Abstract: In order to reduce the costs for systems development, methods for the reuse of specification knowledge have been developed. One approach is to build libraries of reusable specification patterns, i.e. abstract models describing the generic features of a type of situation that may occur in many different domains. In this paper, we propose a novel specification pattern based on a deontic perspective. The basic components of this pattern are object types describing obligations, the parties involved in these obligations and their respective roles, and the speech acts that create and delete the obligations. We argue that this pattern captures specification knowledge at an appropriate level of abstraction, has a wide applicability, and effectively supports designers in the construction of models. Furthermore a number of instances of this pattern are analysed and classified in different categories.

1 Introduction

When constructing large information systems, the key to success lies in eliciting and representing requirements. Experience has shown that these activities are difficult as well as time consuming. Even with the use of CASE tools, capturing and representing requirements remain one of the major costs in building information systems. One reason for this is that requirements and domain knowledge are still often captured from scratch for each new system to be built, even for systems within the same general area. This duplication of effort results in high costs and hinders the construction of larger and more knowledge-intensive systems. To overcome these problems, systems analysts and software engineers must find ways of sharing, reusing, and extending systems.
There are many senses in which the knowledge contained in a system can be shared and reused. One form of reuse is code reuse, which can be realized through modules invoking each other as procedures from a function library. Code reuse can also be realized through the inclusion of source specifications, i.e. the content of one module is copied into another module at design time. Another form of reuse is through the exchange of techniques, meaning that the content of a system module is not directly used; instead, the solution approach behind the module is communicated in a way that facilitates its re-implementation.

Essential to all these forms of reuse is the build-up and maintenance of a library of reusable modules. Such a library can be utilized in many different ways. It can be searched through keywords that retrieve and select modules based on their functionality, as suggested in [Burton87]. However, keywords describing the functionality of systems cannot effectively support reuse across different applications. Faceted classification schemes, [Prieto-Diaz87], overcome some of the problems of simple keyword retrieval by describing non-functional features of modules and by providing a lexicon to support differences in terminology. The contents of the modules in a library may vary widely, from source code to generic objects and models. The latter are abstract models that describe the generic features of a type of situation that may occur in many different contexts; these abstract models are commonly known as patterns.

Patterns come in a large number of varieties. One of these is the design pattern, [Gamma95], which has received much attention in the software engineering community. A design pattern is a description of “communicating objects and classes that are customized to solve a general design problem in a particular context.” [Gamma95]. Thus, a design pattern names, abstracts, and identifies the key aspects of a common design structure that make it useful for creating a reusable object-oriented design. While a design pattern addresses the design stage in systems development, a specification pattern concerns the analysis and specification stage. A specification pattern consists of an application independent model of a domain structure, e.g. a model of time or causality at a higher level of abstraction, or a model of library systems at a lower level. A specification pattern describes, at an arbitrary level of abstraction, a set of real-world objects, their interrelationships, and the rules that govern their behaviour and state. Some examples of specification patterns are domain abstractions as discussed in [Maiden92], the analysis patterns in [Fowler97], and data model patterns as introduced in [Hay96]. The notion of specification patterns is similar to that of ontologies in the knowledge representation community, [Neches91]. An ontology is commonly defined as an explicit specification of a conceptualisation, where a conceptualisation consists of “objects, concepts and other entities that are assumed to exist in some area of interest and the relationships that hold among them”, [Gruber95].
An important quality requirement on a specification pattern is that it be sufficiently general, i.e. it should be sufficiently abstract to be applied to many different application models. Another quality requirement is that specification patterns should minimize the cognitive distance, i.e. it should be easier to understand and apply the specification pattern than to model a part of the application from scratch. Clearly, these two quality requirements are in conflict as highly abstract patterns may be quite difficult to apply. The difficulty to construct patterns satisfying both these requirements is probably a major reason why specification patterns are not yet widely used. An important research task is, therefore, to identify specification patterns at an appropriate level of abstraction.

