ORM-Based Semantics of B2B Transactions

H. Balsters and F. van Blommestein

University of Groningen
Faculty of Economics and Business
P.O. Box 800
9700 AV Groningen, The Netherlands
h.balsters@rug.nl, fred@flowcanto.com

Abstract. After widespread implementation of Enterprise Resource Planning and Personal Information Management, the next wave in the application of ICT is headed towards business to business (B2B) communication. B2B has a number of specific aspects, one of them being negotiation. This aspect has been largely neglected by present implementations of standard EDI- or XML-messaging and by B2B webservice implementations. In this paper a precise model is given of the negotiation process. The requirements of a potential Buyer and the offer of a potential Seller are matched and, if the negotiation is successful, a contract is concluded. The negotiation process model is represented in ORM, extended with dynamic constraints. Our model may be implemented in the databases of the trading partners and in message- or service definitions.

1 Introduction

Most commercial and public organizations nowadays have automated their administrative processes. The implementation of Enterprise Resource Planning systems and Personal Information Management systems is widespread. This use of Information Technology has resulted in a huge increase in productivity during the last decennia.

Interaction between companies, however, is still hardly supported by application systems, although there are considerable amounts of (repetitive) administrative activities, which are performed manually at both sides of a commercial relation. It therefore seems reasonable to expect that the use of computer systems in inter-organizational processes will also lead to, at least, the same improvements. The computer systems of companies must then be interconnected, and nowadays that is quite possible, with the global and simple means of communication known as the internet.

In some industry sectors, Electronic Data Interchange (EDI) or XML messaging are deployed. EDI and XML messaging work perfectly in environments where the data exchange can be highly standardised and where commercial relationships hardly change or do not change at all. Setting up an EDI connection between two computer applications takes considerable effort; EDI and XML messages are exchanged in the framework of rigid business processes.

R. Meersman, P. Herrero, and T. Dillon (Eds.): OTM 2009 Workshops, LNCS 5872, pp. 671–681, 2009.

Many relationships between organisations are not sufficiently stable to justify an EDI connection. Companies are constantly looking for new markets and more suitable suppliers, and are permanently innovating their logistics and work processes. EDI cannot keep pace with these changes, due to rigidness in the technology used; i.e., dynamics and flexibility are not sufficiently supported.

In this paper, we take negotiation analysis as our main focus; an important dynamic aspect of Business-to-Business (B2B) transactions. In most B2B (EDI or XML) messaging systems negotiation is not supported in an automated sense. It is assumed that commercial conditions have been agreed on beforehand, using, e.g., paper documents or the telephone. While negotiation is routine in most business relationships, it is not supported by automated B2B-transactions.

In this paper it is shown how negotiation, leading to data and service alignment, can be formally modeled. The model can be used to configure B2B systems that support dynamic B2B processes.

2 Related Work

In line with standardization efforts of ISO/IEC JTC1-SC32 [http://jtc1sc32.org/] and UN/CEFACT [www.unece.org/cefact/], Hofreiter e.a. [17] propose a methodology for designing B2B processes. The methodology, UMM, is based on UML. UMM starts with the identification of the business domain, after which user requirements are gathered, the process flow is modeled, and followed by data modeling. Although both data structures and process flows are modeled, they are treated separately. In reality, alignment of B2B services, the result of a negotiation process, is interwoven with the alignment of related data, so in negotiation process models, separate treatment of data- and process flow will not lead to desired results.

Jayaweera [19] has proposed to add intentions or speech acts to B2B messages. The work described in [19] is partly based on FLBC [21], a language representing B2B messages as first -order predicate logic clauses, enriched with speech acts. Jayaweera [19] combines UMM and speech acts with a B2B ontology (REA), and the negotiation workflow pattern as proposed by Winograd et al. He however does not construct a precise data- or object model for B2B negotiations.

Angelov [1] has conducted an extensive survey of research dedicated to electronic contracting. He presents many patterns and languages that are proposed by various researchers. None of them, however, has mapped contracting patterns to a precise data model. The conclusion drawn in [1] is that modeling of the negotiation process is still in the initial stage of development.

