What is MODeX?

A domain-specific modeling language for enterprise messaging.

Components include:
- Graphical modeling IDE
- Code generator
- Language-specific runtime components
- Lifecycle management tools
Current State of XML Messaging in the Enterprise

- Direct XML manipulation is the least common denominator.
- Toolkits handle basic SOAP envelopes, transport, message dispatching and handling.
- Developers may use JAXB, XSD.EXE for marshalling/unmarshalling.
- Adopting CxF and WCF as standards
- Ambivalent relationship with schema:
  - Most developers ignore it or treat it as a design-time exercise.
  - Some developers invest heavily and want to use schema as “the model”

- Developers and designers have been left to translate business concepts to XML formats.
  - Elements or attributes?
  - Type substitution or element substitution?
  - Nesting or IDREF for shared references?
  - ...

- Big problems:
  - Cost of coding directly against XML or even mapping is a drag on developer productivity
  - Proliferation of formats makes integration extremely expensive.
  - Weak contracts affect data quality.
Solution:

- Create a simple graphical modeling language for enterprise data and message contracts
- Create a federated model repository for shared models and lifecycle management
- Generate schemas from well-specified message contracts
- Generate code and provide runtime support for multiple languages
- Build on this basic foundation to address other large-scale issues
Model-Driven Development Lifecycle

- Domain Expert / Business Analyst
  - Business Information Model
  - Message Contracts
  - MODeX Domain Model

- Message Contract Designer
  - Reference shared definitions
  - Publish model

- MODeX Code Generator
- Enterprise Model Repository

- XML Schema
- Java / C# Generated Domain API

- Application Code
- Application Developer

- Conforms to...
  - Business Information Model
  - Message Contracts
Eclipse Modeling Technologies

Proven open source technology stack

- GMF: Generative Diagram Editor Framework
- GEF: MVC Pattern for Graphical Editors
- EMF: Model-Driven Development Framework, integrated with GMF
- RCP: Views, Editors, Perspectives
- SWT/JFace: UI Toolkits
- Equinox: Plug-in and Buddy Classloader System
MODex Designer
Demo and Walkthrough

MODex Designer
Entity Modeling
Message Contract Modeling
Generated schemas
Generated Java API
MODEx - Key Features

Inheritance and Subtype Roles
Field Value Constraints and Enumerations
Views and Payload Contracts
Full-Fidelity Messaging
Entity-Centric API
Data Aspects
Model Documentation - Modelpedia
Inheritance and Subtype Roles

Entity Definitions:
- Research Report
  - Report ID: String
  - Publish Date: Date
  - Authors: Reference
  - Report Title: String
- Company Report
  - Primary Company: Reference
  - Secondary Companies: Reference
  - Rating Change Flag: Boolean
  - Earnings Change Flag: Boolean

Generated Domain API:
- ResearchReport
- ResearchReportImpl
- CompanyReport
- CompanyReportImpl

Types are substitutable in the generated API and Schema.

View Definitions
- Report Summary View: View
  - Research Report [v1]: Entity
    - Report ID: String [R]
    - Date Publish Date: [R]
    - Authors: Reference (-> Analyst [v1]) [OI]
    - Report Title: String [OE]
  - Company Report: Subtype Role
    - Primary Company: Reference (-> Company [v1]) [R]
    - Secondary Companies: Reference (-> Company [v1]) [OI]
    - Rating Change Flag: Boolean [R]
    - Earnings Change Flag: Boolean [R]

Schema complexTypes:
- Subtype role: “if the research report is a company report, these additional fields are required.”
Field Value Constraints, Enumerations

Field Value Constraints based on a subset XML Schema facets:
- Range
- String Length
- Regular Expression
- Numeric Format

Static Enumerations (current):
- Available for any primitive type
- Translate to a true enum in Java
- Translate to a “typesafe enum” pattern in C#, C++

Semi-Static Enumerations (planned):
- Allowable values determined at runtime by a lookup into a database or data service.
- Enable/Disable validation, set expiration policy by configuration
Message Contracts

1. Entity Model
   - Trade
     - Trade ID
     - Onside Party
     - Offside Party
     - Trade Product
     - Quantify
     - Price
     - Cash Flows
   - Party
     - Party ID
     - Party Code
     - Name
   - Product
     - Product ID
     - Product Type
     - Description
   - Cash Flow
     - Date/Time
     - Amount
     - From Account
     - To Account
   - Account
     - Account ID
     - Account Code
     ...