The basic components of this pattern are object types describing obligations, the parties involved in these obligations and their respective roles, the speech acts that create and delete these obligations, the subject matters of and the reasons for the obligations. We claim that this specification pattern is both sufficiently general and has a low cognitive distance, which means that it will be useful in a large number of areas as well as simple to apply. The paper is organized as follows. Section 2 introduces the modelling formalism and notation used in the rest of the paper. Section 3 provides a list of specification patterns at an intermediate level of abstraction. The purpose of this list is to show a number of patterns that at first sight may appear quite dissimilar, but actually share many common features. Section 4 presents a generalization of these patterns into a general deontic pattern; the components of this pattern are described as well as its typical variants. Section 5 analyses the differences between deontic objects and classifies them in a tree structure. Finally, Section 6 summarizes the paper and gives suggestions for further research.

2 Modelling Formalism

The basic concept in conceptual modelling approaches is the object. Objects which are similar are grouped together into object types, such as Person and Department. Objects have different properties, which are modelled by attributes or relationships. In our graphical notation, see Figure 1, object types are represented by rectangles, attributes are represented by bulletted lists inside the object types and relationships are repre-
sented by labelled lines. Both a relationship and its inverse is represented with the same line. The name of a relationship is placed closest to the object type which is its domain. The object type on the other side of the line representing the relationship is called its range. The graphical notation can only represent cardinality constraints and generalization relationships. The generalization relationships are shown by drawing the subtypes of an object type inside the rectangle modelling this object type. The cardinality constraints specify for each relationship if it is single-valued, injective, total and surjective. A relationship is single-valued when each instance of its domain is connected with at most one instances of its range. A relationship which is not single-valued is multi-valued and is depicted by a "fork" connecting the line representing the relationship with its range. A relationship is total when each instance of its domain is connected to at least one instance in its range. A relationship which is not total is partial, which is shown by a dotted line on the half of the line which is closest to its domain. A relationship is injective (surjective) when its inverse is single valued (total).

![Figure 1 Employment pattern](image)

3 A Review of a Number of Specification Patterns

In this section, we introduce and discuss a number of specification patterns. Some of these are based on the data model patterns in [Hay96] and the domain abstractions in [Maiden91]. (In the right upper corner of each object type, there is a symbol denoting its classification according to the general deontic pattern introduced in Section 4.)

Employment Pattern

In Figure 1, a specification pattern for employment is given, where the object type EMPLOYMENT is associated to an ORGANISATION which is the employer, a PERSON who is an employee and one or several POSITION ASSIGNMENTS. A POSITION ASSIGNMENT shows the share of an EMPLOYMENT to a POSITION. For instance, the model has the capacity to represent that a person is employed full-time at a company from a specific date, and that the
employment is shared in 40% as project leader and 60% as consultant. An employment gives rise to a large number of obligations, e.g. that the employee should work a specified amount of time and that the employer should pay a salary.

**Work Order Pattern**

A specification pattern for work orders is given in Figure 2. An example of a *work order* is the order that a certain course shall be given by a department during a specified period of time. The department is then responsible to see to it that the course is given during this time period. Furthermore, a number of parties can be involved in a work order and they can play different roles. Examples of different *work order role types* are head teacher and assistant, where a head teacher is also the examiner, and an assistant helps with teaching and administration to the head teacher. The object type *work order role* is introduced to show which role type a party has in a work order.

Furthermore, a work order can give authorization for a number of activities. An *activity* is for instance a specific lecture. Finally, the object type *activity assignment* is introduced to show who is going to perform a particular activity.

---

**Lending Pattern**

Figure 3 shows a specification pattern for lending. Lending is a process between two parties where one of them releases, for a certain period of time, the rights over an object she owns to the other party. The borrower is then requested to return the object she has lent on time. Modelling this domain we introduce the object types *lending, party*
and LENDING PRODUCT. PARTY is connected to LENDING with two relationships: one for the lender and one for the borrower. The object type LENDING is also connected to the object type LENDING PRODUCT. The attributes for lending are from and until showing the lending period, and lending-rate showing the price for a particular lending, which also can be zero. Furthermore the lending products belong to a PRODUCT CATEGORY. Depending on the PRODUCT CATEGORY the lending-rate can vary. In addition the attribute lending-rate for the object type LENDING represents a price which is the total cost for a lending.

