



Vision Document: MSE Project

August 26, 2008

Prepared by Doug Smith
Version 0.2

Table of Contents	
Table of Figures	2
Revision History	3
Introduction.....	4
Context -Enterprise Workflow System	4
Software Architecture: Data Access	5
Software Stack.....	7
Web Caching	8
Caching Content Near Clients.....	8
Caching Content Near Servers.....	8
Application to the Problem Domain.....	9
Caching Solution Dimensions.....	9
Cache Structure and Object Mapping.....	9
Cache Modes	9
Replication Scope	10
Cluster Node Initialization	10
Eviction Policy	10
Group Communication Protocol	10
Object Marshalling.....	11
Transactions and Concurrency	11
References	11
Project Overview.....	12
Goals and Constraints	12
Main Product Features	12
Risks.....	15
Constraints.....	15
Quality Attributes.....	15
External Interfaces	16
Requirements Specification.....	16
Process Metadata Use Cases	16
UC1: Define Process	17
UC2: Retrieve Pools.....	17
UC3: Retrieve Property Definitions.....	17
UC4: Retrieve Swim Lanes	17
UC5: Retrieve Process Names	18
UC6: Retrieve Process Definition	18
Process Execution Use Cases.....	18
UC7: Instantiate Process.....	18
UC8: Update Task	18
UC9: Retrieve Task.....	19
UC10: Retrieve Task List.....	19
UC11: Find Tasks	19

Table of Figures

Figure 1 Scale-Out Deployment Architecture.....	5
Figure 2 Software Stack - Cacheable Data Access	7
Figure 3 System Information Model.....	14
Figure 4 Process Metadata Use Case Model	17
Figure 5 Process Execution Use Case Model.....	18

Revision History

Version	Date	Changes
0.1	2/13/2008	First draft.
0.2	8/26/2008	Additional Hibernate details, web caching overview, reorganization, added specific details of the project.

Introduction

Context -Enterprise Workflow System

The goal of the project is to produce a scalable distributed caching solution applicable to an enterprise workflow system, within the boundaries of producing a software engineering portfolio in partial fulfillment of the KSU masters of software engineering program.

The system the solution is to be applied to is a production workflow application that was recently re-architected. The transactions and data associated with the system can be thought about in terms of defining workflows and executing workflows. Workflow definition involves defining the steps in a process, providing the presentation definitions for each activity in a process, creating the security model around viewing or performing work, specifying the rules that define step transitions, and so on. The definition data is rarely modified in production - typically there will be weekly changes in a tech window to workflow definitions, but after a process has been defined, modifications are rare. Transaction volumes are very low for workflow definitions.

For workflow execution, however, transaction volumes are very high - some instances of the product see sustained volumes of 40TPS, with 5 minute peak volumes up to 80 TPS. Furthermore, we see a roughly 20% yearly increase in volumes, due to both business growth and the implementation of new business processes on the platform. Workflow execution is performed by humans in the front and back offices, as well as from numerous web portals. The application is highly available, and it is likely that business demand will require us to move in the continuous availability space within the next 3 - 5 years. In addition to scalability and reliability requirements, there are hard expectations in terms of transaction response times, for example, to update a work item associated with a process, it would be unacceptable for the response time to increase from 0.5 seconds to 1 second.

To process a workflow execution transaction, workflow definition data must be accessed to validate the transaction request, and to process the transaction against the workflow rules that determine how to route the item, that determine if an event notification must be fired, that determine if data associated with the process instance requires updating, and so on. Given the low volatility of the workflow definition data and high read rates, it is a good candidate for caching in the application server tier to minimize the amount of database access needed to process a transaction. The goal of this project is to leverage a cache to reduce the amount of database access by the workflow application.

To provide the scalability needed for the workflow system, the application servers execute on multiple Java Virtual Machines running on multiple physical and virtual servers.

