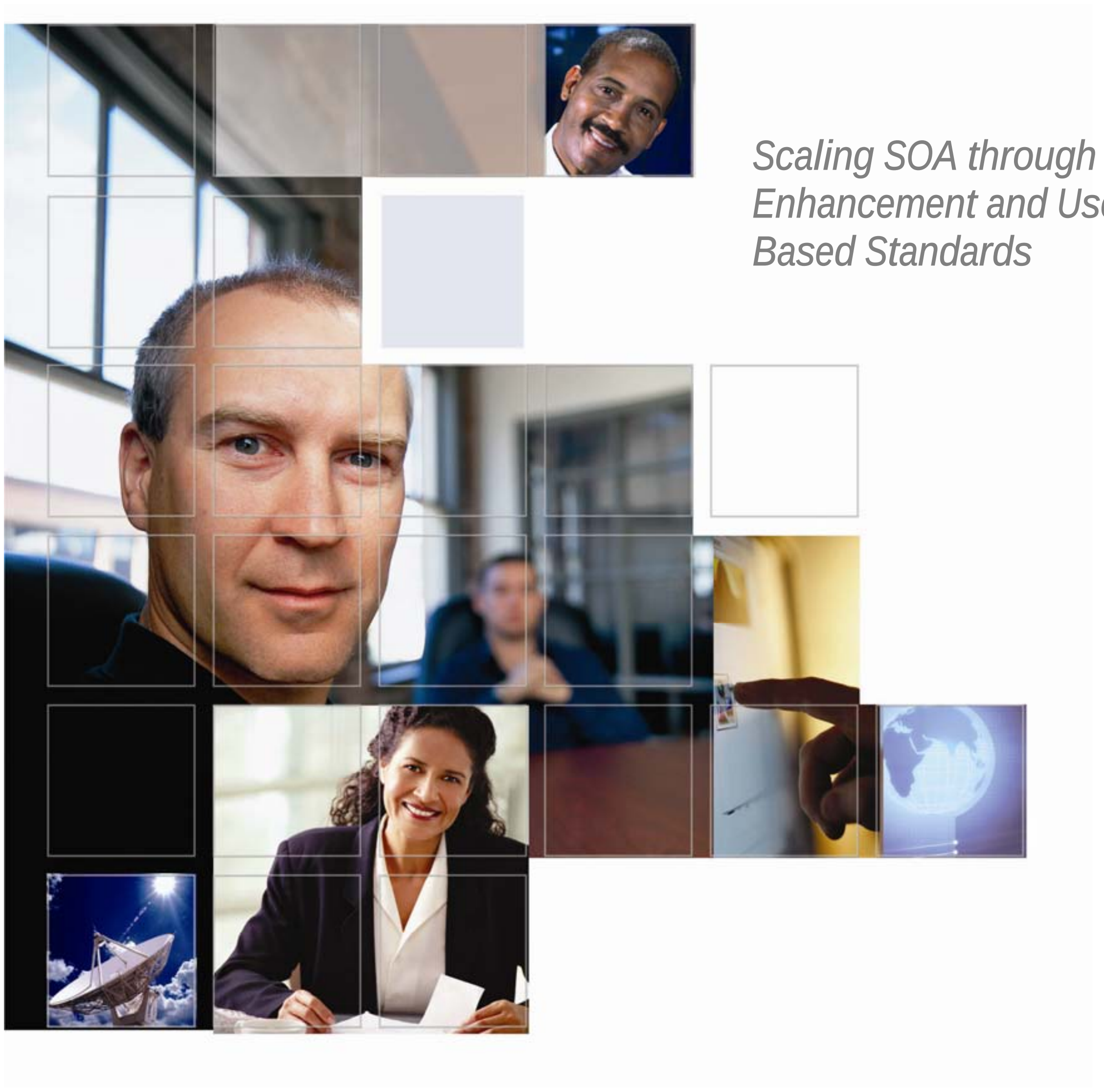


Scaling SOA through EAI Enhancement and Use of Model-Based Standards

PROGRESS SOFTWARE

Table of Contents

Executive Summary

SOA increases business agility by providing:

- Loose coupling instead of the tight coupling of systems,
- Standards-based instead of proprietary technologies,
- Improved vendor interoperability through the use of standards,
- Highly reusable service interfaces.

Many corporations look to their incumbent EAI vendors to provide the tooling to support the build-out of scaleable SOA. These vendors have provided an excellent platform to perform critical systems integration tasks. They have been particularly effective at:

- Core messaging and transport infrastructure via an Enterprise Service Bus (ESB),
- Business process orchestration which designs and controls invocation of services.

However, as corporations attempt to leverage the in-house technologies provided by the leading EAI and SOA platform vendors, they will likely encounter significant productivity, governance, and performance challenges as they seek to scale their SOA pilot projects into large-scale production implementations.

Progress DataXtend Semantic Integrator's (SI) standards-based, model-driven, visual design tools and distributed runtime operate within existing EAI frameworks to ensure the real-time scalability of SOA application integration. Semantic integration with DataXtend SI provides advantages for productivity, governance, and performance as corporations scale their SOA projects.

DataXtend SI Radically Accelerates Productivity

EAI vendors' point-to-point architecture leads to custom code for managing the semantic mapping of data between applications. This custom code cripples productivity and creates brittle integrations that cannot be easily changed, extended, or scaled.

Benchmarks show that using DataXtend SI to perform semantic integration accelerates productivity and time-to-market by nearly a factor of 3 compared to using existing EAI technologies, by automating functions such as complex transformations, extensions to industry standard models, data consistency support, and declarative error handling.

DataXtend SI Simplifies Governance

Often developers attempt to overcome the shortcomings of tools by deploying various and ingenious alternative practices. These efforts cause huge manual governance challenges which create significant project overhead, lengthen timelines and expand costs.

Case studies show that DataXtend SI's Common Model Architecture™ (CMA) radically simplifies the governance complexity found in typical EAI integration projects. Users are showing a 10-1 acceleration of time-to-market for changes that are semantically integrated using DataXtend SI. One metadata repository, one set of mappings against one common model makes the lifecycle management of an integration project far faster and easier. By using DataXtend SI and its CMA, IBM's Telco group claims that they can change out any COTS package in their Service Delivery Platform (SDP) in two weeks or less.

