On the Interaction between Business Models and Software Architecture in Electronic Commerce

Jaap Gordijn
Bakkenist Management Consultants/Vrije Universiteit Amsterdam
De Boelelaan 1081a, 1081 HV Amsterdam, fax: +31 (0)20 444 7653
e-mail: gordijn@cs.vu.nl

Hans van Vliet
Vrije Universiteit Amsterdam
De Boelelaan 1081a, 1081 HV Amsterdam, fax: +31 (0)20 444 7653
e-mail: hans@cs.vu.nl

Abstract. To design a business model for electronic commerce it is necessary to develop an architecture of the software system that will support this model. Such an architecture shows at an early stage to what extent the business model can be supported and implemented by a software system. If constraints of the business model cannot be supported by a software system, the model itself can be changed. We investigate this interaction between business model and architecture by carrying out a case study in the area of electronic commerce: the design and analysis of a software architecture for a directory service.

Keywords. Electronic commerce, business model, software architecture, assessment, directory service, fraud, case study.

Introduction

From our consultancy experience, it shows that the realization of an electronic commerce system is a far from trivial task. On the one hand, such a system poses rather stringent business constraints pertaining to, for example, the prevention of committing a fraud. On the other hand, an electronic commerce system should support a profitable way of doing business.

The development of a business model and the belonging commerce software system are both design problems. A business model shows at a conceptual level the way of doing business by defining the exchange of values between actors such as customers and sellers, and constraints such as the prevention of committing a fraud. Values can be money, services or (digital) goods. Since it is not always clear from the beginning which value exchanges exist or even which actors participate, we have to take many design decisions during the development of a business model. The design of a business model is largely influenced by the interests and power of the various commercial actors involved and the goals they want to reach. Decisions are business oriented. A software architecture of an electronic commerce application shows the conceptual- and logical design of a system by indicating software components and relationships between components. It reflects the earliest set of information technology related design decisions with respect to the system supporting the business model. Decisions are taken by amongst others IT-designers and maintainers but not by business oriented people.

A problem in designing electronic commerce applications is that business- and architectural design decisions are sometimes intertwined. Consider prevention of committing fraud. Possibly, such an issue can be addressed by developing software supporting the prevention of committing a fraud and therefore can be seen as a design decision on the software architectural level. However, it might also be possible or sometimes even necessary, to redesign the business model in such a way that it supports prevention of committing fraud by itself, for instance by creating a conflict of interests between actors. Creating such a conflict is a design decision on the business level.

Therefore, we claim in order to develop an electronic commerce application, it is necessary to assess an initial business model by developing a software architecture. Such an architecture should be evaluated with respect to value exchanges and constraints mentioned in the business model. In some cases, the software architecture does not address all value exchanges well or does not respect all constraints. Another software architecture can be considered but in some cases it is necessary to adjust the business model itself, for instance if it shows up that certain constraints can't be met on the architectural level.

We explore our claim by carrying out a realistic electronic commerce case study. We develop a business model and a software architecture for an electronic commerce system, and next assess the extent to which that architecture supports the business model. It turns out that the software architecture does not allow the realization of certain constraints posed by the business model. The business model is then adapted in order to support these constraints.

Section 2 discusses the relationships between business models, quality attributes, and software architectures. Our choice for a particular business model is elaborated in sections 3 and 4. Section 5 discusses the software architecture, and some tradeoffs between software architectures and business models. Section 6 contains some conclusions.

How to develop electronic commerce systems

A first step in developing an electronic commerce software system is to define one or more *business models*. A business model outlines the electronic commerce service for all actors. It states amongst others the actors involved and the exchange of values between actors. Actors are the participating parties in the electronic commerce application such as sellers, customers and intermediaries. Values can be products, services or money. A business model also states constraints, for example "committing a fraud should be prevented". Developing a business model requires intensive communication between decision-makers, domain experts and business developers.

A next step is the identification of *quality attributes* of the electronic commerce application. Different quality attributes may be important, such as functionality, performance, security, system modifiability, and portability. These quality attributes, as well as other factors such as the developing organization, the technical environment and the experience of the architect all effect the software architecture [Bass98]. It is not easy and in most cases impossible to design a software architecture fulfilling all quality attributes. Using trade-off analysis we can balance quality attributes and make architectural choices. For electronic commerce applications we sometimes have an additional possibility in the case of conflicts between qualities. We can change the business model in such a way that an architecture can be designed which has fewer conflicts between quality attributes. Changing the business model means the introduction of new actors, a change in the exchange of values or new constraints. The resulting relationship between business models, quality attributes and architecture is depicted in Figure 1.