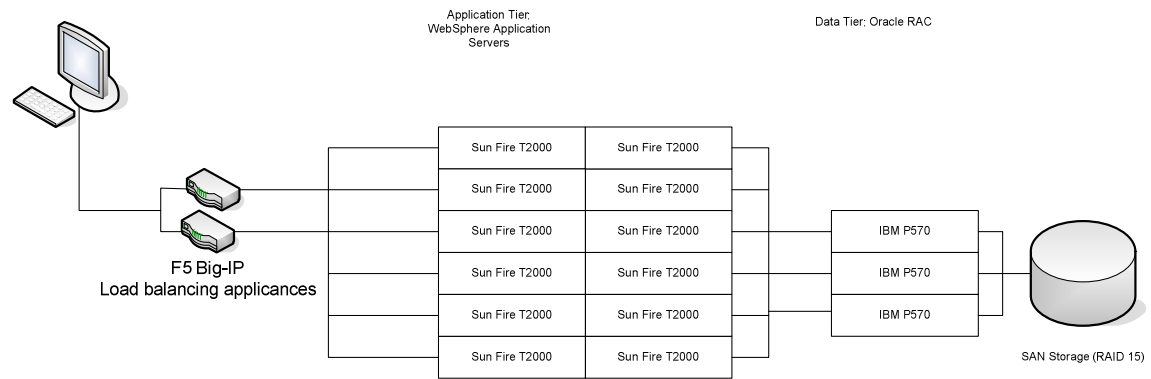


Figure 1 Scale-Out Deployment Architecture

Figure 1 shows the deployment architecture for the application. Requests are routed from clients to servers via load balancing appliances to application servers running on Sun Fire T2000 servers. Clients make web service requests to the application servers, which use the SOAP 1.1 HTTP binding as the messaging and transport framework. The servers connect via JDBC to a clustered Oracle database. In the application server tier, an instance of an application might run on 4 physical servers, with a 4 node WebSphere ND deployment on each physical server, for a total of 16 Java Virtual Machines serving application requests. The 12 application servers provide the capacity needed to run 10 instances of the application.

Notice in Figure 1 the types and numbers of servers in the application server and data tiers. In the application servers, relatively inexpensive servers are used, whereas in cost of the servers in the data tier is much higher. So not only can caching data in the application tier improve performance by eliminating a trip to the data tier, it can also reduce total cost of ownership of the application by using less expensive computing cycles in the application tier, and fewer of the expensive cycles in the data tier.

In addition to reducing the expense of reading data in the application tier for request validation and rule-driven processing, there is also the potential for caching web service results that are formed using workflow metadata. These would be services that clients use to perform client side validation, dynamic presentation, and so on.

The multi-JVM scale out architecture complicates the use of a cache. Since a transaction that modifies workflow metadata is routed to a single JVM in the application tier cluster, the modification must be replicated to caches in the application server tier. Thus, if a data cache is used in each member of a 16 JVM cluster, workflow metadata modified in one cluster member must be replicated to the other 15 cluster members.

Software Architecture: Data Access

The workflow application was written using the Spring framework, with the application business logic and data access code written using plain old Java objects (POJOs). The Data Access Object pattern was used to encapsulate the persistence logic of the application, with Spring managing the life cycle of the objects, and Spring aspects providing transaction management.

Given the low transaction volumes associated with modifying workflow metadata, and the assumption the data could be cached, the Hibernate Object-Relational Mapping (ORM) framework was selected for the ORM technology.

Hibernate has a two-level cache architecture. The first level cache, known as the Session cache, is used to cache objects accessed during a single transaction. Hibernate caches objects in the Session cache to avoid wasted trips to the database. Changes to objects during a transaction are cached in memory, and the Session is flushed at transaction commit time, resulting in changes to the objects being reflected as updates to the database.

The second level cache allows object caching to span transaction lifetimes, providing a JVM level or cluster level cache. When a second level cache is used, the Hibernate session interacts with the second level cache based on configuration. The standard mode of interacting with the second level cache is to read items from and to write items to the second level cache.

The Hibernate framework allows the use of second level cache to be plugged into the framework via configuration files. Through the use of configuration file changes, caching can be introduced transparently to an application: when a second level cache is configured, the data is cached as it is read, and checked for the data on subsequent reads. The JBoss replicated cache is one of the second level cache providers that can be plugged into an application using Hibernate to access data.

Hibernate provides two types of caching: entity caching, and query caching, with the second level cache logically partitioned into entity caches and query caches. Entity caching refers to the caching of individual persistent objects. Additionally, query results can be cached in the Hibernate query cache. Hibernate query caching works as follows:

1. When a query denoted as 'cacheable' is executed, the query is executed against the database.
2. A lookup in the second level cache is performed using the query as the key.
 - a. If there is a cache hit, a collection of object IDs is returned. If the collection of IDs read from the cache is the same as the collection of IDs that are read from the result set, a collection of entities is hydrated based on the cached IDs (with individual entities read from the second level cache).
 - b. If there is no cache hit, the collection of entities is created using the result set (and cached in the entity cache if so configured), and a query result entity id collection is created and stored in the query cache with the query as the key.
3. The collection formed as a result of executing the query is now available for application access.

Perhaps somewhat naively, our group had assumed that we could simply plug in the JBoss replicated cache as our second level cache provider, and realize the benefits of caching without a large investment in software licenses or engineering analysis. When building the system, our assumption was the application would lend itself nicely to a replicated cache solution. However, in scale testing, we found that as the number of Java virtual machines running the application went up, we quickly hit a point where use of the replicated cache actually defeated application scalability, and we had to greatly reduce our use of the cache to get any benefits from it, at the expense of using more compute cycles in the database servers and limiting our headroom for future growth.

Software Stack

Figure 2 shows the software stack associated with access to cacheable data. The top layer represents data access objects written to provide application data access. Hibernate is used as the ORM technology, accessed through the Spring Framework's ORM APIs.