DataXtend SI Ensures Critical Performance

DataXtend SI automatically generates stateless Java as its run-time. It scales economically and linearly with application server capabilities, easily meeting the requirements of Tier 1 carriers who are seeing explosive growth in the delivery of new services. Its run-time eliminates the need for XSLT performance remediation and delivers a true SOA distributed run-time architecture. DataXtend SI's CMA overcomes the semantic gap between applications.

Progress delivers breakthrough technology for integrating business data in SOA's. DataXtend Semantic Integrator is the first scaleable solution to automate the costly challenge of reconciling and managing the business integrity of data (semantics) shared between applications to accelerate application integration. DataXtend SI's unique, metadata-driven CMA leverages a wide variety of industry-standard and application models, replacing hand code and manual governance with reuse and automation, reducing the cost, time and complexity of integrating systems by over 50%.

Key Assumptions

The first important premise of this white paper is that hand-written, custom code in integration projects creates tight binding of applications and prevents loose coupling, a critical feature of SOA. Custom code can be described many ways:

⇒ Custom programming code as in Java or Extensible Style-sheet Language Transformations (XSLT)

⇒ Custom extensions to industry messaging or model standards

⇒ Programmatic scripting languages that are not usable by business analysts

Second, XSLT is essentially a programming language with all the associated benefits and challenges.

Third, the leading EAI vendors provide a variety of mapping tools that generate XSLT as their transformation language. This XSLT might be compiled or installed into accelerators, but the issues surrounding the productivity, maintenance and scalability of XSLT in an SOA context are not eliminated.

Fourth, data modeling as a function of integration projects is growing in importance.

Fifth, this white paper frequently mentions the Telecommunications Shared Information Data Model (SID). There is nothing unique about this model. The reader could replace the SID with any other industry data model (OAG, Acord, HL7, etc.)

Sixth, the term *data consistency rules* in this white paper is used to cover a variety of data-oriented terminology that are considered distinct in many data management contexts:

⇒ Data consistency -- comparing gross revenue and net operating revenue;

⇒ Data integrity -- there is a department head for a non-existent department;

⇒ Data quality -- phone # is 8 digits.

Finally, DataXtend SI is not a replacement for EAI technology. The components of EAI technology that manage business processes and transport and guaranteed delivery of information (such as an Enterprise Service Bus) are essential to the build-out of scaleable SOAs and are not in conflict with this white paper.

Productivity Challenges

EAI integration projects must address several development productivity issues including:

⇒ Executing complex transformations

⇒ Extending standards

⇒ Enforcing data consistency rules

⇒ Handling errors

EAI Resorts to Custom Code for Complex Transformations

To ensure re-use of services across multiple applications, business divisions, and even enterprises, canonical (agreed-upon, standards-based) messages must be sufficiently abstract so that each constituent of the message can “fit” into the abstraction. Thus a retail division that thinks of customers in terms of home phone numbers can share the same notion of customer with a wholesale division that thinks of customers in terms of VAT IDs, DUNS numbers, and so on.

What enables both divisions to share the same notion of customer would be a concept such as business party which is sub-classed into individuals and organizations. In these subclasses are the specifics that distinguish retail from wholesale eliminating the need for consensus. Each division operates independently, interoperating with its notion of customer and the abstract model of business party without affecting the other.

Industry standard models, like the Shared Information Data (SID) model developed for the telecommunications industry, provide a basis by modeling the most common concepts of the industry. However, they require many complex transformations to and from the model because of their abstract nature. For example, the SID contains many complex types that can have many enumerations on each type, each of which could be mapped to different business contexts which would require many-to-1 or 1-to-many transformations.

Most visual mapping editors from EAI vendors such as Tibco or BEA rely on XSLT for transformations, but there is no automated drag-drop facility within these tools to do many-to-1 or 1-to-many transformations. As a result, developers have to hand-code complex transformations directly in XSLT.

A Complex Transformation Example

Suppose you need to map telephone numbers between two schemas as follows:

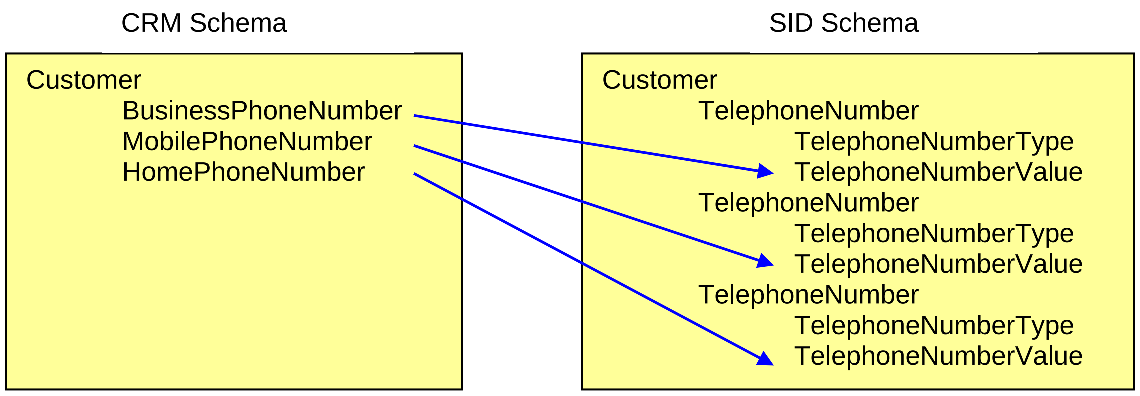


Figure 1: A Complex Transformation for Phone Numbers

When transforming from CRM Schema to the SID Schema, data from one `Customer` instance must map into “Many” occurrences of `TelephoneNumber`. When transforming from the SID Schema to the CRM Schema, data from “Many” instances of `TelephoneNumber` must map into one `Customer` instance.

