On the use of Object-Role Modelling to Model Active Domains

P. (Patrick) van Bommel, S.J.B.A. (Stijn) Hoppenbrouwers, H.A. (Erik) Proper and Th.P. (Theo) van der Weide

Institute for Computing and Information Sciences,
Radboud University Nijmegen,
Toernooiveld 1, 6525 ED Nijmegen, The Netherlands, EU
{P.vanBommel, S.Hoppenbrouwers, E.Proper, Th.P.vanderWeide}@cs.ru.nl

Abstract. Conceptual modelling methods such as Object-Role Modelling (ORM) have traditionally been developed with the aim of providing conceptual models of database structures. More recently, however, such modelling languages have shown their use for modelling (the ontology) of domains in general. In these latter cases, the modelling effort results in a (formally based) conceptual reasoning systems using a *domain calculus* on top of a *domain grammar*.
As the title suggests, this paper is primarily concerned with the application of ORM 'rigour' to the modelling of active domains. In doing so, we will position the *logbook paradigm* as a history-oriented extension of the traditional natural language approach of ORM, and define an accompanying domain calculus (the *Object-Role Calculus*) which is suitable to deal with active domains. Finally, we will show how specific views (with dedicated notations), which zoom in on different aspects (such as flow of activities and actor involvement) of active domains, can easily be derived.

1 Introduction

Conceptual modelling methods such as ER [4], NIAM [16], OOSA [5] and ORM [9] have traditionally been developed with the aim of providing conceptual models of database structures. More recently, however, such modelling methods have shown their use for modelling (the ontology) of domains in general [19, 20]. In the latter case, their use leads to models capturing the concepts of a domain in general, as well as an associated language to express rules (such as business rules) governing the behaviour of the domain.

The above mentioned modelling methods typically take a natural language based perspective on the domain to be modelled. In this perspective, the resulting models are regarded as a *domain grammar* describing the allowed communication about a domain; the *universe of discourse*. This way of thinking dates back to the ISO report on *Concepts and Terminology for the Conceptual Schema and the Information Base* [13], and is at the base of modelling methods such as ER, NIAM, OOSA and ORM. A key advantage of such methods is that having a domain grammer at ones disposal, enables validation of the model by domain experts, since the model can be validated in terms of statements that are close to the language used by these experts.

A domain grammar can be extended to also cover rules (constraints) governing the behaviour of the domain. When combined with a reasoning mechanism, this rule language becomes a *domain calculus*. In the case of ORM, such a domain calculus has been presented in the form of Lisa-D [10], a formalisation of RIDL [15]. In [18, 2] a more practical (from an implementation point of view) version called ConQuer has been introduced. What each of these languages have in common is that they exploit [11] the naturalness of the domain grammer in the construction of rules. As a result, the formulation of rules, as well as chains of reasoning expressed in these rules, closely resembles natural language. Being able to do so, *again*, enables validation of the models produced.

In the use of domain modelling methods, we observe three important trends which fuel our research activities. Firstly, more and more organisations strive for more mature levels of system development [17]. One of the steps towards maturity involves better *defining* development processes in order to make them more *repeatable*. This also applies to modelling processes. Organisations strive to make modelling processes more explicitly *defined* with the aim of achieving more *repeatable* results. Modelling methods such as ORM NIAM and OOSA not only feature a *way of modelling*, but also have a well-defined and explicit *way of working* based on natural language analysis. The *way of working* of a method is concerned with processes, guidelines, heuristics, etc, which are to be used in the creation of models, as opposed to its *way of modelling* which refers to the syntax and semantics of the language in which the models are to be expressed. Having a well-defined and explicit way of working aids towards a defined and more repeatable modelling process.

The second trend fuelling our research, is the use of *controlled languages* as the basis for unambiguous communication [6, 22]. The essential idea of a controlled language is to define a subset of natural language which is rich enough for a specific purpose, but still restrictive enough so as to avoid unambiguities. We claim that a *domain grammer* and associated *domain calculus* provide a good starting point in defining controlled languages for domains. To some extend, a *domain calculus* already provides a (highly) controlled language. Such languages can also be used to represent domain specific reasoning steps, providing an additional form of domain knowledge. In [12] an initial study into the use of a domain calculus for such purposes has been reported.

