SOA Case Study: Agility in Practice

As boundaries within and between enterprises become increasingly permeable, there is a greater need for information flow. This is inhibited by the "information silos" formed by traditional software applications. Service oriented architecture (SOA) replaces these silos with loosely-coupled services, enabling information to flow as needed, and delivering enterprise agility.

This is a case study from ING Card, a division of the ING Group, member of the Jericho Forum of The Open Group. It describes the first phase of their SOA implementation, with services that are hard-wired rather than dynamically discoverable. It illustrates how even this stage of SOA can deliver real business agility, and contains some interesting lessons for SOA implementation.

The case study was written by Alcedo Coenen. Alcedo has built his experience in IT since 1987, although he originally graduated in musicology in 1986. He has been working as programmer, information analyst, and since 1997 as (information) architect for ING and other companies in the Netherlands. Within ING Alcedo has been working on multi-channel architecture, a global SOA for ING Europe, a credit card system and on knowledge systems. Recently he has established a working group on the Business Rules Approach, producing articles and presentations for several architecture conferences and meetings.

SOA Agility in Practice

Service orientation within one application[1]

Alcedo Coenen
alcedo.coenen@gmail.com

ING Card[2] built an application for its customer base that enables it to link to new websites, implement new product features, and maintain credit scoring rules, easily and quickly. It achieved this by applying three construction principles, one of which was service orientation. This article describes the application, and explains how SOA helped to meet the business needs of a credit card issuer with particular challenges.

Challenges

A credit card business combines two financial services: payment and consumer credit. Payment by plastic is accepted worldwide thanks to a large payment network, mainly maintained by credit card brands such as Mastercard and VISA. The card issuer usually also provides a revolving credit facility that allows delayed repayment of the balance. The complete product is often called *revolving credit card* in order to stress this.

ING Card was founded as part of the ING Group[3] in 2002, with the objective of centralizing ING's credit card business and extending it to the European market. ING Card became a significant player in the role of *issuer* in the complex credit card value chain (see Figure 1). ING Card uses two *credit card brands,* Mastercard and VISA, that own worldwide payment networks and provide millions of merchants (shops, hotels, restaurants, car renters etc.) with card reader equipment. The reader contracts are usually handled by *acquirers*: institutions (usually banks) that facilitate the settlement between the merchants and the credit card brands. Another link in the chain is formed by the *processors*: organizations that execute the actual transactions, provide the card authorization at the time of sale, and carry out the settlement between acquirers, credit card brands and issuers. Some institutions combine the role of acquirer and processor.

---

[1] This article was originally published in Dutch. A shorter version was published as "Wendbaarheid dankzij SOA" in *Informatie* november 2005, nr 47/9, page 64-69. The extended version was published in Charles M. Hendriks & J. Arno Oosterhaven (red.), *Architectuur in ontwikkeling. Landelijk Architectuur Congres 2005*, Academic Service Den Haag, 2006, ISBN 9012113199, page 93-105. Translation by the author, assisted by Chris Harding.

