



Contents — FINANCIAL MIDDLEWARESPECTRA — November 1998

2.	Overhauling the Swiss payment system: using RTR with the Web Dr Gerhard Gucher, Systems Architect, EUROPAY (Switzerland) SA Beat Meier, Development Leader, EUROPAY (Switzerland) SA
12.	Message brokers: multiplying or merging? Charles Brett, President, C3B Consulting
20.	Responsive IT systems (RITS): the future or MIS revisited? Phil. Manchester, Consulting Editor, MIDDLEWARESPECTRA
26.	Advanta applies more than just middleware Jim Krzeszowski, Director, Software Development Advanta Business Services
34.	Patterns for using asynchronous messaging Jay Lang, Architect, PSW Technologies
40.	Java in context: from front end to database Dr. Keith Jones, Consultant, IBM Corporation
46.	Will middleware management improve soon? John E. Mann, Vice President, Research, MIDDLEWARESPECTRA

Overhauling the Swiss payment system: using RTR with the Web

Beat Meier and Gerhard Gucher
Development Leader and Systems Architect
EUROPAY (Switzerland) SA

Management introduction

EUROPAY is a member of the Zurich-based Telekurs Group. Telekurs is well known to the financial sector as a provider of payment services including:

- *Telekurs Financial (similar to Reuters or other market data suppliers)*
- *Payserv, which undertakes the actual processing for all Telekurs companies*
- *Telekurs SIC, which underpins the Swiss inter-bank clearing system*
- *EUROPAY, which provides payment and card systems.*

Of great interest to businesses is EUROPAY's initiative to enable business customers (initially, domestic customers will follow) to receive invoices and bills and pay these electronically. Not only does this remove a substantial overhead on invoice dispatch and subsequent debt collection for vendors, but the payment process is automated for the customer.

To achieve this Telekurs has designed and built a secure Web/transaction processing environment using Compaq's Reliable Transaction Router (RTR) middleware to connect customers, providers and the banking system (for payment clearing). As Beat Meier and Gerhard Gucher describe, the EUROPAY approach uses middleware to achieve the necessary scalability and security.

The cost and systems challenge

The goal of EUROPAY is to substitute cash with cards and electronic payment systems — as well as to replace at least some of the paper in the entire payment cycle. As such, EUROPAY's principal focus is the retail banking sector which, in Switzerland, still accepts most payments via the branch network. But this is a cost-intensive approach which Telekurs — owned by the Swiss banks — is being encouraged to address.

In Switzerland retail banking is costly for the banks. In practice they are less and less attracted to this side of the business and would like to introduce any alternatives which would reduce the cost of their retail banking operations. As Telekurs is owned by the banks, and already acts as an inter-bank organization, we are well positioned to introduce new initiatives. With our experience of running the Swiss clearing system, we have credibility and the capability to handle both large volumes of processing, as well as the security needed for customers (and vendors) to have confidence in the solution.

EUROPAY, although it is a member of Telekurs, is a relatively small company with about 200 people (the Telekurs Group as a whole has 1400 people in Switzerland, and 200 outside). Our customers vary, depending on which part of our business they deal with:

- **in the card business we have various types of customers, from end users (Master-Card) to card issuers, co-branders and merchants**
- **for the payment systems our customers are the Swiss banks, which communicate via EUROPAY to exchange financial data**
- **then there is the Automated Teller Machine network (we run the Swiss ATM network of cash dispensers).**

We are not, however, a bank. We do not hold money on behalf of depositors. We have no accounts.

In systems terms, as you would expect today, we have a mix of nearly everything. Sometimes it seems easier to list what we do not have rather than what we have. In any case, the ATMs are supported off Compaq's Tandem systems. Our DTA payment systems run on MVS systems, as does the

Swiss Interbank Clearing and our own financials. Newer systems run on Digital (now part of Compaq) UNIX. On the customer side we give out software toolkits to support connections to our systems: these run on Windows NT.

Our objective

Today, payment systems just support the payment of your bills. In practice, what we have are:

- **the customer**
- **a payment system**
- **communication with financial institutions (Figure 1.1).**

When you receive a bill, you have to extract the important information from the bill (the amount to be paid, the date due, etc.). Then you authorize some form of payment (check, cash deposit, etc.).

The problem we would like to remove from companies is the burden of having to gather that data from paper invoices and enter it into a computer system (the Accounts Payable system). For an individual the issue is similar, except that the logging is likely to be manual (as in a checkbook). Either of these have to happen before an invoice can be paid or an electronic transfer authorized.

