iMatix Corporation

CBR Gent 2002
Project Review

1 Document Control

Author:	Pieter Hintjens <ph@imatix.com>
Version:	1.0
Revised:	16 March 2002
Reference:	i/c/cbr/g2002/r2

Table of Contents

2 Review

2.1 Aims of This Review

This document reviews the project "Gent 2002" carried out by iMatix Corporation for CBR between September 2001 and March 2002.

In this project we implemented a complex architecture on time and on budget. The aim of this review is to document the issues and risks we faced and the decisions we made, so that future projects can benefit from this experience.

This document is an edited form of the complete document (ref.i/c/cbr/g2002/r1) and as such may be distributed to third parties on a limited basis. It is not intended for public distribution.

2.2 Context and Background

The Gent 2002 project was part of a major overhaul of the dispatching and loading facilities at the factory. This laid the groundwork for a doubling of capacity at Gent.

This project at Gent follows similar (but less ambitious) projects at Lixhe (1997), Mons (1998), Harmignies (1999), and Gent (1999). These earlier projects were carried out by the same core team (Theizen and Hintjens).

Gent 2002 links a series of components together:

- CBR's central SAP server, used for the commercial chain.
- An administrative application ("the Dispatcher") running at Gent.
- A network of kiosks that automate interaction with lorry drivers.
- Two industrial networks that control the factory loading facilities – one built by Precia up to 2001, and one built by Siemens in 2001-2002.

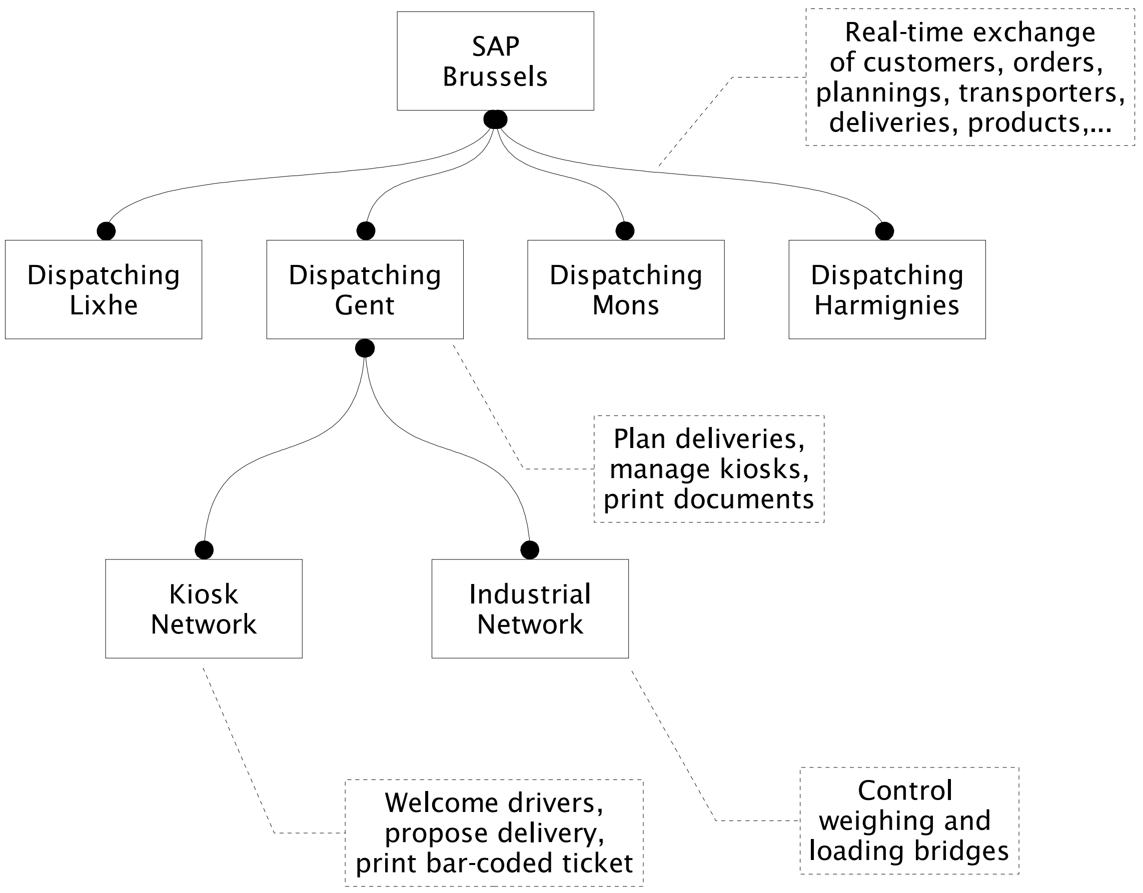
The iMatix personnel working on this project worked as several teams:

- Cretel and two colleagues worked on the dispatching application and kiosk server applications.
- Pesesse and Hintjens designed the kiosk hardware and worked with the kiosk manufacturer to produce the six kiosk machines.
- McNeill and Lucina (iMatix New Zealand) built the kiosk client application, kiosk operating system and kiosk boot system.
- Mazzoleni and Schultz implemented the STEP communications layers.
- Pesesse and Anderlin designed the graphics for the kiosk application.

These teams worked independently, in various locations, with weekly meetings for all on-site staff during the peak periods of work.

All teams worked according to a blueprint ("Gent 2002 Technical Design") written by Hintjens and Theizen during the design phase of the project.

This is the overall architecture of the CBR ordering and dispatching network:


SAP Brussels
Real-time exchange of customers, orders, plannings, transporters, deliveries, products,...
Dispatching Lixhe
Dispatching Gent
Dispatching Mons
Dispatching Harmignies
Plan deliveries, manage kiosks, print documents
Kiosk Network
Industrial Network
Welcome drivers, propose delivery, print bar-coded ticket
Control weighing and loading bridges

The components we developed or upgraded for this project were:

1. Communications between all CBR's factories and their SAP system. The new network uses the iMatix STEP middleware for information exchange. STEP also assures data exchanges between the different networks within the factory. In the above diagram all STEP nodes are shown as black dots.
2. The Dispatching system at Gent. The extensions handle the new industrial network and the kiosk network.
3. The kiosk network. The network consists of a hardware platform, operating system, boot system, and client and server applications, all designed by iMatix for this project.

2.3 Principle Risk and Measures Taken

From our past experience working with CBR and on other integration projects, we identified the greatest risks to this project as:

1. Creating the communications infrastructure. In past projects we often provided 'services' that other parties (e.g. the industrial control team) could access for data exchange. However we found that when we controlled only half of the infrastructure, we were vulnerable to errors in the 'client' layers.
2. Integrating solutions from multiple suppliers 'on-site'. In our experience the difficulty and cost of this phase is often underestimated.

3. The kiosks. The entire Gent expansion project depended on a high-quality kiosk design and implementation. From our previous experience with CBR's industrial suppliers we believed this was a serious risk to our software project.

We raised these issues from the start with CBR. By doing so, we were able to structure the project in such a way as to largely eliminate the risks:

- We took responsibility for point-to-point delivery of all data between the principle systems and applications involved, using our STEP product.
- We took responsibility for some small but important applications running on the industrial network, thus reducing the need for on-site integration.
- We took responsibility for the kiosk design and manufacturing (after trying and failing to find a supplier from CBR's existing industrial suppliers).
- We built a full duplicate of the Gent dispatching system at our offices in Brussels, allowing us to develop and test the software and hardware off-site.