Hibernate defines a pluggable cache provider interface, allowing cache providers to plug in a cache by providing implementations of the Cache and CacheProvider interfaces.

The Cache interface defines a set of operations for interacting with the cache to access data in the cache, add and remove data to/from the cache, lock and unlock items in the cache, and so on. The CacheProvider interface provides an interface to instantiate a cache and to start and stop it.

The Cache interface defines the base capabilities required for caching. Cache strategy classes define algorithms based on the Cache interface for caching solutions with different properties. For a replicated cache that uses transactions, Hibernate defines a class named TransactionalCache that provides the interaction with the Cache implementation that supports transactionally consistent caches. Currently, the TreeCache implementation is the only transactionally consistent cache supported directly via the Hibernate release. Note that configuration parameters associated with a cache provider determines the strategy class used with the cache, with the strategy class implementing the canonical Gang of Four Strategy pattern.

For using the JBoss cache (previously known as TreeCache), the TreeCache and TreeCacheProvider interface implementations provided with the Hibernate release are used. These in turn use the JBoss cache API to as the cache provider implementation.

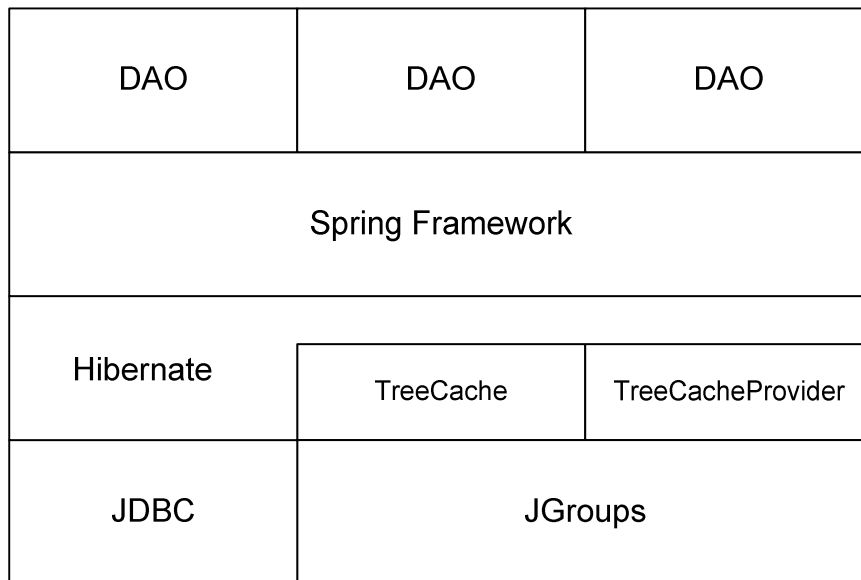


Figure 2 Software Stack - Cacheable Data Access

For cache replication, JGroups is used for communication between cluster members in a distributed JBoss cache configuration. JGroups is a reliable multicast protocol implementation.

Web Caching

Given the distributed nature of the application, and the use of the HTTP protocol in the communication between clients and servers, previous work in web caching should be leveraged in forming a caching solution for the workflow application. At a very high level, web caching strategies can be described in terms of caching data near clients, or near servers.

Caching Content Near Clients

The World Wide Web (the Web) is, according to the Wikipedia, “a system of interlinked hypertext documents accessed via the Internet.” Clients connected to the Internet have access to resources on the Web that are dispersed throughout the globe, and may require several hops between interconnected networks on the Internet to reach referenced content. Given the distributed nature of the network, the unpredictability of resource access, and the desire to minimize latency when accessing resources (and to reduce backbone traffic on the Internet), a common strategy is to cache data close to clients.

A common scheme for doing this is proxy caching. Proxy caching is a scheme where client access to Web resources passes through a proxy server at the edge of their local network. If the resource is cached at the proxy server, it can be returned directly to the client without having to traverse the Internet. If the content is not cached, it can be retrieved and stored at the proxy server prior to returning it to the client to speed access by other clients who may wish to retrieve the same content.

Proxies can also access other proxies, which in general are known as hierarchical caching. This architecture is motivated by the idea that nearer caches can in general satisfy requests faster than far away servers. There has been much work in optimizing the effectiveness of hierarchical caches by having the caches share data, which is known as cooperative proxy caching. Two major design points cooperative proxy caches must consider are location management, which refers to how a proxy determines what other proxies have the content they need, and proxy pruning, which is who to decide whether to obtain content from another proxy or directly from the source.

While conceptually simple, there are many factors that affect the usefulness of caching data near clients. Content type, object size, object popularity, and object volatility influence the degree to which caching improves performance.

Locality of reference, which refers to the ability to predict future access to objects based on past access, must also be considered when weighing the benefits of caching content near clients. There are two dimensions to locality of reference: temporal, which is repeated access to the same object within short periods of time, and spatial, where an access to one object may be followed by accesses to other objects.