For a business where the payment process is electronic, you can today submit what we call a DTA record (credit transfer) to Telekurs. We process this so that the money moves from the company account to the account of the organization which needs to be paid.

In a similar though slightly different way, an individual will receive a bill from (say) the electricity utility, go to the bank and draw out cash before proceeding to the Post Office to make the payment to the electricity utility via the giro system (as still happens frequently in Switzerland). On Saturdays you can still see large queues in front of Post Offices waiting to pay their bills with cash.

To us what seems so inefficient is that the original invoicing company already has the invoice in electronic format: this was needed to generate the invoice. At the same time, the recipient is faced with receiving many invoices, in different formats, from:

- **the electricity supplier**
- **the gas supplier**

- **the telephone service**
- **as well as all other suppliers.**

Each one has manually to be entered to some system (check book or computer) before any payment can be made. In Switzerland we even have other ways for making payments. You can, for example, just collect your deposit slips and send them to your bank to process. But then it is your bank which has to record the information before making the payments.

The electronic PayNet service

What we would like to do is to bring the vendors into the equation (via our PayNet system). We want vendors to transmit their invoices electronically to PayNet.

Once this is done, PayNet will aggregate the invoices by the organization (or person) being invoiced. In turn this will enable the customer to 'pick up' all invoices at once — whether into an Accounts Payable system (for a commercial or similar organization) or via direct access (for individuals). Each invoice could then be examined and scheduled for payment — all this being achieved electronically and without any of the traditional paper flows.

In effect PayNet will electronically forward invoices to customers if they participate. We transmit invoices to registered receiving systems or, if the recipient is a private individual, he or she can attach to PayNet via the Internet to obtain a list of all outstanding bills.

To make a payment, if done via the Internet, all the customer would do is click on a button to (say) 'I want to pay this bill on this date with this amount (which might be more or less than the amount outstanding)'. By using a digital signature — a requirement for our approach to work — the payment is authorized and verified.

Furthermore, you can choose how you pay. It might be as a transfer from your bank. Alternatively it might be a transfer from your postal system. However you choose, PayNet will check (in a manner similar to an ATM) that you have the necessary liquidity to make the payment (linking to the banking or other finance organizations).

From a financial viewpoint, we simplify with one more stage. Rather than passing every transaction

through the banking or other payment systems, we operate a netting approach. Only the net difference overall between all the accounts is exchanged — and then these net differences are processed via the Swiss Interbank Clearing system.

Thus, an electricity utility may bill customers by direct debit. The client is informed electronically (or on paper). For all the electricity invoices which have been collected by direct debit:

- **one credit is made to the utility**
- **all the details about the payment made flow to the utility so that it can automatically update reconcile its accounting systems**
- **one debit is applied to each paying customer account.**

If we can do this we will simultaneously reduce:

- **the effort required to make payments**
- **the entire paper chain involved through the payment process.**

Nothing new, almost ...

In one sense, there is nothing new in this. The broad approach has already worked for tens of years. The only new dimension is that the bill is transmitted electronically to the customer. Either via the Internet or via an in-house system, all invoices are received and are processed without the need for paper and repeated manual inputs.

After the payment authorization has occurred, we utilize the existing Swiss financial infrastructure. This is proven and known technology — not something new.

The implications are, we believe, significant. If everything was new the chances of swift acceptance would be much reduced. Because so much of what PayNet uses is already proven, we believe acceptance will be that much faster once we go live.

Key requirements — and centralized invoice distribution

One key requirement is that customers — individual or corporate — must be known to PayNet and be registered as a PayNet customer in order to be

able to receive electronic data. This is why we require digital verification.

If the customer is not known by PayNet, our approach is to print the invoice as would have been done by the original vendor and then mail this out. While we do not wish to become known as the 'bill printing company of Switzerland', this is what we may seem to be in the early days, as acceptance grows. Invoicing organizations will electronically pass us their invoicing details and for those who:

- are authorized to settle electronically, PayNet will send the electronic invoices
- are not electronically known, PayNet will print and mail invoices (acting almost as a central invoice distribution service).

Thus, over time, the number of invoices needing to be printed will reduce. Each new customer will 'remove' the need to print multiple invoices each month.

