Is There Life Outside Transactions? Writing the Transaction Processing Book

Andreas Reuter
European Media Laboratory
Schlosswolfsbrunnenweg 33
D-69118 Heidelberg
+49-6221-533200
reuter@eml.org

ABSTRACT

In this article I will reflect on the writing of "Transaction Processing – Concepts and Techniques" [1], which appeared at Morgan Kaufmann Publishers in 1992. The process of writing had many aspects of a typical software project: In the end, the book was more than twice as thick as we had planned, it covered only ¾ of the material that we wanted to cover, and completing it took much longer than we had anticipated. Nevertheless, it was a moderate success and served as a basic reference for many developers in the industry for at least 10 years after its publication. It was translated to Chinese and Japanese, and occasionally one still finds references to it – despite the fact that (apart from simple bug fixes) there has been no technical update of the material, and the book deals with "outdated" subjects like transaction processing and client/server architectures.

Categories and Subject Descriptors

K.2 [**History of Computing**]: People

General Terms

Performance, Design, Reliability, Standardization.

Keywords

Transaction processing, fault tolerance, software architecture.

1. INTRODUCTION

In 1986 Jim had signed a contract with a seminar organizer for teaching a one-week course on transaction processing in spring 1987. The course was to take place in Berlin, and because he did not want to teach the full five-day load all by himself, he invited me to share some of it, assuming that university professors have most – if not all – of the course material ready for delivery on short notice. For the following eight months we worked on preparing the slides. The "plan" guiding the process was a list of chapter headings that each of us would work on, with each chapter representing a 90 min lecture. There was very little communication along the way: A phone call now and then, but no exchange of drafts or anything like that. Exchanging emails between a university and a (closed) company environment – Tandem in our case – was not something you could simply do, and slides were either drawn by hand or printed and then copied onto transparencies[1]. Two weeks before the course started we sent a complete set of paper copies of our 980 foils to the organizer, who turned them into handouts for the participants. Delivering them all would have been quite a challenge, but as soon as we found out that the participants were very knowledgeable developers with good ideas and tricky questions (I remember Bill Laing and Paul Shrager being among them), Jim decided to change the course structure completely. Instead of following the table of contents in the handouts we focused on what the participants were interested in and skipped those parts that nobody wanted to hear about. For example, we discussed a design where (for reliability) the requests are sent to two completely independent systems and executed on both. The question was, under which conditions one could guarantee that the transactions are serialized in the same order on both systems. So it was a very lively, highly interactive course. I do not know how much the participants took home, but the two presenters definitely learned a lot.

2. ROBERT BURNS WAS RIGHT

When we reviewed the course, Jim observed that the students' questions had produced something that our initial organization of the material had not suggested: A systems-oriented perspective on transaction processing. So instead of describing TP technology in isolation, and then describing databases, networking, programming issues, etc. we presented transactional concepts as some kind of unifying framework for all layers of a system, from the operating system all the way up to the applications and the user interfaces. Jim went on saying that there were no textbooks taking that integrative approach, and that it should not be too hard to turn our 980 foils into text. We estimated that on average two slides would transform into one page of prose (including tables and figures), so that we would have to write ca. 500 pages – 250 pages per person. Doing this within a year seemed quite reasonable at the

[1] In 1987 Microsoft bought the company that had produced the precursor to Powerpoint, but it took a couple of years for that to impact our teaching habits.

time, given that we had most of the material already. This is how the whole thing started.

Before we actually tried to implement it, the initial plan seemed to make perfect sense: The major portion of the work had already been completed by putting all the technical material on the slides – or so we thought. We would only have to convert bullets into sentences, redo some of the figures, add a chapter drawing all the details into the grand picture of transaction-oriented systems, compile a list of references – and be done! It was the kind of plan that everybody will enthusiastically agree to at the end of a meeting, so they can get on with their real work. In our case it was the review meeting after the course, and neither Jim nor I had a clear idea of how to implement it after we got back from Berlin. However, with the best of intentions we agreed on producing text from the slides some time soon.

