DJazz's Architecture

The architecture of Djazz

Daniel Brown, IRCAM, Feb. 6, 2024

Djazz (by Marc Chemillier)

- Plays music in two ways:
  - selecting sections from scores
  - calculating improvisations using the factor oracle algorithm which is modified with pattern-matching methods

Architecture

- **several independent players which can be of different types (MIDI score reader, MIDI improviser, audio improviser…all the possible types to be determined)**

- **master control for synchronising timing and for broadcasting global commands**

goals for rewriting Djazz

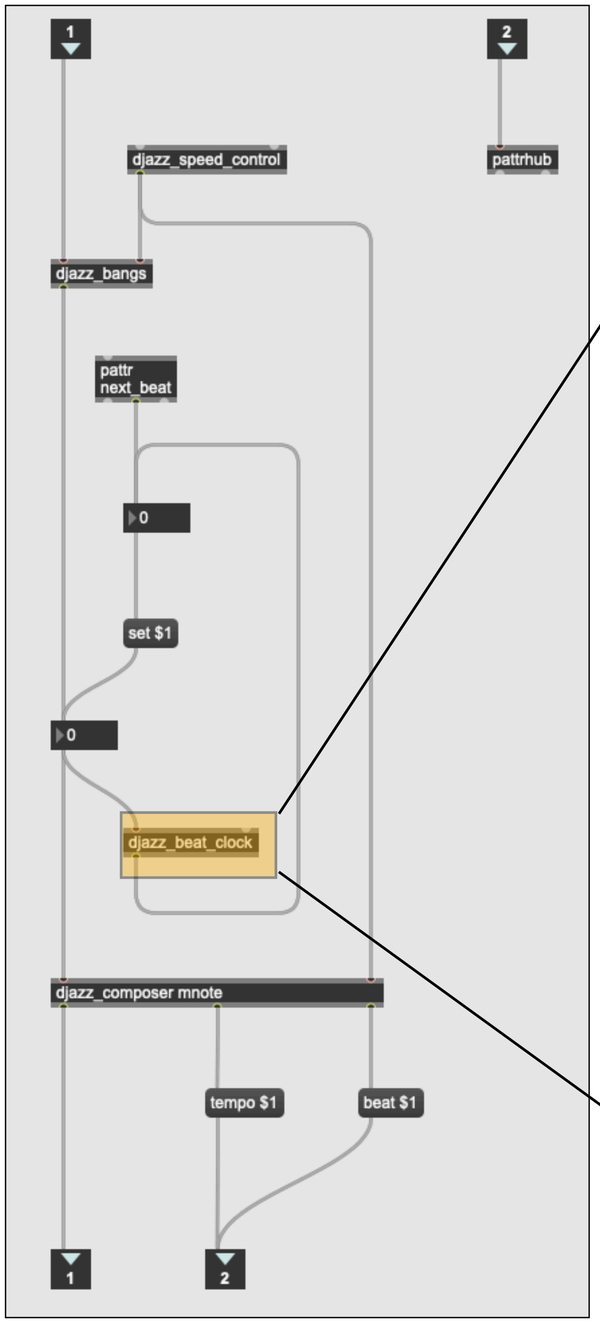
- prepare the software for distribution as a standalone
- design an architecture that is extensible:
  - Changes can be made in one area without creating bugs in other areas (dependence and modularity)
  - new functionality can be added without changing the existing code base
  - new functionality is easy to integrate. There are methods (not quite an SDK) for adding functionality. Some methods are more in-depth than others; some only require putting new max patches in properly named and organised folders.
- debugging is easy