Our business plan is built, therefore, upon adoption of electronic invoicing and payments. But an equally important point is that the invoicing organizations will not need to reach their customers (at least for invoices). By sending all invoices into the PayNet system:

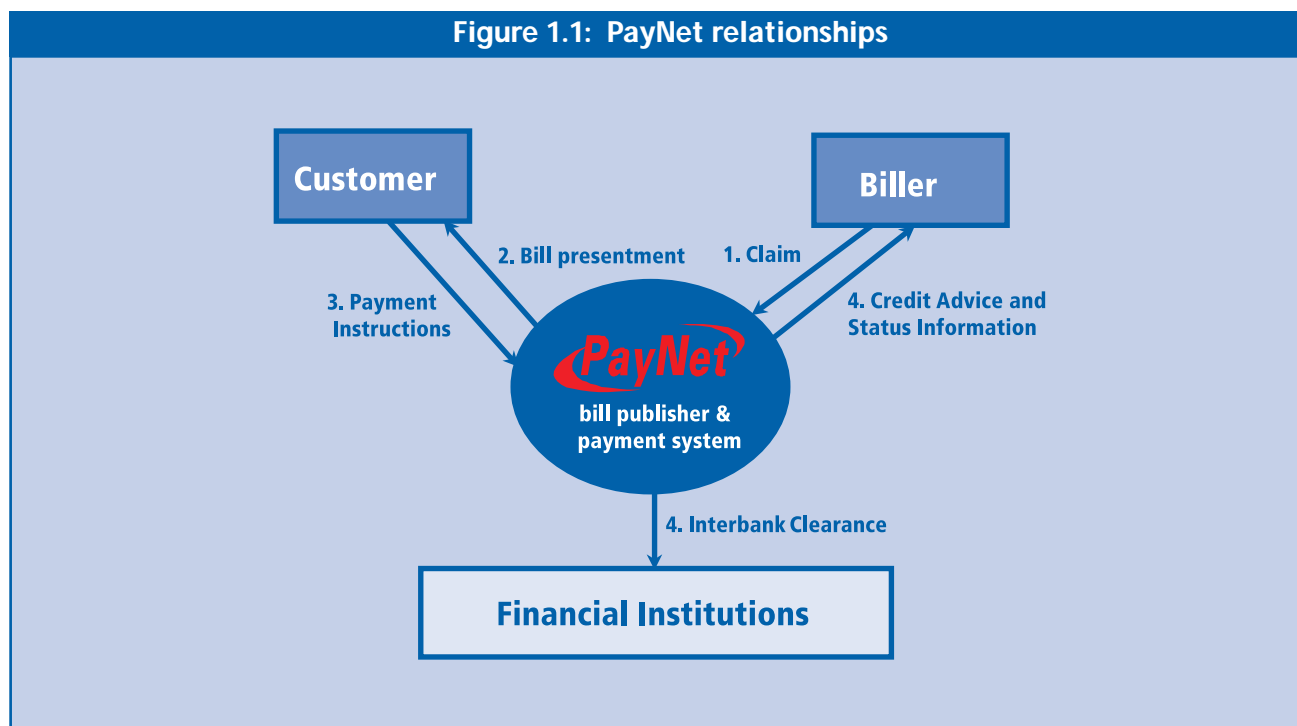
- printing, mailing and distribution costs are reduced
- existing payment methods can still apply
- electronic payments and receipts will be processable directly by Accounts Receivable (from the netted information gathered by PayNet) which, over time should reduce the staff needed for Accounts Receivables
- payments will be made faster, with less cost and time overhead.

Later we see other related services developing. These could include:

- payment collection (for those not paying)
- immediate payment options (like the credit card approach where the invoicing organization receives payment on despatch of each invoice and we then perform the collection of money from the customer).

Both of these services would attract a percentage transaction fee. In contrast, in the initial PayNet implementation, transaction costs are not dependent on amount; the fee is not a percentage of the payment.

Figure 1.1: PayNet relationships



The systems and middleware challenges

The principle challenge — underlying everything — is reliability. PayNet, at least as we have described it above, must work 24 hours a day. Because customers can be companies, organizations or private individuals, we have to assume someone will want to pay invoices on Sunday evenings or Christmas morning or whatever. Even for companies, facilities will need to be available all day. We have, therefore, planned for 24 hours, 360 days a year availability. This is not quite 100% availability, but it is close.

There is a second challenge, although it will only become a major issue as we succeed. This is the potential volume. Our technology target is to be able to process about 120 million bills per year, or around 350,000 a day (on a 360 day basis) or 600,000 a day (on a 200 working day basis).

