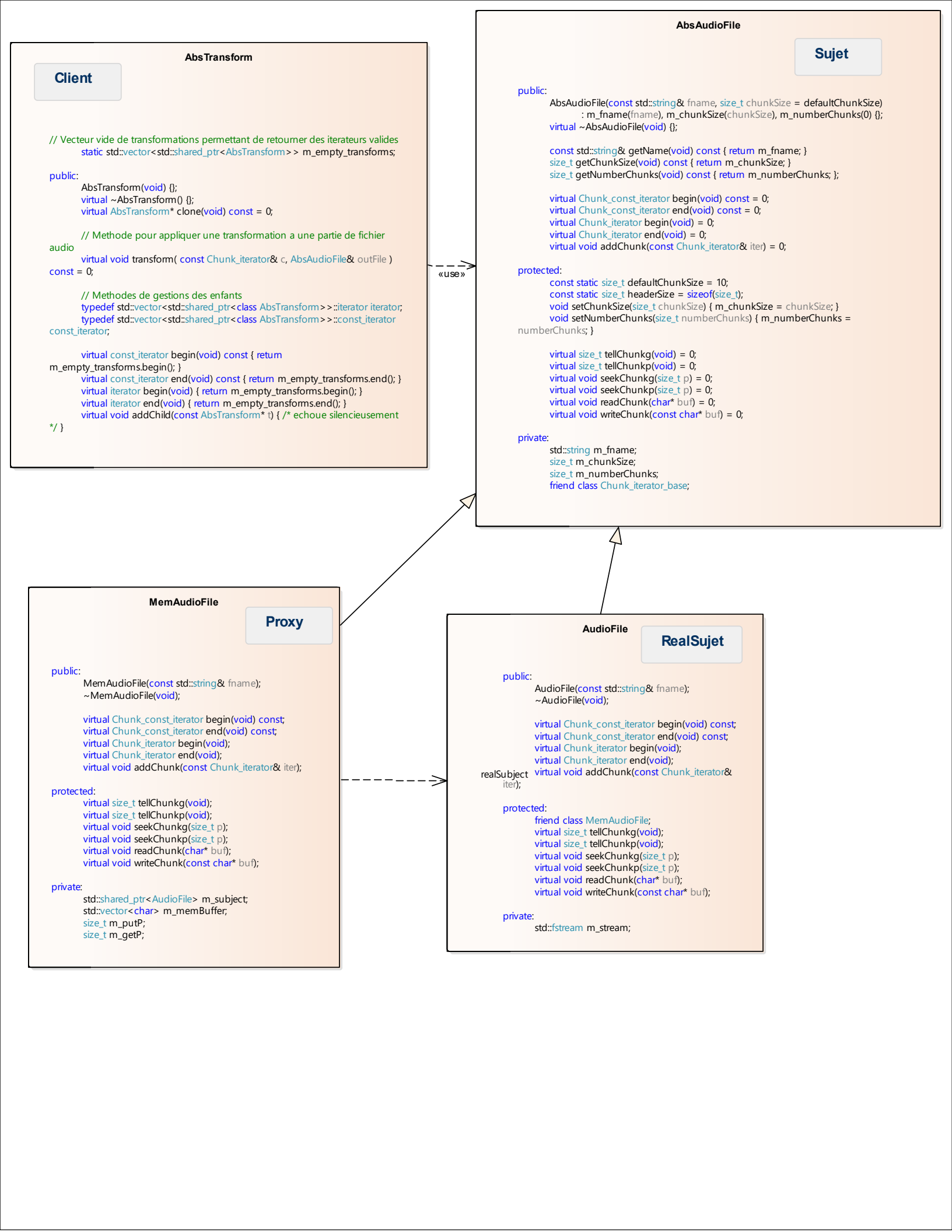
AbsTransform
Client
// Vecteur vide de transformations permettant de retourner des iterateurs valides
static std::vector<std::shared_ptr<AbsTransform>> m_empty_transforms;
public:
AbsTransform(void) {};
virtual ~AbsTransform() {};
virtual AbsTransform* clone(void) const = 0;
// Methode pour appliquer une transformation a une partie de fichier audio
virtual void transform(const Chunk_iterator& c, AbsAudioFile& outFile) const = 0;
// Methodes de gestions des enfants
typedef std::vector<std::shared_ptr<class AbsTransform>>::iterator iterator;
typedef std::vector<std::shared_ptr<class AbsTransform>>::const_iterator const_iterator;
virtual const_iterator begin(void) const { return m_empty_transforms.begin(); }
virtual const_iterator end(void) const { return m_empty_transforms.end(); }
virtual iterator begin(void) { return m_empty_transforms.begin(); }
virtual iterator end(void) { return m_empty_transforms.end(); }
virtual void addChild(const AbsTransform* t) { /* echoue silencieusement */ }
«use»
AbsAudioFile
Sujet
public:
AbsAudioFile(const std::string& fname, size_t chunkSize = defaultChunkSize)
: m_fname(fname), m_chunkSize(chunkSize), m_numberChunks(0) {};
virtual ~AbsAudioFile(void) {};
const std::string& getName(void) const { return m_fname; }
size_t getChunkSize(void) const { return m_chunkSize; }
size_t getNumberChunks(void) const { return m_numberChunks; };
virtual Chunk_const_iterator begin(void) const = 0;
virtual Chunk_const_iterator end(void) const = 0;
virtual Chunk_iterator begin(void) = 0;
virtual Chunk_iterator end(void) = 0;
virtual void addChunk(const Chunk_iterator& iter) = 0;
protected:
const static size_t defaultChunkSize = 10;
