The UML aspect of RUP stretches across the disciplines. A coherent set of models is essential for a team to successfully collaborate, but RUP itself does not provide a clear picture. This white paper fills the gap and does not hesitate to criticize the method. It has been updated for RUP version 7, also including the Service Model and the new business models.

Pitfalls using UML in RUP

Did you ever follow the rules of the Rational Unified Process (RUP) in using UML? “Well, I tried!” is a commonly heard answer to that question. I’ve tried it myself a number of times and yes, I failed. Following RUP blindly simply doesn’t work, but fortunately, you don’t have to say good-bye to the process altogether. Let me help you by pointing out some of the pitfalls and first of all, by providing an overview of the various models in RUP.

Models in RUP

Figure 1 shows an overview of the models specified by RUP. For each project, it has to be decided which of those models add sufficient value, although RUP recommends at least the Use Case Model and the Design Model.

Figure 1 is not a picture you will find in the official RUP product. The vast network of web pages does not give a clear overview of the relationships between the various UML models. Oh yes, there is a lot of information, but it is scattered around in artifact descriptions, guidelines etc. This creates a lot of confusion during projects: which UML diagrams are we going to draw and how are they interrelated?

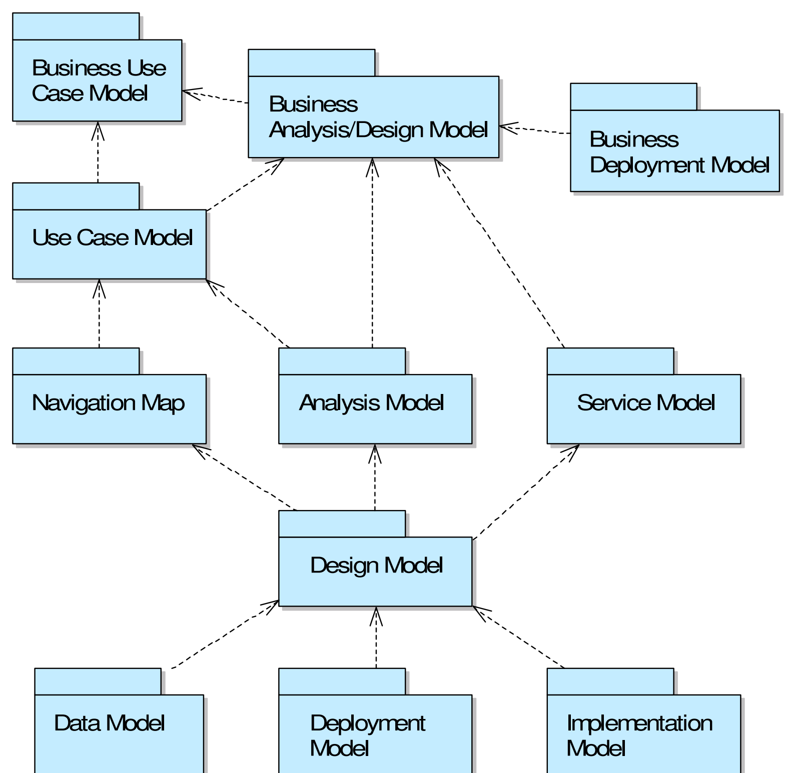

Figure 1. The RUP models and their dependencies.

RUP discerns four disciplines where UML is used:

- Business Modeling
- Requirements
- Analysis and Design
- Implementation

For each of these, I will summarize the view of RUP on UML modeling, the diagram types that can be used, their meaning and the relationships between elements in different models. Apart from that, I will give my personal opinion and suggestions about making practical decisions and dealing with the weak spots of RUP.

BUSINESS MODELING

Not all software development requires business modeling, but for administrative applications, you always need to know the business process to be supported. Large scale business modeling is sometimes performed outside the context of a particular IT project, so that RUP does not play a role. Too often, however, I am involved in software development projects where no proper business models exist. In those cases, if RUP is adopted, business modeling is also done RUP-wise.

RUP defines four business models:

- The Business Use Case Model, which describes the external interactions of the organization in terms of *business use cases*;
- The Business Analysis Model, which shows how the organization behaves internally to realize those interactions.
- The Business Design Model, which refines the Business Analysis Model with business details.
- The Business Deployment Model, which defines the mapping of business elements to geographical locations.

Business Use Case Model

Business Analysis Model

Business Design Model

Business Deployment Model

Figure 2. The business models.

The dependencies shown in Figure 2 should be interpreted as references from one model to another. The Business Analysis Model contains references to the business use cases, because it

provides an organization-internal realization of each business use case. The Business Design Model refers to the analysis elements it refines. The Business Deployment Model adds locality information to the business design elements.

Business Use Case Model

The business use cases specify the interaction between the organization, regarded as a black box, and the outside world. The outside world consists of *business actors*, mostly customers and suppliers. A business use case is described from the actor's viewpoint. For business actor 'Customer', for instance, there might be a use case 'Order a product' and for 'Supplier' a use case 'Supply materials'.

Table 1 shows the UML diagram types that can be used to create this model. Apart from UML, business use cases and business actors are further described in regular text.

Package diagram	Large models are divided in packages. A package diagram provides an overview of these packages and their interrelationships.
Use case diagram	Shows the business use cases, the relationships among them and their relationships with the business actors.
Activity diagram	The flow of events during a business use case.

Table 1. Possible UML diagram types in the Business Use Case Model.

Business Analysis/Design Model

The internal functioning of the organization, meant to realize the business use cases, is the subject of the Business Analysis Model and the Business Design Model. Here, the business processes are decomposed and the work flow is revealed. You also model the organizational structure and the flow of information, as far as relevant.

The Business Analysis Model usually evolves into the Business Design Model, making abstract business elements more concrete and adding details about the communication between business units. I will regard these models as one, from now on.

A variety of UML diagrams could be used to build this model.

Package diagram	An overview of the packages in the model.
Class diagram	The structure of the organization and the information. *Business workers* are active objects: employees, teams and information systems. Passive objects like documents and products are called *business entities.*
Activity diagram	Work flow model, focusing on activities. Business workers are diagram partitions, containing actions. Business entities are the input and output of these actions.
Interaction diagram	Work flow model, focusing on message exchange between business workers.
State machine diagram	The life cycle of a single business entity (states and state transitions).

Table 2. Possible UML diagram types in the Business Analysis/Design Model.

As an alternative, RUP presents the *domain model* as a kind of light-weight business model. The domain model only describes the business entities and their relationships and business rules.

Business Deployment Model

If the project's scope includes several business locations, you may benefit from modeling this in a business deployment model. The recommended representation is a deployment diagram, although according to UML, this kind of diagram is meant to represent IT system topologies only.

RUP does not define links with models in other disciplines, but I would certainly draw connections from services in the service model to the business nodes (the offices) where these services are hosted.

What is really needed for small scale business modeling

Although RUP considers business modeling an optional activity, I would always at least make a domain model. In such a model, I define the concepts from the user's reality and draw them in a class diagram. A business glossary adds a clear definition of each business class.

The domain model is an important terminology framework for the specification of use cases by the requirements discipline. Don't skip it!

One final remark about this example. I didn't show any attributes or operations, but that does not mean they are not there. On the contrary, some domain concepts can best be modeled as attributes or operations, although business *entities* don't have any operations: they are passive.

