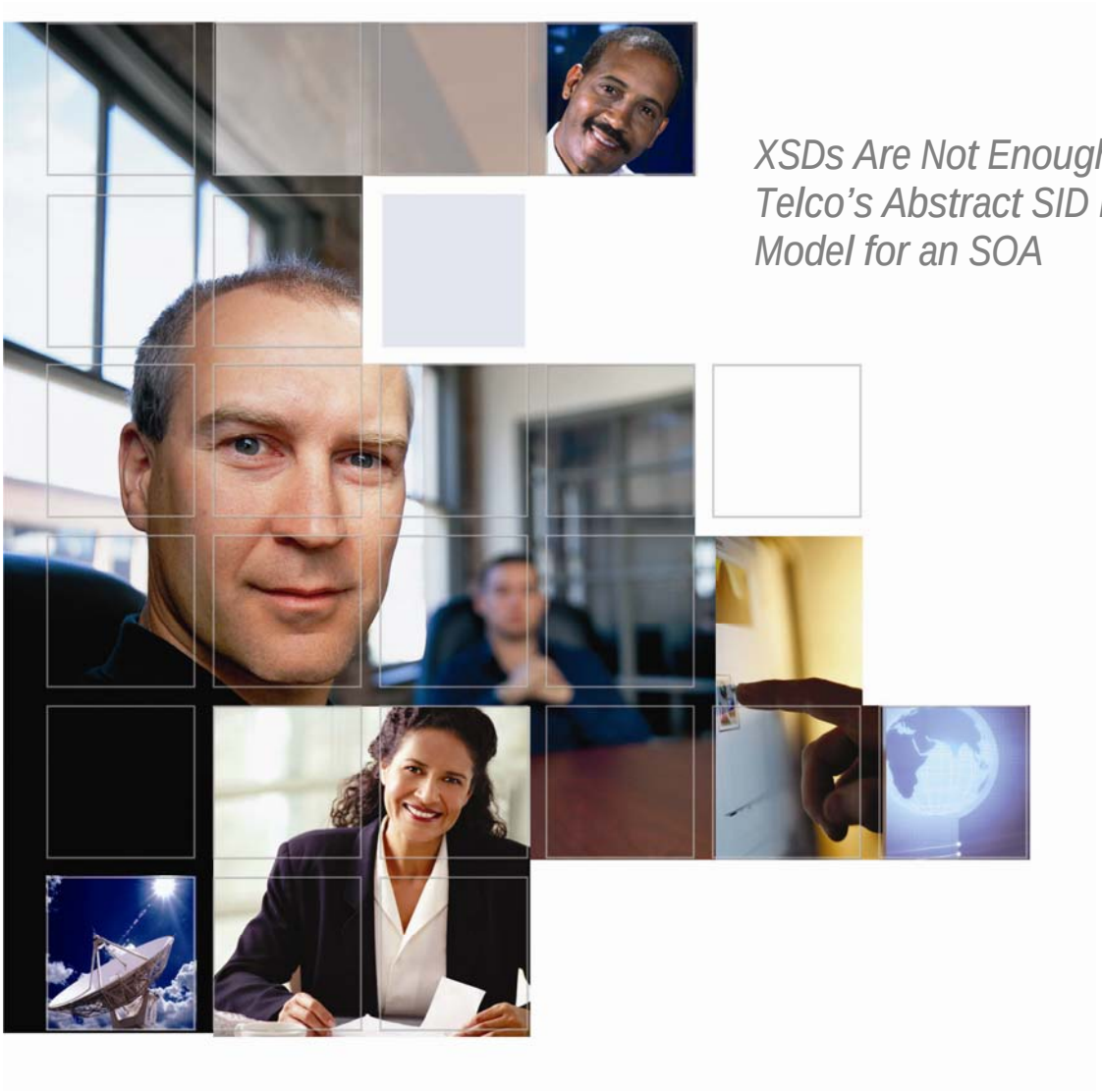

XSDs Are Not Enough: Leveraging Telco's Abstract SID Information Model for an SOA

PROGRESS
SOFTWARE

Table of Contents

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. But the XSDs are just the beginning. In real-world applications, the XSDs will have to be altered and enhanced. The need to change, develop, and maintain the XSDs presents challenges that cannot be resolved easily without additional tools. Progress® DataXtend™ Semantic Integrator (SI) 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 challenges of using the SID model in OSS/BSS integrations can be divided into two parts:

1. Extending and enhancing the SID model itself

2. Using the SID in the context of OSS/BSS integrations

This paper discusses part one: the need to modify, extend, and update the SID model and the role of DataXtend SI in making those changes faster, easier, and more maintainable.

SID: Promise and Challenge

The SID model 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 systems using commercial off-the-shelf technology. NGOSS provides a technology-neutral architectural framework for cooperation among distributed applications, using contracts to govern their interactions. A contract is an agreement between two components about how they will cooperate — not only the interfaces they will use, but the business processes and management capabilities they require.

