The KISS (Keep It Simple and Straightforward) Principles for ORM Products from Software Tree

A White Paper

by

Damodar Periwal

Introduction

Using object-relational mapping (ORM, a.k.a. OR-Mapping) technology to bridge the impedance mismatch between object-oriented programs and relational databases has become a well-established programming practice for modern applications. Eliminating large amount of complex, non-intuitive, and error-prone JDBC/ADO.NET/SQL code surely improves productivity. Good isolation of data integration layer promotes design clarity and facilitates easier maintainability of the system.

ORM products come in different shapes and sizes. Many large and small companies, in-house development teams, and open-source communities offer these products. While all of these products aim for simplifying the data integration aspects of an application, they sometimes vary in approaches used for defining the mapping between object and relational models, and accomplishing the runtime data exchange. Each product has its own set of features and advantages.

This white paper is not about comparing feature-lists of different ORM products. Instead, it's going to describe the KISS (Keep It Simple and Straightforward) principles followed in the design and development of Software Tree's ORM products, JDX™, NJDX™, and JDXA™. JDX is an ORM product for the Java platform, NJDX is an ORM product for the .NET platform, and JDXA is an ORM product for the Android platform. JDX, NJDX, and JDXA all share similar internal designs and expose similar interfaces.

The paper concludes with the essential and cumulative benefits of following these KISS principles.

The KISS Principles

1. Solve the most important problem (object relational impedance mismatch) in the simplest possible way.
2. Don't make the solution more complex than the original problem.
3. Be completely non-intrusive to the object model.
4. Give full flexibility in object modeling.
5. Make it easy to define, modify, comprehend, and share the mapping specification.
6. Avoid source code generation for data access.
7. Keep the mapping engine as much stateless as possible.
8. No mind reading.
9. Avoid creating a new query language.
10. Expose small number of simple and consistent APIs.
11. Absorb database-specific dependencies in the internal implementation.
12. Provide simple and intuitive pass-thru mechanisms for accessing databases directly.
13. Optimize data access logic automatically.
14. Stick to 90/90 rule about product features.
15. Keep the internal implementation simple, extensible, and efficient.
16. Offer intuitive tools to deal with object models, database schema, and the mapping.
17. Provide a straightforward installer, lucid documentation, and readymade examples.

The KISS Principles in JDX, NJDX, and JDXA ORMs (OR-Mappers)

Following are the main KISS principles along with their explanation, rationale, and advantages. They are not in any particular order of importance.

- **Solve the most important problem (object relational impedance mismatch) in the simplest possible way.**

  The most important problem for developers is writing and maintaining endless lines of complex JDBC/ADO.NET/SQL code to exchange data between their business objects and their relational artifacts (tables, stored procedures). Make solving that problem the main focus of the product.

  Advantages: The ORM product focuses on the most important problem and solves it efficiently.

- **Don't make the solution more complex than the original problem.**

  Providing an ORM solution that solves the problem at the expense of exposing a needlessly complex programming model nullifies the advantages offered by using the solution.

  Don't make the developer jump through many hoops just to be able to use your product. Minimize the number of steps and the number of additional files needed by using your OR-Mapper. Avoid adding steps like preprocessing, source code generation, and compilations. Avoid the need to create many different mapping or helper files. Essentially, don't encumber the design, coding, build, and deployment processes with many logistics operations and additional steps.

  Advantages: Rather than becoming a development headache, the OR-Mapper improves developer productivity.

- **Be completely non-intrusive to the object model.**

  Don't impose any rules on business class definitions. Allow developers to use POJOs (Plain Old Java Objects) or POCOs (Plain Old CLR Objects). Don't require any superclasses, superinterfaces, or proxy objects in business class definitions. Avoid byte code or IL code manipulations.

  Advantages: A clean object model helps in easier implementation and smoother evolution of business logic. Additionally, remote clients can be sent the serialized POJOs or POCOs without requiring the remote application to include any of the ORM product specific libraries, which it does not need otherwise. It also reduces runtime dependencies.

- **Give full flexibility in object modeling.**

  Object modeling should be orthogonal to ORM. The developer should be able to create a flexible object-oriented domain model per business needs without having to worry about how to persist that model in a relational database. In other words, don't impose any restrictions on class definitions just because they have to be used with OR-Mapper. Support class-hierarchies, associations and aggregations. Support one-to-one, one-to-

many, and many-to-many relationships. Support deep/shallow operations, lazy fetches, and persistence-by-reachability.

Advantages: Adherence to a true domain model helps in better design and integration of the application.

- **Make it easy to define, modify, comprehend, and share the mapping specification.**

  Mapping specification is the heart of an OR-Mapper. Keeping the mapping separate from the object class definition (source files) is an important decoupling principle.

  Mapping should be defined externally and in a declarative way based on simple grammar. Make the specification compact; most default mapping should be automatically deduced. Avoid verbosity.

