Technical Brief

August 2015

SEARCH
The National Consortium for Justice Information and Statistics

A Unified Modeling Language Approach to Modeling NIEM Exchanges: Analyzing Requirements

By Diane Lacy
Information Sharing Specialist
SEARCH

Introduction

This *Technical Brief* is the second of two briefs intended to illustrate how modelers can use the Uniform Modeling Language[1] (UML) to develop Information Exchange Package Documents (IEPDs) and data exchanges that conform to both the National Information Exchange Model (NIEM)[2] and the Global Reference Architecture (GRA).[3] The first *Technical Brief*, "A Unified Modeling Language Approach to Modeling NIEM Exchanges: Scenario Planning," provides an overview of UML 2.0 and the use of UML in the **Scenario Planning** phase of the IEPD development life cycle (Figure 1). This brief discusses using UML to complete the next phase of the IEPD development life cycle —**Analyze Requirements**—and also briefly discusses the **Map and Model** phase. The intent of these two briefs is to present a "best practice" process to create a UML model that fully conveys the business and technical information needed to define, develop, and deploy a NIEM-compliant information exchange. This brief will discuss the UML object, class, and component structure diagrams, as well as using UML diagrams as input to map to the NIEM model.

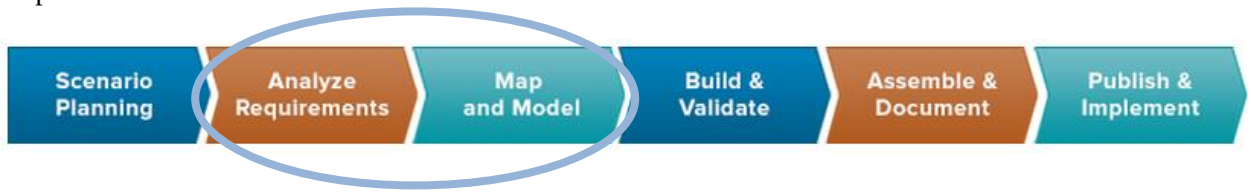

Figure 1: IEPD Development Life Cycle

[1] http://www.uml.org

[2] http://reference.niem.gov/

[3] https://it.ojp.gov/initiatives/gra. Both documents assume the reader is familiar with NIEM, the IEPD life cycle, and the GRA.

UML Overview

UML is a general-purpose modeling language in the field of software engineering, which is designed to provide a standard way to visualize the design of a system. UML uses industry standard modeling notations with a comprehensive set of diagrams and elements. It is widely used because it increases clarity and understanding of the components by visually documenting requirements, information, and processes. UML allows a modeler to represent the relationships and dependencies and establishes traceability back to the foundational business requirements. A full UML model transforms business requirements into technical specifications. The companion *Technical Brief* on scenario planning provides a more thorough discussion of which UML models and components are used in the IEPD life cycle.

The following business process model (Figure 2) depicts the overall sequence of UML modeling and IEPD development and identifies the appropriate UML components for each phase.

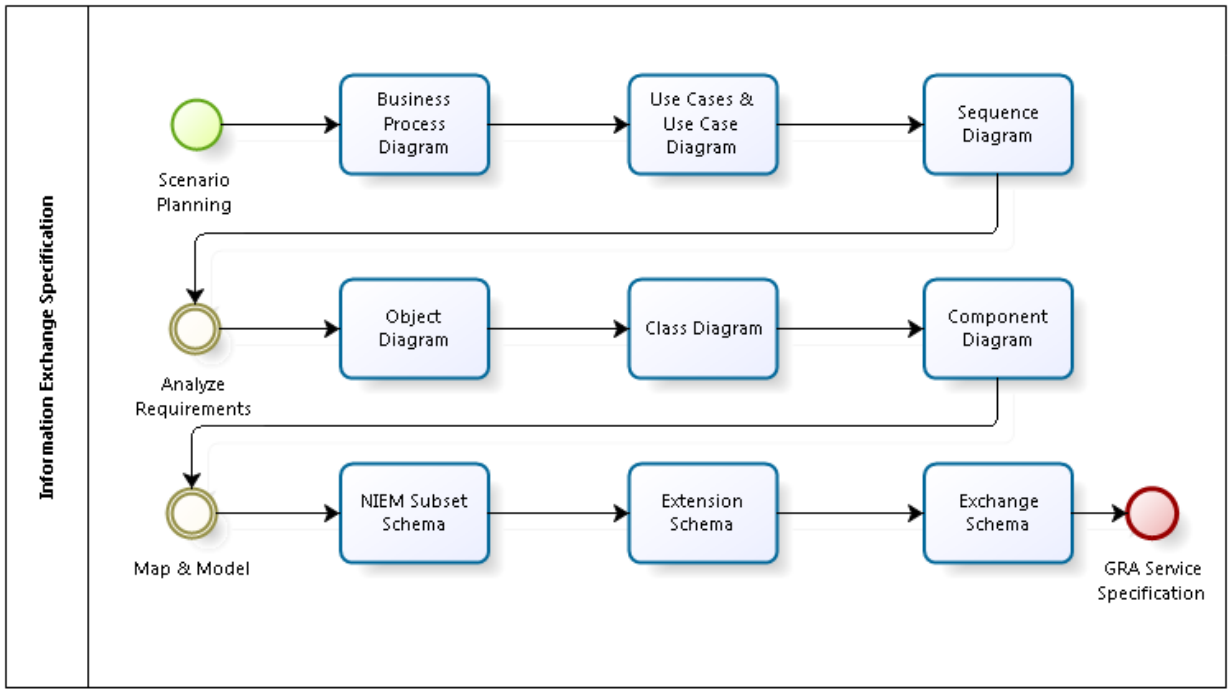

Figure 2: Business Process Model – Sequence of UML Modeling and IEPD Development

The first *Technical Brief* describes how modelers can use the business process diagram, use cases and use case diagrams, and sequence diagrams to complete **Scenario Planning**, the first phase of the IEPD development life cycle. This *Technical Brief* describes how modelers can use UML object diagrams, class diagrams, and component diagrams to complete the second phase of the life cycle, **Analyze Requirements**.