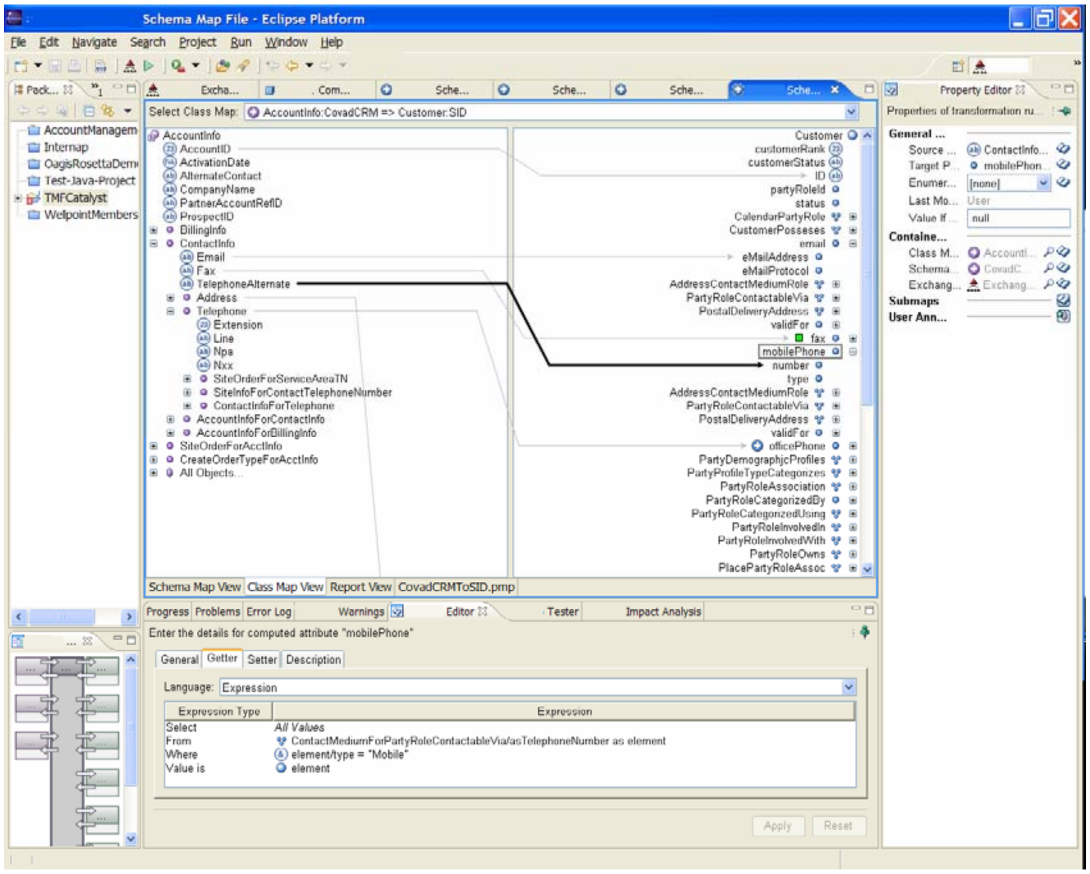
The work done in these mappings to build the XSLT will be custom and cannot be easily used across other mapping files and systems without a significant amount of manual governance. (See a discussion on the governance challenges with EAI mapping tools on page 11.) This likely means duplicate work, duplicate effort, custom code and very real possibilities for error.

Current Practice Alternative

The alternative to coding the complex transformation is to simplify the XSD (XML Schema Definition) that contains the complex types and causes the complex transformation. In this scenario, the XSD is modified to make the abstraction explicit so that the mapping tools can visually create the mapping without resorting to hand-coded XSLT. The implication of this option is discussed in *EAI Must Customize to Extend a Standard*, on page 8. Physically changing schemas can raise significant governance challenges. Changes must be agreed to when this model is shared across mappings and deviations from the standard must be managed carefully to ensure future upgrade paths.

DataXtend SI's Answer to Complex Transformations is Computed Attributes

DataXtend SI's Expression Builder can create computed attributes which are virtual attributes in metadata that enhance or simplify the SID without changing it. For example, while the attribute mobile phone number is not in the SID, DataXtend SI can create a virtual attribute that makes a mobile phone number look like a real attribute in the SID.



Enter the details for computed attribute "mobilePhone"

Language: Expression

Expression Type	Expression
Select	*All Values*
From	ContactMediumForPartyRoleContactableVia/asTelephoneNumber as element
Where	element/type = "Mobile"
Value is	element

Figure 2: DataXtend SI Mapping Complex Transformations Easily into the SID

The advantage is that it makes the common model easier for analysts to understand and is re-usable in mappings across multiple application interfaces. The computed attributes are in metadata and appear to be simple attributes to any other application interface. This is a better approach for several reasons:

⇒ No custom code

⇒ Re-usable attributes across any number of client applications mapped to the common model

⇒ Computed attributes can be leveraged across any number of projects

In the example of telephone numbers, additional types of telephone numbers can be added over time without the need to change the physical model, simply by defining additional computed attributes.

EAI Must Customize to Extend a Standard

There are several good reasons why a standard like the SID might need to change the SID in your environment. These include:

- ⇒ New ISV requirements
- ⇒ Missing information from the standard
- ⇒ The need to simplify the standard
- ⇒ The need to add rules to represent concrete concepts with the SID's abstract concepts

Current Practice Alternative #1 — Physically Change the Model

When using typical EAI mapping tools, the SID is just another schema, so any change to the SID is a physical change to the XSD, as in the case of making a concrete attribute called mobile phone number from the abstraction of number and type in the SID. This physical change to the schema requires significant manual governance to gain agreement across all parties using the SID, and to manage updates of the SID standard over time. The corporation has now begun to diverge from the industry standard, losing the value gained from being standards-based. For more on governance, see *Governance Challenges* on page 12.

Current Practice Alternative #2 — Write XSLT

The alternative to making physical changes in a schema is to handcraft XSLT code to navigate the complex transformations caused by the abstract nature of the SID. Current EAI mapping tools do not provide a visual or declarative IDE, thus forcing the developer to code XSLT.

Implications

The result of this handcrafting and the point-to-point nature of EAI mapping tools is custom code that causes brittleness in the architecture and a tight binding between systems, instead of the loose coupling prescribed by SOA.