[2] See www.ingcard.nl, www.ingcard.be and www.ingcard.de .
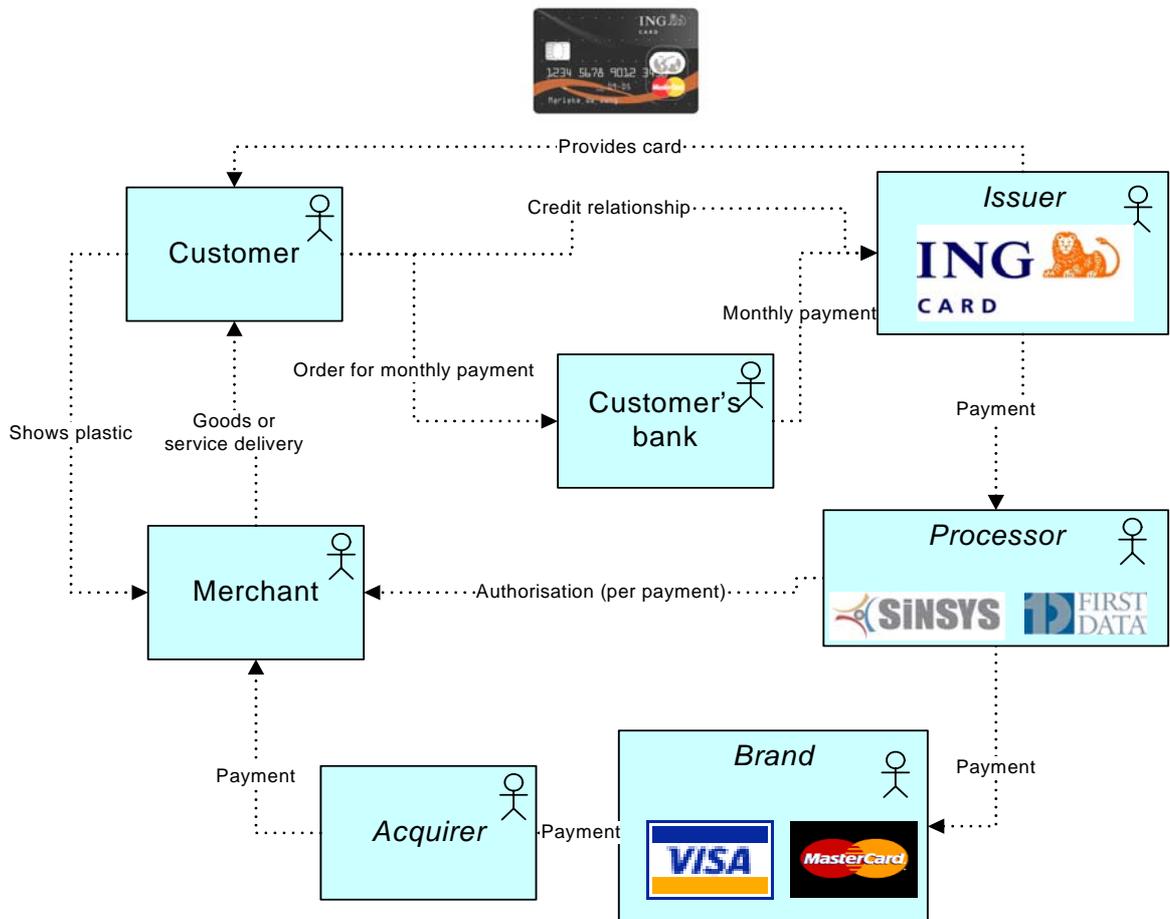
[3] www.ing.com

Figure 1. Roles in the credit card world

From the start, ING Card had some particular challenges that had a major influence on the choice of IT solution.

The first challenge was to handle customers who did not already have a current account with ING. Previously, credit cards were only provided to existing customers with an ING-branded current account, which was used for the monthly payment. Now, new customers could have a current account at any Dutch bank. All customer systems within ING in the Netherlands were identified and linked to an ING customer account; but these systems could not now be used for ING Card. A complete new customer management system was needed.

Secondly, sale would be concentrated on direct channels in the first place (Internet, call center and direct mail), with online account facilities that would provide the customer with information about all his or her transactions on a daily basis. Moreover the customer would be able to change the amount of the monthly payment (with restrictions), and ask for a credit-limit raise, through direct channels also. This marketing strategy meant that:

- A website was needed for online account management by the customer;

- The call center must be able to enter card applications and customer questions;

- All data entry facilities must support all channels, in order to obtain the same information through all channels; and

- The customer management processes should be linked to the card *processor*, which takes care of the distribution of the card, of the actual account, and of all transactions.

The third challenge was related to one of the core competencies of any issuer: credit scoring, which is needed to manage risk. The system must evaluate all card applications using creditworthiness and fraud indicators.

The fourth challenge for IT was to be prepared for some future developments that were clear from the beginning. ING card planned to roll out in multiple countries and there would be different card products. The system must be flexible enough to be rolled out in different countries, with possible changes in processes, products and credit scoring.

The fifth and last challenge for IT arose because ING Card became commercially responsible for all existing credit card portfolios of ING in the Netherlands (the brands being Postbank and ING Bank), with more than 1 million customers. This meant that the new system must fit in with existing processes and organizations. It soon became clear that the credit scoring should be performed centrally and made available to all other card systems within ING, so that there would be a single source for the credit business rules.