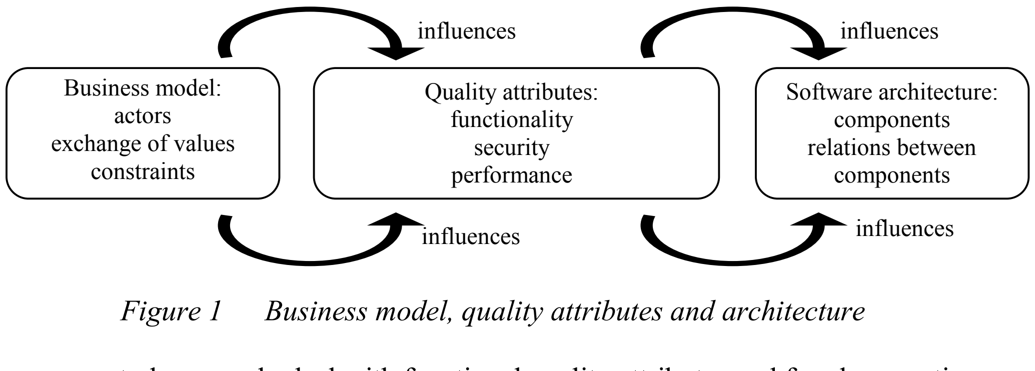

Figure 1 Business model, quality attributes and architecture

In our case study, we only deal with functional quality attributes and fraud prevention quality attributes. First we consider functional quality attributes only, and design a draft architecture meeting these attributes. We evaluate this architecture by assessing the extent to which fraud prevention is achieved. If the architecture does not meet some of the attributes, we redesign the architecture, but leave the functional quality attributes as they are. This approach is proposed in [Bosch98]. Some fraud prevention requirements, though, can hardly be addressed by only re-designing the architecture. However, we can also change the business model, for instance by adding actors or changing the way values are exchanged. If we change the business model, we have to re-identify quality attributes and redesign the architecture.

The business model will be defined using natural language. We concentrate on the actors, the exchange of values between these actors and constraints such as fraud prevention. We use text-based use cases to define the functional quality attributes. Use cases describe a typical interaction between the actor and the system offering the directory service. Fraud prevention quality attributes are also described using natural language. We use interaction diagrams as the leading view on the software architecture [Fowler97,UML97]. These diagrams show components interacting with each other using sequences of messages. Components act on behalf of an actor. An interaction diagram is useful because it gives a clear view of the outline of the directory service. Furthermore, the exchange of messages can be analyzed for opportunities to commit a fraud. We distinguish a number of interaction diagrams, one for each use case.

Business models for a directory service – a case study

Currently, the Internet consists of a large, increasing number of web sites, representing companies selling products or services. For potential customers, it is difficult to find the product or service that best suits their needs. To a large extent, this is due to information overload [Bollier96]. There is a need for an intermediary that brings potential customers and sellers together. Search engines, and more specifically directory servers have recognized this fact. Directory servers try to match a customer and seller based on the product or service the customer is looking for.

More and more, a directory service is becoming a commercial undertaking. Such a service then needs a business model, which states how money can be earned by exchanging values. Several different value exchanges are possible. For instance, since the directory service is a web site, advertisements can be shown for each customer querying the service. The ad to be shown possibly depends on the customer query [Aggar98]. Another value exchange focuses on selling marketing information [Kannan98]. The directory server collects queries from identifiable customers to construct an individual client profile that can be sold. These value exchanges both rely on payment by parties who are not the actual users of the service. A third value exchange involves payment by the actual *users* of the service. In this paper, we consider this latter model.

For each value exchange, one has to associate a pricing scheme. A pricing scheme shows the calculation of the price to be paid for service or product. For instance, a possible scheme is to charge a seller a yearly fee for unlimited mediation by the directory server. The disadvantage of such a pricing scheme is the difficulty to determine whether the fee is reasonable, both for the seller and for the directory server itself. The seller does not know whether the fee to be paid is reasonable. In case only a few mediations occur, the seller pays too much. The directory server will have a hard time convincing the seller to pay the fee and therefore has a selling problem. Another possibility is to use a pricing scheme which is based on the effectiveness of the directory server for a particular user of the service. For this case, we investigate the latter pricing scheme.