Although they constitute a framework for interaction, NGOSS contracts do not solve the problem of semantic incompatibility and ambiguity among integrated

components. This is where the SID comes in. It provides a common language to represent business and data constructs and promote semantic interoperability between components of an OSS integration. Using UML, the SID model supports business, system, implementation, and deployment views of an integration project. Comprising nearly 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.

The SID model is designed to be flexible enough to apply to any reasonable OSS process. It is also independent of any particular implementation. The price of this generality is that the SID requires adaptation and extension when used as a data model in a real-world integration.

Fortunately, two developments have laid the groundwork for using the SID as a model in enterprise applications. The first is the TM Forum's release of the SID XSDs. The second is the development of OSS through Java (OSS/J) APIs. This combination enables telecommunications providers to implement interoperable data services on J2EE-based application servers.

The SID XSDs do not eliminate the need to extend and adapt the SID model in real-world integrations. Application developers who want to use the SID model immediately find themselves asking questions. What is the best model for extending the SID? How can we preserve reuse and keep maintenance costs under control? What tools are available to help?

SID: Extensions Required

The SID is an excellent reference model, but using it in an integration project presents challenges. In a typical integration, a customer relationship management (CRM) system may interact with order and inventory systems. Each of these may have multiple data sources and message formats. For instance, even a simple order system is likely to have different order and invoice formats for residential and business customers. Mapping the specific data elements of these messages to the SID model is a complex task with several common problems to overcome.

Deriving Concrete Data

The SID model makes extensive use of abstract classes with concrete subclasses. Sometimes abstract models require several steps to navigate to specific data elements.

For example, operations in an ordering or provisioning system might require a customer's home telephone number. In the SID model, Customer is a concrete subclass of the abstract class PartyRole. From PartyRole, Customer inherits a collection of ContactMediums. ContactMedium is abstract; one of its concrete subclasses is TelephoneNumber. TelephoneNumber has a type attribute that can be used to distinguish a home number from, say, a mobile number.

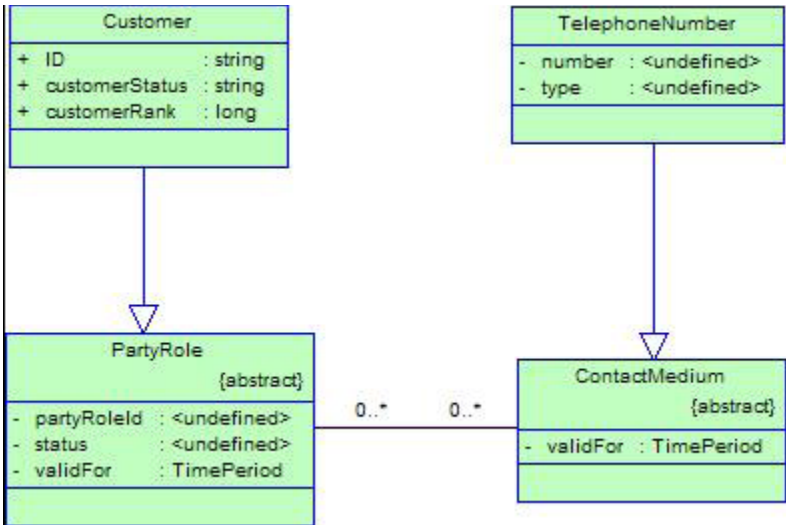

Figure 1. SID Class Relations for Customer and TelephoneNumber

In the case of home telephone number, the SID model has a class (TelephoneNumber) that represents the data we need. In other cases, the SID model has an abstract class that represents a more general concept, but the concrete data type is missing. For example, the SID model has no value-added tax (VAT) registration number. It does have an abstract class OrganizationIdentification, but we would need to define a concrete subclass of that class to use it to represent the VAT registration number.

Mapping Data to the SID Model

The SID model is modular, with separate but associated packages representing such concepts as customer, product, service, business interaction, and the like. In a typical OSS integration, messages contain data that corresponds to several of these domains. Mapping such messages to the SID model presents challenges.

For example, a typical product order message contains, at minimum, an identifier for the customer. In the SID model, the Customer class has an ID attribute that we would want to map to the customer identifier in the order message. But the path from the ProductOrder class to the Customer class, illustrated in Figure 2, traverses two associations and involves five generalization (subclassing) relationships.

