Chapter 4
UML and XML

This chapter provides a short introduction to UML and XML, key underlying standards used in healthcare interoperability.

4.1 Unified Modeling Language

UML stands for Unified Modeling Language. The term "Unified" in the name is a clue to its origins. During the early 1990s, a number of different modeling notations were in widespread use. This was, to say the least, confusing. Between 1995 and 1997, Rational Software (now part of IBM) brought together three leading methodologists (Booch, Rumbaugh, and Jacobson) and, together with the Object Management Group (OMG), they developed a modeling notation and language, which they called the Unified Modeling Language that combined many of the best ideas of the then existing methods. UML is now the standard modeling notation used throughout the IT industry.

UML Version 1.0 was issued in 1997; a major revision, UML 2, was issued in 2003. Most of the changes in UML 2 are technical refinements, intended to support the concepts of executable UML and enable the exchange of models and diagrams between different tools.

UML is a specialized modeling language, not simply a notation for drawing diagrams. It includes a notation, which is used on diagrams, and a meta-model, which is of interest primarily to the developers of UML software tools. Simple things are simple, but complex things require the use of purpose-built tools.

UML makes a critical distinction between models and diagrams. A model is the sum of all the information held about a project in UML. Each diagram is a partial view of this model. When learning UML, it is a convenient simplification to regard a model as the sum of the diagrams. Each diagram shows a small part of the total design. The model is the sum total of all the specifications, comprising hundreds of diagrams and supporting text. This is one reason why specialized UML modeling tools are needed for building UML models. Each tool maintains an internal repository, which facilitates the reuse of common components and avoids all sorts of problems produced by describing the same thing in a different way in different places.

T. Benson, *Principles of Health Interoperability HL7 and SNOMED*, HI,
DOI 10.1007/978-1-84882-803-2_4, © Springer-Verlag London Limited 2010

Fowler (Fowler 2004) identifies three main ways in which people use UML: sketch, blueprint, and programming language. Sketches can be made using white boards or multipurpose tools such as Visio or PowerPoint. We may start off using sketches, but soon need to move on to developing blueprints. The distinction between a sketch and a blueprint is that sketches are incomplete and exploratory, while blueprints are complete and definitive. Serious modeling (blueprints rather than sketches) requires a specialized UML tool.

The step beyond blueprint is when programs are produced directly from the model. Here, UML becomes the source code for executable code. Full use of this approach will lead to automated production of conformant XML schema, documentation, and test rigs. A new generation of tools is being developed using the Eclipse Framework by collaborations such as Open Health Tools (OHT).

UML allows beginners to do simple things simply, yet also supports highly complex applications, underpinned by a rigorous formal language. No other scheme is so scalable or complete.

UML is completely independent of the software used to implement computer applications and is not tied to any development methodology. UML's independence of technology and method is one of the keys to the wide support that it enjoys throughout the IT industry. It fits into any IT organization.

UML has a number of weaknesses. Models and diagrams created using different tools cannot be imported and exported into and out of different tools reliably, although solutions to this problem have been proposed. It does not have a neat way of specifying multiple choices, decision tables or other constraints, although it does have a special Object Constraint Language (OCL), based on predicate calculus. However, this is opaque to those not trained in computer science (Warmer and Kleppe 2003). In many models, unstructured text annotations form an important part of the documentation.

A premise of UML is that no single diagram (or type of diagram) can provide, on its own, a full representation of what goes on, and so we need to use sets of related diagrams. Each type of diagram only shows certain aspects of a situation – everything else is ignored. This simplification provides both the power (it makes the situation understandable) and the weakness of diagrams (each diagram has a limited scope).

4.1.1 UML Diagrams

UML diagrams relate either to information structure or to behavior (Fig. 4.1). UML 2 recognizes 13 diagram types, although most users of UML make do using three or four types of diagram, depending on what they are using it for.

One danger point is UML's principle of suppressing information. This allows information to be omitted from any diagram in order to make it easier to understand. The corollary is that you should never infer anything from the absence of information in a diagram and that UML diagrams should not be read on their own without access to the rest of the model.

