Architecture Documentation

IbDMF Distributed Multiplatform Framework

created by

Lothar Behrens

Template Revision: 6.1 EN
June 2012

Revision History

Version	Date	Reviser	Description
0.1	30.08.2013		Initial
0.2	31.08.2013		Added coarse architecture diagram

Related documents

Document	Description

Table of Contents

Remark: The Microsoft-Word™ variant of this template contains hidden remarks and suggestions. You can toggle display of this text by the appropriate Word-command.

1. Introduction and Goals

Purpose of the System

Software requires a framework to be written with. My first software projects were written with frameworks that are ready made for this purpose. I have migrated my early projects from MFC, a Microsoft Framework to one provided with Power++ by Sybase. But very early in my carrier I were set up with those frameworks from vendors. The cause was simply the end of life for the last framework. Sybase decided to kill Power++.

My first goal therefore is to provide my own framework that does not end. The next goal is to learn how to write a framework and what complications it brings in.

Over time there are added more goals that are listed here:

- The framework should run not only on one platform, such as Windows that I started with.
- It should be modular like COM that is inspiring me.
- No direct dependency to any GUI framework. A GUI can be encapsulated behind a basic API.
- There are more goals, but these are issues within an application that is using this framework. These are things for fast database application prototyping and required techniques. This project contains the application to aid in rapid database prototyping, but it is a candidate for splitting the project.

1.1 Requirements Overview

The most important functional requirements:

- The framework should provide it's functionality by interfaces.
- The implementation language is C++, thus interfaces are pure abstract classes. (Optionally this may change to structs if that really helps to overcome the multiple compiler issue)
- Each implementation of an interface provides a functor (a constructor as function).
- A functor is a helper function that must be exported by the library (DLL, so) and it returns an instance of the implementation. The returned interface is a base interface, all implementations have to implement.
- The base interface provides querying for other provided interfaces of that implementation. This is inspired by COM.
- A macro provides functionality to be used to implement the basic functionality of the base interface. Each implementation has to use this macro.
- An exported implementation should be registered for the ability to be found and used.
- Not registered implementations are for internal usage of DLL's or shared objects.
- A main interface plays the role of a virtual application. It is the interface between a real implementation provided by an API. The real implementation is provided without using an API. Instead a generic marshaling is used.
- The main interface must provide a functionality to load a custom application module. This module implements the application logic that cannot be in the framework.
- The implementing GUI or UI instantiates the virtual application and may load an application on behalf of user login.
- The GUI must provide a basic functionality by predefined marshal interfaces.
- Marshal interfaces can be intercepted, conventional API only by custom interception wrapper that implement the same interface and therefore must be interface APIs.
- The virtual application can automatically do this for the last user and application or by an environment variable pointing to the application module (TARGET_APPLICATION).
- An application module must provide a functor with a name of instanceOfApplication (case sensitive).

1.2 Quality Goals

The quality level for the framework is for daily usage, but not for usage in any backend, server or services based software. This is due to the learning goal to write a framework. If a software is using the framework in such environments, care should be taken to mitigate these quality issues by fault tolerant solutions, restart, recovery and the like mechanisms.

Daily usage means no crashes per day. Occasionally the software can crash, thus the software should not be used within critical environments. Occasionally crashes are tolerated in prototyping applications.

The quality level should increase were needed by the open source community.

1.3 Stakeholders

Me, Lothar Behrens, like to have a tool for software development that decreases time to market.

Open Source Community by participation.

2. Architecture Constraints

2.1 Technical Constraints

Hardware-Constraints	
	Common PC Hardware
Software-Constraints	
	Sqlite Databases must be supported, ODBC may be supported
	wxWidgets must be supported as GUI frontend
Operating System Constraints	
	Windows, Linux and Mac OS X must be supported, Solaris may
Programming Constraints	
	Implementing language is C++

2.2 Organizational Constraints

Organization and Structure	
	Private Open Source project
Resources (Budget, Time, Personnel)	
	One person project, no time constraints, no budget
Organizational Standards	
	Most current language features should be omitted due to multiplatform availability
Legal Factors	
	No legal liabilities are accepted from users due to Open Source

