Conceptual Patterns for Reuse in Information Systems Analysis

Petia Wohed

Department of Information and Systems Sciences
Stockholm University/Royal Institute of Technology
Electrum 230, 164 40 Kista, Sweden
petia@dsv.su.se

Abstract: Reuse of already existing resources and solutions has always been a strategy for reducing the costs in the information systems development process. Construction and organization of small pieces of reusable solutions, also called patterns, in libraries for reuse support, has taken a central place within research during the last years. In this paper, a methodology for collecting conceptual patterns and a navigation structure for suggesting the most suitable one during the information systems analysis process are suggested. The study has, so far, been carried out on one domain only, but it provides a theoretical background for research on other domains as well.

1 Introduction

The concept of patterns is increasingly used in the context of information systems development. A pattern is a piece of a solution for a problem reusable in different situations. The main purpose of the use and reuse of patterns is to reduce the costs for the design and implementation of information systems. The idea of reuse of different parts of a solution is also at the heart of the approaches of object orientation and component-based development. These paradigms become more and more commonly used and are widely spread in the sphere of implementation, a line of development clearly distinguished by the character of the current environments for standard programming languages providing large libraries of reusable objects.

However, the concept of pattern is still not widely used within the initial phases of information systems development, i.e. analysis and design. One of the problems may lie in the lack of standardised patterns collected in libraries and supported by development tools. Another problem may lie in the lack of suitable patterns. In order to make a pattern applicable in several different situations it often tends to result in a too general pattern to be easily applicable, and such a general pattern is sometimes even difficult to understand.

In this paper, both these problems are addressed for the area of information systems analysis. It provides a theoretical foundation for a modelling wizard tool, which collects a number of conceptual patterns as well as navigates the designers to the most suitable one for a particular situation. Such a tool, supporting the initial phases of the database design process, is an attempt to complement database management systems like Microsoft Access, or as a part of a visual modelling tool

B. Wangler, L. Bergman (Eds.): CAiSE 2000, LNCS 1789, pp. 157-175, 2000

supporting an OO-development process of information systems like Rational Rose. A prototype for such a wizard collecting more than 360 patterns has been developed.

The difference from earlier approaches is the idea of preserving a number of different patterns within the same domain, in order to keep them concrete and easy to use. In order to identify the necessary pattern, a navigation structure consisting of a number of queries, to be answered by the designer, is suggested. This navigation structure may also be considered as a method for modelling support and is the main contribution of this work.

The paper is organised as follows. The next section surveys related research in the area. In Section 3, the modelling formalism is defined. In Section 4, a number of patterns within the same domain are analysed and a number of questions are distinguished outlining the differences between the patterns. Section 5 gives an example of the behaviour for a domain dependent modelling wizard tool built by the questions received from Section 4. Section 6 gives a theoretical background for the identified questions. Finally, Section 7 concludes the paper and gives directions for further work.

2 Related Research

The identification, naming, and classification of different phenomena in a Universe of Discourse (UoD) have long been incorporated in the discipline of taxonomy. One of the oldest and most well known taxonomy is Linné's taxonomy for species. However, when dealing with information systems it is not sufficient to only identify and classify the relevant phenomena in the UoD for which an information system has to be built, but it is also necessary to specify their properties and relations, something characterising the discipline of ontology. Slightly simplified and inspired by [10] and [4] we consider an ontology as a conceptualisation of the UoD in a certain language (see [10] for a complete definition of the concept). One of the main reasons for building ontologies is the reuse capability they provide. Some examples of ontologies are CYC [6] developed by Cycorp, The Enterprise Ontology [19] developed at the University of Edinburg, and TOVE (Toronto Virtual Enterprise) [21] developed by the University of Toronto. Common for all of these three ontologies is that they conceptualise the organisational structure of enterprises and are meant to be used for support in the information systems design process. To position our work in the context of the work provided by the ontology engineers, we will refer back to Guarino's framework for ontology driven information systems design [10], where two dimensions are identified. The first one is the temporal dimension distinguishing between the development-time and run-time for a system, and the second one is the structural dimension consisting of the three components of an IS namely application programs, information resources (i.e. databases and knowledge bases), and user interfaces. Hence, the focus of this paper is on the development-time for databases, and the work reported here may be considered as an application of ontology for the purpose of reuse.

Parallel whit the work of ontology, work in the field of reusable patterns has been carried out. Some of the works relevant here is the one provided by Maiden in his doctoral thesis [16], concerning the question of specification reuse. His paradigm builds on reuse by analogies. A number of domain abstractions are identified and

formalised using a structure called meta-schema that defines seven knowledge types as necessary for a requirement specification. The source domain is also specified using this meta-schema. A comparison of the knowledge types of the source domain model with those of the domain abstraction is then possible and provides an analogy analysis, meant to identify possibilities for reuse. Even if Maiden does not call his domain abstractions 'patterns', the idea is still reuse of already constructed specifications. However, the work presented in this paper differs from the work provided by Maiden by concentrating only on the conceptual patterns and not on the whole requirement specification. It also differs by suggesting a navigation structure for identifying the most suitable pattern.

Two other works on patterns, during recent years, have been provided by Fowler [8] and Hay [12], who, after analysing a number of domains, suggest conceptual patterns meant to guide inexperienced systems developers or simply as starting points for those more experienced. However, no navigation support has been discussed by them, so the work reported here may be considered as a continuation of the work done by Fowler and Hay, through the investigation on navigation support.

When talking about patterns, two more works have to be mentioned: the work provided by Coad et al [5] and the work provided by Gamma et al [9]. Each of these works provides a library of patterns that are usually referred to. Goad's library consists of analysis patterns, whereas Gamma's library consists of design patterns. Coad's patterns are quite general, often consisting of no more than two, three classes, and they are meant to be varying consolidated into different object models. Furthermore Coad selects a number of strategies that are aimed to guide a designer in the analysis process. However these strategies are not formalized in the sense that they easily can be implemented in a computer, which is also the difference from our approach. Furthermore, the patterns suggested by Gamma are design patterns and they are meat to support the design process, whereas the patterns analyzed in our work are analysis patterns. The difference between analysis contra design patterns is the perspective from which the patterns are build; analysis: specifying the requirements for a system; and design: specifying the solutions for a system.

