Architecture Documentation

lbDMF Distributed Multiplatform Framework

created by

Lothar Behrens

Template Revision: 6.1 EN
June 2012

We acknowledge that this document uses material from the arc 42 architecture template, http://www.arc42.de. Created by Dr. Peter Hruschka & Dr. Gernot Starke.

Revision History

Version	Date	Reviser	Description
0.1	30.08.2013		Initial

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.

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.

TODO: Show UML package diagram with virtual app used by GUI implementation loading an application module.

3.1 Business Context

3.2 Technical Context

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

Requirements for the Interface

Security Aspects

Quantities

Runtime

[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. ☹

Throughput/Volume

Availability

Logging

Archiving

Participating Resources

Syntax: Data and Formats

Data Formats

Validity & Plausibility Rules

Encoding, Character Sets

Configuration data

Syntax: Methods/Functions

Check data

Interface Process

Logical and technical processes

Semantics

Side effects, consequences

Technical Infrastructure

Technical protocols

Error and Exception Handling

Constraints and Assumptions

Access Rights

Temporal constraints

Parallel Access

Preconditions for using the interface

Operating the Interface

Meta Information for the Interface

Person in charge

Costs of using the interface

Organizational Issues

Versioning

Examples of Using the Interface

Sample data

Sample flows and interactions

Programming Examples

3.3.1 External Interface 1

<insert interface template>

3.3.2 External Interface 2

<insert interface template>

3.3.3 ...

3.3.4 External Interface n

<insert interface template>

4. Solution Strategy

5. Building Block View

5.1 Level 1

The following diagram shows the main building blocks of the system and their interdependencies:

<insert overview diagram here>

Comments regarding structure and interdependencies at Level 1:

5.1.1 Building Block Name 1 (Black Box Description)

<insert the building block's black box template here>

5.1.2 Building Block Name 2 (Black Box Description)

<insert the building block's black box template here>

5.1.3 ...

<insert the building block's black box template here>

5.1.4 Building Block Name n (Black Box Description)

<insert the building block's black box template here>

5.1.5 Open Issues

5.2 Level 2

5.2.1 Building Block Name 1 (White Box Description)

<insert diagram of building block 1 here>

Building Block Name 1.1 (Black Box Description)

Building Block Name 1.2 (Black Box Description)

...

Building Block Name 1.n (Black Box Description)

Description of Relationships

Open Issues

5.2.2 Building Block Name 2 (White Box Description)

<insert diagram of building block 2 here>

Building Block Name 2.1 (Black Box Description)

Building Block Name 2.2 (Black Box Description)

...

Building Block Name 2.n (Black Box Description)

Description of Relationships

Open Issues

5.2.3 Building Block Name 3 (White Box Description)

<insert diagram of building block 3 here>

Building Block Name 3.1 (Black Box Description)

Building Block Name 3.2 (Black Box Description)

...

Building Block Name 3.n (Black Box Description)

Description of Relationships

Open Issues

5.3 Level 3

6. Runtime View

6.1 Runtime Scenario 1

6.2 Runtime Scenario 2

6.3 ...

6.4 Runtime Scenario n

7. Deployment View

7.1 Infrastructure Level 1

7.1.1 Deployment Diagram Level 1

7.1.2 Processor 1

< insert node template here>

7.1.3 Processor 2

< insert node template here>

7.1.4 ...

7.1.5 Processor n

< insert node template here>

7.1.6 Channel 1

7.1.7 Channel 2

7.1.8 ...

7.1.9 Channel m

7.2 Infrastructure Level 2

8. Concepts

8.1 Domain Models

8.2 Recurring or Generic Structures and Patterns

8.2.1 Recurring or Generic Structure 1

<insert diagram and descriptions here>

8.2.2 Recurring or Generic Structure 2

<insert diagram and descriptions here>

8.3 Persistency

8.4 User Interface

8.5 Ergonomics

8.6 Flow of Control

8.7 Transaction Procession

8.8 Session Handling

8.9 Security

8.10 Safety

8.11 Communications and Integration

8.12 Distribution

8.13 Plausibility and Validity Checks

8.14 Exception/Error Handling

8.15 System Management and Administration

8.16 Logging, Tracing

8.17 Business Rules

8.18 Configurability

8.19 Parallelization and Threading

8.20 Internationalization

8.21 Migration

8.22 Testability

8.23 Scaling, Clustering

8.24 High Availability

8.25 Code Generation

8.26 Build-Management

9. Design Decisions

9.x.2 Constraints

9.x.3 Assumptions

9.x.4 Considered Alternatives

9.x.5 Decision

9.1 <Decision Topic 1>

9.2 <Decision Topic 2>

9.3 <Decision Topic 3>

...

10. Quality Scenarios

It contains:

- Quality Tree (sometimes called utility tree), an overview of the quality requirements**Quality Tree**

10.2 Evaluation Scenarios

11. Technical Risks

12. Glossary