Data Documentation Initiative

Best Practice

Subject:

High-Level Architectural Model for DDI Applications (2009-02-22)

Document identifier:

DDIBestPractices_HighLevelArchitectureForApps.doc.PDF

Location:

http://www.ddialliance.org/bp/DDIBestPractices_HighLevelArchitectureForApps.doc.PDF

Authors:

Karl Dinkelmann, Pascal Heus, Chuck Humphrey, Jeremy Iverson, Jannik Jensen, Sigbjørn Revheim, Joachim Wackerow

Editors:

Jeremy Iverson

Intended audience:

This document is for software designers who are developing DDI applications. The designers may be familiar or unfamiliar with the DDI specification.

Abstract:

This best practices document looks at a possible way to design components that can be combined to create DDI applications. Given that object-oriented design is the most common programming paradigm, and that systems are often based around service-oriented principles, and given the modular design of DDI 3.0 itself, this document provides an architectural model that can be a reference point for implementers. The document also takes into consideration issues of maintenance and management of DDI applications, and discusses best practices for application documentation and configuration. The focus is on interoperability of DDI applications.

Status:

This document is updated periodically on no particular schedule. Send comments to editor: ddi-bp-editors@icpsr.umich.edu

Table of Contents

1 Introduction

This best practices document looks at a possible way to design components that can be combined to create DDI applications. The paper is targeted at developers, but it does not assume a high level of DDI knowledge. It is intended to serve as a starting point for developers new to the DDI.

1.1 Problem statement

Software developers who are new to the DDI 3.0 standard may find the standard daunting. This best practices document provides an overview of how an application may be structured so that developers have a starting point for the design of their application.

1.2 Terminology

The key words *must, must not, required, shall, shall not, should, should not, recommended, may,* and *optional* in this document are to be interpreted as described in **[RFC2119].** Additional DDI standard terminology and definitions are found in http://www.ddialliance.org/definitions/.

2 Best Practice Solution

2.1 Definitions

DDI: When used without a version, DDI refers to the latest DDI specification, currently version 3.0. When older versions are referenced, the version number will be explicitly specified.

DDI community: Any person or organization working with the DDI specification.

DDI application: A software application that reads and/or writes DDI XML.

Specification: The DDI specification.

Component: A piece of software with a specific purpose with a well-defined input and well-defined output.

Middleware: In the context of this best practices paper, middleware refers to utilities that manage the interface between the DDI metadata model and application services or high-level end-user tools.

Task: An activity that a person undertakes in order to create, edit, or view documentation about data.

End user: Person performing work in the data life cycle for whom DDI metadata is required. The end user will likely not even be aware of the DDI metadata in the application he or she is using.

Java: Java is a programming language expressly designed for use in the distributed environment of the Internet. It was designed to have the "look and feel" of the C++ language, but it is simpler to use than C++ and enforces an object-oriented programming model.

Eclipse: Eclipse is an ongoing project in support of an open source integrated development environment (IDE). Eclipse provides a framework and a basic platform (called the Eclipse Platform) that allows a company to build an integrated development environment from plug-in software components provided by Eclipse members.

GNU-LGPL: The GNU Lesser General Public License (formerly the GNU Library General Public License) is a free software license published by the Free Software Foundation.

Unicode: Unicode is a computing industry standard allowing computers to consistently represent and manipulate text expressed in most of the world's writing systems.

GUI: A GUI is a graphical (rather than purely textual) user interface to a computer.

API: An API (or Application Programming Interface) is a language and message format used by an application program to communicate with the operating system or some other control program such as a database management system (DBMS) or communications protocol. APIs are implemented by writing function calls in the program, which provide the linkage to the required subroutine for execution.

Internationalization: Internationalization is the process of planning and implementing products and services so that they can easily be adapted to specific local languages and cultures, a process called localization.

DDI instance: A DDI Instance is the top-level wrapper for any DDI document. It may contain a set of top-level elements, which generally correspond to the modular breakdown within DDI. Every DDI Instance will use this wrapper, regardless of its content.

Resource package: A resource package is a means of packaging any maintainable set of DDI metadata for referencing as part of a study unit or group. A resource package structures materials for publication that are intended to be reused by multiple studies, projects, or communities of users. A resource package uses the group module with an alternative top-level element called Resource Package that is used to describe maintainable modules or schemes that may be used by multiple study units outside of a group structure.