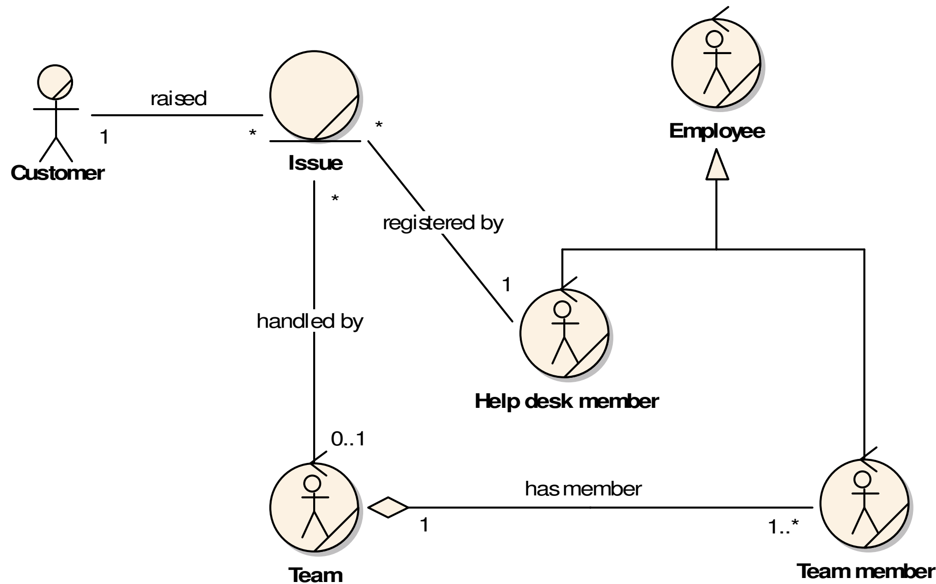

Figure 3. Class diagram showing business classes and their relationships.

More business analysis

If the application is supporting one or more business processes that are not well defined, you should do more business modeling. You could make a Business Use Case Model, but I suggest you just forget it. Instead, I replace it by a top-level activity diagram like Figure 4 in the Business

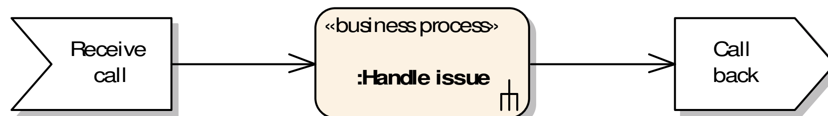

Figure 4. Top-level activity diagram.

Analysis Model. This diagram shows one UML action stereotyped as «business process», for each business process the application needs to support. By using the UML symbols *accept event action* and *send signal action*, I show how the processes communicate with the world outside the organization. The 'Handle issue' symbol covers the complete process of handling a call from a customer, including anything that needs to be done to solve the issue.

Note the rake symbol in the bottom right corner. It indicates that the business process is decomposed in a lower level activity diagram, as you can see in Figure 5. There is a partition for each business worker that participates in the process.

Apart from the extra work required when you create business use cases, there may be business processes that cannot be expressed at all as business use cases, in case they are entirely internal

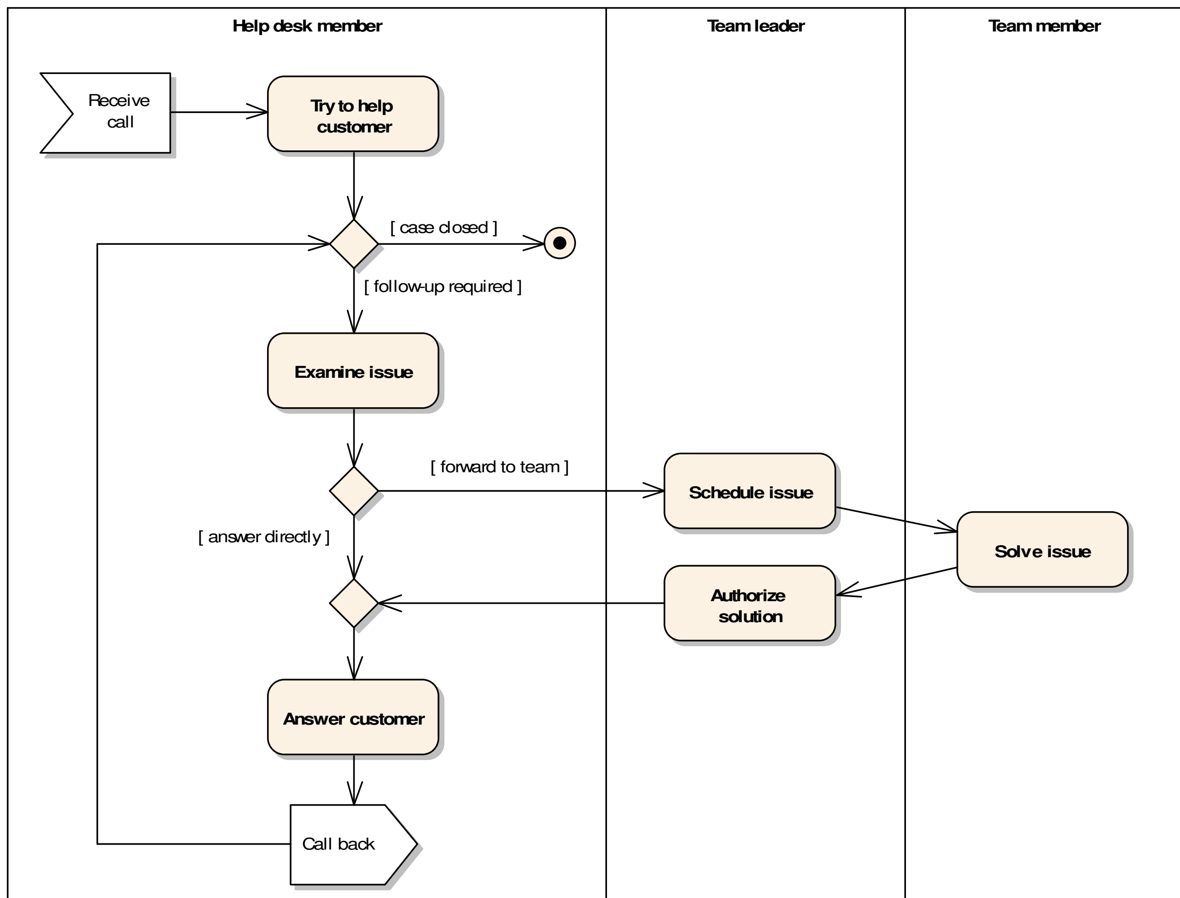

Figure 5. Second-level activity diagram.

to the organization: ‘Check administration’, ‘Update web site’. Another pitfall is, that business use cases should be named from the business actor’s viewpoint (‘Let the organization handle my issue’), while business processes are usually named from the organization’s viewpoint (‘Handle the customer’s issue’). This introduces a viewpoint shift when you compare the Business Use Case Model with the Business Analysis Model.

Now, what about the interaction diagrams? Well, I think that activity diagrams are sufficient and equiped to do all the process modeling. Interaction diagrams only add value if you define detailed business unit interfaces by using so-called business operations (a new artifact in RUP version 7).

Creating state machines

Business entities that have a life cycle of going through particular states, get a state machine. When an issue is registered, for example, it first waits to be examined by a help desk member. If this person decides to forward it to a team, it enters the ‘Forwarded’ state, etc.

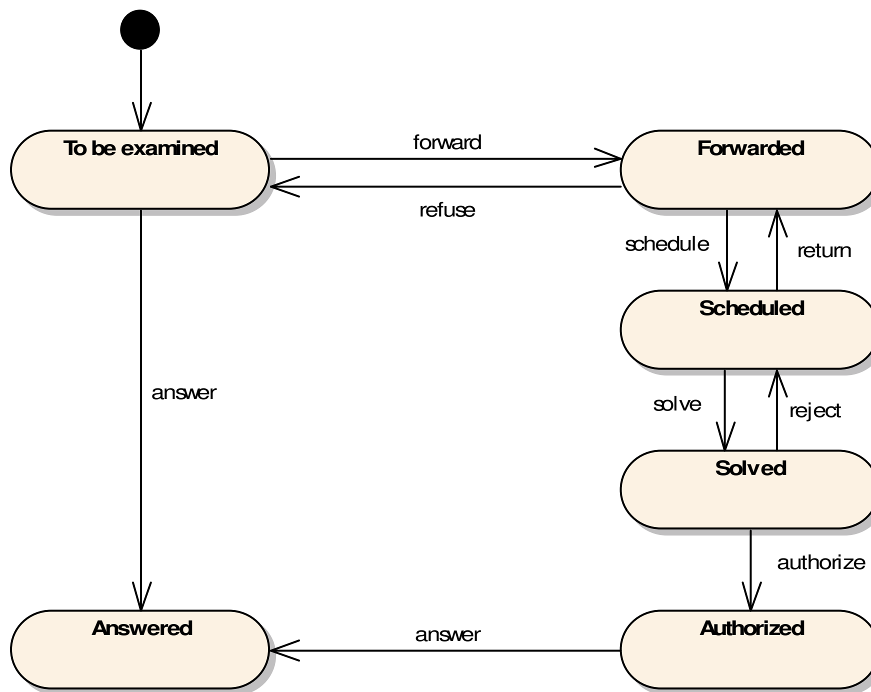

Figure 6. State machine diagram for business entity Issue.

