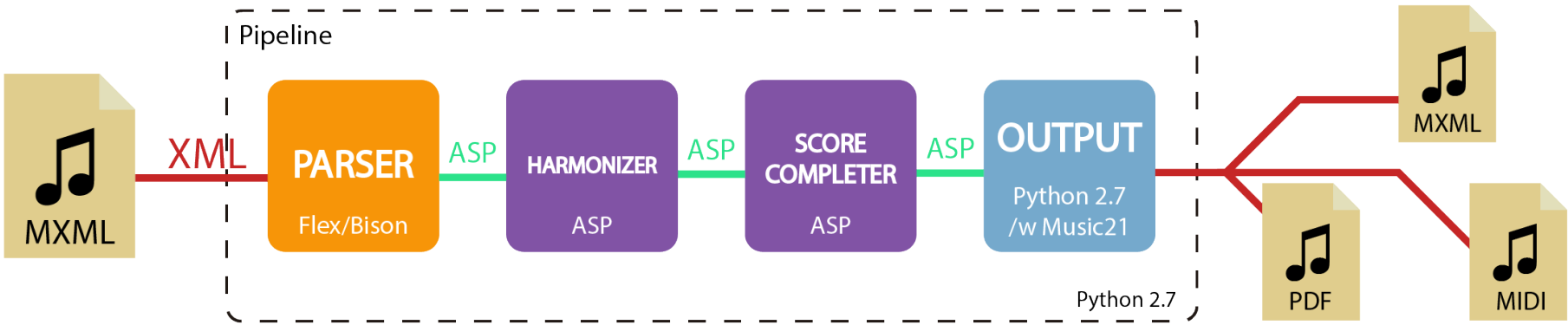
Overview

Demonstration: Greensleeves

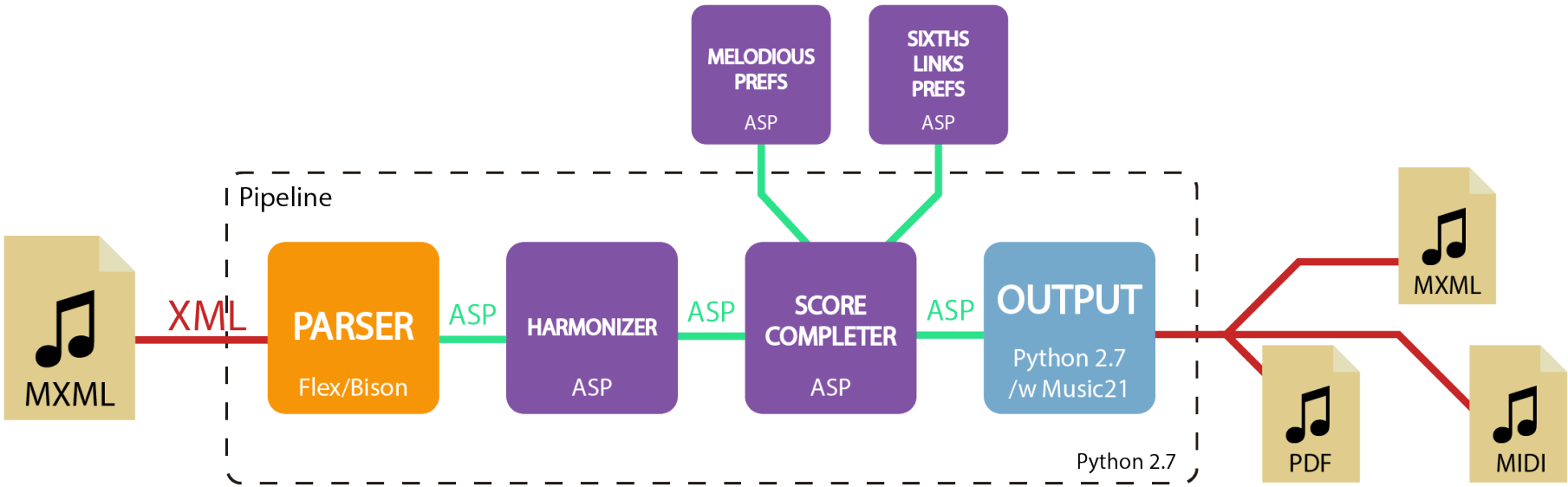


Overview

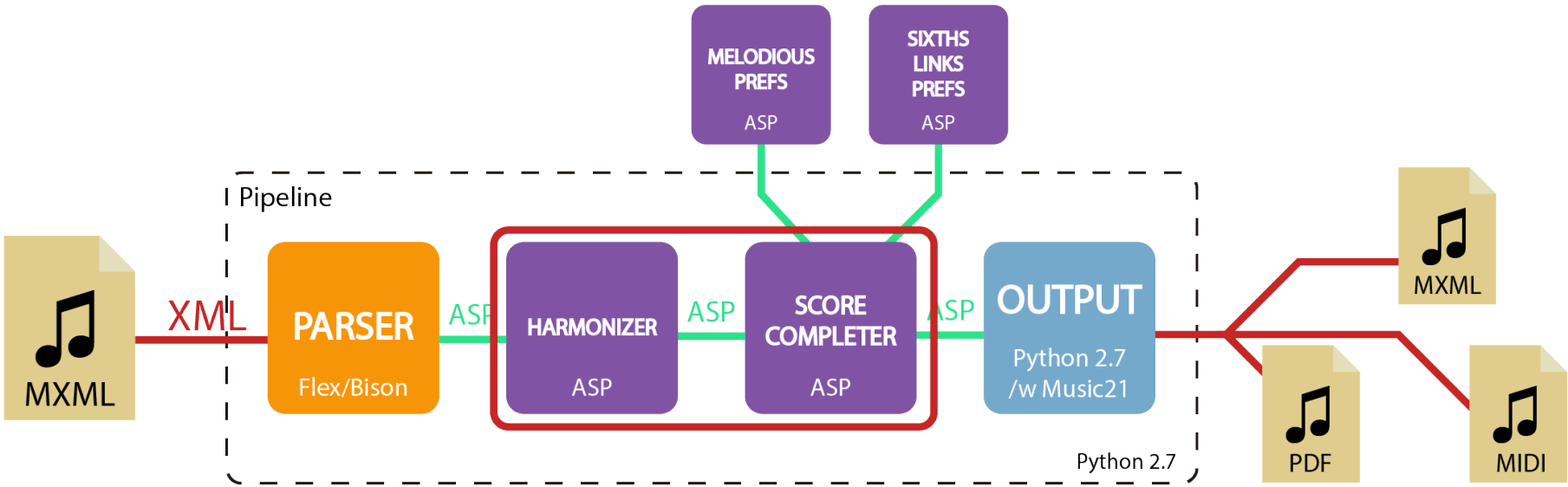
haspie's Architecture



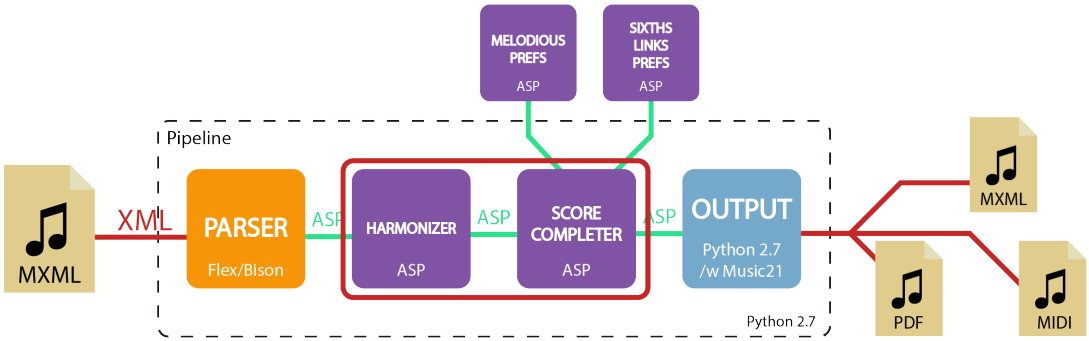
haspie's Architecture



haspie's Architecture



ASP Core



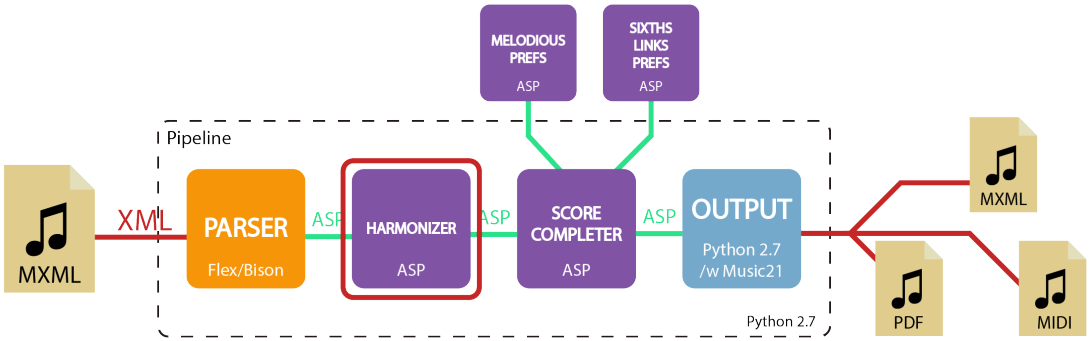
Answer Set Programming:

- Independent of the solving process and its heuristics
- The power and flexibility of ASP lays on this independence
- The problem only needs to be specified by rules and constraints

ASP Example: the 8 queens problem

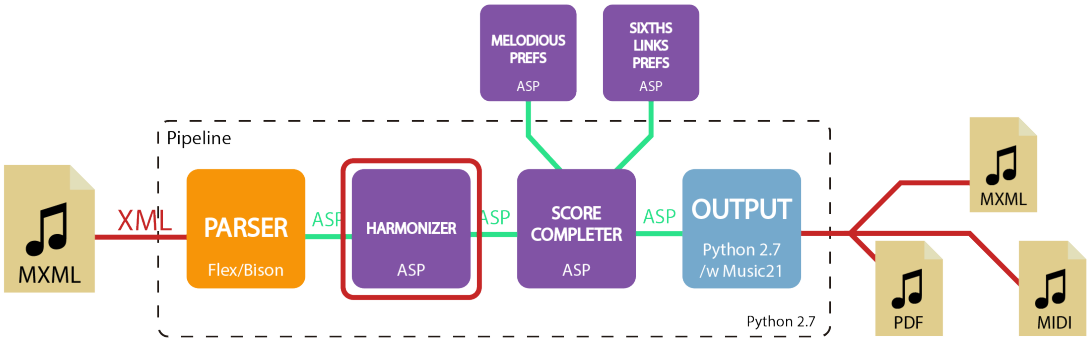
```
% Facts
number(1..8).

