Introduction to the Microsoft Sync Framework

Michael Clark
Development Manager
Microsoft
Agenda

• Why Is Sync both Interesting and Hard

• Sync Framework Overview

• Using the Sync Framework

• Future Directions

• Summary
Why Is Sync Important

• Computing Device Proliferation
  – More Common Daily Access To ...
    • Multiple PCs
    • Devices
    • Services

• Software + Services
  – Improve User Experience
  – Improve Network Utilization
  – Better Availability/Offline Usage
Why Is Sync Hard

- Local Change Detection
- Change Enumeration
  - Avoiding Reflecting Changes
- Conflict Detection/Resolution
- Efficiently Handling Interruption & Restart
- Managing Deleted Items
  - Correct Cleanup of Tombstones
- Synchronization Loops
- ...

Microsoft Sync Framework
Agenda

• Why Is Sync both Interesting and Hard

• Sync Framework Overview

• Using the Sync Framework

• Future Directions

• Summary
What Is The Sync Framework?

- **Metadata**
  - Handles Common Cases Efficiently
  - Correctly Handles Corner Cases
  - Useable in Any Topology
  - Can be Used to Bridge Multiple Solutions

- **Platform**
  - Supports Low Level Use Of the Metadata
  - Provider Model to Abstract Interaction Between Stores
  - Provides ‘Make it Simple’ Services
  - Factored to Enable Expansion

- **Infrastructure**
  - Common Stores
  - Common Protocols
  - Server & Services Integration
How to Use The Sync Framework

• Write **Sync Applications** to synchronize stores
  – Using other people’s or your own providers

• Write **Providers** for your stores and apps
  – Using the Framework’s Sync Runtime
  – Choose your balance of performance vs. complexity
Example Synchronization Session

Sync Application

Sync Orchestrator

Sync Provider

Data Store

Sync Runtime

Meta-data Interpretation Tools

Provider Services

MD Store

Data Store

Microsoft Sync Framework
Example Remote Sync Session

Sync Application

Sync Orchestrator

Sync Provider

Provider Proxy

Provider Stub

Local Store

Remote Store

Microsoft Sync Framework
• The framework does **not mandate** a data model
  – Just a few concepts that can be mapped to most models

• **Sync Scope**: the set of objects being synchronized across a set of partners

• **Change unit**: granularity of change tracking in a store
  – Granularity of change propagation: only changed units need be sent
  – Granularity of conflict detection: independent changes to the same change unit are a conflict

• **Consistency unit**: granularity of consistency
  – All changes within the same consistency unit are sent together
  – Thus, sync can never be interrupted with part of a consistency unit applied
Basic Metadata Concepts

- Peers make changes independently
- Synchronization: making peers aware of all changes

- Each change has a globally-unique version

- Fundamental concept: **Knowledge**
  - A “concise” description of the set of changes that a peer is aware of
  - Knowledge is portable
    - knowledge specification can be understood by any peer
    - not pair-specific: not “what I have received from you”
  - Main operations on knowledge
    - Test if a given knowledge covers a given change
    - Add one piece of knowledge to another to produce combined knowledge

- Each replica maintains its own “knowledge”
Incremental Sync using Knowledge

1. **RequestChanges**: supply your knowledge

2. **Enumerate Changes**: Is my version covered by your knowledge? If not, send.

3. **ConveyChanges**: send along
   - Version of the change
   - Enumerator’s knowledge
     - what the peer making the change knew when he made it
     - what the recipient will learn by applying this change

4. **Apply Changes**: Conflict detection algorithm
   - Is your version covered by my knowledge? If not, you have a conflict
Basic Metadata Details

• Version
  – The ID of the replica making the change + replica-specific number
    • Replica IDs are GUIDs
    • Replica-specific number is ever-increasing at the replica

• Clock vector: X4 Y3 Z7
  – A set of (replica GUID, replica-specific number) pairs
  – Semantics: “all versions authored by this replica up to this number”
  – The simplest example of knowledge
    • Gets more complex as failures, interruptions and such occur
    • But quiesces to the simple form
Platform

• Sync Orchestration Between Providers
  – Simple Interaction for Applications

• Implementation of Core Metadata Services
  – Knowledge management interfaces
    • Learning new things: $K_{\text{new}} = K_{\text{old}} + K_{\text{learned}}$
  – Version-to-knowledge comparisons: $v \leq K$
    • Change enumeration assistance
    • Conflict detection
  – Tombstone management, filtering, fidelity management, much more