2.4 Principle Issues and Challenges

2.4.1 Team Coordination

Working with six teams in three different locations (Molenbeek, Boisfort, Wellington) presented a series of challenges. iMatix already had much of the infrastructure for this kind of project. We can document the most critical aspects:

- Detailed architectural documents. We cut the project into well-defined components, then specified the interfaces between each of these components, and the overall design of each component.
- Clear responsibilities. Each team had a clear set of specifications and deliverables. This made it easy to measure the quality of our initial design against the progress of each team, and to identify and address bottlenecks before they became issues.
- Email and mailing lists. For Gent 2002 we created two mailing lists, one for discussions with CBR, and one for internal discussions. In the future we would probably create two internal lists – one for overall discussions and one for more technical aspects.
- Shared access to documents and code. We already use CVS for all our projects – for Gent 2002 we reorganised the existing source code and created a CVS archive for it. We ensured that all staff had access to this archive, and the necessary training to use CVS.

2.4.2 Data Exchange with SAP

From CBR's central SAP system, orders are sent to the various factories for processing. As orders are delivered, delivery data is returned to the SAP system for invoicing. Along with these basic data flows we also send updates of product information, transporters, customer information, and so on.

The exchange of data between SAP and the factory systems is essential for CBR's business operations. Without the fluid real-time exchange of order and delivery data, CBR cannot deliver cement or invoice their customers. If a single order or delivery is lost, this translates into direct costs for CBR and their customers.

The basic design choices for such an interface are:

1. A direct peer-to-peer connection using the built-in SAP remote access facilities.

2. A decoupled connection using an abstracted interface.

The first approach is the 'obvious' one, and that recommended by SAP. In this design, the factory dispatching systems would work directly with SAP data structures and data exchange mechanisms. We believed (and still do) that this was desirable to hide these internal structures and mechanisms. Firstly, we did not want our teams to be dependent on SAP skills. Secondly we did not want to create dependencies between work done on SAP and work done on our systems. Thirdly we wanted to retain the flexibility to exchange data with other systems in the future.

Following this we therefore recommended a decoupled and abstracted approach. We implemented this in 1998. We defined the various messages we needed to exchange. We built terminals at each end of the connection that could read and write these messages. We built a custom middleware layer to exchange the messages between the various systems involved. In late 2001 we upgraded this to using our STEP product, to eliminate the need for Oracle licenses on the SAP system, and to make the system more robust.

Today the SAP interface consists of:

1. A set of message definitions. Today we would define these using XML. In 1998 we defined simple fixed field layouts.
2. SAP objects and their BAPI methods that can read and write messages from two queues (in and out) held on disk.
3. Shell, C/C++, and COBOL routines at the factory end that can read and write messages from input and output queues held on disk.
4. A STEP network that delivers the messages between the five systems involved.

This design is very robust. It can handle data being lost for many reasons. It continues to work when the network is unavailable or a system is 'down'. The disk-based queues give control to technical personnel when they need to trace problems and recover from unforseen problems.

The migration to STEP in 2001 took place gradually over several months. During this time we ran STEP in parallel with the custom-built middleware. We ran exhaustive tests on STEP, still a fresh product at the time. We migrated the smaller factories at Mons and Harmignies first and finally migrated Lixhe and Gent.

In this case we were constrained to keep compatability with the 1998 implementation. For future SAP interfaces we would make some changes to this design:

1. Use XML definitions for all messages. A 'lightweight XML' model (documented messages without DTDs, stylesheets, or formal validation) has proven very effective in our other projects.
2. Integrate our XML-handling tools (GSL) with SAP's interface languages to create XML-capable routines in SAP.
3. Possibly, build a STEP terminal that can talk directly to SAP.

In 2002 we will help CBR to integrate a new application (a planning tool) to the architecture. We expect to be able to use the existing framework, which justifies our original idea of using an abstracted design.

2.4.3 Data Exchange with Industrial Applications

In this context, as with SAP, data exchanges are critical and we must guarantee that messages arrive, despite the possibility of network problems, unavailable systems, etc.

