

User name: VLADIMIR BLAGOJEVIC
Book: Learning UML 2.0

No part of any chapter or book may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher of the book or chapter. Redistribution or other use that violates the fair use privilege under U.S. copyright laws (see 17 USC107) or that otherwise violates these Terms of Service is strictly prohibited. Violators will be prosecuted to the full extent of U.S. Federal and Massachusetts laws.

2.2. Use Case Relationships

A use case describes the way your system behaves to meet a requirement. When filling out your use case descriptions, you will notice that there is some similarity between steps in different use cases. You may also find that some use cases work in several different modes or special cases. Finally, you may also find a use case with multiple flows throughout its execution, and it would be good to show those important optional cases on your use case diagrams.

Wouldn't it be great if you could get rid of the repetition between use case descriptions and show important optional flows right on your use case diagrams? OK, so that was a loaded question. You can show reusable, optional, and even specialized use case behavior between use cases.

2.2.1. The <<include>> Relationship

So far, you have seen that use cases typically work with actors to capture a requirement. Relationships between use cases are more about breaking your system's behavior into manageable chunks than adding anything new to your system. The purpose of use case relationships is to provide your system's designers with some architectural guidance so they can efficiently break down the system's concerns into manageable pieces within the detailed system design.



In addition to blogs, a CMS can have any number of means for working with its content. One popular mechanism for maintaining documents is by creating a Wiki. Wikis allow online authors to create, edit, and link together web pages to create a web of related content, or a Wiki-web. A great example of a Wiki is available at <http://www.Wikipedia.org>.

Take another look at the [Create a new Blog Account](#) use case description shown in [Table 2-2](#). The description seems simple enough, but suppose another requirement is added to the [Content Management System](#).

Requirement A.2

The content management system shall allow an administrator to create a new personal Wiki, provided the personal details of the applying author are verified using the Author Credentials Database.

To capture Requirement A.2 a new use case needs to be added to the [Content Management System](#), as shown in [Figure 2-11](#).

Now that we have added the new use case to our model, it's time to fill out a detailed use case description (shown in [Table 2-3](#)). See [Table 2-1](#) if you need to refresh your memory about the meaning of each of the details within a use case description.

Figure 2-11. A new requirement can often mean a new use case for the system, although it's not always a one-to-one mapping

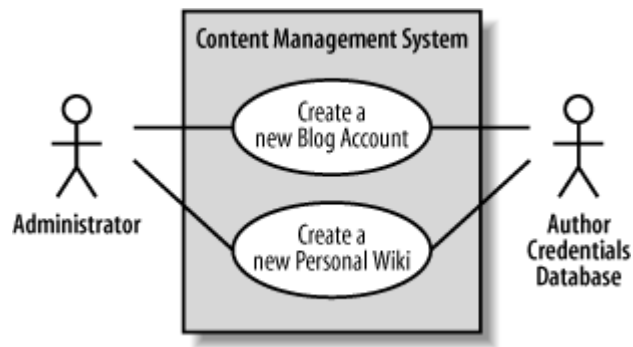


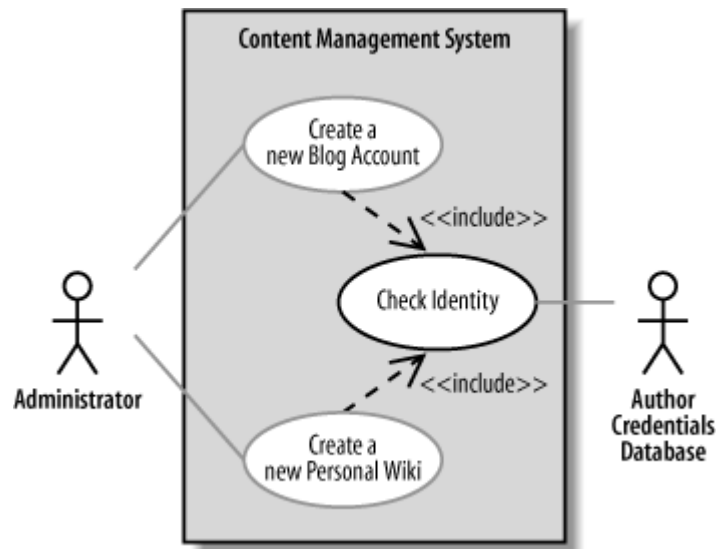
Table 2-3. The detailed description for the "Create a new Personal Wiki" use case