With no deadline at all and many other things to do, we made very little progress in turning the foils into prose – in fact, we did not make any progress at all. I used the material for a variety of courses I taught at the university, extending and changing it as new algorithms, new systems etc. became available. Jim did the same, teaching transaction processing at Stanford, but we still were just using and updating the slides; no prose was being produced as a result of the teaching activities. The only new type of content that proved useful when – much later – we actually wrote the book was a rapidly growing number of problems and exercises related to the various topics that were covered in the foils. Those problems were specifically created for the university courses; they had not been part of the Berlin seminar.

That was the situation in 1987, and it did not change in 1988, or in 1989. In the fall of 1989 we discussed the project and found that the original plan had been a failure. It was obvious that if we wanted to get anything written, we would have to hide in some remote, quiet and pleasant place, equipped with PCs, printer, toner, with easy access to good food, and spend all our time typing – well, most of it. We figured that three months should be enough to produce a complete first draft of the book, the polishing of which could be done later, when we were back in our normal habitats. After some lengthy and careful deliberation it was decided to rent a house in a small village in Tuscany named Ripa (near Carrara) and spend February through April of 1990 there.

This time we got it partially right: At the end of April we had about 600 pages of text, thanks to Jim's strict regime that required each of us to produce 2,000 words per day, no matter which day. 600 pages were very close to our estimate – but they only covered less than half the topics we wanted to discuss. So in order to preserve the investment, we had to plan for a second hideaway, which took place one year later in Bolinas (north of San Francisco), again from February to April. At the end of this period, we had about 1,000 pages of text, plus a number of lessons learned[2]:

- We would not be able to cover all the material that was contained in the foils of the course.
- We would still have to do a lot of work in order to get the book to the printer (glossary, index, and proof reading).
- Writing a book is hard work; we would never do it again.

So Robert Burns was right indeed: The best laid plans …

3. ORGANIZING THE MATERIAL

In the years between the Berlin course and the time of finishing the book, technologies related to transaction processing, distributed computing, parallel databases, etc developed at a rapid pace. There is by far not enough space to list them all, but I will mention some that had a major influence on the way the book was structured.

First, transaction technology for distributed systems started to be used seriously on non-proprietary operating system platforms, i.e. Unix [4]. This partly was the result of transferring research results from academia into real products via start-ups. A particularly notable example for this was Transarc's Encina-system [7], which was the result of a multi-year research effort at CMU, led by Alfred Spector. Since Jim had been discussing with this group on a regular basis, he had very detailed insight into both the architecture and the implementation of the system.

Second, the ideas for making transactions a fundamental mechanism for reliable execution at all levels of a system rather than just use it for database applications were being transformed into real systems. The most advanced example in that category was Tandem's TMF [6], which demonstrated the use of transactions in the operating system and featured real transactional RPCs, among other interesting things. Of course, Jim was particularly familiar with that system, so we often used it as a reference when discussing how the elements of a "good" transaction processing system (for teaching purposes) should play together.

Third, C. Mohan of IBM had started to systematize and clarify many techniques for implementing transactional execution that had been around in various systems for many years and present them in a coherent framework. This resulted in a famous series of papers on the ARIES design [5], which had a significant influence on how we presented recovery mechanisms and methods for synchronization on B-tree structures, among other things.

In several companies and many research labs people were working on new synchronization protocols, disaster

[2] There was yet another lesson, having to do with the deeply rooted connection between transaction processing and the level of precipitation in the area one writes about it - but that is beyond the scope of this short article.

recovery, new access paths, and - with a special emphasis – on generalizations of the classical ACID transaction model. The work that had most influence on our book in that respect was the ACTA model [8] and related ideas by Johannes Klein [9], which we mostly learned about by talking to him, because unfortunately most of those ideas never got properly published.