Both briefs illustrate a typical information exchange scenario in the justice process: Initiating a court case. The business process model shown in Figure 3 provides an overview of this exchange process.

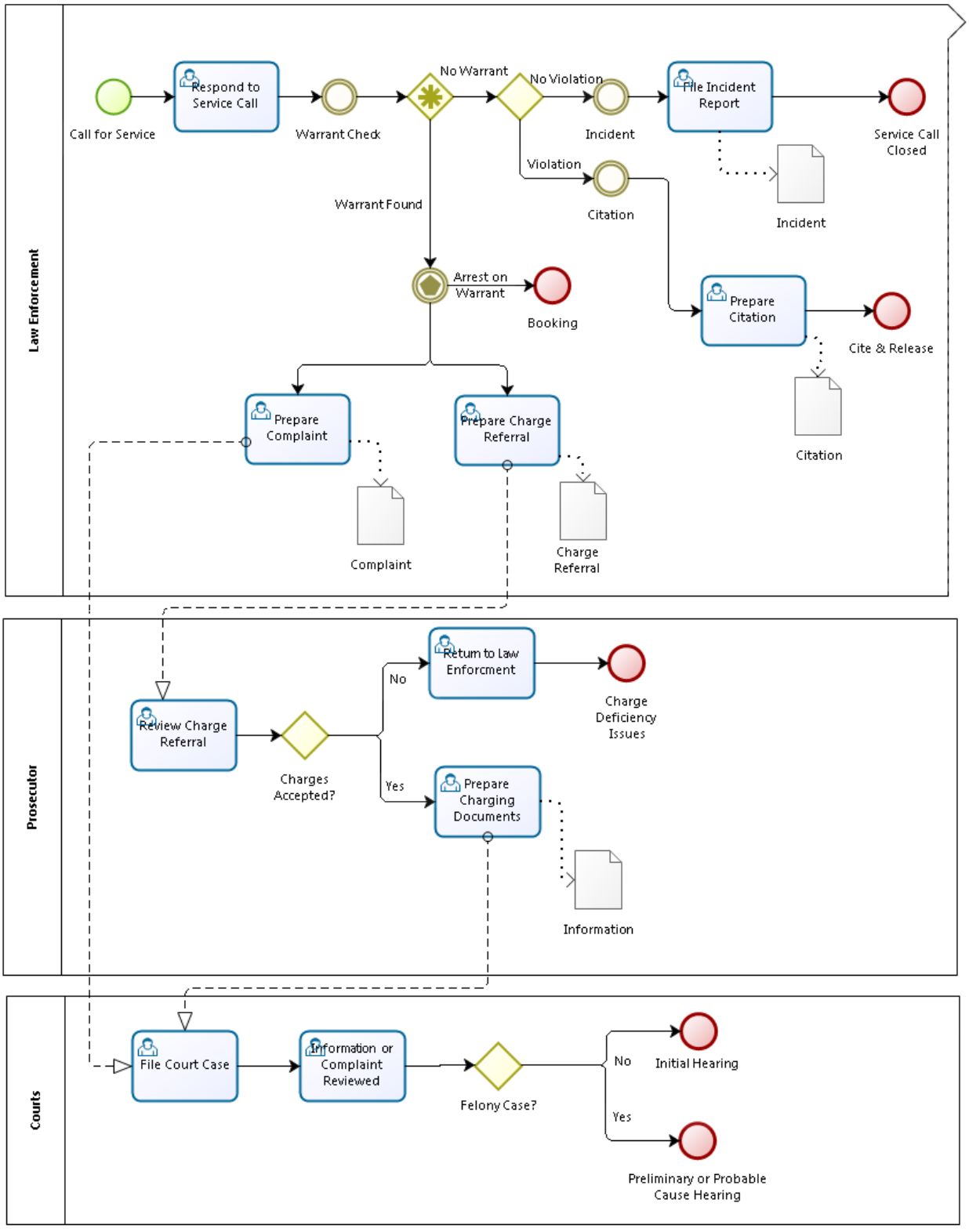

Figure 3: Business Process Model – Initiating a Court Case

Analyze Requirements

Analyze Requirements is arguably the most important phase because it creates the information model and documents the details for the subsequent **Map and Model** phase. The modeler needs to understand the current and "to-be" business state of information, data issues, and information content to be exchanged. The modeler does not need to formally document the current state but does need to collate reference materials for input to the "to-be" information model. Sources for this information can be subject matter experts, paper documents, system screenshots, existing schemas, and file structure layouts of existing interfaces. Second, the modeler needs to understand the sending and receiving data models of the involved systems to ensure semantic consistency in business definitions.

> Traditional relational data design practices tend to *normalize* (decompose) information into the simplest form and then revisit the model and *de-normalize* (recompose). UML design, as presented here, is the opposite: the large high-level objects are defined first in a business perspective. This helps the modeler determine how much data decomposition is appropriate for the message and audience.

Three UML diagrams are used in this phase: Object, class, and component diagrams.