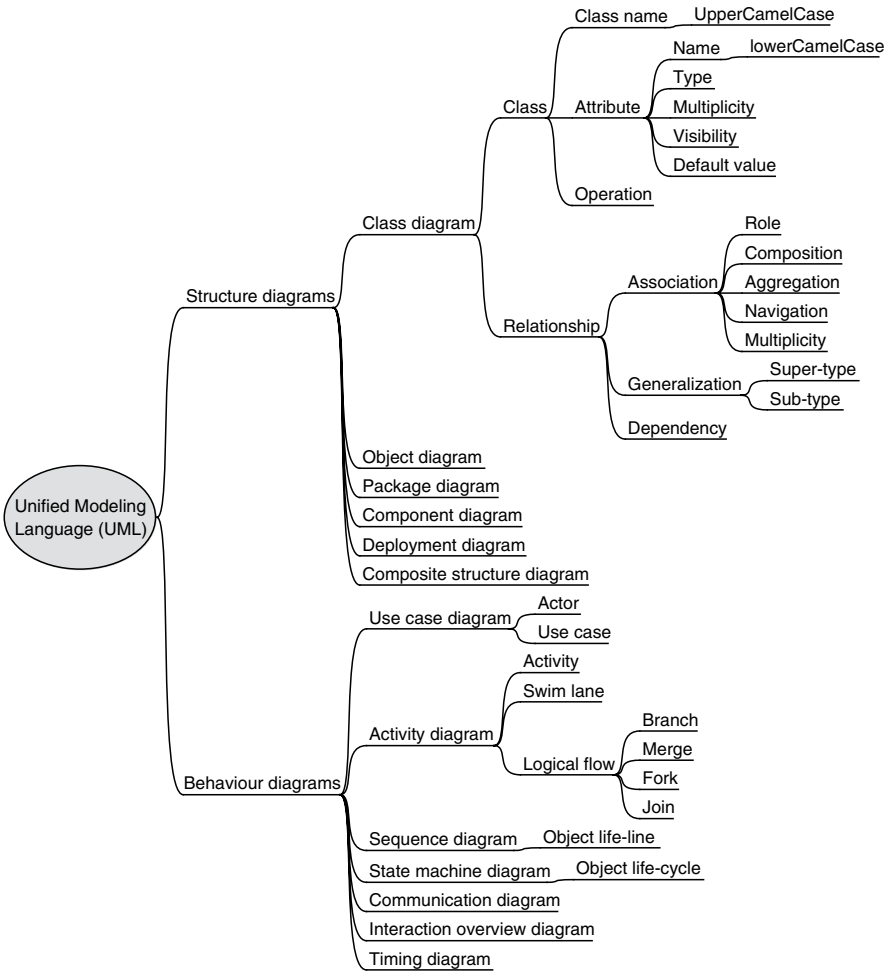


Fig. 4.1 UML diagrams and components

For example, all attributes have a default multiplicity of [1] (mandatory), but if a multiplicity is not shown against an attribute in a diagram, it may either mean that the information is suppressed or that the default value should be used. For example, no attribute multiplicities are shown in the HL7 RIM (Reference Information Model), because they are suppressed. Structural attributes in the RIM are mandatory [1] while all others are optional (but this is not shown on the diagrams). The only way to check is to look deeper into the model and see what is really there. Individual organizations often develop their own conventions about what is and is not shown on each diagram.

Diagram layout and style facilitate understanding (see Ambler 2003). Some guidelines apply to all types of diagrams, such as:

- Diagrams should be laid out so that they can be read left-to-right and top-to-bottom.
- Avoid crossed, diagonal, and curved lines.
- Document diagrams using notes.
- Use the parts of UML that are widely understood, not the esoteric parts.
- Use color coding with discretion.
- Use common naming conventions such as UpperCamelCase (e.g., ClassName) and lowerCamelCase (e.g., attributeName).
- Do not put too much on a single diagram. Restrict diagram size to a single sheet of A4.
- Use consistent legible fonts.
- Show only what you need to show. It is good practice to suppress unnecessary detail.

4.2 Structure Diagrams

4.2.1 Class Diagrams

Class Diagrams are the most widely used UML diagrams. Class diagrams show the static structure of classes, their definitions, and relationships between classes.

A class is a description of a group of objects with properties (attributes), behavior (operations), relationships to other objects (associations and aggregations), and semantics. Classes are shown as rectangles with one, two, or three compartments. The top compartment shows the class name, the second shows attributes. Attributes describe the characteristics of the objects, while operations are used to manipulate the attributes and to perform other actions. Attributes and operations need not be shown on a particular diagram.

Class names should be a singular noun and the class name is conventionally written using UpperCamelCase notation (e.g., ClassName). This class has two attributes, documented and date, and one operation – create().

Figure 4.2 shows a simple class diagram representing a prescription. Each prescription has a prescriber (author) and relates to a single patient. It has one or more prescription lines. Each prescription line includes details of a drug and may have any number (zero to many) of dosage instructions.

The arrowheads on the lines (associations) show navigation. The arrow from Prescription to Patient shows that, in this model, one goes from Prescription to Patient but not the other way round.

The notation for multiplicity, used in associations and attributes, is:

- [0..1] optional, no more than one is allowed
- [*] or [0..*] optional, any number of instances is allowed

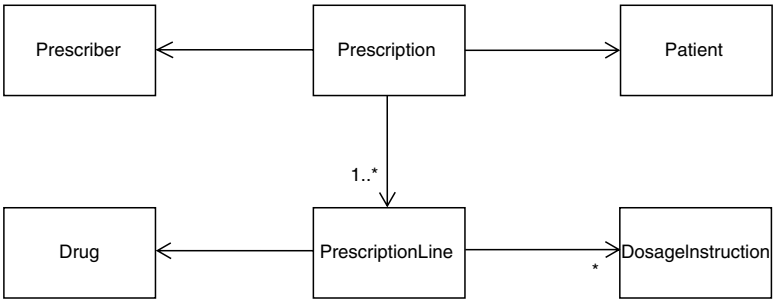
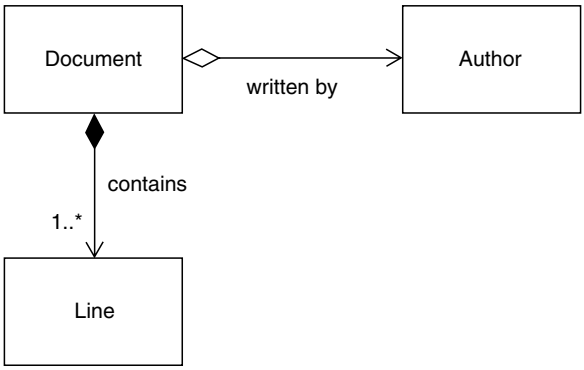


Fig. 4.2 Simple class diagram

Fig. 4.3 Aggregation and composition



- [1] mandatory always one required (this is the default)
- [1..*] mandatory to have at least one instance

If no multiplicity is shown, the default assumption is multiplicity of 1, meaning that exactly one is required, although caution must always be observed when inferring anything from the absence of data on diagrams.

Figure 4.3 illustrates two notations used for showing containment.

The black diamond indicates composition between Document and Line. Each Document contains one or more Lines, but Line cannot exist independently of Document.

The hollow diamond indicates aggregation between Document and Author. Each Document has one Author, but the Author can exist independently of Document.

The multiplicities used in any diagram depend on its purpose. Figure 4.3 (above) is a Document-centric showing Document has just one Author (a one-to-one relationship), but an Author-centric diagram, Fig. 4.4 (below) shows that Author has one or more Documents (a one-to-many relationship). Both are right, it just depends on the purpose of the model.