An alternative work on creating libraries of reusable objects also partly inspired by Fowler [8] and Coad [5] is the work provided by Han, Purao and Storey [11]. The point of departure in their work is so called design fragments - reusable solutions with a lower granularity than domain models, but higher granularity and specificity than analysis patterns. The main results are (a) a methodology for building such design fragments collected in a repository, and (b) two clustering algorithms for identifying common pattern set starting from a natural language requirements specification, which may be used as a navigation structure for identifying possible starting points for a designer. The work reported here may be considered as a complement to the work provided by Han et al. The main differences are, however, that: (a) the navigation structure by Han et al starts with a requirement, something not required in our work; and (b) the navigation structure may suggest more than one design fragment as a relevant starting point for a designer, whereas in our approach only one pattern is suggested.

Finally, viewed from a traditional knowledge engineering perspective, this work builds on the work provided by Moulin and Creasy [18] who integrates the earlier work of Chen [3] and Sowa [20] by adding linguistic elements from Fillmore's case grammar [7]. Moulin and Creasy present a simplified picture of the design process retyped in Figure 1, where the dotted rectangle encircles the phases being subject for

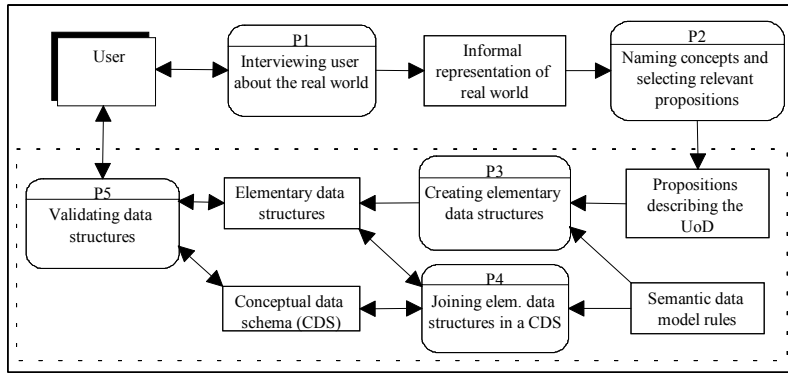


Fig. 1. Modelling the designer's behaviour (retyped from [18])

automation in modern modelling tools. Our approach builds partly on Moulin's and Creasy's ideas for using linguistic elements on conceptual modelling. The difference, however, is that we investigate how to automate the processes that have not been automated earlier, i.e. the processes outside the dotted rectangle. Consequently, we focus on the process P1 'Interviewing user about the real word' (from Figure 1) in order to support the users in building conceptual schemas, or, as it also can be considered, in order to guide the users to find the most suitable pattern. This shift of focus changes slightly the design process as represented in Figure 2 where the modelling process is simplified by the use of patterns. A modelling wizard tool is intended to support the interviewing process P1 and build a conceptual schema gradually, by using a pattern library. During this process the user shall be able to rename and complete the proposed solution, i.e. process P2. Furthermore, an integration module (supporting the processes P3 and P4 in Figure 2) has to be incorporated with the modelling wizard tool in order to offer integration facilities and make it possible to consolidate local schemas into a global one.

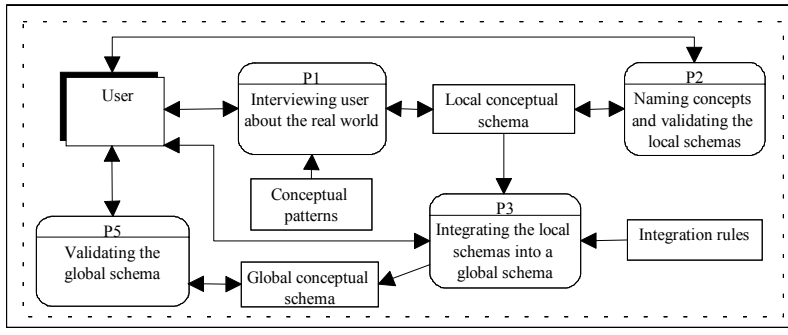


Fig. 2. The process model for our approach

3 Conceptual Schemas

In this section we briefly introduce the modelling language which is used. A formal definition of it, based on the definitions represented in [14] and [15] may be found in the first part of the Appendix.

The basic construct in conceptual modelling approaches is the object. Objects that are similar to each other are grouped together into object types, such as Person and Country. Objects have different properties, which are modelled by attributes, and they can be related to each other, which is represented by relationships. In our graphical notation (see Figure 3) object types are represented by rectangles, attributes by bulleted lists inside the object types, and relationships by labelled arrows. The object type initiating a relationship is called the domain of that relationship and the object type in its end is called the range. Generalisation constraints are shown by dotted arrows where the head of an arrow points to the super-type. For each relationship, the mapping constraints specify whether it is single-valued, injective, total or surjective. A relationship is single-valued when each instance of its domain is connected to at most one instance of its range. A relationship that is not single-valued is multi-valued. A relationship is total when each instance of its domain is connected to at least one instance in its range. A relationship that is not total is partial. A relationship is injective (surjective) when its inverse is single valued (total). Since most of the relationships in our schemas are single valued, not injective, total and not surjective, only the cardinality constraints that differ from this uniformity will be specified explicitly.

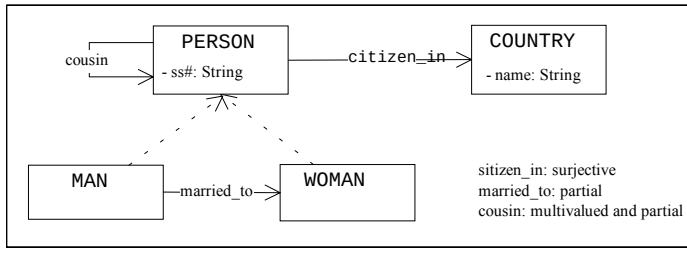


Fig. 3. Example of a conceptual schema

The description of the domain represented by the model in Figure 3 is as follows: a `PERSON` is a `citizen_in` a `COUNTRY` and (s)he may have several `cousins`. The distinction between `MAN` and `WOMAN` is kept. A person may only `marry_to` someone with the opposite sex, and polygamy is only allowed for women.

