Technical Brief

August 2015

SEARCH
The National Consortium for Justice Information and Statistics

A Unified Modeling Language Approach to Modeling NIEM Exchanges: Overview and Scenario Planning

By Diane Lacy
Information Sharing Specialist
SEARCH

Introduction

Purpose

This is the first of two *Technical Briefs* that provide an approach to using the Unified Modeling Language[1] (UML) 2.0 to model web services associated with exchanges that are compliant with the National Information Exchange Model (NIEM)[2] and the Global Reference Architecture (GRA).[3] The intent 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 exchange services.

A NIEM Information Exchange Package Document (IEPD) defines a recurring message in XML and is built to satisfy information exchange business requirements. A GRA Service Specification also describes other required aspects of information exchange implementations, including access controls, security, policy automation, transmission protocol, and others. Both NIEM and GRA are closely associated with Service-Oriented Architecture (SOA), as is UML. UML provides features and techniques to model and assemble components into orchestrated SOA services.

This brief provides an overview of UML and discusses features and techniques that modelers can use to support the **Scenario Planning** phase of IEPD development (Figure 1 shows the six phases in NIEM IEPD life cycle development). The second *Technical Brief* discusses the UML diagrams that support the **Analyze Requirements** and **Map and Model** phases and briefly discusses the service infrastructure environment.

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

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

[3] https://it.ojp.gov/initiatives/gra

Figure 1: NIEM IEPD Life Cycle Development

Figure 2 is a process diagram that summarizes the topics covered in the two *Technical Briefs*. This approach aligns with the required and recommended documentation of a NIEM IEPD and a GRA Service Specification Package (SSP).

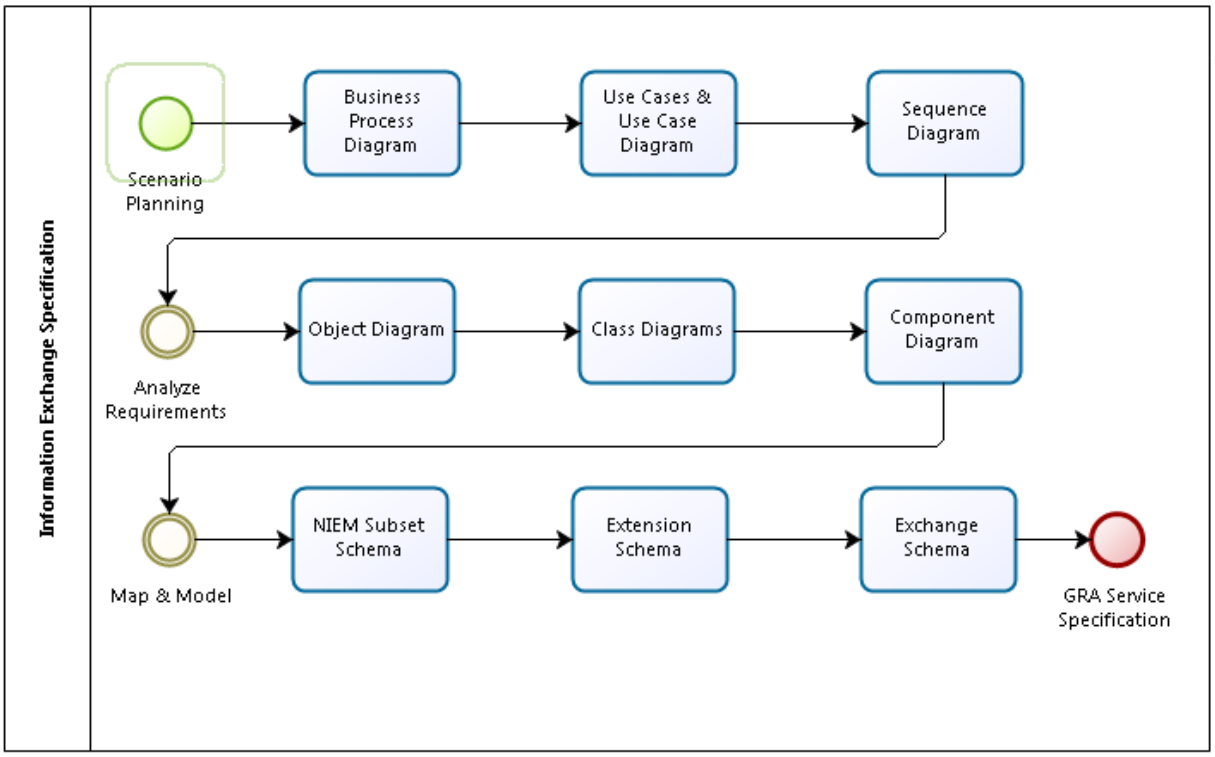

Figure 2: Information Exchange Process Diagram: Scenario Planning, Analyze Requirements, Map and Model