In summary, the following requirements had to be met:

- Functional:
    - Customer administration
    - Online facilities for the customer
    - Multi-channel support
    - Credit scoring
    - Support for multiple countries
    - Support for multiple products
- System features:
    - Agility to different processes
    - Agility to changes in product features
    - Coupling to other processes, based on the same business logic.

The solution

A decision was made at an early stage to build the system from scratch. A further decision was that the system should be built on Websphere and Oracle. The solution was given the name "New Card System" (NCS).

Agility was achieved by using three construction principles:

1. A layered architecture, known as the n-tier architecture.
2. The service concept.
3. Parameterization of some of the business classes.

The traditional three tiers (presentation, business logic, data) were enriched by a *service* layer and a *business model* layer, both within the business logic layer, as the means of introducing the service concept. Moreover the data layer was enriched by the separation of a data access layer from the data store layer. The layers are depicted in Figure 2 together with the techniques used for their implementation.
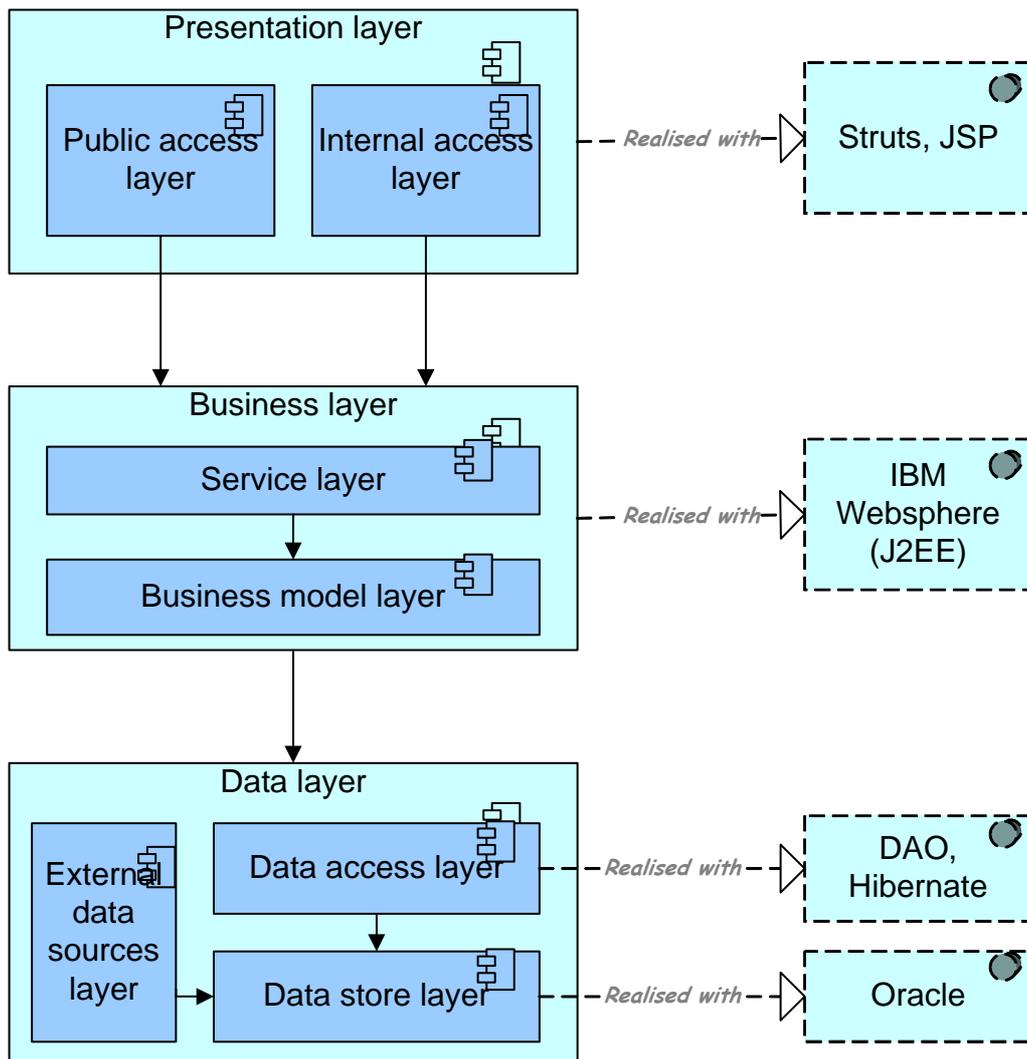
Figure 2 – The layered architecture of NCS