4 The Booking Domain

In this section a number of different patterns in the booking domain are considered. The similarities and differences between them are analysed and a number of questions

are constructed for gathering information about the specific features within the domain.

The analysed patterns are shown in Figure 4 and most of them may even be considered as special cases of Fowler's 'Resource Allocation' pattern [8] pp. 168-172. The conceptual schemas may be used within the corresponding enterprises: a) a dentist cabinet; b) a university; c) a chain of movie theatres; and d) a travel agency. Even if all these examples are taken from the same domain the only common thing in their conceptual schemas, at a first glance at least, seems to be the `BOOKING` object type, which occurs all over. The rest of the schemas differ from each other considerably.

One main difference is that in schema a) a booking may concern only one object, whereas in schema b) and d) a booking may concern several different objects, which the object type `BOOKING_LINE_ITEM` with its relationship to `BOOKING` shows. With respect to this difference, the following question may be placed in order to identify which of those two alternatives pertains in a particular situation.

Does a booking consist of one object, or may it consist of several objects?

Notable is that even if several tickets may be booked within the same booking in schema c) the construction is more similar to the one in schema a) where only one object is booked at a time, but not to those in b) and d) where several objects are booked. Hence, the construction of schema c) also depends on the fact that exactly the same kind of objects are booked which leads further to the next identified difference, namely the difference between the booked object. Schema a) and b) represent the booking of a dentist and a room, correspondingly. In both these cases the booked thing is a concrete object, the dentist Peter or the room 605 at DSV. In contrast the schemas c) and d) do not book a concrete object(s), but they rather describe the kind of object that is booked. I.e. the booking of a flight ticket for flight SA345 on Friday 7 May 99, economy class only guarantees that one ticket but not any particular one is booked. The question resulting from this difference is:

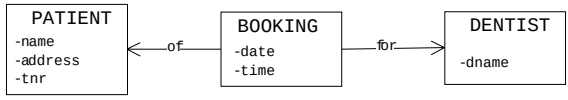
Does a booking concern a (number of) concrete object(s), or does it rather specify the character of the object(s)?

Analysing further, one of the schemas – schema d) represents two different kinds of bookings: hotel room bookings and flight ticket bookings, which differ from each other by their properties. This is shown by specialising the object type `BOOKING_LINE_ITEM` into `HOTEL_BOOKING` and `FLIGHT_BOOKING`. In the rest of the schemas the bookings belong to the same category. In order to identify whether the bookings objects are from the same category or not, the following question may be placed:

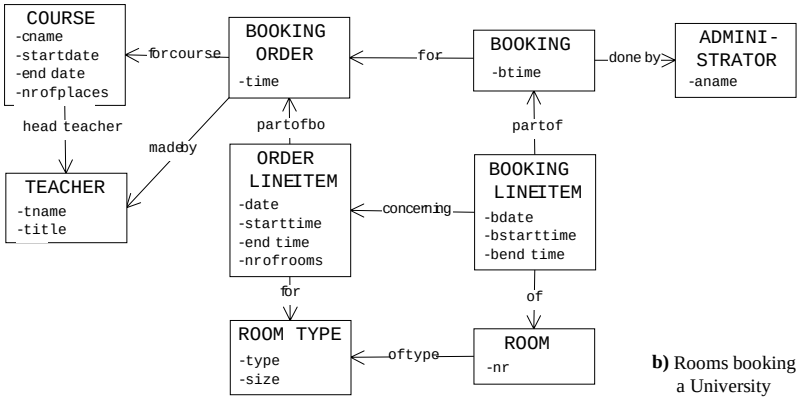
Do all the bookings have the same character or may they be divided into several categories?

Continuing the observations, it also can be noted that in schema b) information about the booking orders is kept by the object types `BOOKING_ORDER` and `ORDER_LINE_ITEM`. This distinguishes b) from the rest of the schemas and gives rise to the next question:

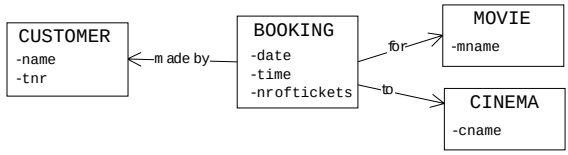
Is it necessary to keep information about the request for a booking, before making the booking?



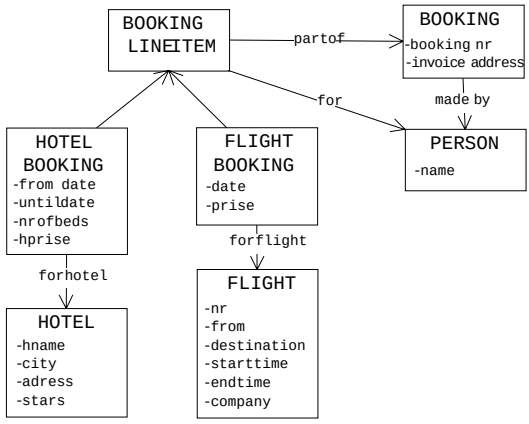
a) Booking a time for the dentist



b) Rooms booking at a University



c) Booking movie tickets



d) Booking a trip in a Travel Agency

Fig. 4. A number of booking patterns

Furthermore, in schema b) a booking order has to be motivated by giving the course for which a booking order is placed. Such a motivation for a booking is not necessary and has not been modeled by the rest of the schemas. The relevant question for identifying this difference is:

Does a motivation need to be given for a booking/booking request?

Finally, in schema d) information about the person who made a booking is kept as well as the information about the person, for whom the booking is made. This fact results in our next question.

May a booking/booking request be done on behalf of someone else?

5 A Modeling Wizard Tool for the Booking Domain

A domain dependent modelling wizard tool for the booking domain can now easily be constructed using the questions identified in the previous section. In this section we exemplify such a tool. The scenario is that the user shall answer the questions identified in the previous section one after another, and after each answer the tool shall build and refine the conceptual schema according to the answer. It is of course desirable that the user shall be able to rename entity types and relationships continuously during the questionnaire process as shown in the process model in Figure 2, but we leave these details for later work and only exemplify the behaviour for such a tool here. The detailed logic for the tool can be found in the second part of the Appendix.