A state machine is often another view on the work flow and therefore creates some redundancy in the Business Analysis Model. On the other hand, this second view brings a great opportunity to check and enhance the activity diagrams. Furthermore, the states can be very useful to refer to in the definition of business rules.

Business Modeling – Summary

Figure 7 shows the diagram types I use for business modeling, and their dependencies. They are not created in a particular order, instead, they evolve simultaneously.

- The class diagrams form the domain model. They should always be there and that's why they have got this blue color.
- The activity diagrams are used when the application supports a particular work flow. From these diagrams, you may refer to certain classes (business entities) in your class diagrams.
- State machine diagrams are used to model the life cycles of certain business entities. Consequently, they depend on the class diagrams. The transitions are triggered by actions of business workers, hence the dependency on the activity diagrams.

– Other diagram types are rare in business modeling, so I left them out from the figure. Large scale business modeling is usually not part of a RUP project.

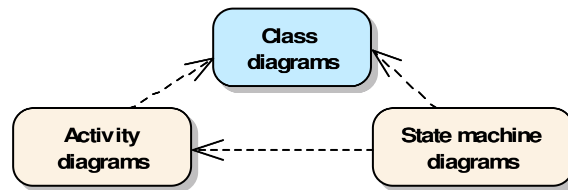

Figure 7. Typical set of diagram types for business modeling (blue means essential).

REQUIREMENTS

The men and women of the requirements discipline use the business models to work out the requirements for the application that should support that business. As far as UML is concerned, the "only" thing we have to do is to create a Use Case Model. This model depends on two business models, as shown in Figure 8. Although I ruthlessly unmasked the Business Use Case Model as a pitfall, earlier in this paper, the figure shows what RUP tells you.

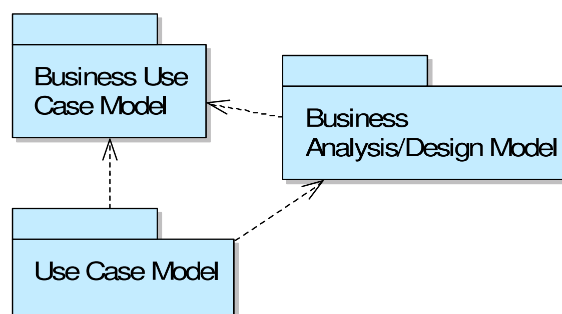

Figure 8. The requirements discipline adds the Use Case Model.

Use Case Model

This model is recommended for all RUP-projects. Basically, this model is shaped by iterating over three main activities:

First, you describe the *actors*: who actually work with the system, in terms of user roles.

Then, the *use cases* themselves are identified: what do the actors want to achieve by using the system? A use case diagram serves as an overview of these use cases (Figure 9).

Finally, for each use case, the interaction between the actor and the system is specified. This is a textual specification, which can be visualized in the form of an activity diagram (Figure 10).

Actors

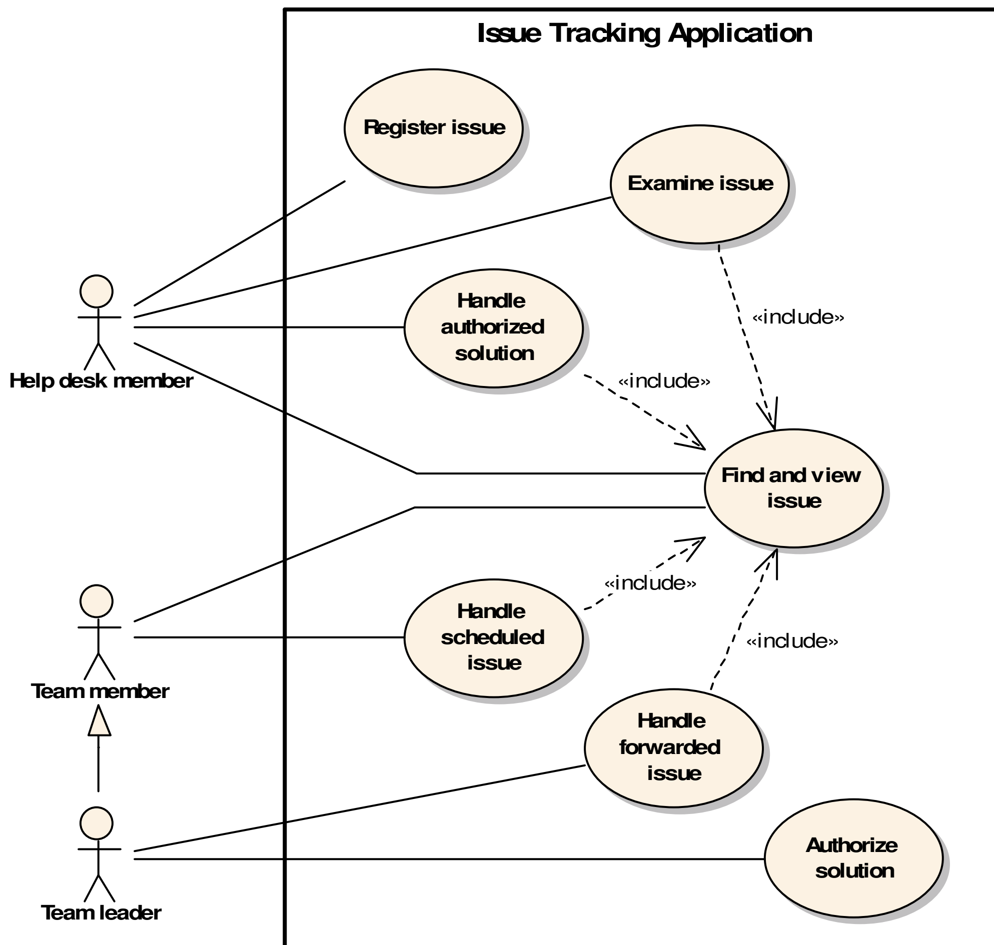

Figure 9. The use case diagram in the Use Case Model.

The actors are often already identified in the Business Analysis Model as business workers. They are the ones that need the system to do their jobs. The actors in Figure 9 were the business workers I had casted in the domain model, remember? An actor could also correspond to a business actor, in case the business actor can access the system directly, through the internet for instance.

Use case identification

The use cases can be derived from the actions defined for the business workers in the activity diagrams. This does not mean that there is a one-to-one correpondence, as you will notice when you compare Figure 9 with the activity diagram in the Business Analysis Model (Figure 5). I recommend documenting the relationships between the use cases and the work flow defined in the business model.

Use case specification

Most RUP practitioners write use case specifications only as structured text. That can work very well. It can also become a small disaster. How often did you renumber the steps in your flows? How often did your alternative flows look like a bunch of snakes, biting each other's tails? Those of you who start smiling at this point, may benefit from the option to use activity diagrams to lay down all alternative paths in a simple picture. Testers can greatly enjoy these pictures too, when doing a test paths analysis.

