Autonomous Computing

Fiefdoms and Emissaries
Outline

- Introduction
- Autonomous Computing: The Basics
- Working With Autonomous Fiefdoms
- Working Across Fiefdoms
- Conclusion
The following page has definitions excerpted from Bookshelf 99

ex·cerpt  /ekˈsɜrp/ noun
A passage or segment taken from a longer work, such as a literary or musical composition, a document, or a film.

Excerpted from The American Heritage Dictionary of the English Language, Third Edition Copyright © 1992 by Houghton Mifflin Company. Electronic version licensed from Lernout & Hauspie Speech Products N.V., further reproduction and distribution restricted in accordance with the Copyright Law of the United States. All rights reserved.
## Terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autonomous</td>
<td>Not controlled by others or by outside forces; independent</td>
</tr>
<tr>
<td>Fiefdom</td>
<td>1. The estate or domain of a feudal lord. 2. Something over which one dominant person or group exercises control</td>
</tr>
<tr>
<td>Emissary</td>
<td>An agent sent on a mission to represent or advance the interests of another.</td>
</tr>
<tr>
<td>Snapshot</td>
<td>A photograph taken with a small hand-held camera.</td>
</tr>
<tr>
<td>Stable</td>
<td>Resistant to change of position or condition; steadfast. Immutable; permanent; enduring. A place where horses live.</td>
</tr>
<tr>
<td>Uncertainty</td>
<td>Unsure about someone or something. A lack of assurance or conviction.</td>
</tr>
<tr>
<td>Idem</td>
<td>Something that has been mentioned previously; the same.</td>
</tr>
<tr>
<td>Potent</td>
<td>Possessing inner or physical strength; powerful.</td>
</tr>
</tbody>
</table>
What Is Autonomous Computing?

- Autonomous Computing starts with independent computer systems
  - These are independently controlled and managed
  - They don’t trust outsiders

- We are going to examine the consequences of this independence
  - How do such machines interact?
  - How is data handled?
  - How can we accomplish work in such an environment?
Outline

- Introduction

- Autonomous Computing: The Basics
  - Fiefdoms and Autonomous Computing
  - Emissaries: Helping Interact With Fiefdoms
  - Rethinking Data
  - Rethinking the “N-Tier” Model
  - Fiefdoms & Emissaries

- Working With Autonomous Fiefdoms

- Working Across Fiefdoms

- Conclusion
The Web is Autonomous Computing

- The Web is lots of autonomous fiefdoms

- Define the term Fiefdom as:
  - Computing function and applications which behaves as an independent entity
  - Has private data
  - An autonomous unit --- managed independently
    - Usually one (or a few) machines

- Fiefdoms don't trust outsiders...
Requesting Service from a Fiefdom

- Fiefdoms don’t trust stuff from the outside
  - Incoming requests will be inspected
  - Fields will be validated
  - Identity will be authenticated

- Data from outside requests is never trusted
  - It must fit within prescribed values or it is rejected

Please, kindly consider my humble request....
A fiefdom keeps its data private
- No one outside can read or write the data
- Only well defined requests are serviced from the outside
  - These requests do not describe the contents of the fiefdom’s data
  - Provide services, not data access
Transactions and Fiefdoms

- Fiefdoms are built using transactions internally
  - It may use 2-phase commit across the nodes of a cluster
- A fiefdom (usually) will not agree to share a transaction with an outsider
Outline

- Introduction

- **Autonomous Computing: The Basics**
  - Fiefdoms and Autonomous Computing
  - Emissaries: Helping Interact With Fiefdoms
  - Rethinking Data
  - Rethinking the “N-Tier” Model
  - Fiefdoms & Emissaries

- Working With Autonomous Fiefdoms

- Working Across Fiefdoms

- Conclusion
Emissaries

- **Fiefdoms may come with emissaries.**
  - It knows how to fill out a request for the fiefdom
  - It understands the rules of the fiefdom and how to (probably) get the request accepted
    - Think of a mortgage broker... not the final approving party...

