UML Tutorial

Table of Contents

1. Introduction

1.1. Problem

The Unified Modeling Language (UML) is a graphical notation for communicating information about a system. Its primary use is modeling software intensive systems to be built. However, it can be used to model other things. [Booch, Rumbaugh, Jacobson, 1999] say that it can also be used in settings other than software including "workflow in the legal system."

1.2. Proposed Solution

I provide a tutorial here. Then, we show you how it can be used to model contract formation as an offer-acceptance problem. Lastly, we show the integration of this model with the legal activities in litigation concerning this contract.

1.3. Assumptions

NA

1.4. Dependencies

This is based on the UML as described in [Booch, Rumbaugh, Jacobsen, 1999] and implemented in Rational Rose 98i (copyright 1999).

2. Specification

2.1. Association Diagrams

The fundamental diagram is the association diagram. This shows the relationship between types of things (or classes) in the system.

There are two important arrows. A regular arrow shows that the things are connected by an "association." This means that one can get from one thing to another because there is a relationship. The offeree and offeror would both be connected to an offer. The plaintiff and defendant would both be connected to a lawsuit. The arrow shows the direction of navigability. For example, if there was an arrow from lawsuit to plaintiff, this means our software could figure out for a given lawsuit, who is the plaintiff, and then proceed to the plaintiff record. However, it could not go easily from a given plaintiff's name to the lawsuits they may have filed.

If we want to represent navigability in both directions, a single line is used. For example, the UML might show a line from lawsuit to plaintiff with no arrow heads. This means that if one is looking at the plaintiff record, one could get a list of lawsuit that they may have filed and go to those objects to find information about them. If one were looking at the lawsuit, one could easily find information on the plaintiff.

The UML is not language specific. If one was using an object-oriented language like JAVA or C++, the classes are implemented as classes. However, they could be structure records in earlier languages.

If one were implementing this via paper records, the plaintiff record and lawsuit would be a form in which various information would be typed or written.

And lastly, if we are using XML, lawsuit and plaintiff would represent XML schema's.

The UML doesn't say anything about how this navigability is implemented. If one were using C, one would probably do this with a pointer from the lawsuit entity to the plaintiff entity:

```
struct Plaintiff {
  char LastName[100];
  char FirstName[100];
  ...
}
...
struct LawSuit {
  Plaintiff *plaintiff;
  ID caseno;
  ....
}
```

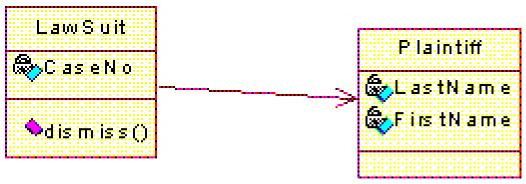
In XML, the "plaintiff" field of the DTD for PLAINTIFF would have an IDREF for Lawsuit. In a paper form, we would have a file number, or some identifying sequence by which a clerk could go from one form to look up another form or folder in the paper archives.

```
ATTLIST LawSuit Plaintiff IDREF
```

Information that is stored with an object is called an Attribute. It would be a field in the structure declaration in C, Java or similar languages. In XML, it could correspond to either an attribute or a tag within this one.

In a paper form, it would be a field in which one typed in the information.

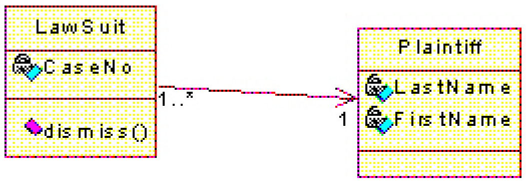
In the UML diagram, these are the entries in the second field. In our example, plaintiff's have two attributes: last name and first name. Lawsuits have one: CaseNo.



The third field would be used for operations. These correspond to actions that can be taken with respect to the class or object. In object-oriented programming, these are methods of a class. They don't make sense with respect to a static entity like a form or piece of XML. In the example, **dismiss()** is an operation. The association diagram does NOT specify what can activate the operation. For example, we could not tell from an association diagram whether judges, court clerks or the plaintiff herself could dismiss the lawsuit. (Collaboration diagrams provide this information-to be discussed later.)

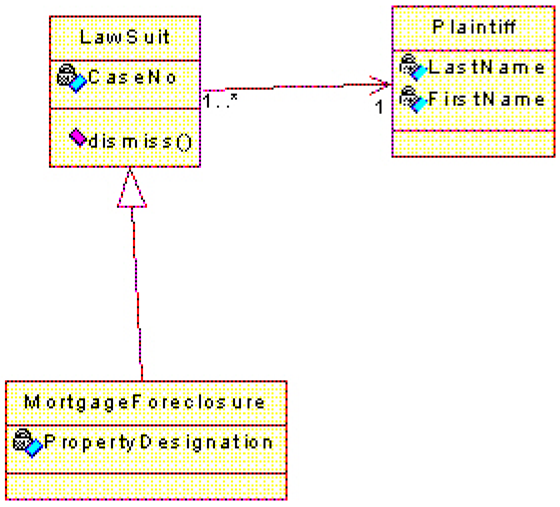
Multiplicities indicate how many of one type of object could be associated with another. They are represented as **1**, *****, **0..1**, **1..*** or **0..***. These mean, respectively: exactly one, any number, zero or one, at least one, or an arbitrary number.