2. Message Payload Model
   - Trade Enrichment View
     - Trade
       - * Trade ID
       - Party ID
       - Party Code
       - Name
     - Offside Party
       - Party ID
       - Party Code
       - Name
     - Onside Party
       - Party ID
       - Party Code
       - Name
     - Trade Product
       - Product ID
       - Product Type
       - Description
     - Cash Flows
     - Date/Time
     - Amount
     - From Account
     - To Account
     - Account
       - Account ID
       - Account Code
     - ...
Views Example – Stage 1

Legal Identity
- Party ID
- Legal Name
- Identity Type
- Tax Treatment
- Legal Entity Status
- Consolidated
- Regulated
- Audited
- Country of Formation
- FY End Month
- Formation Date
- Legal Form
- Trade Type
- EIN
- GL Company
- Approved Organizations
- Responsible Organizations
- Product Types

Organization
- Responsible Organizations
- Segment
- Business Unit
- Division

Product Type
- Product Type ID
- Description

GL Company
- GL Code
- Functional Currency
- Status
- GL Company Name
- Parent GL Company
- GL Stream
- Cost Center CTA
- CC P and L
- CC Retained Earnings
- TAPS Fed
- Mirror Company Security
- Mirror Company FX
- Accounting Jurisdiction

Accounting Jurisdiction
- Jurisdiction ID
- Description
- Governing Body
- Revision
Views Example – Stage 2

Legal Identity
- * Party ID
- * Legal Name
  - Identity Type
  - Tax Treatment
  - Legal Entity Status
  - Consolidated
  - Regulated
  - Audited
  - Country of Formation
  - FY End Month
  - Formation Date
  - Legal Form
  - Trade Type
  - EIN
  - GL Company
  - Approved Organizations
  - Responsible Organizations
  - Product Types

Organization
- Segment
- Business Unit
- Division

Product Type
- Product Type ID
- Description

GL-Company
- GL Code
- Functional Currency
- Status
- GL-Company Name
- Parent GL Company
- GL Stream
- Cost Center CTA
- CC P and L
- CC Retained Earnings
- TAPS Fed
- Mirror Company Security
- Mirror Company FX
- Accounting Jurisdiction

Accounting Jurisdiction
- Jurisdiction ID
- Description
- Governing Body
- Revision
Views Example – Stage 3

Legal Identity
- *Party ID*
- *Legal Name*
- Identity Type
- Tax Treatment
- Legal Entity Status
- Consolidated
- Regulated
- Audited
- Country of Formation
- FY End Month
- Formation Date
- Legal Form
- Trade Type
- EIN
- *GL Company*
  - Approved Organizations
  - Responsible Organizations
  - Product Types

Organization
- Responsible Organizations
- Segment
- Business Unit
- Division

Product Type
- Product Types
- Product Type ID
- Description

GL Company
- GL Company
- *GL Code*
- Functional Currency
- Status
- GL Company Name
- Parent GL Company
- GL Stream
- Cost Center CTA
- CC P and L
- CC Retained Earnings
- TAPS Fed
- Mirror Company Security
- Mirror Company FX
- Accounting Jurisdiction

Accounting Jurisdiction
- Accounting Jurisdiction
- Jurisdiction ID
- Description
- Governing Body
- Revision
Objects:

- Object Graphs may contain shared references and cycles

XML:

- Shared objects are detected by the marshaller, and assigned a unique ID.
- The unmarshaller will reconstruct the object graph with full fidelity.

API:

- Traversal over the object graph across any path yields full access to all fields.
Effects multiply with number of types, clients, services, and message formats.

Additional memory footprint, performance overhead and development effort

Enrichment and request/response scenarios require copying from one representation to another.

Effects multiply with number of types, clients, services, and message formats.

Model-Based Code Gen:

- Single class/interface each entity
- An entity class can be marshalled to multiple views, validated against multiple contracts
- API decoupled from specific message format.

Entity-Centric API

Schema-Based Code Gen:

- Separate class for each complexType.
- Enrichment and request/response scenarios require copying from one representation to another.
- Additional memory footprint, performance overhead and development effort

Separate class for each complexType.

Enrichment and request/response scenarios require copying from one representation to another.

Additional memory footprint, performance overhead and development effort

Effects multiply with number of types, clients, services, and message formats.
Data Aspects

- Concept borrowed from aspect-oriented programming
- Represent cross-cutting concerns in message contracts.
- Data Aspects are included and configured in view definitions.
- Data Aspects contributed “properties” to the XML schema and API

Business Information Model

<table>
<thead>
<tr>
<th>Party ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Account ID</th>
<th>Balance</th>
<th>Margin</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Data Aspects

- Deltas
- Data Source
- Data Security
- Explicit Null Values
  ...