Put another way, this amounts to 350,000 or 600,000 business transactions a day — but many more ‘IT transactions’ once we include all the invoice creation, distribution, payment instructions, debits, credits, etc. In database terms we think the real figure will rise to more than 1B ‘IT transactions’ a year — which is why scaleable transaction processing is so important to us.

Initially, however, volume will not be a major problem. We expect only those organizations that will obtain the greatest benefit to sign up early — because they will obtain the most financial benefit. These are likely to be commercial organizations with business to business payments.

In our estimation, private customers will take longer to join in — not least because Swiss private customers are known to be conservative in selecting their payment systems. For some time there is still likely to be a large percentage who go to the bank, withdraw the cash and then walk to the Post Office to pay their bills there.

If we are successful we will have the problem of massive volumes. Our design has to be able to scale the throughput as customers and invoicing organizations participate. Equally, we will not invest in the largest systems from Day 1: that would not make financial or business sense.

Instead we plan to grow with the pace of activity. We think it will take at least three years. Even then we must prepare in case we ‘overshoot’ and become even more successful than 500M-600M

‘IT transactions per year’. This is the system and middleware challenge.

Choosing a transaction solution

In PayNet we have been running on Digital (now Compaq) UNIX for some time for existing applications. We have, for example, other systems which communicate with the banks which check liquidity — a system called EVA. The new system is a derivation from this.

EVA, however, does not have a scaling challenge. It also has much less of an availability problem since it communicates with the banks — and banks work working hours. They do not work Sundays. So you always have ‘unavailable’ time to administer or maintain the system. Furthermore, EVA is largely a batch processing system. If you have half an hour delay, it is not a customer disaster. In contrast, PayNet is an on-line system. Customers will be able — and expect — to connect through the Internet directly to PayNet. They want immediate responses.

We knew, therefore, that Digital UNIX was going to be our operating system basis. But we realized that we needed some form of transaction system which would allow our software systems to reach the necessary availability and scalability. We looked at Tuxedo and two or three alternatives, but none of the obvious ones met the scalability and reliability combination we felt we must have.

We also considered DCE. Indeed, EVA itself uses DCE as middleware. It is a distributed system. Using DCE in PayNet seemed to make sense initially, as we also planned to use it to connect to banks. For the first phase, however, existing communication paths are used — file transfer, X.400 and FTP — rather than DCE.

Equally, the banks were not sure (at our decision point) whether CORBA was really relevant. To us, if DCE was out of the loop, CORBA was also a ‘non-starter’ — it does not have sufficient security for what we need. Our approach demands an ‘inter-company interface’ and it is not possible to do that today with CORBA. So we suspended any further consideration of that approach.

This brought us to RTR sitting in front of Oracle databases. Our analysis indicated that we could solve our inter-process communication and distribution requirements with RTR — with the added

advantage that we would not need too many different software products, with all their different unpredictable interactions in our system. Add the failover, recovery and load balancing inherent in RTR together with the Alpha processor machines, and we thought we had found the middleware which could provide us with the scaleable and reliable transaction system we needed.

Physical and software architecture

The key concept with RTR on UNIX and Alpha is that we can just plug in additional processing nodes as volumes dictate. Indeed we do not even have to have top of the range Alphas. Modest to mid-range will suffice to start — with RTR being responsible for spreading the transaction loads.

For example, for our physical starter production system (Figure 1.2) we will have:

- two 4100s, with 4 CPUs each, as the RTR/database servers
- two front end servers (1200 systems, with two CPUs each) for handling security and access
- all four systems connected via 100MB Ethernet links.

