

The SOA Magazine

Feature Article



The Content Assembly Mechanism (CAM) and SOA Data Service Layers

by David RR Webber

Published: July 5, 2007 (SOA Magazine Issue IX: July/August 2007, Copyright © 2007)

[Download this article as a PDF document.](#)

Abstract: As use of SOA systems expands the effort needed for integration of content transaction exchanges will become increasingly an inhibiting factor comparable to how legacy EDI systems reached a level of saturation in their traditional deployment niche. The danger is that the very success of SOA-enabled interfaces can threaten to overwhelm an organization's ability to operate them, due to the fact that the cost of supporting and maintaining these interfaces can overshadow their potential return on investment.

The new OASIS Content Assembly Mechanism (CAM) templates standard provides significant technology advances that reduce and spread out data service integration burdens. Furthermore, by providing an open public standard the CAM approach enables industry domains and e-government initiatives to freely share formal business rule sets and transaction use patterns directly in machine processable formats.

For the business data analysts involved in SOA implementations the CAM documentation formats allow direct agreement between stakeholders on business rule details that are then directly applied in the SOA data services layer without requiring re-programming. Achieving a reduction in complexity and consistency in representation remains a continuing business challenge. In this article I explore the progress to date achieved with CAM and look at opportunities and scope for facilitating SOA data services and the various deployment models available.

Introduction

After almost four years of development that traces back to earlier UN/CEFACT efforts on message transactions and core component representation, processing, and handling (BRIM WG) [REF-1], version 1.1 of CAM [REF-2] was approved as an OASIS member standard on June 1st, 2007.

The current CAM release provides templates that allow for the definition of validation data services, specifically for XML-based transactions. While existing techniques using W3C XSD schema, namespaces, XSLT and other tools (such as Schematron and XMLBeans) have been in existence for some time, they provide more developer-centric and potentially brittle design options. The intention behind the CAM initiative is to supply a more agile, fault tolerant, and adaptive method in order to help establish more robust service interfaces. Particularly, CAM templates provide WYSIWYG-style XML structure mapping and context driven rule selections of components (and services) and their content.

Why is this so important to SOA data services? When applying a service-oriented design approach Web services tend to offer information content that matches a pre-defined (standardized) schema structure. Although this may be initially deemed sufficient, in practice the schema may only be fulfilling a broad and vague outline that is light on actual implementation specifics.

Additionally, the XML schema language is very programmer-centric. While developers may be comfortable with schema code, it is not necessarily helpful for agile information interfaces that need to respond to business context

criteria. For example, XML schema definitions cannot be effectively shared with business analysts and general staff members.

Within typical SOA implementations, consistent and reliable service behavior is preferred and often required by consumer programs. As a result, the perception of service quality and the level of effort needed to support and use services can be directly affected by service interface content. CAM is a technology designed to improve these aspects by allowing data service interfaces to be influenced and even shaped by business and data analysts. For example, Figure 1 illustrates how CAM templates can be shared across a business community of practice.