Overview

UML is a widely used, general-purpose modeling language in the field of software engineering; it is designed to provide a standard way to visualize the design of an automated system. UML uses industry-standard modeling notations with a comprehensive set of diagrams and elements. By visually documenting requirements, information, and processes rather than communicating this through narrative documentation, UML increases clarity and understanding of the components. UML allows a modeler to represent relationships and dependencies and establishes traceability back to the foundational business requirements. A full UML model transforms business requirements into technical specifications. Using UML provides these features and benefits:

- Standardized notation, which eliminates ambiguity
- Notation, which mitigates language-dependent documentation
- Diagrams that provide traceability back to requirements
- Support of both traditional and "agile/iterative" software development life cycles (SDLC)
- High reusability, due to decomposition of component-based architecture
- Design integrity (from the modeler's point of view)

- A unified context for all service components (from the developer's point of view)
- Enhanced productivity, reduced rework, and easy refactoring when requirements change
- An effectively governed and widely supported standard

Background

During the early-to-mid-1990s, the expanding use of the Internet and web services provoked the development of service-oriented and object-oriented development methods.[4] The Internet boom brought a plethora of object-oriented development languages into mainstream development, along with new development methodologies like *agile*, *extreme programming (XP)*, and *scrum*.

As web services and governing standards matured, enterprise technology strategies began incorporating SOA to support core business lines, predominantly Enterprise Resource Planning and Multi-Channel Retail. However, traditional SDLC methodologies were at odds with fast and iterative service deployments. Regardless, the rush to join the dot-com environment trumped the software development discipline, resulting in many failed IT projects (see box).[5] Interestingly, these failures contributed to the introduction of project management, project management offices (PMOs), and enterprise architecture frameworks into the corporate structure.

Failed IT Projects

Hershey, 1999. Hershey experienced problems with its new order-taking, customer management, and supply chain planning systems. These prevented the company from delivering $100 million worth of Halloween candy that year and caused its stock to dip 8%.

NIKE, 2000. NIKE's bold $400 million project to unite all their systems into one "super-system" resulted in $100 million in lost sales, a 20% stock market dip, and a collection of class-action lawsuits.

During this time, the Rational Software Corporation began to develop a Unified Modeling Language. They agreed upon a standardized notation (language), since this seemed less elaborate than method standardization. They integrated the *Booch Method* of Grady Booch, the *Object Modeling Technique* (OMT) by James Rumbaugh, and *Object-Oriented Software Engineering* (OOSE) by Ivar Jacobsen, with elements of other methods and published this new notation under the name UML version 0.9.[6] The goal was to adapt, expand, and simplify the existing and accepted types of diagrams of object-oriented methods, such as class and use case diagrams, and incorporate traditional structured methods. For example, activity diagrams are influenced by data flow charts and Petri nets.[7]

Well-known companies—such as IBM, Oracle, Microsoft, Digital Equipment Corp., Hewlett-Packard, and Unisys—were included in the further development of UML. The Object Management Group (OMG) is the standards governing organization and provides oversight for further UML development (current version is UML 2.5.). The UML 2.0 specification significantly improved the notation by supporting Model-Driven Architecture (MDA) and is the basis for this brief.

The UML Model

It is important to note that UML is a "language" for specifying requirements—it is not a method or procedure. The UML is used to define a software system or component, to provide construction detail for

[4] Booch, Grady, *Object-Oriented Analysis and Design with Applications*, 2nd ed., Addison-Wesley Professional (September 30, 1993).

[5] Koch, Christopher, "Supply Chain: Hershey's Bittersweet Lesson," *CIO* (November 15, 2002) and Wailgum, Thomas, "10 Famous ERP Disasters," *CIO* (March 24, 2007).

[6] "RUP Best Practices for Software Development Teams," Rational White Paper (November 2001).

[7] SourceMaking, "History of UML: Methods and Notations," June 2005, at https://sourcemaking.com/uml/basic-principles-and-background/history-of-uml-methods-and-notations

the components for development and deployment. The UML may be used in a variety of ways to support any SDLC methodology, but in itself does not specify that methodology or process.

A balanced UML model sufficiently describes the behavior and structure of the business requirements (for context) and web services (for development and deployment) without being overly complex. UML diagrams are categorized into behavior or structure diagrams.

- **Behavior diagrams** capture the varieties of interaction and instantaneous state within a model as it "executes" over time.
- **Structure diagrams** define the static architecture of a model. They are used to model the "things" that make up a model—the classes, objects, interfaces, and physical components. In addition, they are used to model the relationships and dependencies between components.