Fig. 4.4 Author-centric diagram

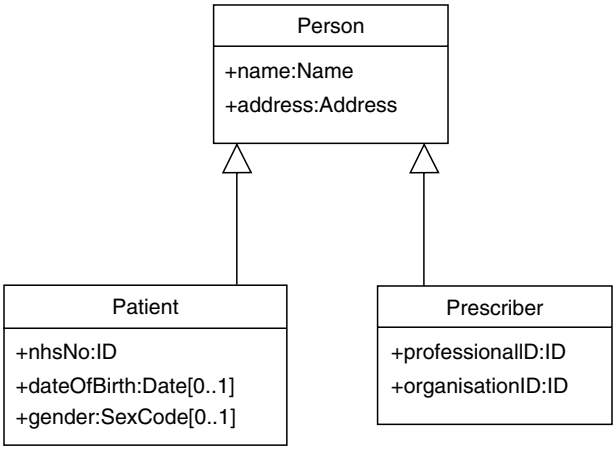


Fig. 4.5 Person specialization

The concepts of inheritance and attributes are illustrated in Fig. 4.5. Patient and Prescriber are both specializations of Person; Person is a generalization of Patient and Prescriber. The triangle arrowheads indicate that both Prescriber and Patient classes inherit the properties of Person. Patient has attributes: nhsNo, dateOfBirth, and gender, but also inherits the attributes name and address from Person. Similarly Prescriber, has attributes professionalID and organizationID, as well as the properties of Person.

Attributes have several properties. For example, the notation + dateOfBirth: Date[0..1] indicates:

- Visibility (+) is public, meaning that it is fully accessible.
- Attribute name is "dateOfBirth."
- The attribute type is "Date."
- Multiplicity is [0..1] meaning that this attribute is optional with a maximum number of occurrences of one.
- Initial values and defaults may also be specified.

Attribute names are usually written in lowerCamelCase (e.g., attributeName).

Operations implement the functionality of a software object. They are the actions that an object knows how to carry out. The syntax for operations includes:

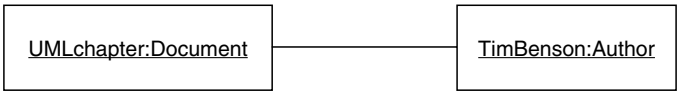


Fig. 4.6 Simple object diagram

- Visibility
- Name
- Parameter list (in parenthesis)
- Return type
- Property string

An object is a unique instance of a class. An object diagram, such as Fig. 4.6, shows the relationships between objects. Each object may have:

- Identity (name)
- State (attributes)
- Behavior (methods)

The object name is underlined (to distinguish it from a class) and comprises the object's name, which is optional, followed by a colon and the class name (e.g., TimBenson:Author).

4.2.2 Package

Packages are used to divide up a model in a hierarchical way. Each package may be thought of as a separate name space. Each UML element may be allocated to a single package.

Packages provide a useful means of organizing the model. Classes that are closely related by inheritance or composition should usually be placed in the same package.

In Fig. 4.7, the Party package might include all classes related to people and organizations, including patients, doctors, and nurses. The Interaction package might include messages and entries in clinical records. The dashed arrow from Interaction to Party indicates that Interaction has a dependency on Party (Interactions involve Parties).

4.2.3 Deployment

The physical organization of computer systems is shown in deployment diagrams (Fig. 4.8). Each piece of the system is referred to as a node. The location of software can be shown as components.

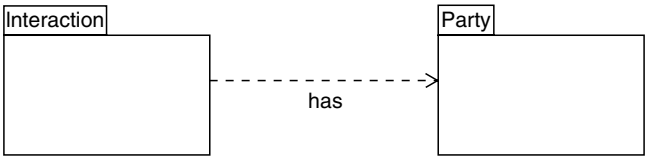


Fig. 4.7 Package diagram

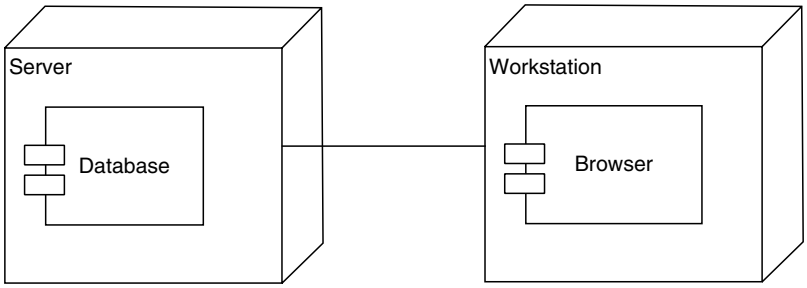


Fig. 4.8 Deployment diagram

4.3 Modeling Behavior

4.3.1 Use Case

Use cases capture the behavioral requirements of business processes and provide a common linkage across all aspects of a project from initial analysis of requirements right through development, testing, and final customer acceptance. They show how people will ultimately use the system being designed. Each use case describes a specific way of using the system. Any real system has many use cases.

Each use case constitutes a complete course of events, initiated by an actor (or trigger). A use case is essentially a special sequence of related transactions performed by an actor and the system in a dialogue.

An actor is an external party, such as a person, a computer, or a device, which interacts with the system. Each actor performs one or more use cases in the system. By going through all of the actors and defining everything they are able to do with the system, the complete functionality of the system is defined.

Each use case is a description of how a system can be used (from an external actor's point of view); it shows the functionality of the system, yielding an observable result of value to a particular actor. A use case does something for an actor and represents a significant piece of functionality that is complete from beginning to end.