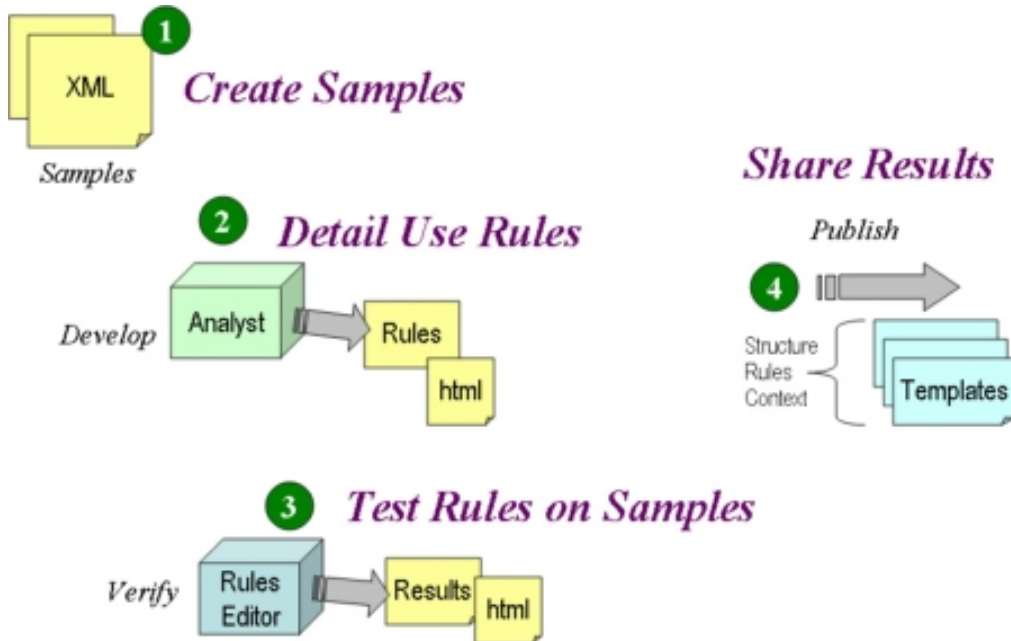


Figure 1: A typical CAM delivery process whereby analysts can contribute rules and other content prior to the templates being published.

An additional feature of CAM platform is that it supplies mechanisms that facilitate service versioning and reuse, an area of service governance that is typically the domain of the software support staff. These mechanisms are designed to overcome traditional issues that resulted from schema-based software being too susceptible to slight variations in transaction formats.

Overall, CAM can be seen as tackling the need to have open public mechanisms that can be shared across participants at both business and software development levels in an organization. This same rigor can also be applied internally between system exchanges for integrations between vendor products and data services.

An Example: Amazon.com Web services (AWS)

Many insights can be gained by studying the well-established AWS platform [REF-3] that has been available now for over three years. Amazon's Web services provide access to almost the same extent of pricing and product information that Amazon.com Web site visitors have.

AWS partners that interface with these services go through transaction iterations before establishing consistent results. They have to learn the nuances of how AWS response messages are constructed and packaged. It is not just a case of looking at their schemas; Amazon uses a self-service versioning approach where the version ID of the request is included in the syntax of the request itself.

Amazon is continually making changes to the content model, and as a result they support version requests going back several years. So here we have a case of flexibility but also the ensuing support burden. Figure 2 shows the overall layout of AWS transactions for product requests and associated context rules that control its use and response details.

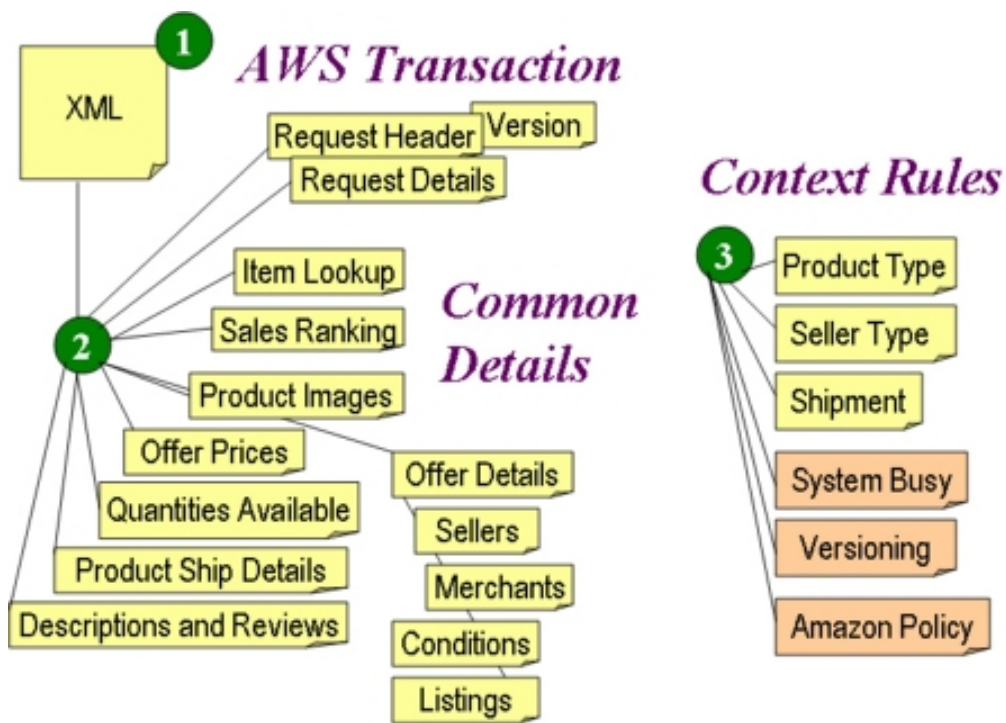


Figure 2: The overall component layout of an AWS transaction.