% (1) Generate: potential solutions
1 { q(X,Y) : number(Y) } 1 :- number(X).

% (2) Define: auxiliary predicates
cell(X,Y) :- number(X), number(Y).
diff(X,Y,Z,T) :- cell(X,Y), cell(Z,T), X!=Z.
diff(X,Y,Z,T) :- cell(X,Y), cell(Z,T), Y!=T.

% (3) Test: constraints prune invalid solutions
:- q(X,Y), q(Z,Y), X!=Z.
:- q(X,Y), q(X,Z), Y!=Z.
:- q(X,Y), q(Z,T), diff(X,T,Z,T), |X-Z|=|Y-T|.
```

Harmonization



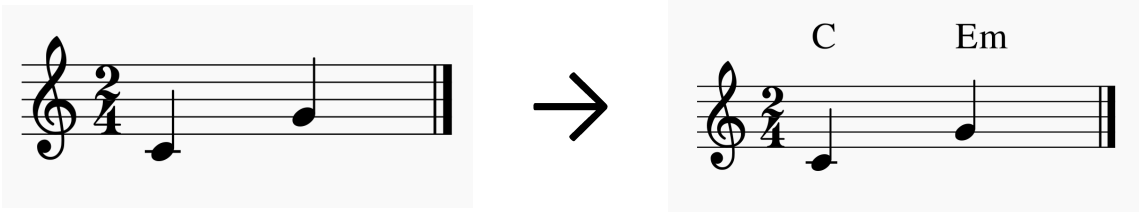
- Notes are converted to grades of the scale given the key and mode
- Chords are assigned to the harmonizable times of the score
- Errors are computed and the solver determines the fittest chords for each section