When we prepared for driving to Tuscany from my place in Germany, two cars loaded with computers, clunky 15" CRTs, printers and the more mundane stuff you need for three months away from home, I was very concerned about the heavy load of papers, proceedings, books etc we would have to take with us – remember, the Web was not there yet, and email was by far not as flexible as it is today[3]. But Jim argued that we would need very little of that, just our foils, some textbooks, some manuals, and a few papers: "We won't write about any exotic stuff; only the things that work – and those we know reasonably well." This principle was re-asserted many times throughout the writing process. Whenever I started saying something like "Shouldn't we try to find a new way of …." Jim would respond: "Don't invent. We just write about what we know." In other words: He wanted the book to be as specific and concrete as possible, in order to provide the highest potential benefit for the readers. Of course, these principles were not used in a dogmatic fashion. There actually were many situations when we had to "invent" a good way of presenting complicated issues in an appropriate manner rather than following the countless complications of some real implementation. This was the case, for example, with the transaction manager, with context management, with the implementation of a log manager, with B-tree locking and a number of other aspects. But in each case we did not come up with new algorithms; the only things we invented were suitable simplifications and/or abstractions that allowed for a more compact presentation.

Regarding the overall "gestalt" of the book, we had two role models: The first one was Patterson and Hennessy's newly published "Computer Architecture – A Quantitative Approach" [2]. Jim was absolutely enthusiastic about it: its basic approach of combining qualitative descriptions with quantitative models; its heavy use of exercises and sample solutions; its level of granularity; its layout – everything. Throughout the six months of writing we spent a good deal of our time figuring out how to make the various issues of transaction processing amenable to quantitative analysis. The second guiding light was Tanenbaum's new book on operating systems [3], which had come out about two years before. The interesting aspect of it was that it contained the complete source code for a basic operating system called Minix. Casting the ideas presented into source code that the reader can easily run (and modify) is certainly as close to "real life" as you can get in a textbook – a genuinely rational, if not quantitative, approach. We were so impressed with this idea that we decided to present all the relevant algorithms in syntactically correct C code (instead of some kind of C-like pseudo code). As a matter of fact, in the beginning we hoped to be able to complement the book with a CD containing a rudimentary yet executable TP system. Everybody who has read the book will remember what happened to that idea.

[3] Not that it would have mattered: The phone in the house in Ripa only allowed incoming calls. And since there were no mobile phones either: Can you imagine how much concentration we were able to focus on writing the book?

Despite all these initial considerations, when writing finally started in Ripa we proceeded in an ad-hoc fashion. Each of us started writing about something he felt comfortable with; there was no grand design, no master plan. Jim wrote about logging, producing the core of what finally became Ch. 9. I began writing about transaction models and transactional execution, and that stuff ended up in Ch. 4, 5, 6, 10, and some other places.

In hindsight I think we just wanted to see some text being written, after all these years without a single line. There was the clear understanding that all the material was just preliminary, ready to be re-arranged whenever that should prove necessary. And, of course, there was Jim's relentless "2000-words-a-day"-rule. We produced about 300 pages in that manner - and were quite proud of how well things were going – when we noticed that in the end we would just have a loosely organized collection of algorithms, techniques and tricks that could be found in various systems and that had to do with transaction processing one way or the other. But the title of the book was supposed to be "TP – Concepts and Techniques", and so far we had not delivered on the "concepts" part. That was a bit ironic, because the experience from the Berlin course, where the "vision" of a fully transactional systems architecture had emanated, was the real reason for getting started with the book writing business – and now we had almost forgotten about it for all the fun we had with describing our favorite algorithms and turning (simplified versions of) them into C code. At that point we began discussing the architectural issues, i.e. the question of how to introduce the concepts underlying everything that was being discussed in the book. Put in a slightly different way: What exactly was our idea of an "ideal" transaction processing system, taking into account all the experiences from existing systems? On one hand, we did not want to invent things. On the other hand, all the existing systems were compromises of sorts, thus introducing complications we did not want to get into. From that moment on, writing got much harder than before, and the 2000-words rule was in serious jeopardy.