While Amazon uses certain schema structures, these support a wide range of products from books to DVDs to toys to electronics and so on. Each product type has different information models (broadly shown in the diagram). A query for a range of product codes will return a block of XML with varying parts. Some of these structure nuances are substantial with XML element tags appearing in different blocks of the structure and with different associated elements. Not only that but each item has information about prices, stock levels and downstream suppliers of new and used items in addition to Amazon.com itself and the ranking of those suppliers.

Amazon is of course selective with this information and prioritizes certain items higher in the list looking to create preferred buying patterns and the use of “lowest price” labeling. However the most interesting change that occurs to information served to requestors is the peak load-based optimization that happens on-the-fly throughout the day. As pressure builds on Amazon.com servers, they start easing the processing burden of services by reducing the content returned by queries.

As a result of this little study, we can quantify the following context drivers:

- Amazon’s own preferred sales and pricing techniques
- product type variances in information structure
- version control of new features and content
- hourly performance optimization
- relationships between information parts and content values

Clearly when you look at W3C schema and namespace mechanisms, none of these contextual behaviors are supported; all you see is the total layout of the overall structure that could occur.

Now that we have examined a typical interfacing example, we will consider how CAM can address these needs in a more systematic way.

Simplifying Complex Information Structures

One of the many abilities designed into CAM from the outset was context awareness, a result of the UN/CEFACT work on the core components technical specification (CTS) REF-4 that is founded on the notion of applying context to determine use. What this means is that you can change the behavior of CAM templates based on context drivers either within the content itself or externally as parameters declared and passed to the CAM processing.

When applying context to structures that have significantly varying parts, CAM provides support for different

techniques. The most obvious is to have more than one master structure template as the base reference for processing content. Next is the ability to include in either sections of structure or invoke a sub-template. This is then followed with the ability to prune parts of a structure contextually or to select from a choice of different structure elements, depending on the content itself.

The **default** structure rules from the CAM template for AWS are shown here:

```
<as:Rules>
<as:default>
<as:context>
  <as:constraint action="makeRepeatable(//asm1:Arguments/asm1:Argument)" />
  <as:constraint action="restrictValues(//asm1:Arguments/asm1:Argument/@Name,'MerchantId'| 'AssociateTag'|
'ItemId'| 'ContentType'| 'Service'| 'SubscriptionId'| 'ResponseGroup'| 'Operation'| 'IdType'| 'Version')" />
  <as:constraint action="makeRepeatable(//asm1:ItemLookupRequest/asm1:ItemId)" />
  <as:constraint action="makeRepeatable(//asm1:ItemLookupRequest/asm1:ResponseGroup)" />
  <as:constraint action="makeRepeatable(//asm1:Items/asm1:Item)" />
  <as:constraint action="makeRepeatable(//asm1:Offers/asm1:Offer)" />
  <as:constraint action="makeRepeatable(//asm1:Offers/asm1:Offer/asm1:Merchant)" />
  <as:constraint action="makeRepeatable(//asm1:EditorialReviews/asm1:EditorialReview)" />
  <as:constraint action="makeOptional(//asm1:OfferAttributes/asm1:ConditionNote)" />
  <as:constraint action="makeOptional(//asm1:ItemAttributes/asm1:UPC)" />
  <as:constraint action="makeOptional(//asm1:ItemAttributes/asm1:DeweyDecimalNumber)" />
  <as:constraint action="makeOptional(//asm1:Offer/asm1:Seller)" />
  <as:constraint action="makeOptional(//asm1:Offer/asm1:Merchant)" />
  <as:constraint action="makeOptional(//asm1:OfferListing/asm1:ExchangeId)" />
  <as:constraint action="makeOptional(//asm1:Item/asm1:EditorialReviews)" />
  <as:constraint action="makeOptional(//asm1:ItemAttributes/asm1:Creator)" />
  <as:constraint action="makeOptional(//asm1:ItemAttributes/asm1:NumberOfPages)" />
  <as:constraint action="makeOptional(//asm1:ItemAttributes/asm1:PublicationDate)" />
  <as:constraint action="makeOptional(//asm1:ItemAttributes/asm1:ISBN)" />
  <as:constraint action="makeOptional(//asm1:ItemAttributes/asm1:Author)" />
  <as:constraint action="makeOptional(//asm1:ItemAttributes/asm1:BatteriesIncluded)" />
  <as:constraint action="makeOptional(//asm1:ItemAttributes/asm1:ESRBAgeRating)" />
  <as:constraint action="makeOptional(//asm1:ItemAttributes/asm1:Feature)" />
  <as:constraint action="makeOptional(//asm1:ItemAttributes/asm1:Format)" />
  <as:constraint action="makeOptional(//asm1:ItemAttributes/asm1:Model)" />
  <as:constraint action="makeOptional(//asm1:ItemAttributes/asm1:Platform)" />
  <as:constraint action="makeOptional(//asm1:ItemAttributes/asm1:ReleaseDate)" />
  <as:constraint action="makeOptional(//asm1:OfferSummary/asm1:LowestUsedPrice)" />
  <as:constraint action="makeOptional(//asm1:OfferSummary/asm1:LowestCollectiblePrice)" />
  <as:constraint action="makeOptional(//asm1:Item/asm1:LargeImage)" />
  <as:constraint action="makeOptional(//asm1:ItemAttributes/asm1:NumberOfItems)" />
</as:context>
</as:default>
</as:Rules>
```