The example below shows the following: Each plaintiff would have at least one lawsuit, but a lawsuit has precisely one plaintiff.



An arrow with a triangle (not filled in) represents "generalization." This corresponds to inheritance in object-oriented programming.

For example, **MortgageForeclosure** is a type of lawsuit. Thus, we draw an open arrow from **MortgageForeclosure** to **Lawsuit**. This means that it has a dismiss operator like any **Lawsuit**. One could thus also deduce that a **MortgageDisclosure** is connected to a **Plaintiff**.



2.2. Collaboration and Sequence Diagrams

Collaboration diagrams show a sequence of activities between specific items. This is where one shows the messages or actions that can be performed on the classes--and which other classes or users perform them.

Also, collaboration diagrams show an example sequence between specific actions.

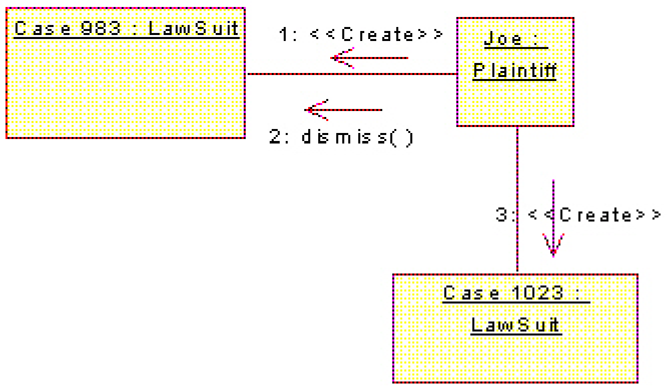
Each occurrence of a specific class may be labled with a name. Thus we might write **Case1:Lawsuit** and **Case2:Lawsuit**.

It would be incorrect to have two Lawsuits on an Association Diagram. It would be correct and quite reasonable to have two Lawsuits on a Collaboration diagram.

Often, when we describe the operation of a software system or a business system with people, we give an example of some actions that might occur and the order in which they may occur. A collaboration diagram would show the activities in that action. One would probably have only one association diagram showing the possible relationships between classes. One would then have several collaboration diagrams that show the ways in which they might interact.

Along the lines showing association between the objects, one indicates the messages by arrows.

The collaboration diagram below shows the following scenario: Plaintiff Joe files a Lawsuit. Then, he dismisses it and files another. Note the use of "<<create>>" This is a "stereotype" message defined in the UML. There are others. All are indicated by the double less than or double greater than. In the UML, we could define our own stereotypes such as <<file>> or <<MakeMotion>>.



2.3. Use Cases

The last type of diagram I discuss in this tutorial is a Use Case. These are useful for verifying that all requirements of the system pare included. In systems analysis, it is important to interview all "users" and "stakeholders" of the system to be built. The use case diagrams help ensure that one has identified all of these.

These represent the participants and external actors in the project in relationship with the system.

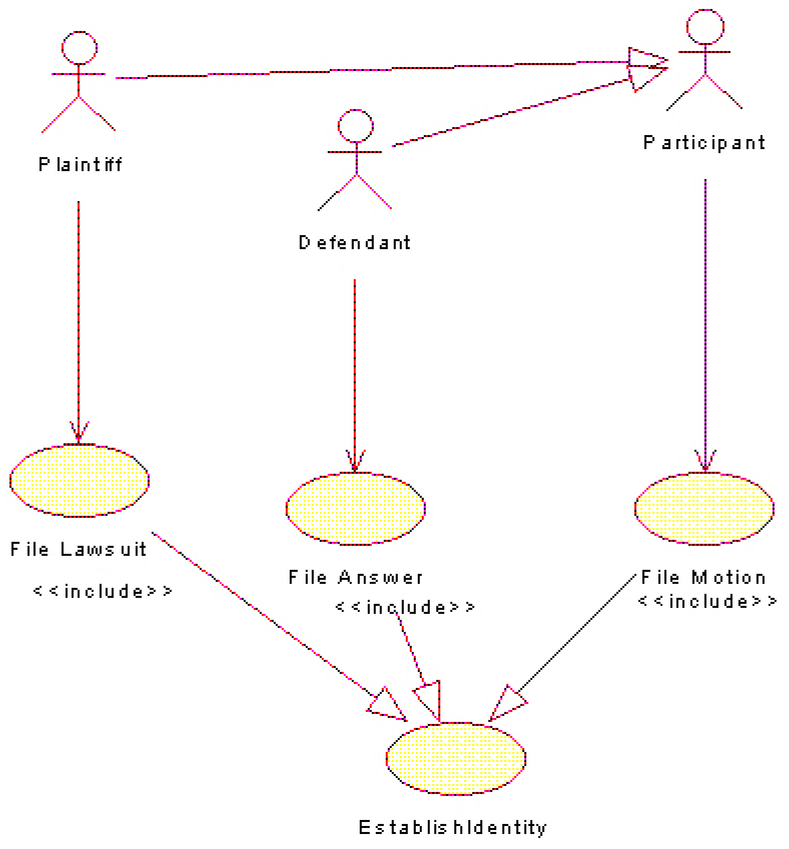
In LegalXML, the external actors would usually represent humans such as litigants, lawyers, judges and clerks. However, they could represent software which we did not control such as the Case Management System in a court. In short, they represent any thing outside our system sending messages into it (or receiving information).