Use case name	Create a new Personal Wiki												
Related Requirements	Requirement A.2.												
Goal In Context	A new or existing author requests a new personal Wiki from the Administrator.												
Preconditions	The author has appropriate proof of identity.												
Successful End Condition	A new personal Wiki is created for the author.												
Failed End Condition	The application for a new personal Wiki is rejected.												
Primary Actors	Administrator.												
Secondary Actors	Author Credentials Database.												
Trigger	The Administrator asks the CMS to create a new personal Wiki.												
Main Flow	<table><tr><td>Step</td><td>Action</td></tr><tr><td>1</td><td>The Administrator asks the system to create a new personal Wiki.</td></tr><tr><td>2</td><td>The Administrator enters the author's details.</td></tr><tr><td>3</td><td>The author's details are verified using the Author Credentials Database.</td></tr><tr><td>4</td><td>The new personal Wiki is created.</td></tr><tr><td>5</td><td>A summary of the new personal Wiki's details are emailed to the author.</td></tr></table>	Step	Action	1	The Administrator asks the system to create a new personal Wiki.	2	The Administrator enters the author's details.	3	The author's details are verified using the Author Credentials Database.	4	The new personal Wiki is created.	5	A summary of the new personal Wiki's details are emailed to the author.
Step	Action												
1	The Administrator asks the system to create a new personal Wiki.												
2	The Administrator enters the author's details.												
3	The author's details are verified using the Author Credentials Database.												
4	The new personal Wiki is created.												
5	A summary of the new personal Wiki's details are emailed to the author.												
Extensions	<table><tr><td>Step</td><td>Branching Action</td></tr><tr><td>3.1</td><td>The Author Credentials Database does not verify the author's details.</td></tr><tr><td>3.2</td><td>The author's new personal Wiki application is rejected.</td></tr></table>	Step	Branching Action	3.1	The Author Credentials Database does not verify the author's details.	3.2	The author's new personal Wiki application is rejected.						
Step	Branching Action												
3.1	The Author Credentials Database does not verify the author's details.												
3.2	The author's new personal Wiki application is rejected.												

The first thing to notice is that we have some redundancy between the two use case descriptions (Tables 2-2 and 2-3). Both `Create a new Blog Account` and `Create a new Personal Wiki` need to check the applicant's credentials. Currently, this behavior is simply repeated between the two use case descriptions.

This repetitive behavior shared between two use cases is best separated and captured within a totally new use case. This new use case can then be reused by the `Create a new Blog Account` and `Create a new Personal Wiki` use cases using the `<<include>>` relationship (as shown in Figure 2-12).

Figure 2-12. The `<<include>>` relationship supports reuse between use cases



The `<<include>>` relationship declares that the use case at the head of the dotted arrow *completely* reuses all of the steps from the use case being included. In [Figure 2-12](#), the `Create a new Blog Account` and `Create a new Personal Wiki` completely reuse all of the steps declared in the `Check Identity` use case.

You can also see in [Figure 2-12](#) that the `Check Identity` use case is not directly connected to the `Administrator` actor; it picks this connection up from the use cases that include it. However, the connection to the `Author Credentials Database` is now solely owned by the `Check Identity` use case. A benefit of this change is that it emphasizes that the `Check Identity` use case is the only one that relies directly on a connection to the `Author Contact Details Database` actor.

To show the `<<include>>` relationship in your use case descriptions, you need to remove the redundant steps from the `Create a new Blog Account` and `Create new Personal Wiki` use case descriptions and instead use the `Included Cases` field and `include::<use case name>` syntax to indicate the use case where the reused steps reside, as shown in [Tables 2-4](#) and [2-5](#).

Table 2-4. Showing <<include>> in a use case description using Included Cases and include::<use case name>

Use case name	Create a new Blog Account	
Related Requirements	Requirement A.1.	
Goal In Context	A new or existing author requests a new blog account from the Administrator.	
Preconditions	The author has appropriate proof of identity.	
Successful End Condition	A new blog account is created for the author.	
Failed End Condition	The application for a new blog account is rejected.	
Primary Actors	Administrator	
Secondary Actors	None	
Trigger	The Administrator asks the CMS to create a new blog account.	
Included Cases	Check Identity	
Main Flow	Step	Action
	1	The Administrator asks the system to create a new blog account.
	2	The Administrator selects an account type.
	3	The Administrator enters the author's details.
	4	The author's details are checked.
	include::Check Identity	
	5	The new account is created.
	6	A summary of the new blog account's details are emailed to the author.