Harmonization

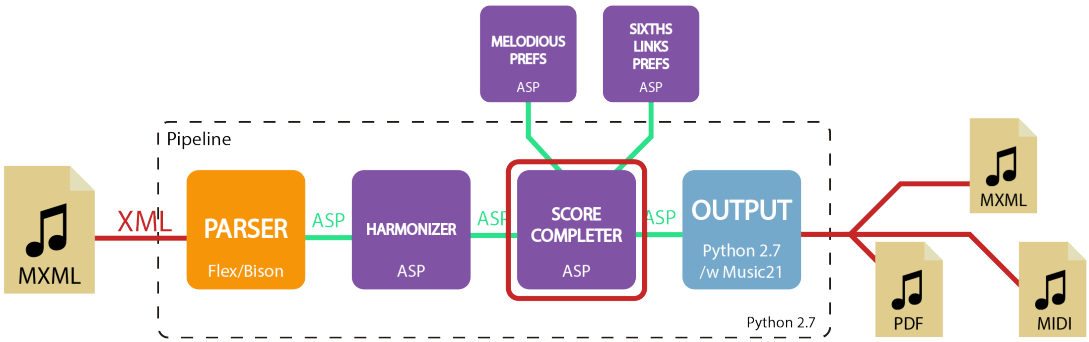


- Notes are converted to grades of the scale given the key and mode
- Chords are assigned to the harmonizable times of the score
- Errors are computed and the solver determines the fittest chords for each section

```
1 { chord(HT,C) : pos_chord(C) } 1 :- htime(HT).
```

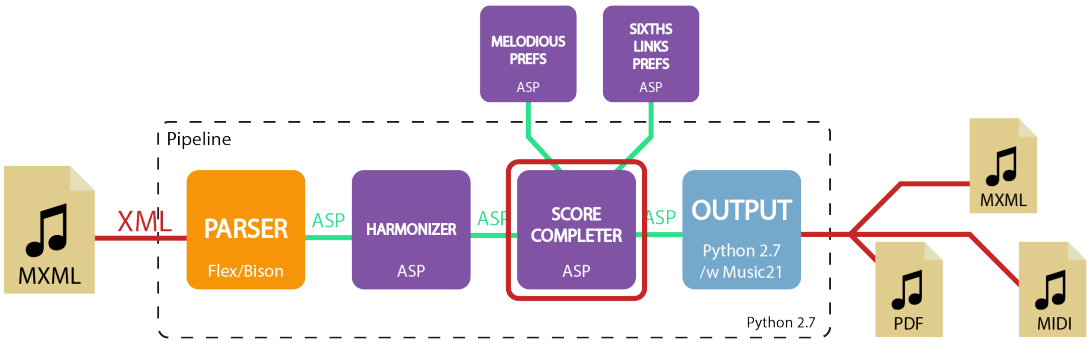


Score Completion



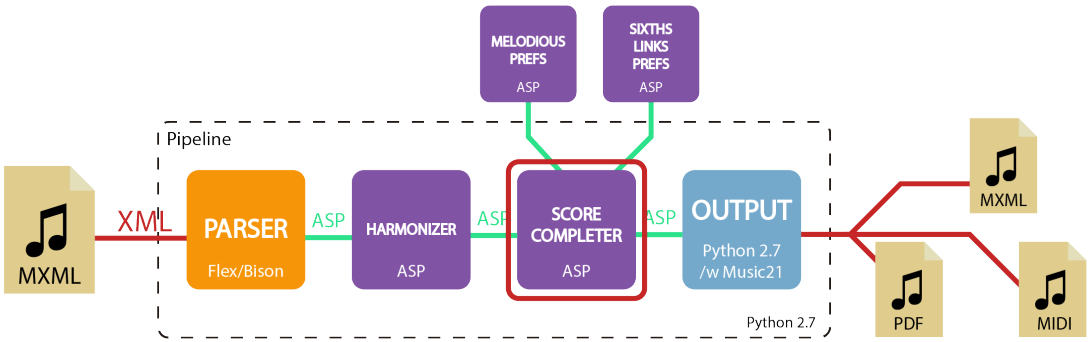
- Only used if there are new voices or sections that need to be completed

Score Completion

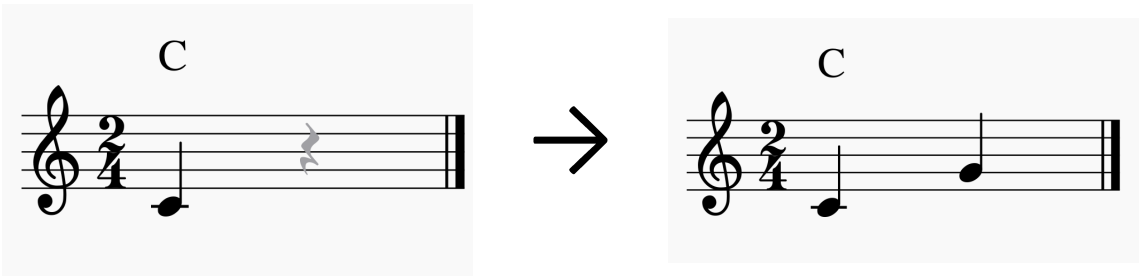


- Only used if there are new voices or sections that need to be completed
- Given the incomplete or new voices' *tessiturae* notes are assigned to the available positions

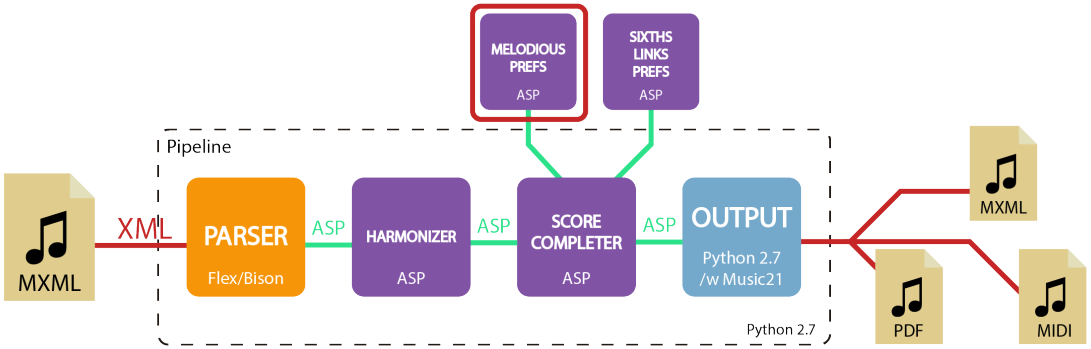
Score Completion



- Only used if there are new voices or sections that need to be completed
- Given the incomplete or new voices' *tessiturae* notes are assigned to the available positions
- Errors are computed and solver determines the fittest notes for each time