So, this data exchange layer is very similar to the design used to exchange data with SAP. We install a STEP node on the various machines in the industrial network that we want to talk to. We use disk-based queues to send and receive messages.

The main issues we had were:

- Defining the messages to exchange. We used a 'lightweight XML' approach for this – documenting each XML message but not using DTDs or stylesheets for formal validation.
- Using STEP creates a latency, since messages are exchanged asynchronously. We were able to bring this down to about one second without problems.
- STEP had to run on a variety of systems: Digital Unix, Windows NT/2000, and Linux. STEP is portable (built using OpenKernel). We still had to write wrappers to create STEP services for Windows and daemons for Unix.
- STEP had to be very robust, especially regarding memory use. Leaking memory would eventually paralyse the servers it ran on. We used OpenKernel's memory tracking to test STEP in heavy loads and ensure that it did not leak any memory.

2.4.4 Kiosks

The kiosks welcome the drivers, propose a delivery, and print a ticket that the driver uses to load his lorry. At CBR Gent we planned three kiosks initially but eventually expanded this to six kiosks.

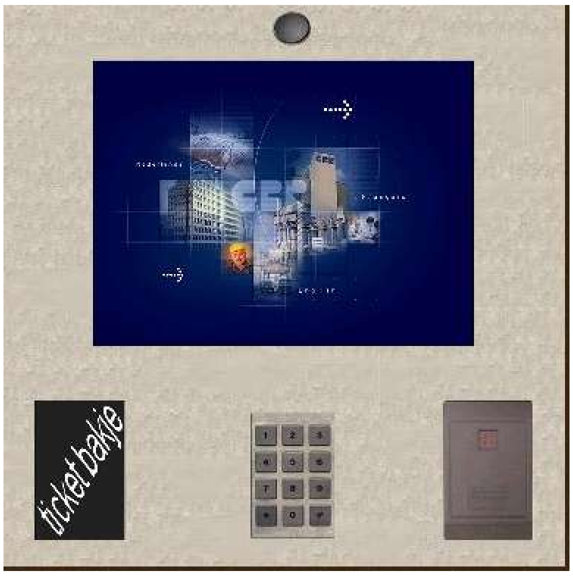
CBR's factories at Lixhe, Mons, and Harmignies have one or two kiosks based on normal PC equipment housed in special cases. We did not build these kiosks but our software had to work with them.

The main issues with the kiosks were:

1. We needed a totally reliable kiosk design. In the other CBR factories, the kiosks are near to the dispatching offices, so when the kiosks fail, manual intervention is easy. In Gent, kiosk failure means lorries cannot enter the system (which is acceptable for about five minutes).
2. The kiosks need to operate outside, in all weather. The main issues are: sunlight on the screen, moisture affecting the printer and paper, and temperature control.
3. We wanted to assemble, test, and repair the kiosks off-site. The other factory kiosks were assembled and tested on-site and in our experience this was a major mistake.
4. The kiosk software had to work in a variety of configurations. For example, if the main application server was unavailable, the kiosk software had to connect to the fall-back server.

2.4.4.1 Kiosk Hardware

Our requirements for the kiosk lead us to make a simple concept design integrating the basic elements (printer, flat panel display, keyboard, badge reader, and PC) into a single module:



We looked at industrial computers and other types of kiosk, but nothing matching our design seems to exist. Initially we assumed this would be fairly easy to assemble from standard components, but when we contacted CBR's existing suppliers, they told us it was an 'impossible' concept.

We decided to develop the kiosk ourselves and find a manufacturer able to produce it. The kiosk hardware components come from ten different suppliers. We selected the printer, screen, badge reader, badges, and embedded PC and designed the overall kiosk layout. Our manufacturer did the detailed design, found suppliers for the keyboard, the paper, the enclosure, the power supply, and assembled the kiosks.

