Transparent Sessions and Transactions

Feasibility study for automating the most important problem related to working with NHibernate

Master Thesis

Anton Lycklama à Nijeholt

UNIVERSITEIT VAN AMSTERDAM

Transparent Sessions and Transactions

Feasibility study for automating the most important problem related to working with NHibernate

ing. Anton Lycklama à Nijeholt

Master Software Engineering
Universiteit van Amsterdam

Prof. Dr. P. Klint, *Thesis Supervisor*
E. van Dillen, *Internship Supervisor*

Institution

University of Amsterdam
Faculty of Science
1098 SM Amsterdam
The Netherlands
www.science.uva.nl

Company

Sogyo
Utrechtseweg 301
3731 GA De Bilt
The Netherlands
www.sogyo.nl

Preface

This master thesis is the result of the one year master course at the University of Amsterdam. The master project has been carried out between 9 January 2007 and 15 August 2007 at Sogyo. In this master thesis I will identify the most important problem related to working with the NHibernate framework. I will then analyse this problem to see if there is a way to automate one or more aspects of this problem to either solve the problem or to decrease the difficulty of it.

This thesis could have not been successfully completed without the help of several people. First of all I'd like to thank my thesis supervisor Paul Klint for giving me great feedback on my work and for guiding me during this master project. I'd like to thank Edwin van Dillen for giving me the opportunity to carry out this master project at Sogyo, for his feedback on my work and the great discussions related to this thesis. I'd like to thank the survey participants for filling in the survey for my master thesis. I'd like to thank Srinivasan Tharmarajah, Pim Vendrig and Vincent Driessen for hearing me out on several problems and aspects of my master thesis. Thanks for the great discussions about several interesting topics. I'd like to thank Ralf Wolter and Arnoud van Zoest for reviewing my thesis and their great feedback on my work. Last not but least I'd like to thank Jurgen Vinju, Hans Dekkers, Jan van Eijk, Hans van Vliet and Paul Klint for stimulating me during this one year master course.

Anton Lycklama à Nijeholt
De Bilt, The Netherlands
August 30, 2007

Content

List of Figures

List of Tables

Chapter 1

Introduction and Motivation

Object persistence frameworks try to lighten the burden of the developer, but do these frameworks introduce new problems? In this thesis we will try to determine if it is possible to create a more efficient development process by automating / generating specific parts during the development of a data-layer with NHibernate (an object persistence framework for .NET). What is object persistence? Object persistence makes it possible to extend the lifetime of objects beyond the execution of a single program [17]. The data of these objects can be saved in several forms (flat text files, structured XML files, relational databases, object databases, etc.). Object persistence can be implemented in several ways. One can hard-code persistence (using SQL code in the source code) or one can make use of persistence frameworks (which is recommended for large-scale applications) [W20].

1.1 Context

The last several years we can see a new trend. More data-layers are making use of object persistence frameworks to simplify development. The developers can focus on more important issues e.g. working on the business layer of an application. It is important that a framework lightens the burden of the developers and does not add more to it. but what kind of new problems do these object persistence frameworks introduce?

1.2 Problem definition

Sogyo makes use of NHibernate as their object persistence framework for .NET. NHibernate is being used in many of Sogyo's software development projects. The purpose of this thesis is to determine whether there are problems related to working with the NHibernate framework. In order to determine which problems developers encounter when using the framework, we will analyse several information acquisition methods and we will pick the best method based on criteria we will define later on (chapter 3). We will then use this method to acquire the information. With the results of this method we will determine the problems related to working with the NHibernate framework. We will focus on the most important problem and we will determine whether it is possible to find a solution for this problem by automating one or more aspects of this problem.

Why do we want to create a more efficient development process? This is considered from an economical and a quality point of view. The more we can automate, the less work we have to do in order to create a data-layer for a new application. If we can complete more projects in the same amount of time the turnover ratio will increase and we might be able to increase our profit. There is however an even more important matter: Quality. If we can reduce the time needed for developing data-layers we will be able to spend more on the requirements and we will likely increase quality this way. If we can decrease time needed for maintenance (e.g. by reducing the number of introduced bugs) it will be possible to create more stable applications.

1.3 Scope

We will investigate whether the process of using NHibernate can be made more efficient by identifying the problems related to working with this framework. When we have identified the problems we will pick the most important problem out of this list based on criteria defined in chapter 3 and we will try to see if we can find a solution for this problem by automating one or more aspects of this problem. It is likely that more solutions can solve this single problem and we will we analyse two approaches in order to determine which solution is best. We first have to determine whether automation is feasible for the most important problem. We will only focus on Web development because of time constraints.

1.4 Research question

The central research question of this thesis is be formulated as follows:

> *Can the development process of data-layers with the NHibernate framework be made more efficient by creating a solution for the most important problem (related to working with the NHibernate framework) by automating one or more aspects of this problem*

There are several sub-questions defined in order to answer the central question.

General

1. How do we find the problems related to working with NHibernate?
2. How do we determine which problem is the most important problem?
3. What is the definition of the most important problem?
4. How do we determine the best solution for the most important problem?

Measurement

5. How do we measure the efficiency gain if we can automate the most important problem?

1.5 Outline

Chapter 1 is an introduction to this master thesis. Chapter 2 will give background information about the NHibernate framework. Chapter 3 describes the problem domain. Chapter 4 gives insight in the subject of this master thesis “Transparent sessions and transactions”. Chapter 5 describes a possible solution to the problem by making use of a Domain Specific Language (DSL). Chapter 6 will make use of a different approach by building a framework to see whether this can solve the problem. Chapter 7 validates the framework based on a target application. Chapter 8 contains the conclusion and a small comparison between both solutions. Chapter 9 is describes the future work of this thesis.

Chapter 2

Background and Context

This chapter will give you background information on the key concepts of Nhibernate [W9]. We will shortly describe the architecture and several important NHibernate constructs. At the end of the chapter, we will discuss several options for session management and we will make use of well-known NHibernate patterns.

2.1 NHibernate

NHibernate is an open-source object persistence framework for .NET. NHibernate is a port of the successful Hibernate object persistence framework for Java. The framework makes it possible to persist objects to a relational database. NHibernate has been started in 2003.

Object persistence frameworks create an abstraction between the data-layer of an application and the technical implementation code needed to retrieve and store data. The business layer will only be aware of the domain objects and doesn't know how these objects are being stored technically (figure 2.1.1). It is possible to reconfigure the way the data is being structured without modifying any code in the application. It is also possible to store data in a different data storage provider, e.g. migrating from MySQL to SQL Server 2000 or Oracle without changing any code within the project. This will significantly reduce the costs needed to modify existing applications when migrating to a different data storage solution.

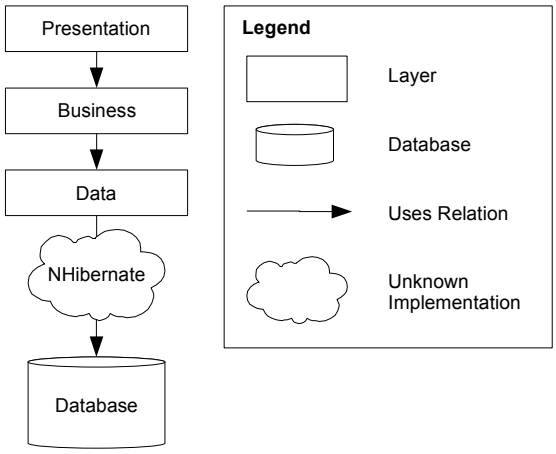

Figure 2.1.1: NHibernate in a 3-tier architecture

2.1.1 Architecture

The NHibernate framework is rather big and several approaches can be used to implement object persistence in an application. There are however a few basic principles that are appropriate for all different approaches (figure 2.1.1.1). An application contains several objects that need to be persisted (persistent objects). These objects will be send to the NHibernate framework. Each object contains meta-data to inform NHibernate on how to store the objects in the database (either with the help of an XML mapping file or declared in the source-code). The technical details on how to persist objects will be discussed later on in this chapter. The key constructs of NHibernate will be briefly explained below.

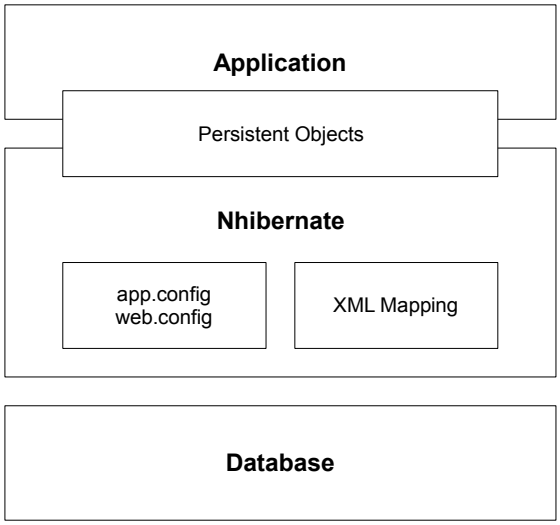

Figure 2.1.1.1: General overview of the NHibernate architecture

Sessions

Within the NHibernate framework, sessions are needed to persist objects. A session is a “unit of work”. Multiple DAO (Data Access Object) operations can be executed in a single session. *Data Access Object (DAO) is a software component that provides a common interface between the application and one or more data storage devices, such as a database or file* [W16]. The purpose of a DAO is to uncouple data logic from an application and to add an abstraction to the application so that maintenance of data access logic will be easier to perform [W16].

To begin a unit of work we open a new session and we end a unit of work by closing the session. Usually we will also flush the session at the end of a unit of work so that the SQL DML statements (Data Manipulation Language statements e.g. update, insert and delete) will be executed. The in-memory session state will then be synchronized with the database. It is possible to execute SQL queries at the session instance or through lazy loading (the data of a property will be loaded at run-time whenever a property of the object instance is being called).

A session can be seen as a gateway to the database, a map of managed entity instances that are automatically dirty checked (to see whether the object instance has been modified), and a queue of SQL DML statements that are created and flushed by NHibernate automatically [W11]. It is also possible to cache entity instances within a session (this will be explained further on in this chapter). The ISession interface is the bridge between the persistence method and the implementation.

Transactions

A transaction is a grouping of units of work. Transactions are being used when multiple objects related to eachother need to be saved (e.g. when you save a parent and a child, it is important that both parent and child are either saved or not saved at all in order to keep the data in the database consistent). If a single object fails to persist, the whole transaction will fail and the transaction will be reverted. Transactions are perfect for complex data structures (either all the objects within the transaction will be saved or nothing will be saved). Normally a transaction is started by calling the BeginTransaction method on the ISession interface. A single NHibernate session can be associated with one or more transactions. An example of a transaction is displayed in figure 2.1.1.2.

```
ISession session = sessionFactory.OpenSession();
ITransaction transaction = session.BeginTransaction();

try {
    session.SaveOrUpdate(@object);
    transaction.Commit();
}
catch {
    transaction.Rollback();
    throw;
}
finally {
    session.Close();
}
```

Figure 2.1.1.2: Example code of using NHibernate sessions and transactions

Data Mappings

NHibernate needs to know how objects in code are stored in the database and makes use of a mapping mechanism to realize this. There are two kinds of mapping mechanisms; XML mapping files and attribute mappings. The XML mapping files (figure 2.1.1.3) make it possible to separate the application's code from the meta-data that is needed to persist the objects (transparent persistence). This way you don't have to pollute the project's domain objects with meta-data.

```
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2">
  <class name="Domain.User, Domain" table="Users">
    <id name="Id" column="Id" type="integer">
      <generator class="native" />
    </id>
  </class>
</hibernate-mapping>
```

Figure 2.1.1.3: Example of NHibernate mapping file

Objects can also be mapped by making use of .NET attributes (figure 2.1.1.4). These attributes will be reflected upon at run-time to determine where objects need to be stored. There is support in the NHibernate framework to generate XML mapping files from code with an assembly as input. The attributes in the assembly will be retrieved with the use of reflection so that the mapping files can be generated. The attributes are tightly coupled to the code. When data related maintenance is being carried out or when changes occur in the application's specifications, these attributes will not easily be overlooked.

```
[Class(Table = "Users")]
public class TestUser {
    private int id;
```

```
    [Id(Name = "Id")]
    [Generator(1, Class = "native")]
    public virtual int Id {
        get { return id; }
        set { id = value; }
    }
}
```

Figure 2.1.1.4: Example code decorated with NHibernate attributes

Configuration

NHibernate makes use of a configuration file which stores all the needed settings to initialize the framework. Mainly connection settings can be found here but transactions, caching and other parts of the NHibernate framework can also be configured here (figure 2.1.1.5).

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  ...
  <nhibernate>
    <add key="hibernate.connection.provider"
         value="NHibernate.Connection.DriverConnectionProvider" />
    <add key="hibernate.dialect" value="NHibernate.Dialect.MsSql2000Dialect" />
    <add key="hibernate.connection.driver_class" value="SqlClientDriver" />
    <add key="hibernate.connection.connection_string" value="..." />
    <add key="hibernate.connection.isolation" value="ReadCommitted" />
  </nhibernate>
</configuration>
```

Figure 2.1.1.5: Part of a NHibernate configuration file

The configuration settings can be included in the application's app.config or web.config (the default mechanism to store settings in .NET Windows and Web applications) but can also be stored in a custom named XML file. Besides the application configuration there is a configuration class that will load all the XML mappings into memory. It is possible to embed the XML mapping files into the project's EXE (executable) or DLL (Dynamic Link Library) file for easy distribution. There are several ways to load the XML mappings. When an assembly is added to the configuration instance, the embedded XML mappings will be retrieved with the use of reflection. It is also possible to add the XML mappings programmatically or they can be specified in the NHibernate configuration file.

The last step in the cycle is to build the SessionFactory. This class is responsible for creating sessions. Usually an application has a single SessionFactory (when working with multiple database more are required). The SessionFactory is controlled by configuration settings which read at run-time. The SessionFactory will parse the XML mapping files to make sure that the files are correctly constructed (figure 2.1.1.6). This is a time and resources consuming process and should preferably be done only once within the application's life-cycle.

```
ISessionFactory sessionFactory;
Configuration cfg;

cfg = new Configuration();
cfg.AddAssembly(Assembly.GetExecutingAssembly());
sessionFactory = cfg.BuildSessionFactory();
```

Figure 2.1.1.6: Example of NHibernate configuration code

Cache

NHibernate has the ability to cache entities at two different levels. There is a first-level cache which caches all the entities at the session level. Each session has its own cache. When a session is destroyed the cached entities are destroyed as well. If we want to make use of an application wide cache we need to make use of the second-level cache. The entities will be cached at the SessionFactory level and are shared with all the sessions created by the SessionFactory instance.

NHibernate makes use of a pluggable caching system, making it possible to use different caching providers (each having its own implementation). Each provider has its own configuration settings. Currently there are two caching providers: Prevalence and SysCache. The Prevalence caching-provider has only a single configuration parameter: prevalenceBase (the directory of the filesystem where the caching-providers stores its data). The SysCache provider has several configuration parameters and these can be set for different regions (figure 2.1.1.7). The expiration is set in seconds and the priority is a numeric value (1-5, where 1 is the lowest and 5 is

the highest priority).

```
<configuration>
  <configSections>
    <section name="syscache"
             type="NHibernate.Caches.SysCache.SysCacheSectionHandler,NHibernate.Caches.SysCache" />
  </configSections>

  <syscache>
    <cache region="articles" expiration="500" priority="4" />
    <cache region="users" expiration="300" priority="3" />
  </syscache>
</configuration>
```

Figure 2.1.1.7: Example configuration of the SysCache caching-provider

Hibernate Query Language (HQL)
HQL is a query language which resembles SQL but contains fully object-oriented notations like inheritance, polymorphism and associations [W18]. The query in figure 2.1.1.8 retrieves all the cats of the type DomesticCat, where their name is between 'A' and 'B'.