Application Pattern

The next specification pattern concerns applications. In the specification pattern in Figure 4, we generalize the applications for employment, for grants, for a student place at a school, and for a membership in an organization by introducing the object types PLACE, MEMBERSHIP, and GRANT as specialisations of the object type APPOINTMENT. An application has always a PARTY as a candidate, it concerns an APPOINTMENT and is made to a PARTY. It is also common that the applicants are informed whether they got the appointment they applied for or not.

Booking Pattern

It is common that an object must be booked before it can be lent. Bookings are also needed for planned activities. A BOOKING object type specifies the RESOURCES that are booked and the period of time for which they are booked (Figure 5). When a booking is made for a party, there is a commitment that the booked party will join the activity she is booked for. If a booking is made for an equipment, the commitment is that the equipment will be available for the booked period of time. An important characteristic
of a RESOURCE is its competence if it is a party, or its type if it is an equipment. The booking pattern also contains an object type ACTIVITY, which specifies the reason for making a booking. As an example, a lesson is an activity that requires the booking of a teacher and a class room. The RESOURCE NEEDS for an activity type can be defined.

![Booking pattern diagram](image)

**Figure 5** Booking pattern

**Ticket Pattern**

The ticket pattern in Figure 6 can be used for representing situations where a person has a TICKET for some PERFORMANCE, e.g. in a theatre. The person who has obtained a ticket has the right to be present at the performance at the specified place, and the party who has issued the ticket has the obligation to let the person be present. The object type TICKET has an additional attribute advertisement, which describes the advertisement printed on the back of the ticket.

**Marriage Pattern**

The marriage pattern in Figure 7 can represent marriages, the parties involved in marriages, and the events that create and end marriages. A MARRIAGE is created by a WEDDING, which can be seen as a declarative speech act in which two parties, besides the wife and the husband, participate – a priest and a witness.
4 A General Deontic Object Pattern

The patterns in the previous section are typical examples of specification patterns at an intermediate level of abstraction. In this section we will further generalize these patterns into a common pattern. In order to do this, we first classify their object types into different categories. The first distinction made is that between concrete and abstract object types. Concrete object types have instances that are physical objects. PERSON, for instance, is a typical example of a concrete object type. Everything that is not a concrete object is an abstract object. An example of an abstract object type is ORGANISATION which has Stockholms University as an instance. Another example of an abstract object type is EMPLOYMENT.

An important characteristic of certain objects is that they entail obligations. For instance, Peter’s employment at Stockholms University entails that Peter is obliged to work a number of hours for the university which in turn is obliged to pay him salary. In contrast, the existence of the object Stockholms University does not by itself entail any obligations. Abstract object types that entail obligations are called deontic object types. Examples of deontic object types are WORK ORDER, ACTIVITY ASSIGNMENT (Figure 2), LENDING (Figure 3), APPLICATION (Figure 4), BOOKING (Figure 5), and MARRIAGE (Figure 7). A common feature of all the patterns in Section 3 is that their central component is a deontic object type. A generalization of these patterns, a general deontic pattern, is shown in Figure 8. In the right upper corner of each object type an abbreviation of the name is given. This abbreviation is used to classify the object types from the patterns in the previous section as specialisations of the object types in Figure 8. The deontic specification pattern contains the following components:

**Figure 6 Ticket Pattern**

**Figure 7 Marriage Pattern**
Deontic objects are objects that group together obligations for different parties. Typical properties for a deontic object are the period for which it exists and a number of parameters representing important information about the created obligations. For instance, the attributes start date, percent and salary for the object type employment in Figure 1 represent information connected to the entailed obligations, namely the start date for the obligations, the working time which an employee is required to do, and the salary which the employer has to pay.

We furthermore divide the deontic object into atomic and composed. The reason for this is that some deontic objects are represented by means of others. An example of modelling deontic object types in this way can be found in the Employment pattern (Figure 1), where the object type employment is an instance of the object type composed and position assignment is an instance of atomic.
**ROLE** and **ROLE DESCRIPTION**

A deontic object usually concerns more than one party, as most obligations are directed from one party to another, [Herrestad95]. As a consequence of this, **DEONTIC OBJECT** is always associated to the object type **PARTY** with at least two different relationships. In the employment pattern, these relationships show the employee and the employer. In the lending pattern, they show the lender and the borrower. However, a deontic object may involve more than two parties as well. To build a general model which can represent a flexible number of parties involved in a deontic object we use the object types **ROLE** and **ROLE DESCRIPTION**. **ROLE DESCRIPTION** brings general information about each particular **ROLE**, while **ROLE** shows who is playing a role and for which time.