Message Structure

```xml
<party dataSource="MSDB">
  <partyId>123</partyId>
  <partyName xsi:nil="true" nullValue="Not Available" />
  <accounts>
    <account dataSource="A2" transType="insert">
      <accountId>XYZ</accountId>
      <balance xsi:nil="true" isRestricted="true" />
      ...
    </account>
    ...
  </accounts>
</party>
```
Modelpedia - Model Documentation

Collaborative Web-Based tool for model documentation. Shows relationships among models entities, across domains. Searchable data dictionary includes entity types, fields, message contracts. Browse generated artifacts, including code and schema.

Future plans:
- Tie-in to model publishing lifecycle
- User-contributed content
- Wiki markup
- Discussions, change requests
- Hyperlinked diagrams
- Usage Statistics
Evolving Features

Cross-Domain Modeling and Code-Gen
Versioning
Advanced Validation
SOA Toolkits Integration
Advanced Modeling Features
Model Mapping and Transformation
Evolution: Cross-Domain Modeling

Goal: Enable Re-use of model constructs across domains:
- Inherit from entity in another domain.
- Reference an entity in another domain.
- Define a message contract for an entity in another domain.

Architectural Requirements:
- Federated enterprise model repository
- Cross-domain modeling in MODeX Designer
- Cross-domain code generation
- Cross-domain runtime marshaling and validation.
- Model lifecycle management.
Evolution: Versioning

- Minor versions can add optional fields.
  Runtimes support minor version bridging, and unknown field data pass-through.

- Major versions can completely alter fields: add, remove, delete, change optional/mandatory status.
  Does not work with existing contracts, but tooling can allow smoother major version migration.

- Modeling tool maintains multiple copies of Entity for every version, minor or major.

<table>
<thead>
<tr>
<th>Trade</th>
<th>v1</th>
<th>v1.1</th>
<th>v2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>M</td>
<td>M</td>
<td>D</td>
</tr>
<tr>
<td>Symbol</td>
<td>M</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>Market</td>
<td>M</td>
<td>M</td>
<td>O</td>
</tr>
<tr>
<td>Quantity</td>
<td>O</td>
<td>O</td>
<td>M</td>
</tr>
<tr>
<td>Price</td>
<td>O</td>
<td>O</td>
<td>M</td>
</tr>
</tbody>
</table>

M = Mandatory
O = Optional
D = Deleted
Evolution: Advanced Validation

MODeX validation constraints are embedded in message contracts.

All validation happens in-memory, independent of wire format.

Available equally to message sender and receiver, runtime is configurable to perform selected validations on send or receive.

Validation produces detailed list of violations that can be inspected, logged, transmitted in a message.

Expression Constraints
- Pluggable expression dialects, default is based on Python expression syntax.
- Executes cross-platform
- Allow arbitrarily complex validation expressions.

Rule Constraints
- Pluggable rules engine, default is JBoss Rules
- Forward-chaining rules engine allows rules to enrich the data, fire other rules based on enriched data.
Evolution: SOA Toolkits Integration

Phase 1 (Complete):
  Interop Message Format

Phase 2 (Planned):
  MarshalStack Integration with CxF and WCF

Phase 3 (TBD):
  MODeX Types in WSDL Contracts
SOA Toolkits Phase 1
Interop Message Format

* New message schema generated by MODeX Designer
  * Specifically designed to produce a straightforward API when used with CxF/JAXB and WCF/Data Contract Serializer
  * Pure WSDL solution, with no runtime dependency on MODeX
SOA Toolkits Phase 2
MarshalStack Integration with CxF and WCF

* MODex MarshalStack runtime plugs in as a first-class marshalling/serialization option.
* Generated entity interfaces can be used as WebMethod arguments and return types.
* Provides full benefits of MODex on the server: Entity-Centric API, Full Fidelity Messaging, Advanced Validation, Field Promotion
* Service consumers can choose pure WSDL import with Interop message format, or can use MODex GDAs with MarshalStack.
SOA Toolkits Phase 3
MODex Types in WSDL Contracts

* Types section of WSDL expressed in terms of MODex models and message contracts.
* Service exposes this WSDL on a separate port from standard, schema-typed WSDL.
* IDE tooling imports MODex service contracts, required MODex GDAs, and endpoint implementations that use the GDAs and MarshalStack transparently.
Evolution: Advanced Modeling Features

Design goal: provide a conceptual modeling language optimized for enterprise data.

- Capture essential technology-independent semantics.
- Map cleanly to multiple data representations.
- Facilitate human-to-human communication and documentation.