These are usually drawn on diagrams by a stick figure representative of a person.

Each person in the use case diagram corresponds to a role such as defendant, plaintiff, judge, clerk. (Of course, a given individual might serve in several of these roles. A single person might be a defendant in one civil case and a plaintiff in another. A court clerk might be sued when she inadvertently collides into someone when on vacation. We consider each stereotypical role as a separate actor, even if occasionally, a single person might fill both roles.

They perform sequences of related actions with the system, such as file a lawsuit or file a motion. Some actions have common features with others. For example, there might be certain identity checks and receiving an acknowledgement that would be in common with all electronic filing activities. We use the extends stereotype to show these.

We can use the generalization arrow (with a triangle not filled in as a head) between actors in a use case diagram. For example, both plaintiffs and defendants are "participants" who can file motions. However, only a plaintiff engages in the activity of filing a lawsuit. This is represented on the diagram below:



There are several other types of diagrams in the UML. The statechart will be explained in the example below. The others are not immediately relevant to the activities of the Legal XML group. They include component diagrams, deployment diagrams and activity diagrams.

3. A Realistic Example

To illustrate these ideas, we show a more complicated example showing the formation of a contract.

The association diagram shows the relationship between clauses in a contract, the participants who perform actions and the court.

XMLServer creates legal relations including contracts. It represents a somewhat abstract computer entity. When it receives an offer from a **Participant** Actor, a **ContractRegister** object is created to track the relationship between the parties. The **ContractRegister** has a connection to both parties to the contract.

The **ContractRegister** object will maintain links to two **Participant Objects**, each of which will have **name** and **address** attributes. They represent the **Participant Actor** in the computer database.

Each contract will also have an association to a **court** where cases are to be tried. This corresponds to the parties agreeing where the litigation should occur. In the next example, we show more involvement between the court and the contract.

Each contract will be associated with a series of clauses defining the responsibilities of each side. Each clause represents a relationship between two of the participants. Each clause may be dependent upon specific events. These may be specific clauses or particular events. For example, payment may not be due until the other party meets their part of the bargain.

The following specific types of clauses are recognized:

1. A payment of money from one party to another of a specified amount

2. A transfer of something from one party to another by a specified party

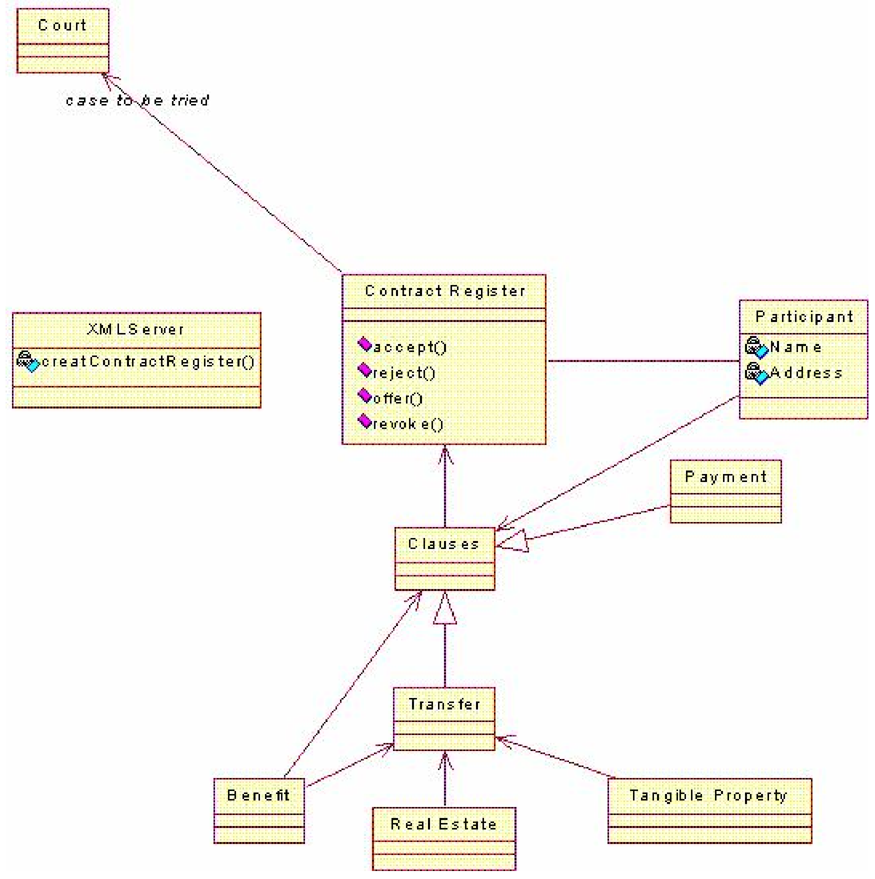
   The something might be:

   1. real property (land or the buildings attached to land)

   2. tangible property

   3. the right to receive the benefit of some other clause in some other contract. This would be used in subletting an apartment. The person leasing the apartment would sell the rights to receive the benefit of living in the apartment under the lease contract.

