Using the SID in OSS/BSS Integration

Challenges and Solutions to Implementing the TM Forum Shared Information /Data Model

PROGRESS SOFTWARE

Table of Contents

Table of Figures

Introduction

With the Shared Information/Data (SID) model, the TeleManagement Forum (TM Forum) has developed a common language for enterprise operations in the telecommunications industry. The TM Forum added XML Schema Definition (XSD) representations to the original Unified Modeling Language (UML) definitions for the SID model. The SID XSDs are an important advance, providing the basis for developing reusable data models for integrated business applications. Through Progress® DataXtend™ Semantic Integrator, Progress® Software provides key support for building robust integrations that gain the value of the SID as a common data model to promote speed, agility, reuse, and data quality in integration projects for operational and business support systems (OSS/BSS).

The tools that help you use the SID to implement OSS integration must easily mediate between the SID and other systems. The tools must provide:

- Complex data mapping and transformation
- Data consistency and validation
- Content-based data routing (semantic mediation)
- Input error management (remediation)
- Impact analysis of changes

This paper elaborates on these requirements for using the SID model within OSS/BSS integration projects and discusses the role of DataXtend Semantic Integrator (SI) in solving them.

Why Use the TM Forum's SID?

The Shared Information/Data (SID) model is an abstract common model that is a central component of the TM Forum's Next Generation Operations Systems and Software (NGOSS) initiative. The goal of NGOSS is to promote open, distributed OSS/BSS systems using commercial off-the-shelf technology. NGOSS provides a technology-neutral architectural framework for cooperation among component applications interfaces within larger system integration projects.

Loosely-coupled systems require data abstraction, which the SID provides as an industry standard model. Reuse requires that the standard model also be abstract to enable different parts of a business to share data. The SID is considered *the* telecommunications industry's standard abstract common model to use for integration. Figure 1 illustrates the advantages of an abstract common model.

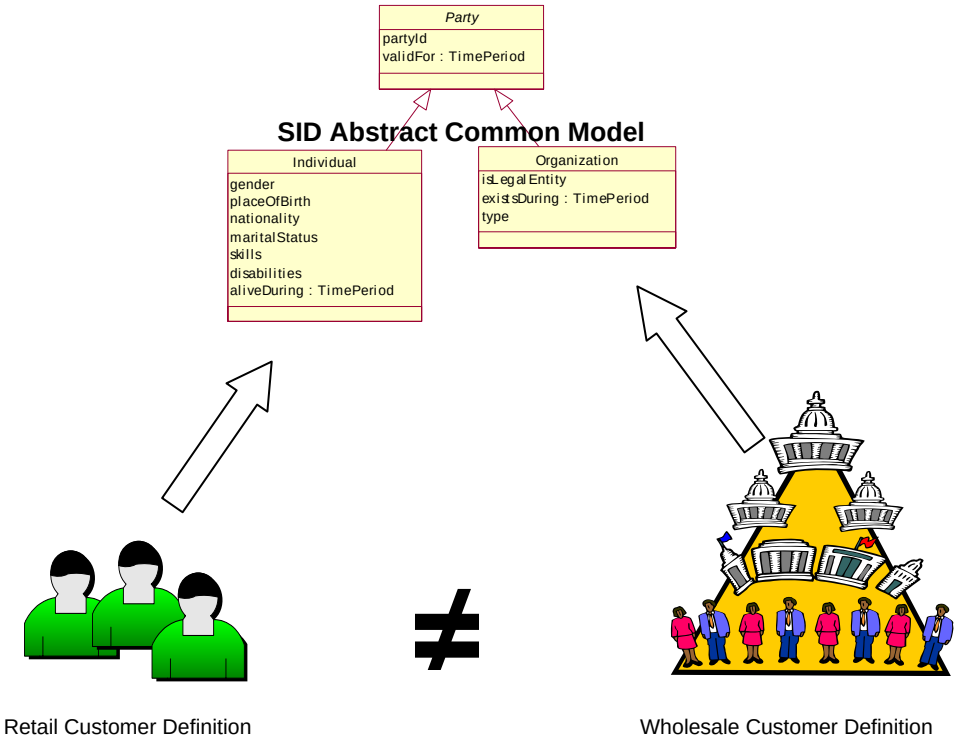


Figure 1: Abstraction Enables Information Sharing

A company's retail division and its wholesale division have different definitions of what a customer is. A retail division has a simple definition of customer, whereas the wholesale customer might include many other attributes such as VAT identifiers, among other things. The SID provides the Party abstraction, which allows individual customers and organizational customers to both be represented, enabling both divisions to share the same information and information model.

