8. Conclusion

As mentioned in the introduction chapter, one of the goals of this project was to create a reusable framework that can support transparent persistence. This goal has has been achieved with the reflection features of Java, with the user able to save and restore objects without having to pre-process or post-process the class definitions first. Object-to-relational mapping as well as the inverse mapping has also been implemented successfully. This chapter summarizes and evaluates the work done, and outlines directions for future work.

8.1 Strengths of the Persistence Framework

The following primary features of the framework are considered essential in performing its function effectively:

- **Transparency** – This is an advantage because it means the framework is domain-independent and can be used to save and restore objects in any application, whether in the medical, manufacturing, or telecommunication fields.
- **Reusability** – In the event that the framework needs to be customized to suit a particular domain or persistence mechanism, this can be done easily by extending the existing classes.
- **Querying** – This enables the user to retrieve objects based on criteria on its attributes. In a real world application, this function is very frequently used and is essential for effective object persistence. In addition, partial retrieval of the object graph is supported, improving performance when only certain data need to be retrieved.
- **Updating** – This allows updating of selected attributes in the object graph in the database. Therefore, parts of objects can be modified and saved without having to save the whole object graph again.
- **Transactions and locking** – In a multi-user environment, this is essential to prevent conflicts and inconsistencies in the database.
- **Portability** – As the framework is written completely in Java, it can be supported on virtually any system that has a Java implementation.

8.2 Limitations of the Persistence Framework

The framework is not suitable for every possible application, due to the limitations of certain implemented functions as follows:

- **Only supports simple queries** – The framework does not allow nested queries, joins on different object graphs, and execution of stored procedures. This is required in large systems optimized for a certain domain. Without it, the framework is only suited for use in small and medium sized systems.
- **Does not support all Java classes** – The framework might encounter problems when trying to save system-specific Java classes. Thus it is not recommended for use in system level applications as it has not been tested fully in this area.
- **Security** – Since the framework can access the private and protected attributes of the user's objects, this can pose a great security risk especially if the object contains sensitive data such as passwords or encryption keys. Therefore it cannot be used in high-security environments.

8.3 Comparison with other Persistence Frameworks

In comparison, the transparent object persistence offered by this framework is a key feature most persistence frameworks fail to implement. Java Blend by Sun Microsystems, CocoBase by Thought Inc., Odapter by Hewlett-Packard, and Secant Extreme POS by Secant Technologies all implement object persistence, but they require the user to either pre-process the classes, or declare the class structure in a proprietary language. Even fully object-oriented databases such as Object Design's ObjectStore does not support transparent object persistence. One product that does support transparent persistence is Java Object Persistence (JOP) 0.4a by David Rothwell, but it utilizes native methods and as such, is not portable across platforms.

In terms of querying, Secant Extreme POS has a more powerful query language than the framework developed in this project. Secant's querying allow for sorting,

grouping, multiple object graph joins, and stored procedures. The other frameworks either support simple queries or none at all.

As the other persistence framework products are based on a non-transparent design, it is relatively easy for them to implement proxy objects where attributes are fetched only when the application accesses them. This is also referred to as late fetching and can improve performance especially when dealing with large object graphs. As proxy objects cannot be implemented in Java without compromising transparency, the persistence framework in this project allows for selective retrieval of attributes in the querying. This allows the user to retrieve portions of an object graph only when it is needed.

8.4 Future Work

In terms of performance, the framework can be improved greatly on this aspect. As the framework has to support various persistence mechanisms of differing capabilities, it has to follow the lowest common denominator and not depend on performance-enhancing features offered by the mechanism such as user-defined types, arrays, stored procedures, or binary large objects. One solution would be to have the persistence broker recognize mechanisms that support certain features and execute methods that take advantage of this. On the other hand, if the broker is specialised too extensively for each kind of persistence mechanism, it would increase the coupling between the broker and the mechanism, and this would lead to difficulties when modifications need to be made.

As mentioned in Section 8.2, the framework can pose a great security risk when used with sensitive data. A malicious program can masquerade as the persistence broker and obtain access to the object's data. This can be solved by using the features in the java.security package, which implements a security manager and various run-time security measures. Otherwise, the user can encrypt any sensitive data in objects that are passed to the framework.

More user control over the persistence process can also be added as currently the framework saves the entire object including all referenced objects. For large objects this will incur a large storage and performance overhead. Moreover, not all attributes in an

object need to be persisted as objects are bound to contain transient variables. The user might want to choose which attributes to save, perhaps by passing an attribute list to the framework or by implementing its own persistence methods. The solution now is just to set the reference to null when passing the object to be saved.