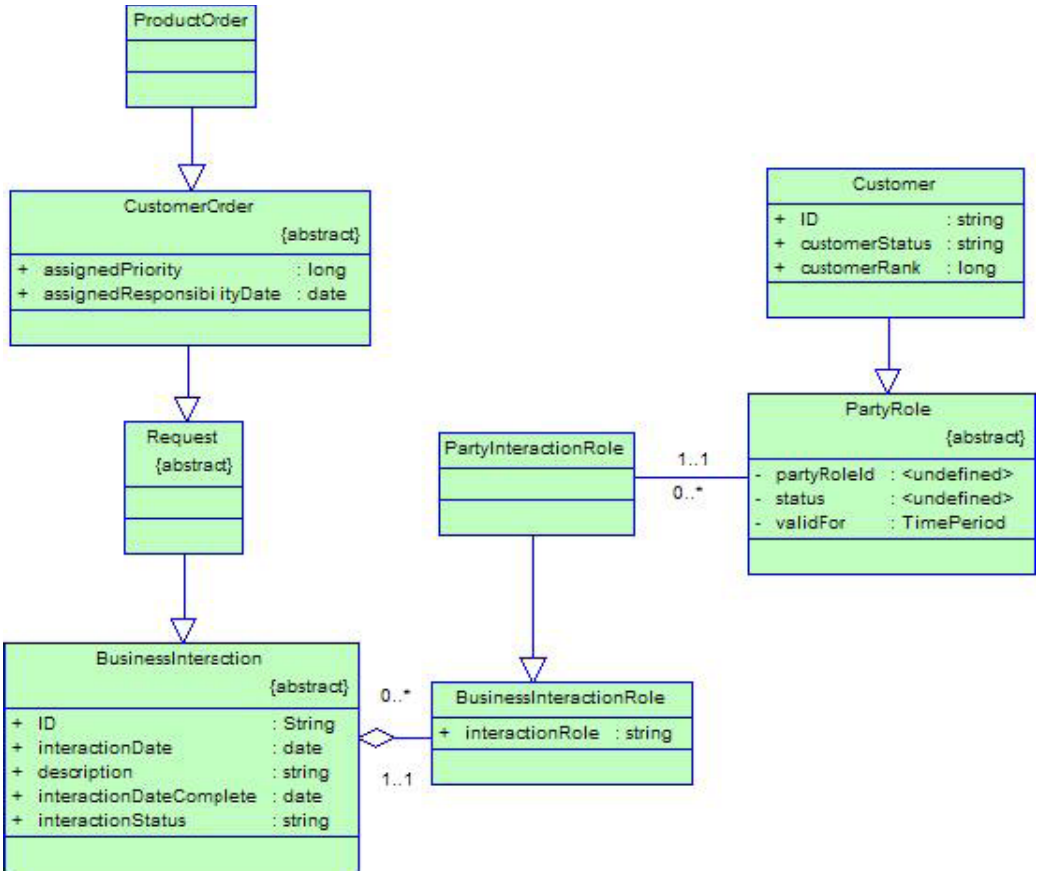

Figure 2. Class Relations from ProductOrder to Customer

Suppose we now need to retrieve the customer name for a downstream operation. This requires traversing two more associations from the Customer class, involving two subclass relationships, as illustrated in Figure 3.

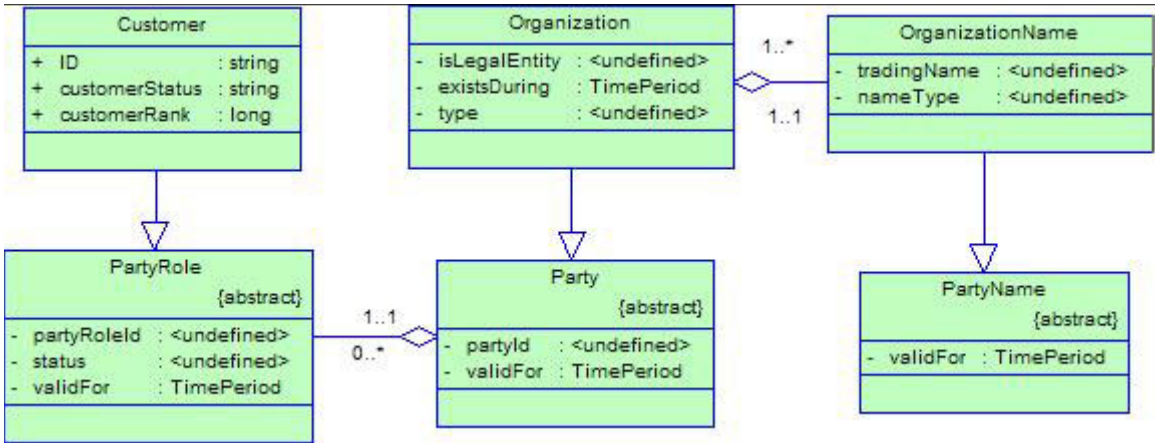

Figure 3. Class Relations from Customer to OrganizationName

Defining Enumerations

Most schema definitions include many enumeration types. Often, these definitions are accompanied by spreadsheets that detail the values for these enumerations. The SID model contains no enumerations. For example, the type attribute for the TelephoneNumber class needs an enumeration to identify such types as home, business, and mobile number, but this enumeration is not defined in the SID. Any integration using the SID needs to define enumerations and map them to enumerations in other formats, many of which are likely to have differing sets of values.