2.2 Best Practice behavior

Given that object-oriented design is the most common programming paradigm, and that systems are often based around service-oriented principles, and given the modular design of DDI itself, it makes sense to have a reference model for the componentization of functions within DDI-based systems. This document provides an overview of the component architecture that DDI applications may use. Additionally, it discusses resources available to developers creating DDI applications, as well as general software application development best practices.

DDI Profiles

DDI applications will inevitably support only a portion of the DDI specification. Applications should describe which portions of the standard they implement. This can be described using DDI profiles. See the Creating a DDI Profile Best Practice document for details.

DDI Application Components

This best practice defines a standard architectural model to be used as a reference point for implementers. This section describes several of the tiers and components that may be used in creating a DDI application.

Applications should be designed in such a way that each component has a single, stand-alone purpose. Components should avoid overlapping functionality. Depending on the development paradigm being used, components may take the form of assemblies, classes, methods, or other programming constructs.

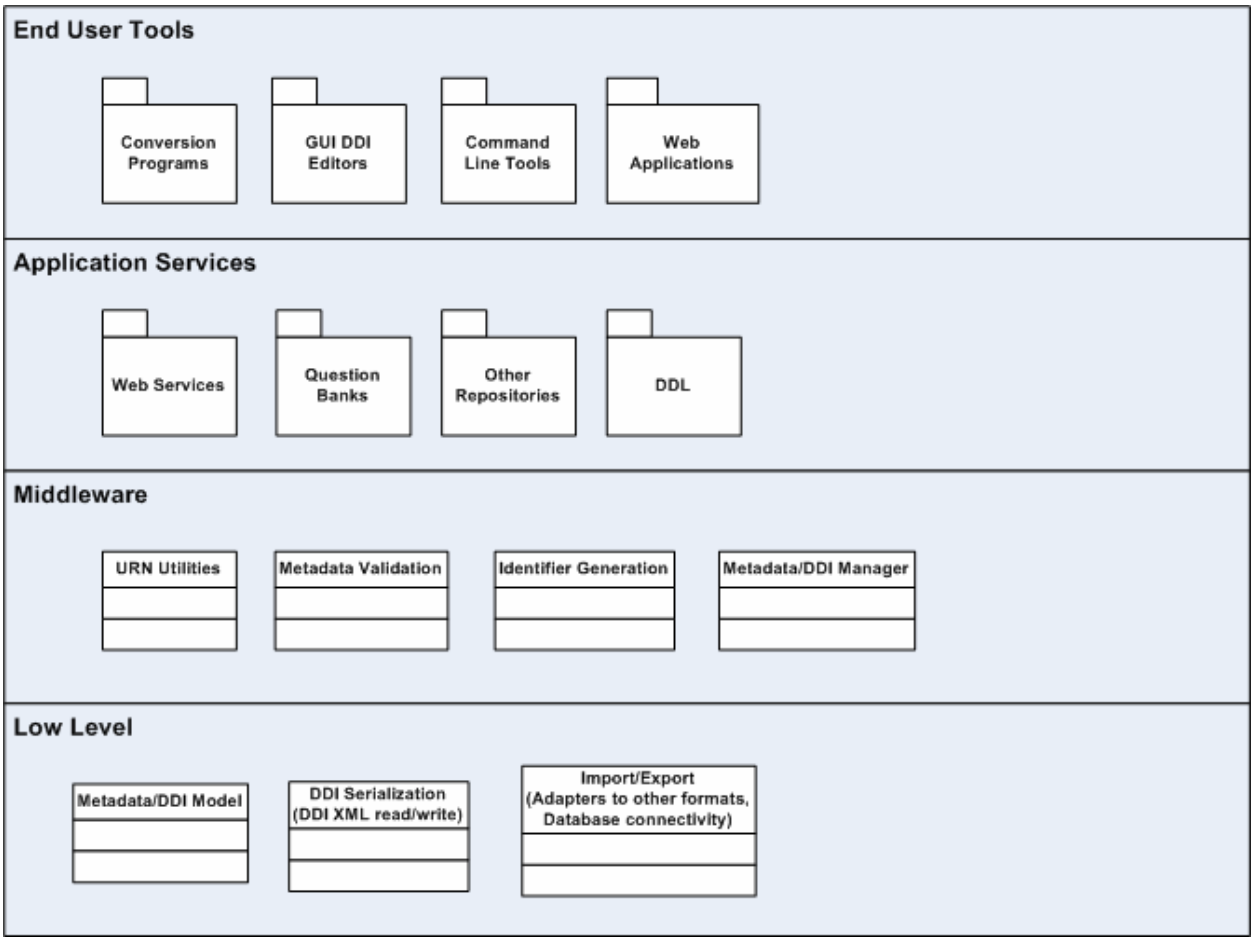

Figure 1: Sample Component Model

Components may be created as needed by application developers. In some cases, it may be possible to use existing third-party components. For more information on possible sources of DDI-related components, see the DDI Application Development Resources section below.

Components that allow users to generate DDI should ideally implement the area of the specification completely so the end user can rely exclusively on the tool for that aspect of metadata management.

High-level components

High level components are considered the software front-ends with which users interact. These could be rich desktop applications, Web applications, or command-line tools. These could include utilities with a single purpose such as extracting metadata from an existing file and storing it in DDI format, full-featured metadata creation and editing suites, or anything in between.

It should be clear to end users which stage of the data life cycle a tool addresses. The purpose of the tool will imply what sorts of things can be used as input, and what sorts of things will be output.

These applications should not necessarily mirror the DDI model, but should provide the user with a highly usable interface for working with the metadata. See the DDI Identifier Best Practices document for more information.

Application services

Application services components are application services called by the high-level components. These could be in the form of Web services.

Examples include repositories based on DDI schemes such as question banks or concept banks.

DDI transmitted from or to application services may be held in resource packages. The DDI should be wrapped in a DDI instance element, even if only a small snippet of DDI is included. See DDI User Guide Part II, line 2215 for details.

Middleware components