Caching Content Near Servers

To offload the serving of static content off origin servers, or to isolate an origin server behind a firewall, a server side cache can be used. These caches are known as reverse proxy caches as they cache content that originates from the servers they front.

To allow origin servers to scale up to meet high demand, content can be replicated to multiple servers acting as reverse proxy caches. This can be done transparently where client requests are

routed to a server containing replicated content. In this arrangement, multicast can be used to ensure changes to content on the origin server are propagated to replicas.

Application to the Problem Domain

In the workflow application, clients cache workflow meta data obtained via web service calls on the client, which is analogous to a browser cache. The current use of the client is either entirely within a private network, or over dedicated VPN access to the internal network. Based on current use of the client and performance of the network, there has not been a need to deploy a proxy cache. This might be re-examined when the client is updated to allow distribution beyond the private network, with clients geographically dispersed and accessing the application from the Internet via a firewall.

On the server side, there is an opportunity to deploy a reverse proxy cache to reduce the load on the database. The purpose of this project is to frame the opportunity as a set of requirements, and to produce a solution. There may be an opportunity to cache at different levels, for example at the database object level to reduce database reads needed to process transactions, to caching web service results based that return infrequently changed workflow metadata.

Caching Solution Dimensions

There are many design decisions and tradeoffs that need to be made when designing a caching strategy and implementation for an application. The design dimensions listed in this section are drawn primarily from those presented as configuration options in the JBoss Cache documentation, but are not unique to that product.

Cache Structure and Object Mapping

The cache structure refers to the structure used to organize the data in the cache. For example, JBoss Cache uses a hierarchical tree structure, with nodes in the tree storing data in a hash map.

Object mapping refers to the mapping of an object to a location in the cache structure. The Hibernate transaction caching strategy implementation maps a Java object to a location in the tree cache structure using the package name specification as the node path to the hash map where instances of the class are cached.

Cache Modes

The following cache modes can be leveraged in a distributed system.

- *Local mode.* In this mode a cache is not shared among nodes in a cluster. Changes made to a local cache are not replicated to other nodes in cache.
- *Synchronous replication mode.* In this mode, changes to the state of the cache are made synchronously among the nodes in the cache cluster (loosely defined as the set of JVMs between which the state is replicated). A two phase commit protocol is executed to ensure all or nothing semantics for changes to cache state.
- *Asynchronous replication mode.* Similar to synchronous replication, except instead of a two phase prepare-commit transaction where failure to commit by any cluster member aborts the transaction, a message from the node changing cache state is sent to all other cluster members asynchronously when the transaction initiating the change commits. If

an error occurs in a cluster member processing the asynchronous data update, the error is simply logged, and processing continues with cluster members having inconsistent views of the cache state.

- *Synchronous invalidation mode.* In this mode, changes to cache state are not replicated between cache members, but instead invalidation messages are sent to each cache member as part of a two phase commit protocol specifying the data the transaction has invalidated. Cluster members evict invalidated data from their copy of the cache, and synchronous invalidation ensures all members in the cluster evict invalidated data as part of the distributed transaction.
- *Asynchronous invalidation mode.* Data invalidation messages are sent asynchronous at transaction commit time from the cluster member where data was changed to all other cluster members when the originating transaction is committed. If an error occurs in a cluster member processing the invalidation message, the error is simply logged, and processing continues with cluster members having inconsistent views of the cache state.

Replication Scope

For non-local cache modes, the scope of the replication can be configured, usually in conjunction with an application data partitioning strategy. This can vary from a simple strategy that replicates all data to all nodes, to having certain data owned by certain nodes (and optionally backed up to provide availability should a node failure) to fully dynamic strategies that partition data dynamically as the cache grows. Somewhere between the simple replication strategy and full dynamic partitioning, the solution moves from the distributed caching space to the data grid space.

Cluster Node Initialization

When a node is added to the cache cluster, a strategy for initializing the state of the cluster member is needed. Strategies run from a lazy load on demand strategy where data is added to the new cache node as data is read through it, to locking the entire cache and copying the current state of the cache to the new cluster member.

Eviction Policy

Most applications cannot assume an infinite growth model for a cache. In the absence of an infinite amount of memory for the cache (or sufficient memory to cache all the data in an application), some sort of eviction policy is needed to determine what is removed from the cache (and when it is removed) to make room for additional data. Standard eviction policies are least recently used, least frequently used, first in first out, by size, by time added, and so on.

On eviction, the cache implementation can either discard the cache entry, or can passivate the evicted entry to the file system or another data store, which presumably can be accessed faster than the database the cache is fronting, should access to the evicted data be needed.

Group Communication Protocol

The group communication protocol is the networking protocol used for communication between the cache nodes in the replicated, distributed cache. JBoss Cache uses the JGroups protocol for reliable multicast communication. There are numerous tuning options exposed via JGroups

configuration that influence the performance, scalability, and reliability of the JBoss cache running on top of JGroups.