The presentation layer forms the exterior of the system and consists of a series of screens, supporting the main processes. NCS currently has the following instances of the presentation layer:

- Several public websites (www.ingcard.nl and www.ingcard.de) which provide the screens for applying for a new card. These screens call services from the service layer that implement the application process.
- A closed and secure web environment that is accessible by the customer. Here the customer can log in, review transactions and monthly statements, and manage his or her account. It provides facilities for requesting a credit-limit raise, changing the amount of the monthly payment, requesting a secondary card, and entering personal changes such as to the customer's address.
- A data entry system for the call center with similar functions to those that the customer has, and with additional functions such as viewing all customer data and history.
- A data entry system for the back office, supporting its work of dealing with all card applications.

In accordance with the layered architecture principle, the business layer is only accessible through the presentation layer. This enables the screen designs to be independent of the business logic.

The business model layer is in turn accessible only through the service layer. Services are functions that are meaningful to the business. Services do not execute these functions themselves, they are just a shell or facade, behind which the real function is executed by the business model layer.

An example of a service is *findCreditRegistrationPerson*, which takes care of finding the right person at the credit bureau's system (BKR is the Dutch national credit bureau). See Table 1 for its full definition.

Name	findCreditRegistrationPerson
Service Group	NewAccountController
Release	2.0
Usage	Finds persons in the BKR system
Input	creditRegistrationPerson
Output	Collection of creditRegistrationPersons
Precondition	The creditRegistrationPerson must have agreed on checking BKR
Postcondition	NA
Actions	1. determine the necessary data 2. sends the data to BKR 3. receives the results from BKR 4. return the information
Business classes involved	creditRegistrationPerson

Table 1- example of a service definition

The presentation layer could say to the business layer, "Find this person at BKR, here are the data that you need for it". The presentation layer can then rely on the execution of this request and on obtaining an answer. This design has three advantages:

1. The presentation layer does not need to be aware of technical considerations (for example, it does not need to know that conversion of some data is needed before it can be used).
2. The sequence in which services are called can be changed easily without any change in the business logic layer.
3. The service call does not need to be changed when its implementation is changed, for example, *findCreditRegistrationPerson* need not be changed when the implementation of the *CreditRegistrationPerson* class is changed.

The independence of presentation from business logic is improved, compared to the traditional 3-tier architecture. The order of tasks can be changed, or another website can be added, for example with a limited selection of services just to show the customer information.

There is no need to modify the business logic layer for such changes. When ING Card is launched in another country, the *findCreditRegistrationPerson* service will have another technical implementation for another credit bureau. The dialogue in the presentation layer does not need to be changed for that, because the name and call of the service remain the same.