Defining Validity Constraints

Many of the challenges in ensuring data quality come down to enforcing constraints on data that are not explicit in the model, often because the constraints require comparison of two or more data elements. One example from the SID involves the ContactMedium class. Each ContactMedium has an associated time period during which the contact medium is valid. The TimePeriod has a lower value and an upper value. But there is no expression of the constraint that the lower value must be an earlier date than the upper value — that is, that the start date of the validity interval must be earlier than the end date.

It might be possible to express such constraints using UML's object constraint language (OCL), but the question remains how to translate these constraints to code and enforce them in the integration. Further, OCL has not reached the mainstream of development, as compared to Java or C#.

Performing Calculations

A typical business process requires calculations using data supplied in messages or data sources. A simple example is age, usually computed by comparing the current date with an individual's birth date — and ensuring that the birth date is earlier than the current date. Other common computations are financial calculations, such as available credit balance, and Boolean classifications, like "has credit approval."

It is often desirable to make calculated results available within the data model so that they can be mapped to messages or used as intermediate results for other computations or integrity constraints.

Challenges of Extending the SID Model

Given the need to elaborate the SID model, what is the best approach to extending it? We'll consider several tacks, all of which have inadequacies.

Modifying the SID Model

There is nothing to prevent us from simply changing the SID UML or XSD model to accommodate the particular needs of a business process. For instance, if we need a new attribute for a customer, we can add the attribute directly to the Customer class in the model. One drawback of this approach becomes evident as

soon as a new version of the SID model is released. How do we keep track of all the modifications we have made, apply the new version of the model, and then re-implement all the modifications in the new version?

Defining Subclasses

A more sophisticated way to extend the model is to define subclasses of existing classes. However, this approach has problems that can make subclass definition unwieldy.

Consider an example. The SID has an abstract class PartyRole that defines characteristics of the role played by any party (individual or organization) in an interaction. The SID has a more specialized class, Customer, which is a subclass of PartyRole.

Suppose we want to add some enterprise-specific attributes of Customer that are not defined in the SID Customer class. We can define a subclass of Customer, which we'll call CustomerExtension. Now suppose we need to define some enterprise-specific attributes for all party roles. We can define a subclass of PartyRole, which we'll call PartyRoleExtension.

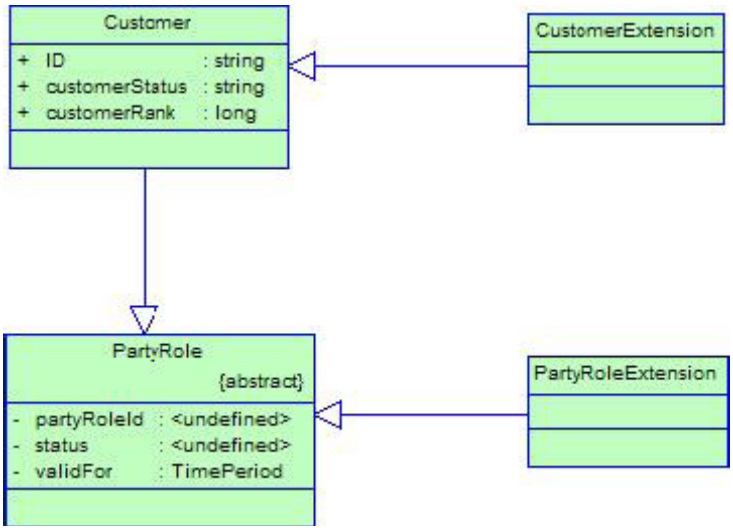

Figure 4. Extending the SID Model by Subclassing

But now we have a problem. Our CustomerExtension class inherits from the SID Customer class, which in turn inherits from PartyRole. But we want our CustomerExtension class to inherit the characteristics we defined in our PartyRoleExtension class. We might consider making CustomerExtension inherit from both Customer and PartyRoleExtension. But many programming languages that could implement the model, including Java, do not allow a class to have more than one superclass (multiple inheritance).

A second approach would be to make our CustomerExtension class inherit directly from our PartyRoleExtension class. But then CustomerExtension would no longer inherit the characteristics of the SID Customer class. We would need to duplicate those attributes in our CustomerExtension class. And when the next version of the SID appeared, we would have to be careful to duplicate any changes that the new version of the SID made to its Customer class.