Object Marshalling

Object marshalling refers to how objects are put on the wire and taken off the wire when being shipped between cluster members as part of replication. The efficiency of object marshalling directly influences the performance and scalability of a distributed cache.

Transactions and Concurrency

For synchronous replication and invalidation, the level of data isolation and associated locking is typically configurable. The standard isolation levels are supported by JBoss Cache, e.g. no isolation, read uncommitted, read committed, repeatable read, and serializable. The standard engineering tradeoff of performance versus data consistency is in play here.

In JBoss Cache, specifying transaction isolation levels implies the use of a pessimistic locking strategy. To boost concurrency, an optimistic locking solution is also provided, where instead of locking data, data is accessed as needed, and at transaction commit time, modified data is broadcast to the cluster. If the data had been modified by another cluster member while the transaction was in flight, the transaction can be aborted via the response of the modifying member.

References

- [1] Data Access Object, Core J2EE Patterns Catalog, Sun Microsystems, <http://java.sun.com/blueprints/corej2eepatterns/Patterns/index.html> (last accessed 8/12/2008)
- [2] Hibernate Reference Documentation, Version: 3.2.2. Hibernate Project, http://www.hibernate.org/hib_docs/v3/reference/en/pdf/hibernate_reference.pdf (last accessed 8/12/2008).
- [3] The Spring Framework – Reference Documentation, Version 2.5, Rod Johnson et al, <http://static.springframework.org/spring/docs/2.5.x/spring-reference.pdf> (last accessed 8/12/2008).
- [4] Reliable Multicasting with the JGroups Toolkit, Bela Ban, <http://www.jgroups.org/javagroupsnew/docs/manual/html/index.html> (last accessed 8/22).
- [5] World Wide Web, Wikipedia.org, http://en.wikipedia.org/wiki/World_wide_web (last accessed 8/22)
- [6] Design Patterns: Elements of Reusable Object-Oriented Software, Erich Gamma et. al., Addison Wesley, 1995.
- [7] World Wide Web Caching: Trends and Techniques, Greg Barish and Katia Obraczka, IEEE Communications Magazine, vol. 38, pp. 178-184, May 2000.
- [8] Comparing Strength of Locality of Reference – Popularity, Majorization, and Some Folk Theorems, Sarut Vanichpun, Armand M. Makowski, IEEE Infocom, 2004

[9] Web Caching and Replication, Michael Rabinovich and Oliver Spatscheck, Addison Wesley, 2002.

[10] Business Process Modeling Notation, V1.1, Object Management Group, January 2008.
Available from <http://www.bpmn.org>

Project Overview

Goals and Constraints

The goals of the project are to:

- Develop a simple process execution engine with transaction characteristics similar to the enterprise workflow system discussed earlier.
- Use a distributed cache in the process execution engine to ensure the system can scale up without the reading of process execution metadata from the database inhibiting scalability.
- Constrain the implementation of the process execution engine such that the distributed caching solution produced for this project is applicable to the enterprise workflow system. In other words, it should be reasonable to expect the caching solution produced in this project to be leveraged by the enterprise workflow system.

Main Product Features

This section provides an overview of the main product features for the sample application used to derive a scalable caching solution.

In deriving the product features, the performance critical transactions and their use of cached data (or potential use of cached data) was analyzed.

Prior to releasing updates to the enterprise workflow system, a suite of scale tests that exercise the performance critical transactions of the system is executed. This involves running a representative mix of transactions at 40 TPS, and measuring the response times of the transactions. System resource utilization is also measured. Release of an update into production requires response times are maintained or improved, and no significant changes to resource utilization are observed.

Looking at the performance critical transactions, they can be classified as one of the following transaction types:











1. *Workflow metadata retrieval.* Workflow clients need to retrieve workflow metadata to render the presentation of work items, to determine the set of data associated with an item, and to apply constraints to the editing of data associated with the workitems.
2. *Work processing referencing metadata.* When workflow clients submit requests to the server to create or update work items, the server must read work item metadata to ensure the transaction requests are valid with respect to the process metadata the work item is associated with.
3. *Meta-data based process instance searches.* Workflow services that search for process instances or retrieve lists of work have metadata associated with them that affect how the search is constructed and the data returned in the results.

To provide functionality that covers these three categories of transactions and to keep it aligned with the workflow domain, this project will produce a simple process execution engine. The process execution engine will allow the definitions of processes to be stored as process metadata, and will provide services to instantiate and execute process instances, as well as to be able to form simple work lists and provide basic search capability.

The process definition will be based on Business Process Modeling Notation (BPMN). BPMN is a set of symbols and loose semantics for defining business processes. To manage the scope of this project, a small subset of BPMN will be supported, specifically that needed to provide for sequential and parallel process tasks. The process execution engine will provide the ability to express and execute process instances defined using the supported subset of BPMN.