Suppose that we want to extend a library database for a university with booking facilities in order to keep information about the students in the booking queue for a book. On the first question:

1. Does a booking consist
   a) one object, or
   b) may it consist of several objects?

if we choose alternative a), the wizard tool should suggest a schema like this in Figure 5a) and ask the next question. If our answer is alternative b) the tool should draw the schema in Figure 5b) and still ask the next question.

2. Does a booking concern a (number of)
   a) concrete object(s), or
   b) does it rather specify the character of the object(s)?

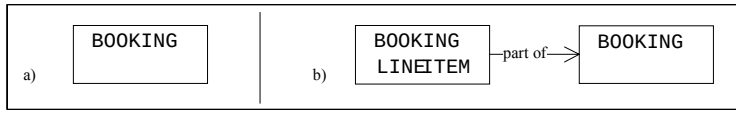


Fig. 5. The alternative solutions after the first question

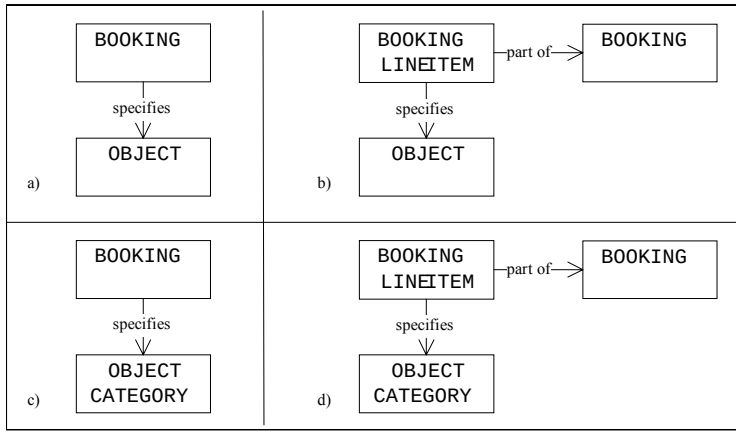


Fig. 6. The alternative solutions after the second question

An answer a) should result in expanding the schema into the one shown in 6a) or b), depending on the point of the departure, which is the result from our previous answer. The answer b) should result in the schema from 6c) or d).

In this way, the wizard tool should interview the user and piece by piece should build a schema depending on the answers. So, if the answers give the information that a booking consists of several objects, that the bookings have two different characters, that both order and motivation for a booking are necessarily to keep information about, and that a booking can not be done on behalf of someone else (se the right part of Figure 7), the resulting schema suggested by the wizard should be the one shown in Figure 7. (When implementing the prototype, the UML notation was used for object types, relationships, and generalization constraints for practical reasons. For the mapping constraints the notation used by Johannesson and explained in the appendix was kept.)

6 Completeness of the Questions

In the previous two sections, we analysed one domain and showed how a tool may support the modelling process within this domain. An important issue is to investigate the completeness of the questions, i.e. whether they can be used for gathering all information for the analysed domain, necessary for the construction of a satisfactory conceptual schema.

To reason about the completeness of the questions we divide them into different categories and analyse whether each category is covered by the questions belonging to it and whether the categories are suitable for discussing the completeness of the analysis. The following four categories are identified:

- questions with case grammar character;
- questions for cardinality identification;
- questions for power types identification;
- questions for generalisation/specification constraints.

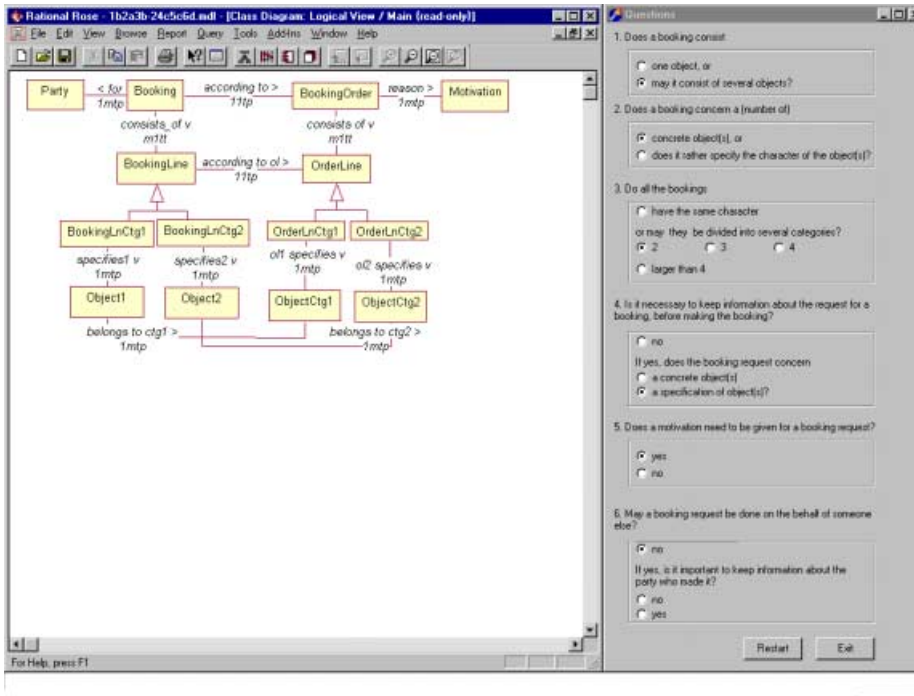


Fig. 7. A screen shot of the modelling wizard

Questions with case grammar character: The concept of case here refers to what usually has been called a deep case, i.e. the categorisation of noun phrases to the conceptual role they play in an action described by a sentence. Some examples of cases are agent: one who performs the action; recipient: one for whom the action is performed; object: one who suffers the action; instrument: an object used to perform the action; location: the place where the action is performed. The set of the cases is obviously discussible, which has resulted in a number of case systems, se [2] for a survey. However, the choice of a case system is not critical in this phase, as the focus here is to show how a case system may be used and also further tailored in order to make it suitable for modelling purposes.