- The **object diagram** is the vehicle for the modeler to interpret the message requirements and facilitates discussion and refinement with business analysts and subject matter experts.
- The **class diagram** is the vehicle for the modeler to communicate the detailed data requirements to the developers.
- The **component diagram** begins the discussion on the service deployment environment.

Object Diagram

An object diagram may be considered a special case of a class diagram. Object diagrams use a subset of the elements of a class diagram in order to emphasize the relationship between instances of classes at some point in time. They are useful in understanding the more detailed class diagrams, although they architecturally do not show anything different than class diagrams, but reflect multiplicity and roles. They are a good start for modeling the high-level complex classes before delving into the nested classes and detailed attributes. The difference between an object element and a class element is that object elements do not have compartments, whereas a class element contains attribute and operation compartments.

The object diagram in Figure 4 depicts the high-level objects in the "InitialChargingComplaint" message. (Note: This is for illustrative purposes only and is not intended to show a complete information model.) This diagram is sometimes referred to as an "association diagram" because it identifies associations and multiplicities (cardinality) between the objects from a centric root object, "Case" in this example. The diagram also illustrates generalizations, e.g. "Person" and "Organization," whose characteristics are inherited (shared) by their respective objects. The modeler decides the root, but in general for NIEM exchanges, it is the document or report.

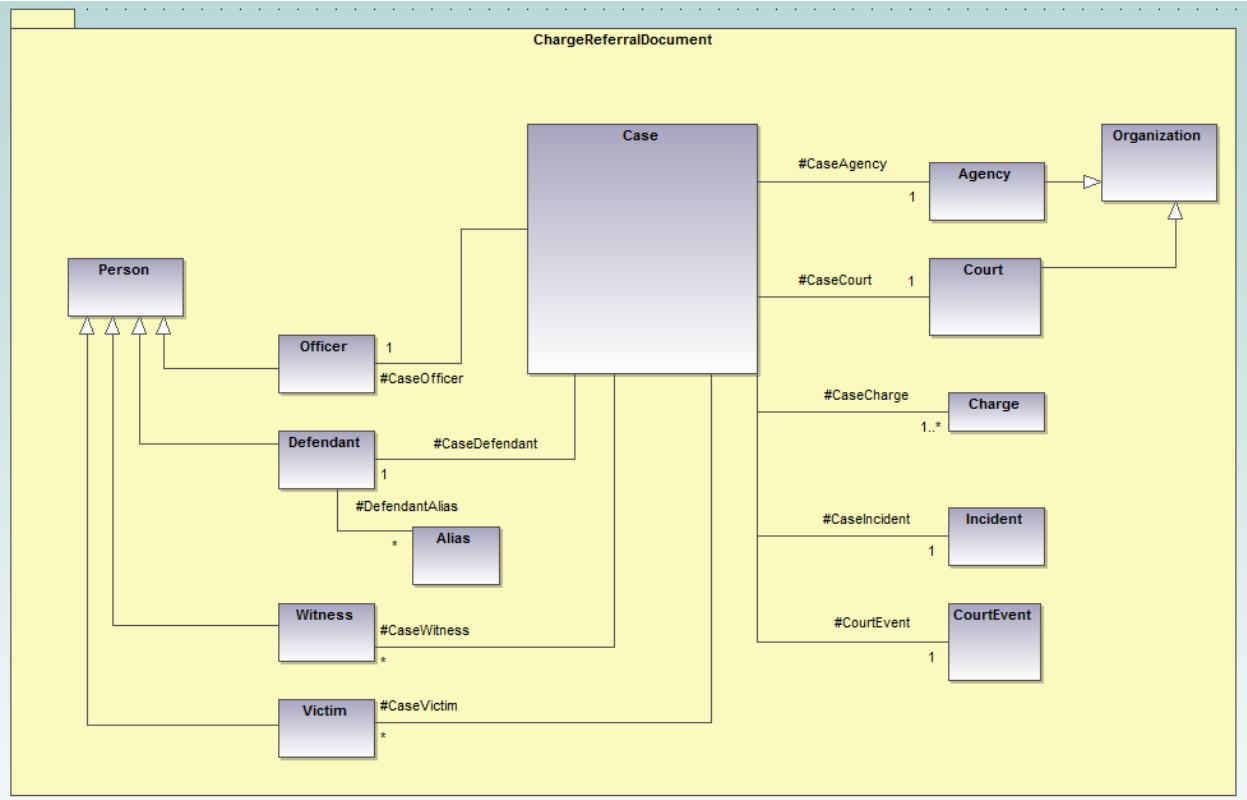

Figure 4: Object Diagram Depicting the "InitialChargingComplaint" Message

Class Diagrams

The object diagram sets the framework for decomposing the information model into the class diagram. Class diagrams show all the elements in the object diagram plus the class properties, property attributes, property cardinality, and association. The class diagram (Figure 5) begins to expand upon the previous object diagram. One advantage of using UML is that elements are unique within the model and reusable in multiple diagrams. For instance, if attributes of the "Case" class are changed in the object diagram, the change is reflected in the class diagram and vice versa.