The collected use cases specify all the ways the system can be used. Nontechnical personnel can understand use cases intuitively. Thus they can form a basis for

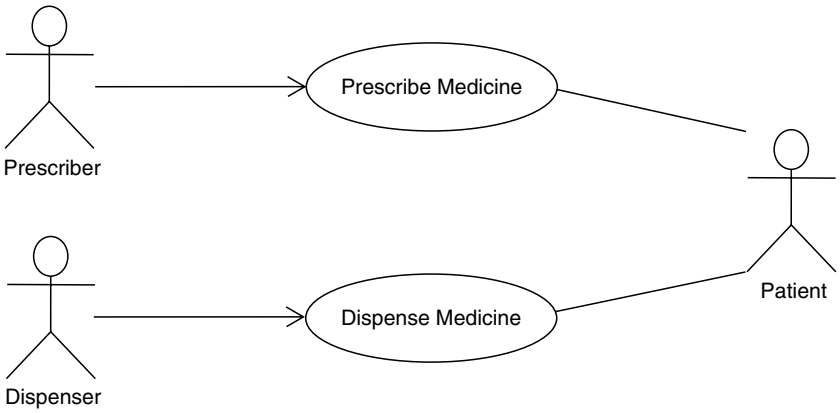


Fig. 4.9 Use case diagram

communication and definition of the functional requirements of the system in collaboration with potential users.

A simple use case diagram is shown in Fig. 4.9. Stickmen represent actors; ellipses represent use cases. In this diagram, the principal Actors are shown on the left. The arrowheads indicate the actor that initiates the use case.

Use cases are fundamentally a text form and should be documented using simple templates, such as:

- Metadata, such as use case name, unique ID, author, date, version, and status
- Scope and context
- Primary and other actors
- Preconditions and trigger event
- Main success scenario describing the normal flow of events using numbered steps from trigger through to post-conditions
- Post-conditions
- Alternative flows, e.g., when errors occur
- Importance and priority
- Open issues

A scenario is an instance of a use case. It is one path through the flow of events for the use case and can be documented using an activity diagram or a storyboard free text description (Fig. 4.10).

4.3.2 Activity Diagram

Use activity diagrams to show the important business processes undertaken by each role, such as validation and database update. Each role may be shown in a separate "Swim Lane." Transactions are communications that cross swim lanes.

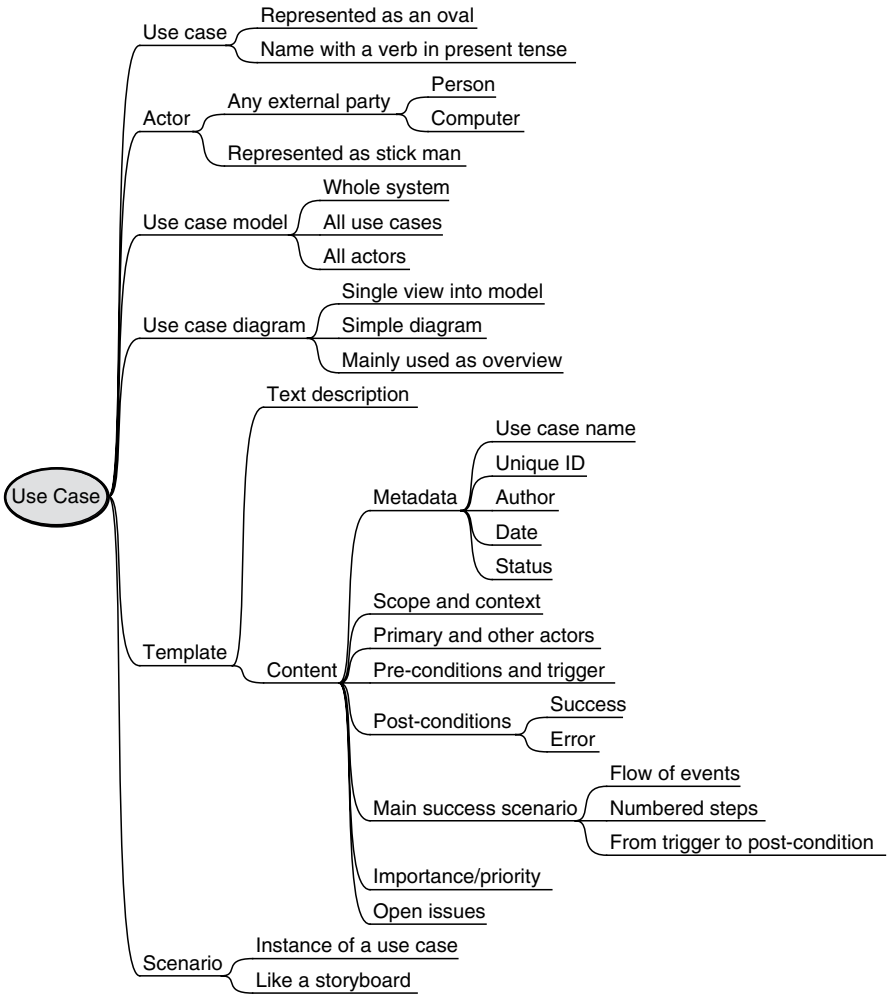


Fig. 4.10 Use cases

Activity diagrams display a sequence of actions (including alternative execution paths) and the objects involved in performing the work. They are particularly useful for describing workflow and behavior that have branches and forks. Figure 4.11 shows a simplified activity diagram for the exchange of a referral and clinic letter between GP and hospital. It is organized in swim lanes to show who or what is responsible for each activity.

Activity diagrams can be used to show logical data flows. A branch has a single entry point, but a choice of exits depending on some condition. Only one route can be taken. Branches end at a merge. A fork has one entry and multiple exits, which can be undertaken in parallel, and the order of activities is not important. A fork ends at a join.