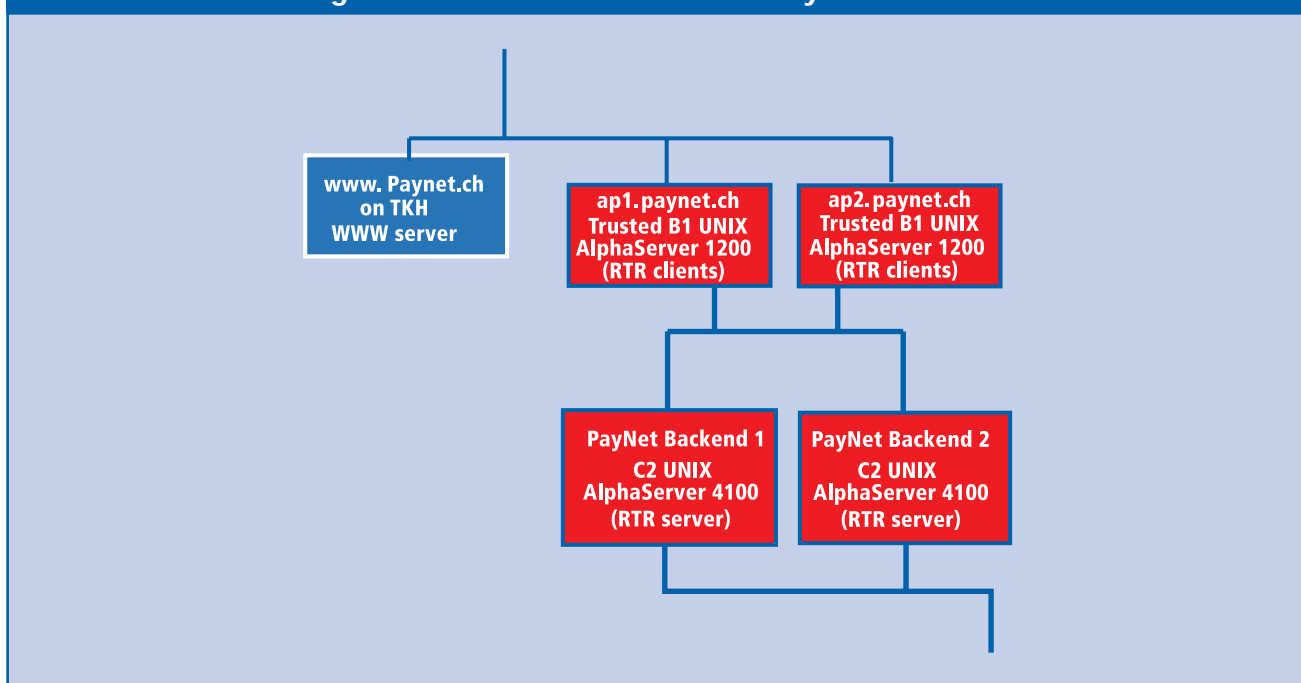
The two 4100's both mirror each other as well as spread the processing load. They also do each oth-

er's work as well so that we have a minimum of database reorganizing to do once we will scale up.

A simpler way to comprehend what we are implementing is to consider Figure 1.3. In this there are four networks:

- at the top there is the Internet; this is where customers access PayNet, through the firewall (the first level of our security)
- behind the firewall is a second network on which our Web server is located — and where the primary security servers (the 1200s, with software rated to Orange Book B1 standard) are; via a hyperlink (from the Web server) your Internet session is handed to either of the 1200s for authorization and identification (balancing the loads) and, if successful, a connection is made to an RTR client
- between the 1200s and 4100s (the RTR/database servers) there is a dedicated 100MB Ethernet network for the RTR clients to talk to the RTR servers; using RTR's middleware features, this enables us to add additional servers as demand requires as well as provide the load balancing, automated shadowing, failover, recovery and transaction processing

Figure 1.2: Core RTR linked to the PayNet Web server



- below the 4100s is the fourth network (also 100MB Ethernet) which provides the connections from the RTR communication clients out to the Telekurs production systems as well as to X.400 servers, gateways to external systems (for example banks), etc.

Initially, all our servers are located in one data center. However, as volumes build, it is planned to exploit RTR's distribution and recovery capabilities across a high speed WAN to add a second data center. This will not only shadow the primary center but will allow us to distribute the processing across both locations without losing the mirroring.

All this runs on Digital UNIX. In theory it could all run on one UNIX box. Our requirements for availability, and scalability, however, prevent this. What RTR offers is the transactional integrity combined with scalability and operational flexibility to achieve the above — without having to build such capabilities ourselves in software.

RTR for us is, therefore, middleware which can tie together several different functions in one product:

- **scalability** — the ability to add processors as needed

- shadowing/mirroring (and even clustering, if we ever need it) for resilience
- recovery, for transaction integrity
- transaction processing, for data integrity
- standardized entry (and exit) points for users — via the RTR clients.