With either approach, there is no gain from using standards because any change to the model means a physical change to the XSDs, which now creates a physical variation from the industry standard. As each project instantiates its own XSD, implementing multiple versions of the abstract model, the corporation increases its divergence from the standard.

DataXtend SI's Solution for Extending Standards is Computed Attributes

DataXtend SI provides computed attributes that can be used to enhance or simplify the standard without directly changing it. By using DataXtend Designer, a developer or analyst can make extensions to the SID to fulfill requirements for the business without physically changing the SID's XSD. DataXtend SI's design and run-time architecture uses metadata captured distinctly from the SID common model.

The capability provided by DataXtend's computed attributes for simplifying a common model is theoretically possible with XSLT by using templates that would be used in endpoint transformations or as global variables in the transformation engine, but this approach has several shortcomings:

⇒ Complexity of implementation.

  Such a complex set of templates would be difficult to implement because complex transformations and information aggregation are beyond the capability of most XSLT mapping tools. DataXtend SI's graphical interface makes it easy to create most abstractions and the code is generated automatically.

⇒ Coordination of experts.

  Development would require experts in the common architecture to work closely with XSLT development experts. Disparate skill sets mean communication overhead and potential for misunderstandings and errors. DataXtend SI empowers architects to design and monitor the implementation of the extensions without programming. The code is automatically generated.

⇒ Architectural brittleness.

  Changes needed to accommodate a new system may result in modifications to the common XSLTs. Without clear impact analysis, all systems using extensions may need to be revalidated. DataXtend SI not only provides system-wide impact analysis but also includes a test environment that can trace sample messages through the system from end to end. For more on impact analysis, look on page 15.

EAI Tools Do Not Enforce Data Consistency Rules

Semantic rules that ensure data consistency cannot be enforced natively by XML. For example, inter-field dependency constraints such as these might be typical:

⇒ Service disconnect dates must be in the future.

⇒ A subscriber's move request requires two addresses.

⇒ If the service type is Voice over IP (VOIP), then there must be a service value provided which specifies the VOIP Port.

EAI mapping tools do not support the building and execution of these types of business rules. Thus, to enforce data consistency, alternative technologies such as script languages, business rules engines, or database reference tools are required beyond current EAI tools.

Current Practice Alternative #1 – Write Code

Developers often resort to writing scripts to enforce data consistency. These scripting languages are provided by the EAI vendor and can be proprietary or can call Java itself.

Implication

Consistent rules application can’t be enforced by writing custom code. Scripts or custom code will be tightly bound to the applications being integrated. The same business logic will have to be written and re-written as many times as different variations of those applications are integrated. In the example above, if a portal and Customer Relationship Management (CRM) system are each processing orders, the business logic that enforces the rules must be written twice, once for each integration. What ensures consistency in the design, build or deployment of the rule?

Current Practice Alternative #2 – Rules Engines

Many developers have looked at deploying rules engines to ensure data consistency. The theory goes that if rules need to be fired in-line, then a rules engine provides abstraction of the rule from the XML and can ensure data consistency. Rules engines accept a set of inputs and produce an output including errors that have occurred in the firing of the rule.

Implications

There are a number of challenges using a rules engine:

1) A rules engine will not enable the rule to be re-usable because of the specificity of the data related to the endpoints. Attributes from each application (portal and CRM) will differ, causing the rule to fail because of an attribute mismatch.
2) Impact analysis will not be accurate because the rules "metadata" is not integrated with the metadata of the schemas captured in the EAI technology.
3) Performance often suffers as these rules engines are built on proprietary servers which often become a bottleneck and single point of failure.
4) These are very complex solutions that exact a significant overhead in training and maintenance in order to fire very simple rules.

DataXtend SI’s Answer to Data Consistency Rules

All manner of data consistency rules can be written in DataXtend Designer and enforced in DataXtend SI’s runtime, all captured as part of DataXtend SI’s metadata without writing any custom code. Since these rules are captured in DataXtend SI’s integrated metadata, all dependencies related to rules are captured when impact analysis is run -- a very powerful feature unique to DataXtend SI.

Finally, as these rules are often understood best by analysts, DataXtend Designer uniquely provides the UI to enable analysts to write the rules instead of relying on communication to developers to then code them.

EAI Error Handling is an Afterthought and Custom

XSLT, the typical EAI transformation language, does not provide a native error detecting and reporting mechanism for determining whether a transformation works as intended, or for handling exceptions when they occur. Consequently, development and change-management lifecycles move slowly because custom code has to be written to handle errors.

Implications of Using XSLT for Error Handling

Using only a basic XSLT error handling mechanism can result in the following problems:

- ⇒ Transactions stop on the first failure.
- ⇒ Cryptic error codes are produced.
- ⇒ Analysts cannot participate in the design of the appropriate error messages and responses back to the sender.
- ⇒ Error handling differs by XSLT processor (Xalan, Saxon, MSXML, and so forth).
- ⇒ Each XSLT processor's own unique set of exception handling tools must be brought to bear in handling errors in input documents being processed.

DataXtend SI's Answer – Error Handling with No Custom Code

DataXtend SI traps every error and models an error response in memory that can be mapped to a response document, picked up by a BPM, written to an error log, and persisted to a database for archival and reporting purposes.

Analysts can define in English the response that goes in the response document when a rule fails, so that there is a clear explanation of the cause of the error, including all details such as data and context behind the error.

Governance Challenges

Governance is a mechanism an IT department uses to ensure compliance with its own rules and regulations. EAI project governance is more difficult than most because:

- ⇒ EAI lacks a common model architecture.
- ⇒ EAI is missing 360^{o} E2E (End-to-End) impact analysis tools.

EAI Lacks a True Common Model

EAI vendors do not have design tools to take advantage of a common model such as the SID standard which forces developers to conduct a significant workaround with multiple manual steps, creating numerous mapping files to accomplish one round trip. Further, using XSLT as the transformation language means re-use can only be achieved by resorting to programming XSLT, creating even more governance challenges.

Current Best Practice Alternative #1 -- Use the SID with Point-to-Point EAI Mapping Tools.