This is shown by the Contract Association Diagram. Observe that **Transfer** and **Payment** are types of clauses. A **Transfer** has to indicate what is being transferred, e. g. **Real Estate** or **Tangible Property** (Chattel). Thus, the arrow between **Payment** and **Clause** and between **Transfer** and **Clause** are represented by the open triangle.

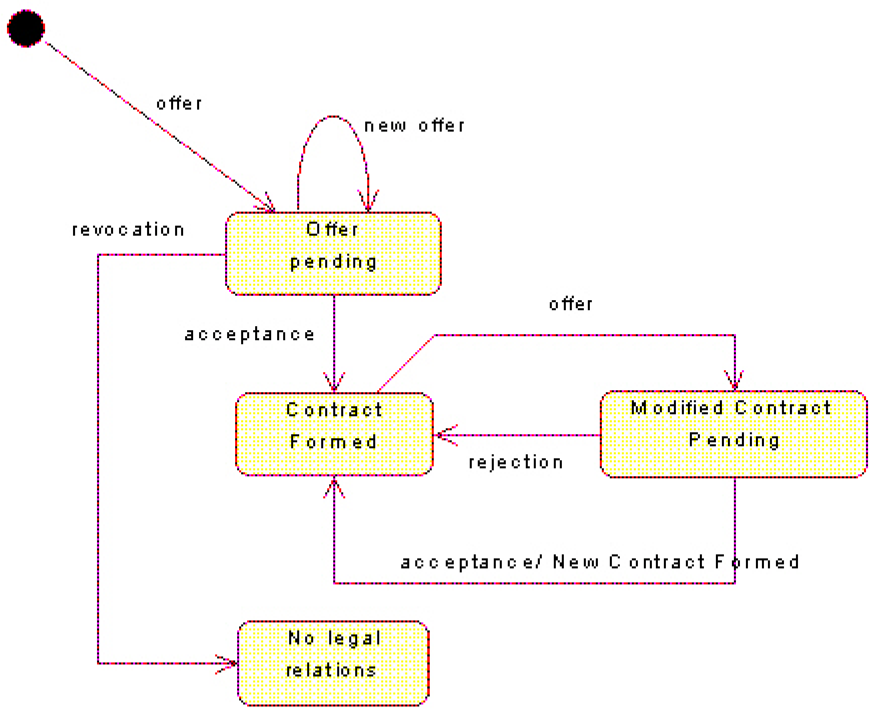


A **Benefit** represents assigning the rights to a clause in one contract. This is a relationship between two clauses.

The arrow between **Contract Register** and **Court** represents the parties agreeing in the contract where the litigation should occur.

I now explain the state diagram. A state diagram is used when the result of one action depends upon a previous history. Sending a proposed clause is handled differently when done at the start (an offer) than when an offer is on the table. Similarly, an offer sent after a contract has been agreed to means that we are in the special state of a proposal to modify pending.

A state chart would be useful for keeping track of what motions would be permissible at various points in pretrial procedure. (See the next example.) This example was inspired by (Gardner 1987).