Middleware components are utilities that manage the interface between the DDI metadata model and application services or high-level end-user tools. Some examples of middleware components are described here.

Identifier Generation

DDI requires specialized forms of identifiers. Application developers will need to create a component to create these identifiers. See the DDI Identifer Best Practices document for details [see References section].

URN Resolution

In DDI, URN identifiers may be resolved to locate resources within a DDI instance. The DDI instance holding the identified resource could be held internally or externally to the current DDI instance. Application developers will want to create or use a specialized component to provide this URN resolution functionality. See the DDI URN Resolution Best Practices document for details [see References section].

DDI Validation

In order to be valid, DDI instances generated by application components must validate against the DDI specification schema. Instances must also validate against second-level validation tools as described in the Interoperability with Other DDI Applications section below. Developers will want to use a component that provides this validation functionality.

DDI Manager

A DDI Manager component allows developers to work with the underlying metadata model in a convenient manner. It provides access to metadata that are required for a specific task. It may provide access to the elements actually used by other components, instead of all DDI elements. It can also provide easy ways of retrieving desired elements by providing methods for searching and filtering.

Group Manager

The DDI grouping mechanism allows the definition and redefinition of hierarchical relationships among objects. See DDI Overview Part I, line 1819 for details.An application may provide grouping functionality such as editing, regrouping, and extracting. This functionality will require a component that allows items to be put into groups, as well as for existing group structures to be edited.

Low-level components

Low level components are the software libraries on top of which middleware and high-level components are implemented. Several examples follow, but this list is by no means exhaustive.

Metadata/DDI Model

All DDI applications require a model of the data with which they work. The model allows developers to query, add, update, and remove metadata. Depending on the scope of the application, the model may represent the full DDI specification, or a subset of the specification.

DDI Serialization

All DDI applications must be able to read and/or write valid DDI XML. A serialization component provides one or both of the following functions:

- Read and parse valid DDI XML and load the data into the application's DDI model
- Write the application's DDI model to an XML file

The DDI XML created by an export component must be a complete DDI instance. See the Interoperability with Other DDI Applications section below for details about ensuring interoperability among DDI applications.

Import/Export Components

DDI applications may wish to read from and write to file formats other than DDI.

Interoperability with Other DDI Applications

DDI applications must interoperate well with other DDI applications written by the DDI community. A DDI application must have output that other DDI applications are able to use, and must be able to take as input DDI generated by other applications.