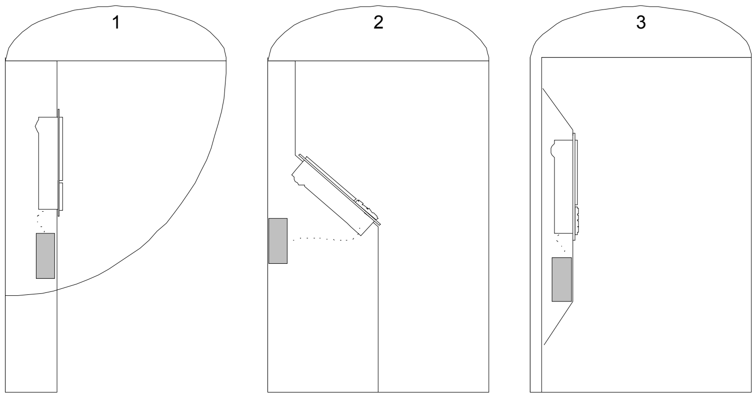
This part of the projct represented a significant risk. The project could not go live without the kiosks. We planned to have the kiosks ready by December, and they were actually delivered in early March, which represents an overrun of almost 100%.

The delay did not become a problem although one more month's delay would have. We tested the applications on other material. We had samples of the printer, badge reader, and PC well before-hand.

However, for future projects of this kind, we must plan for a production cycle of four to six months depending on the number of suppliers involved.

2.4.4.2 Kiosk Housing

Based on our kiosk module design we were able to provide CBR with recommendations for the kiosk housing. Basically this had to be in a cabin of some kind to protect the kiosk from direct rain and sunlight. The kiosk is mounted vertically or at an angle as shown in this diagram (the grey box represents the power supply):



Given this flexibility, CBR's architects were able to design a very elegant arched housing which is a work of art in itself.

2.4.4.3 Kiosk Operating System

Our kiosk design had no moving parts except for the printer. Thus: no hard drive, CD-ROM, or diskette. We wanted to be able to run arbitrary applications on the same kiosk. We wanted to be able to unplug a broken kiosk and replace it with a spare module, without reconfiguration of any kind.

To make this possible we built our own network operating system (RealiNOS) based on Linux. RealiNOS provides network booting capability based on extensions to the DHCP protocol.

When a kiosk is switched-on, it uses the DHCP protocol to find the RealiNOS boot server on the network. This server provides the kiosk with an IP address. The kiosk then loads its operating system, followed by its application programs, from the same server.

Creating the RealiNOS software was possible only because of our prior experience in building diskless Linux terminals (for customers in New Zealand). It works very well for the kiosks – the RealiNOS server runs from a bootable CD-ROM with zero installation. If the RealiNOS PC breaks down, we simply remove the bootable CD-ROM, place it into another PC, and reboot that.

We were able to build most of RealiNOS without access to the physical kiosk material. However, we had to customise the kiosk BIOS (built-in startup code) to work correctly with RealiNOS.

2.4.4.4 Kiosk Applications

We designed the kiosk application using a client-server model. The kiosk client application handles the printer, badge reader, keyboard, and screen. It talks to a kiosk server application which works with the dispatching database.

We built this as two teams: McNeill and Lucina worked on the kiosk client, while another team built the kiosk server application. As a basis for this collaboration, we made a detailed design document. We analysed the overall design of both applications using Libero. We defined all messages using 'lightweight XML' – clear documentation but no formal validation via DTDs or stylesheets. We described all the scenarios that the kiosk had to handle, and we made mockups of all the screens.

The kiosk client works with a primary server, and if that is unavailable, switches to a fall-back server. In CBR Gent's case, the primary server is the Unix dispatching system and the fall-back server is a PC on the industrial network.

In retrospect this careful design work was worthwhile. Only two people in iMatix were intimately familiar with the needs of the project when we started (Theizen and Hintjens). There were many delicate issues that the kiosk had to handle correctly.