3 Use of ORM in B2B Modeling

B2B modeling places high demands on the modeling language used. The often complex negotiation process has to abide to often rigid business rules and there is often a strict order involved in the negotiation process before success can be claimed. Business rules include constraints and derivation rules. *Static rules* apply to each state of the information system that models the business domain, and may be checked by

examining each state individually (e.g. 'In a Transaction participate exactly one Seller and one Buyer'). *Dynamic rules* reference at least two states, which may be either successive (e.g. 'A product may only be delivered after having been ordered') or separated by some period (e.g. invoices ought to be paid within 30 days of being issued).

This entails that a language used to model negotiations in B2B transactions should have high expressiveness in specifying data and constraints, as well as the possibility to model basic process requirements (including dynamic rules). B2B negotiation usually involves business persons that themselves are not versant in technical modeling, but are domain experts in offering the information needed to conclude a successful negotiation. These domain experts are an essential link in the modeling process, by not only offering input information, but also in offering the eventual validation of the negotiation model. In a negotiation, Seller and Buyer parties often first have to create a common ontology before they can actually start the negotiating. This process of creating a common ontology is also called the process of data alignment. Hence, natural language communication plays an important role. Only after establishing common semantics, can Seller and Buyer start to enter a (automated) negotiation process. This is why modeling languages based on fact-orientation, such as ORM [13,14,16], FCO-IM [8], NIAM [15] and OSM [11] are good candidates in this particular modeling environment, because they all provide the linguistic basis providing for close and correct interaction between the non-technical domain expert and the information modeler. Other popular modeling languages (e.g., ER [9] and UML [23,24]) are lesser qualified candidates, due to lack of sufficient expressiveness, but also to the lack of providing a proper linguistic basis.

The ORM language offers, besides excellent tooling facilities, the extra benefit that it can handle dynamic rules, which will prove to be very useful in modeling negotiation in B2B transactions. In the Service Discovery community the process of getting agreement on the end result of negotiation is called service alignment. We have been inspired by the solution in [3,6] and have transformed the UML/OCL-based approach ([28]) used there to an ORM-based solution exploiting dynamic rule facilities. As a result we offer a completely declarative solution for aligning a B2B service (the buying and selling of a product).

ORM, furthermore, is grounded in first-order logic, offering a clear mathematically-based semantics ([12]). The ORM tools ([10,13]) also offer a reverse-engineering facility, which is very useful in re-creating the linguistic basis of legacy databases often encountered in setting up and eventually implementing an automated negotiation environment. In our ORM-based solution a B2B transaction can eventually be implemented as an advanced database transaction.

We want to point out that UML, in particular, has many drawbacks with respect to ORM as a data modeling language and also lacks proper tooling to deal with database design and implemenation. We refer the interested reader to [16] for an in-depth treatment of the relationship between ORM and UML.

One could argue that process-based languages (YAWL [26], BPEL [23]) could also be good candidates to model negotiation processes. These languages, however, lack data-modeling facilities, as well as constraint modeling facilities necessary to capture the static and dynamic business rules (pertaining to data, and not only to the

sequence of process steps) prevailing in B2B-transactions. This makes a purely process-oriented language less suitable for the modeling of B2B-transactions.

4 Modeling B2B Processes

In a B2B transaction, one typically has the situation that a Buyer wishes some product to be delivered by a Seller. In terms of models, this situation translates into two object types (Buyer and Seller) that participate in a Transaction that will be successful under certain explicit conditions. Initially, i.e., when the transaction starts, Buyer and Seller usually are not in a state of immediate success. Typically, Buyer and Seller enter into a negotiation process that eventually might yield a successful transaction resulting in a contract between Buyer and Seller regarding the delivery of some product and the payment for it. Before this negotiation process can start, however, Buyer and Seller have to be able to communicate with each other; i.e., they have to speak the same language to understand each other's concepts and rules. This part of the business process is called *data alignment*. In the case of product delivery, for example, Buyer and Seller negotiate about concepts like "delivery period" and "product price". These concepts may vary in the context of Buyer and Seller in many aspects, e.g., terminology used to indicate these concepts, or units by which these concepts are measured. We can have, e.g., the following informal rule defining a successful transaction: *Seller promises to deliver the product and Buyer promises to pay after delivery of that product, while the Requested Period is within the Offered Period, the Requested Price is higher than the Offered Price, and the Requested Total Quantity is smaller than the Offered Total Quantity; the state of the transaction then changes from Initiated to Negotiation Successful.* Before Buyer and Seller can start working on completing a successful transaction, they first have to agree on the meaning of concepts like *Period*, *Price* and *Quantity*.