The third construction principle of class parameterization was applied within the business model layer. This layer contains the real business logic, implemented as operators on classes that represent the Business Object Model (BOM). The BOM determines the "language" and consists of a set of classes with their interrelationships, implemented in the form of java classes. In Figure 3 the main classes and their interrelationships are shown. Some of the classes are parameterized, which makes them more powerful.
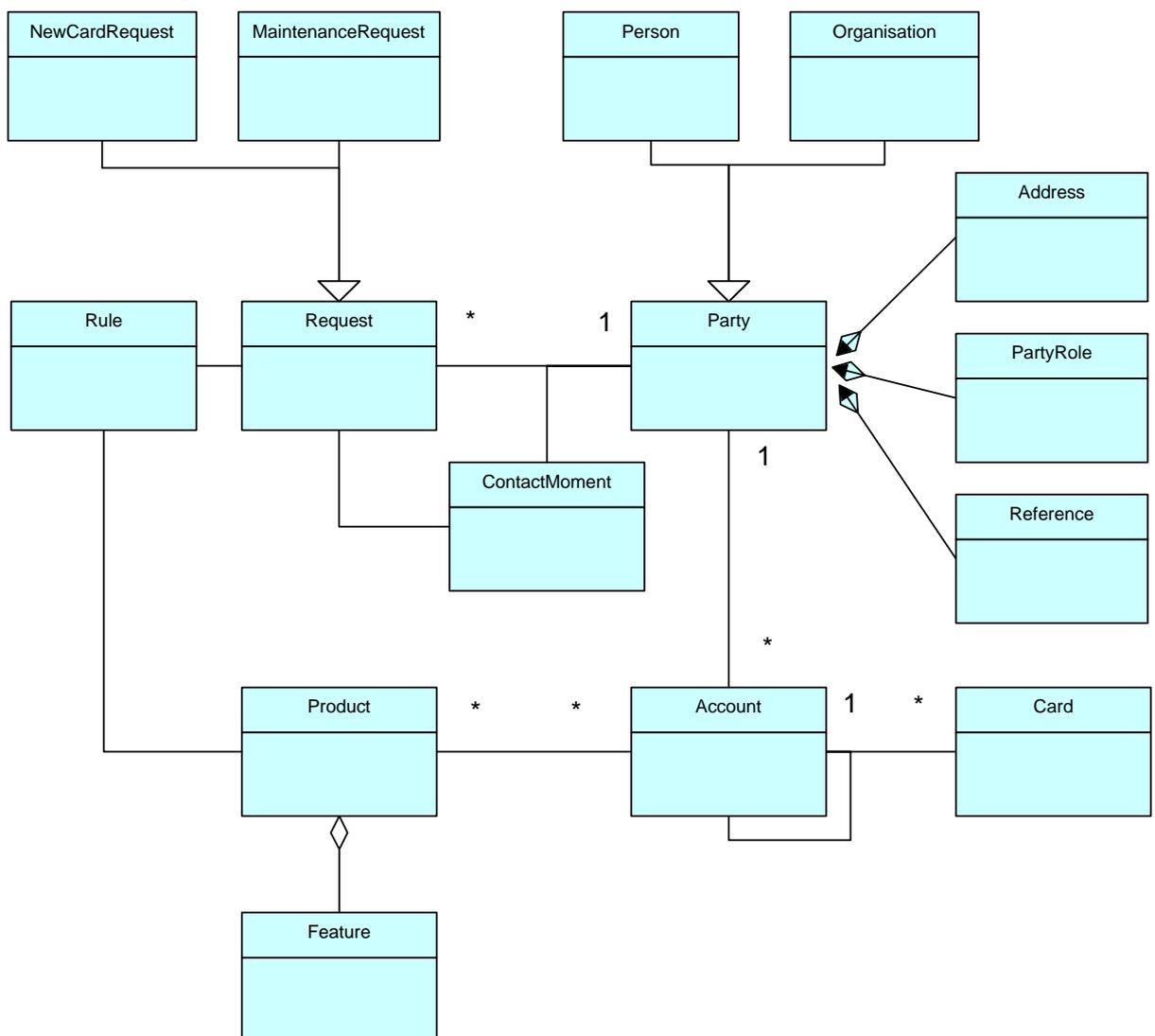

Figure 3 – Extract from the *Business Object Model* (UML class notation)

While the classes Account, Party and Card are straightforward. (they just represent account, customer and card), the classes Request, Rule and Product have a much more parameterized structure to give the flexibility that is needed. When a variation of a product is launched in a marketing campaign, only the configuration of the Product class need be changed, not the

structure of the class itself. That is made possible by defining a generic attribute called Feature, which can contain any possible feature for a product, which makes the Product class configurable as required. A feature such as "one year for free" can be added easily without having to reprogram; the actual implementation of such a feature is not implemented in NCS but is executed by the card processor; NCS only needs to forward the right features to the processor. The Request and Rule classes are defined in a similar way. For the Rule class, there is a separate rule engine that enables flexible configuration of credit rules.

In short, the business layer provides services to the presentation layer and takes care of transactions in the business model layer in a way that reflects the language of the credit card administration. The business model layer is set up with simple and parameterized classes; the parameterized classes enable changes in product features, credit scoring rules and request items without impact on the overall system.

The result

NCS has become the system that takes care of the management of customers, as it was meant to be. Moreover NCS has become the system that executes all services that customers are offered through the website, such as making address changes or requesting a credit limit raise. Finally NCS has also become the system that does the assessment of card applications and offers the opportunity to manage all kinds of credit and fraud risks.