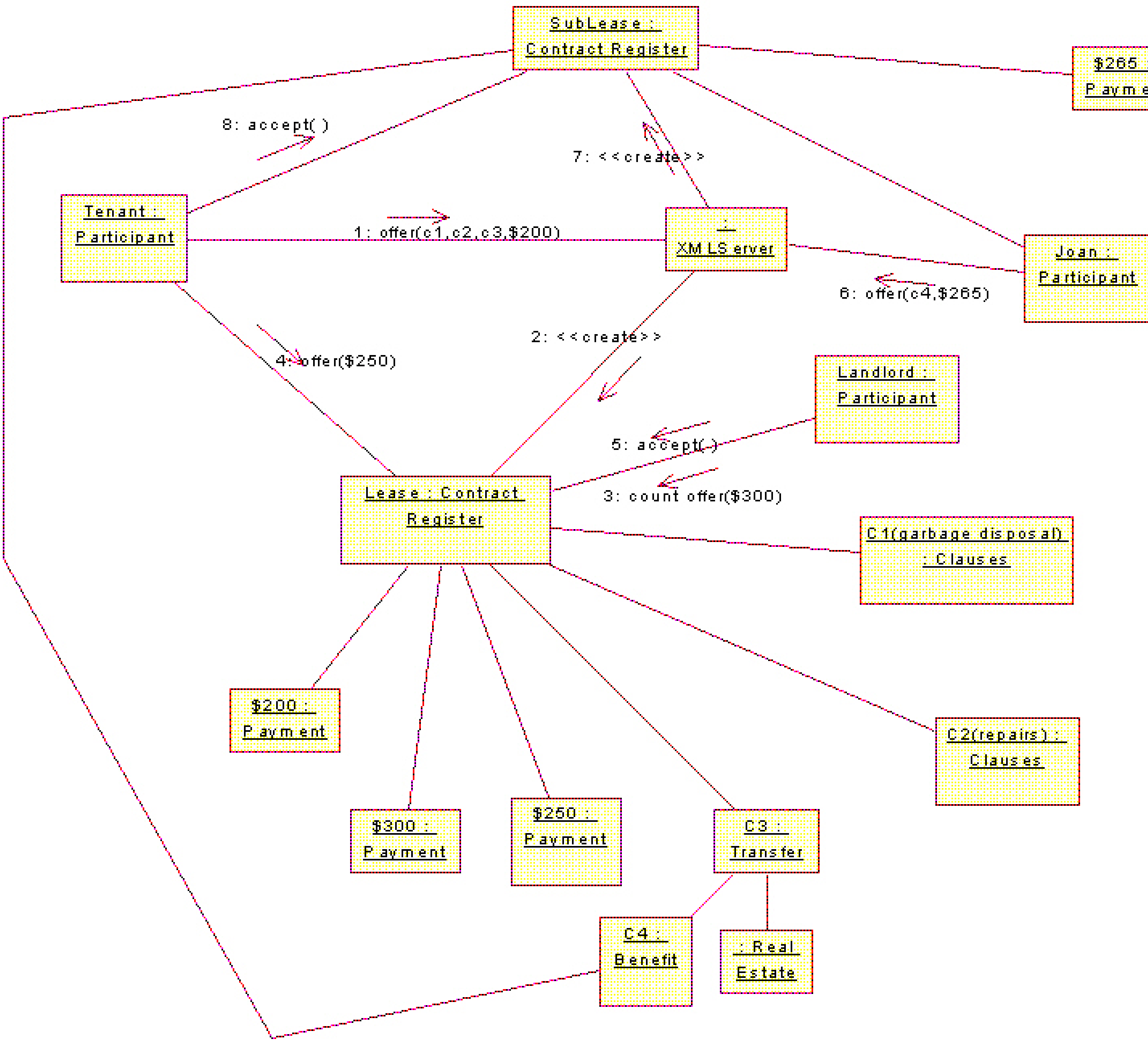


The **XMLServer** manages legal relations including contracts. When it receives an offer from a **Participant** Actor, a **ContractRegister** object is created and goes into the state **offer pending**. This is indicated by the line from the big dot in the upper left (the symbol for a start state) to the **offer pending** state. The other **Participant** can send an **Acceptance**, in which case the **ContractRegister** goes to the state **Contract Formed**. However, if before this, either **Participant** sends a **rejection** message, the **ContractRegister** goes to the state **no legal relations**.

Either can send a new **offer** while the **ContractRegister** is in the **OfferPending** state. In this case, the **ContractRegister** is still in the **ContractFormed** state but with a new set of clauses.

After the **ContractRegister** is in the **ContractFormed** state, then if either **party** sends an **offer** to the **ContractRegister**, its state will be **Modified Contract Pending**. If the other party sends an acceptance, the **ContractRegister** will go back to **ContractFormed** but with a new set of clauses. A **Rejection** will put it back to **ContractFormed** but with the old set of clauses.

The collaboration diagram shows an example of a negotiation between a landlord and a potential tenant for a lease. Once the lease is performed, the original tenant subleases it to Joan after accepting an offer for $265.00.



1. A potential tenant offers a landlord $200.00 a month for an apartment.

2. The tenant specifies an additional clause: the landlord should pay for garbage disposal. (Of course, in practice, there would many additional clauses.)