Comprising about 1,000 classes, the SID model ranges from very general concepts like the OpenGIS geometrical elements of points, curves, and surfaces to very specific concepts like the wide-area network (WAN) protocols PPP and X25. Although the SID may at first seem daunting, rapid acceptance of the SID for system integration will occur when you use implementation tools that easily navigate, display, and interact with the SID.

Figure 2 indicates how DataXtend SI's tools can dynamically navigate and integrate graphically with all aspects of the SID, from the packages, classes, and relationships, to the attributes and rules on the classes.

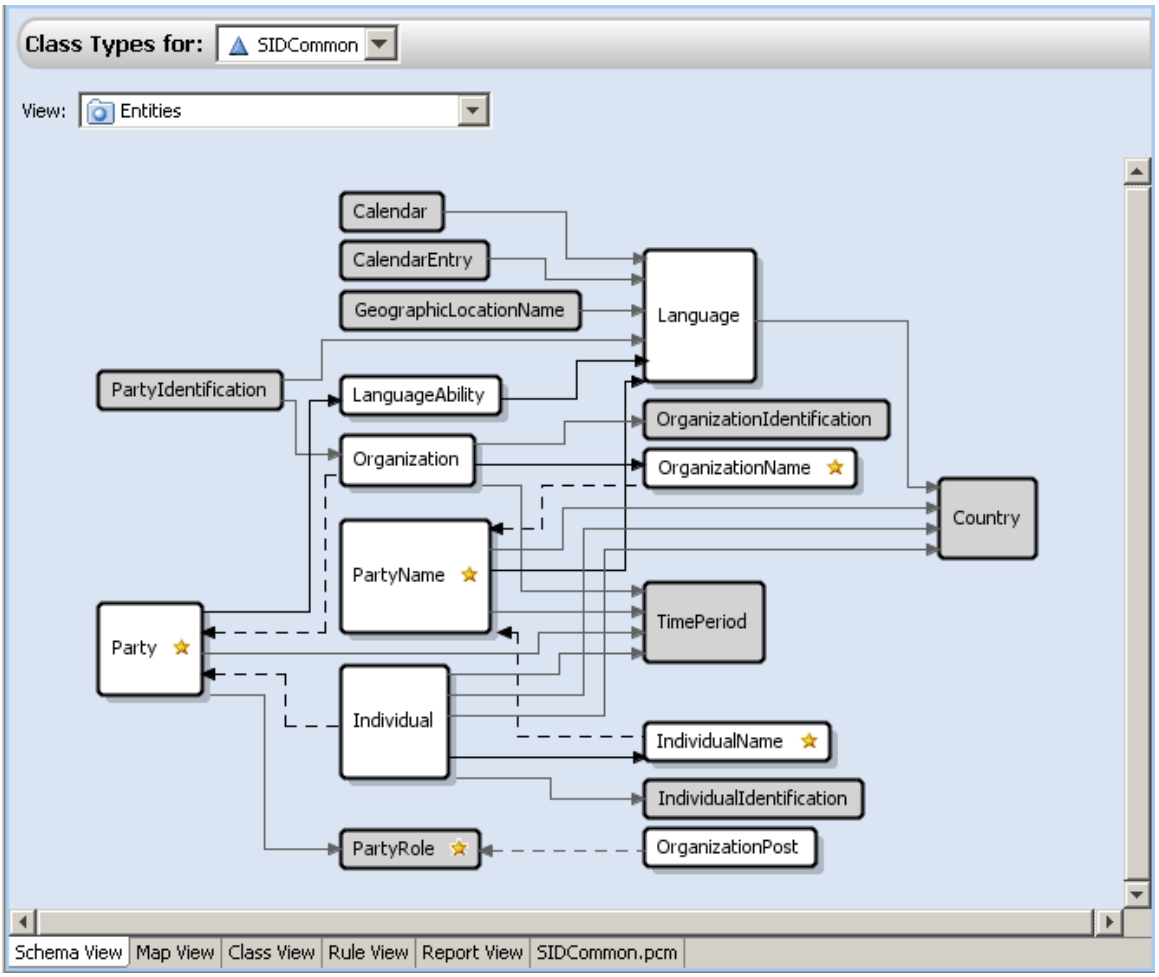


Figure 2: Graphical Tools Provide Interaction with the SID

A Triple Play Implementation Example

The examples in this paper come from a Triple Play implementation that provides the services for customers to order three key products:

- ⇒ Voice, using Voice Over Internet Protocol (VOIP)
- ⇒ Video, using the Internet Protocol Television (IPTV) protocol
- ⇒ Data, using high-speed Internet (broadband)

This Triple Play implementation uses the SID as the common model between a set of application interfaces from various vendors. Over a dozen application interfaces with a total of nearly 100 operations span functionality that includes

inventory, service assurance, customer relationship management (CRM) activation, product pricing, and order management.