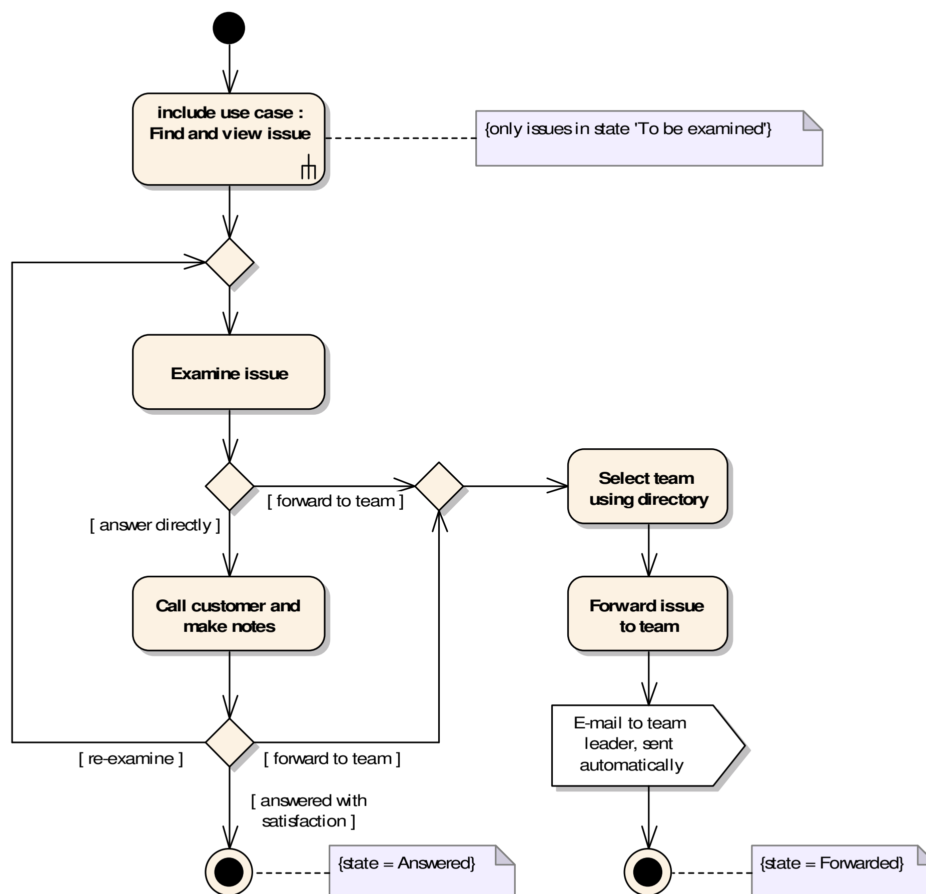

Figure 10. An activity diagram showing the flow during use case 'Examine issue'.

Figure 10 shows the internals of use case ‘Examine issue’. The situation is, that a help desk member takes an issue from the pool of issues that are in the state ‘To be examined’ in order to either answer this issue immediately, or to forward it to the right team (second line help) for further analysis. I hope you can interpret the flow without further explanation, but in practice, when my diagrams have stabilized, I add a piece of text to my diagrams to help future readers. Even more important is the specification per action. Each action in the diagram should be specified either by text (maybe a one-liner, maybe a lot more), or by a separate diagram. I’m afraid I’ll skip that for now. As you can see, the use case starts off with an included use case. The rake symbol in the bottom right corner indicates that there is a separate activity diagram for this action. Do you notice the references to my state machine in the business model?

Requirements – Summary

Altogether, the following UML diagram types may be found in a Use Case Model.

Package diagram	Shows an overview of the packages, in case your system is large enough to require separate use case packages.
Use case diagram	Shows an overview of the use cases, their relationships and their relationships with the actors.
Activity diagram	A visualization of all possible flows of interaction between actor and system during the use case.

Table 3. Possible UML diagram types in the Use Case Model.

The complete picture of UML diagram types so far, looks like this. The blue ones are mandatory in my opinion, even for small systems.

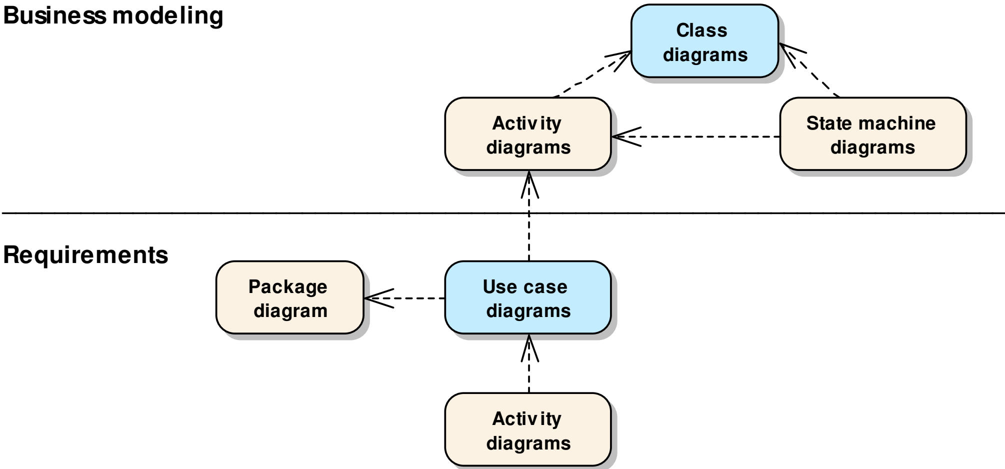

Figure 11. UML diagram types for the business modeling and requirements disciplines.

ANALYSIS AND DESIGN

Now, I will take you to the complex discipline called "Analysis & Design". It adds six models to the set of models produced by the business modeling and requirements disciplines: the Navigation Map, the Analysis Model, the Service Model, the Design Model, the Data Model and the Deployment Model. The other models gave us insight into the business and the requirements, but the six newcomers are models of the actual software to be built. I will discuss these models one by one.

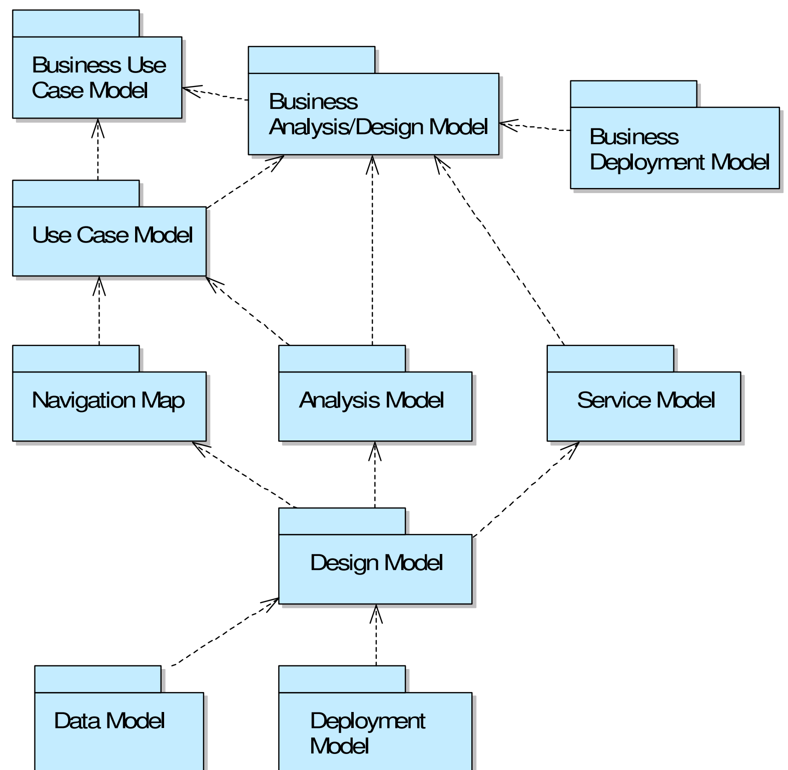

Figure 12. The RUP models of the three disciplines

Navigation Map

RUP defines the activity *Design the User Interface* resulting in the *Navigation Map*. This map is based on the use cases and shows the most important navigation paths. A navigation path is a sequence of screens (windows, web pages) traversed by the user. How does the map look like? In RUP, there are no rules, but a UML class diagram is mentioned as one of