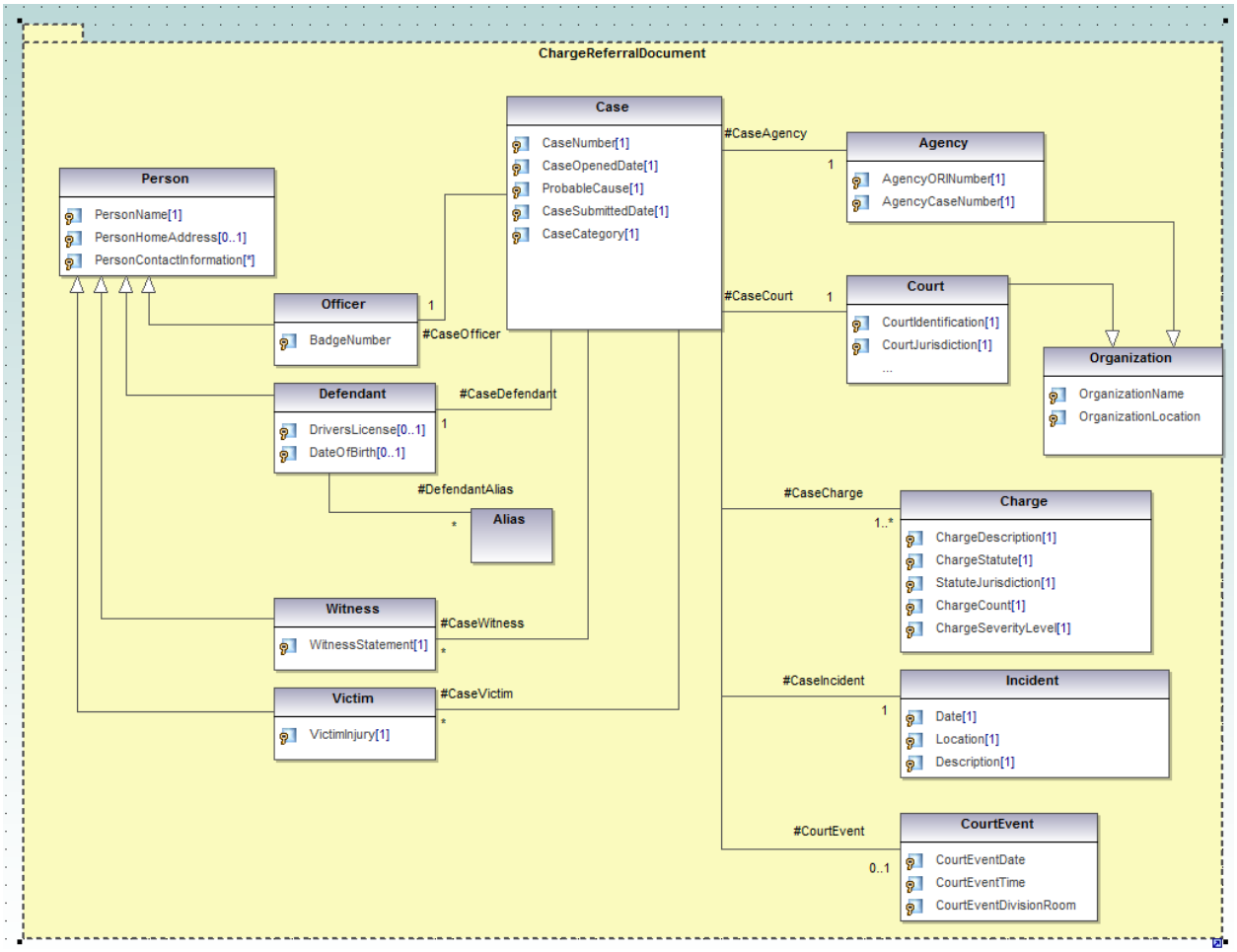

Figure 5: Class Diagram Expanding Upon "InitialChargingComplaint" Message

Information models can quickly become very complex; therefore, modelers can use packages to further organize and more easily visualize complexity. The "Person" package (Figure 6) decomposes the "Person" class in the "ChargeReferralDocument" and a "data association" links the Person class to the package (Figure 7).

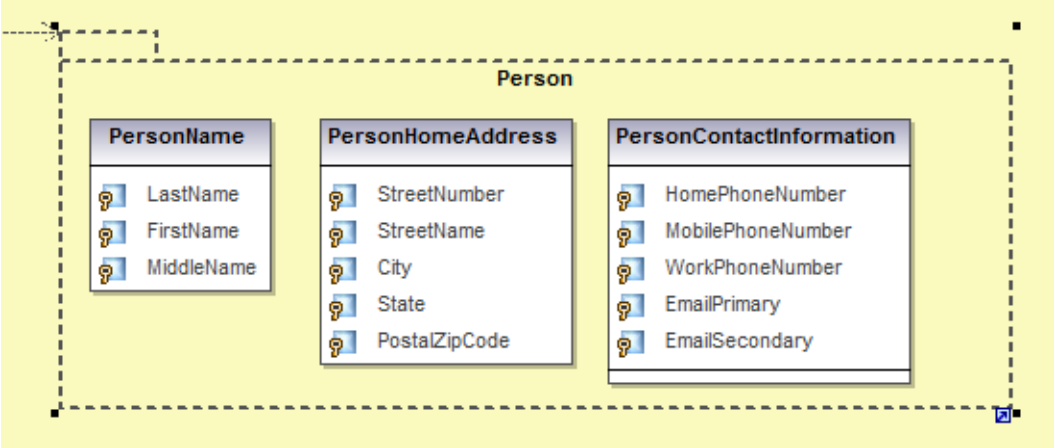

Figure 6: Person Package Decomposing Person Class in the "ChargeReferralDocument"