DDI XML must be wrapped in a DDI instance element. This applies whether the DDI is loaded from a file or retrieved from an application service. The documentation for this element can be found in the DDI User Guide Part II, line 269.

DDI instances must validate against the schema published as the standard. DDI instances should also validate against second-level validation tools like the reference validator available from the DDI Foundation Tools web site (http://tools.ddialliance.org/?lvl1=library) . Identifiers and URNs used in the DDI instance should conform to the best practices described in the DDI Identifier Best Practices document and the DDI URN Resolution Best Practices paper.

XML allows namespace prefix declarations to be used flexibly. Developers should write DDI to use namespace prefix conventions as they are published in the DDI specification. See DDI User Guide Part II, line 208 for more information. Applications should be able to read DDI documents even if an instance does not use the conventional prefixes.

DDI Application Development Resources

DDI foundation tools program

The DDI Foundation Tools Program (DDI-FTP) is an initiative aimed at the development of a Foundation Framework and a Toolkit to support the implementation of DDI applications and utilities. The DDI-FTP implements several of the components described above. The components are mainly available under the open source GNU-LGPL, so developers of both open source and proprietary applications can make use of them. The tools are mainly written in Java. The user interface portions are mainly based on the Eclipse platform.

Application/component catalog

DDI application developers may submit information regarding their development efforts to the DDI Alliance. The DDI Alliance will publish the details of the application or component in a directory. This will allow developers to get exposure for their software, and may help to foster collaboration among developers.

A component listing consists of the following:

- Title
- Developer
- Description
- License
- Screenshots (if appropriate)

DDI Development Pitfalls

Link integrity

Developers should be sure elements being linked to exist in the document. When an object is deleted, links to that object should be removed. If an object identifier is renamed, links to that object should be updated.

DDI namespace URNs

DDI namespace URNs contain the version number of the schema (e.g., “ddi:instance:3_0”). This number will change when new DDI XML schema versions are released (e.g., “ddi:instance:3_01”). This could present issues for DDI parsers that expect certain namespace URNs. DDI components should be careful to work with potential future version numbers, for example, by not depending on explicit namespace URNs.

Avoid circular references

DDI applications should not create circular reference patterns, and should detect circular reference patterns in order to avoid abnormal program behavior.

Loss of metadata

When importing metadata from a DDI file, all metadata should be preserved so that everything in the original DDI instance is reflected in a new version. Identifiers from imported DDI should be preserved.

If an application manages data that cannot be stored in DDI format, it should warn the user that those metadata are not supported when exporting to DDI format.

If an application reads a DDI file that contains metadata that the application will not preserve or manage, it should warn the user that the metadata will be lost.

Be aware of encoding

Make sure the characters being saved in an XML instance are a valid part of the encoding being used. Use an encoding that supports the characters. Work with Unicode (generally UTF-8).

General Software Development Recommendations

This section describes some general software application development best practices. DDI application developers should follow these guidelines in order to provide the best user experience.

Documentation

DDI applications should include effective end-user documentation so researchers can use the application productively. Several types of help should be available to meet the needs of different types of users.

Tutorials

A Tutorials section may contain a series of "how to" articles describing how to perform common tasks. This is useful to users who are new to an application or who only use it occasionally.

Reference

A Reference section contains detailed descriptions of each part of the software. Each form and user interface element of a GUI may be described, along with screenshots where appropriate. This is useful for those who use an application frequently and need quick access to reference information.

Technical Reference

A Technical Reference section provides information for system administrators, programmers, and other technical users. It can include information about installation, file formats, and software developer's kits.

For developers providing their source code or an API, thorough code documentation should be provided. This could be derived from JavaDoc or similar documentation systems.

Knowledge Base

A Web-based Knowledge Base allows developers to deliver up-to-date information to end users. A knowledge base should contain searchable and well-categorized information related to the software. Articles can be added to the knowledge base based on user feedback.

Internationalization

DDI applications should be written with globalization in mind. This will enabled localized versions of applications to be created, increasing the audience for the tool. Applications should support Unicode text encoding.

Logging

DDI applications should write logs with enough detail so that end users can submit the logs to developers in the event of unanticipated behavior, and the logs will provide sufficient information to the developers to diagnose any issues with the software. Users should be able to adjust the level of logging (for example, Debug, Info, Warn, Error). Developers may wish to use existing logging libraries such as log4j or log4net.