Effectiveness of a directory service

Effectiveness refers to the extent to which a stated goal of a particular actor is reached. Depending on the value exchanges outlined in the previous section, a directory server can have multiple goals. Its goal may be to sell as many ads as possible or to sell high quality marketing information. The goal of both the seller and the customer is a successful match. In this paper, we focus on the last goal.

A mediated match between seller and customer can have different meanings: (1) a potential customer gets a *list* with sellers offering a product or service, (2) a potential customer *visits* the shop of a seller, and (3) a *business transaction* between customer and seller occurs. Both seller and customer are willing to pay for each kind of match (Table 1).

Match→ Paying actor↓	list of products	shop visit	business transaction
seller	seller pays to occur (high) on the list of potential sellers in order to increase the chance to be selected	seller pays for shop-entrance to increase the chance the customer buys something	seller pays for a business transaction occurred
customer	Customer pays to get a list of most relevant sellers so he can select the most appropriate seller	customer pays if he thinks a shop can fulfill his needs	customer pays for business transaction occurred

Table 1 Actors paying for different kinds of matches

In this case, we assume that only the *seller* pays the directory server, as is common in current services of that kind. However, no principal reason exists why the customer would not pay. In our approach, seller and directory server are both in need of measuring effectiveness. The directory server is interested since he wants to send a bill to the seller based on effectiveness; the seller wants to validate this bill.

For sellers, it is most interesting to judge effectiveness of a directory service using the *transaction* based definition of matching. Ultimately, the thing they are most interested in is closing a business transaction. In this case, we employ the business transaction based definition of matching. However, the 'list of sellers' and 'shop visit' interpretations of matching as illustrated in Table 1 can be combined with the transaction based definition of matching.

Once we can measure effectiveness of the directory service reliably, payment for the service can be based upon it. The seller pays an amount of money, for instance a dollar per match or a percentage of the transaction value, to the directory server. Presumably, this is only acceptable for both the seller and the directory service if committing a fraud is impossible.

The business model we employ in this paper can be summarized as follows. Actors are a directory server, customers, and sellers. The customer queries the directory server for free; the seller pays the directory server a fee for each business transaction that was mediated by the directory server. Finally, committing a fraud should not be possible.

An architecture for measuring the effectiveness of directory services

To assess the feasibility of our business model, we design a basic software architecture focussing on functional quality attributes. We evaluate this architecture with respect to fraud opportunities and the redesign the software architecture accordingly. As it turns out, we cannot address all fraud opportunities without changing the business model.

An architecture focussing on functional quality attributes

We describe functional quality attributes with use cases below, along with a sequence diagram as our main architectural view. Use cases are derived from the business model.

Use case: Query. The customer queries the directory server. The directory server has a database containing product (types) offered by sellers. The directory server returns hits to the customer. A hit contains information about a seller offering a product.