```
from DomesticCat cat where cat.Name between 'A' and 'B'
```

Figure 2.1.1.8: Example of an HQL query

2.1.2 Session management

There are several NHibernate patterns and anti-patterns to handle session management, each having their own scope and purpose.

Session-per-operation
The session-per-operation pattern is an anti-pattern and creates a new session for each DAO operation (figure 2.1.2.1). This is a bad method because the performance will decrease significantly. *There are extremely rare exceptions when session-per-operation might be appropriate, you will not encounter these if you are just learning Hibernate* [W11].

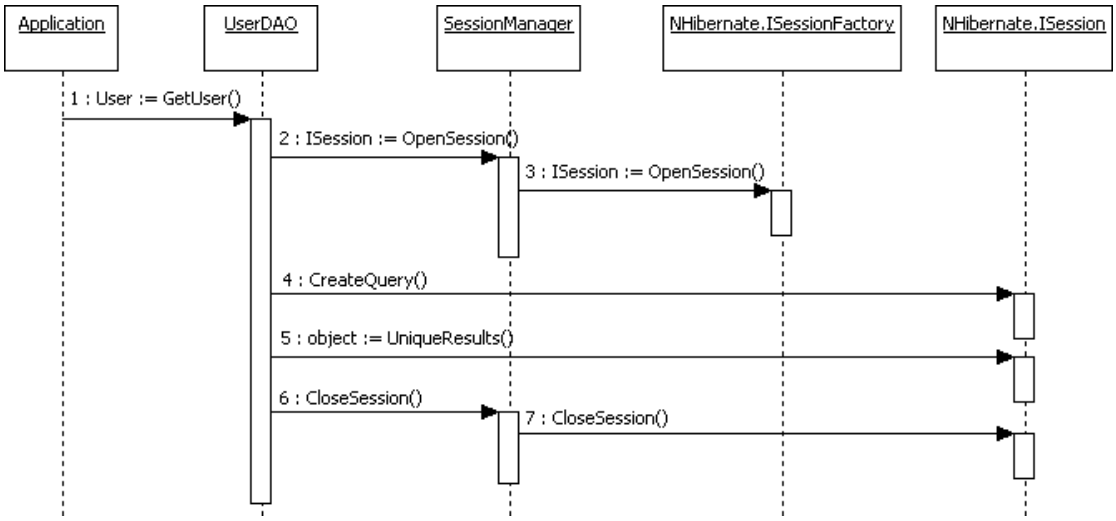

Figure 2.1.2.1: Session-per-operation pattern

Session-per-request
The session-per-request pattern creates a new session for each web-request (figure 2.1.2.2). All the possible DAO operations of a single request will be executed within a single session. The session-per-request pattern is the recommended session management pattern for most applications.

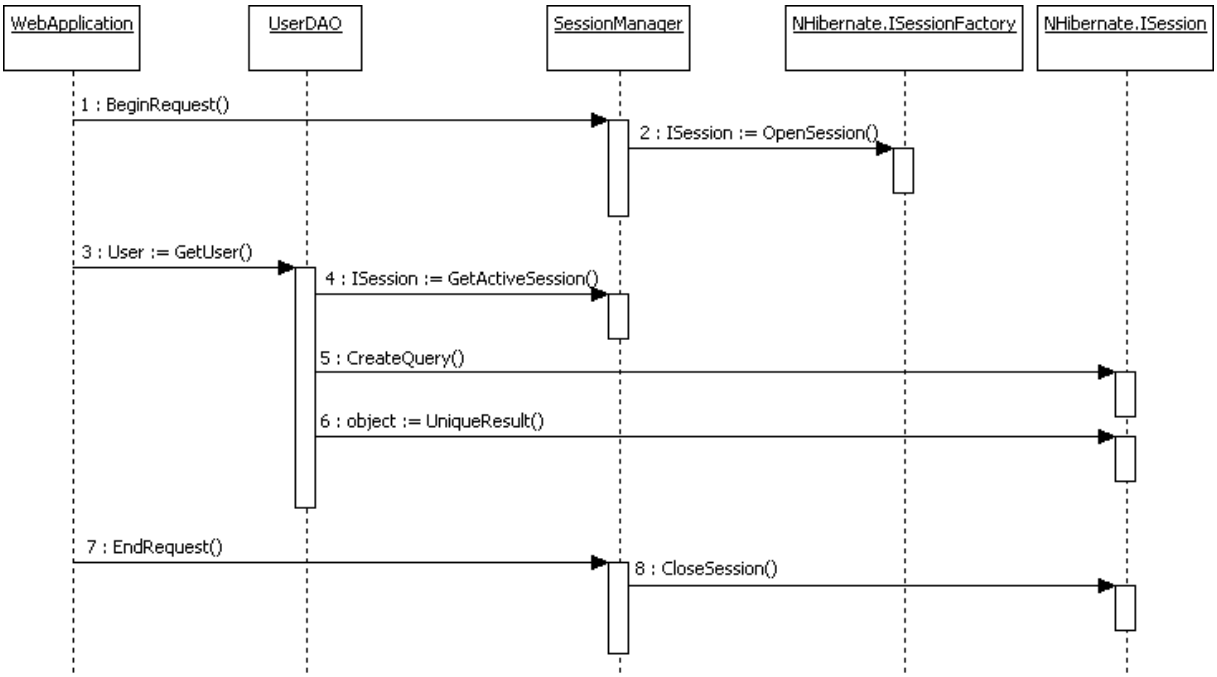

Figure 2.1.2.2: Session-per-request pattern

Session-per-request-with-detached-objects

A common issue with Web applications is that rendering is done after the data has been retrieved. The active session has been closed but when making use of lazy loading an active open session is required. We can attach the objects to a new session when they need to be rendered.

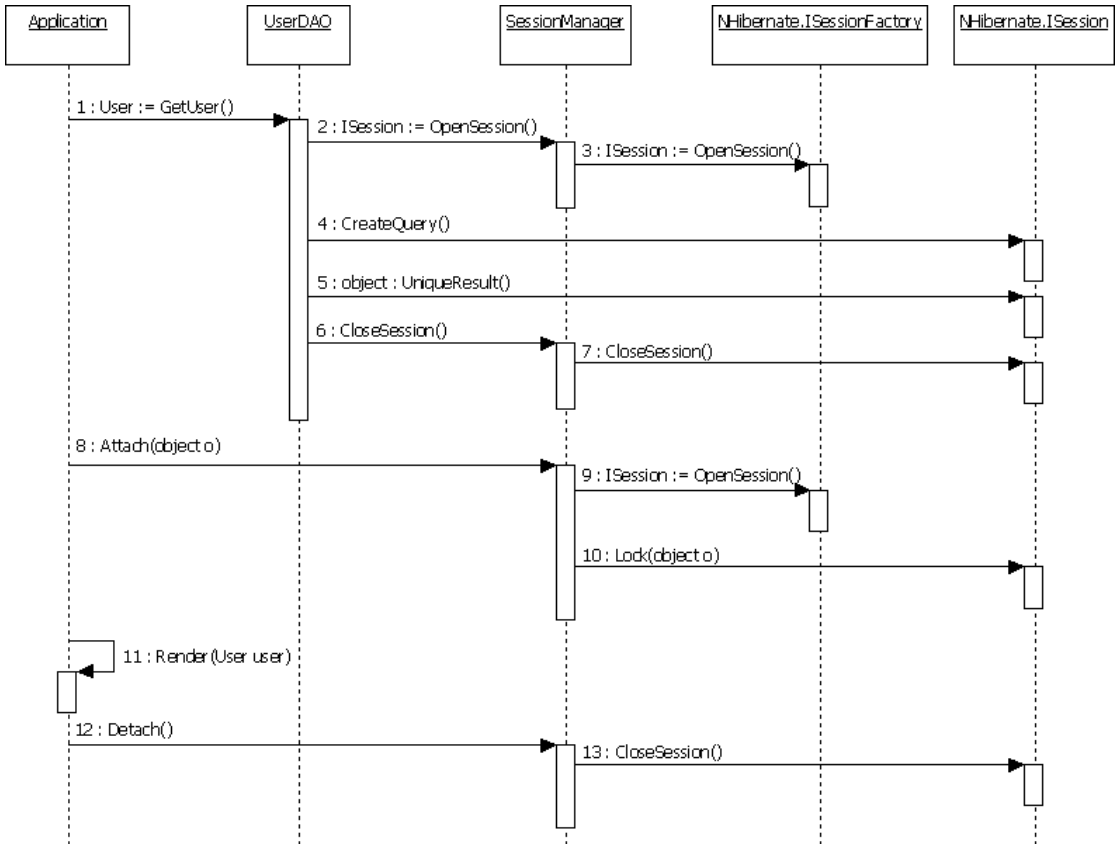

Figure 2.1.2.3: Session-per-request-with-detached objects

Session-per-conversation
In this pattern *a single session has a bigger scope than a single database transaction and it might span several database transactions* [W11] (this does not mean that the session should live as long as the user-session). If we purchase an item at an online store, we would probably need to walk through several steps before the item has been purchased. It is likely that two conversations need to be created: registering the user (if the user hasn't been registered yet) and purchasing the item. A conversation for registering a user might look look like this:
- Choose a user account and assign it to an e-mail address (both should be unique)
- Fill in personal data (surname, last name, city, etc.)
- Confirm the creation of a new account

We don't want to save the data for every single step and a new account should only be created once all these steps have successfully been completed.

Session-per-application
In this pattern a single session is used within the entire application's life cycle. This pattern is mainly used for desktop applications (client / server) and will be omitted because it falls out of the scope of this thesis.

Chapter 3

Problems when working with NHibernate

The purpose of this master thesis is to determine the most important problem related to the NHibernate framework and to see whether automation is a feasible solution. How are we going to determine the NHibernate problems and how do we acquire this information as effective as possible? How do we do this consistently so that we can actually analyse the results?

3.1 Roadmap

We need to determine whether it is possible to automate the most important problem or an aspect of the most important problem related to working with the NHibernate framework. We need to determine which problems people have with the NHibernate framework. This information is needed in order to continue research and we have to determine how we are going to retrieve this information. There are several ways to acquire the information and are discussed in paragraph 3.2.1. We will consult the literature on these information acquisition methods and we will discuss their pros and cons. We will pick the best information acquisition method based on criteria we will define in paragraph 3.2 and we will then use this methods to retrieve the needed information from our participants. The results of this method will be analysed to determine the most important problem related to working with NHibernate and we will identify two possible solutions to the problem. These possible solutions are problem specific and cannot be determined upfront. We will compare both solutions to each other and if possible we will try to measure the effect of the solutions on the development process. Has the development process become more effective / efficient?

3.2 Information acquisition

How do we acquire information without creating all kinds of malformed results, making it impossible to discern the most important NHibernate problem? There are many issues when gathering information and cognitive factors play a major role. *Participants are subject to the limitations of their own memory and communication abilities. Studies in cognitive science show that memories can be lost, distorted or blocked* [2]. The literature can save us from common pitfalls and several issues have been identified and have been listed in table 3.2.1.

Issue	Explanation
Response bias	A respondent might not answer questions honestly because *they may not feel comfortable relating personal feelings or experiences* [2].
Non-response bias	There is a difference between the people responding at the e-mail and the non-responders. *Late responders answer differently than early responders, and that the differences may be due to the different levels of interest in the subject matter* [W2].
Selection bias	Creates a distorted view because the way how the information is acquired. *Selection bias can be the result of scientific fraud which manipulate data directly, but more often is either unconscious or due to biases in the instruments used for observation* [W5]. Selection bias occurs when the sample group you have chosen is not representative of the population you want to generalise your results to [W2].
Frame population	Frame population *refers to the units that can actually be reached* [3]. We will cause selection bias if the units being reached are not representative for the population of the domain.

Table 3.2.1: Problems regarding information acquisition

The impact of above issues should be reduced or avoided where possible. There are several solutions for these problems and are mentioned in table 3.2.2.

Issue	Possible Solution
Response bias	We can make use of hypothetical experience questions and create persona's and scenarios rather than directly asking participants questions. *The hypothetical nature of the scenario can help reduce pressures that the participants may feel to please the interviewers. Participants may not want to risk offending the researchers by relating negative comments* [2].
Non-response bias	Is it necessary to make a difference between responders and non-responders? If we only analyse the results of the responders, we might cause selection bias. If we do analyse the results we might also cause selection bias. For the sake of simplicity I will assume that non-responders may be less interested in the subject but do not distort the results.
Selection bias	It is very well possible that the people responding are not representative for the NHibernate community. The difficulty here is to specify and define criteria to identify a representative NHibernate community member. We can reduce the effects by using the information acquisition method for several different community related spots (forums, mailing lists, etc.).
Frame population	Frame population is closely related to selection bias. Selection bias is a result of frame population. Therefore if we can reduce the impact of frame population, we will also reduce the impact of selection bias. By mixing several information acquisition methods we are not restricted to a single user-group (e-mail users, forum users, etc.) and we create a broader sample group. We might reduce or avoid the impact of frame population this way.

Table 3.2.2: Possible solutions to reduce the impact of obfuscated results

We need to determine the criteria that are important for our information acquisition method. There is only a limited amount of time available for this thesis and with this in mind I have defined several criteria (table 3.2.3). With the help of these specified criteria we can more accurately pick the best method for our case.

Criteria	Explanation
Broad range of users	The expertise level of the users must vary from beginners to expert users. It's not representative to have only beginner users or only expert users. A broad range of users can also reduce the impact of selection bias and frame population.
Accessible	The user should be convenient to answer the questions and shouldn't take too much of the participants time. The questions should also be understandable. If the user doesn't understand the questions he will not be able to answer them and it might make the participant feel inconvenient.
Obtainable	The information should preferably be easily obtainable. The easier it is to obtain the results the more users can be part of the information acquisition method, creating a more objective view of the community.
Processable	The questions should be consistent and quantitative. This means that the questions should be concrete and non-ambiguous. The answers have to be predefined or expressed numerically. A common mistake is to only allow numeric or predefined answers causing response bias; The user will be pushed into a specific direction.

Table 3.2.3: Criteria for the information acquisition method

3.2.1 Information Acquisition Methods

Let's have a look at different kind of methods to acquire the necessary information. In paragraph 3.2.2 these methods will be compared to our defined criteria.

Interview

Interviews are done by an interviewer to obtain information from the interviewee. There are 2 types of interviews: Interviews of assessment and interviews for information [W3]. In our case we're interviewing for information to understand the problems of the interviewee.

Interviewing is a good technique for small groups but *interviews are notoriously problematic, and may yield insufficient, irrelevant and / or erroneous data. The interview settings can create biases, communication problems, and social issues* [2]. *To cope with the problems that arise from these communication factors, cognitive factors, and social factors, researchers offer various guidelines, techniques, and methodologies* [2].

Survey
A survey is *a gathering of a sample of data or opinions considered to be representative of a whole* [W14]. A survey can be performed in several ways (web-based, piece of paper, e-mail, interview, etc.). A survey is a great way to get response from a broad audience but this depends in which way the survey is performed. There is a risk when doing surveys. You have to understand what you want to ask. If it is not clear what you want to accomplish, a survey is useless. You also need to know what you want to do with the results. If the results are non-quantitative it'll be very difficult to analyse the results. There are more issues that can arise, e.g. non-response. Another difficulty is that you don't know when you can analyse the results. It might be possible that you miss valuable respondents. You can't wait forever so at a certain point you need to decide when to analyse the results. If we take a look at extrapolation methods, we can see that *they are based on the assumption that subjects who respond less readily are more like non-respondents* [4], making it only important to catch the first wave of respondents. It'll still be difficult to determine who is a first wave respondent and who is not. If we send reminders to our participants, will this distort the results or are these participants considered non-respondents?

Data mining
Instead of relying on participants we can use existing information that is available in public. Browsing the Web for example is a good way to identify problems and requests (e.g. find forum topics with lot of responses). The identity of a person on the Internet cannot easily be retrieved because of the anonymous characteristics of the Web. It is very well possible that a specific person posts on several forums and other community resources, making it believe that a certain issue or request is actively being discussed among the community. Besides anonymity there are also other difficulties that need to be taken into account. Off-topic discussions can mark a topic as important while in fact it isn't. Finding problems can be a tedious task this way.

Become an expert
I can become my own NHibernate expert and use myself as a reference. A big risk here is that it is hard to decide when you have become an expert. What makes one an expert? There is also a chance that too much time is needed to become familiar with NHibernate leaving no space for analysis and working at a proof of concept. Being your own expert stimulates subjectivity and is thus not advisable when you need to represent a group of stakeholders (NHibernate community members).

3.2.2 Picking the right method
Which of the mentioned information acquisition methods is best for our case? Several issues and solutions have been identified for specific problems. We can estimate the influence of the information acquisition methods on the criteria specified earlier (table 3.2.2.1).

	Broad range of users	Accessible	Obtainable	Processable	Total
Interview	--	-	-	-	-5
Survey	++	-	+	+	+4
Data mining	++	+	--	-	0
Become an expert	--	+	--	--	-5

Table 3.2.2.1: Rating estimates of several information acquisition methods

These ratings may be off and depend on the way how a specific information acquisition method is carried out. If we use hypothetical questions in an interview setting it is likely that the accessibility rating will increase. Table 3.2.2.1 gives a global indication which method is best in our case for our defined criteria.

3.3 Survey

Based on our criteria a survey is the best way to acquire the needed information and our first sub-question has been answered: How do we find the problems related to working with NHibernate? So how are we going to conduct this survey?

We need to validate whether our respondents are representative for the population of our domain and several personal questions have to be asked. If the respondents are representative we need to analyse the results to determine which problem is the most important problem.

What is the definition of the most important problem? *A problem is an obstacle which makes it difficult to achieve a desired goal, objective or purpose. It refers to a situation, condition, or issue that is yet unresolved. In a broad sense, a problem exists when an individual becomes aware of a significant difference between what actually is and what is desired* [W19]. A problem is always subjective. The most important problem will always refer to the person's personal view. In my opinion the most important problem refers to a problem that is shared by a majority of people. The third sub-question has now been answered: What is the definition of the most important problem?

How do we know what the most important problem is? We will consider beginner and expert users as equal. Problems provided by beginners and experts will therefore be treated equally. We will make use of the same principle for small and large companies. We will group possible related answers and count the number of times they occur in the results of the respondents. The problem that occurs the most is our most important problem. The second sub-question has now been answered: How do we determine which problem is the most important problem?

The first thing we have to do when we are going to analyse the results is to determine whether the respondents are indeed a representative population of our domain. We will define evaluation criteria (table 3.3.1) and we will then test the results of the survey. The evaluation criteria are estimates and have not scientifically been measured. Table 3.3.1 displays that the respondents are representative for our domain and further research is possible.

Criteria	Target percentage	Measured percentage
At least 70% of the respondents should have an age between 20 and 45.	70% - 100%	100%
At least 80% of the respondents should be highly educated (Bachelor, Master, etc.).	80% - 100%	100%
At least 70% of the programmers have 4 or more years of programming experience.	70% - 100%	75%
At least 50% of the programmers should have 2 or more years of object oriented design.	50 - 100%	100%

Table 3.3.1: Criteria estimates and evaluation results to check the validity of the respondents

The survey has been filled in by five employees of Sogyo. We haven't received any input from the NHibernate developer mailing-list. There is a chance that the results of the survey are not objective. We have a small sample group and the respondents are all from Sogyo (both can bias the results). It's possible that the results of the sample group do not reflect the real problem of “working with NHibernate”. Despite these shortcomings the survey does reflect an important issue (table 3.3.2): Working with sessions. Since sessions and transactions are closely related I will try to find a solution to either abstract sessions and transactions away from the developer or to decrease the difficulty of both sessions and transactions. The complete survey is available in appendix A.

Problem	Percentage
Session related issues (e.g. managing sessions)	40%
Creating XML mappings is considered a repetitive job	20%
Configuration is considered a repetitive job	20%
Lazy loading in a Windows Communication Foundation environment	10%
Cascade insert using assigned indexes	10%

Table 3.3.2: NHibernate problem list

Chapter 4

Transparent Sessions and Transactions

The developer has session related issues and it is important to understand these difficulties. Is it possible to create an abstraction for sessions and transactions so that the developer doesn't have to understand the concepts behind it? We will make sessions and transactions transparent to the developer and this might result in a more efficient development process since the difficulty of sessions will be reduced (or in the most optimal case the problems are no longer there).

The first thing we have to determine is why sessions and transactions are problematic and difficult to understand. If this is clear we can analyse two common approaches to possibly solve the problem. We will analyse the differences between a Domain Specific Language and framework approach. It will be interesting to see which solution is best for the problem domain.

4.1 Difficulties

There are several difficulties when people are working with sessions and transactions. We will identify these problems and explain why people have problems with them.

4.1.1 Managing sessions in a stateless environment

Web applications are stateless; Every request will be handled independently. Most Web applications today need to maintain a state to store, for example user data. NHibernate makes use of sessions to store data. A session factory is needed to parse the XML mapping files. Creating a session factory is expensive in terms of resources and time. A single instance of the session factory can be used to serve all users. We can store the instance of the session factory in the AppDomain (figure 4.1.1.1) of the application (a representation of the application domain and *helps providing isolation, unloading and security boundaries for executing managed code* [W13]).