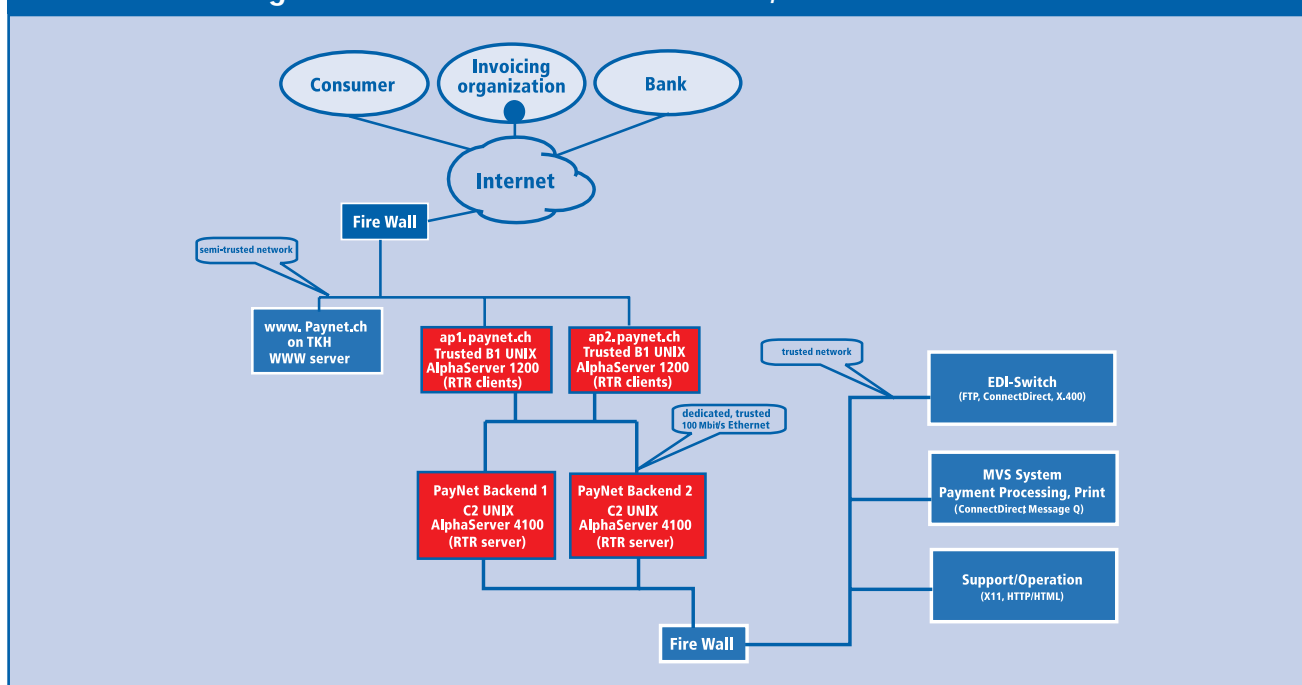
From development to production

We have been working on this PayNet solution for some two years now. In practice it has been an evolutionary development. Initially we had a massive specification — one of those that if you try to do everything at once means that you never complete as improvements and changes always need to be incorporated.

One of the benefits of our architecture is that it has enabled us to evolve with the solution. We did not undertake one huge serial process. Instead we chose to extract a minimal first functionality to build — which only accepted bulk submission of invoices from vendors into the system and then printed the individual invoices for distribution.

This was finished over a year ago. We decided, however, not to make it a product until we had added more. As we said earlier, we do not want to become known as the largest invoice printer in Switzerland. We wanted to move beyond this to the real objective — the electronic processing.

Figure 1.3: From front end to back end, with RTR in the middle



Nevertheless, we could receive invoices, print them and deliver them.

Instead we decided that the second step should include communication with the banks as well as direct access for the invoicing organizations. This we finished at the beginning of this year and we are now using it as a pilot system. We have our first customers and banks connected so that they can test their systems against ours. It is running well.

Now we are working on the customer payments dimension. Here we need to include digital signatures to address authorization, authentication and non-repudiation. Currently the invoices we receive from vendors are not signed. By the end of this year they will be electronically signed. Similarly, payments which customers execute will have to be electronically signed.

To deliver has meant working with Swisskey, a new company (founded by the Swiss Trade Commission, Swisscom and Telekurs) to issue digital keys in Switzerland. Customers — whether invoicing organizations, paying organizations or individuals — must first obtain a digital key before they will be able to use our services.

Indeed, another positive aspect of evolutionary development emerges here. When we started we knew that we would need digital signatures. But we understood that we did not know how to do this, not least because there was no issuing authority. Essentially we decided that this would come (eventually) — and it has, just when we needed it.