4. WORKING OUT THE CONCEPTS

Here was the dilemma: We could not use any existing system as a model for introducing all the aspects of transaction processing for the reasons mentioned, and just accumulating all the different techniques for scheduling, concurrency control, recovery, etc would not magically produce a conceptual super-structure. It also became clear that our original idea of presenting all the techniques separately and then taking things to the conceptual level in a dedicated chapter was not an appropriate solution: For many techniques, especially those for logging, recovery, and commit processing, it is necessary to present them in the context of a specific architecture – otherwise many important dependencies cannot be explained. For example, when discussing logging, it is important to know who actually writes (flushes) the log and when, how many logs are there, who reads the log in which order, etc. Those issues are concerned with the interaction among different components of the system, so there has to be some idea of an architecture that the reader has to understand. Of course, one could make ad-hoc assumptions for each technique when it is being discussed; but if those assumptions are not consistent throughout the book, it will leave the reader confused, and it will not result in a deeper understanding of the architectural issues.

The discussions about the conceptual framework for the book could have gone on for a long time, had it not been for Jim's amazing talent of seeing the structure in an apparently hopeless mess of detail. I had the chance of watching this talent at work at different occasions, and each time it was fascinating with how little effort (or so it seemed) he would come up with an abstraction, with the essence of very complex technical problems. The answers often seemed to be over-simplifications, ignoring too many relevant parameters[4] – at least that was my reaction in some cases – but after having thought about it more carefully, one found that they emphasized exactly the right aspects and allowed for refinements in various directions if needed. There are people who love complexity as an intellectual challenge. Jim's attitude is different: He masters complexity alright, but then he likes to reduce it to the basic concepts and treat the rest as optimizations, special cases, or simply entropy.

The fundamental concept he suggested for the book (and for any good transaction processing system, for that matter) was the transactional RPC (TRPC). At the core, all components of the (abstract) system that we describe in the book contribute to the implementation of TRPCs. That means that all communication inside the system (including the operating system) is based on TRPCs. This was clearly inspired by the designs of Encina [7] and Tandem's TMF [6]. I remember the reaction of a number of colleagues whom we told what we were writing about and which approach we were taking: "But what about high-volume transfers, streams, and things like that?" We explained that we considered those as optimizations/special cases of the basic function, and Ch. 6 of the book discusses these issues.

[4] Take the five-minute rule [1] as an example. It ignores most technological parameters, the access patterns, etc and yet it represents a very important and stable way of looking at storage hierarchies.

A consequence of the TRPC-design is that each resource in the system, no matter who manages it, will be attached to the transaction for which it provides service. In a sense, a TRPCs acts like a spreading disease: If it invokes a new resource (service) that so far had nothing to do with that transaction, the resource just by being called will become part of (be infected by) that transaction. Fig. 5.4 in [1] presents a graphical metaphor of that idea.

Having a TRPC mechanism rooted in the operating system makes life quite a bit simpler for the transaction manager and for the resource managers. For example the "infection" mentioned above is implemented by having the TRPC mechanism automatically call the `Join` interface of the transaction manager when a resource manager gets involved into a transaction for the first time. So the implementers of resource managers can focus on their specific functionality rather than having to participate in general system housekeeping chores.

The interfaces of the transaction manager could be worked out quite easily once the basic decision had been made. True to the principle of just presenting the core concepts, we kept the functions for advanced transaction management to a minimum (`Leave, Resume`).

Another simplification resulting from the assumption of an underlying TRPC service was the callback model of interaction among the different components of the system. Rather than have various types of components (for recovery, for communication, etc) there are basically just resource managers, offering all they have to contribute to transaction execution by appropriate callback entries, and a transaction manager. In the foils we still had a separate recovery manager, but once we had thought through the TRPC-based design, it turned out that this was no longer necessary. By a simple division of labor between resource managers, transaction manager, and the log the orchestration of recovery is a simple generalization of what the transaction manager does anyway. Similarly, in our design, the communication manager is just a resource manager that is implementing the (transactional) resource "session".

Using all these abstractions we suggested the notion of a transactional operating system (TPOS) which manages the highly dynamic and powerful relationships among processes, address spaces, server classes, and transactions, i.e. the whole machinery required for TRPCs, including the TRPC mechanism itself.

Once the structure was clear, we specified the interfaces of all components that are presented in the book in C. Likewise, all the relevant control blocks are explicitly declared, and based on these all the important algorithms are spelled out as C programs – of course just in the form of code skeletons that emphasize the key aspects. This was as close as we got in providing the code of a complete TP system. Ch. 6 contains the central control blocks of the TPOS, and all the other control blocks used throughout the book are anchored there. But a lot more code would have been needed to turn this into a self-contained, executable system. In particular, we would have had to map it onto a specific OS, and we neither had the time nor the page space to do that – and the exercise would not have added much value to the book.