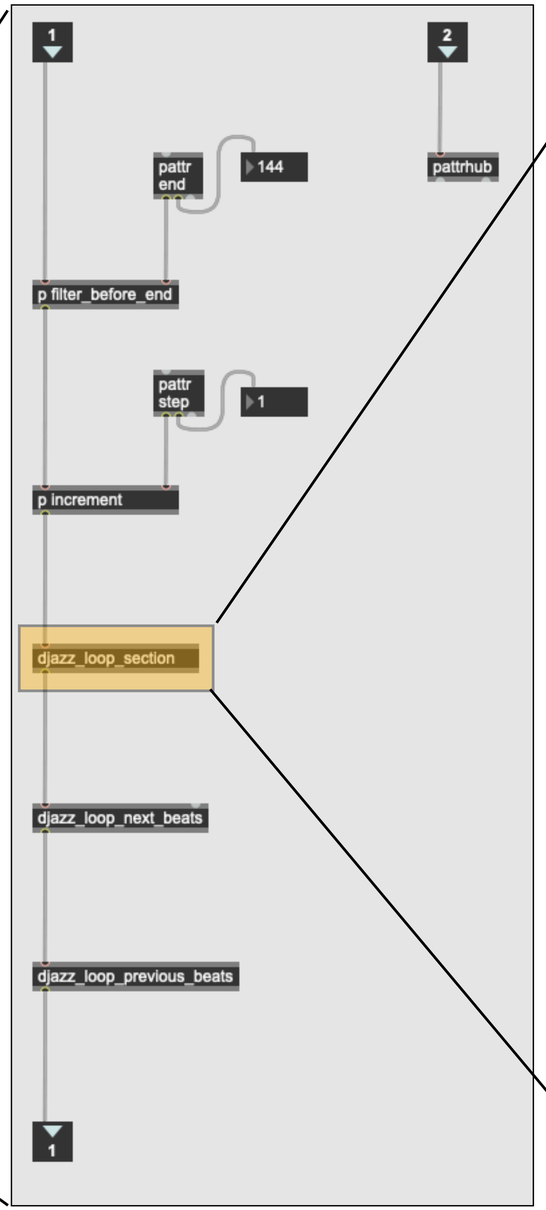
Djazz uses the Model-View-Control design pattern.

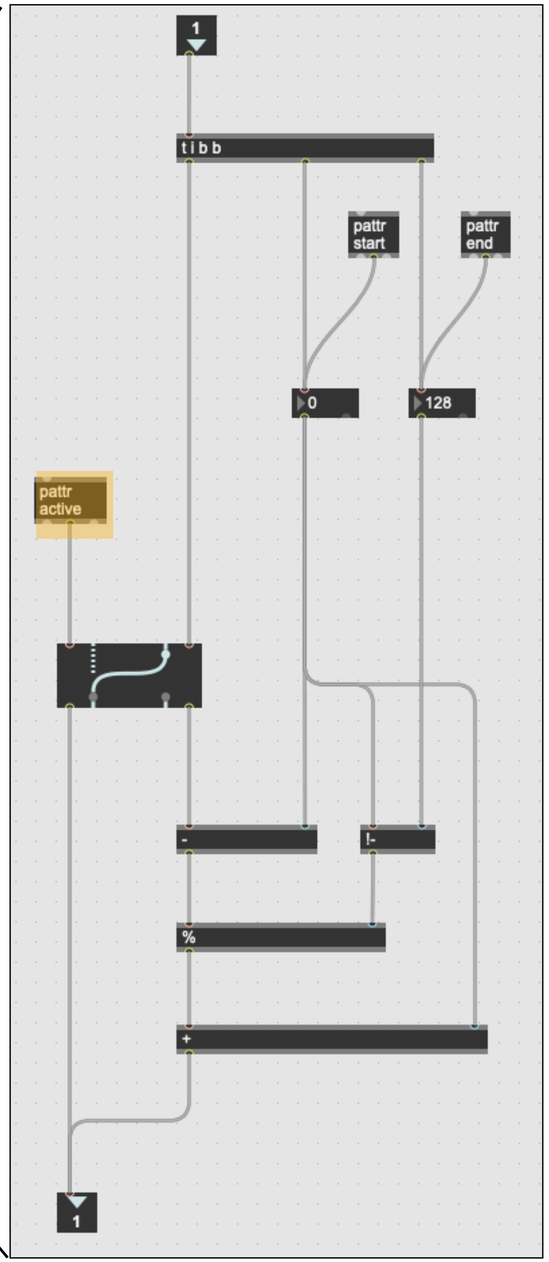
- The model consists of the objects that do the processing. At the top level, there are various players and the master control just described.
- The model is entirely controlled by passing named parameters (not Max “parameters”) in the message format <variable-name> <variable-value>.
- Every parameter is stored in a *pattr* object.