The modeling process starts with business modeling followed by service modeling using both behavior and structure diagrams, as depicted in Figure 3. As services are modeled, there may be a need to reassess the business diagrams to ensure alignment.

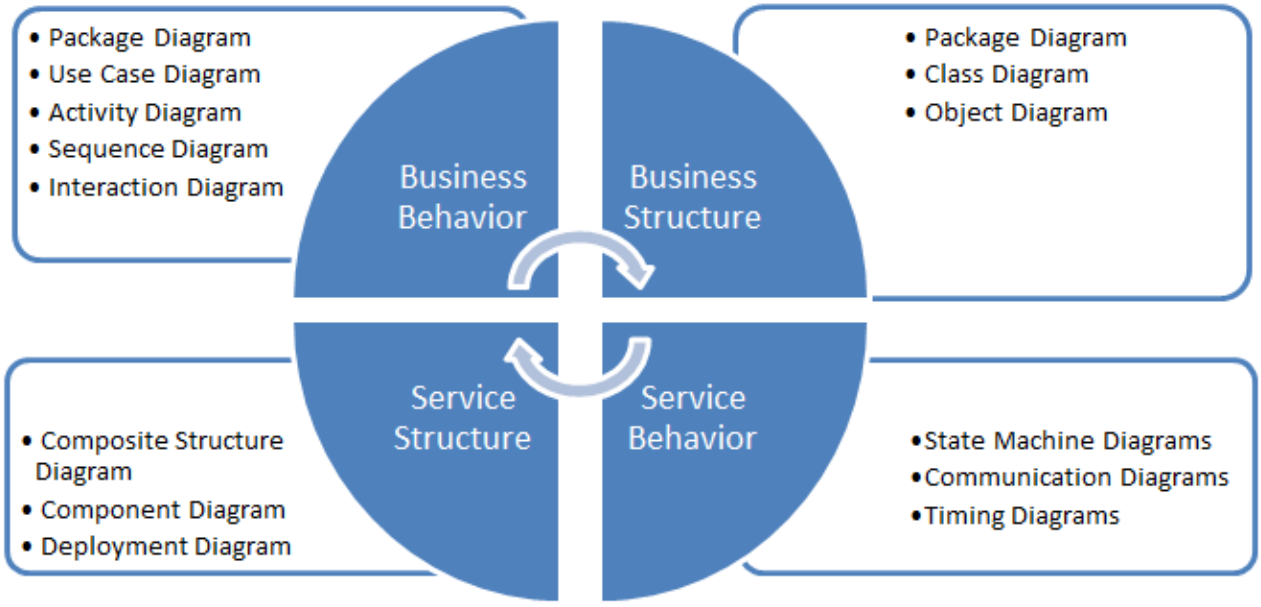

Figure 3: UML Diagram Overview

A singular UML model is the collection of these diagrams. The modeler needs to assess the complexity of requirements, systems, the information model, and more, to determine the diagram set that best meets the needs without over-engineering. UML diagrams can model business applications, user and system interfaces, integrations, information models, and deployment configurations. Since a NIEM exchange is a system-to-system environment, only a subset of diagrams is needed to effectively document the requirements and service architecture.

Table 1 summarizes UML 2.0 diagrams and their utility value to modeling information exchanges. The **Value to NIEM Modeling** column indicates a relative value of the diagram within the NIEM IEPD development cycle of a single exchange. "High" and "medium" value diagrams are the focus of the subsequent discussion. The "low" diagrams are only low from the perspective of a single exchange. Their value rises to medium or high in an orchestration of multiple exchanges, since they allow more complex modeling.

Diagram	Description	Value to NIEM Modeling
Class diagram	Shows a collection of static model elements, such as classes and types, their contents, and their relationships.	High
Object diagram	Depicts objects and their relationships at a point in time, typically a special case of either a class or a communication diagram.	High
Sequence diagram	Models the sequential logic; in effect, the time ordering of messages between classifiers.	High
Component diagram	Depicts the components that compose an application, system, or enterprise. Depicts the components, their interrelationships, interactions, and their public interfaces.	Medium
Deployment diagram	Shows the execution architecture of systems. This includes nodes, either hardware or software execution environments, as well as the middleware connecting them.	Medium
Use case diagram	Shows use cases, actors, and their interrelationships. See Figure 5, UML use case diagram guideline.	Medium
Package diagram	Shows how model elements are organized into packages, as well as the dependencies between packages.	Medium
Activity diagram	Depicts high-level business processes, including data flow, or to model complex logic within a system.	Low
Communication diagram	Show instances of classes, their relationships, and the message flow between them. Communication diagrams typically focus on the structural organization of objects that send and receive messages. Formerly referred to as a “collaboration diagram.”	Low
Composite structure diagram	Depicts the internal structure of a classifier (such as a class, component, or use case), including the interaction points of the classifier to other parts of the system.	Low
Interaction overview diagram	A variant of an activity diagram that overviews the control flow within a system or business process. Each node/activity within the diagram can represent another interaction diagram.	Low
State machine diagram	Describes the states an object or interaction may be in, as well as the transitions between states. Formerly referred to as a “state diagram,” “state chart diagram,” or a “state-transition diagram.”	Low
Timing diagram	Depicts the change in state or condition of a classifier instance or role over time. Typically used to show the change in state of an object over time in response to external events.	Low