5. CONCLUSIONS

As I said in the beginning, we did not cover all the subjects that originally were on our list. In the foils we had chapters on SQL, on database design, on performance optimization, on benchmarking, on application design and a few more. We simply had to give up because we ran out of time and space – making it a two-volume book was never an option. On the other hand, we wrote extra chapters that we had not planned for: The feedback from students (in particular Betty Salzberg's) who helped us a lot with debugging early versions of the text, suggested that we put in a chapter on basic computer terms. We also added a survey of TP systems and a glossary.

In the end we were both quite happy that the writing tasks were completed, and that the design we had "invented" for pedagogical purposes had proven to work very well – if a coherent presentation can be considered as evidence for that. I got the impression that for Jim the whole exercise served as an "upload" of all the things he had learned and thought about for decades, so that other people could pick them up and he was free to take off to new territories. And indeed, the only paper on transaction management proper he wrote after the little black book was the one on Paxos commit [10]. It basically was a reply to the argument used by many people against transactions in a distributed environment: that the two-phase-commit protocol can run into a blocking situation.

In the meantime many arguments have been raised against transactional execution models, mostly along the lines of application complexity vs. the simplicity of ACID. Most of them are missing the point: Our book never advocates transactions as the only model for structuring applications. It rather makes the point that transactions are proper mechanisms for building reliable distributed systems – an issue that is as relevant as ever. It is interesting to see that in the context of web servers (a topic that is not even mentioned in the book) many techniques from the area of TP monitors and other system components have been rediscovered and re-implemented. But that is not a new thing: The history of TP systems up to the point that we summarized is characterized by a lack of communication, garbled terminology and re-invention of many basic concepts.

Anyhow, after we had resolved the question of how to conceptualize our presentation, we so much liked the idea of "transactions everywhere" (inside the system, that is) that we thought of putting the motto "There is no life outside transactions" at the beginning of Ch. 6, which introduces the TPOS. We decided against it, but many times, at the end of a day, when he had churned out his 2,000+ words, Jim would come to my desk and say "Hey, Reuter, stop. Let's go and see if there is any life outside transactions."

6. ACKNOWLEDGEMENTS

Having had the chance of writing this book with Jim was an once-in-a-lifetime experience which I have benefitted from immensely, both professionally and personally. Thanks a lot, Jim.

7. REFERENCES

[1] Gray, J. and Reuter, A. 1992, Transaction Processing – Concepts and Techniques, Morgan Kaufmann Publishers, San Mateo, CA.

[2] Patterson, D. A. and Hennessy, J. L. 1990, Computer Architecture: A Quantitative Approach, Morgan Kaufmann Publishers, San Mateo, CA.

[3] Tanenbaum, A. S. 1987, Operating Systems: Design and Implementation, Prentice Hall, Englewood Cliffs, NJ.

[4] Spector, A. Z. 1991, Open, Distributed Transaction Processing with Encina, International Workshop on High Performance Transaction Systems, Monterey, CA.

[5] Mohan, C., Haderle, D., Lindsay, B., Pirahesh, H., and Schwarz, P. 1992, ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging, ACM TODS 17(1).

[6] Tandem-TMF. 1991, Tandem's Transaction Monitoring Facility (TMF) – Introduction, 12014, Tandem Computers, Cupertino, CA.

[7] Transarc-Encina. 1991, Encina Transaction Processing System, TP Monitor, TP-00-D146, Transarc Corp. Pittsburgh, PA.

[8] Chrysanthis, P. K. and Ramamritham, K. 1990, ACTA: A Framework for Specifying and Reasoning About Transaction Structure and Behavior, ACM SIGMOD.

[9] Klein, J. 1991, Advanced Rule Driven Transaction Management, 36th IEEE Compcon.

[10] Gray, J. and Lamport, L. 2006, Consensus on Transaction Commit, ACM TODS, ACM New York, NY.