varname: midi_player

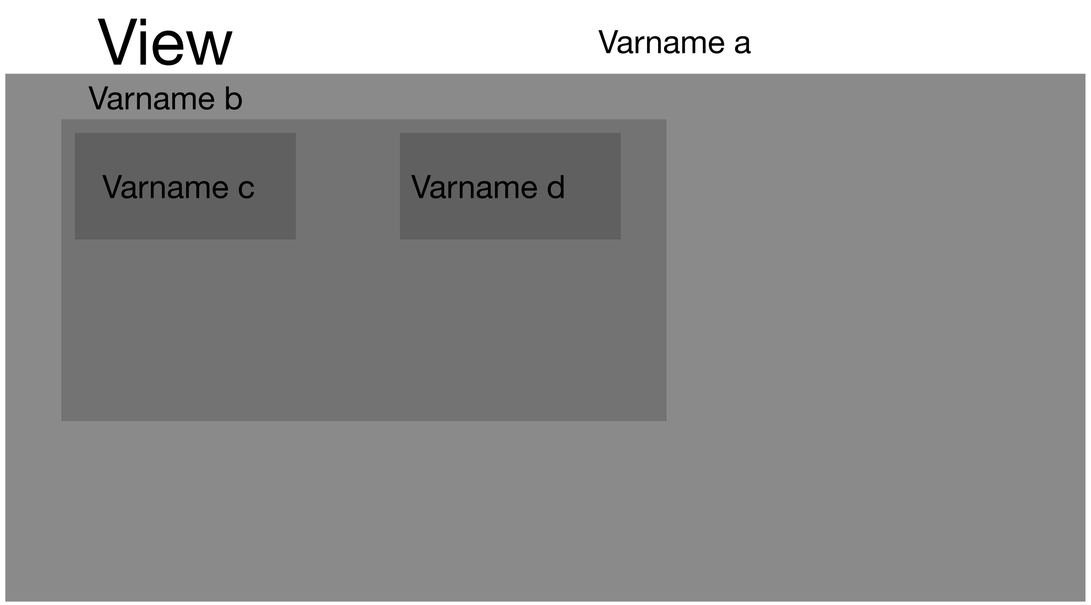
djazz_midi_generator



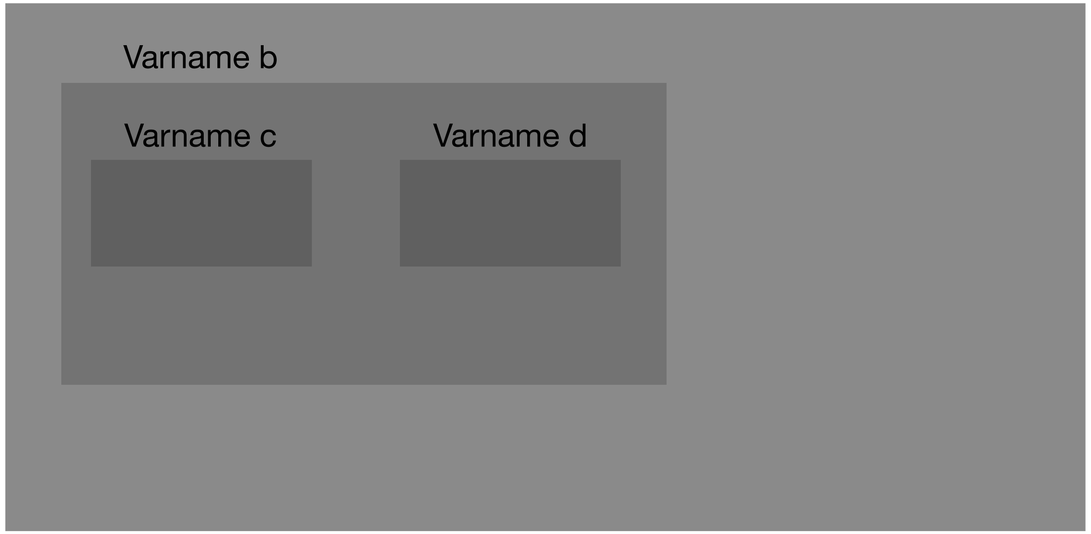




midi_player_1::beat_clock::loop_section::active



Pattrs a::b::c and a::b::d



Pattrs a::b::c and a::b::d

three problems with this method

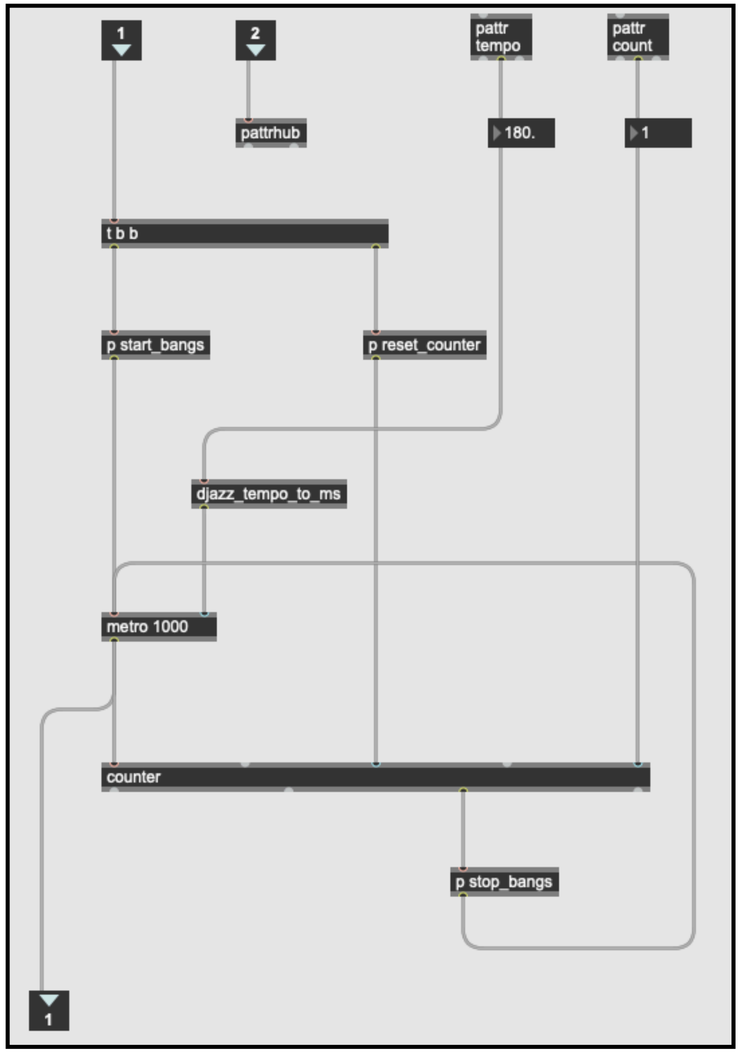
The separation of model and control is supposed to give you the freedom to design the control without worrying about how things are being processed inside the model. BUT…