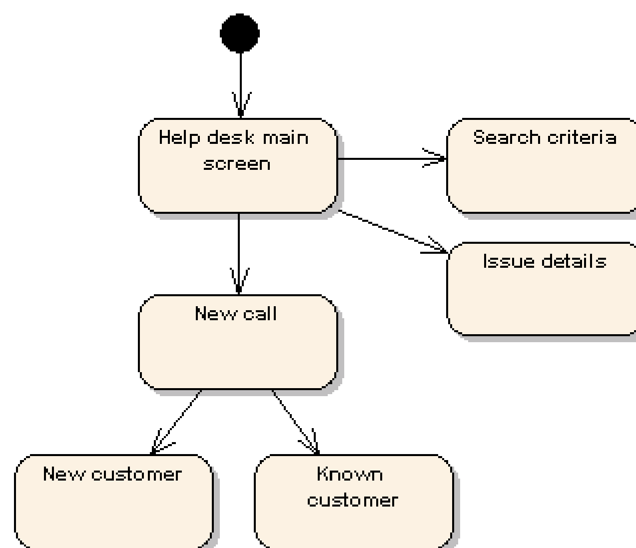

Figure 13. Navigation Map.

the options. I'd rather choose the state machine diagram. The active screen is considered to be the state of the user interface and the transition arrows show the possible navigation paths. In Figure 13, I have drawn a very sober and incomplete state machine. It is missing the triggers that cause the transitions and the actions taken by the system and it is missing exceptions. You may wonder if the user is ever allowed to go back to the main screen. The answer is yes and I explicitly mention in my user interface design document that the user can always traverse backwards, although it's not modeled in the map. The reason is that the map is not a formal, machine-readable model, but an overview, meant to convey the user interface structure to humans. I do display triggers and guards sometimes, but only if they are important to understand the map.

Some people may like to use the full power of UML to model navigation details. If I try that for only two screens, I get something like Figure 14. I promise you very complex state machines if you continue that way for the complete application. I usually write more formal and detailed specifications too, screen by screen, but not using UML. These are important for implementers and testers, but there is no RUP artifact for them. RUP does not even mention the navigation map as input to any test or implementation activity! That's a pity, because the tester and implementer have to take all user interface design decisions into account.

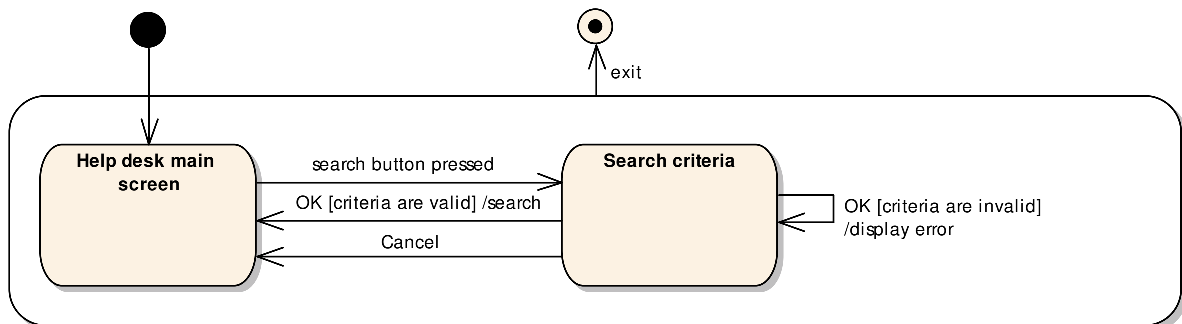

Figure 14. Detailed map of some navigation paths.

The navigation map is optional in RUP. If you have a complete user interface prototype, you may omit this model.

For large systems, the map can be very complex. According to RUP, you should put everything in one diagram, but I would create at least one navigation map for each use case package.

Analysis Model

The Analysis Model and the Design Model together reveal the system's internals that realize the use cases. The Analysis Model does this at a higher level of abstraction than the Design Model. The analysis objects are still "logical" in their nature, while the elements of the Design Model are directly recognizable in the source code. RUP allows you to go from use cases directly to the Design Model, but if this step is too large, you can set up an Analysis Model first. It should be decided per project whether the Analysis Model is replaced by the Design Model, or kept as a conceptual overview of the Design Model. Both the Analysis Model and the Design Model contain class diagrams to lay down the static structure and interaction diagrams that show the realization of use cases in terms of co-operating objects.

Nowadays, I don't make Analysis Models anymore. My Business Analysis Model and Use Case Model together provide enough information to make a first draft component architecture in the Design Model and to start making use case realizations in terms of interacting components.

The main problem with the classical RUP Analysis Model is, that it is not component-based or service-oriented. It consists of a lot of analysis classes that send messages directly to one another, without going through service or component interfaces. In my opinion, you should primarily design the service model. At a lower level of detail, you design the component architecture per service and how the service operations are realized in terms of component operations. Finally, you design then each component's internal classes and the realization of the component operations.

The second problem is, that analysis objects are logical and may not map very easily to the design objects, despite RUP's statement that the design objects are just a more detailed version of the analysis objects.

So my advice is: Put all the relevant business entities and business processes in your Business Analysis Model, then you don't need an Analysis Model anymore.

Service Model

Traditionally, a software development project delivers an application that is either stand-alone or connected to other, already existing applications. Nowadays, we should think service-oriented, i.e. we live in a world of services, some within our company and some outside, and our project is there to add new services and/or adapt some existing ones to achieve a business goal, while a thin application layer provides the user interface. This implies that you need an IT architect, who can oversee the enterprise's IT needs as a whole, instead of only those within the project's scope, and who can identify services that may contribute to more flexibility in business and in IT. An organization that really adopts SOA, should maintain an enterprise-wide Service Model.

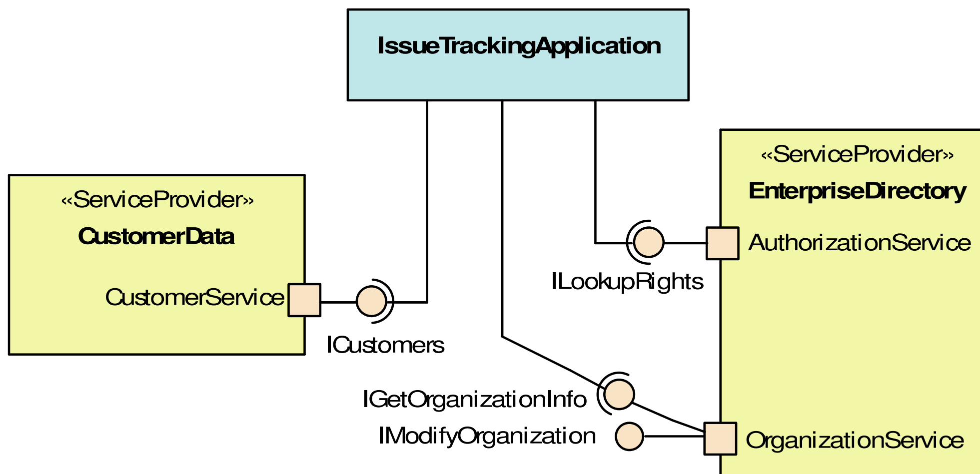

Figure 15. The main diagram of the Service Model.

Within the scope of a single RUP project, a Service Model is meant to provide a picture of the services landscape as far as is relevant for that project. Maybe we need to identify new services, maybe we only need to connect to existing services. In the former case, the services should be derived from the Business Analysis Model. In our Issue Tracking case, we could take the domain model (Figure 3) and define a CustomerService for business entity Customer and an OrganizationService for the business workers (employees and teams). The business entity Issue is very application-specific and I don't expect a need for an issue service. The two services we've just identified are shown in Figure 15, which is a composite structure diagram. We have hosted these services at service providers. The interface IModifyOrganization is not used, because our application does not modify employee or team data; we leave that for another application. One service is not derived from the business model, but from a supplementary requirement: the AuthorizationService, which is used to lookup what the current user is allowed to do with our application.