This problem of aligning data is well known in the subject area of *data integration*, as in design of data warehouses ([18,20]) and data federations ([2,5]). Data alignment in this case deals with modeling of an object type *Transaction* involving the alignment of data elements from Buyer and Seller. The problem of *semantic heterogeneity* arises when different meanings are assigned to shared data terms. Matters such as naming conflicts (e.g. homonyms and synonyms), conflicts due to different underlying data types of attributes and/or scaling, and missing attributes all deal with differences in structure and semantics of the different local Buyer and Seller data. By employing techniques such as renaming, conversion functions, default values, and addition of suitable extra attributes one can construct a common data model in which these data inconsistencies are resolved. We refer the interested reader to [2,5] for a detailed treatment of these techniques.

Let us now look at an example of a situation dealing with data alignment. We remark that in all subsequent ORM models offered in this paper, we will use the notation as offered in [13].

In the model shown in figure 1, on the Buyer side, there is an object type OffDelivery (of *Offered Delivery*) which gets its data from an existing proprietary local object type called ProductStock, equipped with data elements Quantity, Period, and KiloPrice. On the Buyer side, we have an object type ReqDelivery (of *Requested Delivery*) with data

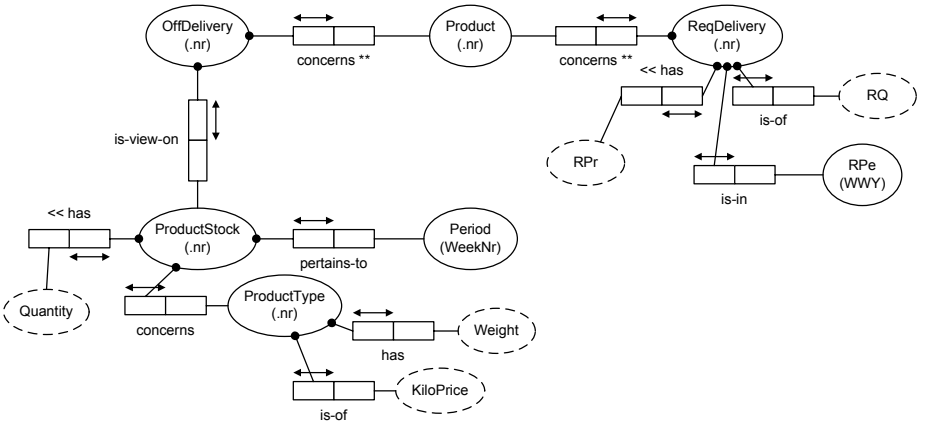

Fig. 1. Data before alignment

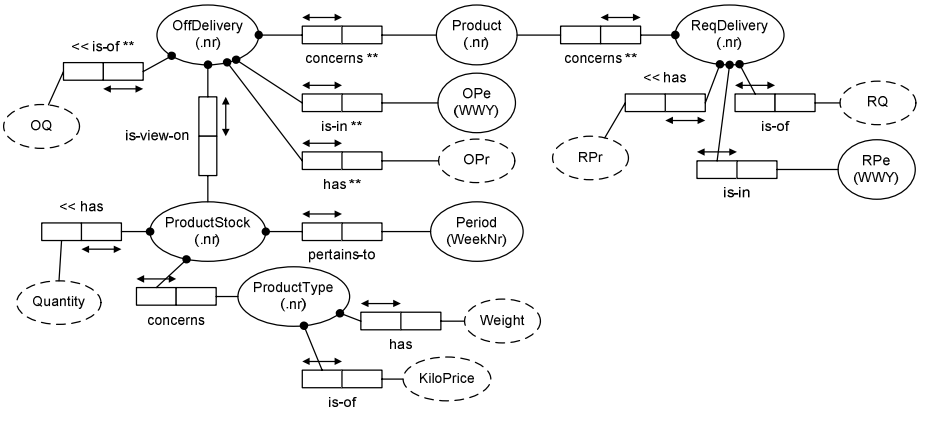

Fig. 2. Data after alignment

elements RQ (*Requested Total Quantity*), RPe (*Requested Period*), and RPr (*Requested Price*). Let us assume that the semantics of RPe is "*the interval of week numbers in which the product is to be delivered*". The reference code WWY indicates that a time interval is denoted as a period beginning and ending with a week number followed by the year in question. OffDelivery must provide a value of some value type, say OPe (of *Offered Period*), denoting "*the interval of week numbers in which the product can be delivered*", so we have to define such a value type as a derived value type from the value type Period (in ProductStock). This would entail aggregating all of the week numbers pertaining to the Producttype of the product in question. Similar remarks can be made about RQ and RPr, involving the introduction of derived value types OQ (of *Offered Total Quantity*) and OPr (of *Offered Price*). This yields the model shown in figure 2.