The third trend we observe is the growing need for integrated models underlying a plethora of viewpoints, fuelled by the demands of MDA [7] and enterprise architecture [14]. UML as well as approaches for enterprise architecting [14] feature a wide variety of diagramming techniques (viewpoints). A domain model can provide a common underpinning of this variety of viewpoints, offering a unified domain ontology. A first elaboration of this role of domain models has been presented in [20]. However, more work needs to be done to make ORM suitable to deal with the modelling of active domains. This is the focus of this paper.

When applying ORM for the purpose of modelling active domains, we are primarily interested in re-using its rigorous way of working in the creation of models. This does require both active aspects (activities, tasks, processes, etc) as well as static aspects (results, documents, actors, tangiable objects, etc) to be expressed as *objects* playing *roles* in the domain.

The remainder of this paper is structured allong the overall way of working we suggest when modelling an active domain:

1. (Section 2) using the *logbook paradigm* the activities taking place in an active domain can be reported in terms of (elementary) facts, which can consequently be used (in principle using ORM's standard approach) to derive a *domain grammar*,
2. (Section 3) any constraints, temporal dependencies, etc, governing the flow of activities in a domain can then be formulated using a *domain calculus* referred to as the *Object-Role Calculus*,
3. (Section 4) finally, special graphical conventions are introduced to provide more compact representations of specific aspects of the active domain, such as the *flow* of activities, or the *involvement* of actors.

2 The logbook paradigm

When focussing on active domains ORM needs to be refined in order to better cater for the active aspects of such domains. The underlying challenge is to extend ORM to be able to cater for such domains, while at the same time maintaining ORM's natural-language based modelling rigour. In doing so, we base ourselves on earlier (partial) results [21, 8].

Modelling an active domain does require a modelling language to deal with the notion of time. In the past, ORM has indeed been extended with the concept of time and evolution [21]. In this paper we propose to formalize this in terms of a *logbook* [8], which is intended to trace/mirror the activities taking place in the domain. Such a logbook will consist of a series of events reporting on the lifecycle of facts in the domain. For example:

Trafic light 20 is green ceased being true at 11:03:20 on 22-05-2006
Employee John works on the completion of order 50 started being true at 09:30 on 19-05-2006

In our view, a logbook approach is a natural extension of the earlier discussed natural language based perspective on modelling. To be more precise, we regard a *history* as an overview of the events that have taken place in the domain, while a *logbook* is a description of such a history using some controlled language.

The facts contained in the descriptions of the events are asumed to be expressed in terms of semi-natural language (controlled language) sentences as is normally the case in ORM's way of working. Using a traditional ORM approach, the set of facts used/allowed in a logbook can be generalised to a set of fact types, which together comprise the ORM model underlying the domain. As such, this ORM model then defines the domain grammar of the controlled language in which the facts are to be formulated.

Traditionally, ORM focuses on the modelling of facts in general. In the context of an active domain, these facts correspond to statements about what is the case and/or has happened in the domain at specific points in time. In ORM, the actual modelling process starts out from the verbalisation of such facts. These verbalisations are the starting point for the creation of the domain grammar. When considering an active domain, the set of facts that can be reported about this domain fall into two categories: (1) *acts* reporting on the performance of *actions* and (2) *effects* reporting the *results* of actions. This dichotomy applies at the instances level (the facts) as well as the type level, leading to *act types* and *effect types* respectively as sub-classes of *fact types*. In the case of *acts*, the objects involved (i.e. playing a role in the act) can be classified further into *actors* (objects responsible for performing the *act*) and *actands* (objects which are the effect of the *act*).