Table 1: Summary of UML Diagrams

NIEM, GRA, and UML

NIEM IEPDs and GRA SSPs contain required and recommended business and technical information; a UML model can describe both. NIEM recommends the inclusion of business process, sequence, use case, and class diagrams. While these are effective for documenting business behavior and the XML message structure, additional information is needed to describe the technical structure and environment of the service itself.

The use of NIEM is generally associated with the use of the GRA, which defines the transport mechanisms used to exchange data. The GRA requires that services conform to web service standards, are

deployed within a SOA environment, and define the format, structure, and allowable content of the service interface.[8] A reliable and effective working service must be well designed. This is even more important if service choreography is involved. NIEM and GRA-based information exchanges act as interfaces between existing systems, replacing manual information exchange (e.g., phone, fax, email) or less timely or efficient electronic methods (e.g., batch processing, file drops). A UML model depicts these exchanges or interactions by decomposing the structure and behavior of business process and services into multiple graphical representations.

In 2013, the NIEM Project Management Office and the OMG published a NIEM-UML 2.1 Profile.[9] This NIEM specification provides for modeling NIEM in UML and producing or reverse-engineering information exchange technical specifications using Model-Driven Architecture (MDA). This reduces the time, cost, and learning curve of information exchange using NIEM. MDA also provides for other aspects of the information sharing solution, such as business processes, SOA services, and back-end system integration. Since NIEM-UML generates 100% NIEM-conformant technical specifications, NIEM-UML architects and developers do not need to worry as much about the technology details. NIEM-UML can be extended to support other technologies, such as JSON and the semantic web.

Scenario Planning

The business process model in Figure 4 depicts the overall sequence of UML models and IEPD development.

Business Process Models

Activities during the **Scenario Planning** phase usually focus on documenting the "as-is" and "to-be" business behavior. The first step in this process usually focuses on defining the context and scope of the problem to be addressed. A common approach to doing this is to use a methodology called business process modeling (BPM). BPM is not a part of UML, but provides foundational inputs to the subsequent UML diagrams and service design decisions, as well as being a recommended NIEM artifact. A UML activity diagram could be used, but they are typically used for detailed business process modeling and detailed business logic. They are applicable to designing a large application, rather than a single service exchange. The BPMN language standard is a more user-friendly presentation and is recommended for the level of detail needed in developing a NIEM exchange.

Many UML products incorporate the Business Process Modeling Notation (BPMN) standard into their product to produce business process, business collaboration, and business choreography diagrams.

BPM is an effective technique to facilitate the discussion with stakeholders to define activities and sequence. For the modeler, it is important to guide the discussion to discover the points of information exchange and not exhaustively document the business process or information flow internal to the agency. The BPM in Figure 4 depicts user activities resulting from a Call for Service event when a subject has been detained on an arrest warrant. The diagram depicts five information objects: Citation, Incident, Complaint, Charge Referral, and Information and three *manual* interagency message flows.

[8] Global Reference Architecture Framework, Version 1.9.1

[9] The specification is at http://www.omg.org/spec/NIEM-UML

The BPM helps the modeler to identify what set of services to further define. In this case, two business services are needed to pass information to the court as part of the case initiation process:

- Charge Referral Service provided by the prosecution
- Court Case Filing Service provided by the court

The Charge Referral Service will allow a law enforcement records management system (RMS) to send a charge referral document to a prosecution case management system (CMS). The Court Case Filing Service will allow either an RMS or a CMS to send a complaint or information document to initiate a court case.

Note that all the end events (Service Call Closed, Cite and Release, Booking, Charge Deficiency Issues, Case Dismissed, and Arraignment) would be decomposed into their respective BPMs and additional services may be identified. In this manner, a robust set of BPMs would guide a strategic plan to develop a portfolio of services.