- **Emissaries have two purposes:**
  - Displaying information to users
  - Preparing requests to send to fiefdoms

- **Emissaries are not trusted by the fiefdom**
  - The contents of the request is still inspected for correctness
Emissaries & Snapshot Reference Data

- An emissary will frequently come with reference data.
  - This information will support the emissary in doing its job

- Imagine an emissary that helps you order from Sears
  - It will arrive with the Fall catalog under its arm

Reference Data
Emissaries & Single User Data

- Emissaries gather information needed to prepare requests
  - For example, a shopping basket accumulates the items to purchase
Outline

- Introduction

- Autonomous Computing: The Basics
  - Fiefdoms and Autonomous Computing
  - Emissaries: Helping Interact With Fiefdoms
  - Rethinking Data
  - Rethinking the “N-Tier” Model
  - Fiefdoms & Emissaries

- Working With Autonomous Fiefdoms

- Working Across Fiefdoms

- Conclusion
The “current” data is always kept inside the fiefdom

- It is the mission critical data that describes the business of the fiefdom
- This information is typically updated within a transaction
- Locks are held only for the duration of the transaction
- Transactions are not shared outside the fiefdom
Once data is unlocked, it must be assumed to be a snapshot
- Snapshot data was accurate and up-to-date at some time in the past... it is not necessarily still current
- We must assume that snapshot data is no longer current...
Stable Data & Immutable Data

- **Stable data is meaningful across space and time**
  - Anything to do with “current” is not stable
  - Timestamping can make data stable
  - Other techniques can make data stable
    - Given an IID (Interface-ID), type-lib info is stable

- **Immutable data is a snapshot that never changes**
  - It is never invalid
  - It may be cached without worries for cache consistency
  - It may be uninteresting, but it is never wrong...
All web data is a snapshot

- Fiefdoms don’t hold locks when working with outsiders...
- This means they have unlocked their data... it might have changed...
Outline

- Introduction
- Autonomous Computing: The Basics
  - Fiefdoms and Autonomous Computing
  - Emissaries: Helping Interact With Fiefdoms
  - Rethinking Data
  - Rethinking the “N-Tier” Model
  - Fiefdoms & Emissaries
- Working With Autonomous Fiefdoms
- Working Across Fiefdoms
- Conclusion
Labeling the Tiers

Rendering
- May accept HTML or some other rendering format
- Produces bits on the screen.

Presentation
- Maps the interactions with the emissary into pages to be displayed to the user. Typically produces HTML.

Emissary Logic
- Knows how to fill out requests to be submitted to the fiefdom. Possibly has snapshot data for submitting requests.

Fiefdom Logic

Emissary Data

Fiefdom Data
Slicing at the Front

- This is classic HTML 3.2 browser support
  - Do most of the work at the server
  - Send HTML to the client
  - Maximum “reach”
Pushing Down the Emissary

- Can run the emissary down on the client
  - Emissaries use snapshot data
  - The data is never incorrect...
  - It’s possible you’re missing some data you want but you will always know it

- We can download the code & snapshot data
  - Run the emissary on the client
  - Send the fiefdom requests over the wire
The Web Farm

- **Can run the emissary separate from the fiefdom**
  - Talks over HTML 3.2 to the browser
  - Talks to a dedicated machine with fiefdom logic and data

- **Scalable solution**
  - Offloads the precious fiefdom
  - Emissary can be replicated on webfarm for scale
Scaling the Web farm

A-J
Per-User Ref Data

K-R
Per-User Ref Data

S-Z
Per-User Ref Data

De-Militarized-Zone

Shared Writable Data

Protected Intra-Net

Ref Data
Contrast With Classic Tiers

- Rendering
- Presentation
- Emissary Logic & Data
- Fiefdom Logic & Data

Rendering & Presentation Combined
No concept of Emissaries
Logic & Data are in the Fiefdom

All the tiers in one fiefdom
Outline

- Introduction
- **Autonomous Computing: The Basics**
  - Fiefdoms and Autonomous Computing
  - Emissaries: Helping Interact With Fiefdoms
  - Rethinking Data
  - Rethinking the “N-Tier” Model
  - Fiefdoms & Emissaries
- Working With Autonomous Fiefdoms
- Working Across Fiefdoms
- Conclusion
Fiefdoms & Emissaries

- **Fiefdoms use multi-user writable data**
  - The business logic protects its integrity