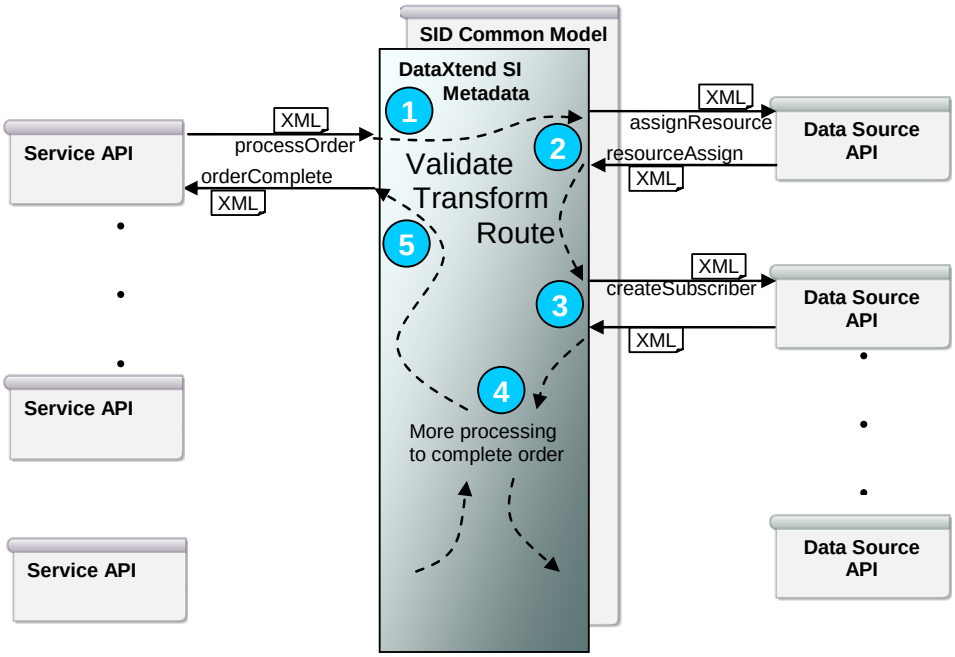


Figure 3: A Triple Play Implementation's Processing

The implementation runs over an ESB application server infrastructure. Most of the application programming interfaces (APIs) are Web services from a variety of vendors. During processing of an order, the Web service APIs pass messages in the form of XML documents that represent aspects of a triple play order. The SID common model is the intervening data model between all the 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.

Complex OSS/BSS systems such as this one with many loosely-coupled interfaces need the abstraction of the SID to provide for rapid and flexible data integration. The rest of this paper elaborates on what the implementation tools need to provide to make this integration happen.

Complex Data Mapping Between the SID and Other Systems

The SID is organized from a very general perspective because it's an abstract model. Data services and sources, however, often contain "flat" data structures because they typically represent APIs of systems that are older and more narrowly defined. This discrepancy sometimes requires some complex mapping between these relatively simple external data structures and the more abstract SID details.

You will likely find that the SID does not have all the attributes where you want them as you map attribute to attribute for an implementation. Customized attributes can be created that do not exist in the SID but are needed by specific data services or sources. As you create a customized attribute, you can also associate an expression with it to automate the attribute's data conversion at runtime.

It is important for implementation tools to provide customization without changing the common model itself, so that the model can remain a standard and be easily upgraded when a new version is published. DataXtend SI tools store all customized information as metadata separate from the SID model so the SID is unchanged.

Some features of the SID that should be customizable without directly modifying the SID itself include:

- ⇒ subclasses
- ⇒ simple attributes
- ⇒ computed attributes
- ⇒ rules

For example, suppose you need a customer name in a format not provided by the SID. The following figure shows a DataXtend SI tool display of the properties of the SID's **Customer** class.