Table 1 shows the subset of BPMN in scope for this project:

Table 1 BPMN Subset Used in Project

Symbol Name	Description	Notation
Start event	Denotes start of process.	
End Event	Denotes end of process.	
Pool	Represents a participant in a process, or an organizational or departmental boundary.	
Swim lanes	Used to partition a pool, and provides a means to organize tasks by department, tasks types, etc.	
Activity	Represents work that is done at a step in a process.	
Exclusive (XOR) Gateway	Used for exclusive merge and decision.	
Parallel Gateway (AND)	Used for parallel merge and decision, aka fork and join.	
Normal Flow	Used to indicate sequence of activities.	
Conditional Flow	Flow with a conditional expression evaluated at runtime to determine if the flow should be followed.	
Default Flow	Used as the default flow if none of the expressions associated with the conditional flow are matched.	

In addition to the BPMN notation described above, the system will also support the specification of data associated with tasks and processes – the model expresses the tasks and flow logic associated with the process, with process decision points using data associated with the tasks and process to evaluate decision expressions. In BPMN, data is associated with process definitions and task definitions as property sets, with process properties visible to all tasks in the process definition, and task properties visible at the individual task defining the properties.

The following diagram presents an information model that incorporates the subset of BPMN shown above.

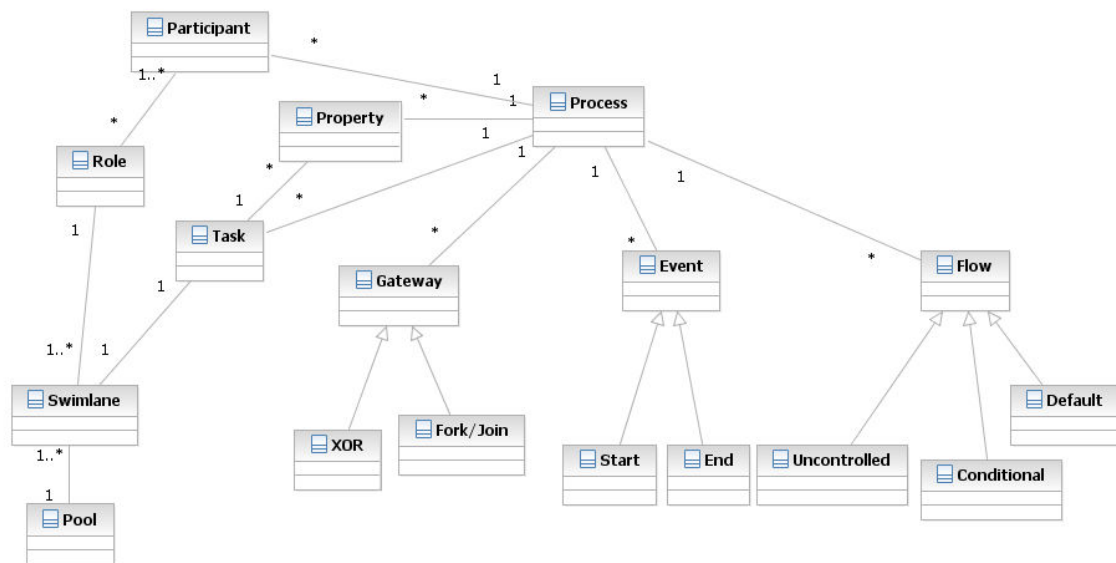


Figure 3 System Information Model

Based on the BPMN subset and associated information model, we can compare the use cases associated with this project with the performance critical transactions to determine if equivalent functionality is being provided in terms of high volume transactions reading relatively non-volatile metadata.

The following table lists the performance critical transactions in the workflow system, the transaction type as defined above, and the use case providing similar functionality covered by this vision document. Note that in some cases there is no analogous use case listed in the table – this is because the transaction type is covered by other use cases and there is no additional benefit to providing an analogue in this project.

Workflow Transaction	Transaction Type	Analogous Use Case
Retrieve Field Definition	Workflow metadata retrieval	Retrieve Properties
Retrieve Queue List	Workflow metadata retrieval	Retrieve Swim Lanes
Retrieve Country List	Workflow metadata retrieval	
Retrieve Item Type List	Workflow metadata retrieval	Retrieve Process Names
Retrieve Nodes	Workflow metadata retrieval	Retrieve Pools
Retrieve Work Item Rule	Workflow metadata retrieval	Retrieve Process Definition
Retrieve Subtype List	Workflow metadata retrieval	
Create Work Item	Work processing referencing metadata	Instantiate Process
Transfer Work Item	Work processing referencing metadata	
Update Work Item	Work processing referencing metadata	Update Task
Add Note	Work processing referencing metadata	
Retrieve Work Item	Work processing referencing metadata	Retrieve Task

Workflow Transaction	Transaction Type	Analogous Use Case
Retrieve Work List	Work processing referencing metadata	Retrieve Task List
Work Item Search	Meta-data based process instance searches	Find Tasks

Risks

The following table summarizes the risks associated with this project. Probability is on a scale of 1 – 10, where one means unlikely, and 10 means guaranteed to happen. Impact is also on a scale from 1 – 10, where 1 means low or insignificant impact, and 10 means severe impact.