```
protected void Application_Start(Object sender, EventArgs e) {
    ISessionFactory sessionFactory;
    // Instanciate and configurate the session factory
    ...
    AppDomain.CurrentDomain.SetData("sessionfactory", sessionFactory);
}
```

Figure 4.1.1.1: Storage of the application wide variable "sessionFactory" in the AppDomain

The NHibernate session can't be shared among all the users. The NHibernate session object is not thread-safe and when multiple users access a single session object it's just a matter of waiting before threading issues will occur. Within the .NET framework there are several constructions that can be used to store user data, e.g. ThreadStatic, CallContext and HttpContext. Let's determine which method is best and why it is best for our case. Where possible, a test-case has been created to analyse the issues that might arise with each method. The default behaviour of the ASP.NET thread pool has been modified to create a bigger chance of threading issues (table 4.1.1.1).

Setting	Value	Explanation
maxWorkerThreads	2	The maxWorkerThreads setting has been lowered from the default value of 20 to 2. A maximum of 2 threads will be handling requests. The chance that threading issues will occur will significantly increase.
minWorkerThreads	2	The minWorkerThreads setting has been lowered from the default value of 20 to 2. A minimum of 2 threads are handling requests. This ensures that threading issues can occur, even with a low server load.

Table 4.1.1.1: Modified settings in the web.config

The test-case consists of three constructions. A fast page, a slow page and a storage class. The storage class is implemented in a specific way and differs per case. Both pages will display and set a value within the storage class (figure 4.1.1.2). When the value has been set, the value will be displayed again. The fast page will do this immediately while the slow page displays the value after five seconds. This time-out has been inserted in order to determine if the value has been changed by a different thread (also known as race conditions where two or more threads are 'racing' each other to influence the output first [W26]).

```
public partial class Slow : System.Web.UI.Page {
    protected void Page_Load(object sender, EventArgs e) {
        Debug.WriteLine(String.Format("-Slow- Current value: \"{0}\"", Storage.Name));
        Debug.Flush();
        Storage.Name = "Slow";
        Debug.WriteLine(String.Format("-Slow- After set: \"{0}\"", Storage.Name));
        Debug.Flush();
        Thread.Sleep(5000);
        Debug.WriteLine(String.Format("-Slow- After timeout: \"{0}\"", Storage.Name));
        Debug.Flush();
    }
}

public partial class Fast : System.Web.UI.Page {
    protected void Page_Load(object sender, EventArgs e) {
        Debug.WriteLine(String.Format("-Fast- Current value: \"{0}\"", Storage.Name));
        Debug.Flush();
        Storage.Name = "Fast";
        Debug.WriteLine(String.Format("-Fast- Value: \"{0}\"", Storage.Name));
        Debug.Flush();
    }
}
```

Figure 4.1.1.2: Test case to determine threading issues

ThreadStatic

With the ThreadStatic attribute, members of a class can be marked as static within a thread. These members act like normal static members except that they have per-thread storage. We can save user data per thread rather than for the AppDomain (making it accessible for other users). The problem however is that we have no influence on the life-cycle of threads within ASP.NET. It is very well possible that the next request comes from a different thread and we'd be getting someone else's user data. *In ASP.NET your code is run on a WorkerThread from the 25 or so threads in the default ASP.NET worker thread pool and the variable that you think is "personal private to your thread" is personal private...to you and every other request that this worker thread has been with. Under load you may well find your variable modified* [W6]. A single request can be handled by one or more threads and ThreadStatic is therefore only useful if we control the ThreadPool and the life-cycle of the threads. This approach is not scalable because we would need as many threads as there are users. The behaviour has been validated with a test-case (figure 4.1.1.3 displays the implementation of the storage class).

```
public class Storage {
    [ThreadStatic]
    private static string name = "";

    public static string Name {
        get { return name; }
        set { name = value; }
    }
}
```

Figure 4.1.1.3: Example storage with the use of the ThreadStatic attribute

If we load a fast page and a few seconds later a slow page, it is possible that both pages are handled by the same thread (figure 4.1.1.4).

```
-Fast- Current value: ""
-Fast- Value: "Fast"
-Slow- Current value: "Fast"   <-- Is handled by the same thread
-Slow- After set: "Slow"
-Slow- After timeout: "Slow"
```

Figure 4.1.1.4: Console output of the ThreadStatic test-case

CallContext

CallContext is a specialized collection object for method calls and provides data slots that are unique to each logical thread of execution. The slots are not shared across call contexts on other logical threads. Objects can be added to the CallContext as it travels down and back up the execution code path, and examined by various objects along the path [W15]. The CallContext class is thread-bound and its behaviour is non-deterministic. With high server loads there is a higher chance on threading issues. It is possible that a web-request is handled by multiple threads. In the worst case each event in the code is handled by a different thread (each thread having its own CallContext). It will not be possible to use the CallContext as our data storage if each thread has its own CallContext instance. We need to save data per request, not per thread. The issues have not been reproducible because of the non-deterministic behaviour.

HttpContext

The HttpContext class resides in the System.Web namespace and encapsulates all the information related to a single request. This class is bound to a request rather than to a thread. The threading issues will be handled by the .NET framework and are part of the class's implementation. This makes the HttpContext a save way to store information for Web applications. The HttpContext instance is however not available in Windows applications and in a Windows application we should use the CallContext instead. The downside of this approach is that there will be a dependency to the System.Web assembly when using Windows applications. Most Windows applications make no use of any functionality within this assembly and adding this assembly is therefore not recommended. It's a bad practice to add unused assemblies as references but since there is no other way we are forced to do so.

Chapter 5

Domain Specific Language

In this chapter I will determine whether a DSL is a possible solution for session and transaction related problems. I will focus my efforts on the generation part of a DSL. The idea is to create a pluggable code generation model for the data-layer of an application. I will make use of DSL Tools by Microsoft as my validator.

5.1 Introduction

A domain specific language (DSL) is a programming language tailored for a particular application domain. Characteristic of an effective DSL is the ability to develop complete application programs for a domain quickly and effectively [6]. A DSL is close to domain experts and supports domain specific notation and expressions and is declarative; Emphasizing "what" has to be done rather than "how" it has to be done. When compared to programming languages like Cobol or Java, you see that in general a DSL consists of fewer language constructions [19]. The main advantage of a DSL is that it describes the semantics of the domain, making it easier to validate it against the domain it describes. A DSL can either be executable or non-executable [11].

If we can make a DSL that describes the data-layer of an application, we can possibly create a more efficient development process. The data-layer consists of source code, XML mappings and optionally SQL queries. The DSL will be responsible for generating these parts. When changes in the application's specifications occur we need to be able to regenerate the complete data-layer (figure 5.1.1).

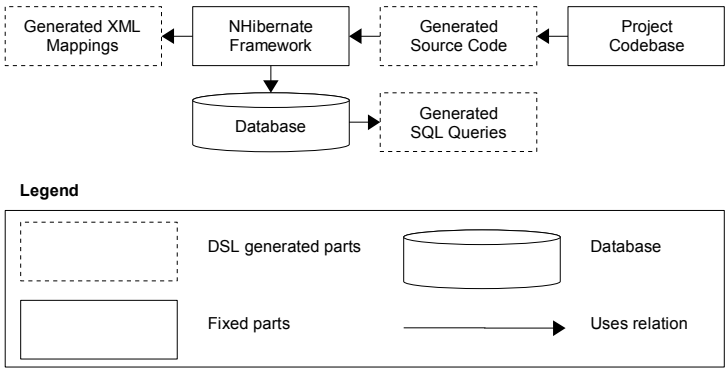

Figure 5.1.1: Output generation model

5.1.1 Advantages

The advantages differ for users and developers of a DSL are listed below:

Users

- A DSL describes the semantics of the domain making it easier to validate it against the domain it describes. *Arguably, a good DSL is at an even higher level than a conventional high-level language, and can often be used by those who are not expert programmers* [6].
- It's a lot easier to describe a domain in a DSL compared to a high-level programming language. A high-level programming language provides generic primitives and not primitives that correspond directly to the domain.
- *DSL programs are concise, self-documenting to a large extent, and can be reused for different purposes* [19].
- *DSLs allow validation and optimization at the domain level* [19].
- The user will only work with domain objects. Everything that has been defined in the DSL can be used in the domain; No more, no less. The DSL will be more easy to understand and to maintain because of this. This will also result in changes being less expensive.

Developers

- The expressive power of a DSL is enormous. It can be used to create code generators, business plans and reports generators, what-if analysis, system execution monitors, etc. [W10].
- There is no need to cover the complete problem domain. When developing a DSL we have to *strive for an 80% solution. If a large percentage (e.g. 80%) of the activities can be compressed using the DSL paradigm, the experts have plenty of time left over to deal with the hard or interesting parts* [14].

– *DSLs enhance productivity, reliability, maintainability and portability* [19].

5.1.2 Disadvantages

Users and developers can experience several disadvantages when using or developing a DSL.

Users

– The output of executable DSLs will generally be hard to understand (e.g. a code generator). When errors occur you want to know where the specification of the domain is wrong rather then digging into generated code to find the problem.
– *Potential loss of efficiency when compared with hand-coded software* [19].

Developers

– It can be very difficult to design and implement a DSL. *Developers need to establish close ties with a domain expert to produce the infrastructure that the system will be translated into* [11]. Without extensive domain knowledge it is not possible to develop a DSL that corresponds with the actual domain.
– It's difficult to find the proper scope for a DSL [19].
– One of today's problems is not lack of domain knowledge (domain experts know more than enough about the domain they are working in) but a way to accurately describe the domain.
– There is a limited availability of DSLs [19] and therefore adapting is not always possible.
– *The difficulty of balancing between domain-specificity and general-purpose programming language constructs* [19].
– When develop a new DSL it might be possible that we never break even because the start-up costs are too high (figure 5.1.2.2). This can be overcome by looking for DSL frameworks or by inheriting the infrastructure of an other language and use it as your DSL base [6].

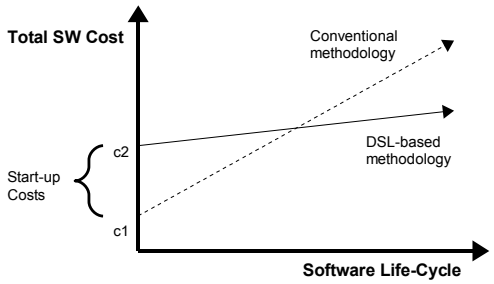

Figure 5.1.2.1: The pay-off of DSL Technology [6]

5.2 Graphical or textual

There are 2 distinct ways to visualize a DSL, either graphical or textual. What are the differences and relations between graphics and text?

– A graphical visualization is less intimidating; Non-experts can more easily work with a graphical representation. In a study of John A. Lehman about graphical and textual notations for documentation of data structures, he noticed that less experienced programmers can work more easily with graphical notations [20]. There is however *no evidence that an organization staffed with professional programmers should choose one notation over the other as an enforced standard. Professional programmers appear to be able to use both notations equally well, and there is no evidence that for this group one notation leads to better performance than the other* [20].
– *By designing visual languages that take advantage of an artist's skills in visual perception and expression, we can allow that artist to take advantage of the expressive potential that modern computing offers* [10].
– If done correctly one can understand a graphical language faster. A textual representation is flat and requires reading before understanding it. Graphics on the other hand can say a lot with just the glimpse of an eye. For example, a shape can have a meaning (rectangular, circular, etc.) or different line types (dotted, striped, solid, etc.). Even colours can have meanings. Once applied consistently this can make a language even easier to understand.
– *One could pack more information into a given space using images than by using words to describe the same information. Images can convey information that words cannot (e.g., we need to define concepts before we are able to express impressions with words)* [21].
– *Images can also deliver information more quickly and efficiently than by using words (these known facts have not gone unnoticed by advertising organizations and the like)* [21].
– *Words could be fuzzy. Images show the truth as it is* [21].
– It's difficult to represent information clearly and it depends on the visual and information perception on past memories, experiences, beliefs, culture of an individual. It's difficult to make effective use of colour and these difficulties illustrate the frailty of image representations [21].

It's important to understand that images may have some disadvantages, and words are sometimes more effective (or powerful) than pictures. Words can fail to describe images but on the other hand images can also fail to capture what can be said with words [21]. Visual comprehension of information *depends on the object, purpose, and sometimes the viewers' preferences, and cannot be expressed with theoretical definitions* [13]. In a paper of Amy Voida and Elizabeth D. Mynatt [22] a table in a book of McCloud [23] is listed (table 5.2.1). McCloud suggests that there are 7 categories of word-picture combinations. It's important to understand that text and images don't exclude each other and we would have to determine how text and images are used together in collaboration with domain experts.

Combination	Description
Word Specific	Pictures illustrate, but don't significantly add to a largely complete text
Picture Specific	Words do little more than add a soundtrack to a visually told sequence
Duo-Specific	Words and pictures send essentially the same message
Additive	Words amplify or elaborate on an image or vice versa
Parallel	Words and pictures seem to follow very different courses — without intersecting
Montage	Words are treated as integral parts of the picture
Interdependent	Words and pictures go hand in hand to convey an idea that neither could convey alone

Table 5.2.1: McCloud's seven categories of word and picture combinations [23]

If we want to create a visual DSL we have to take into account that visualizing large amounts of data is difficult. If done incorrectly the user can lose the overview easily and instead of being more productive a decrease in productivity will occur. If we do have to model large amounts of data we have to provide the user with several levels of detail. The amount of levels we have to provide depends on what and how we are going to visualize the DSL.

5.3 Create or Adapt

Do we need to create a brand new DSL or can we better adopt an existing DSL? *Adopting an existing DSL is much less expensive and requires much less expertise than developing a new one. Finding out about available DSLs may be hard, since DSL information is scattered widely and often buried* [11]. The choice depends on the budget of the project and whether an existing DSL is available to serve as our base.

It has not been easy to find a persistence DSL. It seems that there are not many DSL focused on persistence (and this is an understatement). The only DSL I have found that does something related to object persistence is ActiveWriter [W24], developed by Gokhan Altinoren. The user models the domain objects and the class files and NHibernate XML mapping files will be automatically generated. The object's attributes and properties will be automatically decorated with ActiveRecord attributes. ActiveRecord is a persistence framework, developed by Castle Project (an *implementation of the ActiveRecord pattern for .NET* [W25]). A drawback of using ActiveRecord is that domain objects need to contain meta-data. We will not make use of ActiveWriter therefore.

5.4 Success Factors

There are several important factors for success. Some of these come from the literature and some have been discussed with Sogyo:

- We have to s*trive for an 80% solution* [11].
- An other important factor is to *establish close ties with a domain expert to produce the infrastructure that the system will be translated into* [11]. Without extensive domain knowledge a DSL-based solution is guaranteed to fail.
- It should be possible to maintain the data model with the DSL. We need to be able to regenerate the source code, XML mapping files and optionally SQL queries without the need of changing the current application's source code other than overwriting the existing generated parts.
- The DSL should make it easier to create and maintain the data model. The visual aspect of the DSL can make it possible to give a better insight in the data model of a system (at the source code and at database mapping level).
- The DSL needs to give insight in the objects that are contained within the data model and the internal data of these objects. It is possible that not all the data of an object needs to be persisted. The visualization should clearly show how the objects and the internal data are being mapped to the database.

5.5 Modelling standards

I couldn't find any DSL based on visualizations of object persistence and thus we cannot adapt from an existing solution. We have to look for a different solution. There are many modelling standards: UML (Unified Modelling Language), ORM (Object Role Modelling), ERD (Entity-relationship diagram), etc. Is it possible to make use of the notations of one of these modelling standards? Which modelling standard is the most close to our domain? See appendix D for an example of the above mentioned modelling standards.

We have to determine whether we are going to build a DSL based on a code or data perspective. ORM and ERD diagrams are examples of a data perspective while a class diagram focuses on the code aspect of an application. We will make use of the notations of a class diagram because an example project of a class diagram is already available in the DSL Tools framework by Microsoft. I have no experience with DSL Tools and it might be very difficult to create a DSL from scratch. If we use the class diagram example as our base, we might be able to extend it easily with new notations to generate the persistence services. The advantage of making use of the notations of a class diagram is the flexibility a developer has when developing a data-layer. The whole application architecture and the persistence aspect can be modelled in a single diagram. We have to keep in mind that the model must not become too complex or explodes with all kinds of notations.

5.6 Difficulties

When working with DSLs it is difficult to separate the generated code from the implementation code. From a theoretical point of view it's not desirable to mix generated code with implementation code. From a more practical point of view it's very likely that implementation code needs to be added to the generated code. The difficulty will be to keep the implementation code intact when you regenerate the data-layer. There are several approaches to make this possible, each having their own advantages and disadvantages.

5.6.1 Generate once

We can generate the needed code once and modifications need to be applied to the generated code. This approach is impractical. If we add new functionality to the code generator, we will not be able to make use of these changes in earlier projects with generated parts in their code base. Even worse, if we fix bugs in later releases of the code generator, we'd have to manually modify all the projects that have generated aspects of earlier code generators.

5.6.2 Generating highly customizable objects

When generating source-code there is always a chance that we need to add custom code in order to make it work the way you want to. There are two possible solutions here: We can extend the generated source-code with all kinds of handlers or other mechanisms to allow the programmer to add custom behaviour at design-time, or we can only generate static parts or parts that will most likely not change. From a theoretical point of view the last option is preferred.

It's not recommended to generate highly customizable objects because we will need to create a highly extensible architecture. The drawback is that a highly extensible architecture in general is difficult to understand. The developer will be exposed to all kinds of constructs making it hard to see the overall picture. To understand the difficulties that will arise we will go through several possibilities of an extensible architecture, each having their own drawback(s). Before we'll dive into the several options that are available, we will first look at an important feature of .NET 2.0 for code generation.

Partial classes

The C# language has introduced partial classes in the .NET framework version 2.0. Partial classes are a mechanism to separate the code of a single class in one or more files. The files can be located in any sub-directory of the project's root directory as long as the classes are having the same namespace declaration.

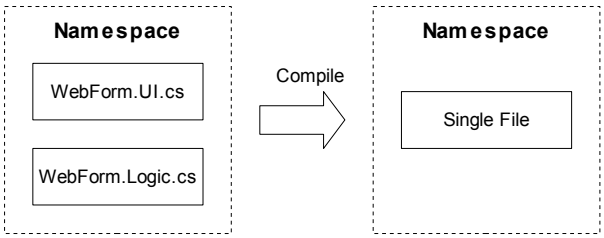

Figure 5.6.2.1: .NET framework 2.0 partial classes

Microsoft uses partial classes to separate generated UI code from implementation code. We can use the same principles. We can store the class's data in a generated partial class while the implementation code can be located in a different file. We can regenerate the data whenever the model changes without losing our changes in the implementation partial class. Problems will occur when we want to model more than just data.

Declaring and implementing partial methods

If we want to model methods within an object we cannot simply make use of two partial classes (figure 5.6.2.2).