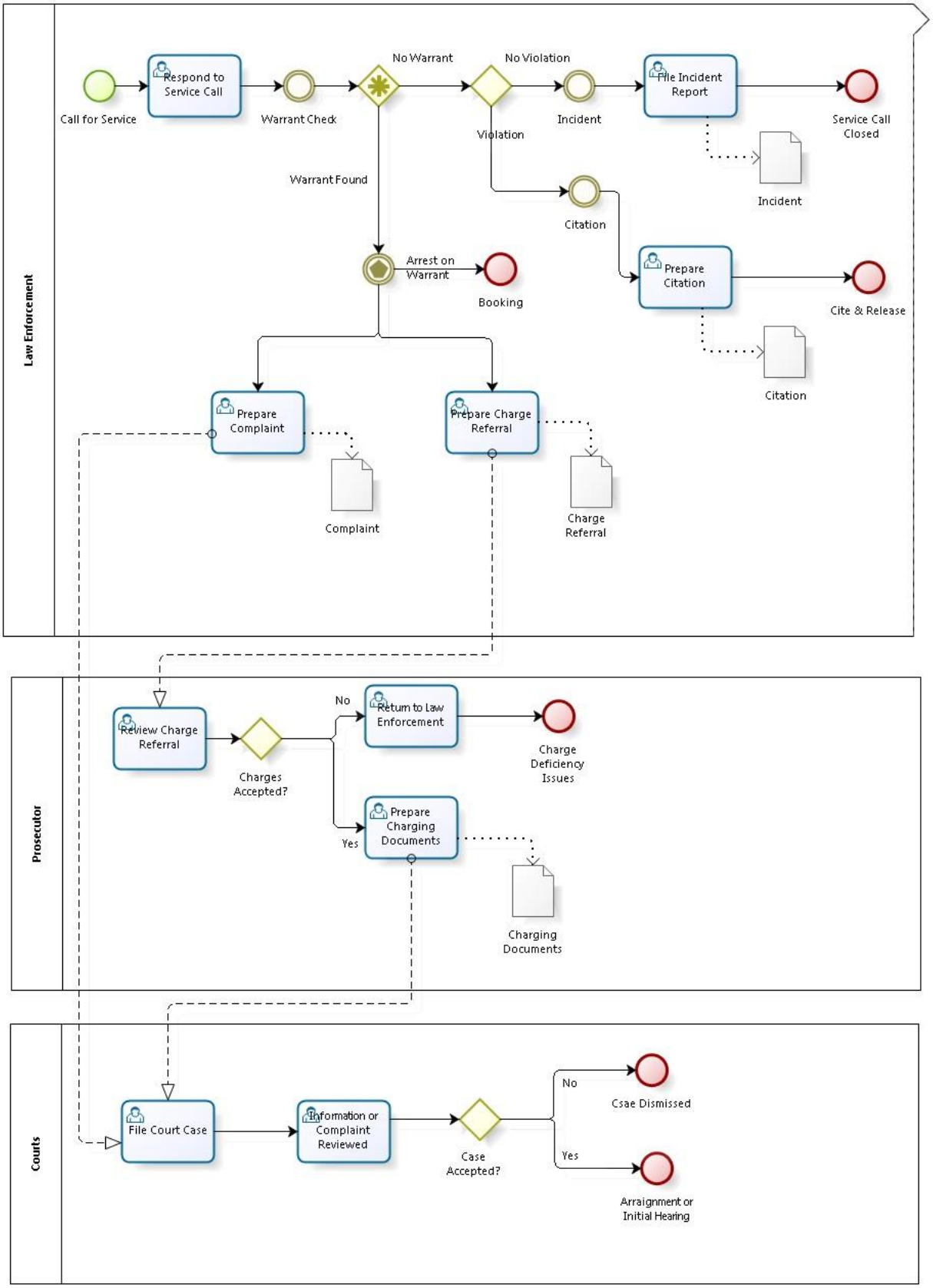

Figure 4: Business Process Model

Use Case Diagram

Use cases are developed during the **Scenario Planning** phase and refined during the **Analyze Requirements** phase. A use case (UC) is a single unit of meaningful work, providing a high-level view of interaction between "someone" and a system. A UC typically includes the following information:

- **Name**: The name of the UC.
- **Brief Description**: A brief description of the role and purpose of the UC.
- **Flow of Events**: A textual description of what the system does in regard to a UC scenario (not how specific problems are solved by the system). Write the description so that the customer can understand it. The flows can include a basic flow, alternative flows, and sub-flows.
- **Key scenarios**: A textual description of the most important or frequently discussed scenarios.
- **Special Requirements**: A textual description that collects all of the requirements of the UC that are not considered in the UC model, but that must be taken care of during design or implementation (for example, nonfunctional requirements).
- **Preconditions/Post-conditions**: A textual description that defines a constraint on the system when the UC starts and ends.
- **Extension points**: A list of locations within the flow of events of the UC at which additional behavior can be inserted by using the extend-relationship.

A UC diagram shows a collection of related use cases to provide a composition of a larger body of work. The UC diagram depicted in Figure 5 illustrates potential use cases that could define the user interfaces to initiate a NIEM exchange, but not the exchange itself. These UCs would be a starting place to define the changes to a law enforcement RMS, or prosecutor or court CMS to create, send, and receive NIEM messages.