3. The landlord overs a counter proposal: the rent should be $300.00

4. The tenant offers a new counter proposal: the rent should be $250.00

5. The landlord accepts this counter proposal--forming a contract.

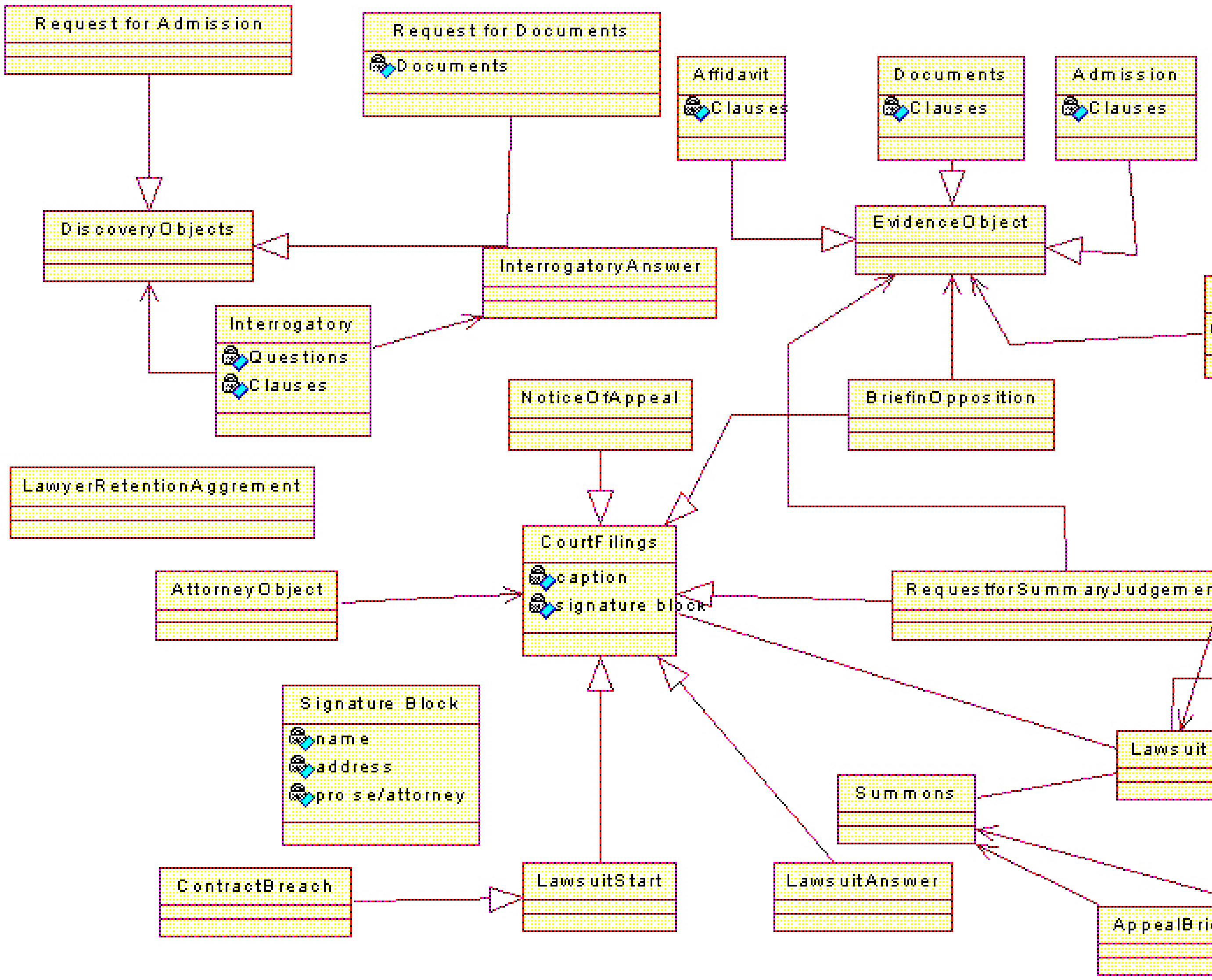
6. Now the tenant receives an offer from Joan to sublease the apartment at $265.00 per month. The tenant accepts this.

Notice how the numbers next to the messages on the collaboration diagram indicate the order in which these events occur.

4. A Second Example with Litigation

In this example, we expand the UML to show the litigation that might occur in a contract.

Let's begin with the classes that occur in handling some of the procedures and motions in civil litigation. This is an association diagram or class diagram.



Many of the items in this class diagram are specializations of Court Filing. These are shown by the open-headed arrows. For example, **Lawsuit Start** is a type of Court Filing. We assume that there is a special form for a **LawSuit Start** concerning a **ContractBreach**.

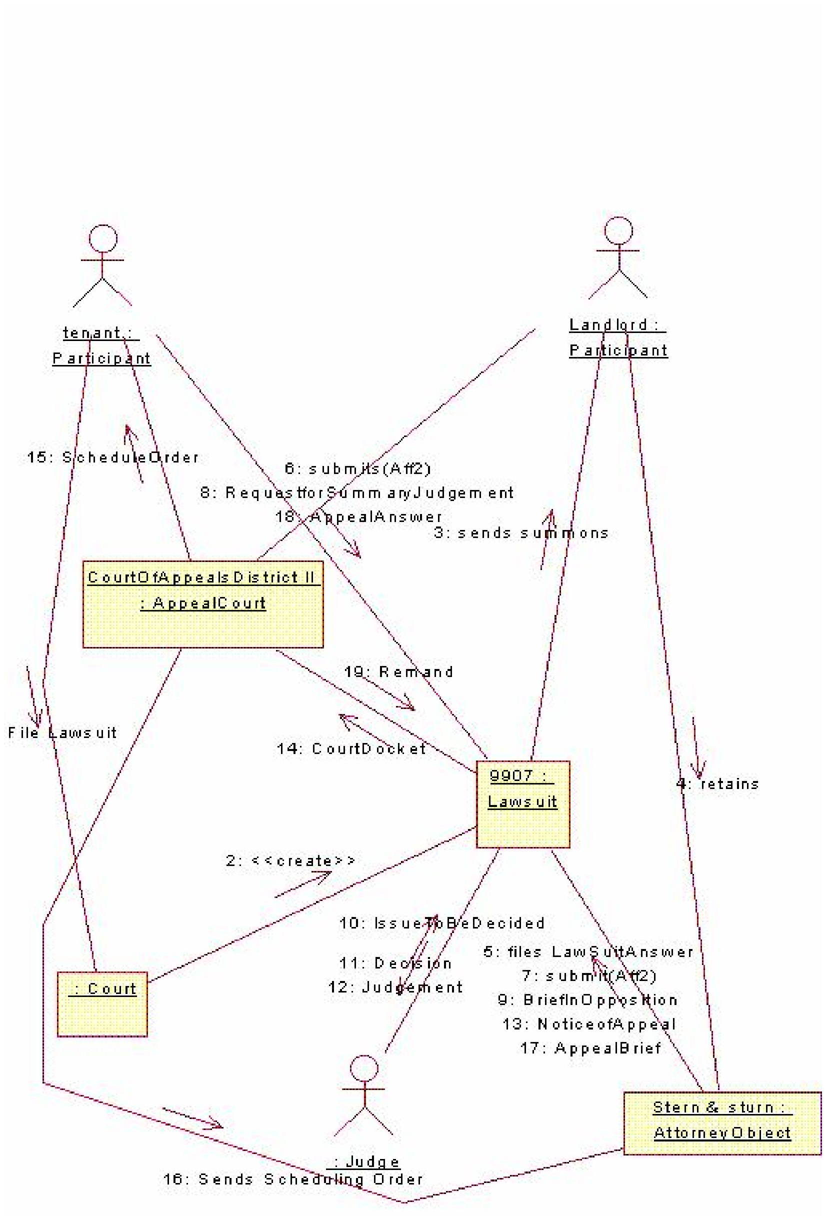
We assume the Case Management System keeps a record or object to track all the matters concerning the **Lawsuit**. Note the line from **Lawsuit** to **CourtFiling** indicating the relationship between these.