```
namespace Nhibernate.Helper {
    public partial class Object {
        public void Operation1() {}
    }

    public partial class Object {
        public void Operation1() {}
    }
}
```

Figure 5.6.2.2: Two partial classes with the same declared operation

It is not possible to implement the method of a partial class if the method has already been declared in a different partial class (figure 5.6.2.3).

```
Error  1 - Type 'Object' already defines a member called 'Operation1' with the same parameter types
```

Figure 5.6.2.3: Compiler error when declaring and implementing a single method in multiple partial classes

Unused model elements

A different solution (figure 5.6.2.4) would be to generate the attributes of an object in a partial class. The public methods of the modelled object can be generated in an interface. We can implement this interface in a partial class.

The problem with this approach is that private and protected operations cannot be put in an interface. We cannot make use of properties either because of this same issue (*properties are members that provide a flexible mechanism to read, write, or compute the values of private fields* [W8]). We'd have to manually declare and implement these methods and properties in a partial class of the object. The downside of this approach is that our model now contains several modelled elements that are not visible in the output of the DSL. It's of no use to model specific things (private / protected operations and properties) if they are not visible in the output.

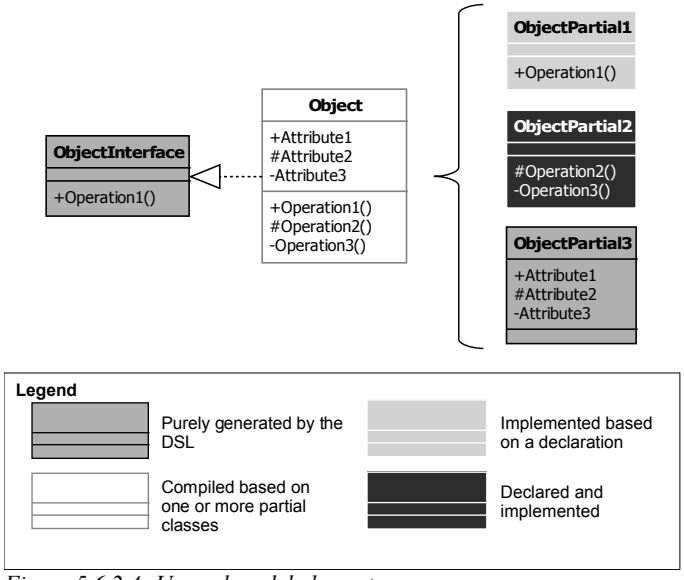

Figure 5.6.2.4: Unused model elements

Accessibility

A different solution would be to make use of inheritance (figure 5.6.2.5). There are several issues with this approach. If the attributes of a business object are generated into a super-class, the private attributes are not accessible in the sub-class. This same issue applies to private operations and properties. These have to be manually declared and implemented in a sub-class. There are more difficulties with inheritance. We cannot just generate a public and protected operations. These will not be implementable in the sub-class. We either have to mark operations as abstract or we have to add a virtual modifier to the operations so they can be overridden in a subclass.

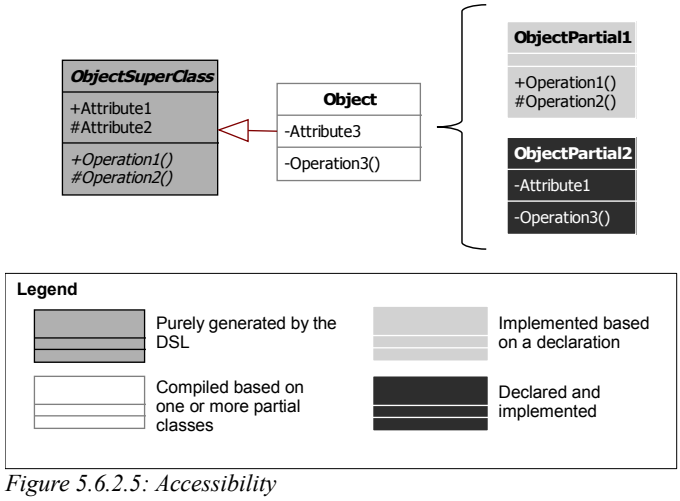

Figure 5.6.2.5: Accessibility

5.7 Future Work

The initial thoughts were to create a class-like notation for a mapping file (figure 5.7.1). The drawback of this approach is that things sooner or later become unclear and the developer will lose the overview of the data-layer. For each persisted class a mapping notation needs to be visible. A different solution would be to apply a thick border (preferably of a specific colour) to a class to mark the class as a persisted class. A developer can instantly see the objects that have been mapped to the database. The downside of this approach is that we only see 'which' objects are persisted but not 'how' they are persisted. We can make use of the Visual Studio properties window to make this visible for the developer but then a user has to select a class before he can see how the objects are persisted. This requires the user to perform an action for every class he wants to view and for medium to huge data-layers this will soon become annoying and tiresome.

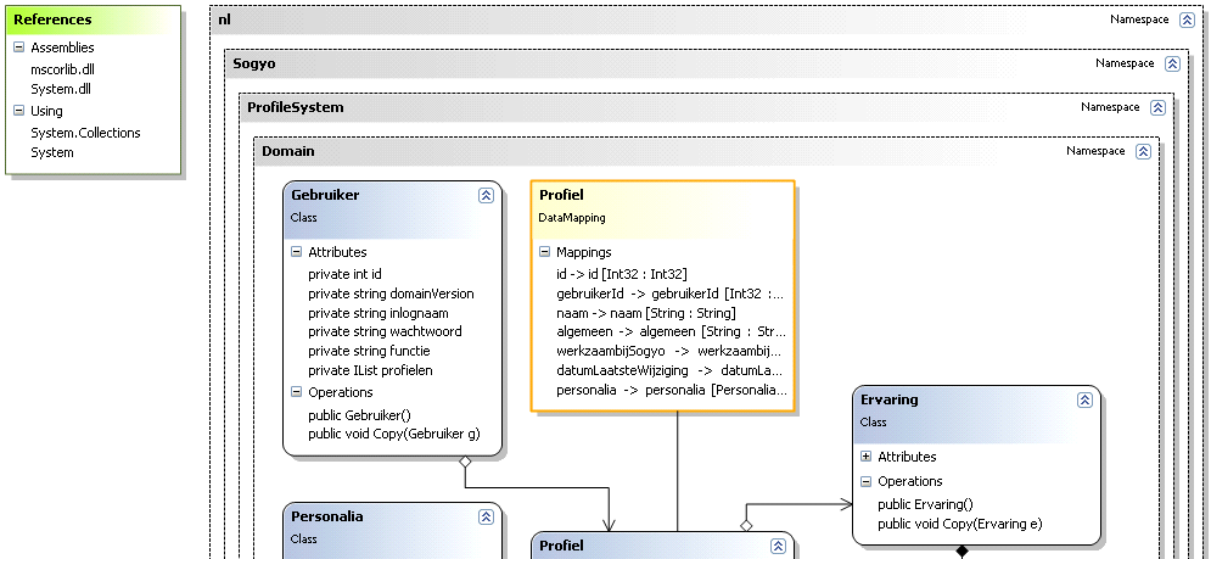

Figure 5.7.1: Example of an extension of class diagram notations

It is important to determine what has to be shown all the time and what can be visible when actions are performed at the model. It might be possible that it is not really necessary to always see 'how' objects are persisted. It is important to further analyse the domain of the DSL before a satisfying answer can be given. If the domain of the DSL is clear we can get a better understanding of how things need to be visualized.

5.7.1 Validation

Validating the DSL is an important aspect of the research. The user needs to understand the DSL. If the domain expert understands the DSL, the right visualization has probably been used since a good DSL can be used by non expert programmers [6] and DSLs are concise and self-documenting to a large extend [19]. If the domain expert doesn't understand the DSL then close ties with the domain experts have likely not been established [4].

We can validate the correctness of a DSL by making use of a target application. We will model and generate the data-layer of the target application and the generated parts will be used in a modified version of this target application. If the application still performs in the same way it used to, we have ensured that the DSL is working correctly. If the target application makes use of test cases (e.g. NUnit) we can easily validate the correctness and determine whether the generated output is correct. If the same tests succeed for the generated code we most likely know that the DSL is correctly constructed (if we assume that the test cases actually test the application and are not bogus tests that do absolute nothing).

5.8 Conclusion

Building DSLs from scratch is difficult. A proper scope needs to be determined [19] and it's difficult to accurately describe the domain of the customer. We can better focus on adopting a DSL. It's less expensive and requires much less expertise [11]. Finding a DSL to use as a base on the other hand is a tedious task and can be very difficult since the DSL information is scattered widely and often buried [11]. We have not found a suitable DSL to adapt from and a different solution had to be found.

We tried to extend the UML class diagram notations by adding an XML mapping notation. By making use of class diagram notations a developer would be able to experience ultimate flexibility. A developer would be able to design the system and at the same time add persistence support. The problem with this approach is that things sooner or later become very complex. We have to visualize a lot of data and the overview can easily be lost when too much data needs to be displayed.

It has been a wrong decision to base our DSL on the notations of a class diagram. The power of a DSL lies in simplicity. A class diagram has many constructions and describes the structure of the code. We need to model our domain and generate code based on our domain model. There is no need for us to structure our code since this should be a task of the DSL. It's the task of the DSL to generate the code in the most optimal way.

An important question still remains. Is a DSL approach practical in this case? In order to answer this we need to know when to develop DSLs. *Although many articles have been written on the development of particular DSLs, there is very limited literature on DSL development methodologies and many questions remain regarding when and how to develop a DSL* [11]. We can assume that DSLs are a serious option if the problem domain is concrete enough. The purpose of DSL is to solve problems in a specific domain and without a concrete domain this will not be possible. If the problem domain is clear we have to determine which parts can be generated and which parts cannot. This cannot be determined upfront and differs per project.

If a DSL is considered to be an option the real problematic part only begins. As mentioned earlier it's difficult to find the proper scope for a DSL [19] and it's difficult to accurately describe the domain of the customer. There will also be a difficulty of balancing between domain-specificity and general-purpose programming language constructs [19]. Will domain experts understand the notations? Are some notations really necessary and is the chosen visualization really the best for the given domain (object persistence in our case)? These questions can easily be subject of a single thesis. Much more time is needed to give a satisfying answer on any of the questions mentioned above. It is most likely that more difficulties will arise when digging deeper into the subject.

Chapter 6

NHibernate Helper Framework

Is it possible to let a framework handle session and transaction management? In this chapter I will determine whether it is possible to create an abstraction for session and transaction management so that developers don't have to deal with these parts any more.

6.1 Introduction

What is a framework? *A software framework is a reusable design for a software system (or subsystem)* [W17]. The framework's design should be generic enough so that more than a single application can make use of it. There are multiple definitions of a framework. If we make the above definition more concrete we'll see that *a framework is a reusable design of all or part of a system that is represented by a set of abstract classes and the way their instances interact* [12]. A framework can also be seen as *the skeleton of an application that can be customized by an application developer* [12]. The difference in the last two definitions is that the first definition describes the structure of a framework and the second definition describes the purpose of a framework [12]. It's interesting to see that d*evelopers often do not even know they are using a framework, but just talk about the "class library". Frameworks differ from other class libraries by reusing high-level design. This means that there is more to learn before a class can be reused, they can never be reused in isolation, and typically a set of classes must be learned at once. You can often tell that a class library is a framework if there are dependencies among its components and if programmers who are learning it complain about its complexity* [12].

Frameworks are considered to be a kind of domain-specific architecture and are ultimately an object-oriented design [12]. *The primary benefits of OO application frameworks stem from the modularity, reusability, extensibility, and inversion of control they provide to developers* [25]. *Inversion of control allows the framework (rather than each application) to determine which set of application- specific methods to invoke in response to external events* [25].

6.1.1 Advantages

The advantages of the users and developers of a framework have have been listed below:

Users

- *Frameworks are designed with the intent of facilitating software development, by allowing designers and programmers to spend more time on meeting software requirements rather than dealing with the more tedious low level details of providing a working system* [W17].
- *Frameworks promote the basic ideas of software engineering* [W20]:
  - Low coupling; *Low coupling refers to a relationship in which one module interacts with another module through a stable interface and does not need to be concerned with the other module's internal implementation* [W21].
  - High cohesion; *Cohesion is a measure of how strongly-related and focused the responsibilities of a single class are* [W22].