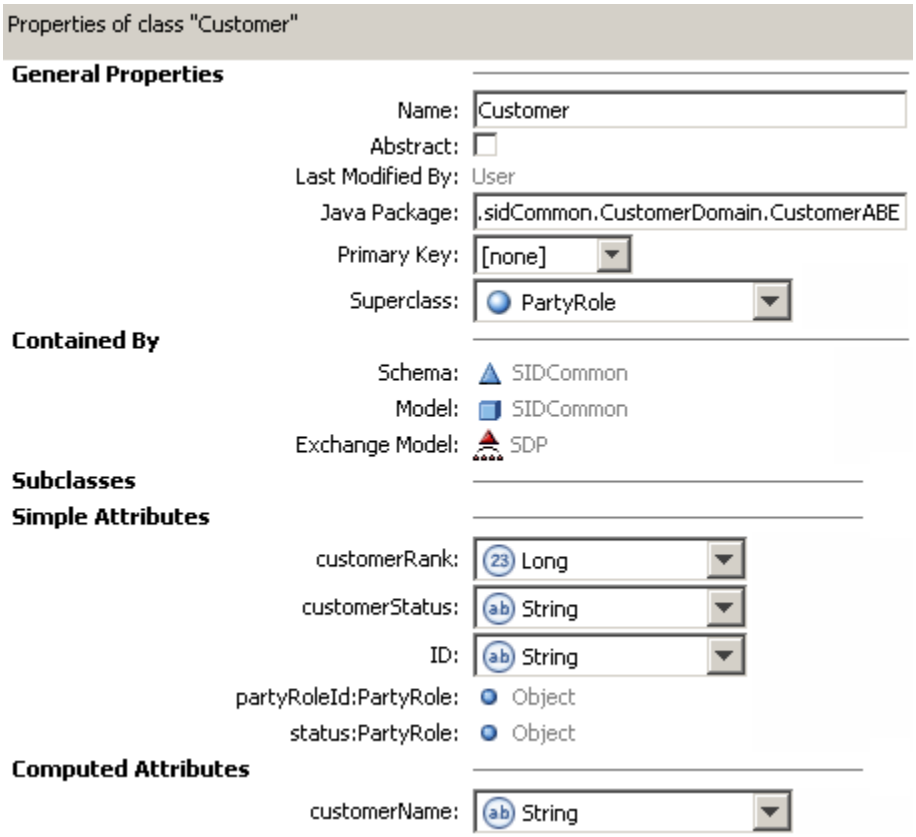


Figure 4: Properties of the SID's Customer Class

Notice that the **Customer** class does not have a customer name property. You can add a computed attribute, such as **customerName**, that computes the name at runtime. For example, a computed attribute's value could be assigned a value from some other attribute in a different class of the SID.

A customer's name might be an organization or an individual. Tools such as DataXtend's Expression Builder can create an expression that computes the attribute's value at runtime. Such an expression might be described as follows:

If the customer ordering a service has its party role as an organization,

then the **customerName** evaluates to the **tradingName**, which is a simple attribute of the SID's **OrganizationName** class.

Otherwise, the **customerName** evaluates to the **fullName**, which is an attribute of the SID's **IndividualName** class. (In this case, **fullName** is a computed attribute that concatenates simple attributes representing parts of a person's name.)

The following schema diagram illustrates from where the computed attribute **customerName** gets its value.

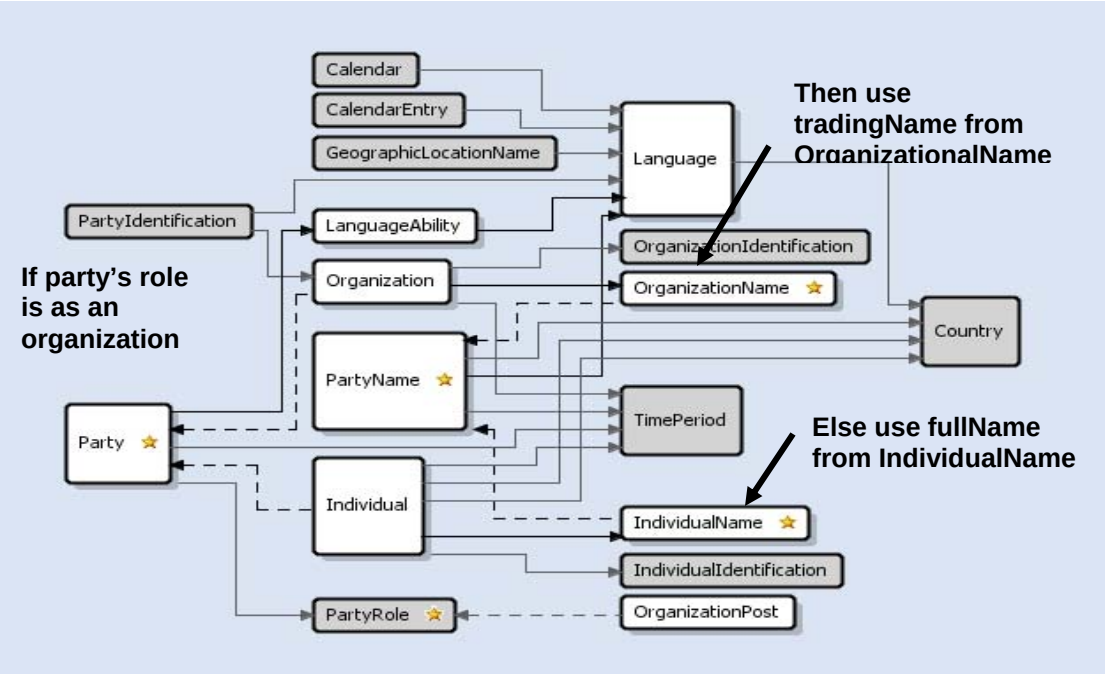


Figure 5: An Example of How a Computed Attribute Might Get its Value

The implementation tools need to be able to easily navigate the complexities of the SID common model, and the data models of the other application interfaces. The mappings need to be easy to graphically create and modify. The following figure shows an example of how DataXtend SI's tool maps the **customerName** computed attribute from the SID's **Customer** class to the **Name** attribute in a data source's **SubscriberInfo** class.

