Multilingual Logging Library

Scenario

You have been given the job of creating a logging library for a mobile application framework being developed at your company. At the moment, the framework is likely to be used by both English and French developers, but support will likely be added in the future for other languages (i.e., when the company can afford to hire a fluent speaker of the language). In addition, the logger should be able to be set to one of two configurations - currently, logging with date and logging without date.

Problem

Clients of the logger must always interact with an instance of the *Logger* class using the *logMessage* method. *logMessage* accepts two parameters: a String containing the actual message, and an int containing the log level (0 = WARNING / ALERTE, 1 = CRITICAL / ETAT CRITIQUE). For the moment, having a date-logging version and non-date-logging version of your logger as subclasses of *Logger* is fine, as this aspect will likely not change, but adding a new language will be very likely.

Logged messages should appear as follows: [DATE (optional)] [ALERT LEVEL] [Message]. For the English logger, dates should be of the format MM/DD/YYYY, for the French logger, dates should be of the format DD.MM.YYYY. Specific examples are:

- [WARNING] Variable is never used (French Level 0 without date)
- [01/15/2014][CRITICAL] Security violation (English Level 1 with date)
- [15.01.2014][ALERTE] Variable is never used (French Level 0 with date)
- [ETAT CRITIQUE] Security violation (French Level 1 without date)

An example client is provided. This client includes a mini-factory, which you will need to complete to generate your various different *Logger* configurations. The actually logging can simply be printing to the console (i.e., cout or System.out.println).

Deliverables

1. Identify the design pattern you used to solve this problem, and the participants (i.e., the roles each class takes).
2. An implementation in a language of your choice. Your implemented classes should be easily extended to include other languages.

3. A class diagram of your solution (including existing classes), so future developers can easily see how to work with your solution.