  - Code reuse; *Code reuse is the idea that a partial or complete computer program written at one time can be, should be, or is being used in another program written at a later time. The reuse of programming code is a common technique which attempts to save time and energy by reducing redundant work* [W23].
  - Hide complexity; You don't need to know 'how' things are done (e.g. 'how' data is being stored). Developers thus won't have to deal with tedious low level details of an application [W17].
- *Use is not mandated. A framework is utilized because of the advantage, (both time and money), that it affords* [24].
- *Enables simple applications to be constructed simply. They are not too general, too complex, or too flexible.*

Rather they are easy to understand, use and extend [24].

Developers
- *An explicit technical architecture and vision is defined and maintained throughout development and evolution. These frameworks mature gradually over time and are not subject to undue influence by any single application* [24]. Different kind of applications do not lead the development of the framework making it possible to have a more stable direction (targets will be more solid and they likely won't change easily once they have been set).
- *Architecture is based on a few key concepts, patterns and techniques that are clearly defined from the onset* [24]. This will most likely result in a thoroughly thought through framework that solves the problem domain in an elegant way.

Both users and developers
- *Frameworks enhance modularity by encapsulating volatile implementation details behind stable interfaces. Framework modularity helps improve software quality by localizing the impact of design and implementation changes, which reduces the effort required to understand and maintain existing software* [25].

6.1.2 Disadvantages
Users and developers can experience several disadvantages when using or developing a framework.

Users
- There is always a chance that our newly created framework performs worse than hand-coded persistence. Hand-coded persistence can be optimized in certain situations while a framework should preferably be as generic as possible. If performance loss occurs, one has to ask himself whether the lost performance is crucial or not. There is of course a difference between performance loss and terrible performance.
- Depending on the size of the framework and the number of constraints within the framework, users can find it hard to understand and make use of a framework [12].
- It is very likely that a framework needs to be configured by the users. The configuration part of the framework can become a time consuming process and one has to ask himself whether the framework lightens the burden of the developer or adds more to it.
- *Application requirements change frequently. Therefore, the requirements of frameworks often change as well. As frameworks evolve, the applications that use them must evolve with them* [25].
- Using a particular programming language *restricts frameworks to systems using that language* [12]. There are solutions to this problem by making use of COM, CORBA or by for example making use of a .NET language so the framework can be used in any .NET language (VB.NET, C#, J#, etc.).
- Users highly depend on documentation or framework experts [12] (e.g. the developers of the framework, community members, etc.).

Developers
- More expensive to develop [12].
- *Require better programmers than normal application development* [12].
- Highly skilled object designers need to establish close ties with the users of the framework [24].
- *Framework maintenance may take different forms, such as adding functionality, removing functionality, and generalization. A deep understanding of the framework components and their interrelationships is essential to perform this task successfully* [25].

Both users and developers
- *It is usually hard to distinguish bugs in the framework from bugs in the application code* [25]. Developers and users need to determine whether bugs caused by the framework, incorrect usage of the framework, bugs in the application, etc.
- *Generic components are harder to validate in the abstract. A well-designed framework component typically avoids application-specific details, which are provided via subclassing, object composition, or template parameterization. While this improves the flexibility and extensibility of the framework, it greatly complicates module testing since the components cannot be validated in isolation from their specific instantiations* [25].

6.1.3 Objectives

The Helper framework should transfer as many direct-calls from the NHibernate framework to the Helper framework (figure 6.1.3.1). It's not possible to transfer all the calls to the Helper framework because it is likely that in some cases custom behaviour needs to be added and this shouldn't be provided by the Helper framework. If we add all kinds of rare behavioural exceptions, the framework will contain a lot of rarely used functionalities. The last thing we want is to expose these rarely used functionalities to the developer (making it harder to understand the framework and a decrease in productivity will likely occur).

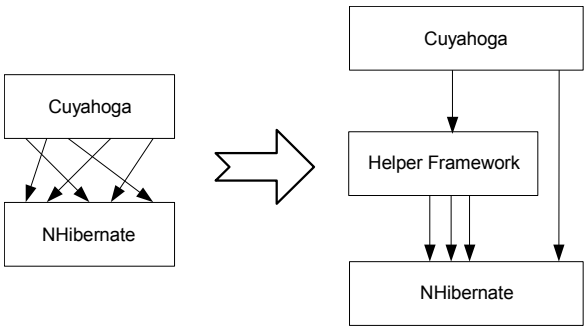

Figure 6.1.3.1: Transfering NHibernate calls to the Helper framework

Besides reducing the calls to the NHibernate framework we should also aim for exposing as little framework constructs as possible. The less a developer needs to be aware of, the better the framework will be understood and this will likely result in a more efficient / effective development process.

Our last objective will be to reduce the LOC (Lines Of Code) needed to persist objects. If the developer needs to write less code to develop a data-layer for an application, we will likely increase productivity. It's arguable whether this framework might decrease the number of defects in the software application. If we look at the defects per function point [8] a decrease in LOC will not decrease the overall defects of an application (there will always be an average of 2.75 defects per function point (table 6.1.3.1) in the development cycle, even if we decrease the LOC needed for specific function points).

If we compare this to the defects per KLOC (1.000 Lines Of Code) we can be quiet sure that the number of defects will decrease if the LOC of an application decreases, since *the number of faults is proportional to the number of lines of code* [18]. A study of Geoffrey Phipps shows us that the number of defects per KLOC differs per programming language. *A typical C++ program had two to three times as many bugs per line of code as a typical Java program* [9]. The .NET framework contains several languages (C#, VB.NET, VC++.NET, J#, etc.) If our Helper framework can reduce the LOC needed to persist objects we might reduce the defects of an application. The amount of defects that can be reduced will differ per used language.

Defect Origins	Defects per Function Point
Requirements	1.00
Design	1.25
Coding	1.75
Documentation	0.60
Bad Fixes	0.40
Total	5.00

Table 6.1.3.1: Defects per life-cycle phase [8]

6.2 Framework design

The framework will be developed in an agile and iterative way. We will make use of a target application that is coupled to NHibernate. The calls to the NHibernate framework will be analysed. The next step is to determine how we can abstract these calls in such a way that these calls can go via the Helper framework. Our target application will be Cuyahoga.

The ultimate framework design is a design that is both extensible and easy to understand. This is however a contradiction because in general an extensible architecture is more difficult to understand. Let's take a look at the Cuyahoga framework to understand how the developers are currently handling sessions and transactions. The complete framework architecture can be viewed in paragraph 6.2.2.

6.2.1 Session Management

There are several DAO layers in the Cuyahoga framework (users, articles, site) to retrieve and store data. Common functionality is shared within a CommonDao class. This class opens a new NHibernate session for almost each DAO operation (figure 6.2.1.1 – before).

```
Before:   [Transaction(TransactionMode.Requires)]
          public void DeleteUser(User user) {
              ISession session = this._sessionManager.OpenSession();
              session.Delete(user);
          }

After:    public void DeleteUser(User user) {
              _commonDao.DeleteObject(user);
          }
```

Figure 6.2.1.1: Removing the session-per-operation anti-pattern

We need to get rid of this anti-pattern and we need the session management go via our framework (figure 6.2.1.1 – after). We can move the CommonDao class to our NHibernate Helper framework and extend or modify it where necessary. The basic supported DAO operations will be: saving an object, saving or updating an object (feature of NHibernate), updating an object, retrieving a single object and retrieving a list of objects.

The Cuyahoga framework makes use of Inversion of Control (IoC), a way to reduce coupling and also known as dependency injection. The developers makes use of a project called Castle Windsor as their IoC framework. We can make use of the IoC design pattern to inject several NHibernate Helper framework dependencies in the Cuyahoga framework (appendix C). The SessionManager and the CommonDao will be injected with the help of this IoC framework. Figure 6.2.1.2 shows us how an object is deleted within the Helper framework.

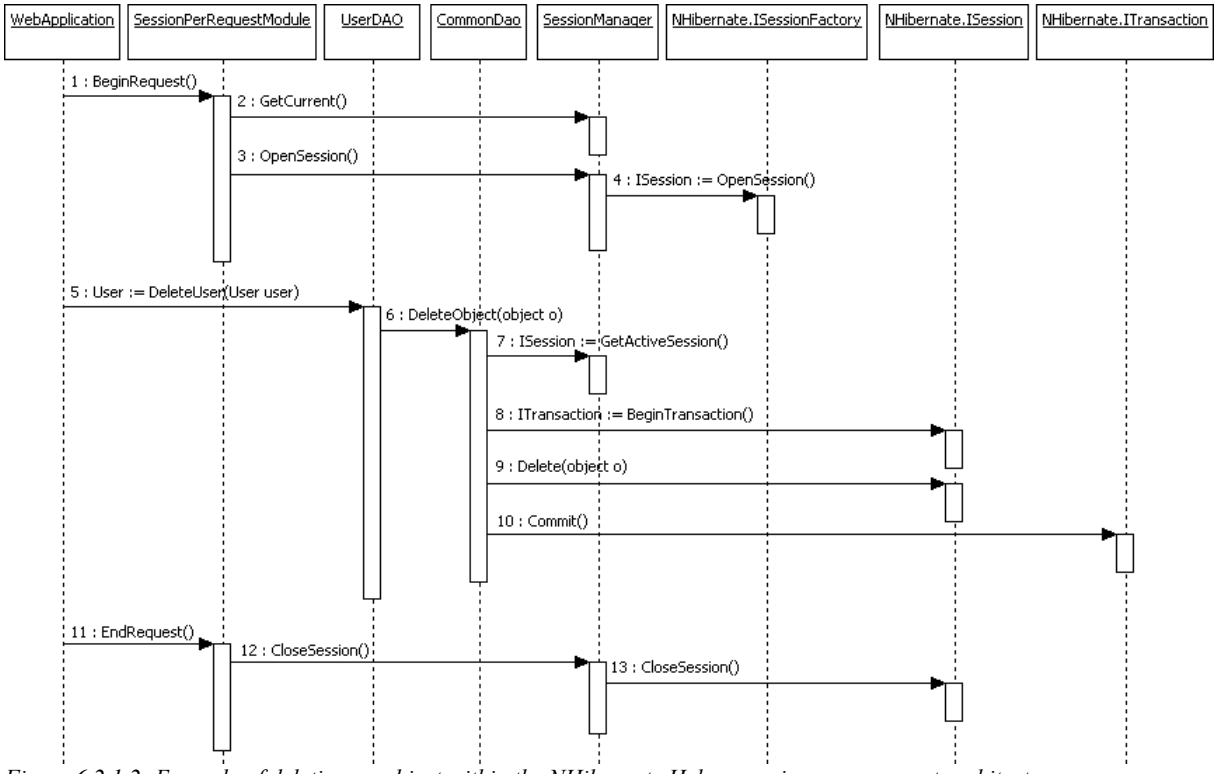

Figure 6.2.1.2: Example of deleting an object within the NHibernate Helper session management architecture

6.2.2 Architectural overview

The complete framework consists of 13 types (figure 6.2.2.1) and a total of 179 LOC. There are just a few key constructs within the framework. The purpose of the different types will be explained below.

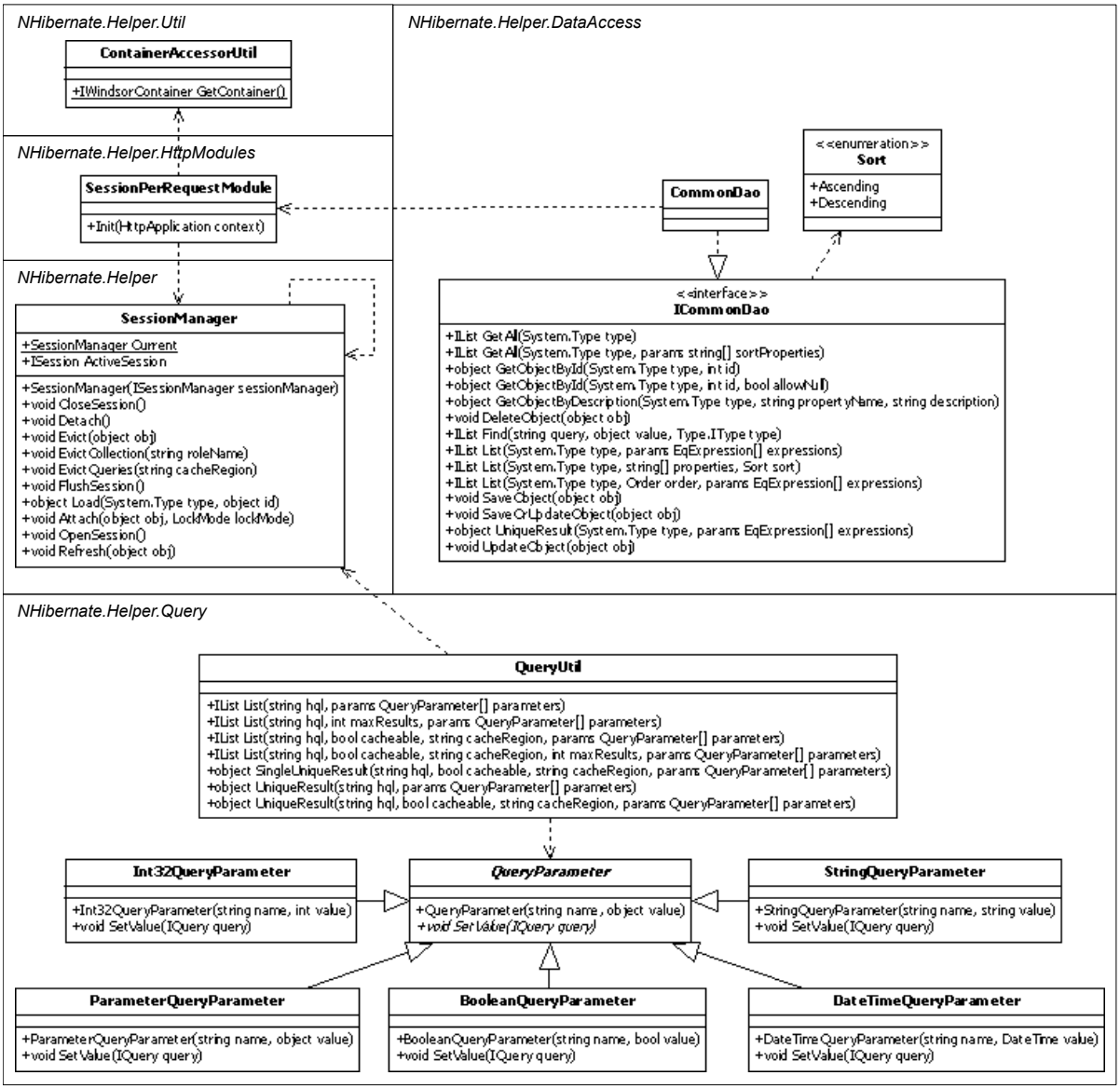

Figure 6.2.2.1: Architectural overview of the Helper framework

CommonDao

The CommonDao has been introduced to handle all the common DAO operations. The Cuyahoga framework consists of several sub-projects and almost each sub-project has one or more DAO's. These DAO's have a lot in common (objects need to be saved, updated, deleted and retrieved in specific ways). All these functionalities have now been centralized within the CommonDao object so that maintenance and extensions are easier to realize.

SessionManager

Retrieves the current active sessions out of the HttpContext instance or the CallContext (depends on whether we're using a Web application or a Windows application). Contains many session related operations like refreshing, attaching an object, opening and closing a session, etc.).

SessionPerRequestModule

The most used and the recommended NHibernate pattern for session management is the session-per-request pattern. This pattern has been implemented in the SessionPerRequestModule and can easily be attached to a new

or existing project. The developer doesn't need to write any code to implement this pattern and it's been now a matter of adding this module to the web.config of an web-application (figure 6.2.2.2).