Although EAI mapping tools are inherently point-to-point, a current best practice promotes the notion of canonical messages. When mapping between source XSDs A and C with an intervening standard such as the SID, the XSLT mapping file transformations created are A→SID and SID→C. The return from the target to the source results in two more transformation files, C→SID and SID→A, for a total of four transformation files for each source-to-target mapping.

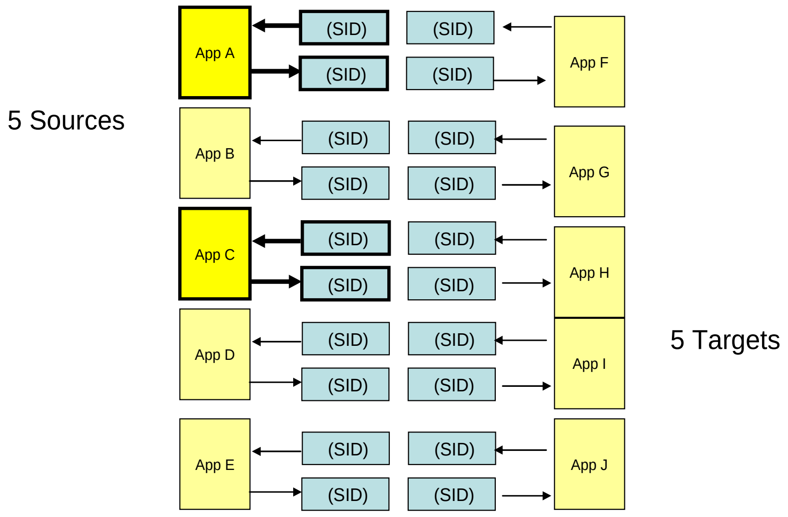


Figure 3: 20 Copies of the SID Schema and 20 Mappings

The sheer number of mapping files created causes considerable governance overhead because any XSD change will require re-testing of every mapping file to ensure accuracy. This testing becomes even more critical and manual if developers had to write XSLT in these mapping files to accomplish complex transformations. (See the complex transformation example on page 6.) As we will see later in this section, the impact analysis provided by these tools is not sufficient to determine which mapping files need to be tested.

Implications of Mapping the SID in a Point-to-Point Architecture

In this EAI point-to-point architecture, the canonical model is represented by a physical XSD which is processed on the message bus and picked up by applications as needed. Each application must map to this canonical model in order to complete the integration.

Mapping the SID in EAI technologies is extremely difficult for at least two key reasons:

⇒ EAI mapping tools will be a productivity burden if the developer must navigate across all 1080 classes to complete a mapping. EAI mapping tools cannot distinguish some classes from others. Every class has the same priority as every other class. Further, the tools cannot show the developer which attributes are left to map. Assuming a GUI that shows mapping lines between schemas, the number of mapping lines between the thousands of attributes becomes overwhelming.

⇒ Second, it is likely that all XSD constraints that would check for data syntax accuracy would have to be moved out of the schema and into BPM to enforce data consistency. With a schema that large, it is very likely that at any one point, information is not available to satisfy a given rule, and thus the rule would be violated. This will "pollute" BPM with specifics, preventing re-use, and dramatically slowing performance. (See the sub-section "Dynamic XSD Constraint Processing Pollutes the BPM" on page 18.)

Current Practice Alternative #2 – Mapping Smaller XSDs with XSLT Templates

One alternative to working with the whole complex SID is to work with subsets based on the SID packages described in UML, such as Customer, CustomerBill, and so on. Each XSD is provided by the telecommunications standards body with all of the direct data elements needed and includes all of the additional data elements from other packages that are needed to fulfill the structure.

Figure 5 shows the mapping of a CRM system to the SID XSDs `Customer` and `CustomerBill` where both schemas share common classes and attributes represented by "C."

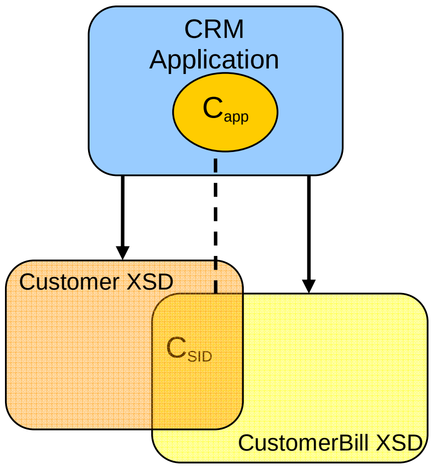


Figure 4: Mapping of a CRM System to the SID XSD

Because of the point-to-point architecture of EAI mapping tools, which results in the creation of two physical transformation files — one for each unique mapping of the CRM application package to the SID XSD — developers would normally have to duplicate the mapping work of "C_{SID}" in the SID XSDs to the "C_{app}" of the CRM application.

Rather than creating duplicate mappings between the "C" of each package and the equivalent attributes in the CRM system (and to build re-use in this architecture), developers will use XSLT templates to map common attributes across packages.

While yet another transformation file has now been created $C_{app} \rightarrow C_{SID}$, bringing the total to 3 as shown in Figure 5 (or 5 if a round trip mapping is required), this XSLT template technique does reduce the need somewhat for duplicate mapping and work.

Implications of XSLT Templates

While XSLT templates promote some re-use in the architecture, it comes at a very heavy governance price.

First, developers must resort to coding XSLT if they are building templates. EAI mapping tooling does not provide UI support for XSLT templates. The net result is code; code that is not documented and not captured in any sort of metadata.

Second, re-use is very limited. The XSLT templates can only be re-used if the source and target attribute names and structure are identical across applications.

For example, if we now want to map an ordering system to the SID Customer XSD, then that ordering system must have attributes that correspond to " C_{app} " spelled identically to the CRM system attributes, if the XSLT template is to be re-used. There is no tolerance for discrepancies.

As the scale increases with additional systems and business processes spanning multiple projects, the governance of all of the unique transformation files and all of the custom XSLT templates becomes very challenging and time-consuming.