  XML, although declarative, is not preferable for mapping specification because it is neither simple to write nor easy to comprehend.

  Putting mapping in source files (through annotations or attributes), although declarative, is not preferable because of the following disadvantages:

  - Clutters source code.
  - Requires recompilation or reprocessing of the source files for even trivial mapping change during development.
  - Requires the knowledge of the source code or requires changing the source code and its recompilation for any deployment time configuration.
  - There is no one place to get a full, clear picture of the mapping specification.
  - Provides no way to deal with legacy or third-party classes for which no source code may be available.

  So, keep the mapping specification for all the persistent classes of a logical mapping unit at one place for easy inspections and modifications. Allow multiple mapping units to be used in one application to promote better partitioning of the application logic.

  Advantages: The ORM system is easy to understand, use, and manage.

- **Avoid source code generation for data access.**

  Although automatic source code generation alleviates the problem of writing tedious and repetitive low-level infrastructure code for data access, it unfortunately imposes maintenance burden of huge amounts of such complex auto-generated source code. Besides, the source code for data access needs to be regenerated for the slightest change in the object model or the relational model. There is also the additional step and overhead of recompiling such code in your application.

  Create a metadata-driven mapping engine that can be leveraged across any object or relational model. Take advantage of reflection facility of the underlying platform to avoid dependence on source code generation.

  Advantages: Avoiding source code generation creates a simpler, cleaner, and more dynamic solution.

- **Keep the mapping engine as much stateless as possible.**

  Don't worry about how the developer uses the business objects in the application. The detached model lets the objects have their own lifecycle. From an OR-Mapper point of view, don't be attached to those objects. Don't waste cycles in keeping track of their states.

  Advantages: The mapping engine remains simple and focused. It does not create unnecessary runtime overhead of tracking the state of every persistent object. This results in better performance.

- **No mind reading.**

  Avoid mind reading by making presumptions about the intentions of the user about the persistence of an object based on certain access patterns. You can never be 100% sure that just because the user modified an object, it should be updated in the database. Let the user decide and direct the mapping engine what to do and when. If the developer has to explicitly invoke insert, delete, and query operations anyway, he can also explicitly invoke an update operation when the application semantics requires that.

  Advantages: The mapping engine does not cause data corruption by saving changes that the user did not intend to be saved. The user remains firmly in control. The user does not have to worry about telling the ORM engine about which objects not to save even though they might have been changed in the course of executing some business logic. The usage of an ORM engine is simple and straightforward.

- **Avoid creating a new query language.**

  New query languages require too many details to be complete and useful. Don't impose new query languages and complex semantics on developers. Don't create complex expression builder APIs. Leverage SQL's well-understood expressive power for creating predicates. Abstract other query-related parameters in simple and intuitive APIs.
  .
  Advantages: Fast learning curve. Easy-to-understand programs. Avoiding the overhead related to query parsing and compilation speeds up internal implementation.

- **Expose small number of simple and consistent APIs.**

  Think through all the possible use cases and provide a minimal set of APIs. Design APIs that are extensible and expose new features in a consistent way. Avoid exposing every new feature with new APIs.

  Advantages: Avoids API bloat that can otherwise make the product confusing and harder to use. Small number of APIs also helps in maintaining backward compatibility.

- **Absorb database-specific dependencies in the internal implementation.**

  Automatically issue appropriate SQL statements (both DDL and DML) as per the backend database. For data exchange, do appropriate runtime data type conversions between the object field data types and the table column data types. Absorb any behavioral differences of different database drivers within the ORM implementation.

Advantages: Applications become database and driver independent. They can easily be ported from one database to another.

- **Provide simple and intuitive pass-thru mechanisms for accessing databases directly.**

  One can never fully anticipate and provide for all the possible usage of relational functionality through the ORM interface. So provide simple APIs that can be used to execute arbitrary SQL statements against the underlying data source.

  Advantages: A simple pass-thru interface can help the developer easily meet those rare needs of directly accessing the underlying database system without having to go around the ORM system.

- **Optimize data access logic automatically.**

  Data access is typically the most time-consuming operation. Minimize database trips, use prepared statements where possible, use connection pools, allow bulk operations, and provide object caching. The more automatic and implicit these operations are, the easier the product would be to use.

  Advantages: High performance without much extra effort on the part of the developer.

- **Stick to 90/90 rule about product features.**

  You can never create a product that can meet 100% of the needs of 100% of the users. Create a product that can meet at least 90% of the needs of at least 90% of the users. Don't add features that are rarely used and are hard to implement.

  Advantages: A practical product that is easy-to-understand and use. Implementation is not overloaded with unnecessary or rarely-used features.

- **Keep the internal implementation simple, extensible, and efficient.**

  External simplicity of an ORM product does not have to come at the cost of a complex internal implementation. Follow best programming practices, write modular and thread-safe code, create reusable components, minimize the code path lengths, and cache and reuse static metadata. Don't over-engineer any module. Don't create dependencies on non-standard libraries. Deliver the product with one runtime library that depends only on the virtual machine and standard database drivers.