**SUBJECT MATTER**

The subject matter of the obligations of a deontic object are represented by the object type **SUBJECT MATTER**. The instances of **SUBJECT MATTER** can be classified into concrete and abstract. An example where the subject matter for the deontic object is a concrete object type is **LENDING PRODUCT** in the Lending pattern (Figure 3). In contrast, in the Work order pattern (Figure 2) the subject matter of the deontic object type **ACTIVITY ASSIGNMENT** is the abstract object type **ACTIVITY**.

**CREATION** and **DELETION**

The object types **CREATION** and **DELETION** model the creation and deletion of deontic objects. Consider for instance the Marriage pattern where the deontic object type **MARRIAGE** is connected to **WEDDING** - the object type representing the creation of a marriage. Moreover a marriage can be ended up by a divorce, which is represented by the corresponding object type. The creation and deletion of a deontic object are performed by means of speech acts; in order to model persons performing such speech acts, additional relationships to **PARTY** are required.

**REPRESENTATION**

The deontic pattern includes an object type **REPRESENTATION**, which models the physical representation of a deontic object. The reason for introducing this object type is that in certain situations the physical representation has properties which may be of interest. An example can be found in the Ticket pattern (Figure 6), where the object type **TICKET** is in fact the physical representation of a booking. Introducing this physical ob-
ject is motivated by the attribute advertisement for a ticket, which brings information about the ticket as a piece of paper and not the booking it represents.

**GOAL**

The purpose of the object type **GOAL** is to represent the reason for the existence of a deontic object. The existence of a deontic object needs to be motivated when it is created under certain conditions. For instance, when a head teacher books an assistant for her course (see the Booking pattern), the reason for doing so is to make it possible to carry out some activity. In such cases it is common to keep information not only for a booking but also about the activity which is the reason for the booking.

All the specification patterns in Section 3 can be seen as specialisations of the deontic pattern introduced above. However, most of these patterns do not include all of the components of the deontic pattern – some components are omitted or have been collapsed into other components. Some of the most common omissions are the following:

- **ROLE** omitted. The object types **ROLE** and **ROLE DESCRIPTION** are often omitted and replaced by relationships between **DEONTIC OBJECT** and **PARTY**. This omission is common when there is only a small and fixed number of established roles, e.g. wife and husband in a marriage, cf. Figure 7, where these roles are modelled by means of the relationships wife and husband. When the number of roles may vary or when information is needed about the roles, the object types **ROLE** and **ROLE DESCRIPTION** are required.

- **PARTY** omitted. In some cases the role of some party is omitted. An example of this is given in Figure 2, where only one role in **ACTIVITY ASSIGNMENT** is modelled – the performer’s role. However, there exists also another role, that of the party who has assigned the activity. The reason that this role is not modelled is that the model is constructed from a certain perspective, in this case from the perspective of the company that assigns activities. In situations like these, where the model is built from a particular point of view, one party is implicit and the roles for this party can be omitted.

- **REPRESENTATION** omitted. The object type **REPRESENTATION** is frequently omitted from schemas. In most cases, only the deontic object is of interest and not its physical representation. A representation object is needed only if it has some physical characteristic of interest. In the patterns of Section 3, this holds only for the **TICKET** object type.
• **DEON**TIC OBJECT omitted. The object type **DEONTIC OBJECT** may be omitted and collapsed into another object type, usually **REPRESENTATION**. An example of this is given in Figure 6, where a deontic object type for booking has been collapsed into the type **TICKET**. The attributes **time** and **place** belong to the deontic object type, whereas **advertisement** belongs to the representation object type.

• **SUBJECT MATTER** omitted. The object type **SUBJECT MATTER** is omitted when the deontic object does not primarily entail obligations for a particular action or object. For example, a marriage entails a large number of obligations in many different circumstances, and it is not possible to single out a specific obligation as more important than all the others. This is why the object type **SUBJECT MATTER** does not occur in the **MARRIAGE** pattern in Figure 7. A work order, on the other hand, entails primarily the obligation to carry out a particular activity, while the other obligations in this context are less important; so the object type **ACTIVITY** is included in the **WORK ORDER** pattern.