DataXtend Semantic Integrator: Managing Adaptation and Change of the SID

DataXtend Semantic Integrator maintains the advantages of using the SID as a central data model while facilitating definition and management of extensions. The basic features of the DataXtend SI approach are as follows:

- ⇒ Rapid creation of a common data model by importing either UML or XSD representations
- ⇒ Rapid creation of data sources (based on relational databases or Web services) and data services by importing database schemas, XSDs, or Web Services Description Language (WSDL) documents
- ⇒ Definition of calculated values as computed attributes and of validity constraints as rules, both using a graphical expression builder accessible to both developers and business analysts
- ⇒ Definition and reuse of enumerations
- ⇒ Graphical mapping between the common model, data sources, and data services
- ⇒ Record keeping for all changes and additions made to imported models, coupled with the ability to re-import new versions of the underlying models while maintaining changes made in earlier versions
- ⇒ Design-time testing without deployment to a server

Using DataXtend SI, you can change and extend the SID model in natural ways, usually without writing any custom Java code:

- ⇒ Define new attributes for existing classes, or change the types of existing attributes
- ⇒ Define new classes or subclasses in the model
- ⇒ Define rules, enumerations, and mappings to and from other message formats

In DataXtend SI, this complex set of models, rules, and mappings that ensure data validity for an integration project is captured as metadata and is collectively known as the exchange model. In the DataXtend SI design environment, DataXtend SI Designer, your extensions and changes appear in the context of the SID model. However, DataXtend SI keeps track of the changes you have made. When you import a new version of the SID, DataXtend SI updates the underlying SID definitions without affecting the changes and extensions you have made. DataXtend SI thus maintains the integrity of the underlying SID model, and your changes are like a transparent overlay on top of that model. DataXtend SI also analyzes the impact of changes you are considering on other parts of the model. In this way, DataXtend SI combines the ability to extend the SID with ease of maintenance as the SID model itself changes.

Figure 5 shows the DataXtend SI design environment, DataXtend SI Designer, integrated with Eclipse.

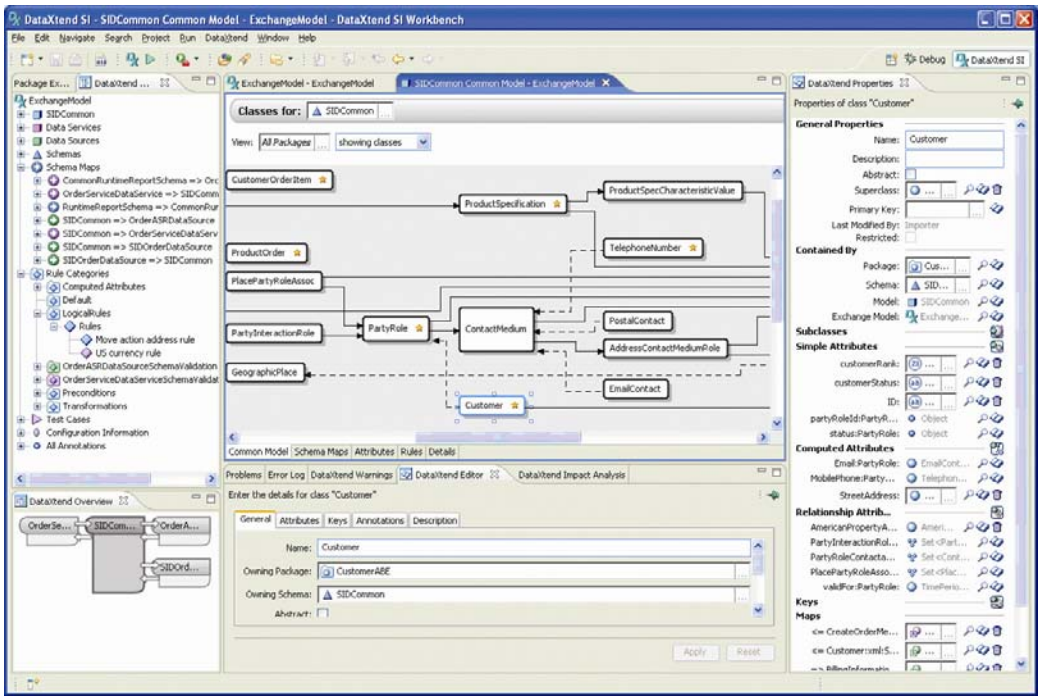

Figure 5. DataXtend SI Designer Integrated with Eclipse

Case 1: Home Telephone Number

Compare the derivation and mapping of a customer's home telephone number using Java code and using DataXtend SI. In Java (or another language), mapping home telephone number requires something like the following pseudocode:

1. getCustomer().getContactMediums()
2. iterate through all ContactMediums
3. for each ContactMedium that is instanceOf TelephoneNumber:
    - cast ContactMedium to TelephoneNumber
    - if the following the following conditions are met:
        - today() is between validFor.startDateTime and validFor.endDateTime
        - TelephoneNumber.type is HomeNumber
    - then map the TelephoneNumber.number to home_telephone_number