Table 2-5. The Create a new Personal Wiki use case description also gets a makeover

Use case name	Create a new Personal Wiki	
Related Requirements	Requirement A.2	
Goal In Context	A new or existing author requests a new personal Wiki from the Administrator.	
Preconditions	The author has appropriate proof of identity.	
Successful End Condition	A new personal Wiki is created for the author.	
Failed End Condition	The application for a new personal Wiki is rejected.	
Primary Actors	Administrator	
Secondary Actors	None	
Trigger	The Administrator asks the CMS to create a new personal Wiki.	
Included Cases	Check Identity	
Main Flow	Step	Action
	1	The Administrator asks the system to create a new personal Wiki.
	2	The Administrator enters the author's details.
	3	The author's details are checked.
	include::Check Identity	
	5	The new personal Wiki is created.
	6	A summary of the new personal Wiki's details are emailed to the author.

Now you can create a use case description for the reusable steps within the [Check Identity](#) use case, as shown in [Table 2-6](#).

Table 2-6. The Check Identity use case description contains the reusable steps

Use case name	Check Identity
Related Requirements	Requirement A.1, Requirement A.2.
Goal In Context	An author's details need to be checked and verified as accurate.
Preconditions	The author being checked has appropriate proof of identity.
Successful End Condition	The details are verified.
Failed End Condition	The details are not verified.
Primary Actors	Author Credentials Database.
Secondary Actors	None.
Trigger	An author's credentials are provided to the system for verification.
Main Flow	Step Action 1 The details are provided to the system. 2 The Author Credentials Database verifies the details. 3 The details are returned as verified by the Author Credentials Database.
Extensions	Step Branching Action 2.1 The Author Credentials Database does not verify the details. 2.2 The details are returned as unverified.

Why bother with all this hassle with reuse between use cases? Why not just have two use cases and maintain the similar steps separately? All this reuse has two important benefits:

- Reuse using `<<include>>` removes the need for tedious cut-and-paste operations between use case descriptions, since updates are made in only one place instead of every use case.
- The `<<include>>` relationship gives you a good indication at system design time that the implementation of `Check Identity` will need to be a reusable part of your system.

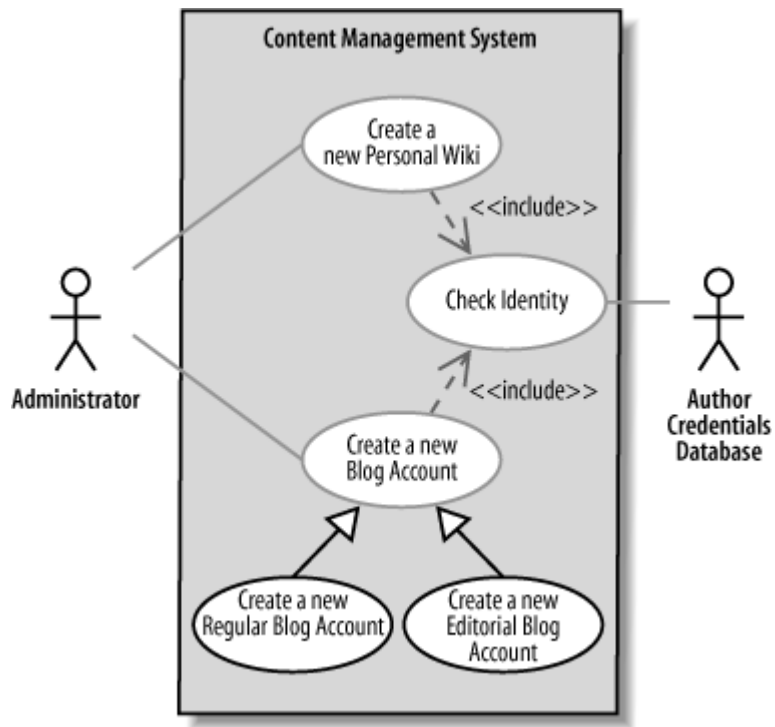
2.2.2. Special Cases

Sometimes you'll come across a use case whose behavior, when you start to analyze it more carefully, can be applied to several different cases, but with small changes. Unlike the `<<include>>` relationship, which allows you to reuse a small subset of behavior, this is applying a use case with small changes for a collection of specific situations. In object-oriented terms, you potentially have a number of specialized cases of a generalized use case.