Turning now to the way cases are used in conceptual modelling, it may generally be said that they are used for semantically enrichment of the schemas. (See [18] for a discussion about the advantages of using case grammar within conceptual modelling and [13] for a concrete application based on this semantically enrichment of conceptual schemas.) Briefly, applying case grammar to conceptual modelling implies that the object types in a schema represent the verb and noun phrases, and the relationships between the object types represent the cases. One restriction is that the domains for the relationships are only object types representing the verb phrases. The semantically enrichment results from the categorisation of the noun phrases into cases. It also should be noted that such a categorisation of noun phrases requires that an action have been described in some way. Hence, the case grammar is only suitable for schemas describing actions and not for schemas describing, for instance, product

structures. With this limitation in mind we are going back to one of our conceptual schemas in Figure 4 and applying case grammar theory to it.

Figure 4c) describes that: "A number of tickets for a movie to a cinema may be booked by a person". Example of information stored in an information base for this schema (see Figure 8) is: "Peter has booked two tickets for the movie 'Lecture on Conceptual Modelling' to the cinema Sergel for the 2 of June '99 six o'clock" i.e. `{booking(b1),date(b1,990602),time(b1,'18.00'),nr_of_tickets(b1,2),customer(c1),name(c1,'Peter'),movie(m1),mname(m1,'Lecture on conceptual modelling'),cinema(cn1),cname(c1,'Sergel'),made_by(b1,c1),for(b1,m1), to(b1,cn1)}`. Analysing now the sentence describing the schema and the schema itself, we note that: the action 'tickets may be booked' is represented by the object type `BOOKING`; and that the rest of the noun phrases i.e. person, movie, and cinema are represented by the object types `CUSTOMER, MOVIE` and `CINEMA`, correspondingly. The cases for the noun phrases 'person', 'movie', and 'cinema' are 'agent', 'object', and 'location', correspondingly, and they are captured by the relationships `made_by`, `for` and `to`, correspondingly. In Moulin and Creasy's notation the cases are explicitly showed in a schema, something we have not used in our modelling language. However, the cases may be used as automatically generated names for the relationships suggested by a tool, which the users shall be able to change afterwards.

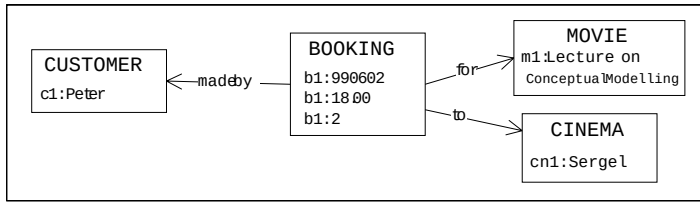


Fig. 8. Instance example for the schema from 4c)

The question is then which of the cases are relevant for a domain. The only way to give an answer to this question is to analyse a number of different patterns within a domain, and investigate the cases that have been used in those patterns, similarly to the analysis we provided for the booking domain. The booking schemas in Figure 4 ware selected to represent some (as much as possible) distinct booking situations we have encountered. Concluding the analysis for this domain, we can point out that the cases occurring in all schemas are 'object', 'agent' and the 'time for which a booking is made'. The 'recipient' is not explicitly modelled since it represents the system, for which the analysis is provided. Neither seems the 'instrument' be a relevant case for this domain. Instead cases for 'order' and 'motivation' for a booking may be of importance as well as cases for 'location where the booking is valid' and the 'time when a booking is made'. Since these cases occur in some of the schemas only, we will call them for *optional cases* for the booking domain. Looking back at our questions, we identify the last three questions as questions for identifying whether an optional case shall be included in a particular schema or not. We also note that the set of questions is not complete since it does not cover all optional cases. In order to get a complete set of questions for a domain, questions for all identified optional cases have to be included.

Questions for cardinality identification: Going the other way around and analysing the first question, we can se that it does not belong to the previous category. It does not address the conceptual role of the noun phrases, but is rather meant to bring information about the cardinality for the modelled object. Hence we classify this question as a question for cardinality identification.

Questions for power types identification: Into this category we classify the second question. The terminology of power types within conceptual modelling was introduced by Martin and Odell [17]. One example of a power type is the object types `BOOK` with the properties `author` and `copyright_owner` representing the general existence of an object. In contrast the object type `COPY`, related to `BOOK`, with the property `exemplar_number`, contains the physical objects and represents the materialisation of the `BOOK`. The distinction of power types has also explicitly been done by Fowler [8], who even divides his schemas into two levels: operational and knowledge level, in order to sort each object type to the correct level.

Questions for generalistion/specification constraints: Since the third question is placed in order to identify whether a specialisation hierarchy shall be introduced to the schema or not, we classify it into this category. Historically, the first formalisms for conceptual modelling (called entity relationship languages) did not had any generalisation/specification facilities. However, the formalisms were soon extended to even capture possibilities to represent this kind of constraints which was considered as a comfortable way for representing some parts of the UoD. The modelling languages with such capabilities were called languages for extended entity relationships diagrams. Since the language we are using have these facilities, it is natural to investigate whether there is any part of our domain that with advantage can be modelled using generalisation/specialisation structures.

Summarising the discussion, the first question category i.e. questions with case grammar character comes from the ideas for using linguistic instruments for enriching conceptual schemas. We have also pointed out how the completeness of the questions within this category can be improved by taking into account the relevant cases. The three other categories are closely related to the specific features of conceptual modelling. A review of five object structures that frequently occur within conceptual modelling is given in [1] (pp 125-131). Our last three aspects cover three of the reviewed in [1] object structures. The other two reviewed structures in [1] are not relevant for this work due to the limitation done on domains including actions by introducing the case grammar.

7 Conclusions and Further Research

In this paper, we have investigated and proposed a method for supporting the process of conceptual modelling. The ambition has been that this support shall be automated, i.e. a modelling wizard tool built for asking a number of questions and suggesting conceptual schema patterns, shall implement the results from this work.