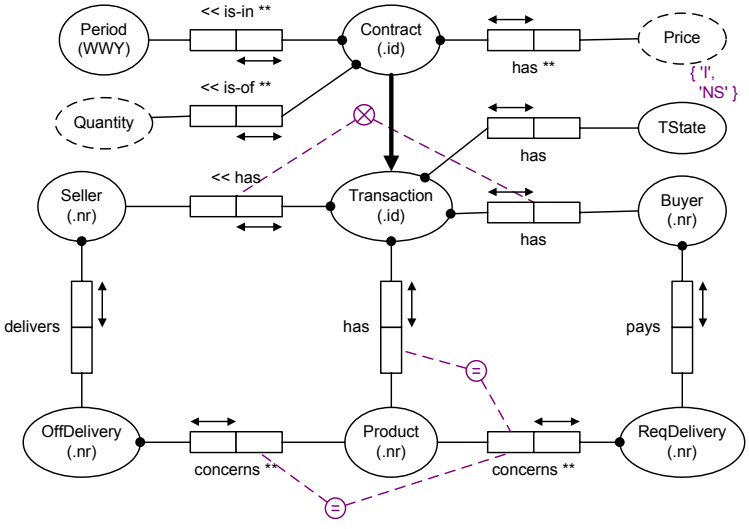

Fig. 3. Product delivery before alignment

In figure 2, the double asterisks on the relations is-in, is-of, and has indicate that these relations are derived relations whose values are stored. In terms of databases, these relations are part of a *materialized view*.

Now that we have the basic data elements, we can start thinking about how to model the B2B transaction between Buyer and Seller. First of all, it should be noted that Buyer and Seller have to agree on some common notion of Product. Usually, Buyer and Seller will have their own, proprietary local data (naming conventions, units of measurements, etc.) pertaining to the product. These local data models of Product can give rise to the problem of semantical heterogeneity, as discussed earlier. We will assume that these local data inconsistencies (naming conflicts, different units, etc.) have been resolved, resulting in a common object type called Product (as depicted in figures 1 and 2) with derived and stored relations OffDelivery concerns** Product and ReqDelivery concerns** Product.

In modeling transactions, we adopt the view as taken in [4,7], where transactions are modeled as objects which can be subjected to *dynamic constraint properties*. We will discuss dynamic constraints further on, and first concentrate on modeling the basic structure of a transaction. In our case, the transaction pertains to a Buyer and a Seller negotiating on the delivery of some product. The transaction will result in a contract if certain conditions are met. Consider the model of Transaction shown in figure 3.

Our Transaction involves a Buyer, a Seller, and a Product. A Buyer requests a delivery (ReqDelivery), and a Seller offers a delivery (OffDelivery). A Transaction is either initiated (state `I') or results in a successful negotiation (state `NS', of *Negotiation Successful*)). Buyer and Seller are never the same entity, and the Product is the same for Buyer and Seller. A Contract is defined as the subtype of Transaction with state `NS'.