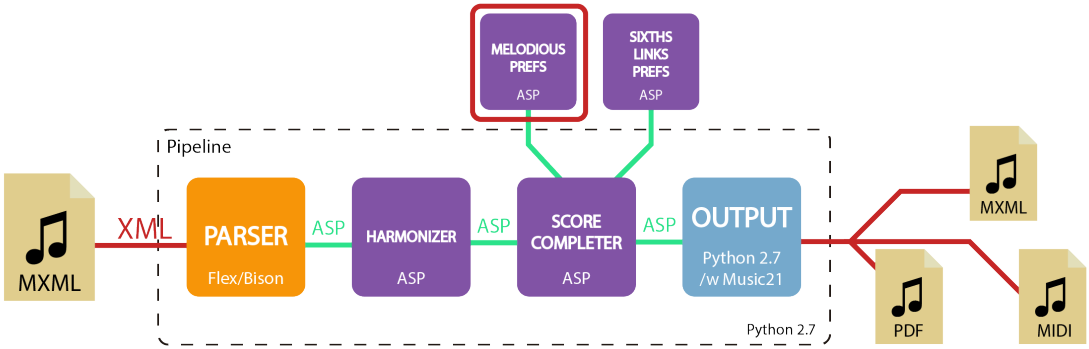


Melodious Preferences Module



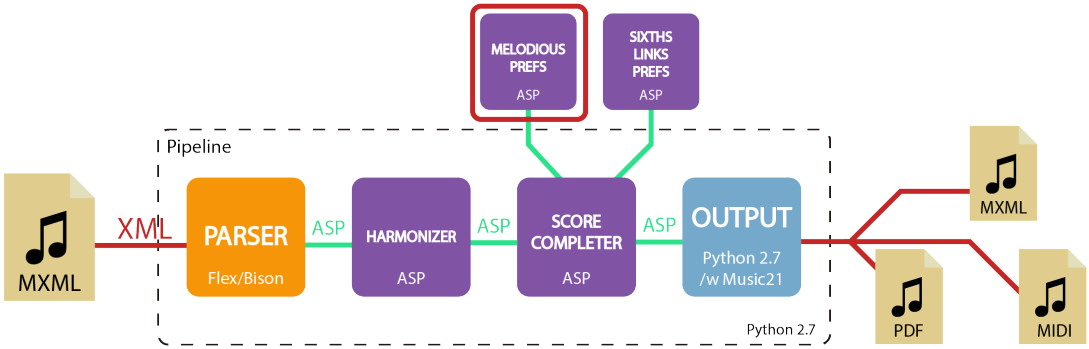
- Although not composing melodiously, this module improves the output in a melodious way

Melodious Preferences Module



- Although not composing melodiously, this module improves the output in a melodious way
- Checks the tendency of the voices already on the score and makes the new voices imitate them

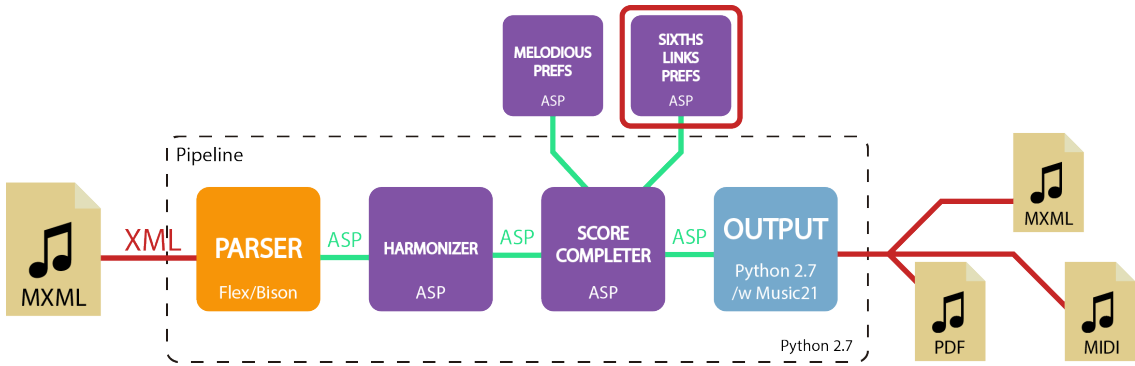
Melodious Preferences Module



- Although not composing melodiously, this module improves the output in a melodious way
- Checks the tendency of the voices already on the score and makes the new voices imitate them
- Smoothens the melodic jumps between notes of a same voice
- Reduces the number of consecutive repeated sounds

```
melodic_jump(V,J,B1,B2) :- out_note(V,N1,B1),
out_note(V,N2,B2),(B1+1) == B2, beat(B1+1), J=|N1-N2|.
```

Sixths Link Preferences Module



- Progressions of the second inversion of chords are very common in choral music
- Creates a per-time harmonization of the score
- Finds patterns of second inversion of chords linked in other voices
- Continues and creates new progressions of this kind if possible

```
second_inversion(C,B) :- harmonic_step(V1,V2,4,B),
harmonic_step(V1,V3,6,B), beat(B), unary_chord(B,C),V2!=V3.
```

User Configuration

ASP optimization:

- The style of the resulting scores produced by the tool is determined by the optimization of many predicates

User Configuration

ASP optimization:

- The style of the resulting scores produced by the tool is determined by the optimization of many predicates
- These optimizations are weighted to be able to specify the significance of each of the measured predicates