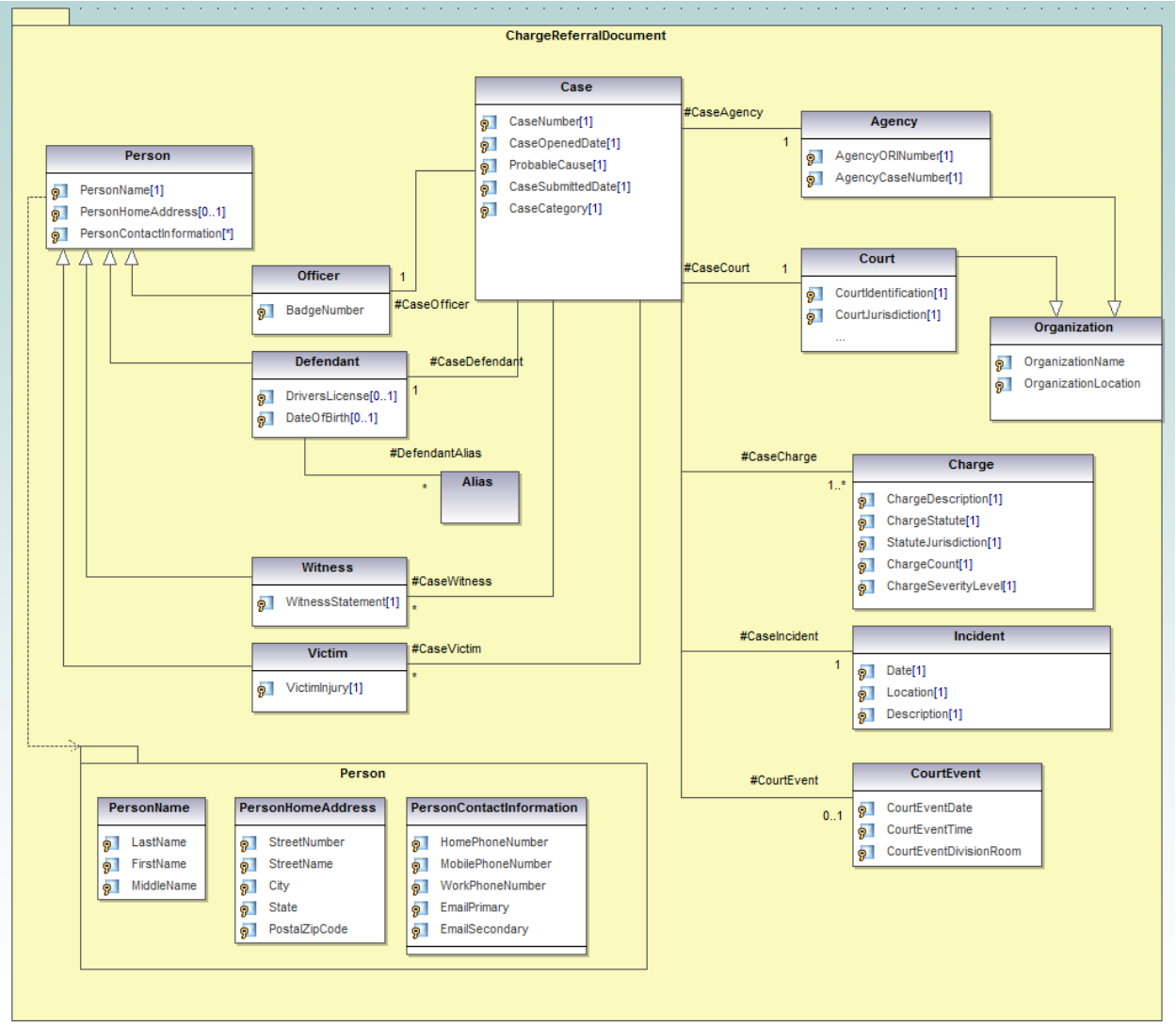

Figure 7: Data Association Links Person Class to the Package

Within the NIEM IEPD development cycle, the "mapping spreadsheet" artifact and a class diagram are closely coupled and modelers can develop these concurrently and iteratively. The mapping spreadsheet associates a business data element with a business definition and is used as input into the object diagram. It may be argued that a spreadsheet and a UML class diagram are redundant. However, they serve two different audiences and are complementary. The spreadsheet is used between the modeler and the business analysts to establish information exchange requirements, while the class diagram transitions the requirements to a schema specification for development staff.

Component and Deployment Diagrams

Class and object diagrams contribute to the design of an exchange, whereas component and deployment diagrams define the design and deployment of the exchange environment. They document the framework and realization of putting the exchange into use. They are valuable to server, network, and database administrators, as well as help desk support staff, and can be essential to troubleshooting.

Component diagrams illustrate the pieces of software, embedded controllers, networks components, etc., that will support the service exchange. A component diagram can be helpful to provide environment context for a service within their home environment. Similar in practice to package diagrams, component diagrams define boundaries and are used to group elements into logical structures. Specifically, depicting ports allows a service or behavior to be specified to its environment, as well as a service or behavior that a component requires. Ports may specify inputs and outputs, as they can operate bi-directionally.

A deployment diagram models the run-time cross-domain architecture of an exchange. It shows the configuration of the hardware elements (nodes) and shows how software elements and artifacts are mapped onto those nodes.

Component and deployment diagrams use many of the same UML elements and the modeler may decide that one or the other is adequate to document the environment. Note that both diagrams use the package element, whereas only the component diagram uses the class element, allowing finer-grained depiction of a component operation on a class. A good practice would be to develop a collection of component and/or deployment diagrams for each exchange partner. Table 1 lists the UML elements for each diagram type:

UML Element	Component Diagram	Deployment Diagram
Package	X	X
Interface	X	
Class	X	
Component	X	X
Artifact	X	X
Port	X	
Realizations	X	
Generalizations	X	X
Usage	X	
Dependencies	X	X
Node		X
Device		X
Execution Environment		X
Manifestation		X
Deployment		X
Association		X

Table 1: UML Elements for Each Diagram Type

Figures 8 and 9 depict component and deployment diagrams.