In DataXtend SI, we can implement home telephone number as a new computed attribute, currentHomeTelephoneNumber, of the Customer class. Using the DataXtend SI expression builder, we define an expression to compute home telephone number. The expression looks like this:

```
Select:   All Values
From:     PartyRoleContactableVia/asTelephoneNumber
Where:    (type = "Home" and
          validFor/startDateTime <= today() and
          validFor/endDateTime >= today())
Value Is: number
```

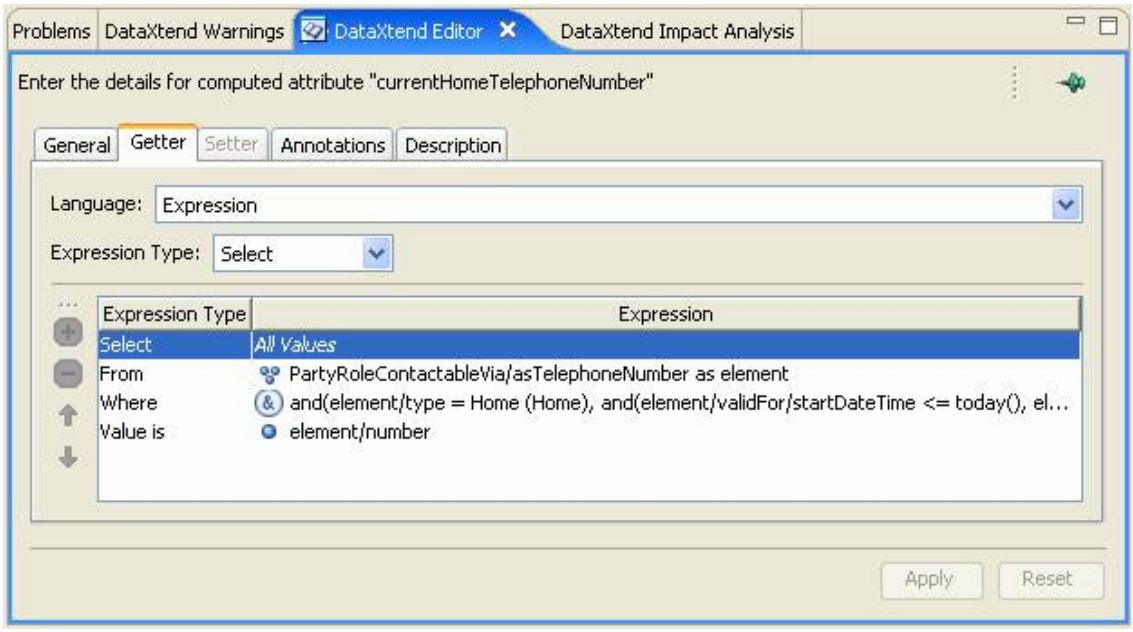

Figure 6. DataXtend SI Editor Showing Computed Attribute Expressions

This expression uses the Select-From-Where construct of the DataXtend SI expression builder, one of several that build expressions in an intuitive way. In this construct, the From clause specifies a path to a collection, and the Where clause specifies a filter for choosing elements of the collection. The expression builder syntax includes navigation of the model's relationship (association) paths and access to all subclasses of a class.

The DataXtend SI approach offers the following features:

⇒ The logic for computing home telephone number is part of the common model, where it is available for reuse.

⇒ The attribute is defined once and, like any other attribute, can be mapped to any number of data sources or data services.

⇒ The semantics of checking for valid start and end dates is included with the definition.

⇒ We could break the definition into finer components if we anticipated extending the definition to, say, business and mobile telephone numbers. For example, we could define a separate computed attribute on the TimePeriod class called "isCurrentTimePeriod" for checking validity dates and reuse that computed attribute in defining business and mobile telephone numbers.

⇒ The DataXtend SI expression builder does not require a Java programmer to define the expressions. DataXtend SI generates optimized Java code automatically from the expressions.

Case 2: VAT Registration Number

In a programming language like Java, mapping a VAT registration number to the SID model requires something like the following pseudocode:

1. define a subclass, VatIdentification, of the OrganizationIdentification class
2. add a VatNumber attribute to VatIdentification
3. getCustomer().getParty()
4. cast Party to Organization
5. getOrganizationIdentifications()
6. for each OrganizationIdentification that is instanceOf VatIdentification:
    - cast OrganizationIdentification to VatIdentification
    - if the following condition is met:
        - today() is between validFor.startDateTime and validFor.endDateTime
    - then map the VatIdentification.VatNumber to vat_number

Using DataXtend SI, we can implement the VAT number itself as a new attribute of the OrganizationIdentification class or, for greater generality, as a new attribute of its superclass, PartyIdentification. We call this new attribute identifierValue. We also define another attribute, identifierType, to identify the value as a VAT number. We can then add a new computed attribute to the Organization class, vatIdentificationNumber, which uses the following expressions to return the VAT number:

```
Select:   All Values
From:     OrganizationIdentifiedBy
Where:    (identifierType = "VatRegistration" and
          validFor/startDateTime <= today() and
          validFor/endDateTime >= today())
Value Is: identifierValue
```

Figure 7. Defining a New Computed Attribute in DataXtend SI Designer

As a further refinement, we can reduce errors by defining an extensible enumeration to hold values for identifierType.