Let's take a look at an example. Currently, the `Content Management System` contains a single `Create a new Blog Account` use case that describes the steps required to create an account. But what if the CMS supports several different types of blog accounts, and the steps required to create each of these accounts differs ever so slightly from the original use case? You want to describe the *general* behavior for creating a blog account—captured in the `Create a new Blog Account` use case—and then define specialized use cases in which the account being created is a specific type, such as a regular account with one blog or an editorial account that can make changes to entries in a set of blogs.

This is where *use case generalization* comes in. A more common way of referring to generalization is using the term inheritance. *Use case inheritance* is useful when you want to show that one use case is a special type of another use case. To show use case inheritance, use the generalization arrow to connect the more general, or *parent*, use case to the more specific use case. [Figure 2-13](#) shows how you could extend the CMS's use cases to show that two different types of blog accounts can be created.

Figure 2-13. Two types of blog account, regular and editorial, can be created by the Management System



Taking a closer look at the [Create a new Editorial Blog Account](#) specialized use case description, you can see how most of the behavior from the more general [Create a new Blog Account](#) use case is reused. Only the details that are specific to creating a new editorial account need to be added (see [Table 2-7](#)).

Table 2-7. You can show that a use case is a special case of a more general use case within the detailed description using the Base Use Cases field

Use case name	Create a new Editorial Blog Account	
Related Requirements	Requirement A.1.	
Goal In Context	A new or existing author requests a new editorial blog account from the Administrator .	
Preconditions	The author has appropriate proof of identity.	
Successful End Condition	A new editorial blog account is created for the author.	
Failed End Condition	The application for a new editorial blog account is rejected.	
Primary Actors	Administrator.	
Secondary Actors	None.	
Trigger	The Administrator asks the CMS to create a new editorial account that will allow an author to edit entries in a set of blogs.	
Base Use Cases	Create a new Blog Account	
Main Flow	Step	Action
	1	The Administrator asks the system to create a new blog account.
	2	The Administrator selects the editorial account type.
	3	The Administrator enters the author's details.
	4	The Administrator selects the blogs that the account is to have editorial rights over.
	5	The author's details are checked.
	include::Check Identity	
	6	The new editorial account is created.
	7	A summary of the new editorial account's details are emailed to the author.
Extensions	Step	Branching Action
	5.1	The author is not allowed to edit the indicated blogs.
	5.2	The editorial blog account application is rejected.
	5.3	The application rejection is recorded as part of the author's history.

Use case inheritance is a powerful way of reusing a use case so that you only have to specify the extra steps that are needed in the more specific use cases. See [Chapter 5](#) for more information on inheritance between classes.

But be careful—by using inheritance, you are effectively saying that *every* step in the general use case *must* occur in the specialized use cases. Also, every relationship that the general use case has with external actors or use cases, as shown with the `<<include>>` relationship between [Create a new Blog Account](#) and [Check Identity](#), must also make sense in the more specialized use cases, such as [Create a new Editorial Blog Account](#).

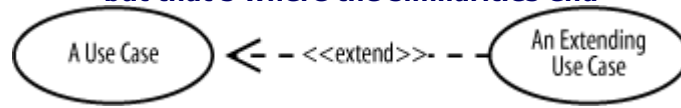
If you really don't want your more specific use case to do everything that the general use case describes, then *don't use generalization*. Instead, you might want to consider using either the `<<include>>` relationship shown in the previous section or the `<<extend>>` relationship coming up in the next section.

2.2.3. The `<<extend>>` Relationship

Any explanation of the `<<extend>>` stereotype should be preceded by a warning that it is the most heavily debated type of use case relationship. Almost nothing is less understood or harder to accurately communicate within the UML modeling community than the `<<extend>>` use case relationship, and this presents a bit of a problem when you are

trying to learn about it. Figure 2-14 shows you how `<<extend>>` works; take a look, and then let's dive into some UML concept and theory.

