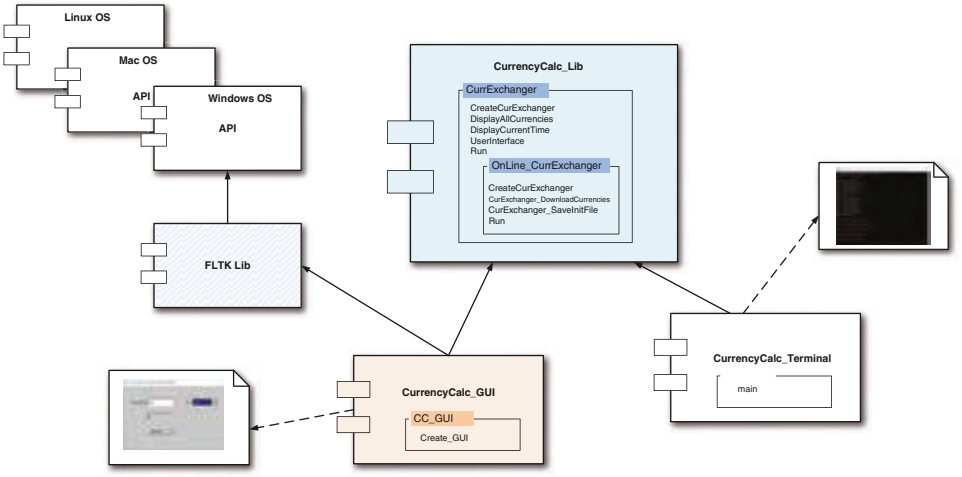
## 6.2.8. Interfejs użytkownika

W poprzednich punktach zobaczyliśmy ewolucję projektu `CurrencyExchange`: jego dwie wersje, obie ujęte w hierarchicznych przestrzeniach nazw `CurrExchanger` i zagnieżdżonej w niej przestrzeni `OnLine_CurrExchanger`. `CurrExchanger` definiuje bardzo podstawową funkcjonalność, podczas gdy `OnLine_CurrExchanger` rozszerza ją pod kątem automatycznego wczytywania informacji walutowych z internetu. Obie zostały przetestowane jako samodzielne aplikacje konsolowe. Zobaczyliśmy również, w jaki sposób możemy podzielić projekt na bibliotekę statyczną o nazwie *CurrencyCalc_Lib* oraz inny komponent, *CurrencyCalc_Terminal*, który buduje aplikację konsolową (punkt 6.2.6). Czas teraz pójść o krok dalej i dodać do tego projektu graficzny interfejs użytkownika (ang. *graphical user interface*, GUI). Z tego powodu utworzymy trzeci komponent oprogramowania do uruchamiania *CurrencyExchanger* jako aplikacji GUI: `CurrencyCalc_GUI`. Diagram komponentów UML na rysunku 6.8 zmieni się teraz na diagram z rysunku 6.9.




**Rysunek 6.9.** Diagram komponentów UML dla różnych komponentów oprogramowania używanych w naszym projekcie *CurrencyCalc*. Ponieważ silnik, tj. główny moduł, wymiany walut został wyodrębniony do osobnej biblioteki, możemy w prosty sposób skonstruować aplikację konsolową lub GUI bez żadnych zakłóceń

W tym punkcie interesuje nas głównie komponent `CurrencyCalc_GUI`, który będzie w stanie rysować graficzne widżety, a także przetwarzać zdarzenia systemowe, co za chwilę zobaczymy. Jednak komponenty GUI są silnie zależne od rodzaju systemu operacyjnego, z którym pracujemy. Z jednej strony oznacza to, że gdybyśmy próbowali napisać nasz komponent `CurrencyCalc_GUI` dla dwóch lub trzech różnych systemów operacyjnych, musielibyśmy napisać dwie lub trzy dosyć różne wersje tego komponentu. Z drugiej strony, nawet jeśli nasze zadanie byłoby dużo prostsze i celem byłby wyłącznie jeden system operacyjny, korzystanie z surowego API nie byłoby najbardziej produktywną opcją. Znacznie lepszym rozwiązaniem jest skorzystanie z bibliotek GUI tworzących rodzaj interfejsu między