2.3 Conventions

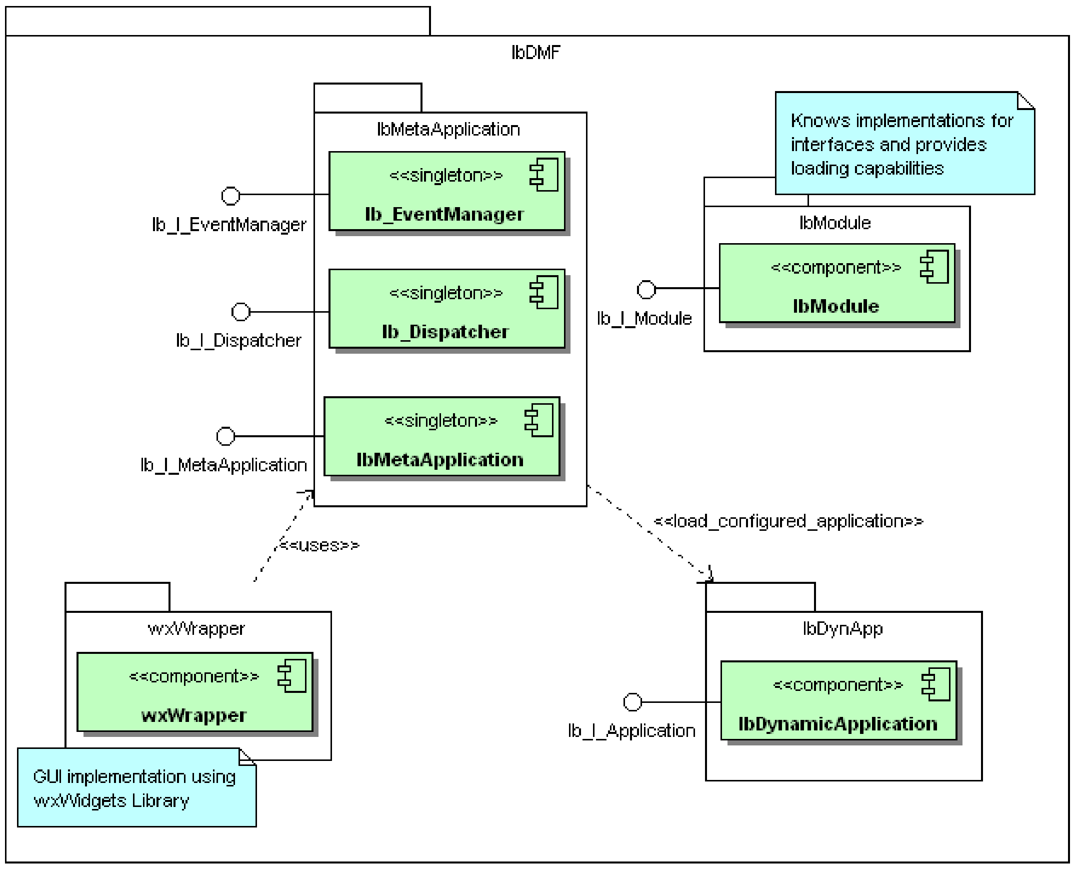
All interfaces are prefixed with lb_I_* to clearly indicate it's origin. All application modules implement a functor with this name: instanceOfApplication. All implementations have to implement lb_I_Unknown. It should be avoided to couple the virtual application against the GUI implementation by using interfaces. Instead, marshal interfaces should be preferred. There are interfaces used that should be deprecated in favor of marshal interfaces. They could be intercepted much easier.

Many of the functionalities in lb_I_MetaApplication should be moved to lb_I_GUI. But still the implementation of lb_I_GUI should use marshal interfaces to decouple from the implementation. That is for sample the function to open a database form for SQL queries or custom forms. This was a very old initial design.

3. System Scope and Context

The framework view separates itself from the application logic. The application communicates with interfaces and thus never with implementations directly. This goes down to the very basic interfaces like integer and other primitives. An application developer may decide not to use these basic primitives internally, but must do so for interaction with the framework.

3.1 Business Context



The main application (lbDynApp) in this case provides the logic to enable database application prototyping (not described in detail here). The lbMetaApplication used by the wxWrapper component is a class that loads the configured application. The wxWrapper provides event handlers that are minimally required by lbMetaApplication to load menu, toolbars and other stuff. The GUI is provided by the wxWrapper in form of an interface and

[1] We often tend to a pragmatic approach – but here we insist on a list of all (a-l-l) neighboring systems. Too many projects have failed because they were not aware of their neighbors. ☹

the instance is passed to lbMetaApplication. This instance is in turn passed to the loaded application to allow direct usage of basic functionality provided by the GUI. As noted in the conventions, this should be changed.

The dispatcher forwards calls to be dispatched to their event handlers that are registered at runtime from wxWrapper, lbMetaApplication and lbDynamicApplication. The event manager is used to map event names to ID's and reverse ID's to names. The overall event handling mechanism is the key to loose coupling at runtime. This allows loading application modules that do not require to define ID ranges for events. The lbModule provides a repository of interfaces and their implementations and is used to instantiate these.

3.2 Technical Context

TBD

3.3 External Interfaces

Interface Id

Name	<name of Interface>
Version	
Changes w.r.t previous release	
Who changed it and why?	
Contact person	

Business Context of the Interface

Business Processes

<Diagram or desciption of business processes relevant for this interface>

Interface Data

<Description of interface data>

Technical Context

Form of interaction