Tareas:

* **\* El Read model Rebuilder.**
* El event store rebuilder: Debe ir tomando los mensajes de a uno, e ir incluyendo todo en repositorios en memoria del eventStore y del message log. Luego debe hacer un super save changes. Debe haber connections strings para el messagelog de destino.
* El eventstore db context debe tener un command bus y event bus en memoria, para poder hacer save changes. Esto debe tener el event store. Si esto funciona, quizás ya pueda hacer, por primera vez, un cqrs SIN DEPENDENCIA DE UN BUS EJECUTANDOSE EN segundo plano. Esto seria excelente.
* Las estadísticas del procesamiento de eventos.
* Ahora el Rebuilder ya tiene event store. Hay que guardar los primeros eventos en el event store, para poder recuperarlos después (esto ayudaría a serializarlo y guardarlo en orden en la base de datos)

Feature Requests

**Que mas faltaria**

1. Que haya una politica de reintento en el worker. A medida que se vaya reintentando va a tardar mas en volver a reintentar, asi como se hace el poll del read model de parte del cliente. Configurable
2. Tambien la política de reintento del event handler, y ver la política de reintento del command handler, que sea incremental. Verificar si esto no es lo mismo que en el punto 1.
3. El TimerTaskFactory de Conference CQRS Journey se podría usar como un poll especifico para ver si se cumple una condición para asi saber que el evento se proceso correctamente. Esta muy buena la implementación. Se podría usar en paralelo a la implementación que hice yo.

Puntos de entrada para implementar un dominio con el Journey Framework:

1. Insertar el domain container en esta sección:
2. 

Arquitectura de un dominio. Assemblies = Layers

1. Domain: Aquí se encuentran los handlers y los aggregates
2. Domain.Commands: Aquí se encuentran los commands. Es el único lugar donde debería conectarse con la capa de aplicación.

Entity Framework Query Optimization

martes, 7 de abril de 2015

11:43

En la capa de aplicación se podrian crear los commands. Quizás allí ejecutar las primeras validaciones de que todo esté bien.

La capa de aplicación contiene todos los objetos de de read model, por ejemplo, y tambien para enviar commands. Esto ayudaria a limpiar los controller, hacer que los command senders sean testeables y esten libres de inyecciones por todos lados, o sea, de interfaces que se repiten constantemente. Mejor es inyectarle una unico objeto de "Aplicación" (DDD).

El disable de entity framework para que sea más rápido.

Machine generated alternative text:
PROJECT 
ocøuG 
TEAM 
TEST 
ANALYZE 
public MerpContext 
public Database() 
this . Context 
Context get; private set; 
new Merpcontext(); 
this .Context. Configuration.AutoDetectChangesEnab1ed false; 
*region IDatabase Members 
public IQueryablecActivityType) ActivityTypes 
get return Context. ActivityTypes; 
public IQueryabIe«BusinesslJnit) BusinesslJnits 
get return Context. BusinessUnits; ) 
31:31 / 

Message Receiver.

Ahora se pueden ver como dos transcciones de vida larga en la base de datos a los procesos que miran las tablas del bus:



SOBRE PORQUE NO USO BUS EN MEMORIA – 3 RAZONES

1. Se pierde la capacidad de programar tareas (poniendo fecha de procesamiento del mensaje en el bus).
2. Se pierde la capacidad de ejecutar largos procesos de análisis de eventos. Si fuera en memoria se tendría que ejecutar todo primero antes de mostrarle cosas al usuario. Esto quizás si no fuera muy largo sería rapidísimo.
3. Es más difícil el multiusuario (cada vez que se hace commit ya puede un usuario ganar un turno en el event store)