> Vendor integration activities are frequently the critical path in deploying NIEM exchanges. It is highly beneficial to document where/when data are created and updated within the participating systems in order to determine what service action capabilities are needed.

Since NIEM exchanges are system-to-system exchanges, UCs and UC diagrams put the services in context with the business process and are beneficial for the overall SDLC, but not essential for the service specification.

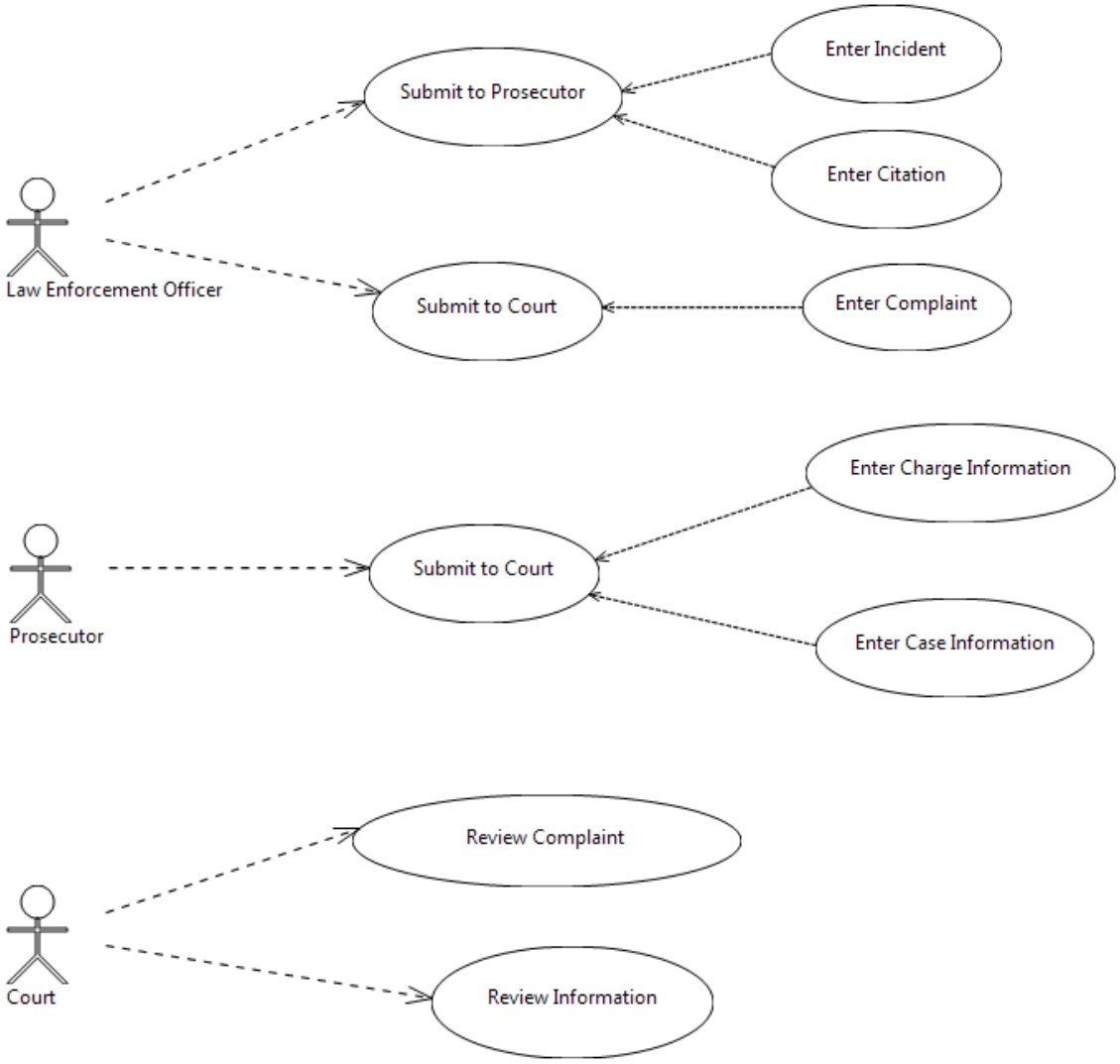

Figure 5: Use Case Diagram

Package elements are used to group and organize other elements and exist in many UML diagrams, but are most commonly used in UC diagrams and class diagrams. UC diagrams are typically used to display use cases within system boundaries. There are three systems shown in the example in Figure 6. Packages can later be assembled into package diagrams.