This may be obvious, but in many projects such a detailed design is either unnecessary (when all team members are well-aquainted with the problem) or impossible (when we simply don't have the information necessary to make the design).

2.4.4.5 Conclusions and Recommendations

We can draw these conclusions from the kiosk project:

1. The kiosk project added significant value to our work for CBR and will be responsible for the smooth running of operations at Gent. iMatix appears to have been the only party with the expertise to develop this product.
2. However, we underestimated the time it would take, and this came close to causing delay on the project. We are used to being cautious with software projects – the same caution must apply to hardware projects.

2.4.5 Dispatching Application

This application, though accounting for a significant amount of work, presented no problems. We used the opportunity to expose two additional people to the internals of the dispatching database, and used our iAF tool to document the database structure.

The dispatching application is thus in better shape than when we started Gent 2002, with the functional knowledge better documented and spread more widely.

2.5 Conclusions and Recommendations

2.5.1 Architecture and Design

It is important to make a clean, simple, and appropriate design. We must use our best designers at the start of the project. For Gent 2002 we were able to make a functional and technical design that correctly identified and answered all major risks. We can standardise the methodology as follows:

1. State the requirements clearly so that any initial assumptions are obvious.
2. Design an overall architecture that answers the needs of the project.
3. Break the architecture into well-defined components. Give the components clear names so that everyone agrees on what is what.
4. Define the interaction between each component. Formalise these interactions as XML messages when possible. Assume that we will use STEP for message exchange.
5. Define the structure of each component in terms of technical implementation, internal design, functionality. Use Libero for event-driven components.
6. Describe all end-user scenarios and detail how these are implemented as processing steps and exchanges of data.
7. Establish a cost estimate for each component. Add the cost of unit design and testing and integration testing. Add project management at about ten percent.
8. From this, define a planning and get the customer's approval of the planning.

This type of document can be seen as a combination of feasibility study, request for proposal, proposal, and technical reference.

2.5.2 Team Organisation

It is possible and desirable to create small teams that each work on one or more components of the overall architecture. Small teams (two or three people) communicate well and resolve problems rapidly.

This type of organisation is possible only with good communications and shared access to source code and documentation.

Recommendations:

- All projects should be housed in a CVS archive.
- All projects with more than three or four participants should get their own mailing lists.

2.5.3 Testing and Quality Control

We (Hintjens et al) experienced the joys of integrating hardware and software on-site (Lixhe, 1997). CBR Lixhe started loading at 6.00 am. This is the time when drivers start using the kiosks. This is the time when the kiosk badge readers, printers, or software fails. There are a few ways to be at a distant factory early in the morning. Wake-up at 4.00am, sleep on the factory floor, or stay awake all night.

For Gent we chose to make sure the system was unbreakable. This meant testing all the software and hardware in our offices, at normal hours.

We built a replica of the Gent dispatching network, using Linux PCs to act as Unix servers (Oracle runs well on Linux), and other PCs to act as kiosk servers and clients. We did not simulate SAP nor the rest of the industrial network but we could have.

This worked. The cost of this material and configuration was not trivial but much less than the alternative of testing on-site.

Recommendations:

- For any project with significant complexity, we recommend therefore that attention be paid to building a realistic test environment at iMatix.
- We have to continue the practice of nominating a 'product manager' responsible for quality control and testing.

2.5.4 Data Exchange

The 'lightweight XML' design approach works very well – it allows us to make a design rapidly with little overhead at design time or at runtime. For example, for all exchanges between the kiosk client and server applications we defined an XML language (Kiosk Control Language, or KCL). We experienced no problems that could have been prevented by a 'heavier' approach involving DTDs or stylesheets.

However, we may consider using formal XML validation in the future, to protect against errors caused by 'specification drift'.

Recommendations:

- For all significant XML exchanges, define an XML grammar using the iAF XNF (XML Normal Form) tool, and generate a validating parser from this grammar. This parser can be built into the kiosk and into STEP.