```
<system.web>
    <httpModules>
        <add type="NHibernate.Helper.HttpModules.SessionPerRequestModule, NHibernate.Helper"
             name="SessionPerRequestModule" />
    </httpModules>
</system.web>
```

Figure 6.2.2.2: Attach the SessionPerRequestModule for a Web application in the web.config

QueryUtil
The QueryUtil is responsible for executing HQL queries. The developer will not have to worry about session management because the QueryUtil will retrieve the active session itself from the Helper framework. QueryParameters will be used for parameterized queries.

QueryParameters
The QueryParameters are being used to make sure that the queries are constructed correctly (type-safe) and to ensure that no SQL injection can occur.

ContainerAccessorUtil
This class is being used to retrieve injected dependencies and has been moved from the Cuyahoga framework to the Helper framework. This class is only used in a small number of cases and most developers will not need to use this class when developing applications. The session-per-request pattern is available in appendix C and makes use of the ContainerAccessorUtil to retrieve the SessionManager.

Sort
This enumeration is being used in the CommonDao implementation so that lists of objects can be retrieved in an ascended or descended way.

6.2.3 Code examples

We can retrieve data in two different ways. We can make use of the CommonDao class or we can make use of the QueryUtil class. The CommonDao class makes use of the NHibernate ICriterion interface and we can supply expressions in code. There is no need for us to construct a query since this will be handled by the NHibernate framework itself (figure 6.2.3.1).

```
EqExpression usernameExpr = new EqExpression("UserName", username);
EqExpression passwordExpr = new EqExpression("Password", password);
IList results = commonDao.List(typeof(User), usernameExpr, passwordExpr);
```

Figure 6.2.3.1: Example of retrieving data with the use of the CommonDao class

With the QueryUtil we will be able to construct queries that might be too complex to construct with the CommonDao class or are easier to understand by creating a plain HQL query (figure 6.2.3.2).

```
string hql = "from Section s where s.ModuleType.ModuleTypeId in (" + String.Join(",", ids) + ")";
return QueryUtil.List(hql);
```

Figure 6.2.3.2: Example of retrieving data with the use of the QueryUtil class

Chapter 7

Validating the Helper framework

In this chapter I will validate the Helper framework by making use of a target application called Cuyahoga. I will make use of an open-source website framework called Cuyahoga as my validator. I will determine the couple points of this framework with NHibernate. If the couple points have been found we can determine whether a framework is feasible.

7.1 Cuyahoga

Cuyahoga is an open-source website framework built in .NET, targeting content management. The main purpose of this project is to show .NET web developers that there is a different way of developing Web applications than the well known sample Web applications. Currently there are 2 active developers working at the project and they have released 4 versions (0.9.1, 1.0, 1.01 and 1.5.0). The project is divided into 9 sub-projects, all having their own responsibilities. The project consists of nearly 9 KLOC (table 7.1.1). It has been downloaded 12.346 times since its first release at the end of 2005.

Sub-Project	# Lines of Code	# Lines of Comments
Cuyahoga.Core	1875	2112
Cuyahoga.Modules	548	1090
Cuyahoga.Modules.Articles	568	847
Cuyahoga.Modules.Downloads	333	209
Cuyahoga.Modules.Flash	166	78
Cuyahoga.Modules.Forum	1455	402
Cuyahoga.Modules.RemoteContent	299	375
Cuyahoga.ServerControls	382	126
Cuyahoga.Web	3063	1276
Total	**8689**	**6515**

Table 7.1.1: Overview of the Cuyahoga framework and its individual sub-projects

7.1.1 Motivation

We will make use of Cuyahoga because it is a medium sized project with a small number of developers. It has been downloaded many times and the development activity is high, making it an attractive framework to use. For Web development this is a representative project because most Web applications are small to medium sized and there are, in general, not many developers working at a Web application either. In the survey 'Investigating Early Web Size Measures for Web Cost Estimation' it is mentioned that *the companies surveyed had from one person to 20 people, with median of 4.5 and mean of 5.44. 16 companies (50%) have no more than four people, and 25 companies (78.1%) have no more than six people, suggesting that Web companies may fit the profile of small companies* [5].

7.2 Success Factors

The Cuyahoga framework is tightly coupled to the NHibernate framework and we have to determine whether it is actually feasible to build our Helper framework. If the code needed to persist objects is too specific rather than generic, it is possible that a Helper Framework will not be feasible. There are three factors for success for the Helper framework:

1. Reduce the number of calls to the Helper Framework compared with the NHibernate framework; Currently the Cuyahoga framework is tightly coupled to the NHibernate framework by making numerous direct calls.

If we can put the same functionality in the Helper framework with a reduced number of calls we can possibly decrease the complexity of developing data-layers (the developer will be exposed to less constructs).

2. Reduce the LOC needed to persist or retrieve objects; If the same functionality can be described in less code, we will decrease the complexity of developing data-layers.
3. Reduce the number of constructions the Helper Framework exposes compared to the NHibernate framework; The NHibernate framework consists of many constructions to persist objects. We assume that we don't need most of the constructions and we will decrease complexity if the Helper framework only exposes the constructions needed to persist objects. It is likely that this will differ for Web and Windows applications.

7.3 NDepend

We need to determine how the NHibernate framework is coupled to the Cuyahoga project. The couple points will then be analysed to determine whether it is feasible to create an abstraction so users won't have to deal with sessions and transactions any more. We will make use of NDepend [W7] to identify the couple points. NDepend is a source-code analyser for .NET and contains an Code Query Language (CQL) to analyse the code of one or more assemblies. These queries can help understand the architecture of an application but can also determine whether certain assemblies are stable or unstable. There are many more quality queries that can be run against code but these will not be mentioned here.

NDepend has a dependency view so one can understand how certain parts are coupled to each other (figure 7.3.1). A drawback of using the dependency view is that this view doesn't accurately show the calls to the assemblies. It only displays how many methods of assembly X are making use of members of assembly Y. If one method of assembly X has two calls to the same member of assembly Y, this will only count as one. In order to determine the number of calls we have to rely on our development IDE (Visual Studio .NET, an IDE to develop .NET applications). Within Visual Studio .NET we can determine the number of references to variables, classes, interfaces and enumerations.

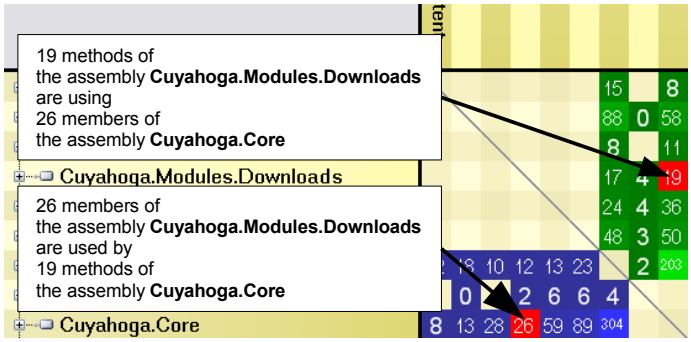

Figure 7.3.1: How to read the dependency view

7.4 NHibernate couple points

When we run NDepend we can see that NHibernate is coupled to several assemblies (displayed in the bottom of figure 7.4.1). The numbers in this figure display how many members of the NHibernate framework are being used in one or more sub-projects of the Cuyahoga framework. This gives us an indication that the Cuyahoga.Core assembly is tightly coupled to the NHibernate framework. We need a new level of detail to determine how the session and transactions are integrated within the Cuyahoga framework. It will be interesting to see where several key constructions in NHibernate related to sessions and transactions are being used in the Cuyahoga framework. By creating a CQL query, we can easily identify the couple points by searching for all the NHibernate session and transaction related types within the Cuyahoga framework.

The CQL query in figure 7.4.2 will search for all the possible types within the loaded assemblies that directly make use of either the ISession, ITransaction or ISessionFactory interface. NHibernate will be excluded from the loaded

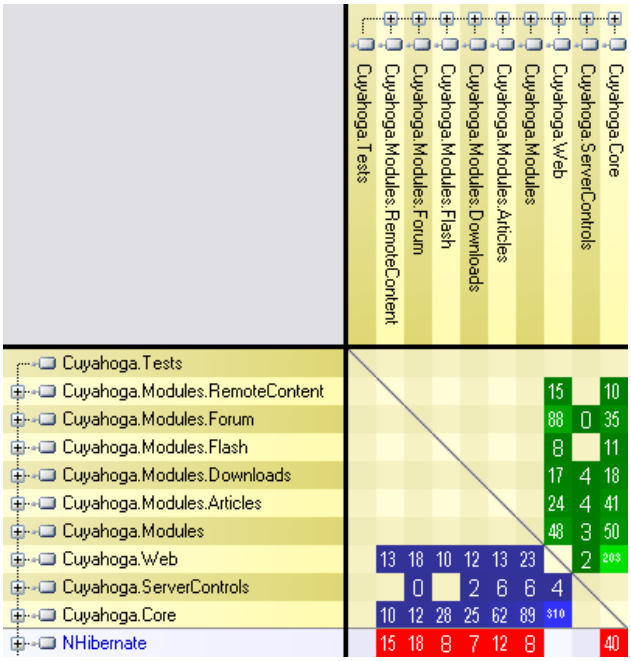

Figure 7.4.1: NHibernate couple points with Cuyahoga

assemblies because we don't want to identify the couple points within the NHibernate framework itself. The loaded assemblies are assemblies of the Cuyahoga framework but also a Castle project assembly called NHibernateIntegration. We cannot analyse the couple points of this assembly because it cannot be read by NDepend and has possibly been obfuscated. All the references to the ISessionManager interface within the NHibernateIntegration assembly have been tracked within Visual Studio .NET.

```
SELECT TYPES OUT OF ASSEMBLIES "NHibernate" WHERE
(IsUsing "NHibernate.ISession" AND IsDirectlyUsing "NHibernate.ISession") OR
(IsUsing "NHibernate.ITransaction" AND IsDirectlyUsing "NHibernate.ITransaction") OR
(IsUsing "NHibernate.ISessionFactory" AND IsDirectlyUsing "NHibernate.ISessionFactory")
```

Figure 7.4.2: NDepend CQL query to determine how sessions and transactions are directly being used in the Cuyahoga framework

There are many calls to the NHibernate framework and they occur in 7 of the 9 sub-projects (appendix B). Is it possible to centralize the calls through the helper framework so that the sub-projects will communicate with the Helper Framework rather than directly to the NHibernate framework?

7.5 Validation

The NHibernate Helper framework has been completely integrated in the Cuyahoga framework and we now need to determine whether the NHibernate Helper framework satisfies any of the earlier defined success criteria:

1. Reduce the number of calls to the Helper Framework compared with the NHibernate framework
2. Reduce the LOC needed to persist or retrieve objects
3. Reduce the number of constructions the Helper framework exposes compared to the NHibernate framework
4. Reduce complexity

7.5.1 Reduce the number of calls

We can easily determine whether the number of calls have increased or decreased by comparing the orginal Cuyahoga framework with the NHibernate Helper framework integrated version. If the number of calls have increased we need to identify the reason of this increase and determine whether it is possible to decrease it.

If we compare the two Cuyahoga frameworks we see that in the original framework 23 types were coupled to NHibernate and these types made 128 calls to this framework (figure 7.5.1.1). With the NHibernate Helper framework, 20 types are still coupled to NHibernate but are now making only 27 calls, a reduction of 101 calls. We however created an even tighter coupling with the NHibernate Helper framework: 30 types within the Cuyahoga framework make 160 related calls to the NHibernate Helper framework. Last but not least we can see that the NHibernate Helper framework couples 11 types to the NHibernate framework and these types are responsible for 28 calls. It's interesting to see that we created the same functionality while making less calls to the NHibernate framework (128 calls in the original framework compared to 55 calls in the modified framework). We have to determine where the 160 calls to the NHibernate Helper framework come from.

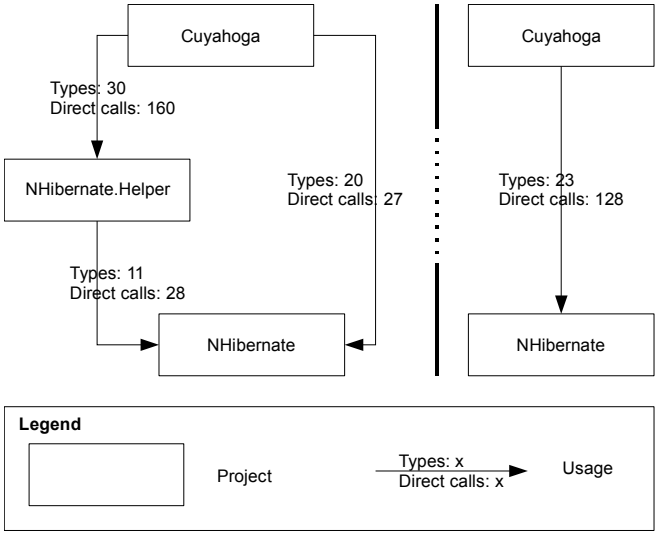

Figure 7.5.1.1: Difference in calls between the NHibernate Helper implementation (left) and the original Cuyahoga framework

Query mechanism

Our query mechanism makes use of an inheritance structure and the amount of sub-classes might be the reason why there are so many dependencies to the NHibernate Helper framework. For each parameter in a query a new object needs to be introduced (figure 7.5.1.2).