It is evident that all of these aspects can be applied to Amazon.com AWS XML responses and structures. Furthermore, specific rules can then be added that relate to each type of product (such as DVDs, books, toys, clothing, electronics, and so on) that trigger different sets of rules related to each product type. What this then does is provide a roadmap that clarifies the use patterns without requiring trial-and-error style learning curves for partners integrating with your systems. Not only does it eliminate the guesswork on initial testing and setup, it also greatly simplifies the process of version changes (as discussed in the upcoming section).

Continuing with the notion of structure simplification, very often in structures there are repeating patterns with slight nuances in their use patterns depending on where they occur in the structure. W3C schemas that lack the ability to use context driven mechanisms implementers must resort to manually repeating the same piece of structure code throughout the schema and then making manual inline changes to each section. This process can severely restrict the agility and flexibility of a service interface and can furthermore prevent the inclusion of reusable structure components. A particular example of this can be seen with the Postsecondary Electronic Standards Council (PESC) schemas [REF-5] for student loan applications for which CAM templates have recently been developed.

Figure 3 shows how the conceptual view of a CAM template structure contrasts with the complexity of raw schemas. By supporting such documentation views of the transaction interchanges, CAM templates become accessible to

business data analysts. This is especially important for these schemas because they are used by services across higher education institutions that need to exchange student application information. The information itself has complex nuances associated with differences between overseas and local students and related US-specific information practices and representations.

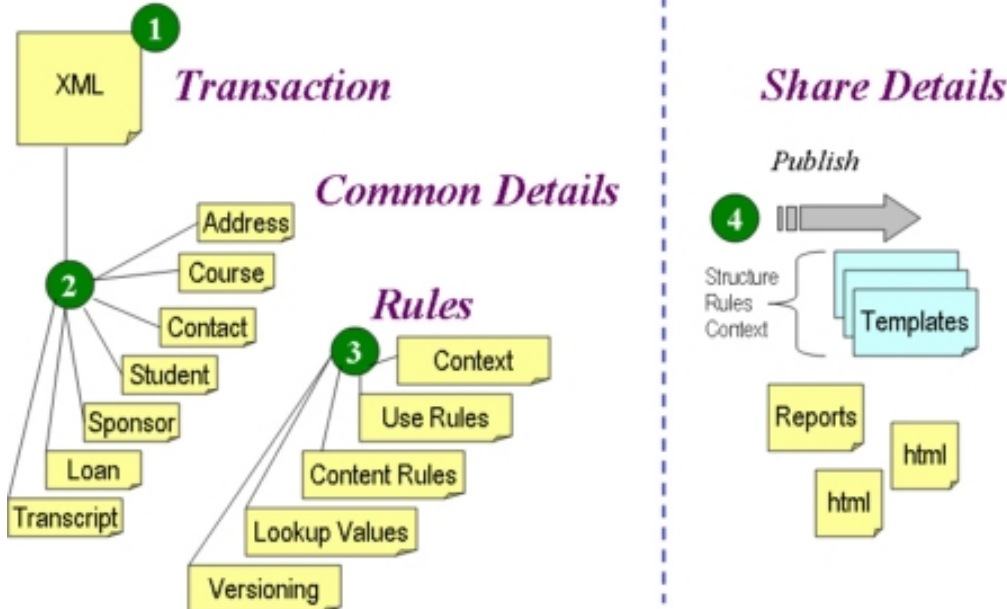


Figure 3: The conceptual components of the PESC Student Loan schema.