In language terms we use C++. We thought of using Java but it was just too new and unproven for a financial system like ours. In addition, we wanted to design the whole application suite using objects oriented methods and the links to RTR were already good for this (see Figure 1.4).

We chose C++ because we wanted a CASE tool. The important point to us is that our design, as it stands today, remains compatible with our code. We can regenerate the entire code based on our design. At the time, for code generation, there was no other language than C++ and we use Rational Rose as the tool and generator.

There was one area where our abortive experiences with DCE did actually pay off. After DCE, RTR seems simple. If you know what a database does and you have the necessary understanding of

transactions, messages and queues, then it is quick to pick up RTR.

What is more, RTR does an enormous amount for you without needing large amounts of application programmer coding or involvement. This has proven extremely beneficial for our developer productivity, especially compared to earlier projects where the middleware was so complicated to use (and administer) that we had more problems in making the middleware function than with the intended application. By comparison, RTR has no such problems: it is consistent, tested and stable.

A third aspect which has pleased us is that only two or three people in our group need to have RTR knowledge. By using an object oriented approach, we can map objects to RTR which hides the sophistication of RTR, at least as far as the application developer is concerned.

Transaction security and the Web

Anything which has dealings with the Internet and the Web in the financial environment requires particular attention. As with a bank, credibility is all important. If you lose that credibility your customers will not use your service.

So far what we have explained is how customers come in through the Internet. This is not the only way. For example, invoicing organizations — with large files to be processed — can:

- **either access PayNet via HTTP — we have a mechanism to transfer files using HTTP (our preferred way)**
- **or they can use our X.400 gateway with batch support for the outside world. A RTR client reads the data out of the X.400 MTA (DEC Mail Bus) and stores them via RTR in our shipment database.**

Once we had worked out how to handle the transaction volumes reliably and with a scalable architecture, a major part of our focus concentrated on security. We did not regard a firewall as being anything like sufficient. Adding digital signatures — as described above — was part of the solution.

However, we decided we needed to go further and place an additional security layer between the Web server and the RTR clients. To ensure security, our front ends (the two 1200 servers) use specific soft-

ware from a local company called Ergon Informatik. Ergon has developed:

- **an HTTP daemon, which fits with B1 UNIX**
- **specific HTML templates (these templates are what interact with the RTR clients which in turn pass information to, and obtain responses from, the back end databases).**

What is critical is that this code on the front end servers is trusted. It does not use anything like CGI or other scripts. Instead, the users 'fill' in the trusted templates which are then used to pass information to the RTR clients. In effect we have a private connection to the back end.

In practice we have one socket to the security manager which recognizes the customer using a cookie and the digital signatures. Based on this cookie, we know which is the correct client session for each particular customer.

Although this may seem an extra layer, we believe it is worth it because our experience is that such an approach produces an excellent system which performs well and is secure. The Credit Suisse bank is another user which further increases our confidence.