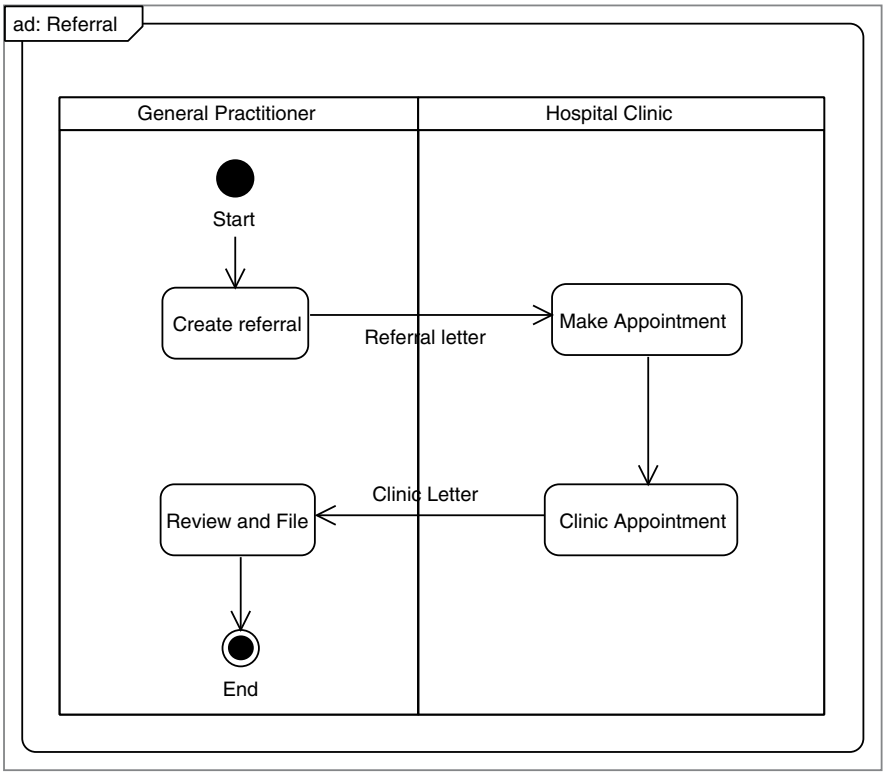


Fig. 4.11 Activity diagram

4.3.3 Sequence Diagram

Sequence diagrams (e.g., Fig. 4.12) show how objects interact with each other. Sequence diagrams show when messages are sent and received.

A Sequence Diagram is a diagram that depicts object interactions arranged in time sequence, where the direction of time is down the page. The objects, which exchange information, are shown at the top of a vertical line or bar, known as the object's lifeline. An arrow between the lifelines of two objects represents each message.

4.3.4 Statechart Diagram

An object state is determined by its attribute values and links to other objects. A state is the result of previous activities of the object. A state is shown as rectangle with rounded corners. It may optionally have three compartments (like classes) for name, state variables, and activities (Fig. 4.13).

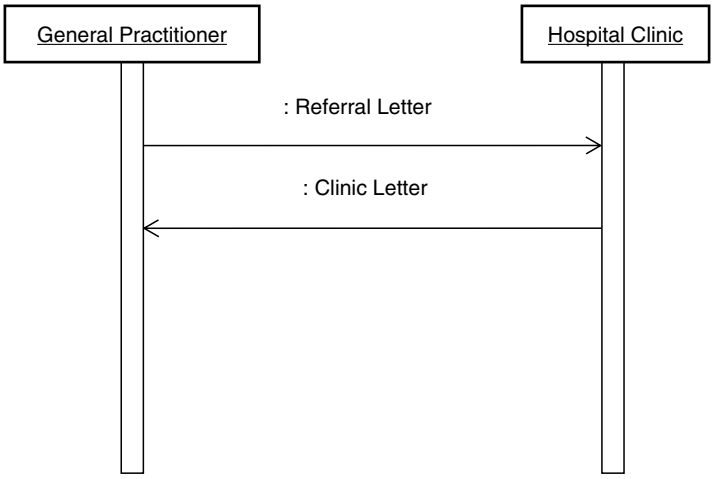


Fig. 4.12 Sequence diagram

Fig. 4.13 Statechart diagram

A statechart diagram shows an object life cycle, and can be used to illustrate how events (messages, time, errors, and state changes) affect object states over time. State transitions are shown as arrows between states.

4.4 UML Summary

UML is the standard modeling notation used for healthcare IT purposes. The basic notation is simple and quick to learn, although this hides much of the complexity that is needed for some purposes. UML is used to prepare rough sketches or to produce stringent specifications (blueprints). Specialized modeling tools are essential for serious work, involving more than a handful of diagrams.

A UML model may comprise many diagrams of different types as well as detailed documentation for each element. All of the information about a model is held in a common repository. This illustrates one of the key advantages of using a purpose-built UML tool, which facilitates reuse of work and ensures consistency.

UML facilitates the design of all types of software systems before coding. It is now one of the essential skills of health informatics.

4.5 Business Process Modeling Notation (BPMN)

Clinical processes are difficult to model with existing tools, in part because they are inherently complex, but more importantly because each patient is different and each clinician may adopt a variety of different paths, depending on the specific clinical situation of the individual patient. This requires business process specifications that take account of the full range of choices that clinicians have open at any one point in time.

BPMN (Business Process Modeling Notation) is a notation for documenting complex business processes (White and Miers 2008). The BPMN notation is understandable by end users and it is also capable of including the technical detail needed to specify messages involved in web services delivery and the generation of XML-based Business Process Execution Language (BPEL). BPMN is now part of the Object Management Group (OMG), which is also responsible for UML.

BPMN is a standard for business process modeling with a notation that is similar to that used in UML Activity Diagrams. Some commentators regard the future of BPMN as a specialized "front end" to UML.