The Service Model is not yet complete. We need to define the operations available for each interface. In UML, this could look like this:

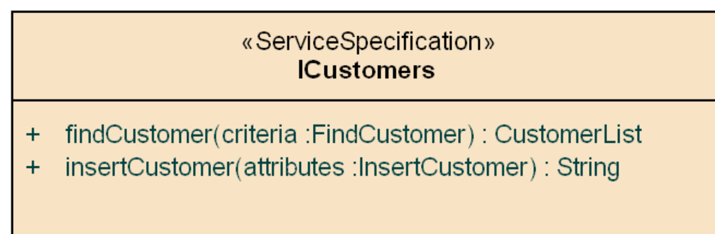

Figure 16. A service specification.

The parameter types are classes with stereotype «message» and should be defined in another class diagram. Figure 17 is an example of such a class diagram.

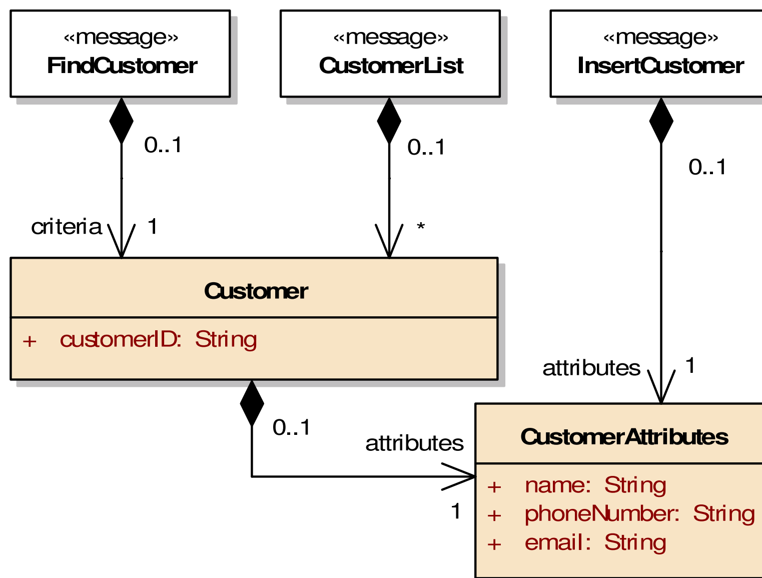

Figure 17. Message definitions in the Service Model.

Typically, the following UML diagram types are found in the Service Model..

Diagram type	Use
Package diagram	RUP examples suggest to package the model elements per element type (messages together, services together, etc.), but I'd rather create a package per service.
Composite structure diagram	Connections among services through interfaces. Allocation of services to service providers.
Class diagram	Service specifications and message definitions.
Sequence diagram	Collaboration of services (messages exchanged over time).
State machine diagram	Defines the protocol required by one particular service, i.e. the possible orders in which its operations may be called.

Table 4. Possible UML diagram types in the Service Model.

RUP also mentions the option to define a use case model for each service, but I would not recommend that. Use cases are more suited for describing user interactions.

Design Model

For most systems, analysis and design is performed on three levels:

- the service level, where applications and services and their collaboration are defined,
- the component level, at which the components of each application and service are defined,
- the object level, for the design of the components' internals.

Small systems may lack the service level and in some cases, only the object level is needed.

The service level is captured in the Service Model, the other levels in the Design Model. At the top level, the Design Model is divided into one package for each application and one for each service that lies within the scope of the project. Within each package, I recommend to strictly separate the component level from the object level. At both levels, the static structure is the basis on top of which the dynamic behavior is modeled.

Design Model: Component Level

The static structure on the component level is represented mainly by two diagrams: a package diagram (Figure 18) that depicts the layered approach and a set of component diagrams or composite structure diagrams (Figure 19) showing which components use which interfaces of other components.

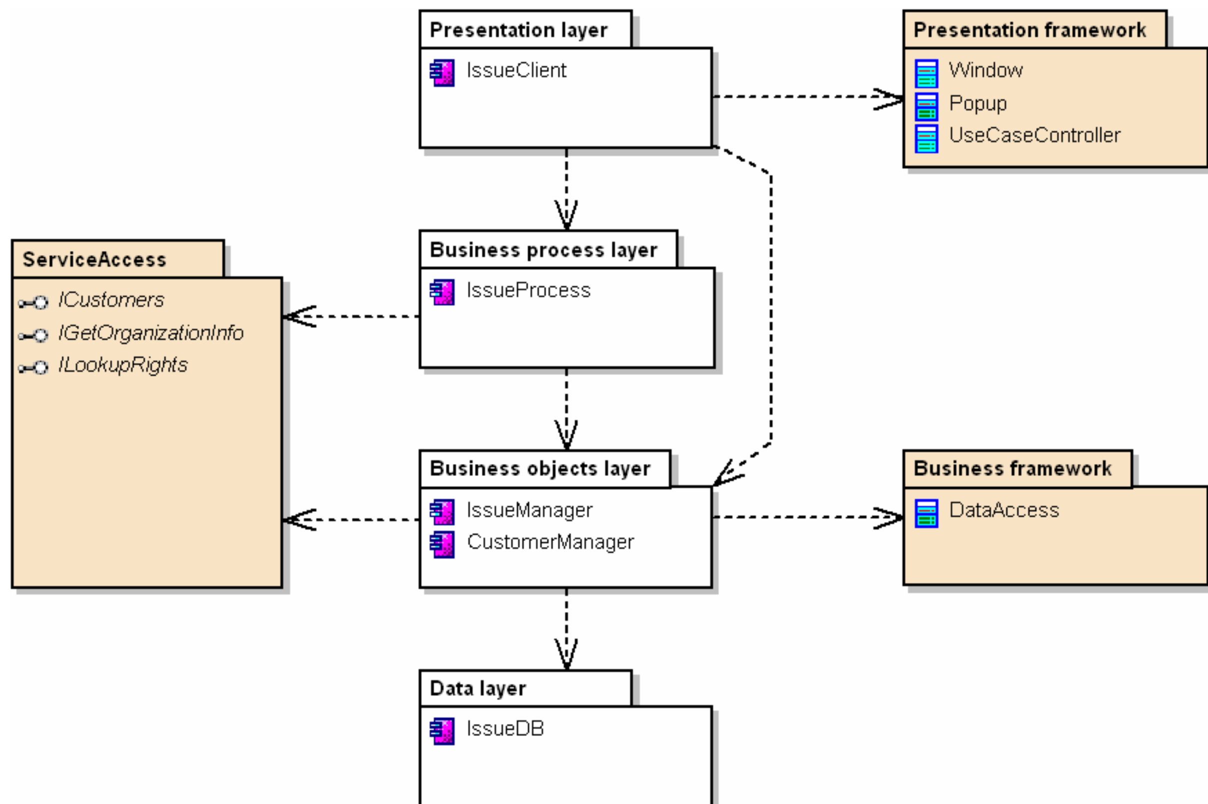

Figure 18. Package diagram showing the layered architecture of IssueTrackingApplication.

It is confusing, that in RUP, a component is represented in the Design Model by an artifact called "Design Subsystem". The component level (as defined by one of the SOA guidelines) is called system level elsewhere in the RUP material. In this white paper, we consistently use the words component and component level.