Figure 2-14. The `<<extend>>` use case relationship looks a bit like the `<<include>>` relationship, but that's where the similarities end



At first glance—particularly if you are a Java programmer—`<<extend>>` seems very similar to inheritance between classes. In Java, a class can extend from a base class. Similarly, in C++ and C#, you can declare inheritance between classes, and you would often say that a class extends another class. In both these cases, the extend relationship between classes *means inheritance*. So, for a programmer, it follows that `<<extend>>` should mean something like inheritance, right?

Alarm bells should definitely be going off now. You already saw in the previous section how use cases declare inheritance using a generalization arrow, so why would you need yet another type of arrow with an `<<extend>>` stereotype? Does the generalization arrow mean the same thing as the `<<extend>>` stereotype? Unfortunately, the `<<extend>>` stereotype has *very little in common with inheritance*, and so the two definitely do not mean the same thing.

The designers of UML 2.0 took a very different view as to the meaning of `<<extend>>` between use cases. They wanted a means for you to show that a use case *might* completely reuse another use case's behavior, similar to the `<<include>>` relationship, but that this reuse was *optional* and dependent either on a runtime or system implementation decision.

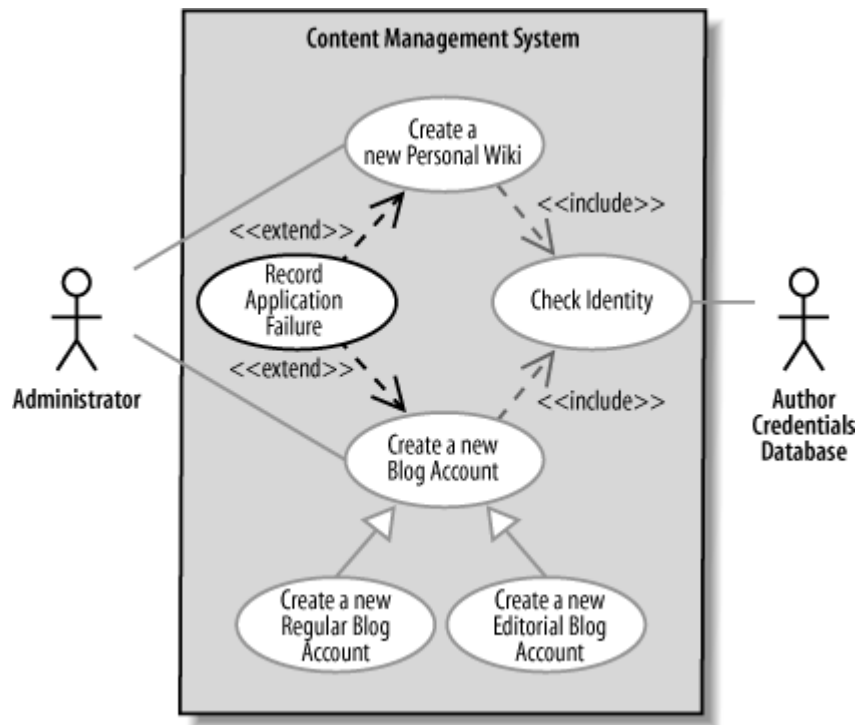
From the CMS example, the `Create a new Blog Account` use case might want to record that a new author applied for an account and was rejected, adding this information to the author's application history. Extra steps can be added to the `Create a new Blog Account` use case's description to show this optional behavior, as shown in Step 4.3 in Table 2-8.

Table 2-8. Behavior that is a candidate for <<extend>> relationship reuse can usually be found in the Extensions section of a use case description

Use case name	Create a new Blog Account	
Related Requirements	Requirement A.1.	
Goal In Context	A new or existing author requests a new blog account from the Administrator.	
Preconditions	The author has appropriate proof of identity.	
Successful End Condition	A new blog account is created for the author.	
Failed End Condition	The application for a new blog account is rejected.	
Primary Actors	Administrator.	
Secondary Actors	None.	
Trigger	The Administrator asks the CMS to create a new blog account.	
Included Cases	Check Identity	
Main Flow	Step	Action
	1	The Administrator asks the system to create a new blog account.
	2	The Administrator selects an account type.
	3	The Administrator enters the author's details.
	4	The author's details are checked.
	include::Check Identity	
	5	The new account is created.
	6	A summary of the new blog account's details are emailed to the author.
	Step	Branching Action
	4.1	The author is not allowed to create a new blog.
Extensions	4.2	The blog account application is rejected.
	4.3	The application rejection is recorded as part of the author's history.