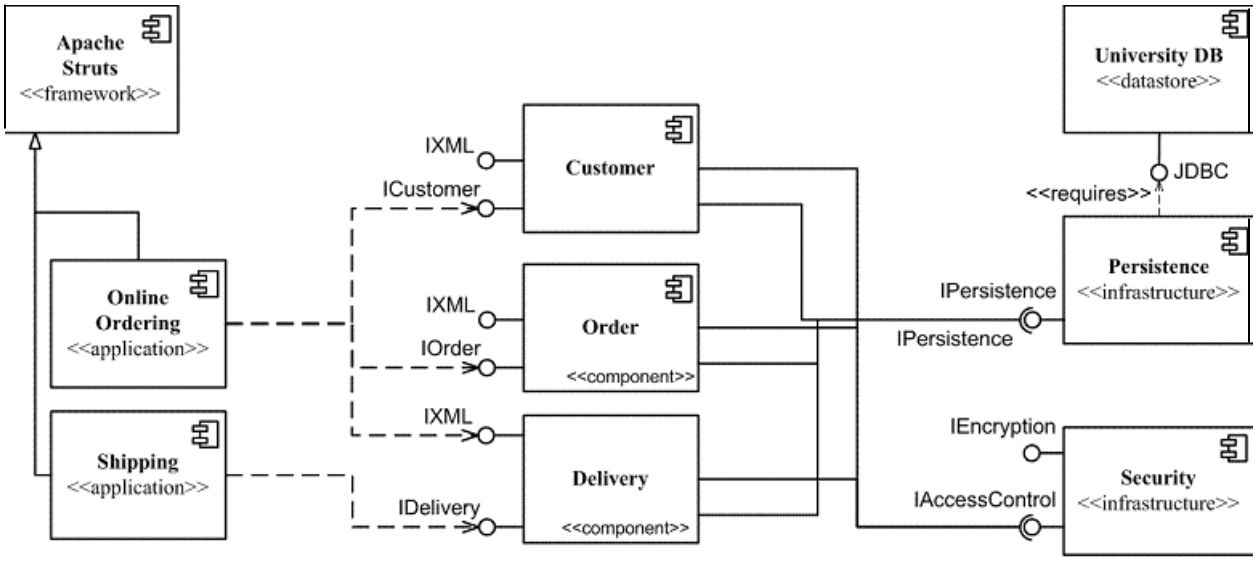

Figure 8: Component Diagram

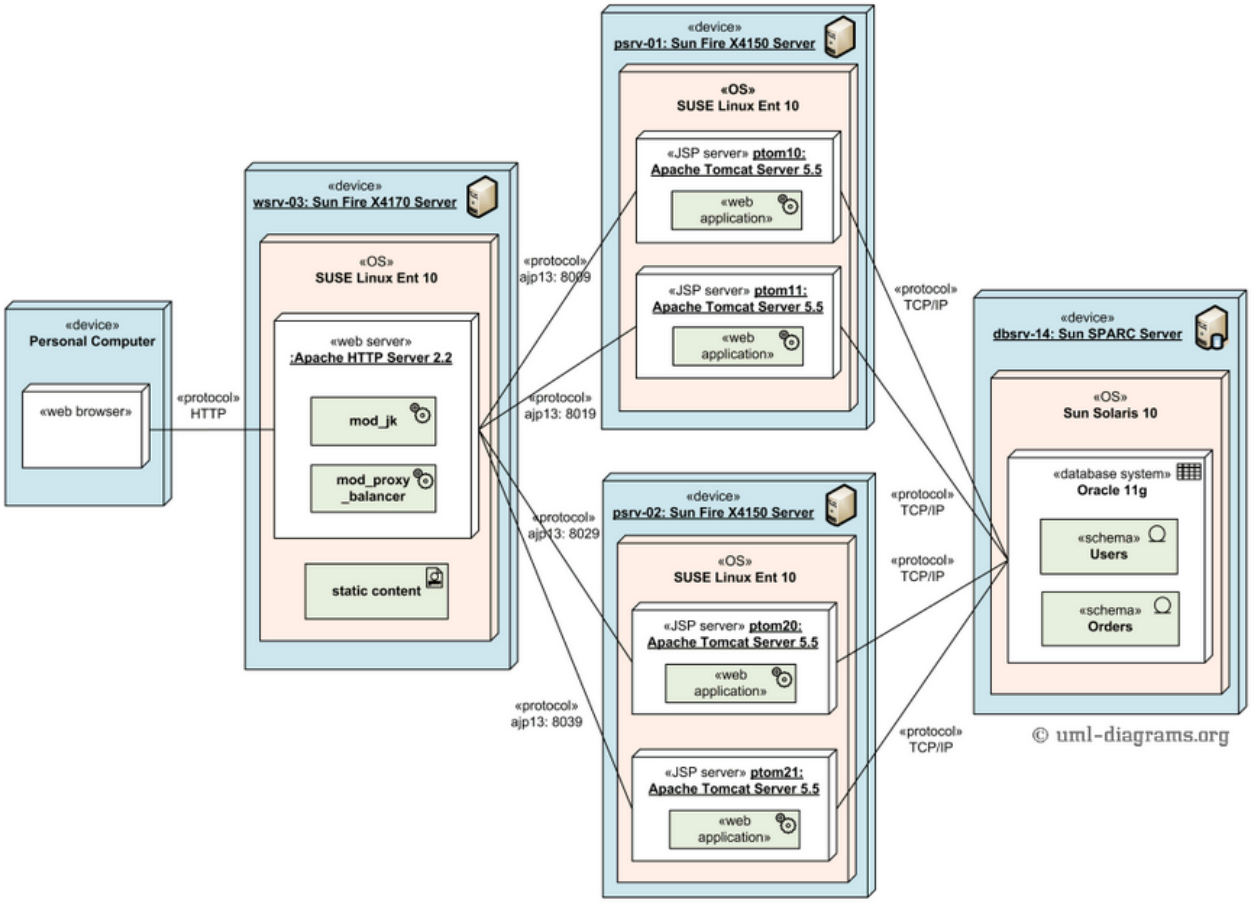

Figure 9: Deployment Diagram

Map and Model

Using the object and class diagrams, the modeler can begin mapping the information model to the NIEM model, navigating the NIEM domain models to map the business elements to NIEM elements. The most generally used tool is the NIEM Schema Subset Generation Tool.[4] It is best to start mapping to NIEM core elements (nc:) and then map to domain extension models (for example, j:) as needed.

> NIEM provides the modeler with many similar choices to model a business element—there are no right or wrong choices. It is up to modelers to map to elements that best fit their business perspective. The following examples are illustrative, not normative.