Use case: Buy

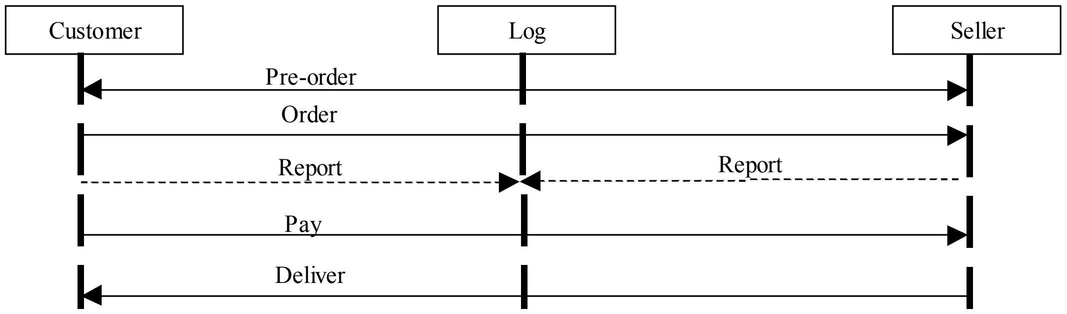

Figure 2 Buying a product

The customer uses hits to select a seller. He may visit several sellers and compare offerings in detail. We refer to this as pre-ordering communication. The pre-ordering communication is not in the scope of this case; we assume that in the end the customer selects a seller. After selecting a seller, a business transaction starts. In principle, the business transaction is between customer and seller and has three parts: (1) ordering, (2) payment and (3) (confirmation of) delivery of the product or service. Either the customer or seller reports the business transaction to a log. The log contains a list of all successful matches. Since we assume an honest environment, the log can be located anywhere. It can be a separate, independent actor or it can be part of the seller, directory server or even the customer. Because we discuss only functionality, we ignore the location of the log and simply show it as a separate entity. Either the seller or the customer has to report the match to the log. At this time, we have no criterion to make this decision. Therefore both options are represented by a dashed line. If the seller reports the transaction, the order should include a field indicating the originating directory server. Otherwise, the seller cannot report the mediating directory server

Use case: Report. Both the directory server and the seller are interested in a periodical report of successful matches. The directory server is interested since the report can be used to charge the seller on a per-match basis. The seller can use the report to evaluate the effectiveness of the directory service, and also to check the bill he has to pay.

Use case: Bill. Periodically, e.g. after receiving a report of successful matches, the directory server sends the seller an invoice. The seller pays the invoice.

An architecture for a dishonest world

The draft architecture we have outlined so far provides an architectural overview of the service to offer. However, we still have to address prevention of fraud. Recall this was a constraint in the business model. This constraint results in the following, more detailed, quality attributes. (1) Committing a fraud by seller and directory server should be prevented. Customer fraud is not assumed. This is a reasonable assumption since the customer is only interested in the content of the directory server and not in its effectiveness. (2) Fraud can being committed during processing (e.g. by the seller) or during communication. We assume that actors are capable of monitoring and tampering with network traffic. Due to space restrictions, we limit ourselves to fraud committed by the seller.

Fault tree-analysis

We evaluate the draft architecture with respect to committing a fraud using fault-tree analysis [Leveson86]. We briefly report on this analysis and continue with one of the more interesting possibilities to commit a fraud. Fault tree analysis is intended to analyze possible causes for hazardous events. We used a fault tree to analyze types of frauds and their causes.

In our case we can have two types of frauds: (1) the log contains too few matches (fraud by seller) and (2) the log contains too many matches (fraud by directory server). The first case refers to matches that did occur but were not logged; the second fraud is about matches that were logged but did not actually occur. For each type of fraud, a tree has been constructed consisting of the type of fraud and possible causes. Causes can be joined by an AND relationship (all causes should occur) or an OR relationship (at least one cause should occur). Causes are decomposed in sub-causes. This results in a hierarchy of causes. The analysis has been carried out for all outlined use cases. Figure 3 shows a fragment of the fault tree dealing with the fraud “too few matches”.

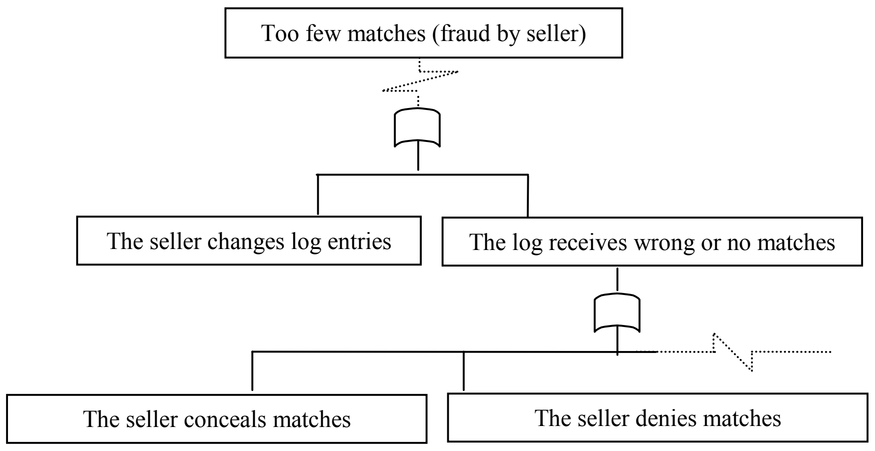
Too few matches (fraud by seller)
The seller changes log entries
The log receives wrong or no matches
The seller conceals matches
The seller denies matches

Figure 3 Fault tree analysis for "too few matches"

This part of the fault tree deals with the situation that the seller commits a fraud with occurred matches. First, the seller can change or delete matches in the log directly if he has access to that log. Second, the seller can conceal or deny matches or he can change matches during reporting of the match to the log. Recall that we did not yet decide on the issue which actor reports the match. In case the seller reports matches, it is easy for him to conceal matches or to change matches. If the customer reports matches, the seller may ransom the customer to conceal the match or change the match. In both cases, the seller is able to deny a reported match.

