**Factory System Report**

**Contents**

* 1 - System Architecture Research
  + - Web Service Architecture
  + - .Net Remoting
  + - Web services vs .Net Remoting
* - Critical Evaluation of Distributed Computing Technologies
  + - Concurrency Control
    - - Pessimistic Concurrency Control
      * - Locks
      * - Monitors
      * - Reader Writer Locks
    - - Evaluation of Using Pessimistic Concurrency Control
  + - Optimistic Concurrency Control
    - - Evaluation of Optimistic Concurrency Control
  + - Timestamp Ordering
    - - Evaluation of Timestamping
* - Design of the Factory System
  + - System Architecture Chosen
  + - Concurrency Control Chosen

**1 - System Architecture Research**

1.1 - Web Service Architecture

Wiki states that *A* ***Web service*** *(also* ***Web Service****) is defined by the* [*W3C*](http://en.wikipedia.org/wiki/W3C) *as "a software system designed to support* [*interoperable*](http://en.wikipedia.org/wiki/Interoperability)[*machine-to-machine*](http://en.wikipedia.org/wiki/Machine_to_Machine) *interaction over a* [*network*](http://en.wikipedia.org/wiki/Computer_network)*"*.  
What this basically means is that a web service created in C# and on Microsoft Windows XP will be able to function with a client written in a completely different language running on a completely different operating system.  
A web service has the following attributes (Robinson):

* + *Simple form of cross-platform RPC*
  + *Objects hosted by Web server*
  + *Interaction via textual XML messages*
  + *Support use of user-defined types as arguments / return values*
  + *Not designed to be called by end users, only by other applications*

1.2 - .Net Remoting

.Net Remoting is a (Robinson) *generic system for exposing objects to remote process usage*.  
.Net Remoting allows and facilitates (Robinson):

* *.NET allows*
  + *invocation of methods of remote objects*
  + *invocation of properties of remote objects*
  + *subscription to remote events*
* *Facilities:*
  + *remote and local invocation syntax is exactly the same*
  + *naming and other required components are all built into runtime and do not need to be explicitly invoked*
  + *remoting mechanism is highly customizable and provides a powerful and flexible architecture for implementing distributed applications*
  + *more powerful serialization and versioning mechanisms than Java*
  + *proxies generated dynamically by client at runtime; proxies can also be customized*

1.3 - Web services vs .Net Remoting

The downfall ultimately of .Net Remoting, whilst in comparison to Web Services, is that interopability suffers, this is because (Robinson):

* + *Each vendor has its own format (protocol, port) for messages*
  + *There are several RPC models and implementations* 
    - *Windows RPC not the same as the Open Software Foundation’s Distributed Computing Environment, for example*
  + *Difficult to call a procedure on a different OS*

On the other hand, unlike .Net Remoting, Web Services do not offer the following (Robinson):

* + *offer a user interface*
  + *run continuously - objects created on demand (i.e. single call semantics)*
  + *preserve state across calls – stateless components*

**2 - Critical Evaluation of Distributed Computing Technologies**2.1 - Concurrency Control

Any distributed system that allows multiple connections/threads to the server must utilise some form of concurrency control. Concurrency control ensures that the application remains in a valid state and that it is concurrent with the action/(s) that have just been performed on it. E.g. updating a bank account and reading from it at the same time, the concurrency control should ensure the bank account that is been read from has finished updating and vice versa.  
  
2.1.1 - Pessimistic Concurrency Control

Pessimistic Concurrency Control assumes the worst case scenario, that, updates to the internal states will conflict often and that the user will not put up with their transaction/(s) been denied.  
There are numerous pessimistic concurrency controls that are a part of the .Net Framework, and they are:

* Locks
* Monitors
* Reader Writer Locks

2.1.1.1 - Locks

Robinson states that *every object has an associated lock* and that the *lock construct specifies the object on which the lock should be obtained*. Robinson also explains how the locking mechanism works, *first thread to execute the lock statement will be granted the lock and can execute protected section of code. Whilst [the] first thread has the lock, any other threads executing the lock statement will block until [the] lock is released. Waiting threads will then compete for the lock.*  
  
The advantages of using locks are:

* Locking is easily implemented
* Prevents any conflicting updates by ensuring (Robinson)*that one thread does not enter a critical section of code while another thread is in the critical section. If another thread attempts to enter a locked code, it will wait, block, until the object is released.*

The disadvantages of using locks are (Robinson):