Using case and charge classes as examples to map to the NIEM model produces the following UML elements (Figure 10):

[4] https://tools.niem.gov/niemtools/ssgt/index.iepd

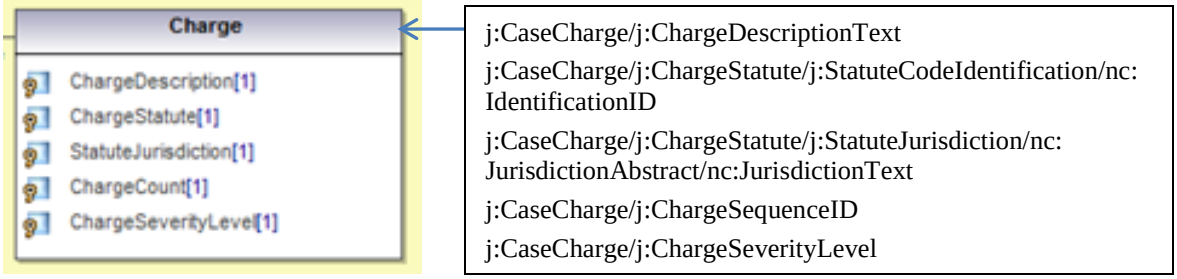

Figure 10: UML Elements Produced by Case and Charge Classes Mapped to the NIEM Model

In this manner, the modeler can walk through the class diagram mapping to the NIEM model and generate the IEPD files (wantlist, spreadsheet, and schemas). If any elements cannot be mapped, then the modeler will use an extension schema for those elements, following the NIEM IEPD development process.

A modeler could use the UML documentation stereotype to indicate the NIEM element (and any other business rules, authorities, etc.). Documentation is illustrated (Figure 11) for the case container. It is not an elegant technique, but it does leave all the information in the model.