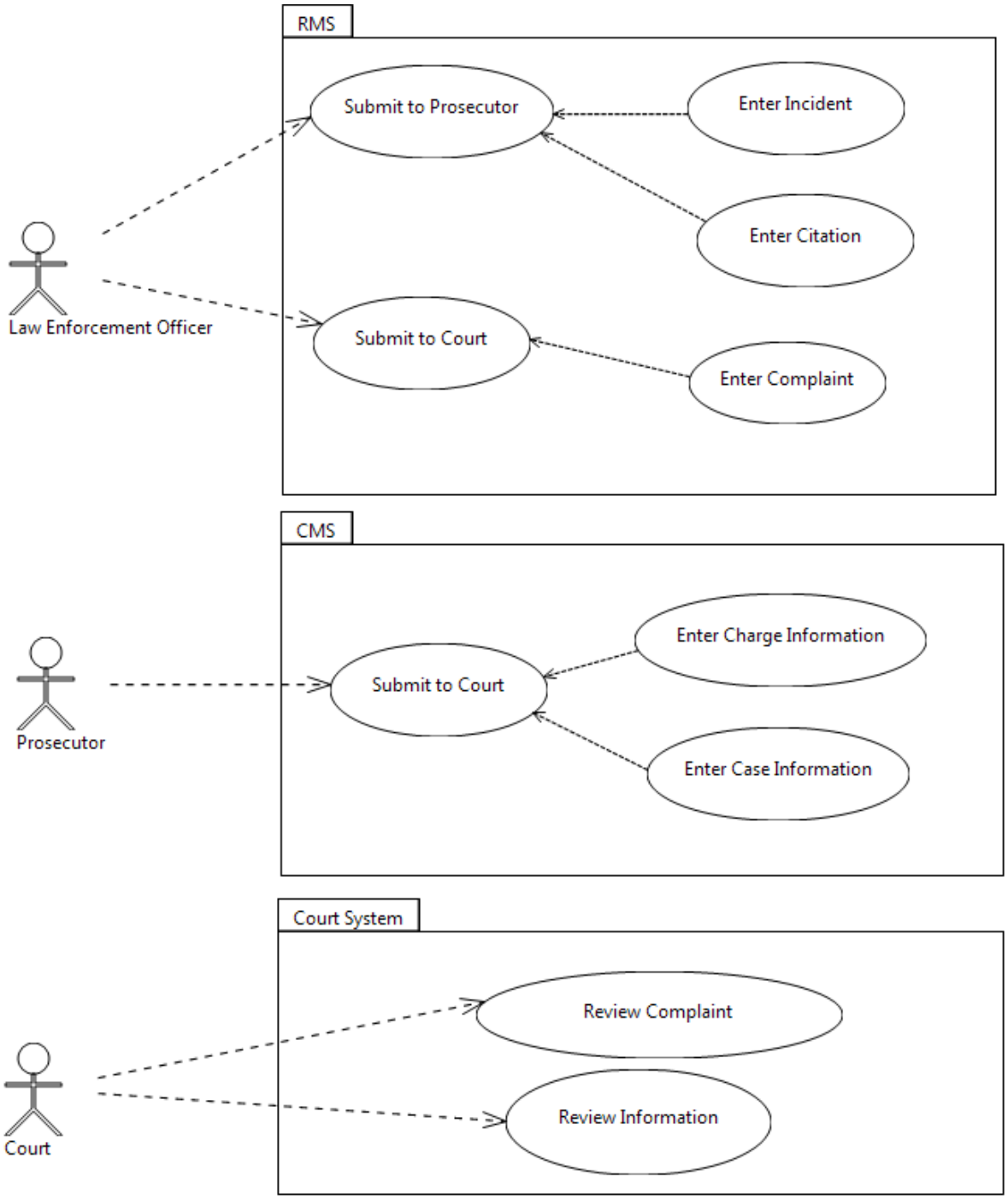

Figure 6: Use Case Diagram with Package Elements

Sequence Diagrams

Sequence diagrams are used to model the sequential logic and ordering of messages and are very effective for transitioning from requirements to more formal technical specifications. A single use case is refined into one or more sequence diagrams.

A sequence diagram is a form of interaction diagram that shows objects as lifelines running down the page, with their interactions over time represented as messages drawn as arrows from the *source lifeline*

to the *target lifeline*. Sequence diagrams are good for showing which objects communicate with which other objects, and what messages trigger those communications. Sequence diagrams are not intended for showing complex procedural logic.

The diagram in Figure 7 illustrates messages flowing from a law enforcement RMS to a prosecution Charging Service and Court Case Filing Service. At this point in the modeling, the message structure has not yet been defined. Once the message structure is defined, the modeler can update the sequence diagram to associate the information object with the message. This is the iterative nature of modeling behavior first, then modeling structure, and then associating behavior with structure.