Address, Sponsor, and Contact constructs occur repeatedly within the schema structures. When looking at the raw schema, however, this pattern is completely obscured due to the manual editing used to create the various sections and the restrictions that prevent defining reusable (included) constructs that can be selectively tailored using contextual rules. The fragment shown here of the CAM template for the student transcript shows a simpler approach for incorporating included constructs.

```
<TransmissionData>
  <DocumentID>%DocumentID0%</DocumentID>
  <CreatedDateTime>%2006-05-04T18:13:51.0Z%</CreatedDateTime>
  <DocumentTypeCode>%Request%</DocumentTypeCode>
  <TransmissionType>%Original%</TransmissionType>
  <Source>
    <as:include> pesc-organization.xml</as:include>
    <NoteMessage>%NoteMessage%</NoteMessage>
  </Source>
  <Destination>
    <as:include> pesc-organization.xml</as:include>
    <NoteMessage>%NoteMessage%</NoteMessage>
  </Destination>
  <DocumentProcessCode>%TEST%</DocumentProcessCode>
  <DocumentOfficialCode>%Official%</DocumentOfficialCode>
  <DocumentCompleteCode>%Partial%</DocumentCompleteCode>
  <RequestTrackingID>%RequestTrackingID0%</RequestTrackingID>
  <NoteMessage>%NoteMessage14%</NoteMessage>
</TransmissionData>
```

These techniques in CAM are designed to allow multiple structure definitions and selective structure models based on the context of use patterns. So in the example shown above the Source and Destination structures can be modified using CAM structure selection rules to suit particular differences in the use models. Note that each individual organization definition is itself 52 lines long and contains over ten structure choices.

Versioning and Change Control

Versioning brings its own challenges and needs apart from business contextual parameters management. These

challenges can be summarized as:

- If the schema version changes how can we ensure it does not break in-place validations?
- How can rule changes be rapidly adapted in a production environment?
- How can we develop user context-driven version control and support the reuse of sub-components?
- How can we enhance and automate test release cycles by improving transparency for bug fix processes and expose the change deltas to speed testing process? (And what about support for regression testing?)

CAM has discreet sections in its template layout for structure, rules, and validation extensions. This transparency is particularly important when exposing change deltas.

Within W3C schemas these three aspects are mixed in together syntactically. What this means is that using software tools such a text DIFF utility on schema versions provides mixed results that may not easily identify changes. DIFF on a CAM template, on the other hand, will display each aspect separately and therefore make action determination simple. That action determination is based on the impact of those changes and whether they are critical, benign, or a new feature to be considered.

Furthermore CAM templates use XPath expressions to mark the linkage between rules and the target items in the structure. These are extremely flexible because they can label sub-element paths without requiring full path descriptions. That allows quite dramatic structure changes while having no impact on the rules being applied because the linkages continue to match.

One especially powerful capability of CAM templates is the ability to support external codelists – sets of allowed values relating to a particular code value in a transaction. Examples include valid US State codes, a catalogue product types, currency codes, country codes, weights and measures, and so on. OASIS has developed a neutral semantic representation called Genericode for handling these types of code lists. Genericode itself is an abstract format [REF-6]. To use it in tools like CAM requires extracting out the code values and storing them in an optimized layout for high speed lookups. CAM provides built-in support for these conversions and representations of corresponding lookup tables.

CAM as part of an SOA Implementation

A sample data services layer deployment within an SOA using a CAM processor is shown in Figure 4. This environment uses such as the jCAM Java open source implementation of CAM [REF-7] and provides three distinct deployment modes for typical interactions associated with an SOA, including Web services, B2B, and standalone local modes.