The same behavior captured in Step 4.3 would also be useful if the customer was refused an account for some reason during the [Create a new Personal Wiki](#) use case's execution. According to the requirements, this reusable behavior is *optional* in both cases; you don't want to record a rejection if the application for a blog account or a personal Wiki was accepted. The <<extend>> relationship is ideal in this sort of reuse situation, as shown in [Figure 2-15](#).

Figure 2-15. The <<extend>> relationship comes into play to show that both the "Create a new Personal Wiki" and "Create a new Blog Account" use cases might occasionally share the application rejection recording behavior



The new `Record Application Failure` use case, as the name implies, captures all of the behavior associated with recording an author's application failure whether it be for a personal Wiki or for a specific type of blog account. Using the `<<extend>>` relationship, the `Record Application Failure` use case's behavior is *optionally* reused by the `Create a new Blog Account` and `Create a new Personal Wiki` use cases if an application is rejected.

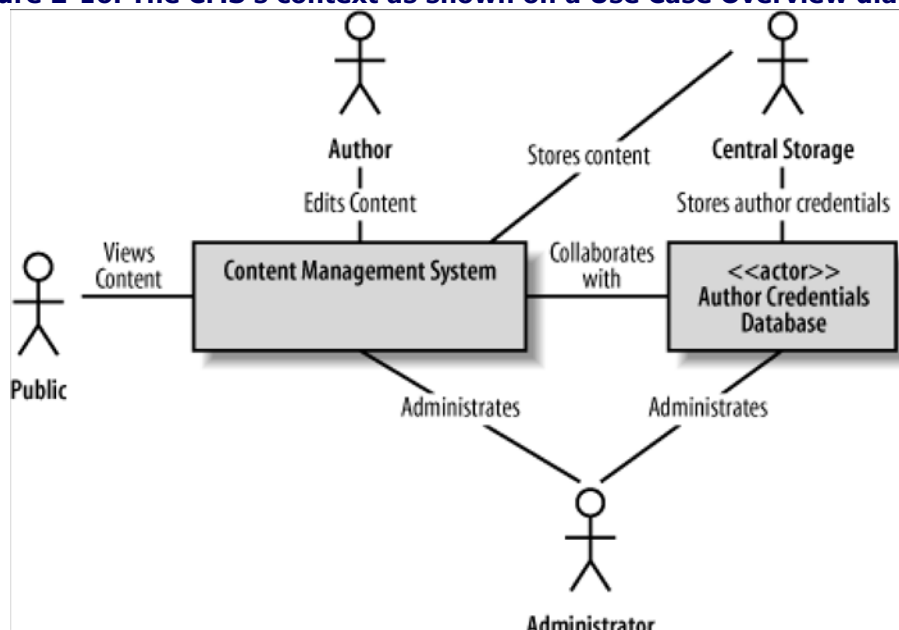
User name: VLADIMIR BLAGOJEVIC
Book: Learning UML 2.0

No part of any chapter or book may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher of the book or chapter. Redistribution or other use that violates the fair use privilege under U.S. copyright laws (see 17 USC107) or that otherwise violates these Terms of Service is strictly prohibited. Violators will be prosecuted to the full extent of U.S. Federal and Massachusetts laws.

2.3. Use Case Overview Diagrams

When you are trying to understand a system, it is sometimes useful to get a glimpse of the context within which it sits. For this purpose, UML provides the *Use Case Overview* diagram. Use Case Overview diagrams give you an opportunity to paint a broad picture of your system's context or domain (see [Figure 2-16](#) for an example).

Figure 2-16. The CMS's context as shown on a Use Case Overview diagram



Unfortunately, Use Case Overviews are badly named as they don't usually contain any use cases. The use cases are not shown because the overview is designed to provide a context to your system; the system's internals—captured by use cases—are not normally visible.

Use Case Overviews are a useful place to show any extra snippets of information when understanding your system's place within the world. Those snippets often include relationships and communication lines between actors. These contextual pieces of information do not usually contain a great deal of detail, they are more a placeholder and starting point to for the rest of your model's detail.