We assume that each event described in the logbook and the objects participating in the event, can be uniquely identified in that logbook. We will call this the *Event Identification Principle*. This identification principle is used as the base for all other identification mechanisms. This principle does not inhibit different events to occur on the same moment. In order to distinguish between accidently coincidence and necessarily coupled events, we assume that events may also have a compound nature, in such a way that: (1) different events in a logbook are independent of each other, (2) events are not splittable into multiple independent events.

We take the perspective that the state of an active domain is the result of the sequence of actions leading up to that state. These actions may either take place in the domain, or outside the domain (such as the very creation of the domain). As a result, we take the position that the *effects* are actually derivable from the set of reported *acts*. This is what we call the *Action Dominance Principle*. This principle does lead to the question on how persistent properties, such as the speed of light, are to be treated in our logbook approach. This is covered by the *Property Origination Principle*, which states that each domain property pertains to: (1) either some act taken place in the domain, (2) or some effect of some act in the domain, (3) or some effect of the domain's creation (i.e. the result of a 'big bang' act). As a consequence, at each moment the state of the system is the result of all the effects of the domain's creation and the acts that were reported since then.

An important consequence of the *Property Origination Principle* is that (for most objects in the domain), the property of *being alive* should be the result of some act. Therefore, objects that are not present in the initial state require an explicit birth event. This is called the *Birth Principle*. Obviously, an object can not be responsible for its own birth, as it can not be active before coming into existence. The consequence is that some other object has to be responsible for causing this event, thus playing a dominant role in that event. If the existence of an object may terminate, then there should be an explicit death action that enforces an object to have the property of being death.

An immediate consequence of the Birth Principle and the Event Identification Principle is that objects may be identified by their birth event. If an event starts life for more objects, then we require that the individual objects in this case may be identified by this event and their role in this event.

3 Object-Role Calculus

This section is concerned with a conceptual language in which rules can be expressed describing the behaviour that may be observed in a logbook compatible with the domain being modelled. The language presented, referred to as *Object-Role Calculus* (ORC) is a variant of Lisa-D [10], a formalisation of RIDL [15]. Lisa-D has originally been designed to describe all computable sets of facts that can be derived from the elementary facts defined in the underlying conceptual schema. The conceptual schema specifies all elementary sentences applicable for that domain. The semantics of Lisa-D have been described in terms of multi-sets. In this paper we will provide a light-weight definition of the ORC variant of Lisa-D, which is intended to describe temporal and statical aspects of the underlying domain.

3.1 Grounding in temporal logic

The semantics of ORC are grounded on Kripke structures [3]. In terms of Kripke structures, an application domain is seen as a Kripke structure $\langle S.R.s_0, \Pi, L\rangle$, where:

1. S is a non-empty set of states,
2. $R \subseteq S \times S$ is a total transition function, i.e. $\forall_s \exists_T \left[(s,t) \in R\right]$,
3. s_0 is the initial state,
4. Π is a non-empty set of atomic propositions, and
5. L is a labelling function that maps each state on a subset of Π.

Our main assumption is that the state of an application domain is described by its history so far. As a consequence, a state corresponds uniquely to a logbook. Consequently, the transition function extends a logbook with a new event description, and the initial state is obtained as the empty logbook.

From the structure of the events in the logbook, the elementary object types. Their possible instantiations form the set Π of atomic propositions. The labeling function L then assigns the population of object types that is constructed by a logbook.

A linear-time temporal logic is syntactically described by the following BNF grammar:

$$\phi \rightarrow \mathit{true} \mid \mathit{false} \mid \Pi \mid \neg\phi \mid q \wedge \phi \mid \phi \vee \phi \mid \phi \Rightarrow \phi \mid \mathsf{X}\,\phi \mid \mathsf{F}\,\phi \mid \mathsf{G}\,\phi \mid \phi\,\mathsf{U}\,\phi$$

The expression $\mathsf{X}\,\phi$ states that ϕ will hold in the next state, $\mathsf{F}\,\phi$ that ϕ will eventually hold, $\mathsf{G}\,\phi$ that ϕ will globally hold and $\phi\,\mathsf{U}\,\psi$ states that at some point