```
string hql = "from User u where u.UserName = :username and u.Password = :password";
QueryParameter usernameParam = new StringQueryParameter("username", username);
QueryParameter passwordParam = new StringQueryParameter("password", password);
IList results = QueryUtil.List(hql, usernameParam, passwordParam);
```

Figure 7.5.1.2: Current way of handling HQL queries

Currently we have five kinds of parameters (Int32, String, Boolean, DateTime and Parameter) but this can significantly increase if new projects need different parameters than the ones defined. The query parameters construction tightens the coupling between the Helper framework and 47 methods of the Cuyahoga framework make use of 6 types of the Helper framework (the QueryUtil has been excluded from the number of calls because without this class we will not be able to execute HQL queries). A total of 52 dependencies will be removed if the query mechanism is replace by a better approach (some methods were making use of more than one NHibernate type and this results in a higher number than you might expect).

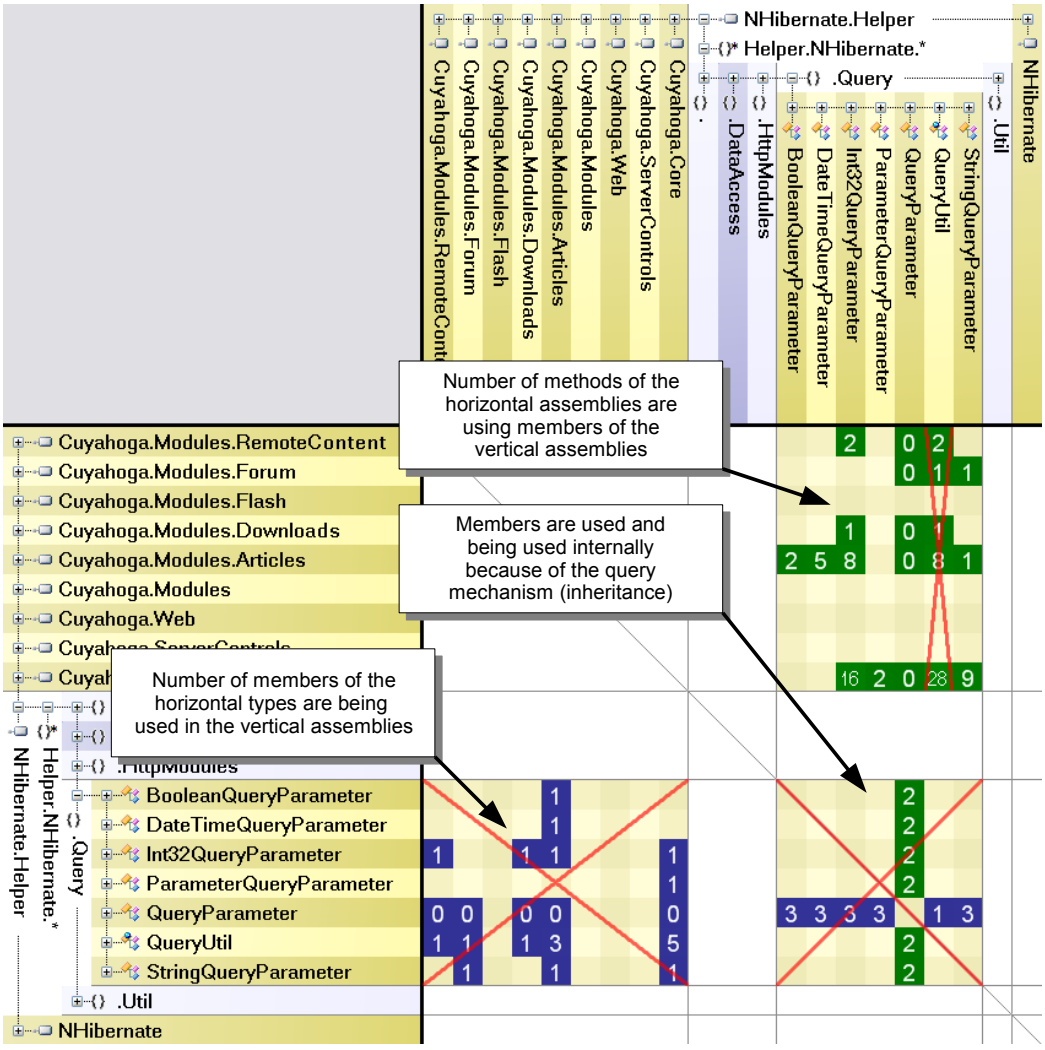

Figure 7.5.1.3: Query mechanism couple points within the NHibernate Helper framework

The "number of members of the horizontal types that are being used in the vertical assemblies" are not the calls. The constructor of the Int32QueryParameter is called numerous times in the vertical assemblies but it will still count as only one. It only matters that "a" member is being used and it doesn't matter how many times it is being used.

7.5.2 Reduce the LOC needed to persist objects

The difference in LOC between the original Cuyahoga framework and the modified Cuyahoga framework is 302 LOC (table 7.5.2.1). The Helper Framework consists of 179 LOC. We need 123 less LOC to create the same functionality. We have reduced the LOC needed to persist objects.

Sub-Project	#Lines of Code	#New Lines of Code	Difference	Percentage
Cuyahoga.Core	1875	1732	143	7,63
Cuyahoga.Modules	548	543	5	0,91
Cuyahoga.Modules.Articles	568	558	10	1,76
Cuyahoga.Modules.Downloads	333	326	7	2,1
Cuyahoga.Modules.Flash	166	156	10	6,02
Cuyahoga.Modules.Forum	1455	1357	98	6,74
Cuyahoga.Modules.RemoteContent	299	277	22	7,36
Cuyahoga.ServerControls	382	382	0	0
Cuyahoga.Web	3063	3056	7	0,23
Total	**8689**	**8387**	**302**	**3,48**

Table 7.5.2.1: Differences in LOC between the original Cuyahoga framework and the modified framework

7.5.3 Reduce the number of exposed constructions

The NHibernate Helper framework exposes 11 types to the Cuyahoga framework. Six of these types are query parameters and it's likely that the can be removed (paragraph 7.6 – Future work). If we take this into account we will see that there are only 5 constructs that are heavily used. If we add a reference to the NHibernate framework the developer will be exposed to minimal 41 constructs (if we only import the root namespace). Since custom behaviour has been added to the Cuyahoga framework these constructs are still accessible for a developer. In the most ideal situation a developer is only exposed to these 5 Helper framework constructs.

7.5.4 Reduce complexity

There are several interesting things that we can see if we compare the original and modified Cuyahoga framework (table 7.5.4.1). We have removed / moved 3 classes from the original framework (1 of these classes was abstract). The number of IL instructions has been decreased with 880. An IL instruction is an instruction for the Intermediate Language. A decrease in IL instructions is a good sign because the same behaviour can be created with less instructions. This will likely decrease the difficulty of creating data layers. It's interesting to see that the afferent coupling decreased slightly while the efferent coupling increased a lot. Afferent Coupling (Ac) displays *the number of classes from other packages that depend on the classes within the package* [15]. A high Ac indicates that an assembly is highly used in the project and are often the most stable assemblies. A low Ac means that fewer or no assemblies depend on the assembly. Efferent coupling (Ec) displays *the number of classes from other packages that the classes within the package depend upon* [15]. A high Ec indicates that an assembly is highly dependent on an assembly while a low Ec indicates that an assembly is less dependent or not dependent at all.

Assembly	# Types	# Abstract Types	# IL instruction	Afferent coupling	Efferent coupling	Relational Cohesion	Instability	Abstractness	Distance
Cuyahoga.Core v1.5.0.0	-2	-1	-447	-1	4	-0,05	0,03	0	-0,03
Cuyahoga.Web v1.5.0.0	-1	0	-42	0	2	-0,04	0,01	0,01	-0,01
Cuyahoga.Modules.Articles v1.5.0.0	0	0	29	0	6	0	0	0	0
Cuyahoga.ServerControls v1.5.0.0	0	0	0	0	0	0	0	0	0
Cuyahoga.Modules.Downloads v1.5.0.0	0	0	-17	0	4	0	0	0	0
Cuyahoga.Modules.Flash v.1.5.0.0	0	0	-45	0	0	0	0	0	0
Cuyahoga.Modules.Forum v.1.5.0.0	0	0	-276	0	2	0	0	0	0
Cuyahoga.Modules.RemoteConent v.1.5.0.0	0	0	-72	0	2	0	0	0	0
Cuyahoga.Modules v.1.5.0.0	0	0	-10	0	0	0	0	0	0
Total	**-3**	**-1**	**-880**	**-1**	**20**	**-0,09**	**0,04**	**0,01**	**-0,04**

Table 7.5.4.1: Relative differences between the original and modified Cuyahoga framework

We've made the Cuyahoga framework less dependent to itself but increased the classes and interfaces it "knows" about (Ec). It is very likely that the Ec has been increased this much because of the query mechanism.

The Relational Cohesion is the *average number of internal relationships per type. The relational cohesion represents the relationship that this assembly has to all its types. As classes inside an assembly should be strongly related, the cohesion should be high. On the other hand, too high values may indicate over-coupling. A good range for RelationalCohesion is 1.5 to 4.0. Assemblies where RelationalCohesion < 1.5 or RelationalCohesion > 4.0 might be problematic* [W27]. The relational cohesion of the Cuyahoga Core and the Cuyahoga Web project have decreased from respectively 1,87 and 2,9 to 1,82 and 2,86. These numbers are still in the acceptable boundaries between 1.5 and 4.0.

The project has become a little bit more instable and a little bit more abstract but these numbers are so small that they can be neglected.

The distance is *the perpendicular normalized distance of an assembly from the idealized line A + I = 1 (called main sequence). This metric is an indicator of the assembly's balance between abstractness and stability. An assembly squarely on the main sequence is optimally balanced with respect to its abstractness and stability. Ideal assemblies are either completely abstract and stable (I=0, A=1) or completely concrete and instable (I=1, A=0). The range for this metric is 0 to 1, with D=0 indicating an assembly that is coincident with the main sequence and D=1 indicating an assembly that is as far from the main sequence as possible. The picture in the report reveals if an assembly is in the zone of pain (I and A both close to 0) or in the zone of uselessness (I and A both close to 1)* [W27]. The distance has also decreased a little bit and this value is also significantly small that it can be neglected.

It seems that our framework does lighten the burden of the developer. We have to make sure that the efferent coupling decreases since in the ideal situation we don't want the Cuyahoga framework to "know" about many more classes and interfaces. If we can modify the query mechanism this can most likely be achieved.

7.6 Future Work

There are still a few interesting parts that have not been touched because of time constraints.

7.6.1 Query mechanism

The way we handle queries can be significantly improved. Currently we need to manually declare all the needed parameters but this can possibly be skipped all together (figure 6.3.1.1). If we can parse the HQL query we will be able to build the parameters based on reflection. The HQL query is directly linked with the domain object and any properties used in the query (e.g. user.UserName) can be easily reflected to determine the type of the property. It is important that the parameters in the HQL query are in the same order as the values that are being used in the method's arguments. The drawback of this approach is that we cannot determine at compile-time whether the queries are correctly constructed or not. If the types of the values do not match the parameters in the HQL query we will only know about it at run-time.