BPMN has several advantages over standard UML activity diagrams:

- Shows explicitly who does what, where and in what sequence using the Pool and Swim-lane notation, distinguishing between messages which flow between actors from the flow of activities by a single actor or team
- Explicitly shows trigger events, delays, and messages that precede or follow on from each activity
- Allow drill down of subprocesses into greater detail of activities and tasks
- Provide additional structured and/or free text documentation for any element
- Executable output, using Business Process Execution Language (BPEL), an XML-based language, which has industry support from Microsoft, IBM, etc.

4.5.1 Activities

Activity is the generic term for Business Process, Process, Sub-Process, and Task. These have a hierarchical relationship. To use an analogy: a business process is a group of one or more trees; a process is a single tree; a subprocess is a branch (and may have further subbranches, sub-subbranches, and so on); task is a leaf, which is not subdivided further.

Business Process is the top of the Activity hierarchy in BPMN. It is defined as a set of activities that are performed within an organization or across organizations, shown on a Business Process Diagram (BPD).

Process is limited to the activities undertaken by one Participant (organization or role). Each Business Process may contain one or more Processes. A Process is an activity performed within an organization, and is depicted as a set of activities (Sub-Processes and Tasks) contained within a single Pool (see below Fig. 4.14).

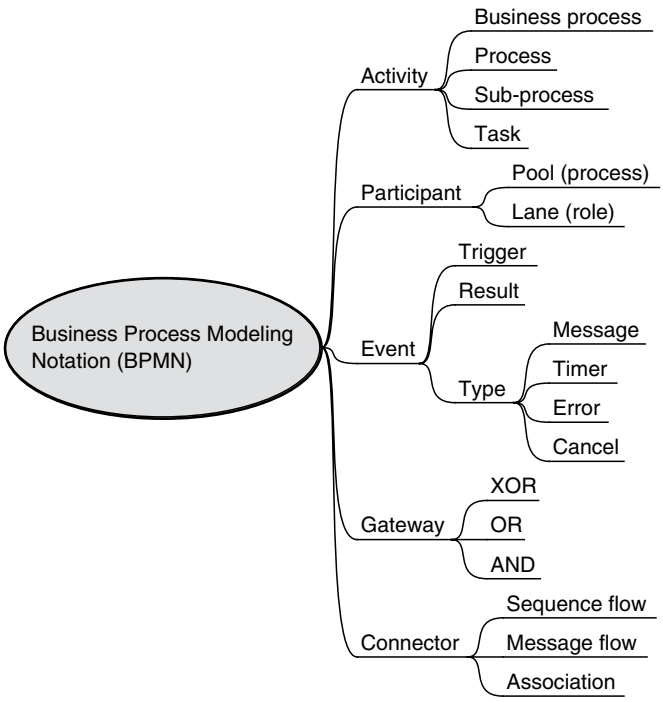


Fig. 4.14 Business Process Modeling Notation

Each Sub-Process may be expanded as a separate, linked diagram, showing its component Sub-Processes or Tasks. The facility to expand or consolidate Sub-Processes is a useful feature of BPMN.

A Task is an atomic activity, showing that the work is not broken down to a finer level of detail. Sub-Processes and Tasks are shown as rounded rectangles. Sub-Processes, which can be expanded, are shown with a "plus sign" at the bottom center of the icon.

Participants are each represented by a Pool, which may contain Lanes. Each Pool contains a single Process. A Pool may be subdivided into Lanes (like swim lanes in UML activity diagrams). Lanes may represent different roles within an organization. If a diagram contains a single Pool, the Pool boundaries need not be shown. A Pool is a container separating each Process from others and showing the Sequence Flow between activities.

Shown as a small circular icon, an Event is something that happens during the course of a business process that affects the flow. Events may represent triggers for activities to begin or their outcomes. Start, Intermediate, or End events are indicated by the thickness of the circle perimeter. An additional icon inside the circle shows the type of Trigger or Result (Message, Timer, Error, Cancel, Compensation, Rule, Link, Multiple, or Terminate).

A Gateway, shown as a square diamond, is used to control branching, forking, merging, and joining of paths. An icon inside the diamond shows the type of control (exclusive XOR, inclusive OR, parallel AND, or complex).

Connectors link the flow objects (Activity, Event, and Gateway). There are three types of Connectors:

- Sequence Flow (a solid line with arrow head) shows the order that activities are performed within a Process.
- Message Flow (a dotted line with arrow head) shows connections between Processes (crossing the boundary of a Pool).
- Association (dotted line, no arrow head) is used to associate information (such as Data Objects) and Annotations with Flow Objects.

4.5.2 Business Process Example

A complete business process from start to finish is shown in Fig. 4.15, which illustrates the traditional OP referral pattern for a patient suffering from a bowel problem.

The Pools and Lanes show clearly who does what in what order. The dotted lines represent movement of information (messages) or of information sources (e.g., the patient).

Each of the tasks shown could be represented as subprocesses and analyzed further in subsequent diagrams. Clinical care is essentially fractal and can usually be decomposed into smaller and more detailed subprocesses and tasks.

Trigger events are shown as circles, with an icon indicating the type of trigger – an envelope indicates a message and a clock indicates a time trigger, such as an appointment slot.

4.6 XML

XML (eXtensible Markup Language) is a universal format for encoding documents and structured data, which is used in interoperability between different applications. XML documents are independent of the applications that create or use them (Bos 1999).

XML is relatively modern, initially defined in 1998. XML Schema is even newer, published in 2001.

XML is derived from SGML (Standard Generalized Markup Language) which became a standard in 1986 (ISO 8879) at about the same time as EDIFACT (ISO 9735: 1987). SGML is an international standard for the definition of device-independent, system-independent methods of representing texts in electronic form. XML is strictly speaking a meta-language for formally describing a markup language. For example, XHTML is a version of HTML which is fully XML-compliant.