Now we still have to define what a successful negotiated transaction is. This is called *service alignment* [6,27]: we have to align the requested service of the Buyer

with the offered service of the Seller. In our case, we have assumed the following informal rule defining a successful transaction: *If Seller promises to deliver the product, and Buyer promises to pay after delivery of that product, and furthermore the Requested Period is within the Offered Period, the Requested Price is higher than the Offered Price, and the Requested Total Quantity is smaller than the Offered Total Quantity, then the state of the transaction changes from Initiated to Negotiation Successful.* Note that in this informal rule, most of the concepts have already been introduced in previous model fragments, but some essential concepts are still missing; in particular we miss the following fact types:

- *Seller promises to deliver the product*
- *Buyer promises to pay after delivery*

We model these concepts by adding to OffDelivery a fact type OffDelivery has ODState; this ODState denotes the state of the offered delivery, which can have two values: `PD' (*promised to deliver*) and `NPD' (*not promised to deliver*). To ReqDelivery, we similarly add fact type ReqDelivery has RDState, where RDState denotes the state of the requested delivery, also having two values: `PPD' (*promised to pay after delivery*) and `NPPD' (*not promised to pay after delivery*).

We can now add a dynamic rule, stating that if a transaction satisfies the requirement RDState=`PPD' and TState=`I' and RPe ≤ OPe and OPr ≤ RPr and RQ ≤ OQ, then we can change the ODState to `PD' and the TState to `NS'. In a more formal fashion (employing the notation as introduced in [4,7]), we have the following specification for this rule:

For each Transaction
added TState=`NS' **if and only if**
previous existing ODState=`PD' **and previous existing** RDState=`PPD' **and**
previous existing TState=`I' **and added** RPe ≤ **previous existing** OPe **and**
added OPr ≤ **previous existing** RPr **and added** RQ ≤ **previous existing** OQ

The reason why we call this rule *dynamic*, is now obvious: adding a new transaction is governed by checking a previous state of that transaction object against the next state of that object. This dynamic rule offers a formal description of a particular *service alignment* between Buyer and Seller, and shows how a negotiation process has to be guided to reach a successful result.

At the time of writing this paper, dynamic rules in ORM are subject of further research aimed at proposing a standard syntax (FORML, of Formal ORM Language) for both static and dynamic constraints within the ORM modeling language, which will be followed through by extending the NORMA tool [10] with support for FORML-expressions. A formal semantics for dynamic rules in ORM, based on first-order predicate logic, was given in [7].

Key to the formalization of a dynamic rule is that the transaction to which the rule applies is given a log; i.e., a history is kept of all events (add, delete, update) that have taken place with respect to this transaction. The semantics of a dynamic rule is then captured as a static constraint on the log of the associated transaction.

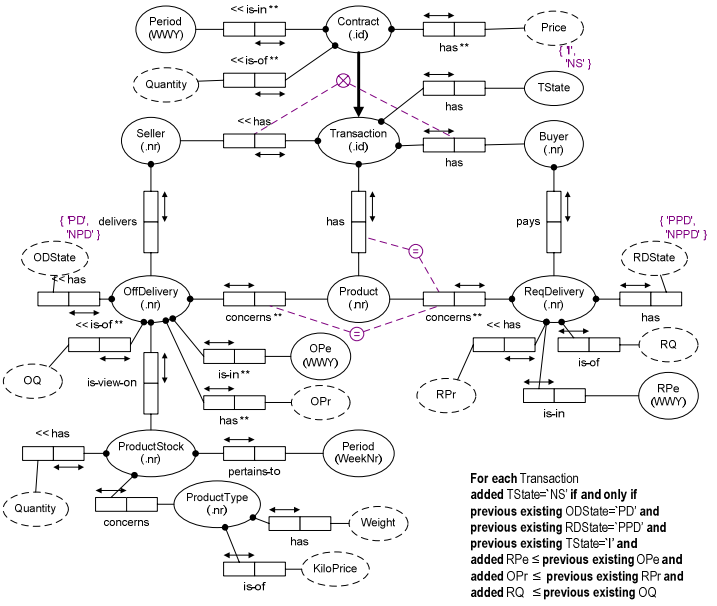

Fig. 4. Product delivery after alignment

If we put all of the pieces of the B2B process together (cf. figures 1, 2, and 3, plus the dynamic rule specification), we obtain a complete model of our B2B transaction, as pictured in figure 4.