Lessons learned

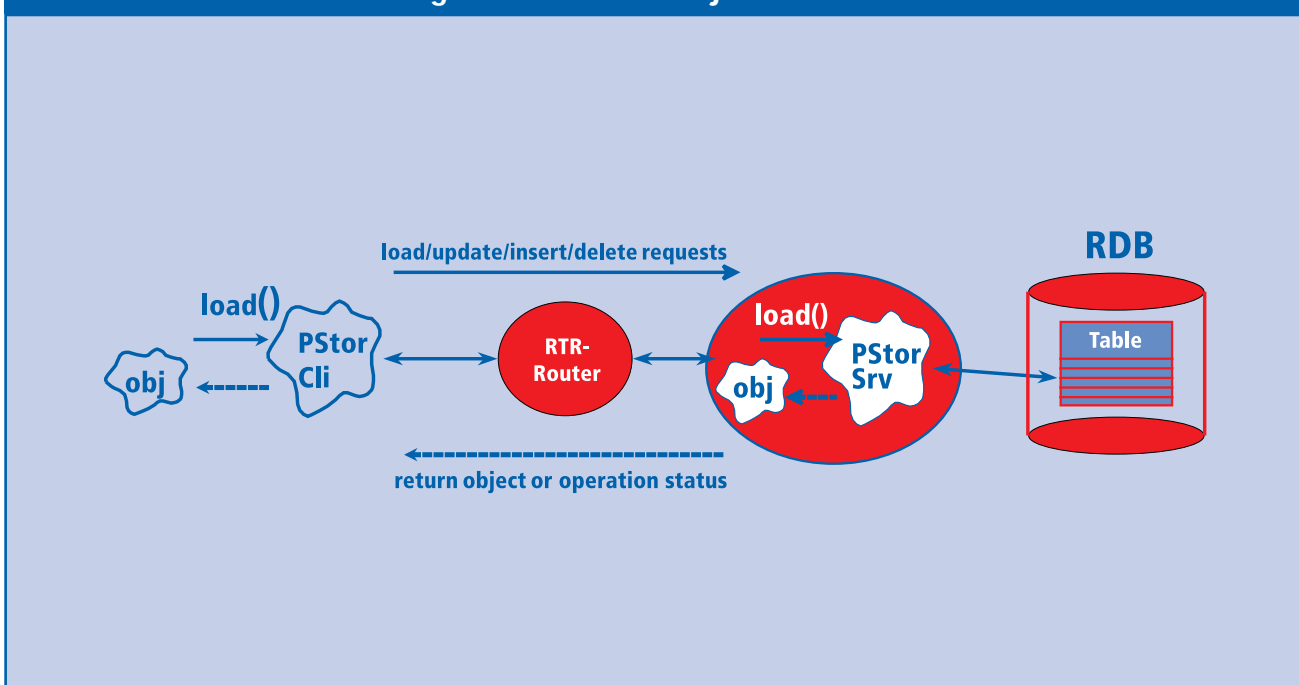
The first is, keep middleware simple. From earlier projects — where we used much more middleware — we learned that it is generally good to reduce the number of middleware products. This is what we have done.

Indeed, we can go further. At the beginning, in the first designs, we thought to use many technologies and investments from other earlier projects. As time went on and our plans achieved greater maturity we found ourselves constantly rejecting middleware products — until we were left with RTR and the Web-specific technology.

DCE and DEC MessageQ (now BEA MessageQ), for example, will vanish as we concentrate on our selected middleware technologies. The rationale for doing this is simple. It is to keep the design and implementation simple. Adding people to the project, or to related projects, takes less effort. Finally, managing the production system is that much easier with coherent middleware at the core.

We have also learned, as has our management, that earlier project management methods — like the waterfall methods — are outmoded. Decoupled development, breaking the challenges into separate pieces which can separately be produced, is both more efficient and carries less risk. We have proved that a large project like PayNet can be developed in

Figure 1.4: RTR and object orientation



a given time frame using an evolutionary approach. Developing PayNet by a conventional waterfall model would have started with an analysis lasting a year, followed by a design of three quarters of a year and then an implementation of maybe another three quarters of a year. This is too much.

Instead it is much better to break the entire project into smaller projects which in themselves made sense. We can then develop these to the stage where they can go into production — before proceeding to the next step. This gives us the opportunity to correct mistakes as well as the opportunity to test thoroughly our key technologies over an extended period of time. In doing so you reduce the project risk as a whole.

Developing this way means that you have something running early which you can use for production. As we discovered, if redesigns of certain aspects are necessary (and some always seem to be inevitable), this is possible.

One particular thing we learned time and again was that it is almost impossible to design a complex system right the first time — especially where middleware is concerned. Middleware adds to the complexity (it is, after all, solving a highly complex set of problems). We implicitly knew this. We proved it to ourselves. We also proved that you do not have to allow yourself be trapped by such complexity.

Management conclusion

EUROPAY is in the electronic payments business. This covers a range of possibilities — from bill presentation to SET for credit cards and micro-payment systems. The PayNet example discussed here is for mass payments of invoices. For settlement, PayNet is based on standard inter bank transactions (as used by S.W.I.F.T., Swiss Interbank Clearing and others) where the electronic systems already exist.

The PayNet solution described initially is for businesses, and later for individuals. Its purpose is to replace paper invoices where possible and enable businesses (and individuals) to pay their bills without having to receive many different forms of paper. Delivering this requires:

- ***a universal access method (the Internet/Web combination)***
- ***production strength security***
- ***digital signatures***
- ***transaction processing***
- ***connections with existing clearing and settlement systems.***

The first of these was available. The last of these existed, through the Swiss InterBank Clearing system (run by Telekurs for its owners, the banks). Robust middleware was needed to satisfy the other three requirements. With a specific architecture implemented around RTR, PayNet demonstrates how innovative financial services can be married to the Web without losing integrity and without the middleware needing to become too complex.