ψ will hold, while in all states before ϕ is valid. Let M be a Kripke structure over logbook LB, and let σ be a history. We further will assume an environment E for evaluation, consisting of a partial assignment of values to a set V of variables. The standard semantic interpretation of the temporal operators is:

$$\begin{aligned} M, E, \sigma \models \mathsf{X}\,\phi &\triangleq M, E, \sigma^1 \models \phi \\ M, E, \sigma \models \phi\,\mathsf{U}\,\psi &\triangleq \exists_n \left[\forall_{0 \le i < n} \left[M, E, \sigma^i \models \phi\right] \wedge M, E, \sigma^n \models \psi\right] \end{aligned}$$

where $\sigma(i)$ denotes the i-th element of sequence σ and σ^i the subsequence of σ starting at position i. The other temporal operators are defined in terms of these base operators: $\mathsf{F}\,\phi$ is equivalent with $\mathit{true}\,\mathsf{U}\,\phi$, and $\mathsf{G}\,\phi$ is defined as $\neg\,\mathsf{F}\,\neg\phi$. The propositional operators are also interpreted in the standard way:

$$\begin{aligned} M, E, \sigma \models \neg\phi &\triangleq \text{not } M, E, \sigma \models \phi \\ M, E, \sigma \models q \wedge \psi &\triangleq M, E, \sigma \models \phi \text{ and } M, E, \sigma \models \psi \end{aligned}$$

The constant *false* is introduced as $p \wedge \neg p$ where p is any proposition from Π, and *true* is derived by $\neg$*false*. The other logical operators ($\vee$ and $\Rightarrow$) are defined in the usual way. The conversion from a temporal proposition to a static expression requires the evaluation of the static expression for the population $L(\sigma(0))$ at the required point of time. This will be further elaborated in section 3.4.

3.2 Historical information descriptors

History descriptors in ORC are meant to provide a language construct for reasoning in an historical setting about the application domain. For the purpose of this paper, it will be sufficient to make more or less direct transcriptions of the basic temporal operators. For this the syntactical construct *history descriptor* is introduced. Let H be a history descriptor, then the semantics of H are denoted as $[\![H]\!]$:

$$[\![\mathsf{always}\, H]\!] \triangleq \mathsf{G}\,[\![H]\!] \qquad [\![\mathsf{X}\, H]\!] \triangleq \mathsf{X}\,[\![H]\!]$$

In addition we introduce the following abbreviations:

$$\begin{aligned} \mathsf{sometime}\, H &\triangleq \neg\,\mathsf{always}\,\neg H \\ H_1\,\mathsf{precedes}\, H_2 &\triangleq \mathsf{always}((\mathsf{F}\,H_1)\,\mathsf{U}\,H_2) \\ H_1\,\mathsf{during}\, H_2 &\triangleq \mathsf{always}(H_1 \Rightarrow H_2) \\ H_1\,\mathsf{triggers}\, H_2 &\triangleq \mathsf{always}(H_1 \wedge \neg H_2 \Rightarrow \mathsf{X}(\neg H_1 \wedge H_2)) \end{aligned}$$

The first rule will be a target for the educational organisation. The later rule states describes a trigger that, whenever the condition $H_1 \wedge \neg H_2$ is met, will respond by setting the condition $\neg H_1 \wedge H_2$ at the next moment. Some example expression would be:

sometime Lecturer lectures Course

Lecturer sets up Course precedes Lecturer lectures Course

This latter expression, however, is misleading as it does not bring about a connection between lecturer nor course being set up and being lectured. In natural language, indicatives are used in most cases to make such references. We furthermore introduce:

$$\begin{aligned} x \llbracket D_1 \,\textsf{PRECEDES}\, D_2 \rrbracket\, y &\triangleq (x \llbracket D_1 \rrbracket\, y)\,\textsf{precedes}\,\exists_z\,[z \llbracket D_2 \rrbracket\, y] \\ x \llbracket D_1 \,\textsf{DURING}\, D_2 \rrbracket\, y &\triangleq (x \llbracket D_1 \rrbracket\, y)\,\textsf{during}\,\exists_z\,[z \llbracket D_2 \rrbracket\, y] \end{aligned}$$