- The control architecture is completely linked to the model architecture, which becomes very constraining, especially as the control logic in a complicated system probably does not reflect the way things are processed.

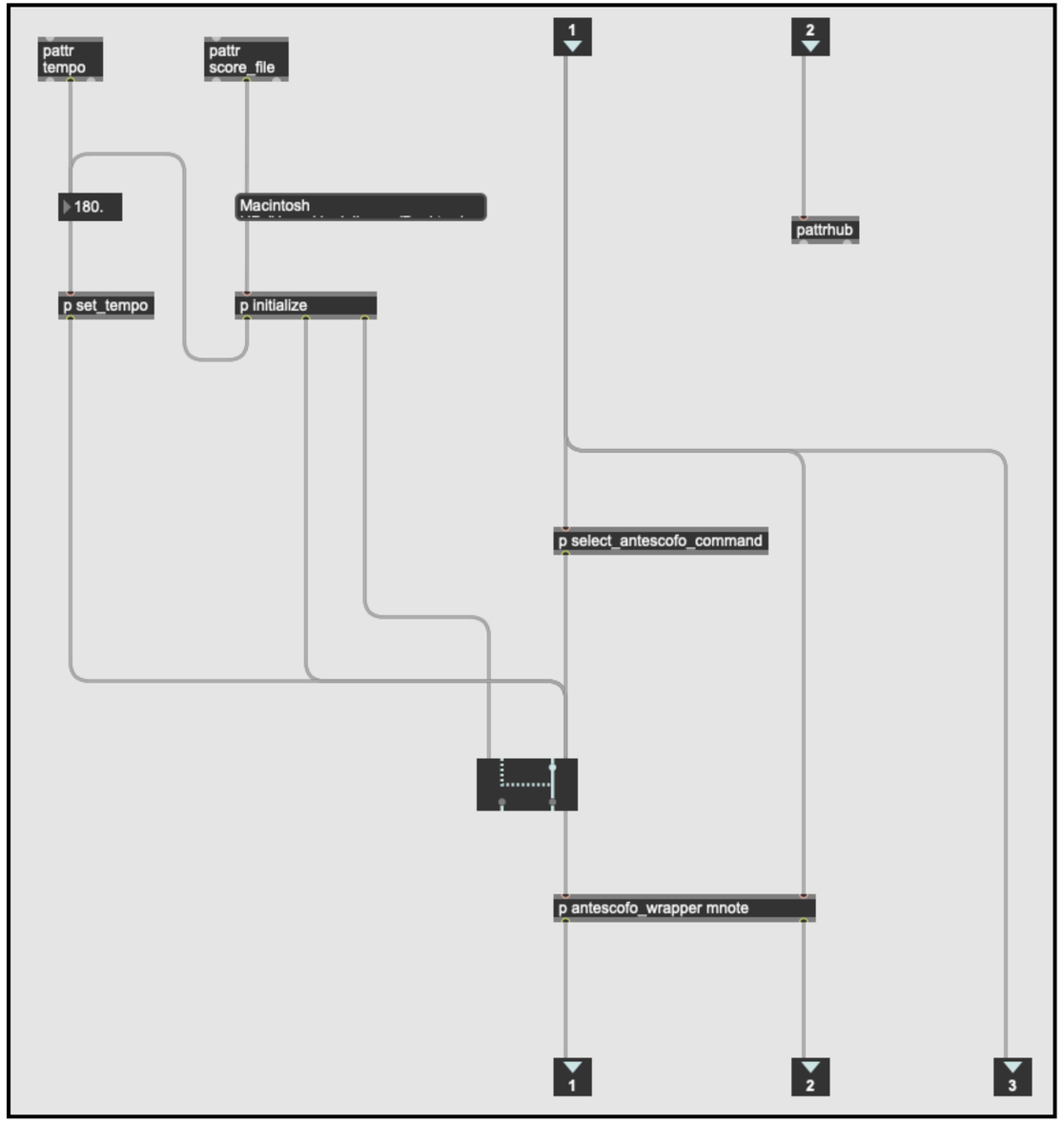
- You don’t necessarily want to control all the pattrs in the model. Some you might find unnecessary. This is not a big problem, but there is another related one:

- Some pattrs in the model are not actually independent of each other at runtime. Objects are designed independently, but their functions in the system are dependent on each other or other objects

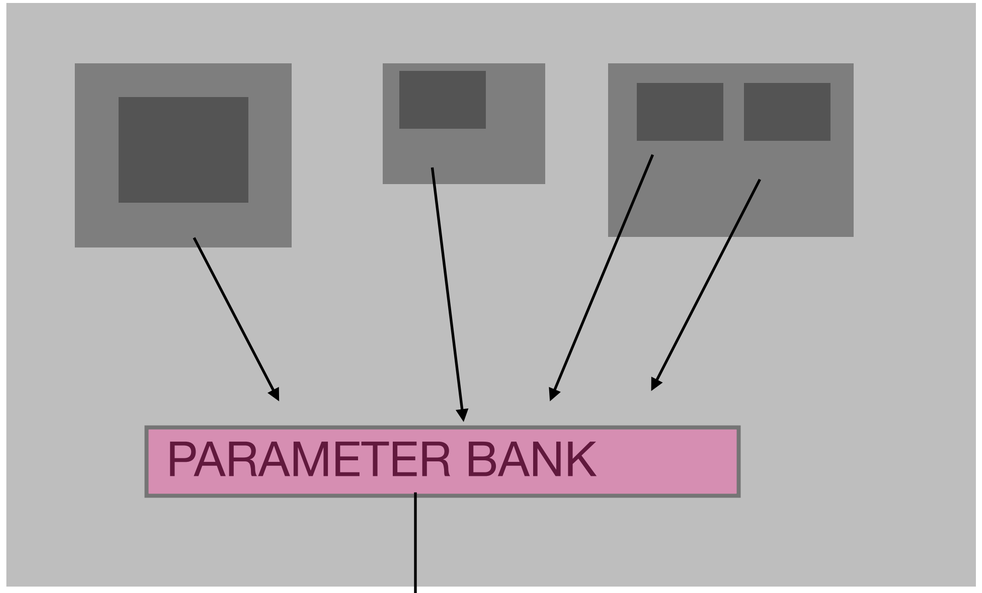
djazz_bangs



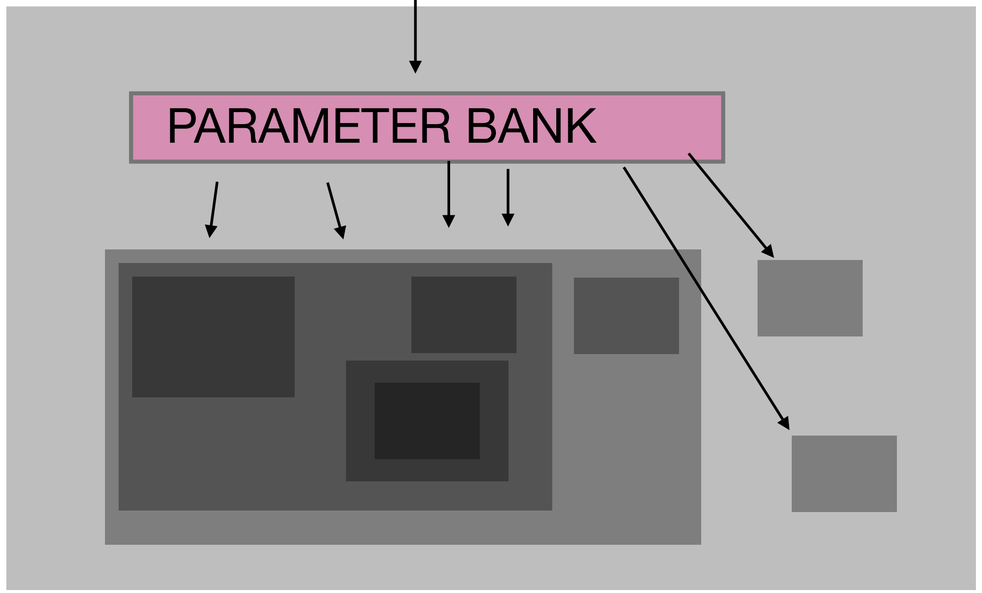
djazz_composer



CONTROL



MODEL