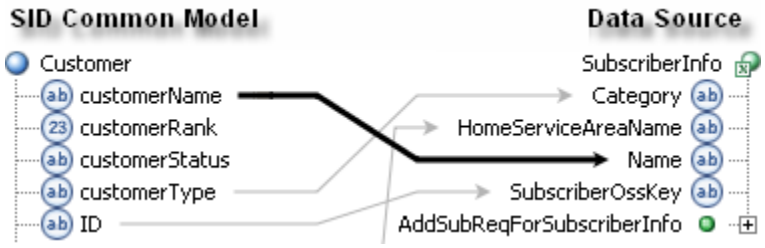


Figure 6: Mapping from the SID to a Data Source

Note also that the gray arrows in Figure 6 indicate how the implementation tool is used to make all simple and complex data links required by the implementation.

Model-Based Data Consistency Based on Business Rules

Any useful implementation tool must provide a mechanism for applying business validation rules to the data in order to keep the data consistent between applications. For example, a business validation rule for a Triple Play implementation might be:

A subscriber of voice over IP (VOIP) must also subscribe to Internet broadband service.

This kind of business logic contains the following characteristics:

1. These are rules that cannot be enforced in XML alone. While XML does an excellent job of ensuring the format integrity of the message (field lengths, numeric versus alpha-numeric and so on), XML cannot ensure inter-field dependencies or conditions like this VOIP example.
2. These rules are often obvious to the business yet are not consistently defined or implemented across multiple integration projects and can cause significant data quality or operational issues. Typically, these rules are written in procedural code all over the architecture — in the adapters, or on the bus.
3. These rules are too simple to warrant the use of business rules engines, which can perform very complex rules processing but at a heavy expense in terms of performance and training.

We call these rules validation or semantic rules as opposed to “business rules.” Business rules managed by business rules engines are typically associated with performing calculations such as tax or pricing computations.

Implementation tools should be able to easily define any rule like this without having to do any coding. See the following figure.

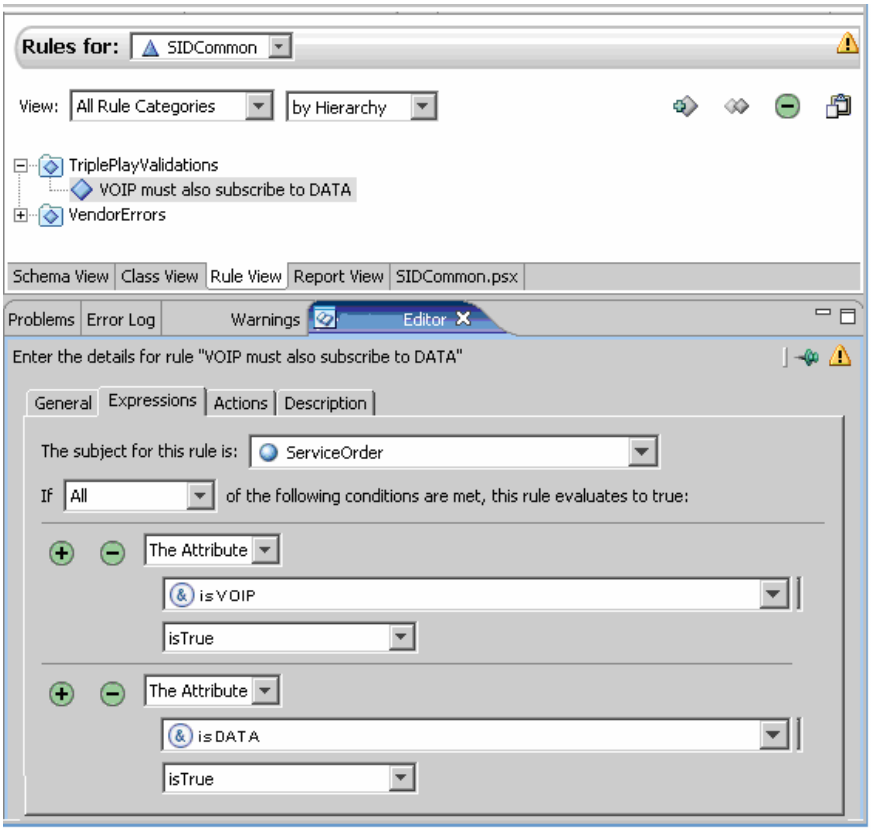


Figure 7: Defining a Business Validation Rule

In this example, DataXtend's Expression Builder is used to create the following rule for the **ServiceOrder** class.