EAI tools do not provide automated governance capabilities to alert the developers that a given change will cause a given transformation to stop working because the XSLT template that is associated with it will not work any longer. The corporation is left to enforce governance manually.

DataXtend SI Delivers a Common Model Architecture (CMA)

DataXtend SI implements a true common model architecture which enables corporations to combine all of the best practices associated with canonical message development and deployment and radically simplifies the effort through automation and full metadata capture.

Without DataXtend SI, each individual best practice must be manually governed which increases complexity and risk dramatically as SOA projects scale. DataXtend SI delivers these best practices (canonical message deployment, re-usable assets, data consistency and impact analysis) in a centralized design environment.

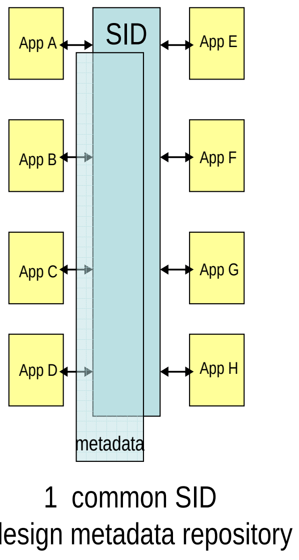


Figure 5: A Common Model Architecture

The considerable benefits of a common model architecture include:

1) No manual governance of mapping files and schemas.

   One metadata repository, one set of mappings against one common model makes the lifecycle management of an integration project far faster and easier. This common model architecture is why IBM's Telco group can claim that they can change out any COTS package in their SDP in two weeks or less.

2) An entirely re-usable asset within and across business domains.

   Because any extensions to the common model are maintained in metadata and checked in and out of source control, the integrity of the common model can be leveraged across projects effectively without custom code.

3) 360^{o} End-2-End (E2E) Impact Analysis.

   With a common model architecture, any interaction with the common model resides in the single metadata repository without resorting to multiple files that have to be coordinated, loaded into other file systems or scanned manually.

EAI is Missing 360^{o} End-to-End Impact Analysis Tools

To make changes quickly, systems professionals must know the cost of change before it has to be made. EAI technologies do not provide a full impact analysis of *how* or *where* an attribute is used, so you cannot determine the exact effect of a given change to an integration element such as a schema, class, attribute, rule, or operation. Thus, everything must be inspected and tested, creating long test cycles for even the smallest change.

DataXtend SI's impact analysis reports on the use of any attribute, object/class, rule, schema and operation, fully integrated with and interactively accessible from DataXtend Designer.

Current EAI Best Practice Uses an Impact Analysis Repository

Some EAI technologies, such as Tibco's XML Canon, provide a limited XML repository which can index XML schemas and provide limited search capabilities on parameters such as *name* or *name space*. However, this type of repository has some very large limitations.

Implications of Using an EAI Impact Analysis Repository

⇒ It can only show direct dependences of the affected attribute with no levels of reverse chaining. For example, if an attribute that was changed from numeric to string is used in a function or global variable, the impact analysis won't find that function or variable. You are left to manually search each individual XSLT file to figure out what has to be changed and re-tested.

⇒ It cannot show you XSLT that had to be hand-crafted to accomplish complex transformations, XSLT templates or global variables, which are indicative of relationships that must be tracked to determine elements of potential change.

⇒ It cannot show you custom code that had to be written to enforce data consistency and integrity (inter-field dependency constraints).

⇒ It cannot show different versions of a schema. If a new common model is made available from the standards bodies, system professionals should be able to verify the changes that the new model would cause to the existing integration projects. The only way to do this in these XML repositories is to manually determine the difference between the new and old schema and then search each change individually – a very laborious and error-prone process.

DataXtend SI's Answer to Impact Analysis -- 360^o End-to-End (E2E)

DataXtend SI provides an end-to-end understanding of any change to the integration architecture across all applications in the integration without having to manually load schemas into a separate metadata repository.

DataXtend SI's impact analysis provides complete reverse chaining through all of the indirect associations and relationships to the affected attribute, object, class, rule, schema or operation.

DataXtend SI's impact analysis not only shows every schema, class and attribute, it also captures every rule created (data consistency, computed attribute and semantic routing) on any piece of the integration.

DataXtend SI's impact analysis can also compare whole schemas to the existing integration. For example, if SID v.10 has just come out and integrations are in place with previous versions, DataXtend SI's 360^o E2E impact analysis analyzes and tells analysts and developers the impact that the new model will have on the entire project.

Even the operations that are executed are understood in terms of the impact of change, providing a complete and accurate picture of what has to be re-tested to ensure that all of the pieces continue to work.

Performance and Run-Time Challenges

EAI uses point-to-point mappings for integrating applications resulting in a proliferation of mapping files. These files must be invoked by external business logic, either in the service bus or in the BPM.

System performance issues discussed here include:

- ⇒ Using XSLT
- ⇒ How point-to-point mapping affects the business process

XSLT Causes Significant Performances Challenges

As the SOA scales up and XML messages proliferate in size and number, performance takes a "nosedive." This is often due to using XSLT to transform XML to XML. For certain transformations, XSLT offers more than one solution, further complicating the task of writing XSLT scripts because performance and efficiency issues need to be considered.

As previously discussed, a typical EAI mapping between source XSD (A) and target XSD (C) results in two unique XSLT mapping files, A→C and C→A, and any necessary custom code does not carry over from the first mapping. So for each pair of systems interacting with each other, there are at least two mapping files and associated custom code. The complexity increases dramatically as the number of systems increases.

Current Best Practice Solutions

The issues around XSLT performance are well known and have led a number of vendors to provide the following solutions:

- ⇒ Compile the XSLT in the vendor's runtime (Tibco).
- ⇒ Provide XML accelerators to increase performance (DataPower).
- ⇒ Provide XSLT compilers (XSLTC) that create Java classes to speed up performance (Sun).

IMPLICATIONS OF PERFORMANCE "SOLUTIONS"

Each approach introduces into the architecture either proprietary runtimes (which mitigate the value of using XSLT), or additional points of interoperability and change management (such as managing support for consistent XSLT syntax across all of the additional accelerators). As with any programming language, XSLT development requires governance and an adherence to standards to minimize architectural complexity.