Keep in mind that this log data may be useful metadata usable at different stages of the life cycle. Log file formats should be documented.

Performance

DDI applications should operate responsively and should not overuse a system's resources.

Configuration

DDI applications should provide a reasonable level of end-user customization. Commonly used configuration systems should be used, for example, Java properties files.

Platform independence

Ideally, DDI applications should be written in a platform-independent manner so they may run on Windows, Mac OSX, Linux, and other operating systems.

Intellectual property issues of third-party components

Developers should be aware of licensing implications of any third-party components they use. Third-party licenses should not be more restrictive than the desired license of the developer's application.

2.3 Discussion

The paradigm used in this document for structuring the identified components is based on an object-oriented, tiered approach. Developers who do not use this approach should still be able to gain insight from the description of the functionality identified in this document.

These best practices provide the foundation for implementing feature-rich, interoperable DDI applications. They do not address every aspect of the DDI specification. Applications that wish to provide certain functionality (e.g., geographical metadata management or a statistical engine) will need components to provide those features.

The DDI Alliance is committed to digital preservation, which may require applications to incorporate additional preservation standards (e.g., PREMIS, http://www.oclc.org/research/projects/pmwg/).

2.4 Examples

Applying Architectural Components to Data Life Cycle Application

This section contains several examples of how a developer might apply the architectural components to create specific applications for the data life cycle.

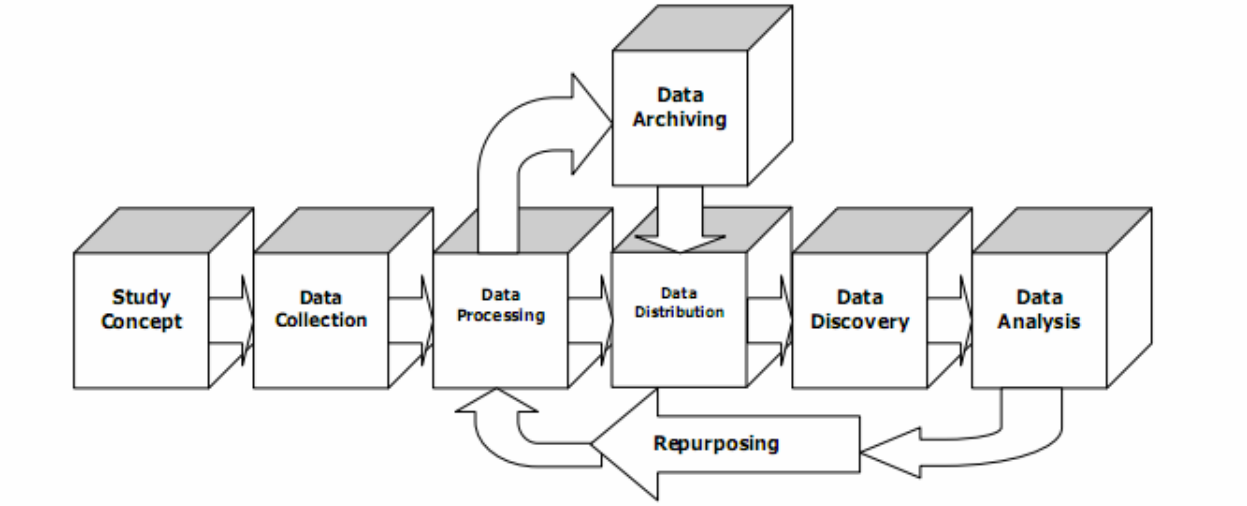

Figure 2: From the DDI User Guide Part I Page 6

Data collection example

A sample application that addresses the data collection stage of the data life cycle would be a survey instrument documentation application. Such an application might read source code from a Computer Assisted Interviewing system, populate a metadata model, generate documentation, and store a representation of an instrument in DDI format. The application may submit questions to an online question repository.

In order to create this application, the developer may make use of the following components.

- Low level
  - DDI/Metadata model
  - Serialization
  - Import and export
- Middleware
  - ID generation
  - Validation
- Application services
  - Question bank Web service
- High level
  - Graphical user interface

- Questionnaire visualization component

Data discovery and data analysis example

A Web application allows researchers to browse metadata to discover data suitable for their work. The application might dynamically produce descriptive statistics based on various physical data product formats, and allow users to generate custom datasets.