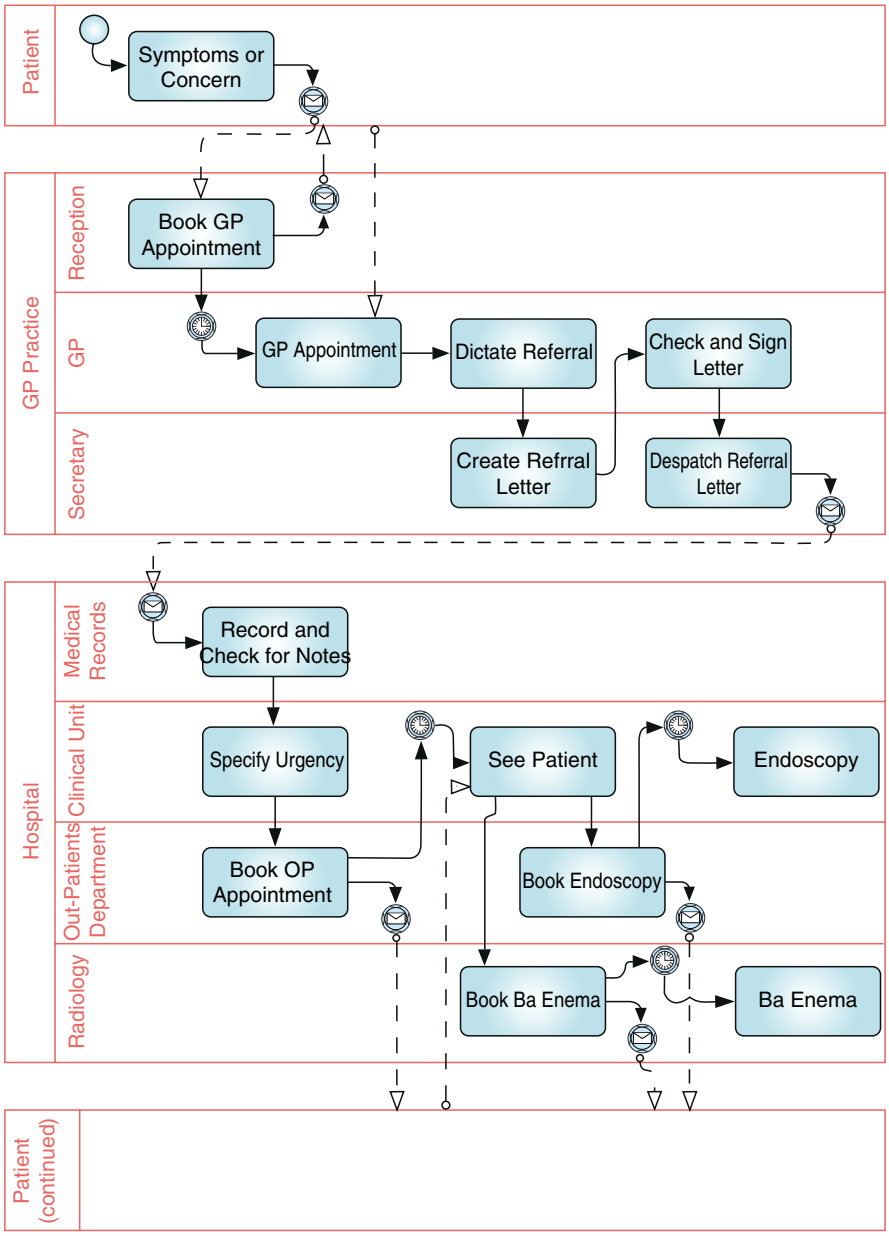


Fig. 4.15 Traditional OP referral for bowel problem

4.6.1 Markup

Markup is a term that covers any means of making explicit an interpretation of a text. In electronic documents, including all word processors, the system does this by inserting special coded instructions into the text, which are not normally seen on a printed copy. A markup language specifies:

- What markup is allowed
- What is required
- How markup is to be distinguished from text
- What the markup means

XML provides a method of doing the first three; additional documentation is required for the last. Such specifications may be extensive. The real work is in the definition of message structures.

4.6.2 Descriptive and Procedural Markup

Most word processors include markup instructions embedded within their text. However, XML differs in one vital respect from this sort of markup. XML is a descriptive markup scheme, while most schemes used in word processors and desktop publishing are procedural. A descriptive scheme simply says what something is (for instance, a heading), while a procedural scheme says what to do (for instance, print in 18 point Ariel font, bold, left-hand justified). XML specifies how to render information using style sheets, coded using related style sheet languages called CSS (Cascading Style Sheets) and XSL (Extensible Stylesheet Language).

This separation of description (providing names for parts of a document) from procedure is the secret to platform-independence and universality, which are XML's greatest strengths. A text marked up using a descriptive scheme can be processed in different ways by different pieces of software. Procedural markup can only be used in one way.

4.6.3 XML Files

XML files are text files that people should not have to read, but they are human-readable when the need arises, for example, when debugging applications. XML is verbose by design.

XML files are nearly always larger than comparable binary formats. The rules for XML files are strict, and forbid applications from trying to second-guess an error. These were conscious decisions by the designers of XML.

XML documents can include images and multimedia objects such as video and sound. The data may also include metadata – information about itself – that do not appear on the printed page.

XML and HTML are closely related. Indeed, XHTML is a version of HTML which is fully XML-compliant. Like HTML, XML makes use of tags (words bracketed by "<" and ">") and attributes (of the form name="value"). The key difference is HTML specifies what each tag and attribute means, and how the text between them will look on a browser, but XML uses the tags only to delimit pieces of data, and leaves the interpretation of the data to the application that reads them.

An XML document is said to be well-formed if it complies with a concise set of well-defined rules. One of the most important is that all nonempty elements are delimited by both a start-tag (e.g., <tag> and a matching end-tag </tag>. Element names are case-sensitive.

4.6.4 XML Schema

The structure of an XML document is specified in a schema, which is also written in XML (van der Vlist 2001). The schema defines the structure of a type of document that is common to all documents of that type. It identifies the tags (elements) and the relationship among the data elements. This means that any document of a known type can be processed in a uniform way, including checks that all of the elements required are present and in the correct order.