As a first step, an analysis on an example domain is provided. A number of different conceptual patterns, for the booking domain, are considered in order to identify the similarities and differences between them. The differences are used for

constructing a number of questions intended to support the modelling process within this domain. The logic for a domain dependent wizard tool is proposed in order to verify the possibility for automating such support. We also reason about the nature of the proposed questions, and whether they are complete and suitable for the purpose they were created for. The main idea has been the usage of case grammar in questions construction, but question covering some other conceptual modelling features have also been necessarily. In order to verify the results from this work a prototype for a modelling wizard based on the results from it has been developed. The prototype implements the booking domain only, but it is considered as a necessary step before continuing the work on a domain independent wizard.

Beside the work with the evaluation of the prototype, more theoretical work needs to be done in order to be able to build a domain independent wizard, namely an investigation on the cases occurring in different domains. Domain specific questions, as well as domain independent questions, have to be identified and constructed. Also, the different solutions or patterns, which shall be proposed by a tool, have to be specified, similarly to those proposed here for the booking domain wizard tool.

Furthermore, it is also necessary to provide research on how a general tool shall be constructed in order to support the modelling of complex UoD ranging over, not only one but, several different domains. One possibility should be to equip the modelling wizard with schema integration facilities. Still a number of questions should need to be considered in order to co-ordinate the behaviour of the tool with the behaviour of a user. One such question is to investigate the human's techniques when modelling complex domains and her way of dividing complex problems into smaller parts, either to make them easily to grasp and solve, or to distribute the responsibilities between different parties.

Another direction for further research is work on attributes that are not cases. For instance, name and address for a person are usually not considered as cases, but rather as simply properties. Then the case grammar should not help us identifying these attributes, but it still should be useful if a modelling wizard tool is able to suggest the most commonly such properties for an object type.

Finally, work for eliminating the limitation, resulting from the use of case grammar, needs to be provided. As we already pointed out earlier the case grammar may only be used for domains where an action is involved. Therefore, it is important to investigate how the modelling process within domains excluded from this study may be supported.

Acknowledgements

I would like to thank my advisor Docent Paul Johannesson for his valuable comments on earlier drafts of this paper.

References

[1] M. Boman, J.A.Bubenko jr, P. Johannesson and B. Wangler, *Conceptual Modelling*, Prentice Hall Series in Computer Science, 1997.

[2] B. Bruce, “Case Systems for Natural Language”, *Artificial Intelligence*, vol. 6, pp. 327-360, 1975.

[3] P.P. Chen, “The Entity-Relationship Model – Toward a Unified View of Data”, *ACM Transactions on Database Systems*, vol. 1, no. 1, pp. 9-36, 1976.

[4] R.M. Colomb, “Completeness and Quality of an Ontology for an Information System”, in N. Guarino (ed.) *Formal Ontology in Information Systems*, Frontiers in Artificial Intelligence and Applications, IOS Press, pp. 207-217, 1998.

[5] P. Coad, D. North, M. Mayfield, *Object Models: Strategies, Patterns, and Applications*, Prentice Hall, 1995.

[6] CYC® Ontology Guide: Table of Contents, url: http://www.cyc.com/cyc-2-1/toc.html, June 10 1999.

[7] C.H. Fillmore, The case for case, in Bach and Harms, eds. *Universals in Linguistic Theory*, Holt, Rinehart and Winston, New York, 1968.

[8] M. Fowler, Analysis Patterns: Reusable Object Models, Addison-Wesley, 1997.

[9] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[10] N. Guarino, “Formal Ontology and Information Systems”, in N. Guarino (ed.) *Formal Ontology in Information Systems*, Frontiers in Artificial Intelligence and Applications, IOS Press, pp. 3-15, 1998.

[11] T.-D. Han, S. Purao, V.C. Storey, “A Method for Building a Repository of Object-Oriented Design Fragments”, in J. Akoka, M. Bouzeghoub, I. Comyn-Wattiau, E. Métas (eds.) *Conceptual Modeling – ER’99*, Lecture Notes in Computer Science, Springer, pp. 203-217, 1999.

[12] D.C. Hay, Data Model Patterns: Conventions of Thought, Dorset House Publishing, 1996.

[13] P. Johannesson, “Using Conceptual Graph Theory to Support Schema Integration”, in *12th International Conference on Entity-Relationship Approach*, Ed. R. Elmasri, pp. 280 - 289, Dallas, Omnipress, 1993.

[14] P. Johannesson, "Schema Transformations as an Aid in View Integration", in Fifth International Conference on Systems Engineering, Ed. C. Rolland, pp. 144 - 251, Paris, Springer, 1993.

[15] P. Johannesson, *Schema Integration, Schema Translation, and Interoperability in Federated Information Systems*, Dissertation at Department of Computer and Systems Sciences, Stockholm University and Royal Institute of Technology, Sweden, 1993.

[16] N.A. Maiden, *Analogical Specification Reuse During Requirements Engineering*, Dissertation at Department of Business Computing, Scholl of Informatics, City University, London, 1992.

[17] J. Martin and J. Odell, *Object-Oriented Methods: A Foundation*, Prentice Hall, 1994.

[18] B. Moulin and P Creasy, “Extending the Conceptual Graph Approach for Data Conceptual Modelling”, *Data and Knowledge Engineering*, vol. 8, no. 3, pp. 223-248, 1992.

[19] Ontology ENTERPRISE-ONTOLOGY, url: http://www-ksl-

svc.stanford.edu:5915/FRAME-EDITOR/UID-21&sid=ANONYMOUS&user-id=ALIEN, June 10 1999.

[20] J.F. Sowa, *Conceptual Structures: Information Processing in Mind and Machine*, Addison-Wesley, 1984.

[21] TOVE Manual, url: http://www.ie.utoronto.ca/EIL/tove/ontoTOC.html, June 10 1999.

Appendix

Formal Definition of a Conceptual Schema

The formalism presented here is based on those proposed by Johanesson in [14] and [15].

Let P and C be two sets of symbols. A (first order) *language* based on <P,C> written L(P:C) is defined on an alphabet consisting of connectives, quantifiers, punctual symbols, variables, constants C, and predicate symbols P, where each predicate symbol has an arity. A *formula* in a language L is defined as usual. A *term* is a constant or a variable. An *atom* is a formula of the type p(t1,…,tn), where p is a predicate symbol from P and t1, …, tn are terms and we say that p has arity n. A *ground formula* is a formula without variables.