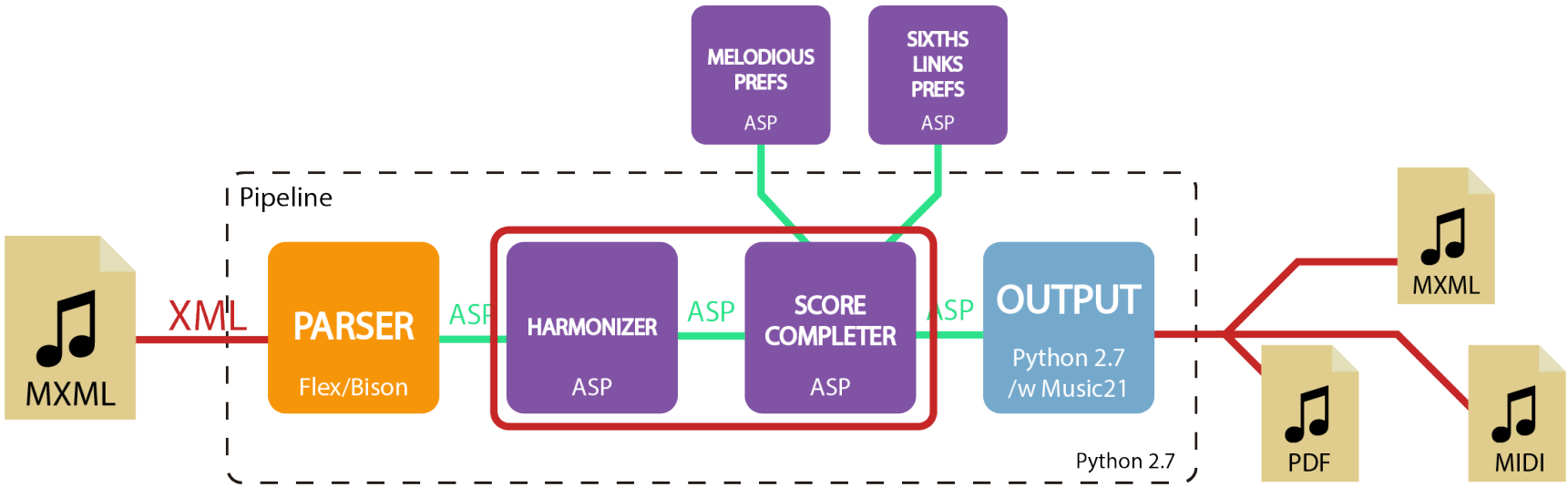
User Configuration

ASP optimization:

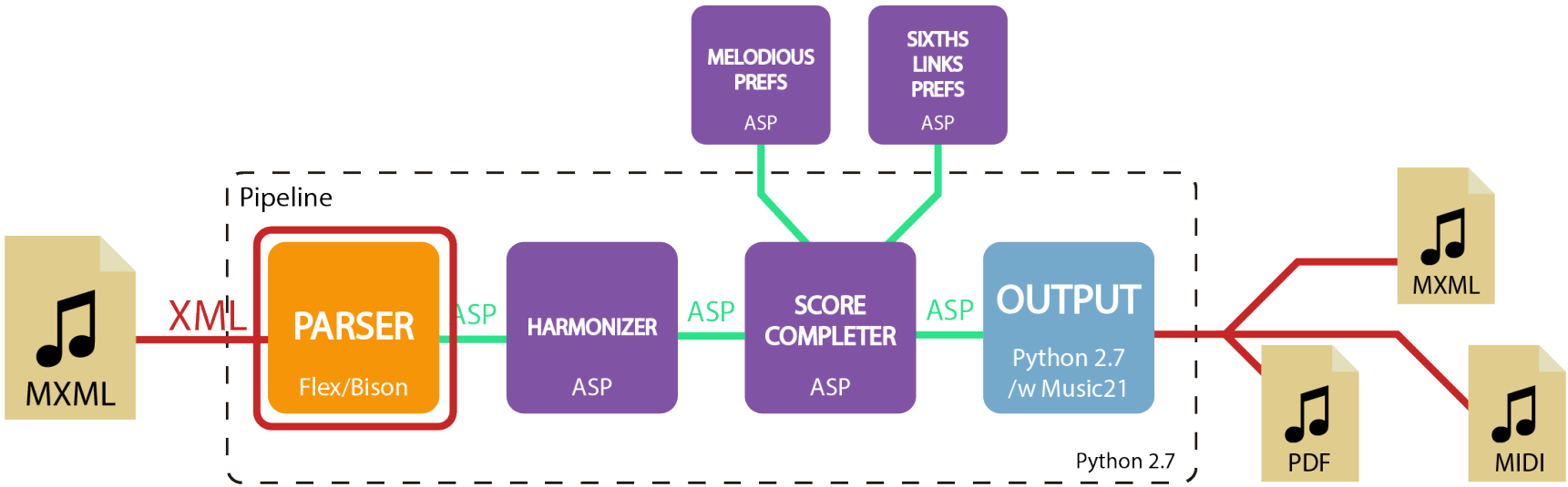
- The style of the resulting scores produced by the tool is determined by the optimization of many predicates
- These optimizations are weighted to be able to specify the significance of each of the measured predicates
- Users can define their own preferences by making use of configuration files

```
#minimize[out_error(_,_) = chord_errorinstrongw
                         @ chord_errorinstrongp].
#minimize[same_chord(_,_) = chord_samechordw
                         @ chord_samechordp].
#minimize[out_error_weak(_,_) = chord_errorinweakw
                         @ chord_errorinweakp].
```

haspie's Architecture



haspie's Architecture

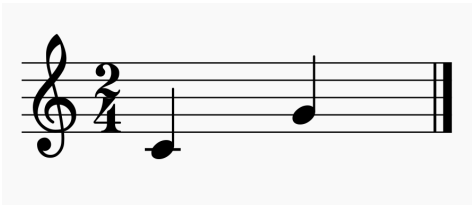


Parser and Preprocessor

- The project also included the development of a lightweight MusicXML parser
- Written in C with the libraries Flex and Bison
- Transforms the score in MusicXML to the ASP logic facts that the ASP module uses later