We note that there is a striking similarity of previous work done on Service Discovery ([27]) as offered in [6]. Paper [6] treats modeling of outsourcing services in a context of negotiation between customer and provider. The main differences with the approach adopted here is that in [6] the modeling took place in the UML/OCL framework ([28]), and that the negotiation process itself had not been modeled as a database transaction governed by a dynamic constraint on transaction objects, hence offering a less precise modeling result, and a result further away from actual implementation.

Another useful aspect of modeling a B2B transaction as a transaction object is that the B2B model can be used as a schema for a negotiation database. The transaction is then kept as an object with historical information (we wish to maintain the history of the negotiation process). The ORM tools ([10,13]) will yield complete DDL-scripts of the associated database. We note that dynamic constraints could be implemented by database triggers.

We also wish to point out that we have modeled a B2B process (negotiation) as an object in a database having both static and dynamic behavior. The Transaction object represents the structure of a negotiation (Buyer, Seller, Product) embedded in a context of data alignment (are we talking about the same objects and do we understand all of the constraints involved on both sides (Buyer and Seller) of the negotiation) and service alignment (do both parties agree on what constitutes a successful negotiation resulting in delivering a product). Hence, because we are now

talking about a combined data (static aspects) and process model (dynamic aspects), we take the modeling process one step further than those researchers that focus on process or data alone or separate, such as Hofreiter et al [17].

5 Transaction Life Cycle

In this paper the Transaction is modeled as one single entity. In practice a Transaction may consist of multiple deliveries and multiple payment events. In the negotiation process Buyer and Seller increase their commitment level for making the delivery and payment events happen. Initially the Buyer and Seller identify the goods and services to deliver and the amount of money to pay. When the Seller issues a legally binding quotation or offer, in fact he commits to deliver the identified goods and services under the condition that the Buyer commits to pay after delivery. Awaiting the commitment of the Buyer, the Transaction then is in the state "Quoted". When the Buyer expresses his commitment to pay after delivery by signing the contract, the Transaction state shifts to "Negotiation Successful". In this state the Seller has a firm commitment to deliver (his Offered Delivery is in the state PD) and the Buyer has a commitment to pay under the condition that delivery has taken place (the Requested Delivery is in the state PPD). When delivery happens, the Transaction state changes to "Delivered" and the commitment of the Buyer to pay is now firm. Finally, when the payment is made, the Transaction arrives in its final state "Paid", and no residual commitments exist anymore (cf. figure 5, below).

Fig. 5. Business transaction lifecycle

In this paper the various state transitions of the business transaction have been defined as a set of database transactions. Static and dynamic constraints on the database transactions define which business transaction state transitions are allowed at any time. The full transaction history is logged: tuples are only added, not changed or removed.

6 Conclusion and Future Work

We have shown that a basic business process such as a business transaction negotiation can be modeled as a database equipped with static and dynamic constraints. The mechanism to model state transitions by means of dynamic constraints can also be applied to more complex situations. Our hypothesis is that any arbitrary complex B2B business process may be modeled by means of the mechanisms as shown in this paper. In future work we shall challenge this hypothesis and elaborate on more business process classes.

As our modeling language we have chosen ORM. We have used ORM in conjunction with an ORM constraint language as proposed in [4,7]. ORM is basically

a data modeling language; the use of dynamic constraints, however, makes it possible to represent business processes in an ORM model in a coherent and consistent manner.