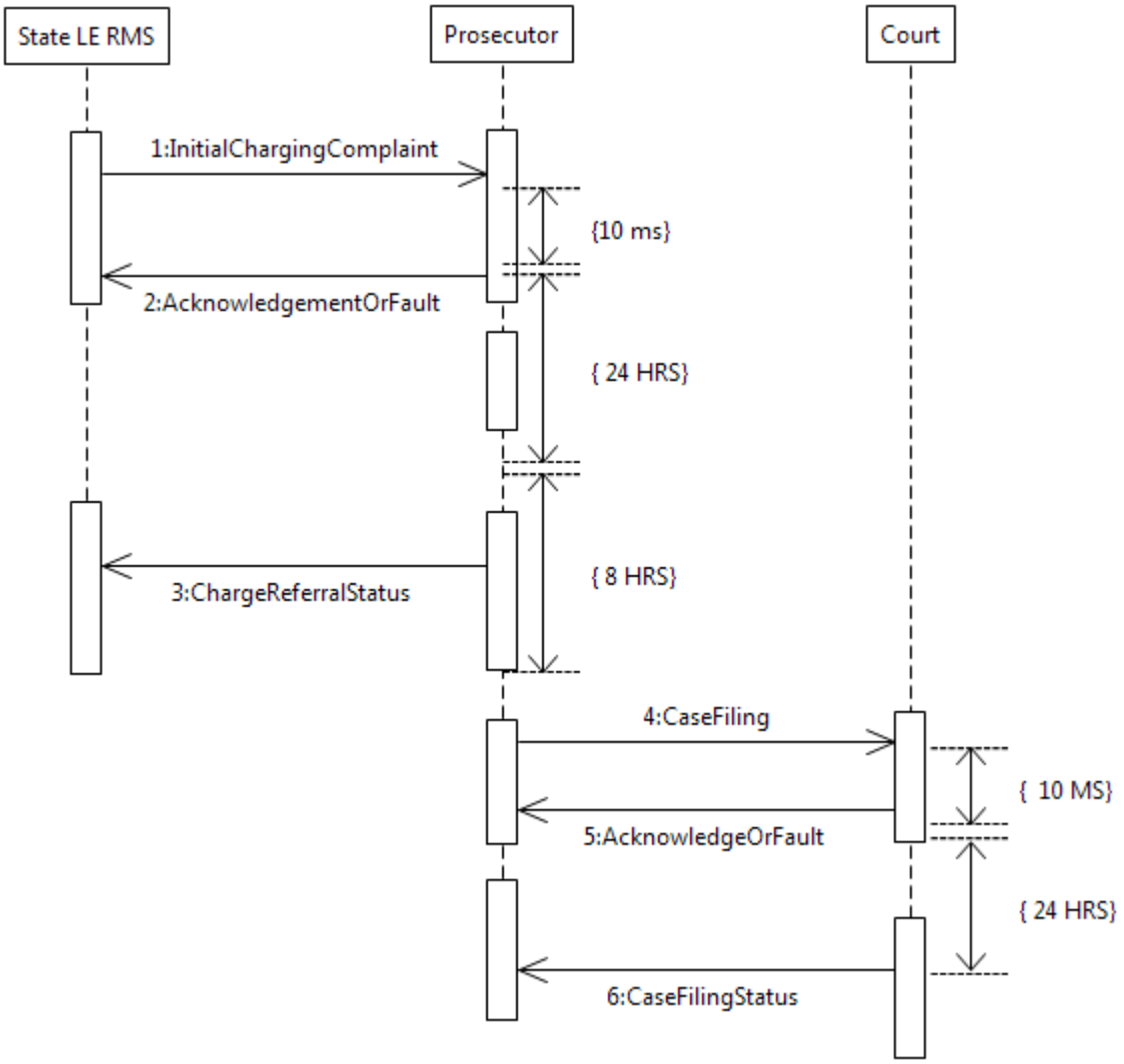

Figure 7: Sequence Diagram

Summary

At this point, the UML diagrams for the **Scenario Planning** phase are modeled. They will be used as input for the **Analyze Requirements** phase to model the information and service components structures. Analysts will review the complete set of diagrams for consistency and may develop a supporting business process narrative. Within a NIEM IEPD structure, these artifacts are stored within the Service Model folder:

IEPD Root Folder:

- Artifacts (folder)
- Service Model (folder)
    - Behavior (folder)
        - Business Process Model(s)
        - Use Cases
        - Sequence Diagrams
        - Business Process Description Document
    - Information (folder)
        - Object Diagrams
        - Class Diagrams
        - Component Diagram

The object, class, and component diagrams are discussed in a companion document, *A Unified Modeling Language Approach to Modeling NIEM Exchanges: Analyzing Requirements.*

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.*

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

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.