control (user)
parameters

surjective,
non-injective

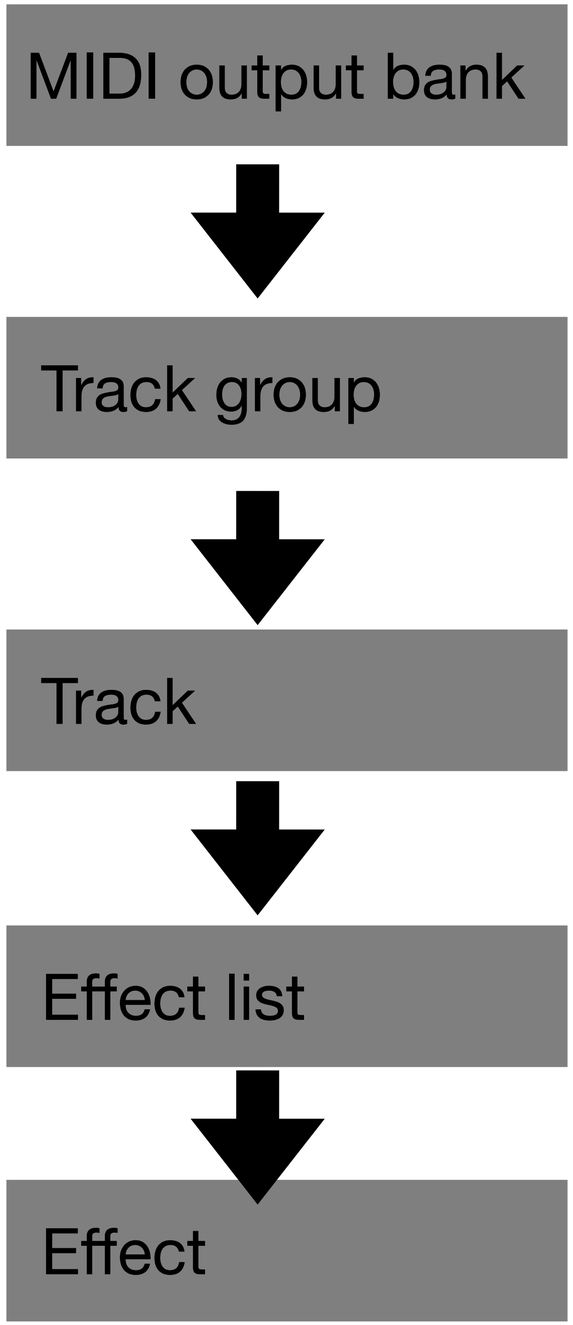
control bank
parameters

BIJECTIVE

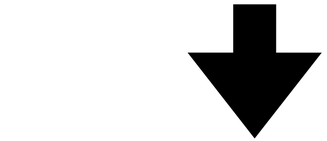
model bank
parameters

non-surjective,
injective

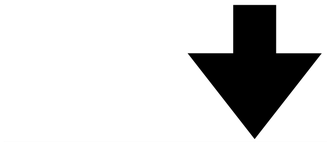
model interior
parameters


MIDI output bank
Track group
Track
Effect list
Effect