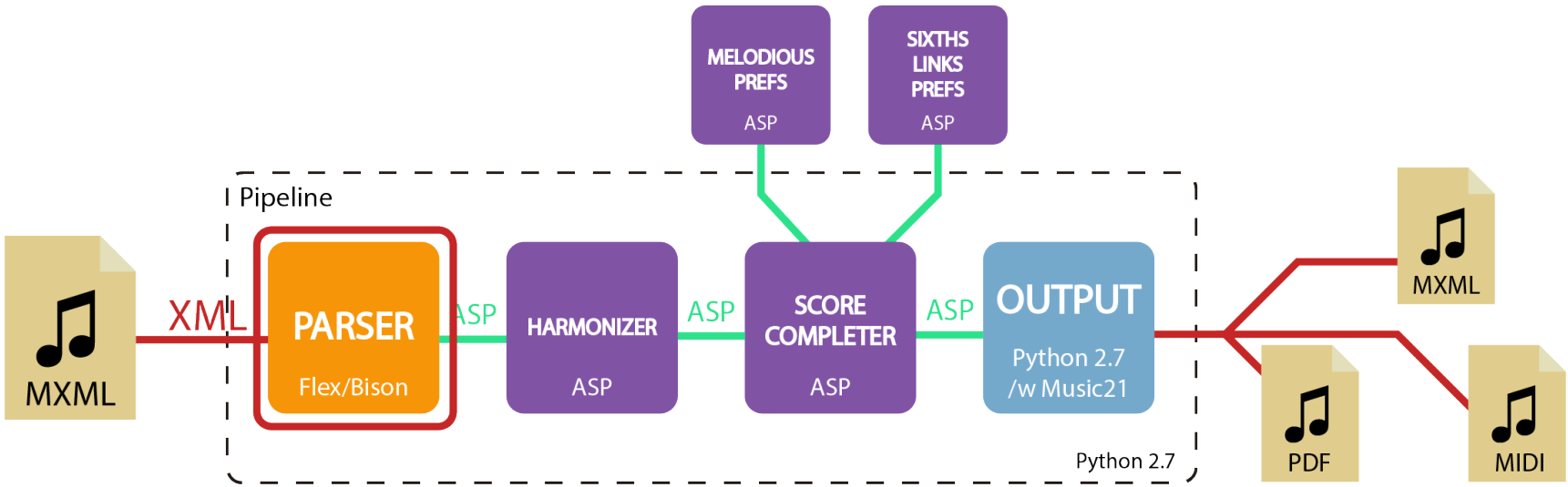
Parser and Preprocessor

- The project also included the development of a lightweight MusicXML parser
- Written in C with the libraries Flex and Bison
- Transforms the score in MusicXML to the ASP logic facts that the ASP module uses later
- Performs various tasks as:
  - Subdivides notes to the length of the smallest figure in the score
  - Detects most likely key from the score's clef
  - Reads measure sizes
  - Transforms chord names annotated on score to grades

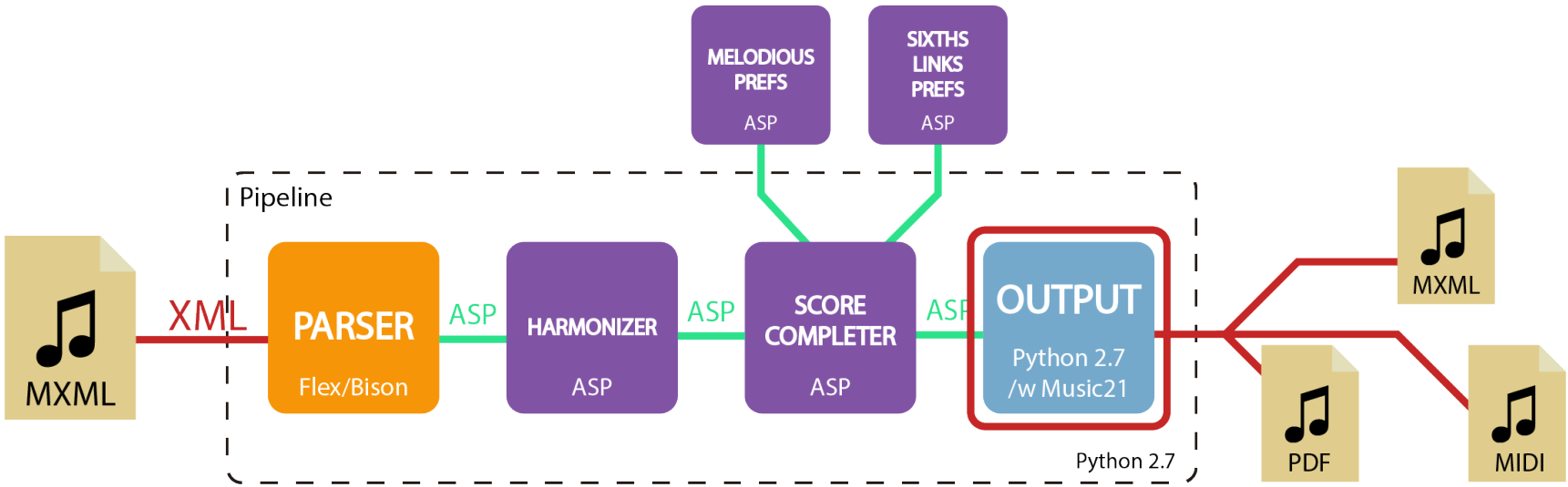


```
voice_type(1, violin).
figure(1,1,1).
note(1, 60, 1).
figure(1,1,2).
note(1, 67, 2).
measure(2, 0).
real_measure(2, 4, 0).
```