The DataXtend SI approach has the same benefits in this case as for home telephone number. The VatIdentificationNumber attribute is defined in the common model, without using Java, and is available for mapping to and from other message formats.

Case 3: Customer ID

In Java, mapping a customer ID from a product order to the customer ID in the SID models requires something like the following pseudocode:

1. getProductOrder().getBusinessInteractionRoles()
2. iterate through all BusinessInteractionRoles
3. for each BusinessInteractionRole that is instanceOf PartyInteractionRole:
    - cast BusinessInteractionRole to PartyInteractionRole
    - getPartyRole()
    - if the following the following conditions are met:
        - today() is between validFor.startDateTime and validFor.endDateTime
        - PartyRole is instanceOf() Customer
    - then:
        - cast PartyRole to Customer
        - map the Customer.id to customer_id

Using DataXtend SI, we can define a computed attribute on the ProductOrder class to represent the customer identified by the customer ID in the order. The expression for this computed attribute, named OrderingCustomer, looks as follows:

```
Select:   All Values
From:     BusinessInteractionInvolves/asPartyInteractionRole/
          PartyInteractionRoleIdentifiedBy/asCustomer
Where:    validFor/startDateTime <= today() and
          validFor/endDateTime >= today())
Value Is: customer
```

Notable in the From clause are the two subclass designators, asPartyInteractionRole and asCustomer. DataXtend SI Designer makes all subclasses available for navigation and mapping. Figure 8 shows how the DataXtend SI expression builder displays paths through subclass relationships.

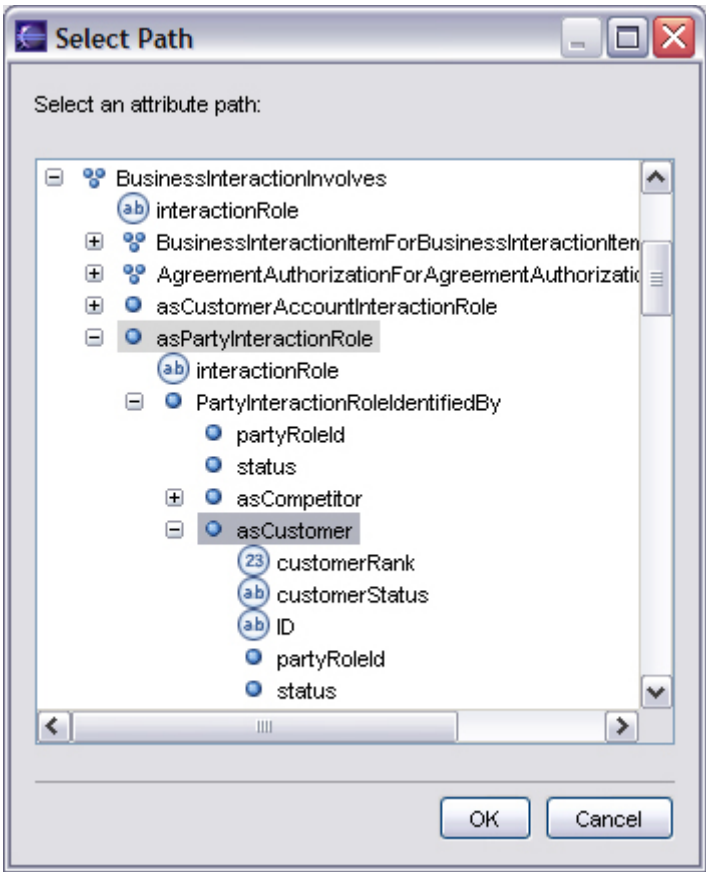

Figure 8. Navigating through Subclass Relationships

Defining a computed attribute on the ProductOrder class to represent the ordering customer makes it easy to map from the customer identifier in the order message directly to the customer ID in the SID model. Figure 9 illustrates DataXtend SI's graphical mapping interface showing the mapping for customer ID from the order message to the SID model via the OrderingCustomer computed attribute on the ProductOrder class.