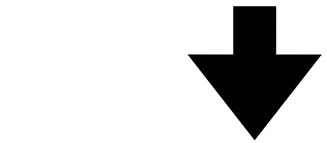
MIDI output bank



Track group

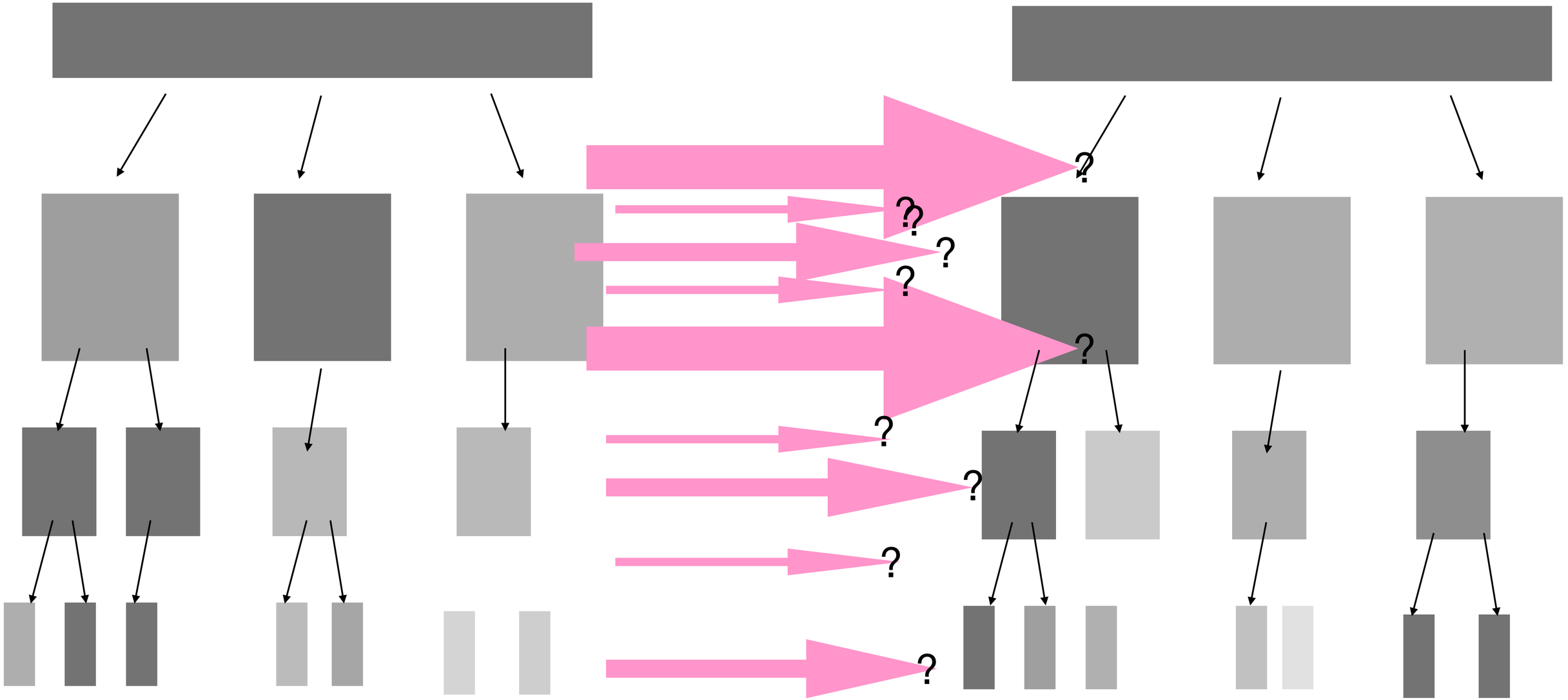


Track



Effect list

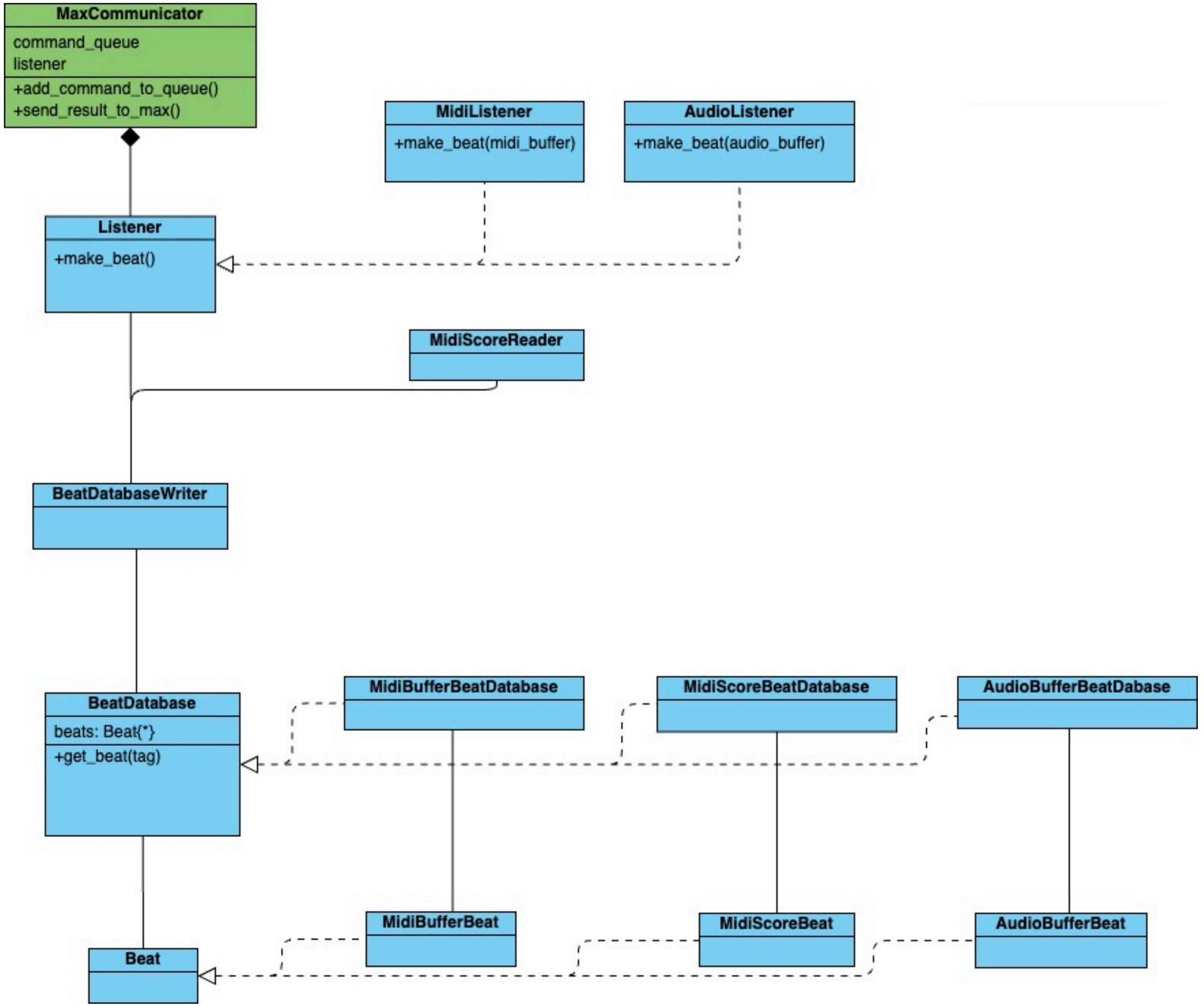
Effect



r(dict) —> program data

w(program data) —> dict

T(r, w): human-readable/buildable dict —> machine-readable dict


MaxCommunicator
command_queue
listener
+add_command_to_queue()
+send_result_to_max()
MidiListener
+make_beat(midi_buffer)
AudioListener
+make_beat(audio_buffer)
Listener
+make_beat()
MidiScoreReader
BeatDatabaseWriter
BeatDatabase
beats: Beat{*}
+get_beat(tag)
MidiBufferBeatDatabase
MidiScoreBeatDatabase
AudioBufferBeatDabase
MidiBufferBeat
MidiScoreBeat
AudioBufferBeat
Beat