VIOP must also subscribe to DATA

The rule checks to see if both Boolean attributes **isVIOP** and **isDATA** are true, and if so, the rule evaluates to true.

Model-Based Semantic Data Routing

Good implementation tools are able to map any data item to a SID data item, without requiring any custom coding. Also, by using computed attributes as shown in the previous section, automatic data transformation can be achieved.

However, the implementation tools should also 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 physical system. Semantic routing is a distinct but necessary routing compared to message bus routing, which occurs after the application interface message has been created and where the ESB routes the messages to the physical system endpoints.

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 on the model and captured as metadata, rather than in code. This enables reuse and quick implementation for any

change. Without this capability, you would have to write complex if-then-else coding that is not reusable either as custom code in your ESB or in your BPM. Either way this logic is not captured in metadata with the other metadata of the SID, even though it is metadata that is closely tied to the SID.

For example, suppose that even though a triple play order might be for all three services, the implementation might require that orders be routed to different data sources depending on the service. In the backend, the broadband order might go to one data source, the VOIP order might go to another data source, and the IPTV order might go to yet a third data source as shown in the following figure.

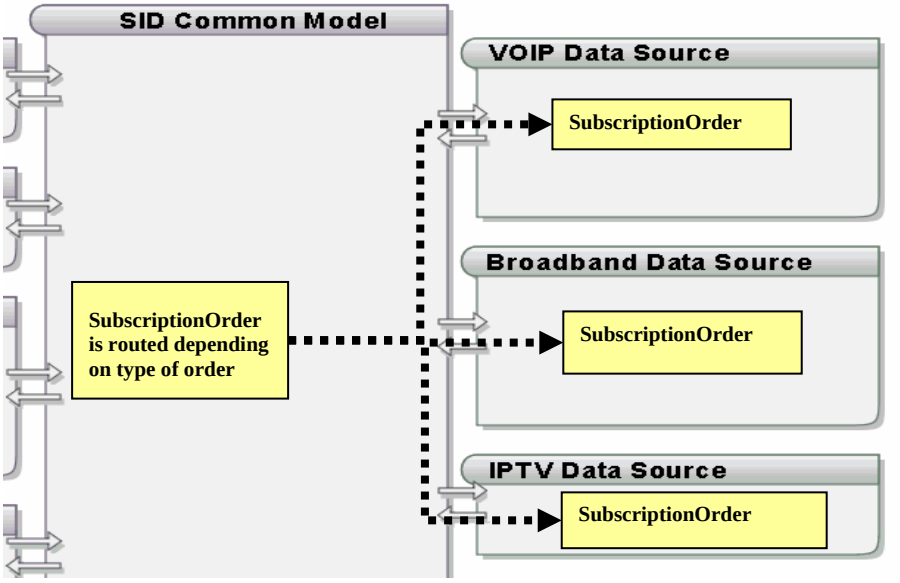


Figure 8: Model-based Semantic Data Routing

This routing is based on the semantics of the model, in this case most likely driven off of the kinds of products that are being ordered, which are already understood and captured in the SID.

Model-Based Error Management to Handle Inaccurate Input

Implementation tools must provide data validation to avoid sending inaccurate data into the system that could corrupt data sources and databases. Schemas already provide some level of validation for data input errors to avoid such errors as accepting strings when numerical values are required, for example.

Most organizations have written custom error handling systems in an attempt to lessen the labor involved in error handling. However, these are often project based and not for the enterprise as a whole. Also, any effort spent on an enterprise-wide error handling system is not captured in metadata for easy re-use across projects.

More sophisticated model-based error management would return informative messages and reject bad input or perhaps even correct some input errors. Model-based error management should not end on the first failure but should evaluate

the whole message regardless of failures. XSLT alone cannot provide this sophisticated evaluation and thus must be “overridden” with custom error handling code to provide a more complete analysis.

Secondly, when an error occurs in model-based error management, the returned response should be declarative and rich in details so that the Business Process Management application or other receiver of the response can respond accurately. Current technologies such as XSLT return only cryptic error codes with little detail, requiring analysts to spend significant labor interpreting the error code and determining the cause of the error.

A model-based error management approach enables analysts to describe an error response for a rule, such as with DataXtend SI’s tools as shown in the following figure.