const static size_t headerSize = sizeof(size_t);
void setChunkSize(size_t chunkSize) { m_chunkSize = chunkSize; }
void setNumberChunks(size_t numberChunks) { m_numberChunks = numberChunks; }
virtual size_t tellChunkg(void) = 0;
virtual size_t tellChunkp(void) = 0;
virtual void seekChunkg(size_t p) = 0;
virtual void seekChunkp(size_t p) = 0;
virtual void readChunk(char* buf) = 0;
virtual void writeChunk(const char* buf) = 0;
private:
std::string m_fname;
size_t m_chunkSize;
size_t m_numberChunks;
friend class Chunk_iterator_base;
MemAudioFile
Proxy
public:
MemAudioFile(const std::string& fname);
~MemAudioFile(void);
virtual Chunk_const_iterator begin(void) const;
virtual Chunk_const_iterator end(void) const;
virtual Chunk_iterator begin(void);
virtual Chunk_iterator end(void);
virtual void addChunk(const Chunk_iterator& iter);
protected:
virtual size_t tellChunkg(void);
virtual size_t tellChunkp(void);
virtual void seekChunkg(size_t p);
virtual void seekChunkp(size_t p);
virtual void readChunk(char* buf);
virtual void writeChunk(const char* buf);
private:
std::shared_ptr<AudioFile> m_subject;
std::vector<char> m_memBuffer;
size_t m_putP;
size_t m_getP;
AudioFile
RealSujet
public:
AudioFile(const std::string& fname);
~AudioFile(void);
virtual Chunk_const_iterator begin(void) const;
virtual Chunk_const_iterator end(void) const;
virtual Chunk_iterator begin(void);
virtual Chunk_iterator end(void);
realSubject
virtual void addChunk(const Chunk_iterator& iter);
protected:
friend class MemAudioFile;
virtual size_t tellChunkg(void);
virtual size_t tellChunkp(void);
virtual void seekChunkg(size_t p);
virtual void seekChunkp(size_t p);
virtual void readChunk(char* buf);
virtual void writeChunk(const char* buf);
private:
std::fstream m_stream;