The development of schemas is the central analysis and design task of working with XML. Schemas can be prepared for use with existing text and databases. Tighter rules may be specified where uniformity of document structure is desirable. The schema makes the rules explicit. XML allows data to be tagged with literally any information that may be considered useful.

Schema processing tools are used to validate XML messages or other XML documents using one or more schemas. Schema validation is applied to elements within a well-formed XML document.

Three schema languages in widespread use are W3C's Schema Definition Language (XSD), RELAX NG, and Schematron.

In XML a number of words such as *element* and *attribute* are used in specific technical ways, as described below.

4.6.5 XML Element

In XML, documents are made up of elements. Each element is tagged using a start-tag and an end-tag. For example, a diagnosis element in a text might be tagged as follows:

<diagnosis> Diabetes mellitus </diagnosis>.

Note that in HTML all of the elements are predefined; in HTML it is not possible to have a tag such as <diagnosis>.

A start-tag takes the form <name> while the end-tag takes an identical form except that the opening angle bracket is followed by a slash character </name>.

Every XML document has a hierarchical tree-structure. Elements may be nested (embedded) within elements of a different type. For example, the line of a poem may be embedded within a stanza, which is embedded within the poem, which is embedded within an anthology.

An XML element definition is a detailed specification of the form and content of an XML element, which includes the name (a generic identifier in XML terminology) for the element. A simple type contains no sub-element definitions. A complex type may include sub-elements.

4.6.6 XML Attribute

Attributes are used in XML to add information to the start-tag of an element to describe some aspect of a specific element occurrence. Attribute values are written in quotes and are separated from the attribute name by an equals sign. For example, a hypertext link in HTML is shown as <A HREF = "URL">, where URL is the address of the uniform resource location (URL). Any number of attribute–value pairs may be defined for any element.

4.6.7 Entity

XML entities are named bodies of data, which can be referenced by an entity reference. Entity references always begin with "&" and end with ";". A small number of entities are used to represent single characters that have special meanings in XML, such as < (<), > (>) and & (&). Numeric character references can be used to represent Unicode characters. For example, © is used to represent the © symbol. Other entities can be defined.

4.6.8 Namespace

A Namespace mechanism is provided to eliminate confusion when combining formats. This is used in XML schema to combine two schemas, to produce a third which covers a merged document structure.

4.6.9 The XML Family

XML is a family of technologies.

- CSS and XSL are style sheet languages.
- XPath provides a way to refer to individual parts of an XML document.

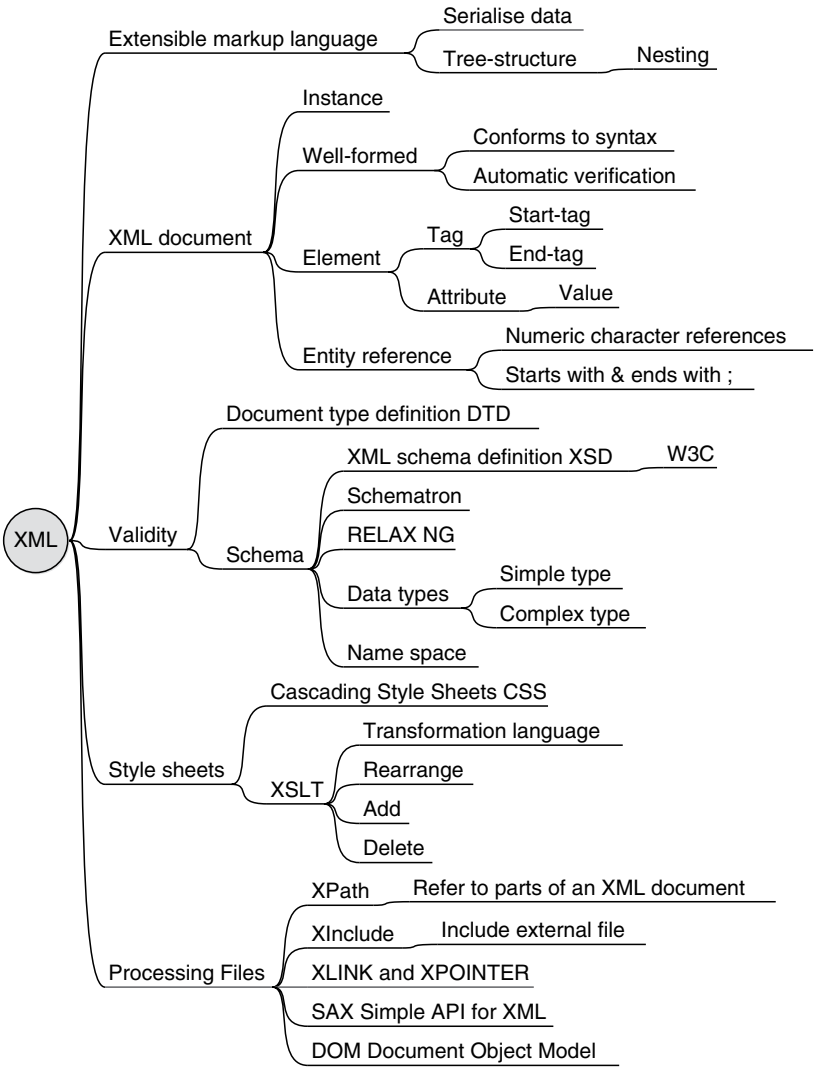


Fig. 4.16 Aspects of XML

- XSLT is a transformation language for rearranging, adding, and deleting tags and attributes.
- DOM is a standard set of function calls for manipulating XML (and HTML) files from a programming language.
- SAX is a simple API for XML.

A short introduction such as this can do no more than scratch the surface of XML (Fig. 4.16).