DataXtend SI's Answer to XSLT Performance Issues is to Generate Java Code from Metadata

DataXtend SI's CMA in run-time, stateless Java, uses model-driven semantic mapping logic to provide automatic mediation, transformations, and routing in memory without relying on external invocations from other technologies. DataXtend SI's Java run-time improves performance over XSLT without resorting to external accelerators.

EAI Limitations often "Pollute" and Tax the Business Process

BPM tools found in EAI suites provide powerful technology to orchestrate and call activities and perform many programming-like functions using industry-standard languages like BPEL. These tools are often used to overcome deficiencies in other components of these suites. The effect, however, diminishes the re-usability and scalability of processes created with BPM tools.

Mapping Tool Limitations Drive Developers to Overburden the BPM

The mapping tools of the major EAI vendors are point-to-point in nature. For simple EAI mappings from one schema to another, the transformation takes one input document and creates one form of output document (A→C).

However, suppose that there are two choices for document A's transformation, as in two different inventory systems (C or D). Ideally, the tool should be able to do *semantic routing* by defining rules that can analyze the data at runtime and automatically determine and formulate the correct message interface for the ESB to route it to the appropriate physical system.

Current Practice Is To Write Code in the BPMS

But typically, to invoke the transformation A→D instead of A→C, a developer must write code in the Business Process Management System (BPMS) or in the EAI message broker to choose the correct mappings. How else does the integration technology know which set of mappings to execute in XSLT?

Implications

This approach does not scale because it "pollutes" the BPMS, and hence SOA, with references to schema names and mapping file decisions that have nothing to do with the BPMS.

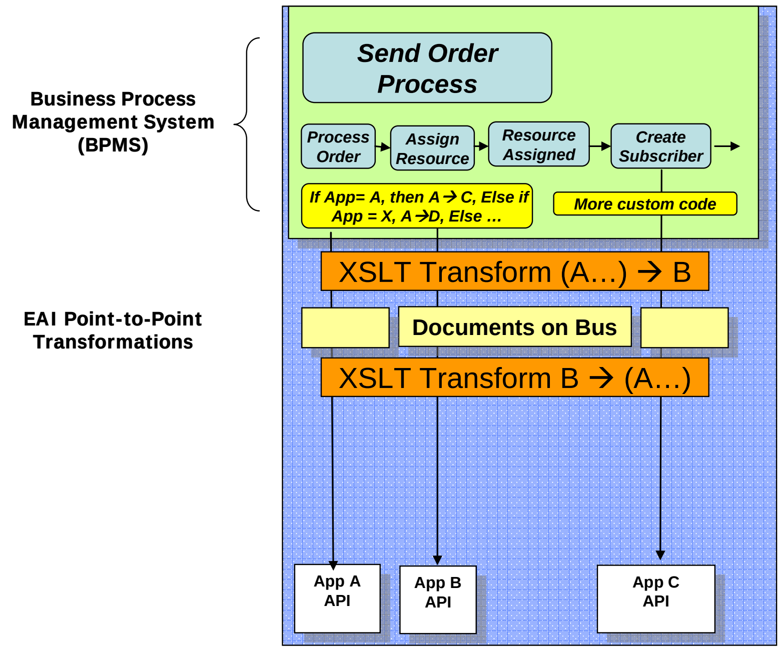


Figure 6: Point-to-Point Mapping Forces Custom Code into the BPMS

DataXtend SI Provides Model-Based Semantic Routing and Mediation via CMA

DataXtend SI's Expression Builder can manage transformation and routing based on the message content, and map preconditions defined in the SID without requiring low-level code. With DataXtend SI, conditional logic for data routing and transformation is defined once in the model and captured as metadata, rather than in code.

Through the use of a common model and DataXtend's integrated Expression Builder, the decision to generate transformations for one schema instead of another is activated in runtime metadata dynamically by the firing of rules and mappings. No specific schema names or endpoints have to be specified at runtime.

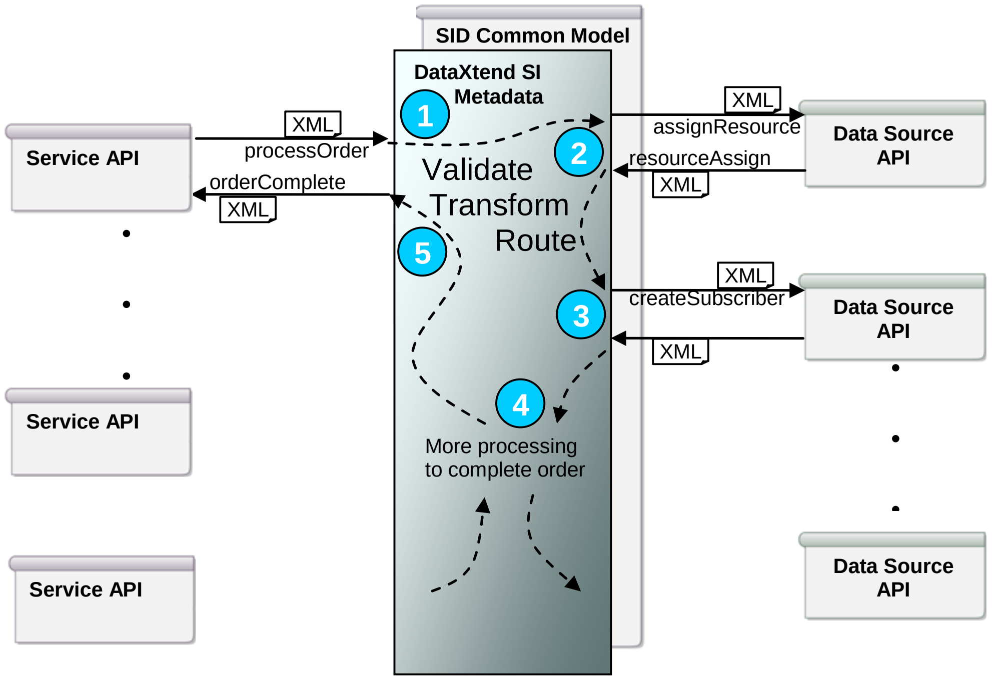


Figure 7: A Sample Semantic Mediation in An Order Processing Implementation

This example uses the SID as the common model between a number of application interfaces (mostly Web services) from various vendors, with operations spanning many functions. The implementation runs over an ESB application server. As an order is processed, the web services APIs pass messages in the form of XML documents that represent aspects of the order. The SID common model intervenes between all application interfaces. DataXtend SI tools provide the data and operations for validating, transforming, and routing the XML document messages throughout the system and between the various vendor APIs.