Risk	Probability	Impact	Strategy
Inability to procure sufficient hardware to characterize the caching solution at the hardware scale the solution is targeted towards.	6	6	Make solution hardware and O/S agnostic. Choose common components that can be leveraged across a wide variety of platforms. Investigate using on demand computing facilities such as Amazon Elastic Compute Cloud.
Incompatibility of latest Hibernate and JBoss Cache versions may require using older versions of software, which might produce results that are not relevant when latest versions are compatible.	10	5	Track progress of versions, determine if timelines will allow incorporating a converged solution at some point in the project lifetime. Consider writing a Hibernate cache provider using the latest version of JBoss Cache.
Caching solution produced in context of sample application may produce results not applicable to target application.	4	5	Constrain the software components and architecture patterns to mimic the target system.

Constraints

The goals must be accomplished subject to the following constraints:

- The solution must be accomplished using existing Open Source software components, specifically via the use of Spring, Hibernate, JBoss Cache, and the JGroups protocol.
- Modifications to Open Source components needed to accomplish the goal must be submitted back to the appropriate Open Source community.
- The architecture of the solution must be similar enough to the workflow system to ensure the results of this work can be readily applied to the workflow system.

Quality Attributes

- QA1 - The solution may not degrade average response time or throughput for transactions involving cached data.
- QA2 - Functional correctness must be maintained when caching is enabled.

- QA3 - Application availability must not be compromised by the caching solution. The solution must provide the ability to survive failure of cluster members without compromising application availability or system correctness.
- QA4 - The solution shall provide a role-based security model to constrain access to the functions and data associated with the system based on roles.
- QA5 – the solution shall provide basic authentication and identity management, where users are required to authenticate prior to accessing the system, and user identities associated with one or more roles.
- QA6 – the solution must be scalable to 16 JVMs evenly distributed among 4 physically distinct servers, and reduces database requests for workflow metadata by at least 50%.
- QA7 - data access must be consistent across all JVMs serving application requests. It is unacceptable for results produced by the application to differ based on the server or JVM servicing the application request. In other words the view of the data across all cluster members must be consistent.
- QA8 - the solution must provide the ability to add and remove cluster members as needed without causing errors.

External Interfaces

- Management facilities for configuring and monitoring the cache while it is in operation shall be built using Java Management Extensions (JMX).
- The functionality of the process execution engine shall be exposed via web services. No user interface will be provided as part of this project.

Requirements Specification

The use cases listed in this section define the functionality of the system to be delivered. These have been broken down into two classes of use cases: process metadata use cases, and process execution use cases.

Process Metadata Use Cases

The following diagram shows the process metadata use cases.