- Low level
  - Metadata model
  - Serialization for reading DDI
  - Exporters for generating custom datasets (which could be used by the statistical engine)
- Middleware
  - DDI Manager to access underlying metadata model.
- Application services
  - Variable bank
  - Statistical engine
- High level
  - Web pages to display study metadata and allow the user to navigate through it
  - Dynamic Web pages to display variables selected by the user (data visualizations), let the user select descriptive statistics, and display results

Integrating components across the data life cycle

DDI allows applications that address different stages of the data life cycle to work together. An example application could allow researchers to discover data based on questions found in survey instrument documentation, and then proceed to extract the relevant data and perform analysis on it. This process can be done using the two previously described applications if they both follow interoperability best practices.

Publicizing applications

Developers who want information about the application or component to be available to the DDI community should submit an entry to the DDI application/component catalog. The catalog can be found at the DDI Foundation Tools site -- http://tools.ddialliance.org/. The application should contain a DDI Profile that describes which parts of the DDI specification it uses.

3 References

DDI User Guide Part I (http://www.ddialliance.org/ddi3/index.html)

DDI User Guide Part II (http://www.ddialliance.org/ddi3/index.html)

DDI Tools Foundation Web site (http://tools.ddialliance.org/)

DDI Tools Foundation Roadmap (http://tools.ddialliance.org/?lvl1=ftp&lvl2=roadmap)

DDI Profile Best Practices:
http://www.ddialliance.org/bp/DDIBestPractices_CreatingAProfile.doc.PDF

DDI Identifiers Best Practices:
http://www.ddialliance.org/bp/ManagementOfDDI3Identifiers.doc.PDF

DDI URN Resolution Best Practices:
http://www.ddialliance.org/bp/URNsAndEntityResolution.doc.PDF

3.1 Normative

[RFC2119] S. Bradner, Key words for use in RFCs to Indicate Requirement Levels, http://www.ietf.org/rfc/rfc2119.txt, IETF RFC 2119, March 1997.

OASIS, Best Practice, http://www.oasis-open.org/committees/uddi-spec/doc/bp/uddi-spec-tc-bp-template.doc, 2003

Appendix A. Acknowledgments

The following individuals were members of the DDI Expert Workshop held 10-14 November 2008 at Schloss Dagstuhl, Leibniz Center for Informatics, in Wadern, Germany.

Nikos Askitas, Institute for the Study of Labor (IZA)

Karl Dinkelmann, University of Michigan

Michelle Edwards, University of Guelph

Janet Eisenhauer, University of Wisconsin

Jane Fry, Carleton University

Peter Granda, Inter-university Consortium for Political and Social Research (ICPSR)

Arofan Gregory, Open Data Foundation

Rob Grim, Tilburg University

Pascal Heus, Open Data Foundation

Maarten Hoogerwerf, Data Archiving and Networked Services (DANS)

Chuck Humphrey, University of Alberta

Jeremy Iverson, Algenta Technology

Jannik Vestergaard Jensen, Danish Data Archive (DDA)

Kirstine Kolsrud, Norwegian Social Science Data Services (NSD)

Stefan Kramer, Yale University

Jenny Linnerud, Statistics Norway

Hans Jørgen Marker, Danish Data Archive (DDA)

Ken Miller, United Kingdom Data Archive (UKDA)

Meinhard Moschner, GESIS - Leibniz Institute for the Social Sciences

Ron Nakao, Stanford University

Sigbjørn Revheim, Norwegian Social Science Data Services (NSD)

Wendy Thomas, University of Minnesota

Mary Vardigan, Inter-university Consortium for Political and Social Research (ICPSR)

Joachim Wackerow, GESIS - Leibniz Institute for the Social Sciences

Wolfgang Zenk-Möltgen, GESIS - Leibniz Institute for the Social Sciences

Appendix B. Revision History

Rev	Date	By Whom	What
0.1	10 Nov 2008	Jeremy Iverson	Initial draft
0.9	2009-02-15	Stefan Kramer	Changed sections above ToC to heading 3, rebuilt ToC for heading levels 1-2 only. Added rev. date to Subject. Added future URLs for DDI BPs to References section.

Appendix C. Legal Notices