In pretrial procedure, the parties engage in discovery. The materials that one side could request from the other parties are the different types of **DiscoveryObjects**. As the information is collected, **Evid-enceObject**'s are accumulated.

When one party submits a **RequestforSummaryJudgement**, they state they have already proved their case. They refer to the items already collected as evidence plus affidavits from their witnesses. This is shown on the UML by the line from **RequestforSummaryJudgement** to **EvidenceObject**. Also, when the other party replies to this message, they will also refer to **EvidenceObjects**'s.

Associated with each **Court** is the **AppealCourt**. When a lawsuit is submitted to **AppealCourt**, the Appeals Court generates a **SchedulingOrder** which indicates when briefs and answering briefs must be filed.

We model a scenario as a collaboration diagram.



The tenant feels that the landlord has not made necessary repairs, i.e., there has been a breach of contract with respect to Clause Two. The tenant is pro se (without an attorney.) The lawsuit files a **Lawsuit Start**. This is sent to the **Court** which creates the **Lawsuit**. I believe it is more convenient to model subsequent actions as going to that object--even though we know that currently, documents are filed with the Clerk of the Court.

The **Lawsuit** sends a **summons** to the landlord. the **landlord** retains the attorneys **Stern and Sturn** who files a **LawsuitAnswer**.

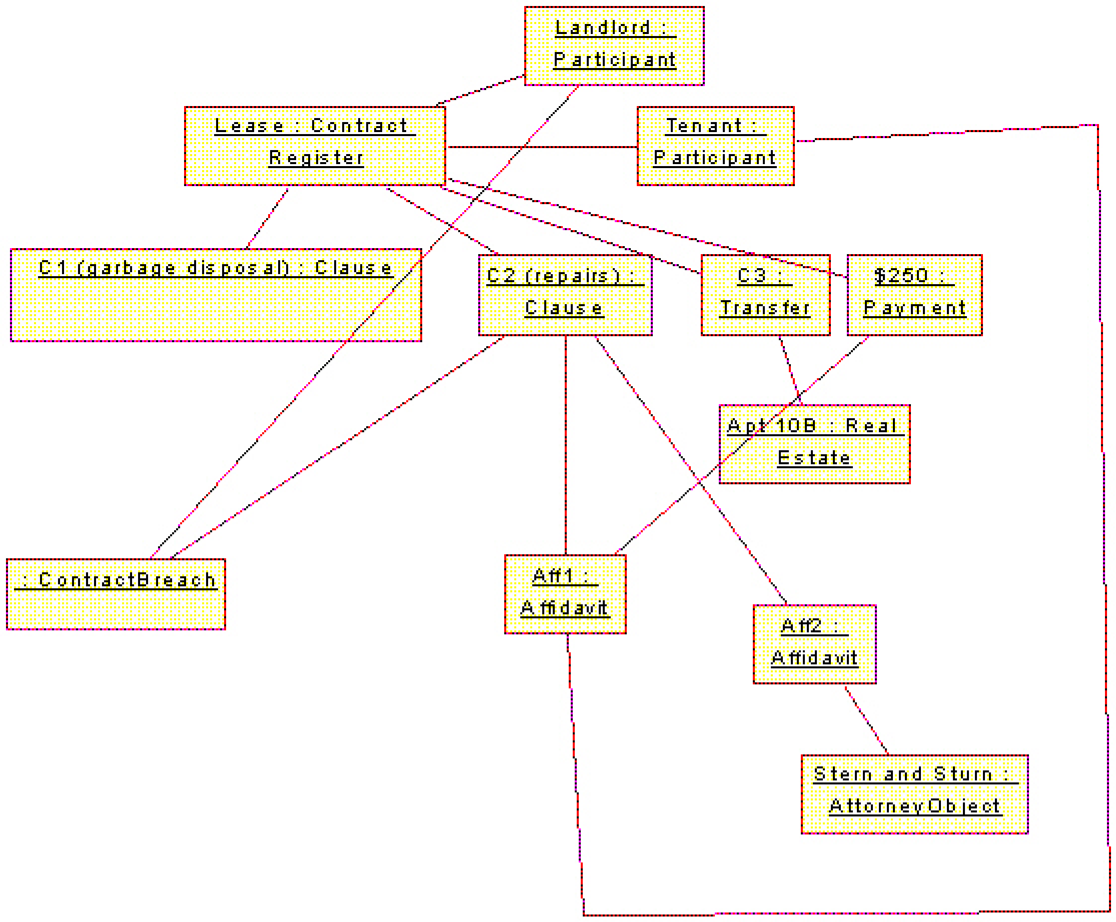
The tenant submits an **Affidavit** describing the leaky ceiling and also stating the rent was paid. The landlord's attorney submits an **Affidavit** saying that the leaky ceiling was in fact repaired.