3.3 Indicative descriptors

The main idea behind ORC, as present in its early ancestor RIDL [15] is a functional, variable-less description of domain-specific properties (and queries). RIDL did contain a linguistic reference mechanism (the indicative THAT). In ORC variables have been introduced to handle more subtle referential relations that can not be handled by indicatives. Variables are special names that are instantiated once they are evaluated in a context that generates values for this variable. The environment is used to administrate the value of variables, in environment E, the variable v will evaluate to $E(v)$. Some examples of the use of variables:

Lecturer:x being hired precedes x sets up Course

Lecturer:x sets up c precedes x lectures Course:c

In this example, the expression Lecturer:x is a defining occurrence of variable x in which Lecturer has the role of value generator. The environment is used to administrate the variable-value assignment (see [10] for more details).

3.4 Information descriptors

The syntactic category to retrieve a collection of facts is called *information descriptor*. We will discuss the semantics of elementary information descriptors, and briefly summarise the construction of information descriptor (for more details, see [10]). Information descriptors are constructed from the names of object types and role type. The base construction for sentences is juxtaposition. By simply concatenating information descriptors, new information descriptors are constructed.

Information descriptors are interpreted as binary relationships, they provide a binary relation between instances of the population induced from the history. The semantics of information descriptor D are denoted as $\llbracket D \rrbracket$, we will write $x \llbracket D \rrbracket\, y$ to denote the relationship between x and y. The statement $M, E, \sigma \models x \llbracket D \rrbracket\, y$ asserts that for Kripke structure M in environment E from history σ the relationship $x \llbracket D \rrbracket\, y$ can be derived.

A population assigns to each object type its set of instances. Let n be the name of object type N and r the name of a role type R, then n and r are information descriptors with semantics:

$$\begin{aligned} M, E, \sigma \models x \,[\![n]\!]\, y &\triangleq x \in L(\sigma(N)) \wedge x = y \\ M, E, \sigma \models x \,[\![r]\!]\, y &\triangleq (x, y) \in L(\sigma(R)) \end{aligned}$$

A single role may, in addition to its 'normal' name, also receive a *reverse role name*. Let v be the reverse role name of role R, then we have:

$$M, E, \sigma \models x \,[\![v]\!]\, y \triangleq (y, x) \in L(\sigma(R))$$

A combination of roles involved from a fact type may receive a *connector name*. The connector name allows us to 'traverse' a fact type from one of the participating object types to another one. If c is the connector name for a role pair $\langle R, S\rangle$, then the semantics of the information descriptor c are defined as:

$$M, E, \sigma \models x \,[\![c]\!]\, z \triangleq \exists_y \left[M, E, \sigma \models x \,[\![R]\!]\, y \wedge M, E, \sigma \models z \,[\![S]\!]\, y\right]$$

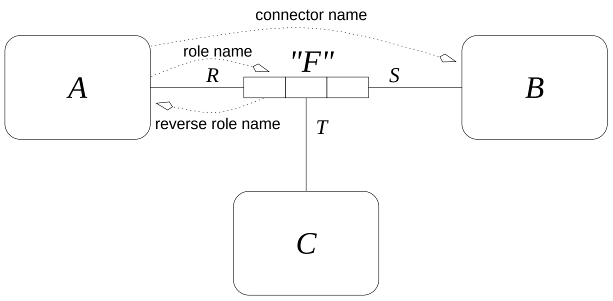

Fig. 1. Role names

Elementary information descriptors can be composed into complex information descriptors using constructions such as *concatenation*, *conjunction*, *implication*, *disjunction* and *complement*. These may refer to the fronts alone or both fronts and tails of descriptors. For more details, see [10]. In this paper we will use:

$$\begin{aligned} x \,[\![D_1\ D_2]\!]\, y &\triangleq \exists_z \left[x \,[\![D_1]\!]\, z \wedge z \,[\![D_2]\!]\, y\right] \\ x \,[\![D_1 \ \textsf{AND ALSO}\ D_2]\!]\, y &\triangleq \exists_z \left[x \,[\![D_1]\!]\, z\right] \wedge \exists_z \left[x \,[\![D_2]\!]\, z\right] \wedge x = y \end{aligned}$$

where D_1 and D_2 are information descriptors and x, y and z are variables. Some example expression would be:

Person working for Department 'I&KS'
People working for department 'I&KS'

Person (working for Department 'I&KS' AND ALSO owning Car of Brand 'Seat')
People working for department 'I&KS' who also own a car of brand Seat

Note that the natural language likeness of the ORC expressions used in this paper can be improved considerably.

3.5 Rules

ORC has a special way of using information descriptors to describe rules that should apply in a domain. These rules can be used to express constraints and/or business rules. We will use the more general term *rule* for such expressions. These rules consist of information descriptors that are interpreted in a boolean way; i.e. if no tuple satisfies the relationship, the result is false, otherwise it is true. Some examples of such constructions are:

$$\begin{aligned} [\![\mathsf{SOME}\, D]\!] &\triangleq \exists_{x,y}\, [x\, [\![D]\!]\, y] \\ [\![\mathsf{NOT}\, R_1]\!] &\triangleq \neg\, [\![R_1]\!] \\ [\![\mathsf{NO}\, D]\!] &\triangleq [\![\mathsf{NOT}\, \mathsf{SOME}\, D]\!] \end{aligned}$$

where D is an information descriptor and R_1 a rule.

4 Graphical representation

Using the ORC *temporal dependencies* can be formulated governing the behaviour of a domain. Currently, we are experimenting with effective graphical representation of some key classes of temporal dependencies. In [20] we have provided some examples using notations inspired by the field of workflow modelling [1].

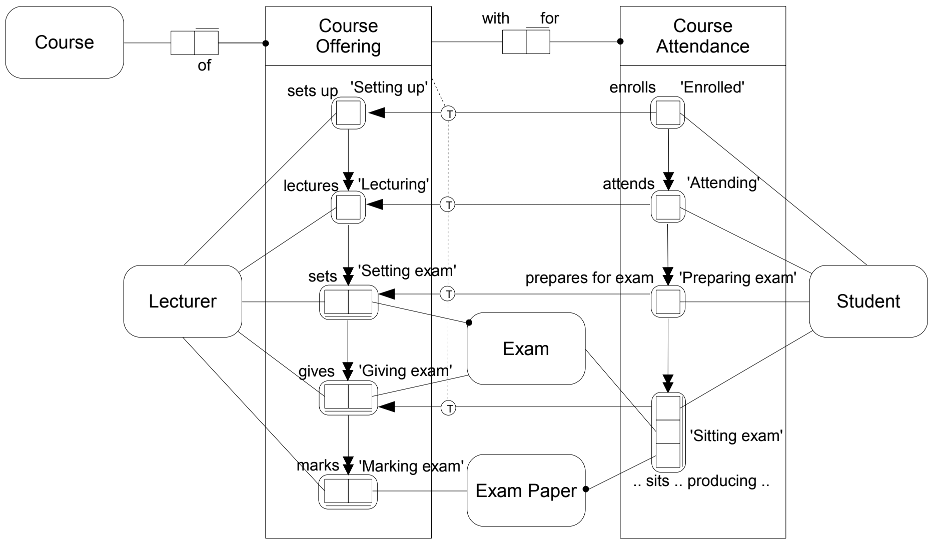

Fig. 2. Lecturing example

An important modelling construct is the notion of a *life-cycle type*. An example of its use is provided in Figure 2, which contains two inter-linked life-cycle types: Course Offering and Course Attendence. Each of these life-cycle types comprise multiple *action types*.

In the example domain, courses are offered to students. In offering a course, a lecturer starts by setting up the course offering. This is followed by the actual lecturing. After lecturing the course, the lecturer sets an exam. This exam is given to the students attending the course, after which the lecturer marks the exam papers produced by the students. Students attend the course by enrolling. After their enrollment they attend the course. Once the course is finished, they prepare themselves for the exam, which is following by the actual exam, leading to an exam paper.