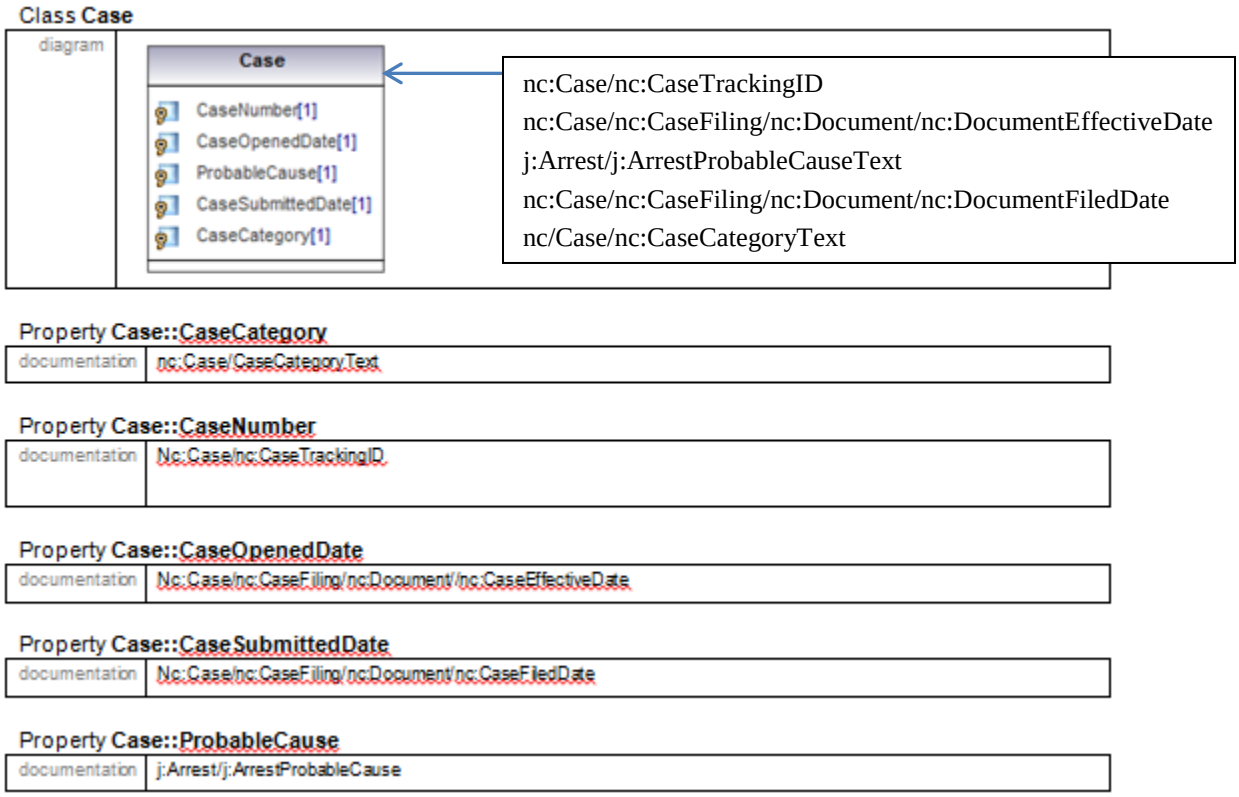

Figure 11: Case Container Example

Summary

There are many uses for "data modeling" and over the years the term has come to mean several things. Some people model data to support the design of databases, others model data to capture the underlying nature of an enterprise, while still others model the structure of a particular data management technology, such as a relational database or an XML specification. This *Technical Brief* addresses the data design model (alternatively referred to as the "logical model"), which organizes the data into tables, UML classes, or XML "complex" types.

NIEM information exchanges are about XML and messaging. Each message is inherently hierarchical and can contain anything the sender might want to send, organized in terms of XML that describe information pieces and defined in an XML schema of types, relationships, and properties. While not a one-to-one match, UML is highly proficient at modeling these elements. Appendix A shows the NIEM UML Profile equivalents of NIEM elements to UML elements.

As important as the data design model, UML provides the ability to depict the physical exchange environment and deployment in an integrated model. Many NIEM initiatives languish in design without coming to real-world fruition. UML can help communicate the intent and specifications to the supporting technologists and support staff and hopefully, increase the NIEM success stories.

Appendix A: NIEM UML Profile

The NIEM UML profile maps NIEM components to UML, as summarized in this table:

NIEM Concept Reference	Stereotype	UML Element
Namespace	<<Namespace>>	Package
TYPES		
Object Types	<<ObjectType>>	Class
Role Types	<<RoleOf>>, <<RolePlayedBy>>	Class
Association Types	<<AssociationType>>	Class, Association Class
Metadata Types	<<MetadataType>>	Class
Augmentation Types	<<AugmentationType>>	Class
Adapter Types	<<AdapterType>>	Class
Choice Groups	<<Choice>>	Class
Property Holders	<<PropertyHolders>>	Class
Complex Types	<<isAbstract>>	Class
Unions	<<Unions>>	Datatype
Lists	<<Lists>>	Datatype
Simple Types	<<ValueRestriction>>	Datatype
Primitive Types	None	PrimitiveType
Code Types	None	Enumeration
RELATIONS		
Content and Reference Properties	None	Aggregation
Complex Types (Extension Type)	None	Generalization
Augmentation Types	<<Augments>>	Generalization
Role Types	<<RolePlayedBy>>	Generalization
Property Holders and References	<<References>>	Realization
Subsets and Reference Models	<<Subsets>>	Realization
Augmentation Types	<<AugmentationApplication>>	Usage dependency
Metadata Types	<<MetadataApplication>>	Usage dependency
PROPERTIES		
Properties	<<XSDProperty>>	Property
Properties	None	Multiplicity (Property)
Substitution Groups	None	Subsets (Property)
Substitution Groups	None	Derived Union
GENERAL		
NIEM Names	None	Name (Named Element)
Namespaces, Complex Types, Simple Types, Properties	<<Documentation>>	Comment

Bibliography

Ambler, S. W. (2004). *The Object Primer, Agile Model-Driven Development with UML 2.0.* Cambridge University Press.

Ambler, S. W. (2014). *Effective Practices for Modeling and Documentation*. Retrieved from Agile Modeling.

Booch, G. (1993). *Object-Oriented Analysis and Design with Applications* (2nd ed.). Addison-Wesley Professional.

Bureau of Justice Assistance, U.S. Department of Justice. (2012). *Global Reference Architecture Framework.*

Hay, D. (2014). *National Information Exchange Model Core Evalution.* Houston, TX: Essential Strategies International.

Koch, C. (2002). "Supply Chain: Hershey's Bittersweet Lesson." *CIO*.

Object Management Group. (2013). *UML Profile for NIEM (NIEM-UML).*

Rational Software Corp. (2001). "Rational Unified Process: Best Practices for Software Development Teams." Rational Software White Paper.

Sparx Systems. (2007). *Using UML Part 1 – Structural Modeling Diagrams.* Using UML series.

Sparx Systems. (2007). *Using UML Part 2 – Behavioral Modeling Diagrams.* Using UML series.

Weiss, P. (2007). *Modeling of Services and Service Collaboration in UML 2.0*. Brno University of Technology, Czech Republic.

This project was supported by Grant No. 2009-BE-BX-K030 awarded by the Bureau of Justice Assistance. The Bureau of Justice Assistance is a component of the Office of Justice Programs, which also includes the Bureau of Justice Statistics, the National Institute of Justice, the Office of Juvenile Justice and Delinquency Prevention, the SMART Office, and the Office for Victims of Crime. Points of view or opinions in this document are those of the author and do not represent the official position or policies of the United States Department of Justice.