We can overcome some problems by applying technical measures. For instance, access to the log by a specific actor can be restricted. Non-repudiation of reported matches can be enforced by signing reports of matches by the seller [Menezes96]. The log stores signed matches which, if needed, can be verified afterwards. To use such a signature solution, the seller has to report the match. These measures imply a small change in the software architecture since messages exchanged between components change slightly.

We have not been able to find solutions such that concealment of matches is impossible by a mere technical solution. However, concealment can be addressed by changing a functional quality attribute and thereby changing the business model. In the next section, we propose an adapted model ensuring report of all matches by the seller.

In our opinion, it was necessary to carry out a fault-tree analysis on the initial architecture to reveal the problem of concealing matches.

Reward of the customer

Our fault tree analysis revealed that the seller or customer may conceal matches. It is in the interest of the directory server to have a report of *all* matches. We propose a solution that guarantees a complete log of occurred matches. Basically, we create a conflict of interest between seller and customer. The customer receives bonus points for each business transactions. He receives these points from the directory server or an actor representing the server. However, the directory server only gives his points away if he gets a report of a match from the seller. The seller pays a fee to the directory server for each match plus an additional charge for the bonus points. The conflict of interest results in a change in the exchange of values and thus changes the business model.

The business model can be redefined as follows. Actors are a directory server (plus a log operating on behalf of the directory server), customers, and seller actors. Customer queries the directory server for free. Seller pay the directory server a fee for each business transaction that was mediated by the directory server. The log pays the customer bonus points if the seller reports a match to the log. Sellers pay the directory server a fee for each paid bonus point. Customer receive gifts from the directory server in return for bonus points. This business model leads to the following changed use cases (Table 2):

Use case	Actors	Description
Buy	Customer Seller Directory server	Customer buys a product and the match is reported to the log (operating on behalf of the directory server) by the seller. Customer receives bonus point from the log (operating on behalf of the directory server)
Bill	Seller Directory server	The seller receives a bill for each match and pays
Handle gifts (new)	Customer Directory server	The customer receives gifts from the directory server in return for bonus points

Table 2 New use cases

A changed set of use-cases results in architectural changes too. For brevity, we discuss architectural changes caused by changes in the use-case 'buy' only.

Use case: Buy. After a customer orders a product, he receives bonus points. Many companies already are offering such bonus points as part of a customer loyalty program [Kannan98]. The directory server is the only actor who can convert the points to gifts. The log is the only actor who issues the bonus points (on behalf of the directory server). After the seller receives the order, he reports the match to the log. The log sends the bonus to the customer. The customer pays the seller and the seller delivers the product.

Transferring the bonus, reporting the occurred business transaction, paying the seller by the customer, and delivering the product must be an atomic operation, called good atomicity [Camp96]. Good atomicity ensures fair exchange of values. In this case, the customer receives his ordered good and bonus points, the log receives a report and the seller receives payment or in case something goes wrong, nobody receives anything. A trusted third party can be helpful to achieve this.

New frauds

Initially we assumed the customer does not commit frauds. However, by changing the business model, the customer has an interest in more bonus points. Therefore a change in the quality attribute regarding fraud prevention occurs. Besides the seller and directory server, the customer is a possible actor who may commit a fraud. A new fault tree has to be constructed for the new situation. As it turns out, the tree indicates a new way to commit a fraud.