• Core services are platform, storage, and data-type-independent
  – Applicable regardless of protocol being used
  – Unmanaged implementation for device portability
  – Convenient managed wrappers

• ‘Make it Simple’ Services
  – Support for Change Application
  – Metadata Storage
Some Infrastructure We Provide

- **File Sync Provider**
  - Useable on FAT as well as NTFS Filesystems

- **Relational Sync Provider**
  - Supporting any ADO.Net Enabled Database

- **Feedsync**
  - Produce or Consume Feeds in RSS or ATOM

- **SyncToy**
  - Useful UI for Configuring Filesync Partnerships

- **Other**
Agenda

• Why Is Sync both Interesting and Hard

• Sync Framework Overview

• Using the Sync Framework

• Future Directions

• Summary
Public Class MySyncController

    Public Sub SynchronizeFolders()
        Dim SyncOrchestrator As New SyncOrchestrator
        Dim LocalProvider As New FileSyncProvider(mySourceReplicaId, _
            "c:\folder1")
        Dim RemoteProvider As New FileSyncProvider(myDestinationReplicaId, _
            "d:\folder2")

        With SyncOrchestrator
            .LocalProvider = LocalProvider
            .RemoteProvider = RemoteProvider
            .Synchronize()
        End With
    End Sub

End Class

Microsoft Sync Framework
Recall: The Sync Session

Sync Application

Sync Orchestrator

Sync Provider

Sync Provider

Data Store

Data Store

Microsoft Sync Framework

Sync Runtime

Meta-data Interpretation Tools

Provider Services

MD Store
Microsoft Sync Framework

Provider Interactions

Orchestrator

GetSyncBatchParameters

Provider Code

Metadata

Knowledge
A15,B37,C8

Versions
I: A12
J: A9
L: B25
M: C8

Sync Runtime

Data

read
Microsoft Sync Framework

Provider Interactions

GetChangeBatch(Kd)

Orchestrator

Provider Code

Sync Runtime

Data

Metadata

Knowledge
A15, B37, C8

Versions
I: A12
J: A9
L: B25
M: C8

minLocalTick(Kd)?
isContained in Kd?
AddToBatch

ChangeBatch

read

read
Provider Interactions

Microsoft Sync Framework
Metadata store

- Internal Databsae storage for provider’s metadata
  - Knowledge, versions, tombstones, etc

- Extremely useful for those who can’t store metadata in their store
  - E.g. FAT

- Makes it easy to write “maintaining providers”:
  - Whenever you detect a change, tell metadata store
    - It will update the metadata (new version, tombstone)
    - Can happen in notifications, or during sync
  - Change enumeration is taken care of
    - You just read the data from the store
  - Change application is largely taken care of
    - You just write the data to the store and forward the calls

Microsoft Sync Framework
public override ChangeBatch GetChangeBatch(
    uint batchSize,
    SyncKnowledge destinationKnowledge,
    out object changeDataRetriever)
{
    ChangeBatch batch = _metadata.GetChangeBatch(batchSize,
        destinationKnowledge);
    changeDataRetriever = this;  // this is where the transfer
    // mechanism/protocol would go.
    // For an in memory provider,
    // this is sufficient

    return batch;
}
public virtual void SaveItemChange(SaveChangeAction saveAction, ItemChange change, SaveChangeContext saveChangeContext)
{
    ...
    switch (saveAction)
    {
    case SaveChangeAction.Create:
        LocalId localId = MyStore.CreateItem(saveChangeContext.ChangeData);
        im = replicaMetadata.CreateItemMetadata(change.ItemId, localId);
        im.CreationVersion = change.CreationVersion;
        break;
    case SaveChangeAction.UpdateVersionAndData:
        im = replicaMetadata.FindItemMetadataById(change.ItemId);
        MyStore.Update(im.LocalId, saveChangeContext.ChangeData);
        break;
    case SaveChangeAction.DeleteAndStoreTombstone:
        im = replicaMetadata.FindItemMetadataById(change.ItemId);
        im.IsDeleted = true;
        MyStore.Delete(im.LocalId);
        break;
    }
    replicaMetadata.SaveItemMetadata(im);
}
FeedSync Support