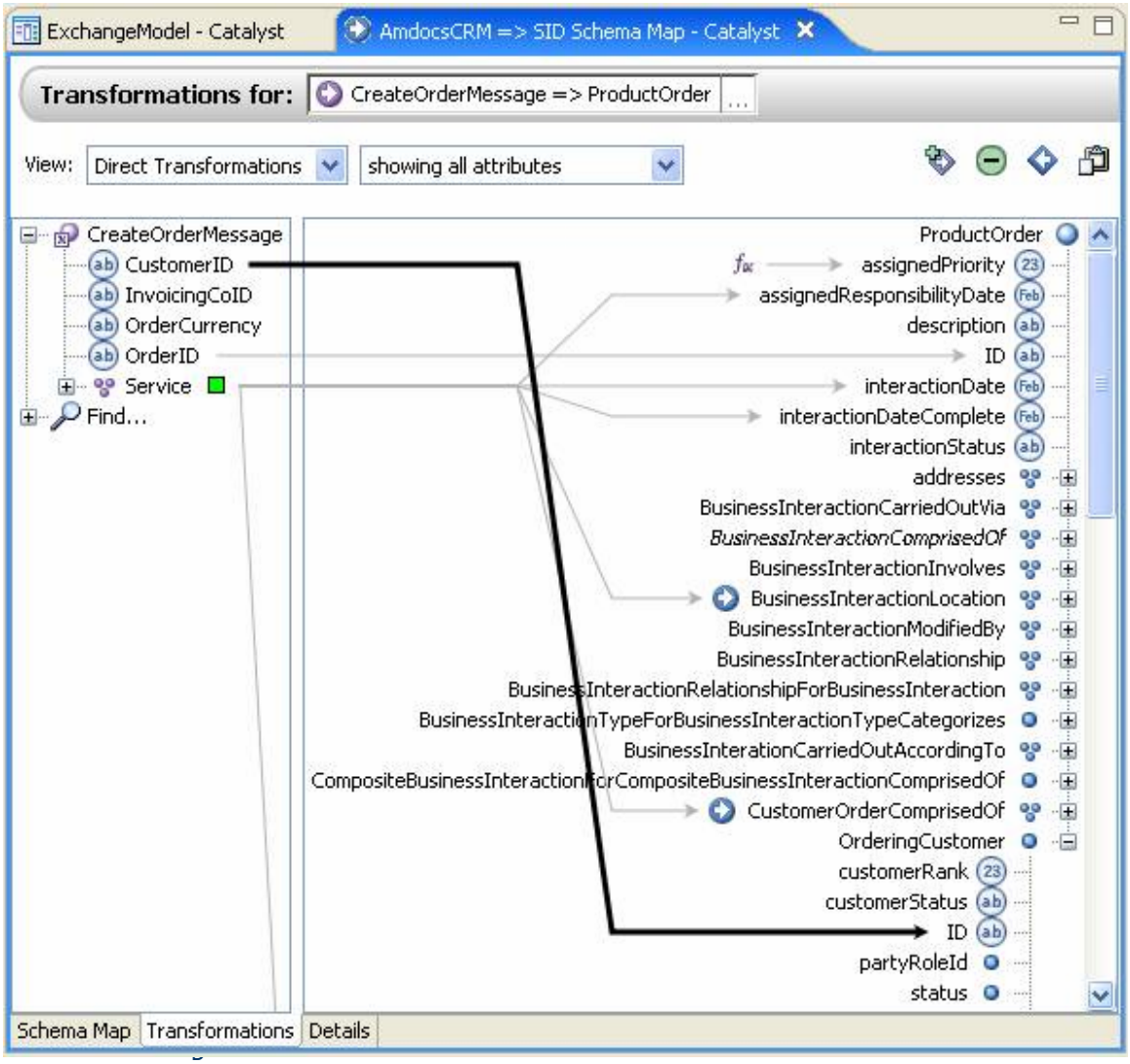

Figure 9. Mapping for Customer ID from Order Message to SID Model

DataXtend Semantic Integrator Benefits

The TMForum SID model promises to help telecommunications providers simplify OSS integrations by providing a common semantic model for mapping and transforming data. But the SID is not a panacea. In real-world applications, it must be modified and extended.

DataXtend Semantic Integrator provides the design and runtime environment that makes use and extension of the SID model manageable and cost effective. DataXtend SI offers the following benefits:

⇒ Implementation of the SID as a single reusable common model.

⇒ Rapid creation of common, data source, and data service models through the import of UML, XSD, and WSDL representations.

⇒ Definition of computed data and rules for validity constraints using graphical expressions, usually without Java code.

- ⇒ Graphical mapping of attributes between the common model and other message formats, with the ability to define custom source expressions where needed.
- ⇒ Integration of extensions into the SID-based common model.
- ⇒ Modular addition and change of integrated systems requiring only mapping to the common model, without changing all other systems in the integration.
- ⇒ Ability to incorporate new versions of the SID by re-importing the model while maintaining user-defined changes.
- ⇒ Tools to analyze the impact of modifications and better manage the costs of change over the integration lifecycle.
- ⇒ Capability to model errors and define recovery paths, providing more satisfying and reliable user experiences.

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