The three construction principles of layered architecture, service orientation and class parameterization have each contributed to the meeting the requirements, in particular the requirements for agility (see Table 2).

realized in *requirements*	Layered architecture	Services concept	Class parameterization
Online facilities	▓		
Multi-channel	▓		
Credit scoring			▓
Multi-product			▓
Multi-country		▓	
Agility of processes	▓	▓	
Agility of product definitions		▓	▓
Possibility to link to other processes		▓	

Table 2 – which requirements are satisfied by which principle

The layered architecture is needed to deliver multi-channel support. If the same information, and partly the same functionality, is to be presented to the customer via Internet, call center and the back office employee, then a separation between presentation and business layer is essential to avoid duplicate implementation of functions.

Class parameterization is needed to deliver the required time-to-market for the introduction of new products, and for adapting credit scoring and fraud detection rules.

The service concept turns out to be the most important of all three principles. Offering this facade of business logic makes it possible to roll out NCS relatively easily in other countries, enables optimization of processes without having to change the system, and facilitates linking with other processes or other systems.

The best example of the power of the service layer is in international roll out. It is now relatively easy for ING Card to use NCS in any other country, even when there are some major differences. Checking the national credit bureau, for example, needs a different implementation in each country. The work of building the interfaces remains, but the service that is called to execute the check does not need any change. Another example is the ability of a call center to use its own preferred system that supports more than just the credit card product; this preferred system only needs to call the appropriate NCS services when a call center agent is handling a credit card request, using the screens and dialogues of its own system; there is no need for a deep integration between the systems. There are many situations where the independence of the service layer shows its value, some of which are illustrated in Figure 4.
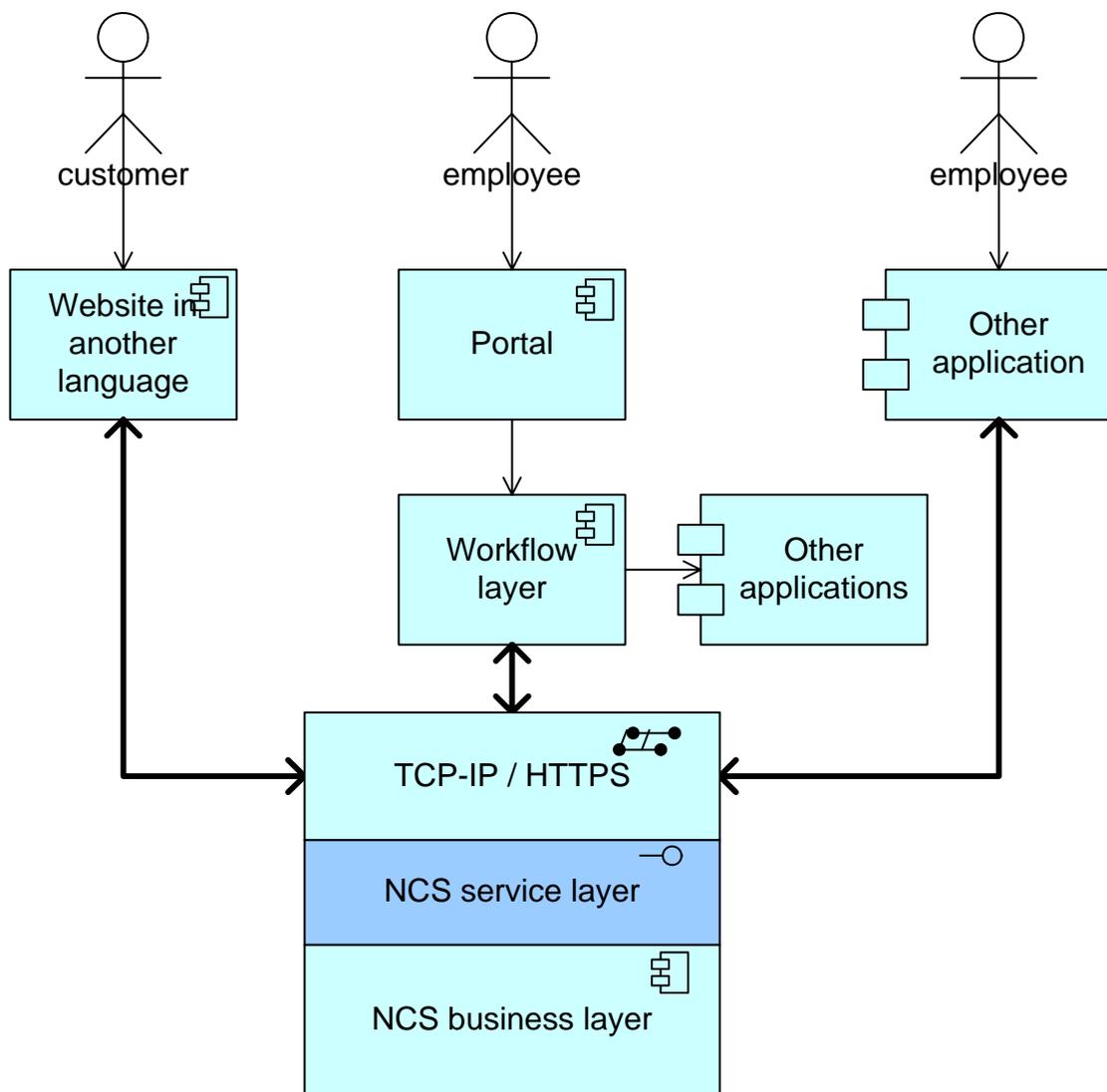