Features:
- Model/Contract Separation
- Rich, Extensible Constraints
- Optional Rules Engine Integration
- Borrows from multiple Multiple Paradigms:
  - Object Oriented: References, Inheritance, Associations*
  - AOP: Data Aspects
  - Relational: Uniqueness Constraints, Semi-Static Enumerations, Queries*
  - XML Schema: Derive by Restriction*
  - Ontology: Multiple Classification*

* Proposed
Evolution: Model Mapping and Transformation

Bidirectional solution supports forward code generation or code-first mapping.

Maplet architecture maps a MODeX model to a technology-specific representation.

Maplet enables code gen or runtime transformation between model instances.

Captures syntactic and topological transformations.

Allows N-way transformations and runtime validation.

Import and Synchronization capabilities for design-time management.
Applications in Financial Services

Enterprise Data
- Shared, authoritative data definitions
- Centralized services
- Client-side API for enterprise data consuming applications.

Sales and Trading
- Common trade lifecycle model
- Data integrity controls with business transparency
- Seamless cross-platform interop

Investment Banking
- Improved business/IT collaboration, transparency
- SOA governance, consistent standards.
- Significantly improved reuse, developer productivity, time to market
Experience with Eclipse Technology

- Extremely bright, engaged community
- Highly evolved technology
- Center of activity and thought leadership for model-driven software development
- Significant learning curve to technology
- High investment, high payoff
Recap: How MODeX Addresses Enterprise Messaging Challenges

- Elevates the model to a first-class form of source code.
- Partitions enterprise information into domains of manageable scope, with clear ownership and managed lifecycle.
- Reduces integration barriers by defining message payloads in terms of a consistent information model.
- Provides an expressive, graphical language for message protocol definition.
- Provides multiple views of information depending on lifecycle and messaging context.
- Enables orderly model evolution with entity-level versioning.
- Preserves object topology, including shared references and cycles.
- Uses schema validation where appropriate, but allows for rich, rules-based validation where required.
- Hides the wire format from developers, providing a path to more efficient wire representations.
- Reduces impedance mismatch between messages and objects.
Q&A
Appendix: Positioning Relative to Other Technologies
MODeX and JAXB

Limited to a single language.
Model is inferred from code, all other manifestations suffer.
Generic model inference does not encourage standard implementation.
Enforces one-to-one mapping of Java class to XSD type.

Annotated Java Classes \(\rightarrow\) JAXB Mapping Model \\
JAXB Mapping Model \(\leftrightarrow\) XML Schema \(\rightarrow\) XML Message \\
Model inference \\
Model inference \\
Java Objects \(\rightarrow\) Runtime Marshaller \\
MODeX and JAXB

Putting the model at the center gives the power of abstraction.

Code generation guarantees standardized implementations.

Model captures rich semantics encompassing all target languages.

Allows mapping a single class to multiple XSD types.

Supports multiple languages.
MODeX and Service Oriented Architecture (SOA)

- SOA promotes reuse and transparency.
  - Services are composable building blocks for applications.
  - BPEL, SCXML & other orchestration languages build on this.
  - Service catalog forms a business-oriented vocabulary of available functionality.
  - SOA organizes the *verbs*.

- Problem: what about the *nouns*?
  - XML Schema fails as a modeling language.
  - Databases tend to be specialized, don’t translate directly to services.
  - Object models are application-specific.
  - Result: Impedance mismatch at the boundaries, very little reuse

- MODeX provides the missing plank in the SOA platform.
  - Helps you organize around a shared set of entity definitions.
  - Expresses message contracts in terms of these entity definitions.
  - Schema and programmatic models are tied directly to the data definitions.
  - Frees SOA from dependencies on today’s XML, WSDL and WS-* base technologies.
Model-Driven Development and CASE

"Isn't this the same promise that CASE tools were making in the 1980's and '90's? If it didn't succeed then, why do we think we can make it work now?"

<table>
<thead>
<tr>
<th>CASE</th>
<th>Model-Driven Development</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focus on round-trip engineering</td>
<td>Focus on meaningful abstraction</td>
</tr>
<tr>
<td>General-purpose modeling</td>
<td>Domain-specific modeling languages</td>
</tr>
<tr>
<td>Monolithic design</td>
<td>Rich graphical modeling and IDE integration</td>
</tr>
<tr>
<td>Primitive, inflexible code generation</td>
<td>Sophisticated model-to-model and model-to-code transformation pipeline</td>
</tr>
<tr>
<td>Limited cross-platform design</td>
<td>Broad applicability to distributed software architecture</td>
</tr>
</tbody>
</table>