5 A Classification of Deontic Objects

In the previous section, a number of conceptual patterns was generalised by considering them from the deontic perspective. The concept deontic object type was introduced to cover objects that entail obligations. Furthermore, a number of object types related to a deontic object was identified and described, in order to give a better understanding for the concept of deontic object. The generalisation of deontic objects made in section 4 emphasizes the similarities between different deontic objects. To complete the analysis, we investigate in this section the differences between the deontic objects and classify them into sub-categories.

Since the existence of obligations is the most important characteristic of a deontic object, we first consider the obligations that a deontic object entails. For instance, we agreed for the lending pattern in section 3 that when lending a product the borrower is obliged to return it to the lender. Another example is when an order has been placed by a customer and has been accepted by the supplier. The supplier is then obliged to deliver the ordered product and the client is obliged to pay for it. In these two examples it was possible to exactly specify the obligations entailed by the deontic objects. We call such obligations **core obligations**.

However, it is not equally simple to exactly specify the obligations entailed by certain other deontic objects. There is no doubt that an employment, for instance, entails obligations, but it is almost impossible to count all entailed obligations. Besides,
the obligations can differ for different employments, which make them situation dependent. In this case we can not simply point to a set of core obligations. We can of course generally say that an employment binds an employer and an employee to follow a number of rules, which are either clearly specified in the contract, are parts of laws or which even can be implicit without being specified. But what the last sentence says is in fact that one is obliged to perform his/her obligations. Since this utterance is tautological it can be applied to all deontic objects and is thereby nothing which can be used for distinguishing them from each other. Consequently, the problem of not being able to specify some core obligations for an employment remains, which results in the first distinction we draw between deontic objects, namely: deontic objects with core obligations and deontic objects with no core obligations. Lending and ordering are then examples of the first category and employment is an example of the second category.

**Deontic Objects with Core Obligations**

In this section we will determine a number of aspects that distinguish deontic objects from each other. Consider for instance the deontic object of giving a quotation. A quotation obliges to delivery if an order according to this quotation has been made. But if no order is made, the deontic object does not entail any obligation. So the obligation exists only when certain conditions are fulfilled. We can, however, not identify any conditions for lending and ordering examples, as they always entail obligations independently of the circumstances. This observation results in distinguishing between conditional and unconditional deontic objects, where a quotation is a conditional deontic objects and lending and ordering are unconditional ones.

Considering quotation, ordering, and lending once again one more difference can be identified, namely how the obligations are distributed over the involved parties. For instance, lending a product brings obligations to only one of the involved parties: to the borrower to return the borrowed product. Even in giving a quotation the entailed obligations concern only one of the parties. In contrast, an order brings obligations to both parties: it obliges one of them to deliver and the other one to pay the delivered product. These considerations motivate a distinction between deontic objects that bring obligations to one party and deontic objects that bring obligations to all parties.

Figure 9 depicts a tree the leaves of which are deontic objects. The right-hand side of the tree refers to deontic objects with core obligations. It remains to discuss the left-hand side of the tree covering deontic objects with no core obligations.
Deontic Objects with no Core Obligations

Deontic objects classified as not having core obligations are in fact some kind of contracts with a very large number of rules entailing different obligations in different situations. A characteristic of a contract is that it binds several parties to an agreement, i.e. it brings obligations to all parities involved in the contract.

Furthermore, an interesting aspect is how and by whom a contract can be revoked, rather than each simple possible obligation involved in it. A remarkable example in this sense is a Greek citizenship which is a deontic object since it entails a number of obligations but which can not be revoked. Even if it is quite an extreme situation as most deontic objects can be revoked in one way or another, we make a distinction between revokable and not revokable deontic objects.

Moreover, not all of the revokable deontic objects are revokable by all parties. For instance, a citizenship, other than a Greek one, can be revoked only by the state that has issued it. In contrast, an employment can be revoked by both the employer and the employee. So we divide the revocable deontic objects into revocable by one party and revocable by all parties.