Figure 4 – Several possibilities for the use of services

Not all aspects of SOA have been realized in NCS. The NCS layers communicate via J2EE objects, not messages. Therefore there is no message bus and the web protocols SOAP, WSDL and UDDI are not used. The services have been published in a Word document, and management and definition of services are manual operations. In this respect the solution

appears little different from a traditional one based on APIs. There is a significant difference, however. Services are empty, they do nothing themselves, except calling the appropriate business functions in the right order. This makes services much more independent than the functions of an API design.

The SOA implementation of NCS was designed to be realized in phases. In the first phase, NCS is a simple system with a limited number of presentation layers; that is why the services need not be published and can be invoked without using standardized protocols. But as soon as NCS services need to be called from other applications, it is necessary to invoke them in a standardized way and provide standardized mechanisms for publishing and finding them. NCS has the potential to participate in a network of multiple systems, which contributes to the agility needed by the business. NCS is a form of agile SOA.

Lessons learned

Although many architectural principles were settled before development started on NCS, many issues requiring new principles emerged during the development process. Some interesting lessons were learned.

- Application management must be prepared for a new role of guardian of the service catalogue. SOA needs active management of the collection of services (the so-called catalogue). When a project implements new functionality, the danger of duplication of services must be avoided. Application management must be prepared for this new task.
- All parties must be familiar with the service concept; and this includes business stakeholders as well as application managers and software developers. When one of these parties is forgotten there is a major chance for misunderstandings and bad decisions. For instance, when a business stakeholder does not understand that a service can be used by an additional system, the service could be rebuilt in that system. When a traditionally-educated designer only knows about object orientation, there is a danger that he will make the service too tightly-coupled to the actual function.
- The granularity issue (how much functionality does a service cover?) should be resolved during the design, not beforehand. Discussions about granularity can become unmanageable and meaningless if not related to the actual analysis of functions.
- It should be a rule that the business model layer can only be accessed via the service layer and not directly from the presentation layer. It is tempting to call business class operators directly from the presentation layer, especially when the system is still isolated and not yet connected to other applications or presentation layers; the service layer seems redundant, and the project may have strict deadlines that impel the programmer to take shortcut solutions. Such shortcuts, however, have a disastrous effect on the time to market of future developments.
- All workflow-related functions should be kept out of the service layer. When the order of a series of functions depends on external knowledge that is not available in the system, it is advisable to put its control into a separate workflow layer.
- The use of tools that force consistency between (UML) models is highly recommended. In the NCS development project, using a single tool for class, activity and sequence diagrams proved to be a significant time-saver.

Conclusion

Service orientation is a construction method more than a technology. Therefore SOA can be applied to just one system, particularly when this system needs to offer functions to other systems or presentation layers. Service orientation is the next step after component based development, because the services provide a shell or facade that is meaningful to the business and hides the technical components.

Applying service orientation can help the business to create the flexibility and agility that it needs, as was proven in this case of ING Card.