In this example, the `set_error_handler` method redirects all errors (that can be redirected) to the method `errorHandler`. When the method `trigger_error` causes `E_USER_ERROR` to occur, the handling of the error is redirected to the `errorHandler` method. This method then gathers the information from the error to throw an exception (`errorException`). The exception is captured by the `catch(errorException $e)` method, which causes the message `"User Error"` to be displayed.

In PHP 7, the Error object captures potential system errors as exceptions.

```
try {
call_method(null); // no such method!
} catch (Error $e)
{ echo $e->getMessage; }
```

Previously, the call to a non-existent function would cause a fatal error. Using the `EngineException` object allows the programmer to handle the error. The example shown previously (before PHP 7) would only capture some errors; this new technique is designed to allow the programmer much more control over errors. If this `catch` block is not included, any "error" would cause the program to crash with the `"Fatal error: Uncaught exception"` message.

If you have PHP 7 installed, use the new `Error` object along with the `Exception` object to avoid fatal errors whenever possible.

> *Security and performance—Usually the use of throwing and catching exceptions can reduce the amount of code needed in a program. However, there is a trade-off. Several studies of different object-oriented program languages have concluded that exception handling is less efficient (performance) than using developer created routines. The developer should use exceptions as true "exceptions" to the normal flow of the application. For more frequently occurring situations, the developer should create situation handling routines in the application.*

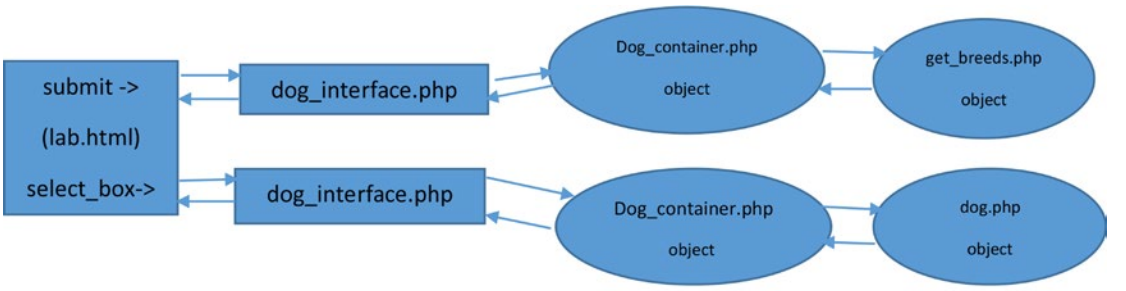

Figure 5-1. *Data flow for the dog application*

In the Dog application, the information flows between many different programs. Each of these programs must be able to handle exceptions properly. However, message handling should all occur in the interface. Any objects that are part of the business rules tier (`dog_container`, `dog`, and `get_breeds`) should pass any exception messages to the interface to handle. At first this may sound like a complex and confusing task. However, the hierarchy of exception handling will greatly simplify this task. As you are about to see, using exception handling will reduce the amount of code necessary.

When exceptions are thrown, the environment will look in the program (or class) itself to determine if there is a `catch` block that can handle the exception. If there is not a `catch` block, it will go up one level in the hierarchy and check any calling program (or program that has made an instance of the class) for a `catch`