In the new model, the seller pays the value of the bonus plus a fee to the directory server. The seller can cheat by offering the customer the bonus itself plus a fraction of the fee he would have paid to the directory server. In fact, it is interesting for the seller to offer the customer the following compensation:

$$\$\text{compensation}_{\text{to customer}} < \$\text{fee}_{\text{to directory server}} + \$\text{value of bonus}_{\text{to directory server}}$$

This fraud can be prevented as follows. The bonus is not a normal currency, but an own currency of the directory server (bonus points). It is interesting for the customer to save these points since the directory server has nice gifts. Therefore, the customer wants to collect these gifts from all sellers he is buying products from. The customer makes no exception for one seller; he wants bonus points instead of money. Moreover, the fee represents a small fraction of the value of the transaction so the customer is not interested in a few dollar cents.

Now, consider the existence of a significant number of fraudulent sellers. The customer accumulates money received from fraudulent sellers and buys gifts in a normal shop. To prevent this, the gift he wants to receive from the directory server must be substantially cheaper to obtain using bonus points than with real money. Therefore, the following should hold:

$$\$\text{gift}_{\text{obtained via directory server}} < \$\text{gift}_{\text{obtained via regular shop}}$$

The directory server can accomplish this by buying gifts in large amounts and negotiating bulk prices. Therefore, the maximum fee directly depends on the ability of the directory server to buy gifts against low prices.

Lessons learned

This paper began with the statement that it is useful to develop a software architecture of an electronic commerce software system to assess to what extent that systems supports the chosen business model. In case the software architecture reveals that some constraints of the business model cannot be met, the business model itself can be adjusted.

We found, by investigating a case, that quality attributes resulting from an electronic commerce business model may in some cases only partly be addressed by technical means in a software system. Other quality attributes might have to be addressed by changing the business model. For instance, non-repudiation of reported matches can be supported using digital signatures, a technical measure. Also, a direct change of entries in the log can be addressed by using access control and limited usage of system functions by actors. However, we have not found any technical measure to prevent concealment of matches by the seller. It is possible to handle this case by changing the business model, for instance by creating a conflict of interest between seller and customer. This way, the seller is forced to report all matches. We consider the creation of conflicts of interest as a change in the business model. The conflict of interest is created using bonus points which are exchanged between actors. This implies a change in the exchange of values.

We assessed to what extent a software system supports an electronic commerce business model, using a software architecture. We chose an initial business model and we identified quality attributes. The assessment of a first draft architecture revealed that some quality attributes can be fulfilled by a software system but other quality attributes forced us to change the initial business model. Therefore, designing a software architecture in an early stage, was very helpful to come to a more mature business model.

References

Aggar98	C.C. Aggarwal, J.L. Wolf and P,S. Yu, A Framework for the Optimizing of WWW Advertising,Trends in Distributed Systems for Electronic Commerce, W.Lamersdorf, M.Merz (eds.), Springer, Berlin 1998
Bass98	L. Bass, P. Clements, R. Kazman, Software Architectures in Practice, Addison-Wesley, 1997
Bollier96	D. Bollier, The Future of Electronic Commerce, The Aspen Institute (publications@aspeninst.org), 1996
Bosch98	J. Bosch, P. Molin, Software Architecture Design: Evaluation and Transformation, http://bilbo.ide.hk-r.se:8080/~bosch/ , Submitted, 1998
Camp96	L. Jean Camp, Privacy & Reliability in Internet Commerce, CMU-CS-96-198, Carnegie Mellon Technical Report, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA. 1996
Choi97	S.Y. Choi, D.O. Stahl, A.B. Whinston, The Economics of Electronic Commerce, Macmillan Technical Publishing, Indianapolis, 1997
Fowler97	M. Fowler, K. Scott, UML Distilled, Applying the Standard Object Modelling Language, Addison-Wesley, Reading Massachutes, 1997
Kannan98	P.K Kannan, A.M. Chang and A. B. Whinston, Marketing Information on the I-Way: Data Junkyard or Information Gold Mine?, Communications of the ACM, Vol41, No3, March 1998
Kazman98	R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, J. Carriere, The Archicture Tradeoff Analysis Method, 4th International Conference on Engineering of Complex Computer Systems, Augustus 1998
Leveson86	Nancy G. Leveson, Software Safety, Why, What, and How, Computing Surveys, Vol. 18, No 2, June 1986
Menezes96	A.J. Menezes, P.C. van Oorschot, S.A. Vanstone, Handbook of Applied Cryptography, CRC Press Inc., Boca Raton, Florida, 1996
UML97	UML Notation Guide, http://www.rational.com/uml/documentation.html, 1997