References

1. Angelov, S.: Foundations of B2B electronic contracting; Technical University Eindhoven; PhD Thesis (2005) ISBN 90-386-0615-X; NUR 982
2. Balsters, H., de Brock, E.O.: Integration of integrity constraints in federated schemata based on tight constraining. In: Meersman, R., Tari, Z. (eds.) OTM 2004. LNCS, vol. 3290, pp. 748–767. Springer, Heidelberg (2004)
3. Balsters, H., Huitema, G.B., Szirbik, N.B.: Semantics of Agent-Based Service Delegation and Alignment. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM-WS 2005. LNCS, vol. 3762, pp. 109–120. Springer, Heidelberg (2005)
4. Balsters, H., Carver, A., Halpin, T., Morgan, T.: Modeling Dynamic Rules in ORM. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM 2006 Workshops. LNCS, vol. 4278, pp. 1201–1210. Springer, Heidelberg (2006)
5. Balsters, H., Halpin, T.: Data Integration with ORM. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM-WS 2007, Part I. LNCS, vol. 4805. Springer, Heidelberg (2007)
6. Balsters, H., Huitema, G.B.: Semantics of Outsourced and Interoperable Information Systems. In: Morel, M. (ed.) Interoperability for Enterprize Software and Applications. Springer, Berlin (2007)
7. Balsters, H., Halpin, T.: Formal semantics of dynamic rules in ORM. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM-WS 2008. LNCS, vol. 5333, pp. 699–708. Springer, Heidelberg (2008)
8. Bakema, G., Zwart, J., van der Lek, H.: Fully Communication Oriented Information Modelling. Ten Hagen Stam, The Netherlands (2000)
9. Chen, P.P.: The entity-relationship model—towards a unified view of data. ACM Transactions on Database Systems 1(1), 9–36 (1976)
10. Curland, M., Halpin, T.: Model Driven Development with NORMA. In: Proc. 40th Int. Conf. on System Sciences (HICSS 40). IEEE Computer Society Press, Los Alamitos (2007)
11. Embley, D.W.: Object Database Development. Addison-Wesley, Reading (1997)
12. Halpin, T.: A Logical Analysis of Information Systems: static aspects of the data-oriented perspective, doctoral dissertation, University of Queensland (1989), `http://www.orm.net/Halpin_PhDthesis.pdf`
13. Halpin, T.A., Evans, K., Hallock, P.: Database Modeling with Microsoft® Visio for Enterprise. Morgan Kaufmann, San Francisco (2003)
14. Halpin, T.: ORM 2. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM-WS 2005. LNCS, vol. 3762, pp. 676–687. Springer, Heidelberg (2005)
15. Halpin, T.: ORM/NIAM Object-Role Modeling. In: Bernus, K., Mertins, K., Schmidt, G. (eds.) Handbook on Information Systems Architectures, 2nd edn., pp. 81–103. Springer, Heidelberg (2006)
16. Halpin, T., Morgan, T.: Information Modeling and Relational Databases, 2nd edn. Morgan Kaufmann, San Francisco (2008)
17. Hofreiter, B., Huemer, C.: Modeling business collaborations in context. In: Meersman, R., Tari, Z. (eds.) OTM-WS 2003. LNCS, vol. 2889, pp. 829–844. Springer, Heidelberg (2003)

18. Inmon, W.: Building the Data Warehouse, 1st edn. Wiley and Sons, Chichester (1992)
19. Jayaweera, P.: Unified Framework for e-Commerce Systems Development: Business Process Pattern Perspective; Department of Computer and Systems Sciences Stockholm University and Royal Institute of Technology; PhD Thesis (2004)
20. Kimball, R.: The Data Warehouse Toolkit, 2nd edn. Wiley and Sons, Chichester (2002)
21. Moore, S.A.: A communication framework for applications. In: 28th Hawaii International Conference on System Sciences, HICSS 1995 (1995)
22. Oasis 2007, Web Services Business Process Execution Language Version 2.0, OASIS Standard (2007)
23. Object Management Group, UML 2.0 Superstructure Specification (2003), http://www.omg.org/uml
24. Object Management Group, UML OCL 2.0 Specification (2005), http://www.omg.org/docs/ptc/05-06-06.pdf
25. Object Management Group, Semantics of Business Vocabulary and Business Rules (SBVR) Specification (2007), http://omg.org/technology/documents/bms_spec_catalog.htm#SBVR
26. Van der Aalst, W.P.M., ter Hofstede, A.H.M.: YAWL: Yet Another Workflow Language, QUT Technical report, FIT-TR-2002-06, Queensland University of Technology (2002)
27. Vinoski, S.: Service Discovery 101. IEEE Internet Computing, 7(1) (2003)
28. Warmer, J., Kleppe, A.: The Object Constraint Language, 2nd edn. Addison-Wesley, Reading (2003)