With DataXtend SI, the business processes stay generic, invoking only business operations. The BPMS contains no technical decision nodes to invoke a given set of mappings. The analyst is able to understand the business process much better without extra custom code, and re-use of the business processes are possible. Finally, DataXtend SI's runtime performs considerably faster because all of DataXtend SI's semantic routing takes place near the data and in memory, where it was designed to run efficiently.

Dynamic XSD Constraint Processing Overburdens the BPM

Executing XSD constraints in large canonical messages is another area where the BPM becomes overburdened and performance suffers. As noted above, a large canonical model might be architected to simplify governance challenges. However, because the XSD is so large, some of the constraints must be conditionally enforced because all of the information might not be available to ever fire the rule in a specific period of time.

For example, in an order provisioning process, there may be a requirement that every order must have an estimated provision date. But when the order is first taken within the CRM system, this information is not yet known and the estimated provision date is still blank. This would cause an XML parser to fire and fail for the wrong reason.

This kind of conditional data integrity issues becomes very significant when a large standard like the SID model with over 1080 class types is used. Because it is so large, data syntax rules spanning multiple processes would all be in one XSD, but need to execute differently depending on the runtime context.

Current Practices For XSD Constraint Processing

Two options are considered to deal with this validation:

⇒ Develop an XML Custom Parser.

Some developers use a 'validating' XML parser and a custom error handler to catch and process errors. However, they will still have trouble interpreting the errors. You cannot relate them back to actual locations in the XML document and there is no easy way to customize the error message to be more meaningful. Custom code becomes standard fare with this option.

⇒ Validate documents in the BPM.

The other approach is to build validation utility classes that can be invoked conditionally by your BPM system so that you don't have validation code containing parser error handlers all over your project which would be hard to maintain.

IMPLICATIONS

Most developers migrate to using the BPM approach as there is less custom code and more metadata captured about the rules. However, this approach has the following important implications:

⇒ System changes are difficult.

Business process analysts will not easily follow the business processes because the custom code includes considerable technical information that is not their expertise.

⇒ Re-use of the business process is lost across other integrations.

Custom code makes the ability to change out systems much more challenging, given that this kind of code exists throughout this implementation for invoking mappings.

⇒ Performance slows.

Performance suffers with so much logic and interaction complicating the BPM. In this approach, the BPM becomes heavily weighted with extra code, mapping decisions, and sub-processes that are data integration details instead of business processes. XSLT accelerators cannot fix this performance hit because they do not influence the performance of the processes.

DataXtend SI's Approach to XSD contraints

DataXtend SI can enforce all of these constraints in memory with conditional expressions that can ensure that these rules fire at the right time to ensure integrity. The constraints are imported automatically with the XSD and can be modified with if/then/else expressions to support the business conditions.

The BPM remains simple and free of any low level data integrity decision nodes. Analysts can continue to understand and modify the business processes as appropriate.

Performance is also maximized as the BPM is invoked only on process and workflow changes, not on integrity checks that belong at the data tier.

Conclusions

As SOA adoption grows, corporations are increasingly faced with the new challenges of scaling an SOA. The requirements are to deliver:

⇒ Flexible and agile systems that easily interoperate;

⇒ Model-based data consistency based on business rules;

⇒ Correct data in a dynamic runtime environment.

All while improving productivity, reducing manual governance, and optimizing performance.

EAI tools, while enabling a services-based business process environment, don't by themselves allow IT departments to readily meet these requirements. This situation has spawned a number of novel approaches, some commonly used, but which, when applied to one issue, often exacerbate another.

The solution seems to center on using an industry standard data model (i.e. SID) as the basis for mapping data between applications, since it overcomes the most obvious issues of hard-coding point-to-point XML transformations directly in XSLT. But use of the model by itself still falls short of solving more esoteric issues such as enforcing data consistency rules and executing conditional constraints at runtime.

DataXtend SI's suite of tools was designed to deliver a semantic metadata-driven solution that simultaneously meets this specific set of requirements, working within an EAI environment using a model such as the SID.

Semantic solutions hold promise but tend to be unfocused and unproven. That is what sets DataXtend Semantic Integrator apart. DataXtend SI has been chosen and is being deployed because of unique capabilities that enhance an EAI environment and enable SOA while improving productivity, reducing manual governance, and optimizing performance.

⇒ DataXtend SI tools support attribute customization for model expansion, and conditional logic for dynamic data routing and transformation at runtime, but store it as metadata separate from the model so the model remains *standard.*

⇒ DataXtend SI enables business analysts to define business rules, captured as metadata, enforced at runtime to ensure consistency, using no custom code.

⇒ DataXtend SI's E2E impact analysis takes time and risk out of making changes:

- Works across all applications;
- Provides complete reverse chaining through all indirect associations;
- Covers every schema, attribute, class, object and operation, as well as all rules.

About Progress Software Corporation

Progress Software Corporation (Nasdaq: PRGS) provides application infrastructure software for the development, deployment, integration and management of business applications. Our goal is to maximize the benefits of information technology while minimizing its complexity and total cost of ownership. Progress can be reached at www.progress.com or +1-781-280-4000.

PROGRESS
SOFTWARE

www.progress.com/dataxtend

Worldwide and North American Headquarters
Progress Software, 14 Oak Park, Bedford, MA 01730 USA Tel: +1 781 280 4000

UK and Northern Ireland
Progress Software, 210 Bath Road, Slough, Berkshire, SL1 3XE England Tel: +44 1753 216 300

Central Europe
Progress Software, Konrad-Adenauer-Str. 13, 50996 Köln, Germany Tel: +49 6171 981 127