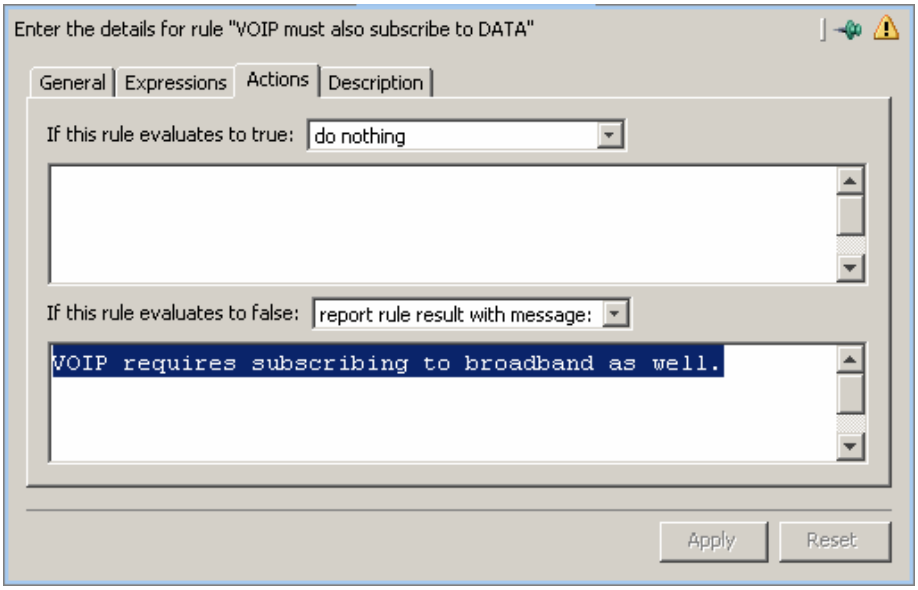


Figure 9: Model-based Error Management

Recall this rule (established on page 10) required a subscription to both broadband service and VOIP if a subscriber requested VOIP service. Error messages could be as detailed as necessary and can include variables that are interpreted at runtime to greatly enhance interpretation after an error has occurred. In DataXtend SI’s tools, if this rule evaluates as false at runtime, this message gets added to a **RulesResults** list (shown in the following figure) which can be interpreted by an API’s operation.

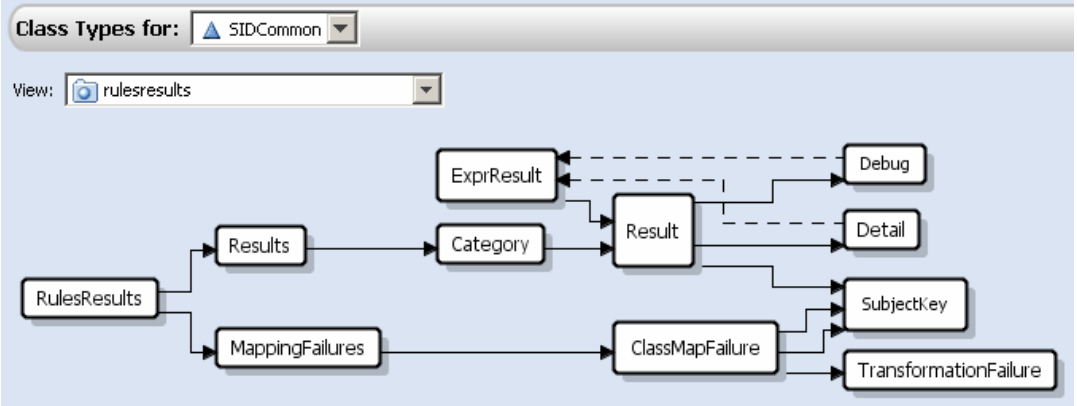


Figure 10: RulesResults Contains the Error Messages

Impact Analysis to Manage Change

As systems become larger and more complex, impact analysis provides techniques and tools to predict and control the effects of software changes. You need to know what parts of the implementation affect each other and how. Implementation tools should be able to provide analysis on any SID and vendor entity, including customized features.

The tools should not only show *where* a specific entity is used in other schemas and so on, but they should also show *how* the entity is used by providing a clear understanding of the operations that use the entity. This includes all operations no matter where they are in the system, and therefore indicates which operations probably have to be retested when this change is made.