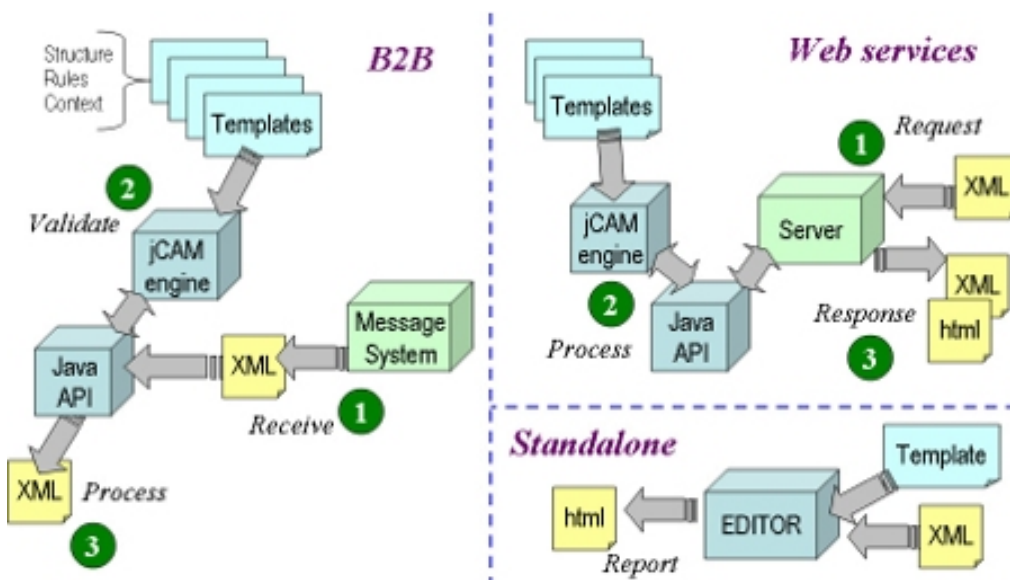


Figure 4: Examples of CAM deployment and use modes.

Typical vendor implementers will have existing data services embedded into their own proprietary transformation methods associated with their transport and delivery methods. The CAM processor can be included into these as

either a standalone component or via a call-out service. The advantage of using the CAM open standard templates is that rules and structure definitions can be freely shared across all partners systems, whereas vendor transformation tools require installing proprietary software tools. Similarly internal transformation rules often contain private rule checks and links to backend database repositories for lookups and evaluations that cannot be shared publicly. This dilemma is solved by separating out industry content and structure checks into the CAM template.

Furthermore, implementers can use open source tools to assist in the development and integration of their partners systems. The jCAM implementation facilitates this with support for standard Java APIs and for the use of XML DOM pointers to exchange transaction content and templates without incurring processing overheads. The jCAM open source implementation also contains an Eclipse based editor tool, as shown in Figure 5.