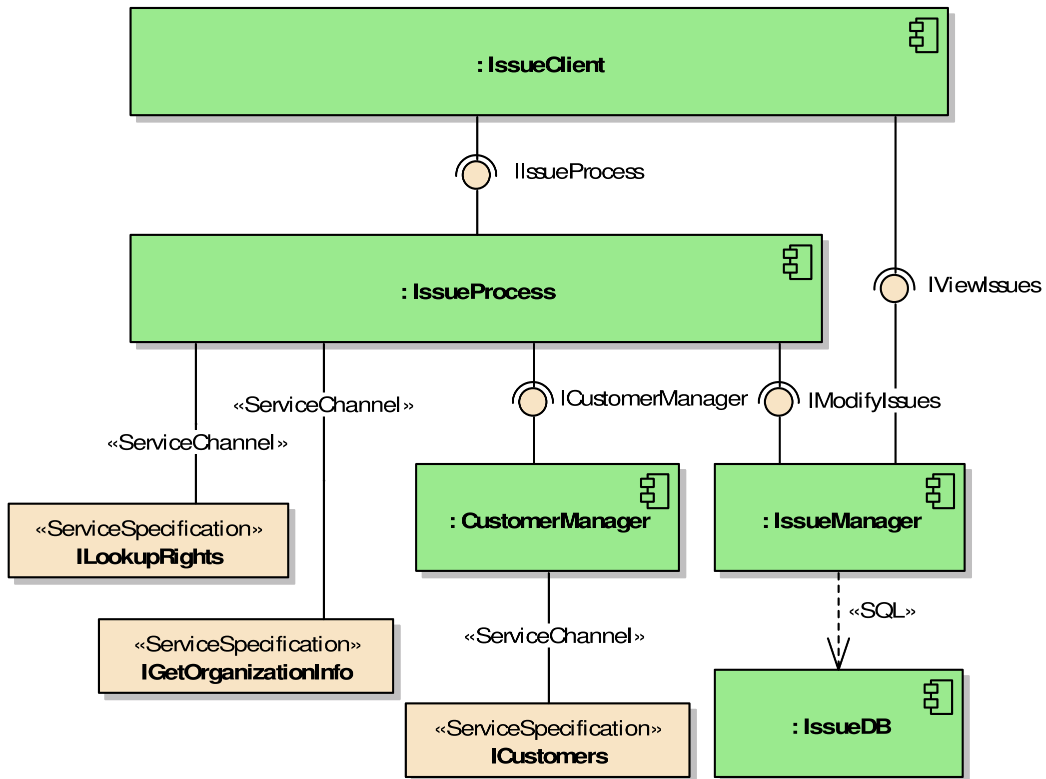

Figure 19. Composite structure diagram for IssueTrackingApplication

Additionaly, as in the Service Model, we have to define the interfaces of the components, similar to the service specifications (Figure 16) and the non-primitive parameter types, similar to the message definitions (Figure 17). I've done both in Figure 20 for the interface IViewIssues.

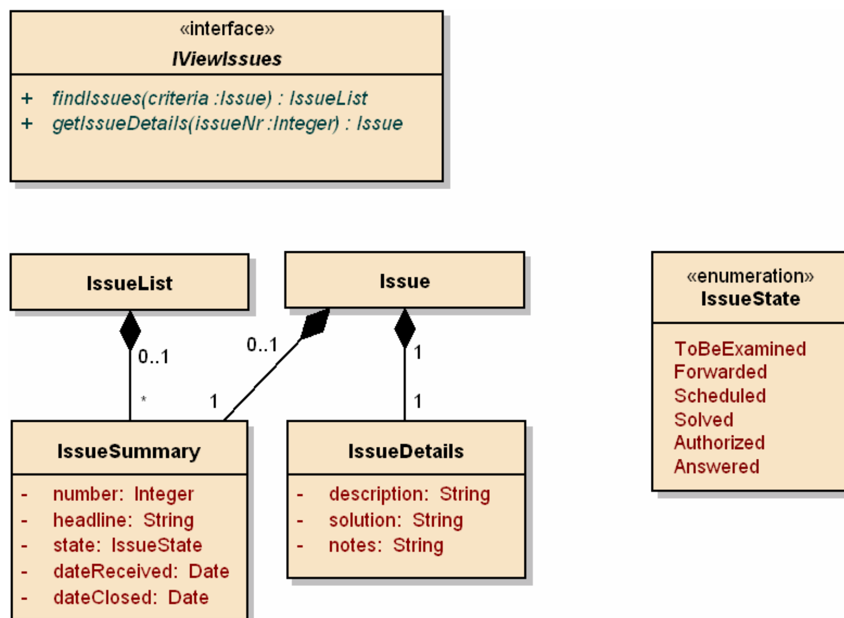

Figure 20. A class diagram defining an interface and the types of its parameters.

The dynamic part consists of use case realizations (UCR) in case of an application, or operation realizations (OpR) in case of a service. A UCR or OpR consists of:

– an interaction diagram (sometimes two or three) – the most common kind of interaction diagram is the sequence diagram;

– a class diagram to display the participating classes and their associations – this may help the implementer if the application is too complex to oversee all of its components.

These diagrams stay on the component level, i.e. they do not show internal elements of any component. Figure 21 is an example of a sequence diagram for the realization of use case “Find and view issue”. The diagram of participating classes is not needed in our example.

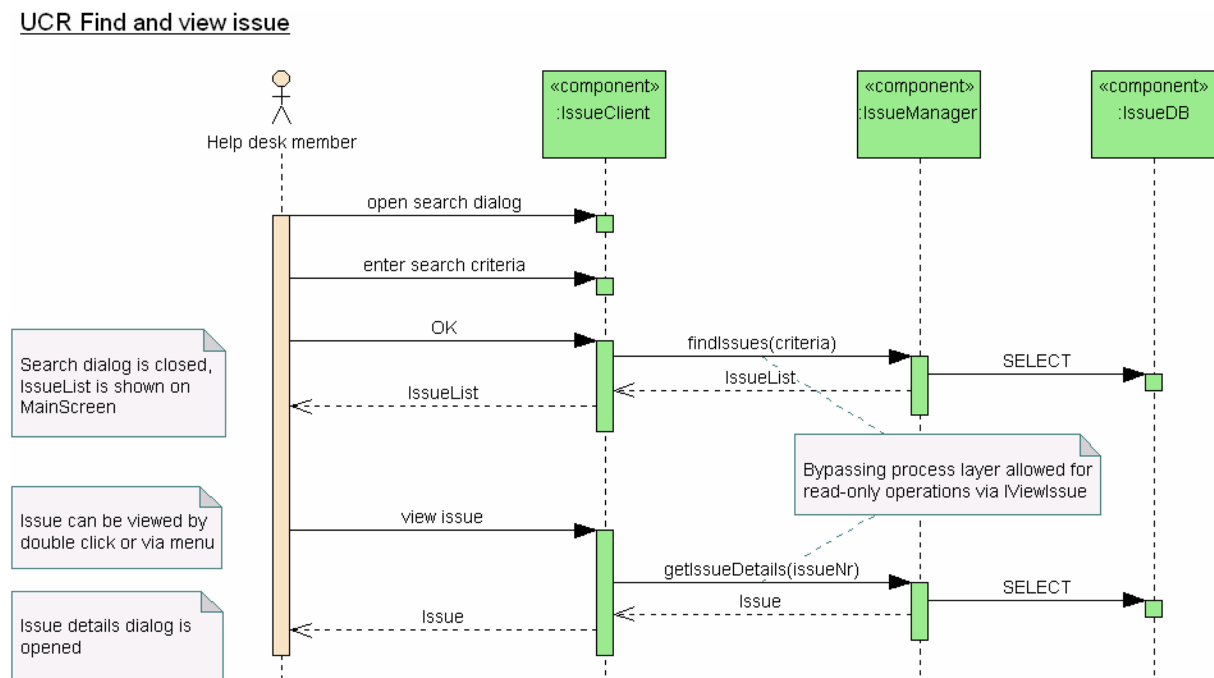

Figure 21. Sequence diagram: The realization of use case ‘Find and view issue’.

If the component level design of the application or service is done, we have done an important architectural job, crucial to the success of the project.

Design Model: Object Level

The internals of the components are far less important and may be left undesigned for simple components. Still, most components are not so trivial and need modeling to be able to understand the source code. At the object level, we encouter, again, a static and a dynamic part.

The static part consists of class diagrams. The classes shown will be programmed to implement the component.

The dynamic part is a collection of operation realizations. For each operation with a non-trivial implementation, an interaction diagram (sequence diagram or communication diagram) is created. Depending on the agility of the project and the skills of the implementers, you determine what "non-trivial" means.