An *integrity constraint* is any closed first order formula. Certain special cases of constraint in conceptual modelling are *typing constraints*, *cardinality constraints* and *generalisation constraints*, which we are defining below.

- A *typing constraint* is either of the form
  - $\forall x \forall y(A(x,y) \rightarrow D(x))$; or of the form
  - $\forall x \forall y(A(x,y) \rightarrow R(y))$:

  where the first one is read "the domain for A is D" abbreviated "domain(A) = D" and the second one "the range for A is R" abbreviated "range(A) = D".
- *Cardinality constraints* concern the cardinality of a conceptual relation. The following four type of cardinality constraints are necessary to define a relationship.
  - The relationship A is single valued iff $\forall x \forall y \forall z(A(x,y) \wedge A(x,z) \rightarrow y=z)$
  - The relationship is A injective iff $\forall x \forall y \forall z(A(y,x) \wedge A(z,x) \rightarrow y=z)$
  - The relationship is A total iff $\forall x(P(x) \rightarrow \exists y A(x,y))$, where P is the domain of A
  - The relationship is A surjective iff $\forall x(P(x) \rightarrow \exists y A(x,y))$, where P is the range of A
- A *generalisation constraint* is a formula of the form $\forall x(P(x) \rightarrow Q(x))$ abbreviated "$P \subset Q$"

A *conceptual schema* CS is a pair <L,IC>, where L is a language and IC is a set of integrity constraint as defined above. L is restricted to only contain unary and binary predicate symbols. The unary predicate symbols are called object types and the binary predicate symbols are called relationships. Furthermore, we distinguish a subset LP of the set P of predicate symbols i.e. LP $\subset$ P, and call the elements of LP for lexical predicates symbols. This distinction is necessary in order to keep away data values from object identifiers. We restrict the domain for a relationship to only belong to P-LP. Relationships whose range is from LP are called attributes. In Figure 9 a graphical representation of a schema is shown using a UML similar notation. The graph shows that the schema contains five predicate symbols {`PERSON`, `MAN`, `WOMAN`, `COUNTRY`, `String`} where `String` is a lexical predicate symbol i.e. `String` $\subset$ LP. The schema contains, further, five binary predicate symbols, three of which are relationships, namely {`cousin`, `citizen_in`, `married_to`}, and the other two, {`ss#`, `name`}, are attributes. The figure also shows a number of typing constraints i.e. 'domain(`citizen_in`) = `PERSON'` and 'range(`citizen_in`) = `COUNTRY'`. The generalisation constraints '`MAN` $\subset$ `PERSON'` and '`WOMAN` $\subset$ `PERSON'` are showed by dotted arrows. Furthermore the quadruples next to the relationships

shows the mapping constraints in the order they are presented above i.e ‘single valued, injective, totality, surectivity’. ‘1’ stands for single valued/ingective, ‘m’ otherwise, and ‘t’ stands for total/surjective, ‘p’ otherwise.

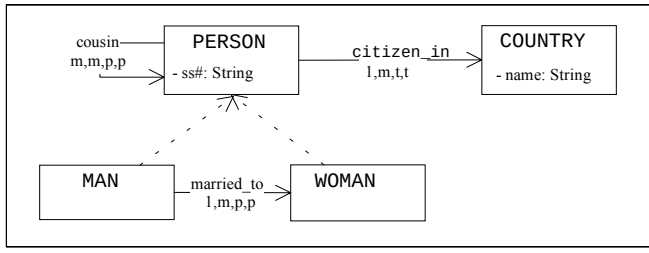


Fig. 9. Example of a conceptual schema

We finally give a definition for an information base. An *information base* for a conceptual schema CS = <L(P,C), IC> is a set IBcs of ground atoms whose predicate symbols belong to P. An example of an information base for the schema in Figure 9 is {man(p1), person(p1), woman(p2), person(p2), ss#(p1,’Peter’), ss#(p2,’Mary’), married_to(p1,p2)}

Formal Definition for a Domain Dependent Modelling Wizard Tool for the Booking Domain

In this part of the appendix the questions identified in Section 4 are reformulated in order to be suitable for a user dialog implemented in a modelling wizard tool. Each question is followed by a formal definition for the logic of the tool i.e. the schema it will build and suggest to the user.

1. Does a booking consist
- **a) one object, or**
- **b) may it consist of several objects?**

a) CS:= {BOOKING}

b) CS:= {BOOKING,BOOKING_LINE,consists_of,
domain(consists_of)=BOOKING,
range(consists_of)=BOOKING_LINE}

2. Does a booking concern a (number of)
- **a) concrete object(s), or**
- **b) does it rather specify the character of the object(s)**

a) CS:= CS ∪ {OBJECT,specifies,range(specifies)=OBJECT}
∪ {domain(specifies)=BOOKING_LINE|BOOKING_LINE exists}
∪ {domain(specifies)=BOOKING|BOOKING_LINE does not exist}

b) CS:= CS ∪ {OBJECT_CATEGORY,specifies,
range(specifies)=OBJECT_CATEGORY}
∪ {domain(specifies)=BOOKING_LINE|BOOKING_LINE exists}
∪ {domain(specifies)=BOOKING|BOOKING_LINE does not exist}

3. Do all the bookings

a) have the same character,

or may they be divided into several categories? If yes, is the number of categories

b) smaller than, or equal to four (give the number X of categories), or

c) larger than four.

```
a) CS does not change

b) CS:= CS ∪ {OBJECT1,…,OBJECTX,specifies1,…,specifiesX,
               range(specifies1)=OBJECT1,…,
               range(specifiesX)=OBECTX|OBJECT exists}
          ∪ {OBECT_CATEGORY1,…,OBECT_CATEGORYX,
             specifies1,…,specifesX,
             range(specifies1)=OBJECT_CATEGORY1,…,
             range(specifiesX)=OBECT_CATEGORYX|
                                          OBECT_CATEGORY exists}
          - {OBJECT,OBJECT_CATEGORY,specifies,domain(specifies),
                                            range(specifies)}
          ∪ {BOOKING_LN_CATEGORY1,…,BOOKING_LN_CATEGORYX,
             BOOKING_LN_CATEGORY1 isa BOOKING_LINE,…,
             BOOKING_LN_CATEGORYX isa BOOKING_LINE,
             domain(specifies1)=BOOKING_LN_CATEGORY1,…,
             domain(specifiesX)=BOOKING_LN_CATEGORYX|
                                             BOOKING_LINE exists}
          ∪ {BOOKING_CATEGORY1,…,BOOKING_CATEGORYX,
             BOOKING_CATEGORY1 isa BOOKING,…,
             BOOKING_CATEGORYX isa BOOKING,
             domain(specifies1)=BOOKING_CATEGORY1,…,
             domain(specifiesX)=BOOKING_CATEGORYX|
                                       BOOKING_LINE does not exist}

c) CS does not change
```

4. Is it necessary to keep information about the request for a booking?

a) No.