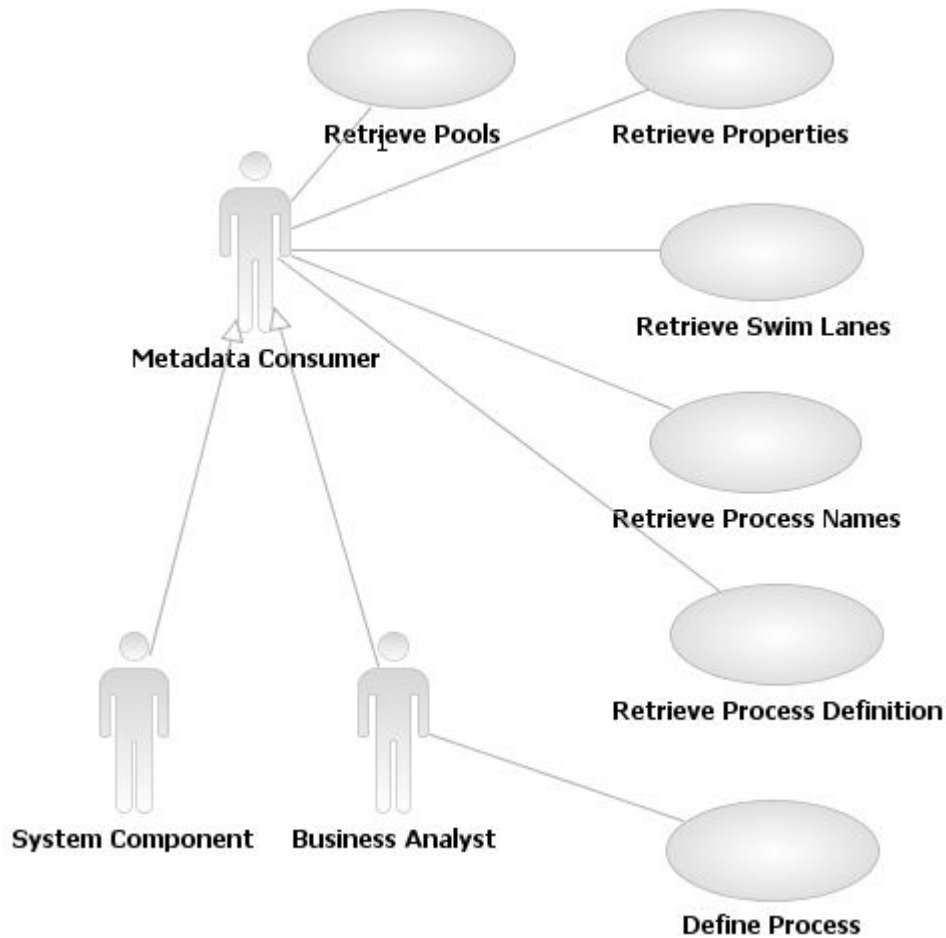


Figure 4 Process Metadata Use Case Model

UC1: Define Process

Provide the ability to define a process in sufficient detail to allow execution of the process. This includes the ability to define all the related information as shown in the information model in Figure 3.

UC2: Retrieve Pools

Provide the ability to retrieve all pools defined in the system.

UC3: Retrieve Property Definitions

Provide the ability to retrieve all property definitions in the system for a given process or a given task.

UC4: Retrieve Swim Lanes

Provide the ability to retrieve the swim lanes defined in the system for a specific pool.

UC5: Retrieve Process Names

Provide the ability to retrieve the names of all process definitions in the system.

UC6: Retrieve Process Definition

Provide the ability to retrieve a process definition for a specified process name.

Process Execution Use Cases

The following figure shows the process execution use cases:

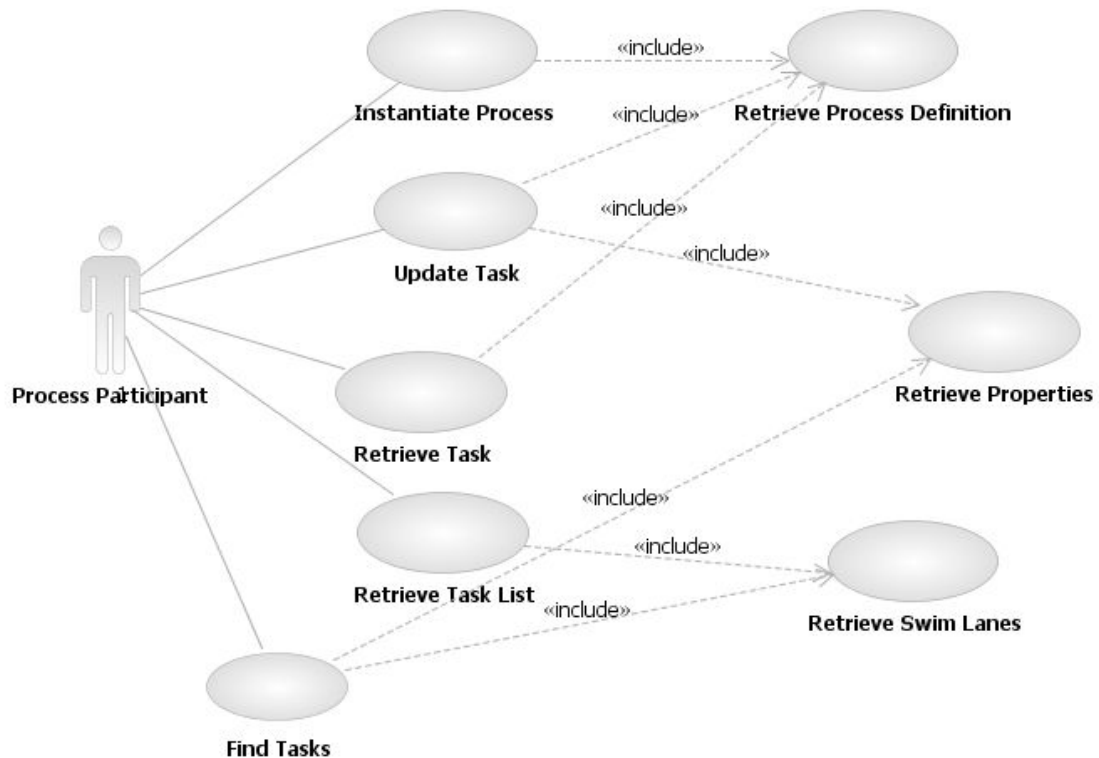


Figure 5 Process Execution Use Case Model

UC7: Instantiate Process

Provide the ability to instantiate an instance of a named process in the process execution environment.

UC8: Update Task

Provide the ability to update a task associated with a process instance, including the setting or modification of task or process properties, as well as indicating the completion of a task.

UC9: Retrieve Task

Provide the ability to retrieve a specified task associated with a specific process instance.

UC10: Retrieve Task List

Provide the ability to retrieve a list of tasks (and their associated process instances) in a given swim lane for active processes.

UC11: Find Tasks

Provide the ability to find tasks that match criteria based on process definitions; namely property values.