- **Emissaries use:**
  - Read-only reference data, and
  - Single-user writable data (e.g. a shopping basket)
Outline

- Introduction
- Autonomous Computing: The Basics
- **Working With Autonomous Fiefdoms**
  - Requesting Services
  - The Flow of Data in an Autonomous World
  - Offline Work
- **Working Across Fiefdoms**
- Conclusion
Computing With Messages

- Incoming work arrives in a message
- The response leaves in a message
Classic OLTP & 3-Tier

- Sometimes online workers are part of the fiefdom
  - They are trusted and work using online transactions
  - This is classic OLTP and 3-tier
  - Less & less common with Intranet
Making Requests Idempotent

- Requests get lost...
  - Gotta retry them to handle lost requests
- Requests arrive more than once...
  - Those pesky retries may actually arrive
- Idempotent means it's OK to arrive multiple times
  - As long as the request is processed at least once, the correct behavior occurs
- In today's internet, you must design your requests to be idempotent
Challenges With Idempotent Requests

- Any request may arrive multiple times
  - Sometimes after quite a while
  - You might be a few messages farther along when an old one arrives

- The combinatoric complexity can be staggering
  - This leads people to build very simple applications since only then can they cope with the failure complexity...
Outline

- Introduction
- Autonomous Computing: The Basics
- Working With Autonomous Fiefdoms
  - Requesting Services
  - The Flow of Data in an Autonomous World
  - Offline Work
- Working Across Fiefdoms
- Conclusion
Content Syndication

- To fill out requests, an emissary needs snapshot reference data
  - The fiefdom must publish this reference data
- Content Syndication is the buzzword for the publication of reference data

Price list for Joe's Fiefdom
Valid 10/14/01 thru 10/17/01

Joe's
Emissaries use syndicated content to prepare requests for service

- The processing of the request must be designed to tolerate some staleness
The Flow of Data

- **Data flows in a big cycle:**
  - The fiefdom published reference data
  - This reference data is used in requests for service
  - The processing of requests may impact the new reference data that is published later
Another buzzword is Content Aggregation

- You process syndicated content from multiple sources to glean interesting data
- For example, which supplier has the best price, quality, and availability

Some emissaries perform sophisticated processing to aggregate content
Outline

- Introduction
- Autonomous Computing: The Basics
- Working With Autonomous Fiefdoms
  - Requesting Services
  - The Flow of Data in an Autonomous World
  - Offline Work
- Working Across Fiefdoms
- Conclusion
Offline and Online: Points on a Spectrum

- All online emissary work is really a flavor of offline
  - The data is always a snapshot
  - Offline just requires caching more snapshot reference data

- The fiefdom must cope with stale requests
  - All requests will be somewhat stale...

Sub-second access to the fiefdom

Disconnected for days

Staleness of the snapshot data used to prepare the request to the fiefdom
Emissaries & Offline Processing

- An emissary works with 2 kinds of data:
  - Snapshots of data published by a fiefdom, and
  - Per-client information

- You can package up an emissary, along with its snapshot data and download it to a smart client
  - On the client, the emissary prepares requests

- Requests are queued up until reconnection
Bringing Enough Content Along

- **“Online”** applications don’t need to bring along snapshot reference data
  - They go fetch it on demand

- **Offline** applications need to bring reference data with them
  - Organizing what to bring and how to group it together is a new challenge
Outline

- Introduction
- Autonomous Computing: The Basics
- Working With Autonomous Fiefdoms
- Working Across Fiefdoms
  - Tentative Work
  - Canceling and Confirming Tentative Work
  - Bounding Uncertainty
- Conclusion
Transactions Don’t Work Across Fiefdoms

- Fiefdoms will not agree to share transactions with outsiders
  - No self-respecting fiefdom would hold locks waiting for some other fiefdom

- How can work get coordinated across fiefdoms???
Tentative Operations

- Sometimes fiefdoms will accept requests for tentative operations
  - Similar to a reservation, this can be cancelled later on....
- If the operation is cancelled, the invoked service must deal with the effects
Semantics of Tentative Operations

- Because a tentative operation may be cancelled later, they must be commutative
  - Commutative means reorderable...
    - When we cancel, we compensate
    - Other stuff may have happened in between
  - It is important to ensure that the operation and the cancellation are commutative
Outline

- Introduction
- Autonomous Computing: The Basics
- Working With Autonomous Fiefdoms
- Working Across Fiefdoms
  - Tentative Work
  - Canceling and Confirming Tentative Work
  - Bounding Uncertainty
- Conclusion
Semantics of Cancellation

- When a tentative operation is cancelled, somehow, the invoked service must compensate for the operation
  - This is not an undo
  - It is another operation that makes things right...