aplikacją użytkownika, taką jak *CurrencyCalc*, i jednym lub więcej systemami operacyjnymi. Dodatek A.4 zawiera krótkie omówienie dostępnych bibliotek GUI dla języka C++. Niestety każda róża ma swój cierń, więc każda z nich ma swoje własne dziwactwa i ograniczenia. W naszym przykładzie zdecydowaliśmy się przetestować bibliotekę FLTK, głównie dlatego, że jest darmowa, napisana w całości w C++ i działa na wielu systemach.

Spójrzmy więc na diagram komponentów UML na rysunku 6.9. Najbardziej istotną zmianą jest to, że oddzieliliśmy wszystkie komponenty ściśle związane z wymianą walut od operacji wejścia/wyjścia powiązanych z użytkownikiem. Komponenty te są teraz ujęte w osobnych bibliotekach, podczas gdy akcje użytkownika przeniesione są do różnych projektów aplikacji. W rezultacie uzyskujemy następujące istotne cele:

- *Czytelność kodu* – jest teraz znacznie łatwiej skoncentrować się na różnych poziomach naszego systemu. Inaczej mówiąc, podczas pracy nad dalszymi mechanizmami dla wymiany walut będziemy pracować wyłącznie z biblioteką *CurrencyCalc_Lib*. Podczas pracy nad interfejsami użytkownika nie będziemy musieli zmieniać niczego w bibliotece wymiany walut.
- *Ponowne wykorzystanie kodu* – znacznie łatwiej jest teraz o ponowne wykorzystanie naszej biblioteki *CurrencyCalc_Lib* w dowolnym innym projekcie (z interfejsem lub bez interfejsu użytkownika), który wymaga akcji wymiany walut. Również znacznie prościej będzie można teraz rozszerzyć funkcjonalność tej biblioteki.

Dlatego zgodnie z regułą nie powinniśmy używać w naszych klasach obiektów `cout` i `cin`. Operacje wejścia/wyjścia powinny być implementowane za pomocą zewnętrznych przeciążonych `operator <<` i `operator >>`.

Nasz kod będzie wyświetlał grafikę. Aby tak się stało, musimy zainstalować bibliotekę FLTK. Szczegóły tej operacji opisane są na stronie projektu FLTK (FLTK 2019) i zależą od naszego systemu operacyjnego oraz zainstalowanych narzędzi. Jednak pomocny powinien być poniższy przewodnik:

1. Pobierz FLTK ze strony internetowej tego projektu i wypakuj zawartość w taki sposób, aby katalogi *FLTK* i *CurrencyCalc* znajdowały się na tym samym poziomie.
2. Przejdź do katalogu *FLTK* i uruchom `cmake .` [kropka] w celu zbudowania projektu FLTK w katalogu bieżącym.
3. Otwórz projekt i zbuduj bibliotekę. Po ukończeniu katalog *FLTK/Lib/Debug* powinien zawierać bibliotekę *fltkd.lib* (zweryfikuj jej datę i czas budowy).

### 6.2.8.1. Definicja klasy `CC_GUI`

Listing 6.8 pokazuje definicję klasy `CC_GUI`, które definiuje interfejs użytkownika dla naszej aplikacji *CurrencyCalc*, zaś listing 6.9 zawiera jej pełną implementację. Omówmy pokrótce sposób ich działania.

Jak zwykle na początku nowej definicji powinniśmy poinformować kompilator o tym, z których klas zamierzamy korzystać. W tym przykładzie dokonujemy tego na dwa sposoby. Najpierw w liniach [1-5] dołączane są pliki nagłówkowe, zawierające definicje pewnych kontenerów biblioteki standardowej, a także definicji klasy `XML_CurrencyExchanger`