The tenant sends in **RequestForSummary** judgement pointing to his affidavits and Stern and Sturn files a **BriefinOpposition**, pointing to their affidavit.

The Judge rules in favor of the tenant and Stern and Sturn goes through the appeal process and the tenant sends an answer. The Appeals Court eventually remands the case back to the trial court.

These events can also be shown on a UML Sequence Diagram. This makes it easier to visualize the order in which things happen.

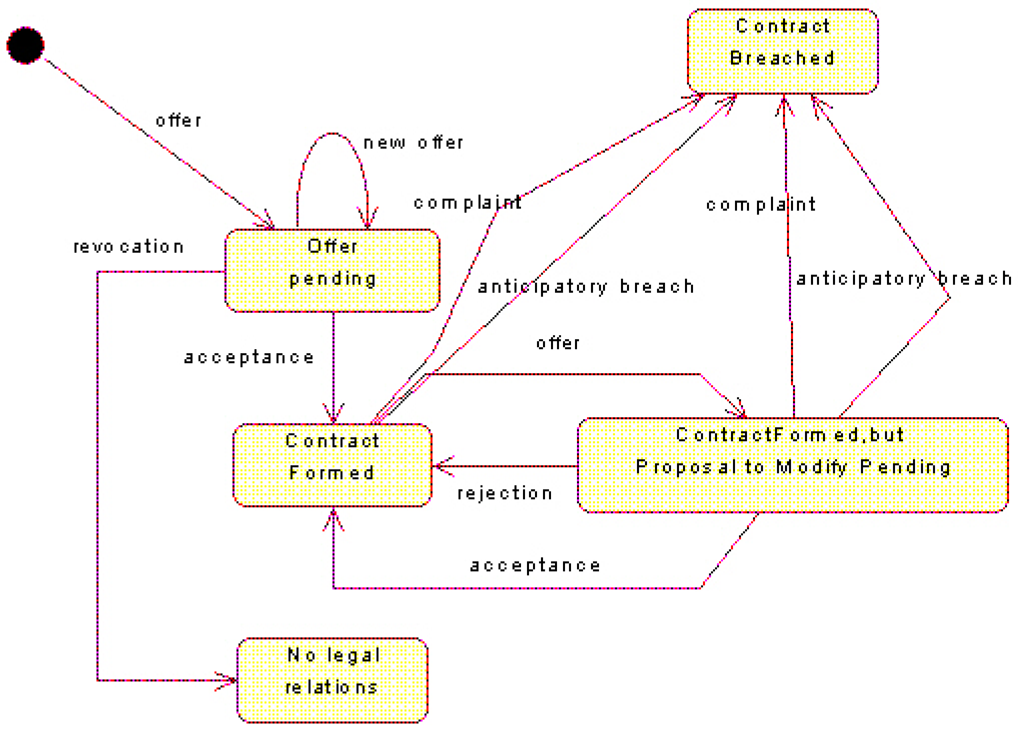
However, we need another diagram to make clear the relationship between the litigation and the contract. We thus show an expanded version of the earlier collaboration diagram for contracts with links to the affidavits.



It shows how the first affidavit (from the tenant) concerns clauses two and the payment clause. The second affidavit (from the landlord's attorney) only concerns clause two.

In complicated situations, the same class or object may appear on several diagrams. Each diagram illustrates different aspects of the class's relationship to the other entities in our system.

We also have a new state diagram for our contract, showing how the contract can go from a formed contract to one in breach. (There are two types of breach shown. In one, one party simply observes that another party hasn't completed their side of the bargain. An **anticipatory breach** refers to the situation where one party announces that they will not or cannot do their part of the bargain.



References

[umlguide] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley. Copyright © 1999.

[gardner] Anne von der Lieth Gardner. *An Artificial Intelligence Approach to Legal Reasoning*. Bradford Book. Cambridge Massachusetts. Copyright © 1987.