- Part of supporting tentative operations is to ensure the operation is cancelable
  - Again, commutativity is the main technique used to support cancelable operations
    • You can reorder the cancellation with other work
Relinquishing the Right to Cancel

- When an invoking fiefdom chooses to never cancel, it may confirm the operation
  - When you confirm, you give up the right to cancel

- Every tentative operation will be either confirmed or cancelled
Outline

- **Introduction**
- **Autonomous Computing: The Basics**
- **Working With Autonomous Fiefdoms**
- **Working Across Fiefdoms**
  - Tentative Work
  - Canceling and Confirming Tentative Work
  - **Bounding Uncertainty**
- **Conclusion**
Increasing & Decreasing Uncertainty

- Each tentative operation increases your uncertainty
  - You get more and more confused each time you accept a tentative operation

- Each confirmation or cancellation decreases your uncertainty
  - It resolves the confusion imparted by the tentative operation it is confirming or canceling
Bounded Uncertainty

- You can track the worst case situations for data values you are managing
  - If you keep inventory, you can know the lowest possible and highest possible values
  - Tentative operations move lowest and highest values apart
    • This increases uncertainty
  - Confirmations and cancellations move lowest and highest values together
    • This decreases uncertainty

- Knowing the bounds, you have Bounded Uncertainty
Acting on Bounded Uncertainty

- Knowing bounds on uncertainty allows many different business rules:
  - Refuse an order which may (in the worst case) result in widgets overflowing the warehouse
  - Calculate probability of worst case overflowing the warehouse
    - Cost of temporary storage vs. value of accepting order...
  - Order food for hotel restaurant based on reservations and probabilities

- May result in interesting work by applying risk management algorithms...
Outline

- Introduction
- Autonomous Computing: The Basics
- Working With Autonomous Fiefdoms
- Working Across Fiefdoms
- Conclusion
Rambling Philosophy...

- **Atomic transaction are singularities**
  - Locking makes them appear to be at a single point in time
  - Two-phase commit removes distribution concerns
- **The new challenges happen when spreading work across space and time**
  - Different fiefdoms are in different spatial locations
  - Work across different messages gets processed at different times
- **Commutativity helps relax space and time**
  - By reordering with acceptable answers, it’s OK
    - Note that DB write is not commutative
- **Emissaries don’t have concurrency problems**
  - They only deal with read-only and single-user data
  - They make things “stick” by sending requests to fiefdoms
    - The fiefdom must have commutative requests… it sorts out the interleaving of work
## Terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autonomous Computing</td>
<td>A style of computing supporting the interaction of independent systems.</td>
</tr>
<tr>
<td>Fiefdom</td>
<td>An autonomous computing entity with private data and logic.</td>
</tr>
<tr>
<td>Emissary</td>
<td>Code that goes out into the world to assist in working with a fiefdom.</td>
</tr>
<tr>
<td>Snapshot Data</td>
<td>A copy of data sent out to an emissary as a references source. For example, a price-list.</td>
</tr>
<tr>
<td>Stable Data</td>
<td>Data that doesn’t change... it’s meaning is clear anytime and anywhere. Appropriate for messages and snapshots.</td>
</tr>
<tr>
<td>Bounded Uncertainty</td>
<td>The management of ongoing work to track tentative operations and their confirmation and cancellation.</td>
</tr>
<tr>
<td>Idempotent Request</td>
<td>A request message that may be repeatedly processed while still giving behavior the same as a single execution. Essential to cope with retries...</td>
</tr>
</tbody>
</table>
Summary

- **Autonomous Computing**
  - Allows distrusting systems to cooperate
  - Defines a framework for the distribution of data to use in messages
  - Loosely-coupled collections of tightly coupled services
  - Explains web, offline, B2B, B2C, long-running work, etc.