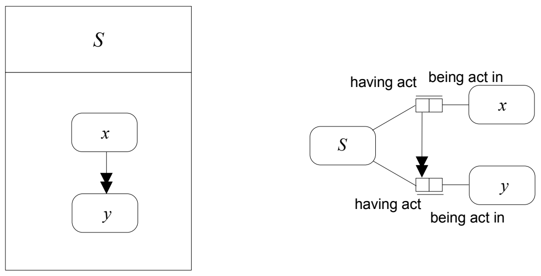

Fig. 3. Life cycle types

In general, the *life-cycle type* typically involves multiple *action types*, and can best be regarded as an abbreviation as illustrated in Figure 3. The temporal dependency between x and y is defined as follows:

$$x \twoheadrightarrow_S y \triangleq x \text{ being act of } S \text{ PRECEDES } y \text{ being act } S$$

The enrollment by students in a course should take place *during* the setup phase of a course. This is enforced by means of the temporal subset constraint from the Enrolling action type to the Setting up action type. The connection between the temporal subset constraint and the Course Offering life-cycle type type signifies that the temporal subset constraint should be evaluated via this object type. In general, the semantics are expressed as: $x \subseteq_\mathcal{T} y \triangleq x$ DURING y. In the case of Figure 2, we have specified a join path, leading for example to:

Enrolling being act of Course attendence for Course offering
DURING
Setting up being act of Course offering

Finally, a model as presented in Figure 2 can be used as a base to derive specialised views such as depicted in Figure 4 focussing on the flow of activities performed by a lecturer.

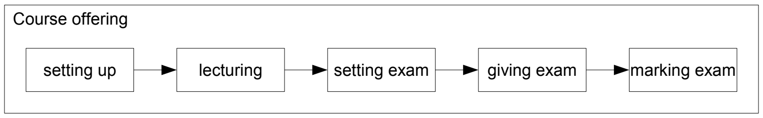

Fig. 4. Lecture activities

5 Conclusions

The research reported in this paper is part of our effort to find a suitable generalised domain modelling method to model active domains. In this paper we have focussed on a strategy to apply ORM rigour in modelling active domains. In doing so, we have introduced the *logbook paradigm* as a history-oriented extension of the traditional natural language approach of ORM. To be able to define rules governing the behaviour of active domains, we have introduced the Object-Role Calculus (ORC). The semantics of this rule language has been defined in terms of Kripke structures. Finally, we have shown how ORM can be extended with graphical constructs, in particular life-cycle types, focussing on temporal dependencies in a domain. This notation allows us to also derive specific views on a domain focussing solely on temporal behaviour.

References

1. W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: yet another workflow language. *Information Systems*, 30(4):245–275, 2005.
2. A.C. Bloesch and T.A. Halpin. ConQuer: A Conceptual Query Language. In B. Thalheim, editor, *Proceedings of the 15th International Conference on Conceptual Modeling (ER‘96), Cottbus, Germany, EU*, volume 1157 of *Lecture Notes in Computer Science*, pages 121–133, Berlin, Germany, EU, October 1996. Springer.
3. B.F. Chellas. *Modal logic: an introduction*. Cambridge University Press, Cambridge, United Kingdom, EU, 1980.
4. P.P. Chen. The Entity–Relationship Model: Towards a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9–36, March 1976.
5. D.W. Embley, B.D. Kurtz, and S.N. Woodfield. *Object–Oriented Systems Analysis – A model–driven approach*. Yourdon Press, New York, New York, USA, 1992.
6. G. Farrington. *An Overview of the International Aerospace Language*. 1996.
7. D.S. Frankel. *Model Driven Architecture: Applying MDA to Enterprise Computing*. Wiley, New York, New York, USA, 2003.
8. P.J.M. Frederiks and Th.P. van der Weide. Deriving and paraphrasing information grammars using object–oriented analysis models. *Acta Informatica*, 38(7):437–88, June 2002.
9. T.A. Halpin. *Information Modeling and Relational Databases, From Conceptual Analysis to Logical Design*. Morgan Kaufmann, San Mateo, California, USA, 2001.
10. A.H.M. ter Hofstede, H.A. (Erik) Proper, and Th.P. van der Weide. Formal definition of a conceptual language for the description and manipulation of information models. *Information Systems*, 18(7):489–523, October 1993.