However, sometimes the possibilities to revoke a contract are limited. For instance a tenancy right in Sweden can be revoked by the tenant whenever he wants, but it can only under certain circumstances be revoked by the landlord. An example of such circumstances is when the tenant has not paid the rental charge. So the contract can be revoked by one of the parties only under some well stated conditions. This gives rise to a distinction between conditionally revocable and unconditionally revocable deontic objects which are revocable by all parties as well as if they are conditionally revocable for one or conditionally revocable for all of the parties.

Finally we will consider the time aspect, namely how a deontic object with no core obligations is placed in time. We have distinguished between time limited and not time limited deontic objects. Examples of not time limited contracts are citizenship, marriage, tenancy rights. Examples of time limited contracts are employment limited by the laws to the age of 65 years, pension which starts from 65 years and also one year compulsory military service for all men at age of 19. Even the distinction that some of the time limited deontic objects are limited in the beginning, some of them are limited at the end and some of them are limited both at the beginning and at the end can be made (see Figure 9).

The reason that the time limitation aspect was not considered for the deontic objects with core obligations is that we consider core obligations always to be limited in time. An obligation is in our approach something that results in a punishment or a
compensating action if it has not been fulfilled. A time limit is therefore necessarily, to make it possible to decide whether an obligation has or has not been fulfilled.

![Deontic Object Tree](image)

**Figure 9** Deontic Object Tree

## 6 Summary and Further Research

The deontic pattern introduced in this paper can serve as a very abstract specification pattern in a library of reusable modules. The other modules will contain specialisations of the deontic pattern, such as lending, booking, and sale. The deontic pattern will serve as an entry point to the library and as a template for structuring the other patterns. An open research question is how to organize the specialisations of the deontic pattern into a structure that is easy to navigate and search. The tree-structure in section 5 is one possible solution. When a designer wants to identify a specification pattern for a particular application, she can start at the top of the tree and transcend it downwards by choosing the appropriate arcs with respect to the application. When she has arrived at a leaf she will find a specification pattern that is directly, or with small modifications, useful for the application.

The deontic pattern may be used in schema specification in many different ways. First, one of its specialisations can be integrated directly into a schema without any
modifications. Secondly, a fragment of a schema can be compared to the deontic pattern in order to check the correctness and completeness of the schema. Utilizing the deontic pattern in these manners can improve the quality of schema specification in several respects. In particular, the completeness of a schema can be improved, as comparing a schema fragment to a pattern will assist a designer in identifying aspects of the application that have been left out in the specification. Furthermore, the stability of a schema can be increased by adding components from the deontic pattern, which are not strictly necessary for the current application, but can make later requirements easier to accommodate. The use of the deontic pattern can also support the designer in focusing on the conceptually relevant aspects of an application. For example, a common error in schema specification is to model the physical representation of a deontic object instead of the deontic object itself. Using the deontic pattern will make the distinction clear and prevent this kind of error. Finally, the deontic pattern can facilitate the documentation of a schema by providing templates for the different types of obligations that may occur. Instantiating these templates for a particular application can be a most effective way of constructing a comprehensive documentation.

The specification pattern introduced in this paper is described solely in terms of objects and relationships, and it therefore provides only a superficial representation of deontic structures. In order to provide a deeper representation, the deontic structures must be described by means of complementary formalisms. Adequate formalisms for this purpose may range from DEMO models, [Dietz92], to illocutionary and deontic logics, [Assenova96], [Johannesson98], [Dignum95]. Utilizing these formalisms, some of the vague notions in the deontic pattern, in particular subject matter and goal, can be made more precise. Furthermore, it will become possible to analyse in greater detail the different variants of the deontic pattern, and thereby construct a systematic structure of these variants which will help to build a library of reusable specifications. The deontic object tree in Figure 9 is only a first step in such an analysis. Another line of research is to empirically investigate the usefulness of the deontic pattern. Such an investigation would focus on two distinct issues. First, the applicability of the deontic pattern should be measured by studying how frequently it occurs in applications from different domains. Secondly, one should investigate how well the deontic pattern supports a systems designer in the specification task - this can be done by comparing designers that are familiar with the deontic pattern with those that are not with respect to results and ways of working.
References


[Gamma95] E. Gamma, R. Helm, R. Johnson and J. Vlissides, Design Patterns, Addison-Wesley, 1995.