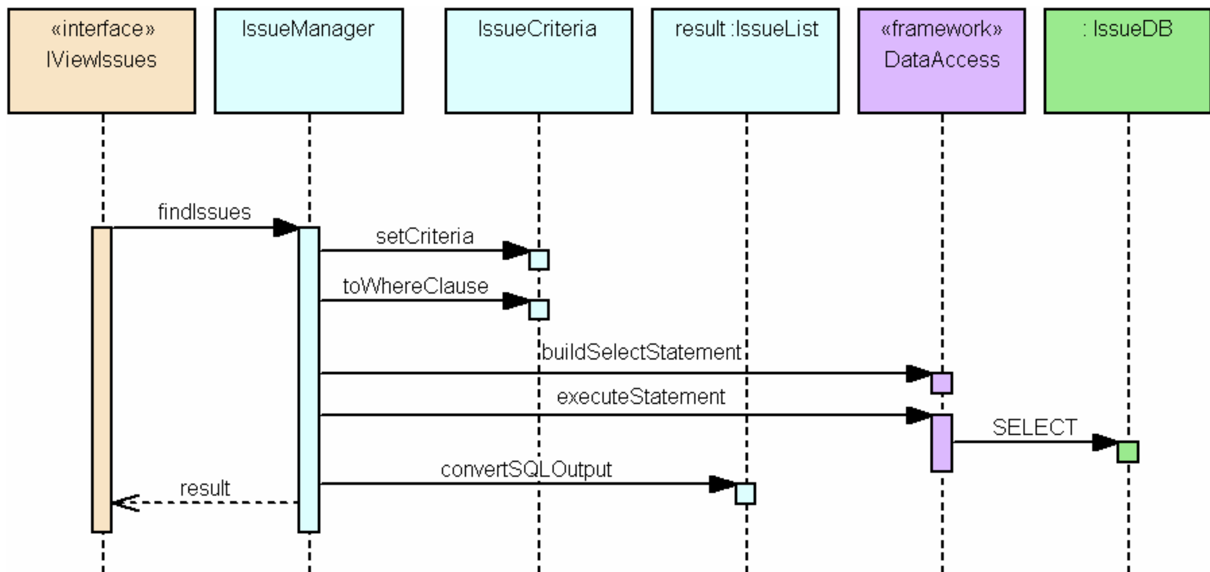

Figure 22. Sequence diagram: The realization of operation 'FindIssues'.

The user interface components are not triggered by operations, but by user-initiated events, like a button that is being pressed. For those components, the dynamic part consists of interaction diagrams that represent the reaction of the system to those events.

Figure 23 is a summary of the diagrams that are most valuable in the Service Model and the Design Model. For more light-weight designs, I would recommend at least one composite structure diagram or component diagram and for each component a class diagram (the blue elements). In practice, you may want to deviate from this picture here and there. For example, some components may also need a composite structure diagram at the object level and for complex algorithms, you may benefit from activity diagrams at the object level.

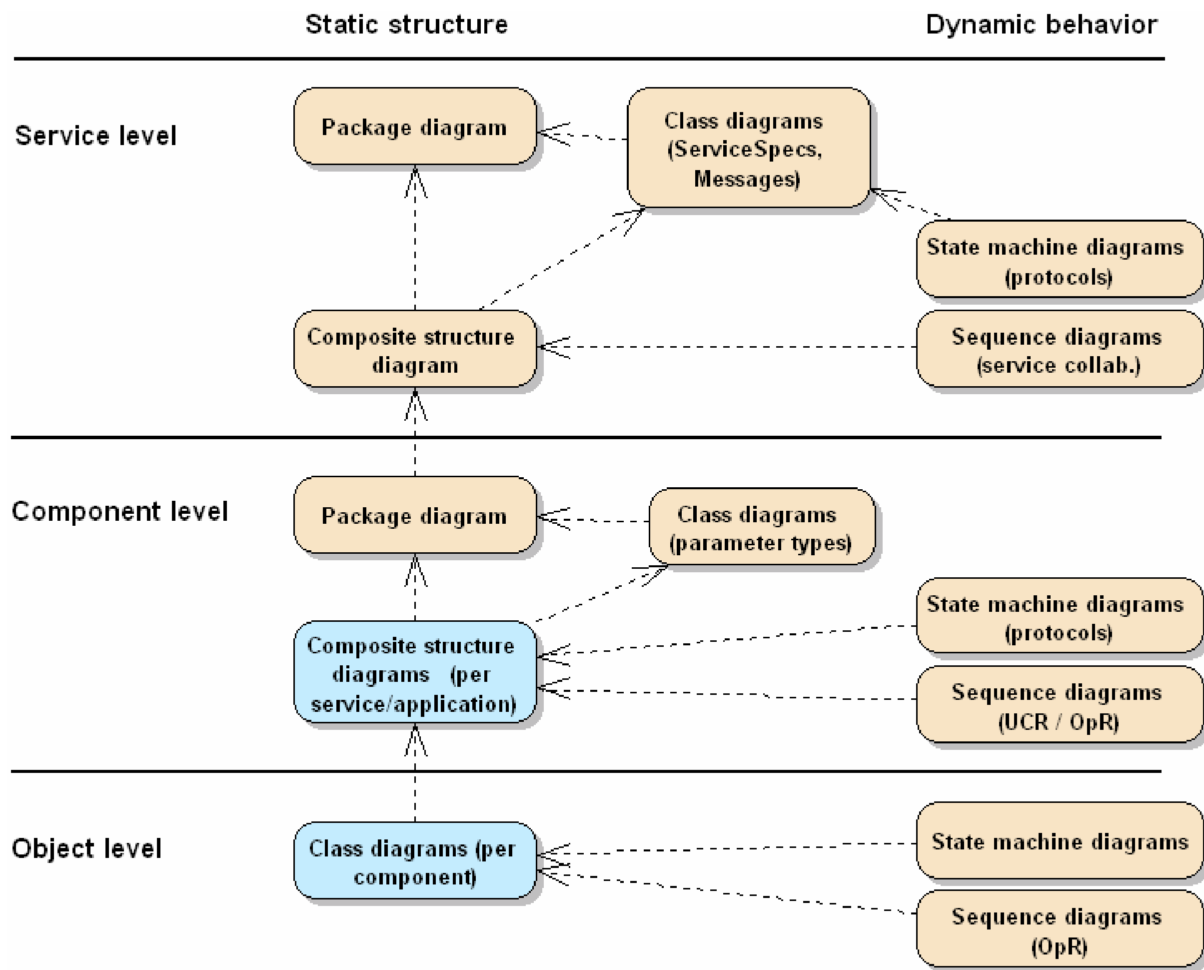

Figure 23. Diagram types in the Service Model and the Design Model.

Data Model

The last couple of models lay in a relatively safe corner of the field of pitfalls. We can switch from careful steps to a final sprint, if you don't mind.

The Data Model is a model of the database. If an RDBMS is part of the application, then the Data Model will specify the tables, columns and foreign key relations, and usually also stored procedures and triggers. These elements all fit in class diagrams, using special stereotypes like «table» and «column». When using multiple databases, each database should be shown as a component in the Design Model.

Sometimes, the Data Model is divided in packages, for example one for each database schema. A package diagram shows the dependencies.

Deployment Model

The Deployment Model specifies the required hardware and network connections. Within that framework, you allocate the software components to the machines on which they should be installed. UML's deployment diagram is meant to display this model.

IMPLEMENTATION

Implementation Model

The Implementation Model is only needed if the organization of the physical source code differs from the package and component structure defined in the Design Model. In that case, the Implementation Model specifies the source code structure (the directories, for example) and the compilation order. If you like to visualize this, you can use a package diagram. The relationships between these packages and the packages or components in the Design Model should be clear, either by using a uniform naming convention or by explicit mapping.

GOOD LUCK USING UML IN RUP!

It is often difficult to arrange the development process such, that a clear set of models is produced, taking the preferences and skills of the various team members into account. The RUP documentation is too fragmented. In this paper, I tried to bring the fragments together, mixed up with my own experiences. Did it help you? You and your team will have to sit together and find your own way. Remember, RUP is always too big. Create only those models that add value. I'm very interested to hear your comments and questions!

Hans Admiraal, (freelance IT architect and RUP trainer)
admiraal@aol.nl
www.admiraalit.nl