11. A.H.M. ter Hofstede, H.A. (Erik) Proper, and Th.P. van der Weide. Exploiting Fact Verbalisation in Conceptual Information Modelling. *Information Systems*, 22(6/7):349–385, September 1997.
12. S.J.B.A. Hoppenbrouwers, H.A. (Erik) Proper, and Th.P. van der Weide. Fact Calculus: Using ORM and Lisa–D to Reason About Domains. In R. Meersman, Z. Tari, and P. Herrero, editors, *On the Move to Meaningful Internet Systems 2005: OTM Workshops – OTM Confederated International Workshops and Posters, AWeSOMe, CAMS, GADA, MIOS+INTEROP, ORM, PhDS, SeBGIS, SWWS, and WOSE 2005, Agia Napa, Cyprus, EU*, volume 3762 of *Lecture Notes in Computer Science*, pages 720–729, Berlin, Germany, October/November 2005. Springer–Verlag.
13. *Information processing systems – Concepts and Terminology for the Conceptual Schema and the Information Base*, 1987. ISO/TR 9007:1987.
14. M.M. Lankhorst and others. *Enterprise Architecture at Work: Modelling, Communication and Analysis.* Springer, Berlin, Germany, EU, 2005.
15. R. Meersman. The RIDL Conceptual Language. Technical report, International Centre for Information Analysis Services, Control Data Belgium, Inc., Brussels, Belgium, EU, 1982.
16. G.M. Nijssen and T.A. Halpin. *Conceptual Schema and Relational Database Design: a fact oriented approach.* Prentice–Hall, Englewood Cliffs, New Jersey, USA, 1989.
17. M.C. Paulk, B. Curtis, M.B. Chrissis, and C.V. Weber. Capability Maturity Model for Software, Version 1.1. Technical Report SEI–93–TR–024, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, February 1993.
18. H.A. (Erik) Proper. ConQuer–92 – The revised report on the conceptual query language LISA–D. Technical report, Asymetrix Research Laboratory, University of Queensland, Brisbane, Queensland, Australia, 1994.
19. H.A. (Erik) Proper, A.I. Bleeker, and S.J.B.A. Hoppenbrouwers. Object–Role Modelling as a Domain Modelling Approach. In J. Grundspenkis and M. Kirikova, editors, *Proceedings of the Workshop on Evaluating Modeling Methods for Systems Analysis and Design (EMMSAD'04), held in conjunctiun with the 16th Conference on Advanced Information Systems 2004 (CAiSE 2004),*, volume 3, pages 317–328, Riga, Latvia, EU, June 2004. Faculty of Computer Science and Information Technology.
20. H.A. (Erik) Proper, S.J.B.A. Hoppenbrouwers, and Th.P. van der Weide. A Fact–Oriented Approach to Activity Modeling. In R. Meersman, Z. Tari, and P. Herrero, editors, *On the Move to Meaningful Internet Systems 2005: OTM Workshops – OTM Confederated International Workshops and Posters, AWeSOMe, CAMS, GADA, MIOS+INTEROP, ORM, PhDS, SeBGIS, SWWS, and WOSE 2005, Agia Napa, Cyprus, EU*, volume 3762 of *Lecture Notes in Computer Science*, pages 666–675, Berlin, Germany, October/November 2005. Springer–Verlag.
21. H.A. (Erik) Proper and Th.P. van der Weide. EVORM – A Conceptual Modelling Technique for Evolving Application Domains. *Data & Knowledge Engineering*, 12:313–359, 1994.
22. R. Schwitter. *Controlled Natural Languages.* Centre for Language Technology, Macquary University, Sydney, New South Wales, Australia, 2004.