  Advantages: A fast, robust, and lightweight implementation.

- **Offer intuitive tools to deal with object models, database schema, and the mapping.**

  A good ORM product should come with easy-to-use tools to deal with new or existing object models and new or existing database schemas. Provide both command-line and GUI tools to accomplish the following tasks:

  - Forward-engineer a database schema from an object model
  - Reverse-engineer object models from an existing database schema
  - Verify mapping with live data.

Advantages: Easy to leverage legacy data for ORM. Streamlining of development process by quick synchronization of object and relational models. Simplified integration with script-based configuration and deployment facilities. Easy verification of mapping specification helps in rapid prototyping and painless diagnosis of any mapping problems.

- **Provide a straightforward installer, lucid documentation, and readymade examples.**

  Installer, documentation, and example programs provide a crucial help in introducing any technology quickly and properly. Make the product installation simple and straightforward. Set or modify the needed environment variables automatically. Provide extensive documentation with comprehensive user manuals, detailed explanations of APIs, plenty of ready-to-run example programs, and tutorials.

  Advantages: Easy installation, lucid documentation, and working examples simplify the understanding and usage of the product, resulting in improved productivity.

Summary

This white paper has identified and discussed the main principles of simplicity (KISS) that we have followed in designing and developing our ORM products. It also explains the importance and advantages of these principles for creating a practical, efficient, and user-friendly solution.

Simplicity does not mean not providing essential features. Simplicity does not mean being unsophisticated. Simplicity does not mean lacking performance. Simplicity is not a position of compromise; rather it's a position of strength. Penchant for simplicity often leads to elegance, robustness, and ease-of-use.

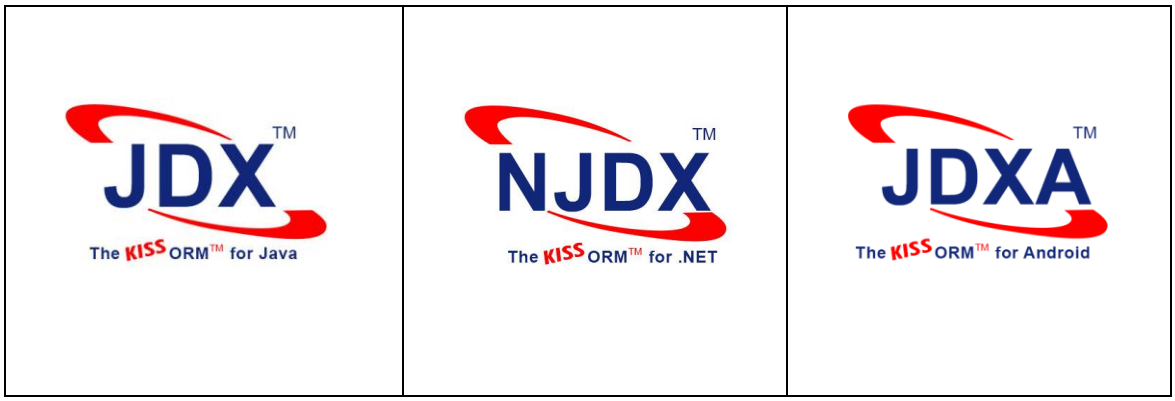
The KISS principles explained above have helped us create fast, flexible, versatile, lightweight, robust, database-agnostic, and easy-to-use OR-Mappers, JDX for Java, NJDX for .NET, and JDXA for Android. Resulting technologies are easy to understand, extend, adapt, and integrate. JDX was released in early 1998. NJDX was released in late 2005. JDXA was released in 2015. Free evaluation versions of these ORM products are available from Software Tree's web site at http://www.softwaretree.com.

Acknowledgements

The author would like to thank Richard Brewster, Prashant Periwal, and Neha Sharma for reviewing this paper and offering valuable feedback to improve the contents and the presentation of the material.

About The Author

Damodar Periwal is the founder and President of Software Tree, Inc. He is the architect and designer of JDX, NJDX, and JDXA ORM products. Damodar has an extensive background in databases, transaction processing, and distributed and object-oriented technologies. He has more than 25 years of industry experience having worked at leading companies like Tandem Computers, Ashton-Tate, Borland International, and TIBCO. He has an MS (Computer Science) degree from University of Wisconsin (Madison) and an undergraduate engineering degree from Birla Institute of Technology and Science, Pilani (India). You can contact Damodar at dperiwal@softwaretree.com.



Note: Java is a trademark of Oracle. .NET is a trademark of Microsoft. Android is a trademark of Google. JDX, NJDX, JDXA, JDX logo, NJDX logo, JDXA logo, The KISS OR-Mapper, The KISS ORM, Software Tree logo are trademarks of Software Tree. All other brands and product names are trademarks or registered trademarks of their respective holders.