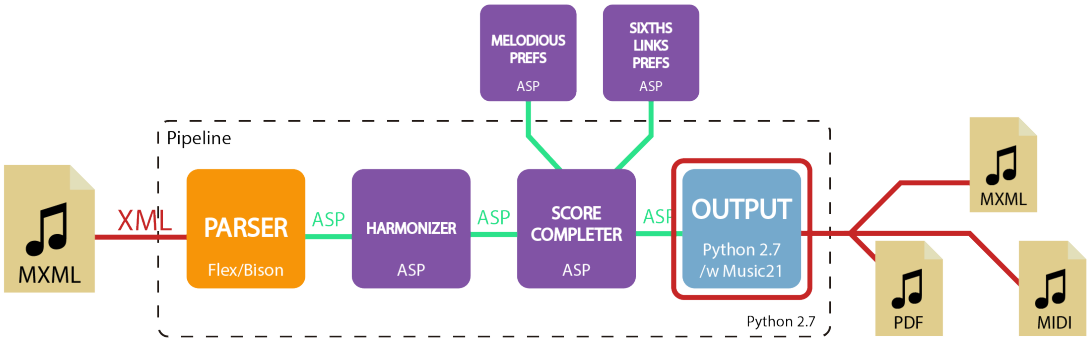
haspie's Architecture



haspie's Architecture

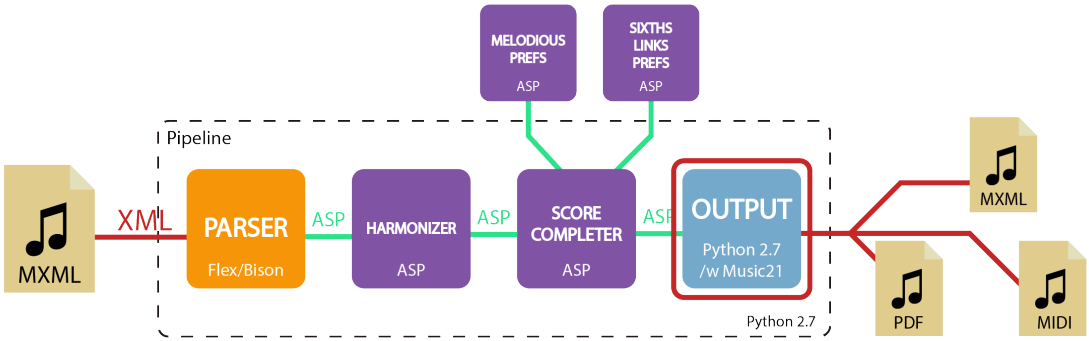


Output Module



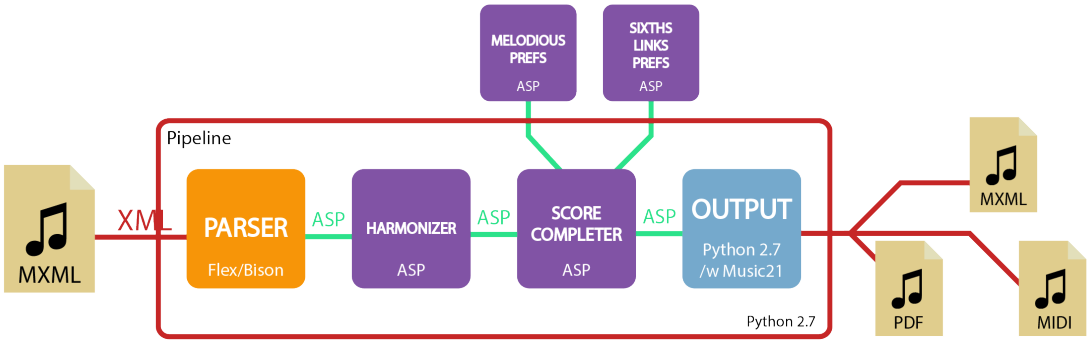
- Written in Python with the toolkit Music21
- Transforms the internal representation of the solution to a Music21 representation

Output Module



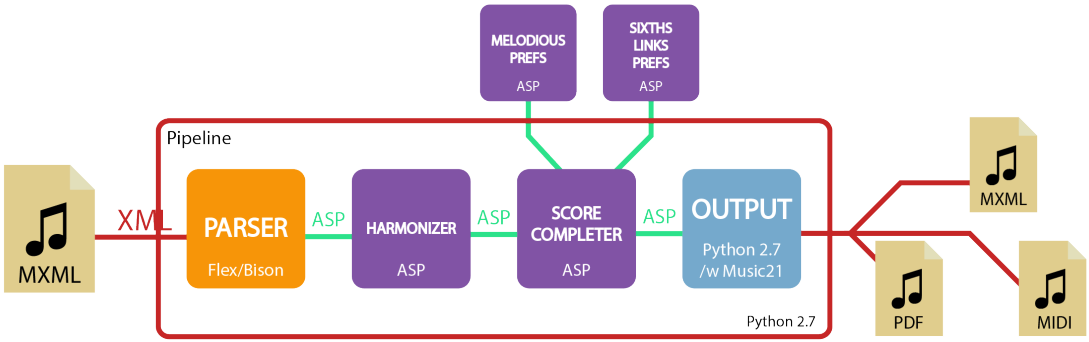
- Written in Python with the toolkit Music21
- Transforms the internal representation of the solution to a Music21 representation
- Exports the Music21 representation to the desired format
- Some supported formats are Lilypond, PDF, Musescore, MusicXML or MIDI
- Allows the result to be saved or directly shown/played

Pipeline



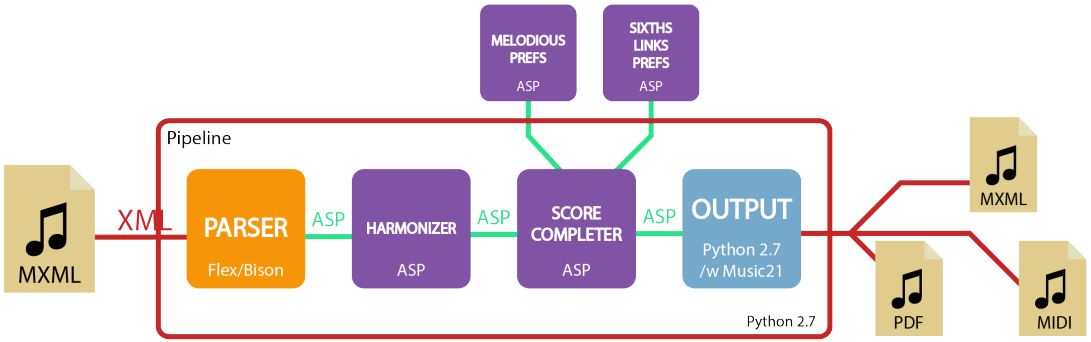
- Written in Python

Pipeline



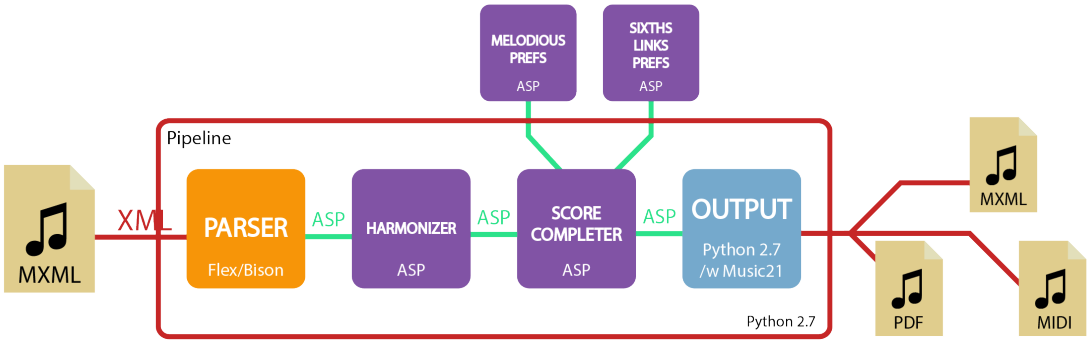
- Written in Python
- Coordinates the different modules secuentially
- Gives feedback to the user throught the command line