* *Locking is essential to preventing conflicting updates but can adversely affect performance*
* *If scope of lock is too great*
  + *forces some threads to wait*
  + *decreases potential concurrency and throughput / performance*
* *If scope of lock is too small*
  + *forces reacquisition of locks, leading increased lock contention and reduced throughput / performance*
  + *can result in difficult-to-debug race conditions*
* Deadlock can occur

2.1.1.2 - Monitors

* Monitors are an advanced form of locking. (Robinson)  
  *Monitor* 
  + *code construct that provides both mutual exclusion and condition synchronization*
  + *acts like a ‘smart’ lock*
  + *Holds ‘ready queue’ of threads which are waiting for the lock*
  + *Holds ‘waiting queue’ of threads which require notification of lock change*
* *Thread enters the monitor and attempts to acquire lock*
  + *if successful, thread executes under mutual exclusion; thread frees lock when finished*
  + *if unsuccessful, thread blocks until monitor is available*

2.1.1.3 - Reader Writer Locks

MSDN states that a reader writer lock is *Defines a lock that supports single writers and multiple readers*To extrapolate further, a readerwriter lock allows for a resource to be read by multiple readers and only allows one writer at any given time. They work by acquiring and releasing locks.

2.1.2 - Evaluation of Using Pessimistic Concurrency Control

Overall the use of Locks is a good idea, since locking will provide the distributed system with concurrency control and will definitely grant mutual exclusion and guard critical sections. However deadlock can occur so it must be implemented with great care and planning.

2.2 - Optimistic Concurrency Control

* Optimistic Concurrency Control (Robinson)*works on the assumption that the likelihood of concurrent access to an object is actually rather low, that transactions proceed as if conflict impossible. At the end of transaction, conflicts are checked for. If conflict occurs, one or more transactions are aborted.*  
    
  *Optimistic Concurrency Control in Detail:  
  Working phase*
  + *Transaction keeps tentative versions (TV) of all objects*
    - *TV is invisible to other transactions*
  + *Reads access TV or last committed version (LCV) of objects*
  + *Writes happen on TVs only*
  + *Records are kept of objects read or written*
* *Validation phase*
  + *Checks to see if conflicts with other transactions have occurred*
  + *If so, no changes are made (abort)*
  + *If not, all changes may be committed*
* *Update phase*
  + *TV is made permanent and committed*
* *Validation and update are usually short in comparison with working phase, so it’s efficient*
  + *If working phase is always short, use of locks may be more sensible*

2.2.1 - Evaluation of Optimistic Concurrency Control

Optimistic Concurrency Control is a great approach to concurrency control if, and only if, conflicts are rare this is because there are no locks such it eradicates the deadlocking issue imposed by locks, otherwise the performance hit will be too great.

2.3 - Timestamp Ordering

Timestamp-based concurrency control is a non-locking form of concurrency control that uses read and write timestamps for each database object and each transaction is given a timestamp when it starts.

The rules of timestamping are:

* *Ta* writes are not permitted if
  + Any object to be written has been read by a transaction with a later timestamp
  + Any object to be written has been written by a transaction with a later timestamp
* *Ta* reads are not permitted if
  + Any object to be read has been written by a transaction with a later timestamp

2.3.1 - Evaluation of Timestamping

Just like optimistic concurrency control, timestamping does not use any locks, therefore deadlock cannot possibly occur. This is a good accolade of timestamping, unfortunately timestamping can be prone to restarts, this is because if a transaction arrives late then it will be aborted.

**3 - Design of the Factory System**

3.1 - System Architecture Chosen

The System Architecture chosen is Web Services. This has been chosen over .Net Remoting because only a simple form of cross-platform RPC is needed and because the objects can be hosted by a web server.

3.2 - Concurrency Control Chosen

The concurrency control chosen is pessimistic concurrency control (locks). This is because locks are simple to implement and they guarantee mutual exclusion and guard critical resources. Even though deadlock can occur great care, planning, and testing has taken place to ensure deadlock won’t occur. Also ReaderWriter locks have been utilised with a none-infinite timeout implementation to help prevent deadlock from happening as well. Also locks have been used because a lot of updates will be occurring (every second or so), whereas if optimistic concurrency control was implemented instead then a severe degradation of performance would occur because of the high volume of updates.

**4- Bibliography/Background References**

Robinson Dr P A, 2009, *08346 lecture slides 08 2008-9.pptx* [online].  
MSDN, 2009, *ReaderWriterLock Class (System.Threading)* [online], Available: http://msdn.microsoft.com/en-us/library/system.threading.readerwriterlock.aspx [Accessed 20th April 2009].  
Wiki, 2009, *Web Service* [online], Available: http://en.wikipedia.org/wiki/Web\_service [Accessed 20th April 2009].