```
IList results = NHibernate.Helper.Query.QueryUtil.List(hql, typeof(User),
                usernameParam, passwordParam);
```

Figure 7.6.1.1: Possibly improved way of handling queries

This approach makes it possible to significantly decrease the coupling with the NHibernate Helper framework, resulting in an easier to understand framework and possibly a more effective / efficient development process. As mentioned earlier in paragraph 7.5.1, 52 calls to the NHibernate framework will also be removed. The LOC needed to persist objects will decrease with an additional 0,59% to a total number of 4,07%.

7.6.2 Design measurement

It would be interesting to see if our Helper framework's design is optimal. Ralf Reißing has made a model towards object-oriented design measurement [16]. From an architectural point of view it is desirable to see any possible flaws or wrong design constraints in the architecture. I have tried to minimize this impact by having several conversations about the architecture. The conversations have been held with my internship supervisor at Sogyo and two software architects of Sogyo.

7.7 Conclusion

Creating a framework is difficult, better programmers are required [19] and is also more expensive compared to normal application development [12]. A framework should be usable in multiple projects and should simplify development. Developers need to be able to make use of the framework and understanding the concepts of the created framework is an important part. Developers might find it difficult to make use of the framework and comprehension depends on the size of the framework and the number of constraints within this framework.

The created Helper framework can likely simplify the development process. We have reduced the total number of LOC needed to persist objects with 3,48%. This number will increase to 4,07% if we manage to improve the query mechanism. We have been able to reduce the number of calls to the NHibernate framework but since there were certain 'fixes' and some custom behaviour within the Cuyahoga framework we've not been able to complete remove all the calls to the NHibernate framework (this was not a goal though). There is still a large number of NHibernate types used within the Cuyahoga framework and we have only been able to slightly reduce this number (thanks to the 'fixes' and custom behaviour). The Helper framework contains substantial less constructs compared to the NHibernate framework and it is very likely that developers find it easier to understand.

A disadvantage of our Helper framework is that we will not be able to use NHibernate in every project. We've only implemented the session-per-request pattern but this will do for most Web applications. Due to time constraints it has not been possible to determine how most Windows applications are managing NHibernate sessions. It is possible that two frameworks need to be created. A Web Helper framework and a Windows Helper framework but this depends on the differences in session management. If there are substantial differences in session management it might be best to separate the projects physically in two new projects.

Chapter 8

Conclusion

We have conducted a survey to identify the most important problem related to working with the NHibernate framework. The survey has been picked out of several information acquisition methods based on defined criteria. It is possible that the defined criteria in this thesis to determine the best information acquisition method are biased and that in certain situations the weight of these criteria are different (depending on how a certain information acquisition method is performed). There was not much user input and one has to ask himself whether we have identified the most important problem related to working with the NHibernate framework. The survey participants were all employees of Sogyo and this might have biased the results. Five people of Sogyo have participated and sessions have been problematic to 40% of the users (only 2 people). It's likely that with a bigger sample group a different problem would have been revealed.

The most difficult part of this thesis was the dynamic aspect of finding the most important problem “on the fly”. We had to determine the problem during the process and this couldn't be done upfront. Without this problem further research was not possible since the thesis heavily depends upon it.

We've analysed two approaches to solve a single problem: Can we create an abstraction for NHibernate so that developers don't have to work with sessions and transactions any more? The first approach made use of a DSL while the second approach made use of a custom created helper framework. A DSL approach is not recommended for highly customizable code and is only useful when it supports the developer with services or parts that are static and will not likely change or can be regenerated without any problems (e.g. XML mapping files). The framework approach will only work well if we do not simply shift the responsibilities from the NHibernate framework to the Helper framework. The Helper framework should lighten the burden of the developer and if we only shift parts of one framework to another we will not get the desired results. The framework has been build in an iterative and agile way. New functionality has only been added when necessary. It is likely that the framework will continue to grow if more functionality is needed.

To understand the effect of both solutions in the development process we have to measure the gain or loss in efficiency. Because of time constraints it has not been possible to measure the effect and we cannot determine in a real-life case which solution is best. We can only give insight in a framework approach based on the metrics in this thesis. A framework approach is a good approach because it can easily be extended and is easier to maintain when comparing a framework approach with a DSL-based approach. A DSL-based solution is only maintainable if the generated parts can be regenerated without loss of functionality and this can only be realized when we generate parts that will never change or if we can separate the generated code from the implementation code. If there is great overlay of code in the generated code, it is better to create a framework.

It is important to understand that DSLs and frameworks don't exclude each other. It is very likely that these methods can both solve the problem domain of object persistence. We can even combine these methods but the target of these methods will then differ. We can for example create a DSL to generate XML mapping files while a framework is used to simplify session and transaction management.

Chapter 9

Future Work

Most of the future work has already been discussed in chapter 5 (Domain Specific Language) and chapter 7 (Validating the Helper Framework). There are still some questions that remain at the end of this thesis and have not been answered. An important question that remains is: “When do we develop a DSL”? This is a difficult question and it would be a great if criteria can be determined to answer this question.

An other important question that still remains is: How do we determine the best solution for the most important problem? We have identified two possible solutions to solve the problem domain. A DSL approach and a framework approach. It might be possible that there is a better approach to solve the problem domain. We can only compare the identified solutions.

References

[1] V. Srinivasan, D. T. Chang, Object persistence in object-oriented applications in IBM systems journal VOL 36, no 1, 1997

[2] Tira Cohene, Steve Easterbrook, Contextual Risk Analysis for Interview Design in Proceedings of the 13th IEEE International Conference on Requirements Engineering (RE'05), Volume 00, ISBN: 0-7695-2425-7, Pages 95 – 104, 2005

[3] Vasja Vehovar, Zenel Batagelj, Katja Lozar Manfreda, Metka Zaletel, Non-response in web surveys in R. Groves et all (Eds.), Survey Nonresponse, Wiley, 2002

[4] J. Scott Armstrong, Terry S. Overton, Estimating Nonresponse Bias in Journal of Marketing Research, 14, Pages 396 – 402, 1977

[5] Emilia Mendes, Nile Mosley, Steve Counsell, Investigating Early Web Size Measures for Web Cost Estimation in Journal of Systems and Software archive Volume 77, Issue 2 (August 2005), ISSN: 0164-1212, Pages 157 – 172, 2005

[6] P. Hudak, Modular Domain Specific Languages and Tools in Fifth International Conference on Software Reuse ICSR'98), Page 134, 1998

[7] Arie van Deursen, Paul Klint, Henri Ford in software engineering, 2003

[8] Norman Fenton, Martin Neil, A Critique of Software Defect Prediction Models, Volume 25, SSN: 0098-5589, Pages 675 – 689, Sep/Oct 1999

[9] Geoffrey Phipps, Comparing Observed Bug and Productivity Rates for Java and C++, Software – Practice and Experience, 29(4), Pages 345 – 358, 1999

[10] J. B. Gross, Programming for Artists: A Visual Language for Expressive Lighting Design, Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05), 0-7695-2443-5/05, 2005

[11] M. Mernik, J. Heering, A.M. Sloane, When and how to develop domain-specific languages, Report Sen-E0309, 2003

[12] Ralph E. Johnson, Frameworks = (Components+Patterns) - How frameworks compare to other object-oriented reuse techniques in Communications of the ACM Vol 40 No. 10, 1997

[13] Takafumi Saito, Tokiichiro Takahashi, Comprehensible rendering of 3-D shapes in Proceedings of the 17th annual conference on Computer graphics and interactive techniques SIGGRAPH '90, Volume 24, Issue 4, September 1990

[14] David Wile, Lessons Learned from Real DSL Experiments in Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS'03), 0-7695-1874-5/03, 2002

[15] Kirk Knoernschild, Using Metrics To Help Drive Agile Software in Agile Journal, 2006

[16] Ralf Reißing, Towards a Model for Object-Oriented Design Measurement in 5th International ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering, 2001

[17] Ali Ibrahim, William R. Cook, Automatic Prefetching by Traversal Profiling in Object Persistence Architectures, Proc. of the European Conference on Object-Oriented Programming (ECOOP), 2006

[18] Thomas J. Ostrand, Elaine J. Weyuker, Robert M. Bell, Where the Bugs Are in Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis, ISBN:1-58113-820-2, Pages: 86 – 96 , 2004

[19] Arie van Deursen, Paul Klint, Joost Visser, Domain-Specific Languages: An Annotated Bibliography in ACM SIGPLAN Notices, Volume 35, Issue 6, Pages: 26 – 36, ISSN:0362-1340, 2000

[20] J.A. Lehman , An Empirical Comparison of Textual and Graphical Data Structure Documentation for Cobol Programs, pp. 1131-1135, September 1989

[21] Nahum D. Gershon, Breaking the Myth: One Picture is NOT (Always) Worth a Thousand Words in Visualization '96 Proceedings, On page(s): 441-443, ISSN: 1070-2385, ISBN: 0-89791-864-9, 27 Oct-1

Nov 1996

[22] Amy Voida, Elizabeth D. Mynatt, Challenges in the Analysis of Multimodal Messaging in Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work, Pages: 427 – 430, ISBN:1-59593-249-6, 2006

[23] S. McCloud, Understanding comics: The invisible art. New York: Harper Collins, 1993

[24] Todd Hansen, Development of successful object-oriented frameworks in Conference on Object Oriented Programming Systems Languages and Applications, Pages: 115 – 119, ISBN:1-58113-037-6, 1997

[25] Mohamed Fayad, Douglas C. Schmidt, Object-Oriented Application Frameworks in Communications of the ACM, Volume 40, Issue 10, Pages: 32 – 38, ISSN:0001-0782, 1997

Web References

[W1] Conducting a survey, http://www.studentbmj.com/back_issues/0501/education/143.html

[W2] Survey non response bias, http://www.statpac.com/surveys/nonresponse-bias.htm

[W3] Response bias, http://en.wikipedia.org/wiki/Response_bias

[W4] Interview, http://en.wikipedia.org/wiki/Interview

[W5] Selection bias, http://en.wikipedia.org/wiki/Selection_bias

[W6] ThreadStatic, http://www.hanselman.com/blog/ATaleOfTwoTechniquesTheThreadStaticAttributeAndSystemWebHttpContextCurrentItems.aspx

[W7] NDepend, http://www.ndepend.com

[W8] Properties, http://msdn2.microsoft.com/en-us/library/x9fsa0sw(VS.80).aspx

[W9] NHibernate, http://www.hibernate.org/343.html

[W10] J. A. Silva, “DSL Tools VS2005 joseas ISEL”, http://labnet.cc.isel.ipl.pt/docs/Eventos/6-04-2005/20050405 DSL Tools VS2005 joseas ISEL.ppt

[W11] http://www.hibernate.org/42.html

[W12] Chapter 19. Best Practices, http://www.hibernate.org/hib_docs/nhibernate/html/best-practices.html

[W13] AppDomain Class, http://msdn2.microsoft.com/en-us/library/system.appdomain.aspx

[W14] Survey Definition, http://www.thefreedictionary.com/survey

[W15] CallContext class, http://msdn2.microsoft.com/en-us/library/system.runtime.remoting.messaging.callcontext(VS.80).aspx

[W16] Data Access Object, http://en.wikipedia.org/wiki/Data_Access_Object

[W17] Framework, http://en.wikipedia.org/wiki/Framework

[W18] Chapter 11. HQL: The Hibernate Query Language, http://www.hibernate.org/hib_docs/nhibernate/html/queryhql.html

[W19] Problem, http://en.wikipedia.org/wiki/Problem

[W20] M. Nankov, Object Persistence and Nhibernate, http://www.aubg.bg/cssu/seminars/pdf/Object_Persistence_and_Nhibernate.pdf

[W21] Coupling (computer science), http://en.wikipedia.org/wiki/Coupling_(computer_science)

[W22] Cohesion (computer science), http://en.wikipedia.org/wiki/Cohesion_(computer_science)

[W23] Code reuse, http://en.wikipedia.org/wiki/Code_reuse

[W24] altinoren.com - ActiveWriter Introduction, http://www.altinoren.com/activewriter/

[W25] ActiveRecord :: Castle Project, http://www.castleproject.org/activerecord/index.html

[W26] Race condition, http://en.wikipedia.org/wiki/Race_condition

[W27] NDepend Metrics, http://www.ndepend.com/Metrics.aspx

Appendix A

Survey

1. Introduction

1.1. Have you ever used NHibernate or are you currently using NHibernate?
   a) Yes, go to the next section
   b) No, go to section 2 and then go to section 5 at the end of the survey

2. Personal Information

2.1. What is your age?
   a) ... Years
2.2. What is your gender?
   a) Male
   b) Female
2.3. What is your background / degree (fill in your highest degree e.g. Software Engineering at the University)?
   a) ...
2.4. How long have you been programming?
   a) ... Weeks / Months / Years
2.5. Have you been doing object oriented design?
   a) Yes, go to 6
   b) No, go to 7
2.6. How long have you been doing object oriented design?
   a) ... Weeks / Months / Years
2.7. I consider my object oriented skills as
   a) Beginner
   b) Intermediate
   c) Expert
2.8. I consider my programming skills as
   a) Beginner
   b) Intermediate
   c) Expert

3. Company Information

3.1. Do you use or did you use NHibernate for professional purposes?
   a) Yes, go to 2
   b) No, go to the next section
3.2. How many employees are working in your development department?
   a) Employees
3.3. Who is responsible for the data-layer of a project?
   a) Specialists (please specify), go to 4
      • ...
   b) Everyone, go to 5
   c) Other (please specify), go to 4
      • ...
3.4. How many employees at the development department are working at the data-layer?
   a) ... Employees
3.5. In what kind of projects are you using or have you used NHibernate (multiple answers possible). Please specify per project size what percentage of the projects is making use of NHibernate? If you do not use NHibernate in certain project sizes then you should not specify a percentage.
   a) Small projects (LOC < 5.000),
      • ... percent
   b) Medium projects (>= 5.000 LOC < 50.000)
      • ... percent
   c) Large projects (>= 50.000 LOC < 250.000),
      • ... percent
   d) Huge projects (LOC >= 250.000)
      • ... percent

4. NHibernate

4.1. How long have you been working with NHibernate?
   a) ... Weeks / Months / Years

4.2. How long did it take you to become familiar with the NHibernate principles and constructs (e.g. if you spend 1 hour a day learning NHibernate for 3 months (5 days a week) this will result in about 8 days)?
   a) ... Days / Weeks / Months

4.3. When working with NHibernate, do you see any repetitive pattern (e.g. duplicating code / other parts of the project or doing specific things over and over but with minor changes, etc.)?
   a) Yes, go to 4
   b) No, go to 5

4.4. Which repetitive patterns do you see?
   a) ...

4.5. Did you have any problems when you first started working with NHibernate?
   a) Yes, go to 6
   b) No, go to 7

4.6. Which problems occurred when you first started working with NHibernate?
   a) ...

4.7. Do you currently have any problems when working with NHibernate?
   a) Yes, go to 8
   b) No, go to 9

4.8. Which problems do you currently have when working with NHibernate?
   a) ...

4.9. What can be done to make NHibernate easier to use?
   a) ...

4.10. What percentage of the a project's development time (when NHibernate is being used), is spend on one of the following parts:
   a) Database reconfiguration (updating / creating tables)
      - ... percent
   b) Updating the data within the database because of structural changes
      - ... percent
   c) Creating / updating mapping files
      - ... percent
   d) Updating NHibernate code (not considering business logic, presentation code, etc.)
      - ... percent
   e) Other NHibernate related tasks (please specify per task)
      - ...

4.11. How do you couple NHibernate to your database?
   a) XML mappings files, go to 12 then go to 15
   b) Attributes in the source code, go to 13 then go to 15
   c) Both, go to 14

4.12. Why do you prefer coupling NHibernate with XML mapping files?
   a) ...

4.13. Why do you prefer coupling NHibernate with attributes in the source code?
   a) ...

4.14. Why do you use both XML mappings and attributes? Do you mix these concepts in a single project or does it differ from one project to another (please explain why)?
   a) ...

4.15. What are the main reasons to make use of NHibernate within your projects.
   a) ...

4.16.Do these reasons differ per project size?
   a) Yes, go to 17
   b) No, go to 18

4.17. Please specify per project size why it differs (only for the project sizes that differ):
   a) Small projects (< 5.000 LOC)
      - ....
   b) Medium projects (>= 5.000 LOC, < 50.000 LOC)
      - ...
   c) Large projects (>= 50.000 LOC, < 250.000 LOC)
      - ...
   d) Huge projects (>= 250.000 LOC)
      - ...

4.18. Do you manage your own sessions within NHibernate?
   a) Yes, go to 19
   b) No, go to 20

4.19. Why are you managing your own sessions within NHibernate?
   a) ...

4.20. Do you let NHibernate handle concurrency?
   a) Yes, go to 22
   b) No, go to 21
4.21. Why are you managing your own concurrency?
   a) ...
4.22. Are you using data locking or have you used data locking?
   a) Yes, go to 23
   b) No, go to 25
4.23. If you need data locking, do you let NHibernate handle it?
   a) Yes, go to 25
   b) No, go to 24
4.24. Why are you creating your own data locking mechanism?
   a) ...
4.25. Are you managing your own database connections?
   a) Yes, go to 26
   b) No, go to 27
4.26. Why are you managing your own database connections?
   a) ...
4.27. In my development department, the percentage of the projects making use of NHiberate are
   a) Stateless
      • ... percent
   b) State-full
      • ... percent
4.28. Data transactions are handled by
   a) NHibernate
   b) Database
   c) Other (please specify)
      • ...
4.29. Have you used other OR mappers?
   a) Yes, go to 30
   b) No, go to section 6
4.30. Please specify per OR mapper the pros and cons compared with NHibernate. You can then go to section 6.
   a) ...

5. Other information

5.1. Are you currently using an object persistence framework?
   a) Yes, go to 2
   b) No, go to 4
5.2. Which persistence framework do you use?
   a) ...
5.3. Why do you favour this persistence framework over NHibernate (please list the pros and cons)? If you are ready, go to 5.
   a) ...
5.4. Why are you not using an object persistence framework?
   a) ...
5.5. When will you consider using NHibernate (e.g. if certain features are available, etc.)?
   a) ...

6. Conclusion

Thank you for filling in this survey! If you are interested in the results, just let me know and I can send you a copy of my thesis once it has been successfully completed.

Appendix B

Cuyahoga NHibernate couple points

All the direct couple points with the NHibernate framework related to sessions and transactions.

Type	Assembly	ISession	ITransaction	ISessionFactory	ISessionManager
CommonDao	Core	x			x
CoreRepository	Core	x	x	x	
CoreRepositoryAdapter	Core	x		x	x
DatabaseUtil	Core			x	
ModuleBase	Core	x			x
SessionFactory	Core	x		x	x
SessionFactoryHelper	Core			x	
SiteStructureDao	Core	x			x
UserDao	Core	x			x
LanguageSwitcherModule	Modules	x			
ProfileModule	Modules	x			
SearchInputModule	Modules	x			
SearchModule	Modules	x		x	
SitemapModule	Modules	x			
StaticHtmlModule	Modules	x	x		
UserModule	Modules	x			
ArticleDao	Modules.Articles	x			x
ArticleModule	Modules.Articles	x			
DownloadsModule	Modules.Downloads	x			x
FlashModule	Modules.Flash	x	x		
ForumModule	Modules.Forum	x	x		x
FeedFetcher	Modules.RemoteContent	x	x		
RemoteContentModule	Modules.RemoteContent	x	x		

Appendix C

Session-per-request implementation

A session-per-request implementation in the NHibernate Helper framework. The SessionManager will be injected into the Cuyahoga framework by making use of Castle Windsor (an inversion of control framework).

CuyahogaContainer.cs:

```
/// <summary>
/// The CuyahogaContainer serves as the IoC container for Cuyahoga.
/// </summary>
public class CuyahogaContainer : WindsorContainer {
    /// <summary>
    /// Constructor. The configuration is read from the web.config.
    /// </summary>
    public CuyahogaContainer() : base(new XmlInterpreter()) {
        RegisterFacilities();
        RegisterServices();
        ConfigureLegacySessionFactory();
    }

    private void RegisterFacilities() {
    }

    private void RegisterServices() {
        // The core services are registrated via services.config
        ...
        // Inject the session manager in the Cuyahoga framework
        AddComponent("sessionmanager", typeof(SessionManager));
    }

    private void ConfigureLegacySessionFactory() {
        SessionFactoryHelper sessionFactoryHelper =
            this[typeof(SessionFactoryHelper)] as SessionFactoryHelper;
        sessionFactoryHelper.ConfigureLegacySessionFactory();
    }
}
```

SessionPerRequestModule:

```
public class SessionPerRequestModule : IHttpModule {
    public void Dispose() {
        // No implementation
    }

    public void Init(HttpApplication context) {
        context.BeginRequest += new EventHandler(Context_BeginRequest);
        context.EndRequest += new EventHandler(Context_EndRequest);
    }

    void Context_EndRequest(object sender, EventArgs e) {
        IWindsorContainer container = ContainerAccessorUtil.GetContainer();
        SessionManager sessionManager = (SessionManager)container["sessionmanager"];
        sessionManager.CloseSession();
    }

    void Context_BeginRequest(object sender, EventArgs e) {
        IWindsorContainer container = ContainerAccessorUtil.GetContainer();
        SessionManager sessionManager = (SessionManager)container["sessionmanager"];
        sessionManager.OpenSession();
        HttpContext.Current.Items.Add("SessionManager", sessionManager);
    }
}
```

Appendix D

Data Modeling Standards

There are several data modeling standards and three common diagrams are displayed below. The first picture displays an ORM (Object Role Model) diagram. The second picture displays an ERD (Entity Relation-ship Diagram) diagram. The last picture displays part of a class diagram.