Pipeline



- Written in Python
- Coordinates the different modules secuentially
- Gives feedback to the user throught the command line
- Allows the user to pick the desired solution for harmonization and score completion

Pipeline



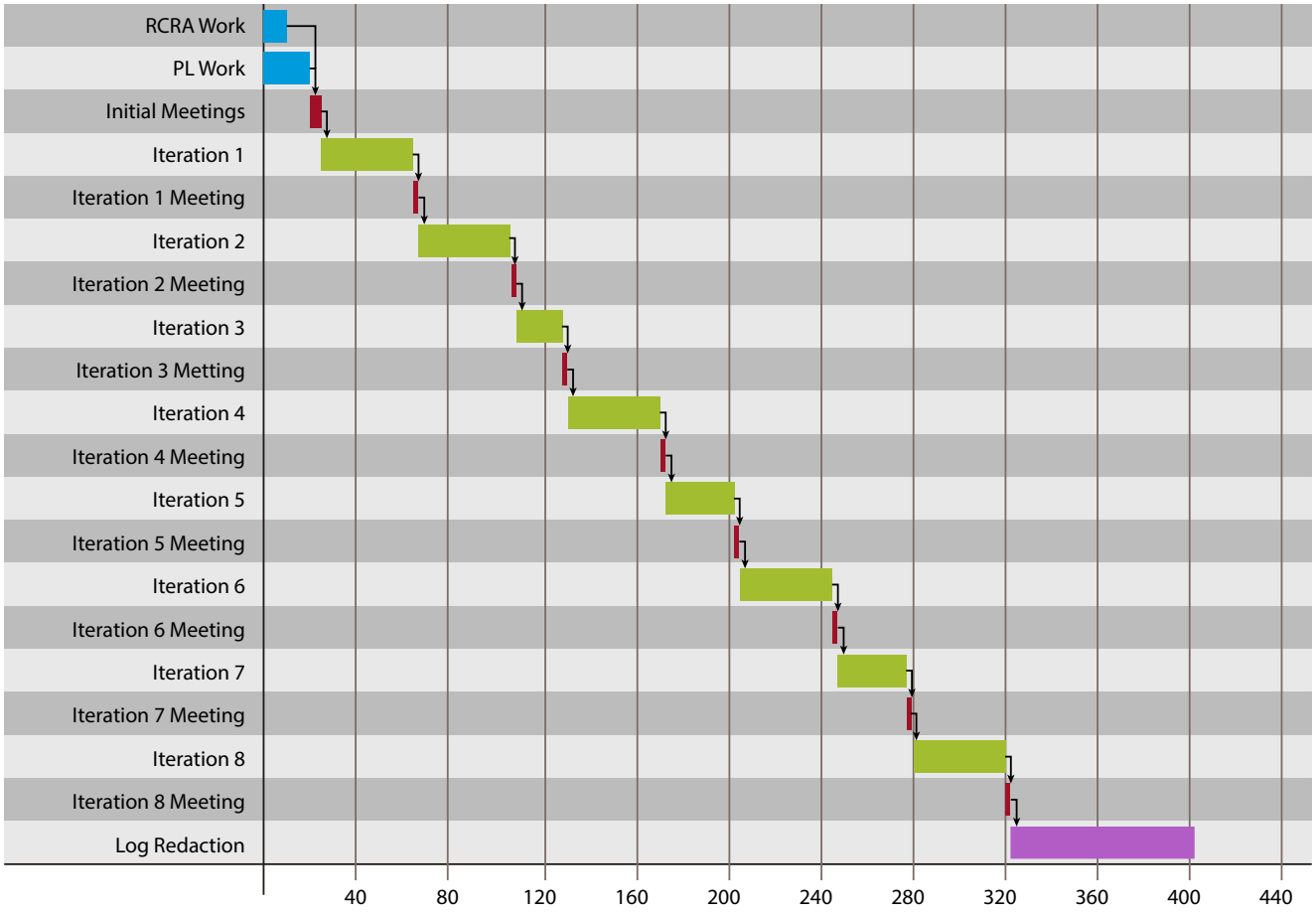
- Written in Python
- Coordinates the different modules secuentially
- Gives feedback to the user throught the command line
- Allows the user to pick the desired solution for harmonization and score completion
- Calls to the internal representation classes to store the results of the harmonization and completion as Python objects.

Overview

Custom development cycle

- Mix of Spiral and SCRUM
- Each iteration revises previous works and evolves current prototype
- Each iteration always has the same phases and these phases are planned beforehand
- The work planned for each iteration is directed by the objectives
- Short iterations (1-2 weeks)
- Allows objective redistribution for those that can't be achieved in one particular iteration
- Prototypes are revised with the Director after each iteration

Iteration Breakdown and Costs



Profile	**Cost/Hour**	**Hours**	**Total**
Student	5.5€	362	1991€
Director	9€	12	108€
Total			2099€

Overview

Conclusions

We developed a system that:

1. Harmonizes and annotates chords over almost any score
2. Given a certain harmonization, completes blank sections of the score
3. Adds brand new voices to the score

Strengths and Weaknesses

- Achieved maximum flexibility
- The tool produces correct scores in good times

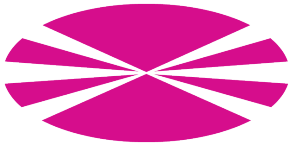
Strengths and Weaknesses

- Achieved maximum flexibility
- The tool produces correct scores in good times

- About 200 ASP lines that are hard to debug
- User still needs Computer Science and ASP knowledge to use it

Future Work

- Improve output and correct representation mistakes
- Implement a plugin interface for MuseScore 2 so the tool can be used through the editor itself
- Research about modulation and implement it in the tool
- Improve execution times for the inclusion of new voices
- Include rhythmic patterning in the new generated voices
- Ask for feedback from professional harmony teachers and polish the tool so it can be used in teaching



UNIVERSIDADE DA CORUÑA

Herramienta para armonización musical a través de Answer Set Programming

Rodrigo Martín Prieto

February 18, 2016

Director:
José Pedro Cabalar Fernández

Thank you!