- **FeedSync** (previously known as SSE) is a set of extensions to RSS and ATOM to
  - Enables bi-directional multi-master synchronization
  - Spec publicly available on MSDN
  - Intended to provide interoperability for Web Service synchronization

- **FeedSync** metadata is fully compatible with Sync Framework

- **Sync Framework** includes built-in support for generating and consuming **FeedSync** feeds
  - Publish and consume feeds by pulling and pushing SSE feeds to their provider
void PublishAllItems(MySyncProvider provider,
    FeedIdConverter idConverter,
    FeedItemConverter itemConverter,
    Stream feedStream)
{
    FeedProducer feedProducer = new FeedProducer(provider,
        idConverter,
        itemConverter);

    feedProducer.ProduceFeed(feedStream);
}
Agenda

• Why Is Sync both Interesting and Hard
• Sync Framework Overview
• Using the Sync Framework
• Future Directions
• Summary
Simple Ways to Write Providers

• Build on Existing Make it Simple Services

• Higher Level Abstraction Model
  – Provider Focuses on Interacting with the Store
  – How to Create Read Update and Delete data
  – How to Perform Local Change Detection
    • Fast Anchor Based Detection for Stores Supporting One
    • Really Simple Change Detection Based on Enumerating Contents if no Other Method is Available
Enhanced Filtering

• Support for Filtered Replicas
  – This replica only ever syncs and stores a subset of the data

• More Tools for Rolling Window Scenarios
  – Just Keep the Upcoming 2 weeks of calendar appointments

• Nearly Always Filtered Syncs
  – Adds even more flexibility to the concept of synchronization scope
Fidelity

• Sync between Stores with different schemas
  – Endpoint 1 stores 3 email addresses for a contact and syncs with Endpoint 2 that only stores 1 email address
  – Endpoint 1 only stores 45 characters for name field and syncs with Endpoint 2 that only stores 30

• Parts of the data may be transformed as part of the sync operation to be more useful on the destination
  – Pictures are converted to 640x480 when transferred to a device
  – Video is stored in a low bit-rate codec

• How can we make this work within an arbitrary topology
  – Challenge to record the loss of information if changes are passed through nodes with lower fidelity
Current and Future Offline Support

• Sync Services for ADO.Net (aka OCS)
  – Take any ADO.Net Enabled Database Offline
  – Work Locally Against Cached Data in SQL CE
  – Sync Data Back to Central Store

• ADO.Net Data Services (aka Astoria)
  – Extend Support to Stores Utilizing ADO.Net Data Services REST Style Data Access

• SQL Server Data Services
  – Take Cloud Backed Data Services Offline

• Others?
Agenda

• Why Is Sync both Interesting and Hard

• Sync Framework Overview

• Using the Sync Framework

• Future Directions

• Summary
Layering

Applications

OCS Frameworks and APIs
- Sync for ADO.NET
- Astoria offline
- More offline?

Protocol and Store Providers
- FeedSync
- File Provider
- Relational Provider
- Astoria Provider
- Third-party providers

“Make-it-simple” Services
- Metadata Store
- Anchor Providers
- “Simple” Providers

“Better-together”
- SQL Change Tracking

Common Sync Metadata and Runtime
- Knowledge
- Versions
- Change Enumeration
- Conflict Detection
Flexibility: varying entry points

- “I need to cache my (service) data offline”
  - Put your cache in SQL, use OCS

- “No, I need to sync particular stores”
  - Use Sync Providers for those stores
  - Use Orchestrator to orchestrate

- “But how do I communicate my changes remotely?”
  - Use Harmonica FeedSync support to generate and consume feeds
  - Alternatively, extend or create your own protocol

- “But there is no provider for this store”
  - Write one easily using Metadata Store (on SQL-CE) and Simple Provider models

- “I need better performance and integration”
  - Using SQL? Use SQL 2008 Change Tracking to make it simple
  - Use Knowledge Services to store metadata yourself
Conclusion

- V1 Shipped Sept 2008
  - Available for Download on MSDN

- V2 Currently Being Built
  - More Details Will be Coming out in the Next Year

- [http://msdn.com/sync](http://msdn.com/sync)
  - Downloads
  - Forums
  - Sample Code