Architecture logicielle du projet ossia – Annexes

Annexe A – définition d'un *widget* graphique

Le fichier correspondant dans le code source de *score* est `src/plugins/score-plugin-media/Media/Step/Inspector.hpp`

```
//Inspecteur pour un séquenceur pas-à-pas
class InspectorWidget final : public ...
{
public:
  InspectorWidget(
      const Model& model,
      const score::DocumentContext& document,
      ...)
  {
    // Spécification des propriétés d'un objet visuel (SpinBox)
    m_count.setRange(1, 24);

    // Lorsque le modèle change, la vue est mise à jour :
    connect(&model, &Model::stepCountChanged, this, [&] {
      m_count.setValue(obj.stepCount());
    });

    // Lorsque la vue est modifiée par l'utilisateur :
    connect(&m_count, &QSpinBox::editingFinished, this, [&]() {
      // une command d'undo-redo, SetStepCount, est créée
      // et appliquée au document
      CommandDispatcher<> dispatcher{document.commandStack};
      dispatcher.submit<SetStepCount>(model, m_count.value());
    });

    // Affichage de l'objet dans un formulaire
    lay->addRow("Count", &m count);
```

Annexe B – interface des commandes

Le fichier correspondant dans le code source de *score* est `src/lib/score/command/Command.hpp`

```
class Command
{
public
  // Les commandes s'exécutent dans le cadre ("context") d'un Document donné
  virtual void undo(const score::DocumentContext& ctx) const = 0;
  virtual void redo(const score::DocumentContext& ctx) const = 0;

  // Utilisé pour l'identification des commandes lors de leur déserialisation
  virtual const CommandGroupKey& parentKey() const noexcept = 0;
  virtual const CommandKey& key() const noexcept = 0;

protected:
  // Fonctions de sérialisation / désérialisation d'une commande
  virtual void serializeImpl(DataStreamInput&) const = 0;
```

Annexe C – définition d'un nœud type-safe

Le fichier correspondant dans le code source de *score* est `src/plugins/score-plugin-fx/Fx/Gain.hpp`

```
struct Node
{
  struct Metadata : Control::Meta_base
  {
    // Des méta-données supplémentaires (description, etc)
    // ont présentes dans le vrai code source

    // On liste ici les contrôles. Des éléments d'interface graphique
    // seront générés automatiquement.
    static const constexpr auto controls = std::make_tuple(
              Control::FloatSlider{"Gain", 0., 2., 1.}
    );

    // Liste des entrées et sorties audio. Des E/S MIDI, ou de messages,
    // sont aussi possibles
    static const constexpr audio_in audio_ins[]{"in"};
    static const constexpr audio_out audio_outs[]{"out"};
  };

  using control_policy = ossia::safe_nodes::last_tick;
  // Le compilateur s'assure que les paramètres de la méthode "run"
  // correspondent aux types définis dans la structure Metadata -
  // Les entrées doivent être listées en premier, puis les contrôles, puis les sorties.
  static void run(const ossia::audio_port& p1,
      float g,
      ossia::audio_port& p2,
      ossia::token_request t,
      ossia::exec_state_facade st)
  {
```