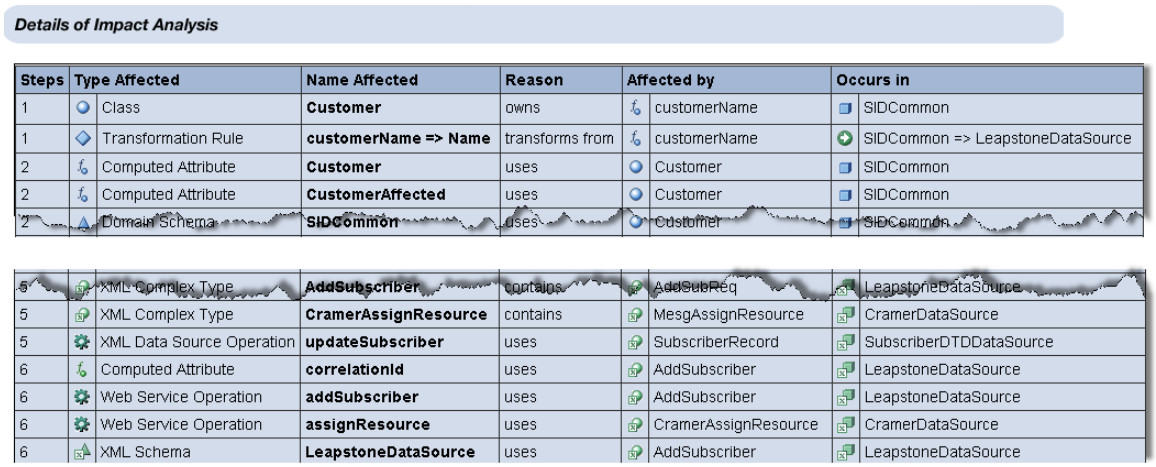
For example, the following DataXtend SI report shows where changes and testing would be required if changes were made to the **customerName** computed attribute.

Type Affected
1 Class
3 Computed Attributes
1 Domain Schema
6 Rules
4 Simple Attributes
4 Transformation Rules
6 Web Service Operations
13 XML Complex Types
1 XML Data Source Operation
2 XML Schemas

Figure 11: A Summary of the Impact of Changing customerName

From this table you can see that changes to **customerName** will affect 1 class, 3 other computed attributes, 1 domain schema, and so on. The actual class,

computed attributes, and other entities affected are available in a detailed report such as that shown in the following figure.



Details of Impact Analysis

Steps	Type Affected	Name Affected	Reason	Affected by	Occurs in
1	Class	**Customer**	owns	customerName	SIDCommon
1	Transformation Rule	**customerName => Name**	transforms from	customerName	SIDCommon => LeapstoneDataSource
2	Computed Attribute	**Customer**	uses	Customer	SIDCommon
2	Computed Attribute	**CustomerAffected**	uses	Customer	SIDCommon
2	Domain Schema	**SIDCommon**	uses	Customer	SIDCommon
5	XML Complex Type	**AddSubscriber**	contains	AddSubReq	LeapstoneDataSource
5	XML Complex Type	**CramerAssignResource**	contains	MesgAssignResource	CramerDataSource
5	XML Data Source Operation	**updateSubscriber**	uses	SubscriberRecord	SubscriberDTDDataSource
6	Computed Attribute	**correlationId**	uses	AddSubscriber	LeapstoneDataSource
6	Web Service Operation	**addSubscriber**	uses	AddSubscriber	LeapstoneDataSource
6	Web Service Operation	**assignResource**	uses	CramerAssignResource	CramerDataSource
6	XML Schema	**LeapstoneDataSource**	uses	AddSubscriber	LeapstoneDataSource

Figure 12: Some Details of an Impact Analysis

This illustrates a partial list of affected entities indicating the detailed affects for each. For example, the **Customer** class of the **SIDCommon** model is affected directly (1 step) because it owns the attribute **customerName**. For another example, the computed attribute **CustomerAffected** is 2 steps away from **customerName** because it uses the **Customer** class, which in turn owns **customerName**. Impact of a change is manageable out to any level in all the application APIs.

Useful impact analysis tools and reports can not only help assess the time and effort needed for a change, they can also provide help in deciding between alternate approaches to a change or fix to the system.

Conclusions

The TM Forum SID model will help telecommunications providers simplify OSS/BSS integrations by providing a common semantic model for mapping and transforming data. However, implementation tools must provide the following features and benefits for model-based implementations:

⇒ **Complex data mapping and transformation to and from the SID**
Graphically map attributes between the common model and other message formats, with the ability to define custom source expressions and computed attributes for data transformation where needed.

⇒ **Data consistency and validation not provided via XML**
Define rules for validity constraints using graphical expressions, without resorting to custom code.

⇒ **Content-based data routing (semantic routing)**
Define rules in the implementation that evaluate data at runtime and build the appropriate message for the appropriate application interfaces.

⇒ **Input error management**
Capability to model errors and detailed response documents which will enable organizations to model and define appropriate recovery paths, providing more satisfying and reliable user experiences.

⇒ **Impact analysis of changes**
Provide analysis of the impact of modifications that help manage the costs of change over the integration lifecycle.

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