If yes, does the request concern

b) a concrete object(s), or

c) a specification of object(s).

```
a) CS does not change

b) CS:= CS ∪ {BOOKING_ORDER,according_to,
               domain(according_to)=BOOKING,
               range(according_to)=BOOKING_ORDER}
          ∪ {ORDER_LINE,consists_of_ln,according_to_ln,
             domain(consists_of_ln)=BOOKING_ORDER,
             range(constist_of_ln)=ORDER_LINE,
             domain(according_to_ln)=BOOKING_LINE,
             range(according_to_ln)=ORDER_LINE|BOOKING_LINE exists}
          ∪ {ORDER_CATEGORY1,…,ORDER_CATEGORYX,
             ORDER_CATEGORY1 isa BOOKING_ORDER,…,
             ORDER_CATEGORYX isa BOOKING_ORDER|
                       BOOKING_CATEGORY1,…,BOOKING_CATEGORYX exist}
          ∪ {ORDER_LN_CATEGORY1,…,ORDER_LN_CATEGORYX,
             ORDER_LN_CATEGORY1 isa ORDER_LINE,…,
             ORDER_LN_CATEGORYX isa ORDER_LINE|
                 BOOKING_LN_CATEGORY1,…,BOOKING_LN_CATEGORYX exist}
```

```
CS:= CS ∪ {OBJECT1,…,OBJECTX,o_specifies1,…,o_specifiesX,
           range(o_specifies1)=OBJECT1,…,
           range(o_specifiesX)=OBJECTX,
           domain(o_specifies1)=ORDER_CATEGORY1,…,
           domain(o_specifiesX)=ORDER_CATEGORYX|
                    ORDER_CATEGORY1,…,ORDER_CATEGORYX exist}
        ∪ {OBJECT1,…,OBJECTX,o_specifies1,…,o_specifiesX,
           range(o_specifies1)=OBJECT1,…,
           range(o_specifiesX)=OBJECTX,
           domain(o_specifies1)=ORDER_LN_CATEGORY1,
           domain(o_specifiesX)=ORDER_LN_CATEGORYX|
                   ORDER_LN_CATEGORY1,…,ORDER_LN_CATEGORYX exist}
        ∪ {OBJECT,o_specifies, range(o_specifies)=OBJECT,
           domain(o_specifies)=BOOKING_ORDER|ORDER_LINE∨
                  ORDER_CATEGORY1∨…∨ORDER_CATEGORYX do not exist}
        ∪ {OBJECT,o_specifies,range(o_specifies)=OBJECT,
           domain(o_specifies)=ORDER_LN_CATEGORY|
                    ORDER_LINE exist, ORDER_LN_CATEGORY1∨…∨
                           ORDER_LN_CATEGORYX do not exist}

CS:= CS ∪ {belongs_to1,…,belongs_toX,
           domain(belongs_to1)=OBJECT1,…,
           domain(belongs_toX)=OBJECTX,
           range(belongs_to1)=OBJECT_CATEGORY1,…,
           range(belongs_toX)=OBJECT_CATEGORYX|
                    OBJECT_CATEGORY1,…,OBJECT_CATEGORYX exist}
        ∪ {belongs_to,domain(belongs_to)=OBJECT,
           range(belongs_to)=OBJECT_CATEGORY|
                                    OBJECT_CATEGORY exists}
```

c) like b) and exchanging OBJECT and OBJECT_CATEGORY

5. Does a motivation need to be given for

- **a booking?** (Relevant only if the information about a booking order is not necessary, since the booking order is concerned as a motivation itself)
  - **a) yes**
  - **b) no**
- **a booking order?** (Relevant when it is necessary to keep information about the booking orders)
  - **c) yes**
  - **d) no**

a) CS:= CS ∪ {MOTIVATION,motivated_by,domain(motivated_by)=BOOKING,
range(motivated_by)=MOTIVATION}

b) CS does not change

c) CS:= CS ∪ {MOTIVATION,motivated_by,range(motivated_by)=MOTIVATION,
domain(motivated_by)=BOOKING_ORDER}

d) CS does not change

6. May a booking be done on behalf of someone else? (relevant only when a booking requirement is not necessary to keep information about)

a) no

If yes, is it important to keep information about the party who made the booking?

b) no

c) yes

May a booking request be done on behalf of someone else? (Relevant only when it is necessary to keep information about the booking requirement)

d) no

If yes, is it important to keep information about the party who placed the request?

e) no

f) yes

```
a) CS:= CS ∪ {PARTY,booking_for,domain(booking_for)=BOOKING,
               range(booking_for)=PARTY}

b) like a)

c) CS:= CS ∪ {PARTY,booking_for,booked_by,
              domain(booking_for)=BOOKING,range(booking_for)=PARTY,
              domain(booked_by)=BOOKING,range(booked_by)=PARTY}

d) CS:= CS ∪ {PARTY,booking_for,domain(booking_for)=BOOKING_ORDER,
              range(booking_for)=PARTY}

e) like d)

f) CS:= CS ∪ {PARTY,booking_for,ordered_by,
              domain(booking_for)=BOOKING_ORDER,
              range(booking_for)=PARTY,
              domain(ordered_by)=BOOKING_ORDER,
              range(ordered_by)=PARTY}
```