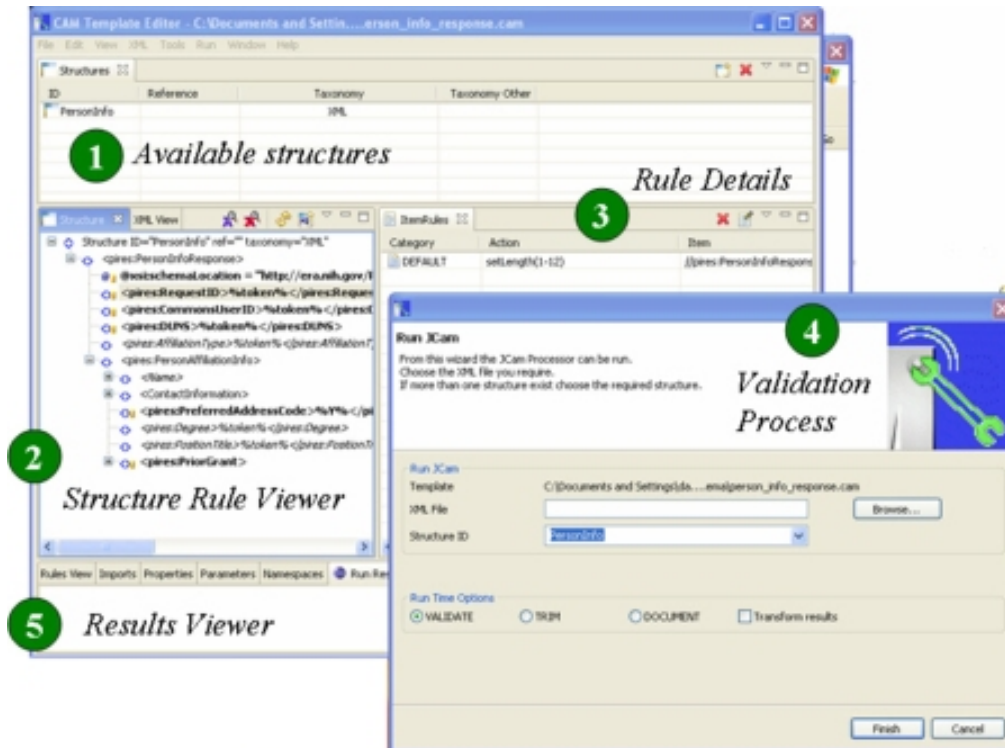


Figure 5: The jCAM Eclipse editor tool in action.

The Eclipse editor enables local standalone use of CAM templates in support of a SOA implementation. Much frustration and time can be saved prior to actual live SOA transaction exchanges via the particular transport delivery methods provided by an SOA system. These pre-testing evaluations on content allow both business analysts and software developers to fine-tune interactions much more quickly without having to wait for message turnarounds.

Also reduced is the need for inquiries to partner support staff when confusing or conflicting results are returned from the use of subtly invalid transaction structures. More often than not these invalid transactions simply “disappear” into the partners systems with little or no feedback required to the service requestor. Instead, the support staff is required to spend time locating and diagnosing transactions problems internally. This reveals that providing CAM templates that can pre-test for an extensive set of pre-requisite conditions facilitates the rapid integration of SOA from external partners.

The same applies to the SOA transport envelopes themselves. For example, a service-oriented solution may be relying on messages that use SOAP headers with SAML profiles. CAM templates can be built to allow partners to pre-test their transport configurations for those profiles and their header content. Similarly, for SOAP and ebXML envelope structures (where role, action, and transaction details along with extensions like CPA ID values) formatting can be pre-checked using CAM templates.

A further advantage of CAM templates approach is the extensive use of includes and lookup templates. This facilitates both re-use and agility in a SOA environment. Partner systems can receive rapid updates and changes via libraries of common lookups and includes that are then applied to their local systems.

Conclusion

The OASIS CAM technology standard provides significant new facilities for implementers of SOA data service layers that have not been previously available in a publicly distributable package. Of particular importance is the ability to involve business analysts in the process of constructing transaction interchanges and documenting and implementing business rule sets rather than relying purely on software developers. This extends over to shorter testing and integration cycles and more rapid adoption.

The jCAM open source implementation of the CAM specification provides both the runtime deployment engine and an interactive visual Eclipse editor for developing and testing templates. The Eclipse editor enables industry domain teams to jointly develop transactions and rules and share documentation and templates in a consistent way. The jCAM editor produces rule and structure reports that mirror the traditional spreadsheets and tables commonly produced by business analysts.

The commercial success of SOA is determined by the business facilitation and value it provides, offset by the level of effort needed to attain and support its implementation. Hence OASIS CAM templates provide industry standard technology that can address critical service facilitation requirements with measurable improvements in time to build, test, deploy, and support service-oriented solutions.

References

[REF-1] UN/CEFACT BRIM (Business Reference Information Model), http://www.unece.org/cefact/tmg/openplenary_sep02.ppt

[REF-2] OASIS CAM specification, <http://docs.oasis-open.org/cam>

[REF-3] Amazon.com Web services (AWS), <http://aws.amazon.com>

[REF-4] UN/CEFACT CCTS, Core Components Technical Specifications, <http://xml.coverpages.org/ni2007-04-20-a.html>

[REF-5] PESC conference presentation on scalable use of CAM, <http://www.pesc.org/events/techstandards/fourth/presentations/Scalable%20PESC%20Transaction-Handling%20Using%20OASIS%20CAM.ppt>

[REF-6] Genericode codelists examples, <http://docs.oasis-open.org/ubl/os-UBL-2.0/cl/gc/>

[REF-7] The jCAM open source implementation, <http://www.jcam.org.uk>

THE PRENTICE HALL SERVICE-ORIENTED COMPUTING SERIES FROM THOMAS ERL

