CD219: Agile custom ABAP development
Develop like never before …
…with Business Object Processing Framework and Floorplan Manager

I'd like to present to you our experience with development tools just recently provided by SAP.

For the last 18 months, we've been using this toolset aims at facilitating development of custom transactional applications with ABAP and our project was the first one to use it outside of SAP.

As you may guess, I'm quite enthusiastic about it (just like any other speaker here at TechEd) and I hope that you will share this after my presentation.
Agenda

OUR PROJECT

THE ENVIRONMENT

USING THE TOOLSET
In general you can easily imagine, that Deutsche Bahn has quite specific **requirements** when it comes to provisioning of IT solutions which support the core business. The reason is quite obvious: There are not too many companies with a >50*10^9€ (A. E. Billion) business which aim at running some 32.000 trains on about 34.000km of tracks.

DB Systel is the internal provider for IT solutions at Deutsche Bahn.
Those tracks are the legal property of DB Netz AG and the reason why a custom development project was initiated:

While the maintenance of those tracks itself is operated within one of the biggest SAP plant-maintenance systems around the globe, the purchasing process for supply and disposal with goods for constructing the tracks is quite complex.

Construction and renewal of tracks requires the movement of material which you can’t order shipment included at an online-store: Only few suppliers provide ballast (Schotter), sleepers (Schwellen), switches (Weichen) and rails (Gleise) and the actual cost of the procurement highly depends on the shipping cost which again depend on the distance between supplier and location of the construction site. A lot of interaction with the supplier is necessary in order to agree on possible dates – and this has to be done quickly in case of emergency processing.

Our system (project name ISM-PrIMa) shall optimize the purchasing process and integrate with many processes which are quite special to the railway-maintenance. An adaption of an ERP-module was analyzed as to be highly disruptive and thus was rejected.

Since all processes which involve materials shall be operated via SAP and due to the need for integration with other SAP-components (particularly materials management) it was decided to start a custom development project on ABAP technology.
• **Duration** for development (incl. developer’s test) about 1.5 years → Support for legacy-application is discontinued

• Average development **capacity about 10FTE** (plus consultants, testers, operations team, administration) → Huge scope, highly parallelized work

• Strategic **partnership** with external Indian company → Budget

• With respect to our processes, the customer requested to have „**classical waterfall-principle-based**“ phases and milestones, such as “design”, “implementation” and “testing”.
Now that you’ve got a short glimpse at the boundary conditions which were set directly or indirectly by our customer, we’ll shortly look at how we as a project team *set up our development process* and which *technical components we decided* to use.

This shall just give you an idea under which circumstances we have made these experiences and lets you judge which aspect might also be *applicable to your projects*. Also, I’d like to inform you about the technical necessities.
Before we head-over to the technological side, let me explain how the project is organized.

This is important to understand as the technology should support the process – and the process should make use of what the technology offers.

**Input**

was a *classical specification* which was believed to be complete from the functional side.

After a „quality gate“, the development team („the techies“) took over the responsibility for the phases „design, implementation and component testing“ – before moving to “integration testing” after another quality gate. For this timeframe of about 14 months, we decided for an *iterative* approach: It’s not „Scrum“ – but an adapted model:

- **Cycles** of about two to three weeks
- **Consultants represent customers** which are abstracted by the level of the specification document which was signed-off
- Before each iteration, the consultants and the development lead agree on a
potential scope which is chopped from the functional specification

- In a planning meeting, the „customer“ (=consultants) explains the business scope of the upcoming iteration to the whole team
- Afterwards, the tasks are divided location-wise and the teams break down the functional requirements to technical entities
- In a daily standup-meeting, the status is captured and current issues are being addressed
- During the review-meeting, the final status of the issues completed is demonstrated by the responsible developer and signed-off by the consultant – or returned to the backlog

The developers

involved in the project are more-or-less at the same age and have a similar technical background with ABAP experience of two to eight years.

After the last iteration, there are the classical three stages of testing – we’re doing continuous testing nevertheless throughout the whole development timeframe. This cannot be bad as bugs and concept-related issues are much cheaper to correct the earlier they can be recognized.
From the beginning, we started of to use a technical toolset provided by SAP. It consists of ABAP-based frameworks which are also the foundation for SAP Transportation Management (TM): **BOPF** (for the “classical backend”), **FPM** (for the user interfaces) and integration layer between them which is part of the BOPF re-usables: **FBI**, the Floorplan Manager BOPF Integration.

**Motivation:** An interactive transactional application is quite complex to develop from scratch and involves many technical services which are tricky to provide and which can slow-down implementation if developed in parallel to the actual business logic.

**Using components provided by SAP** promises significantly improvements in this area as they appeared on the screen some time ago, have already been consumed, corrected and are maintained by SAP if something does not work as expected.

**Expectations:**

- Lower TCD
• Lower TCO

• Increased stability, particularly during early development phases

The rest of this presentation is dedicated to short introductions towards the frameworks, their services, responsibilities, where their use facilitated development and which challenges had to be faced by the team in order to exploit the benefits.

Furthermore, I’ll try to picture how technology integrates into the development process chosen.

The first thing which I’ve been usually asked is about the technical requirements for using those components, their maturity and about their price tags.

• 0 cost – included in Business Suite foundation and Netweaver

• Minimal release is ECC 6, EhP 6 - our customer upgraded from EhP 0 because of to the benefits expected.

• BOPF and FPM are both established components which have been used within the SAP development for more than 7 years. Although they are mature, both are still being continuously enhanced in order to keep up with state-of-the-art technologies. FBI has been part of SAP transportation management and has been around as separate component for only two years.
I would also like to go much more into details, but this presentation shall give you at least an idea about the experiences, show benefits and potential pitfalls, so I'll try to keep quite a high level of abstraction. If you want to get more into the details, I'd enjoy if you contact me afterwards or check the resources mentioned in the end.
Just like we as a development team did not jump into the deepest depths of the technology, *I'd like to iteratively share the experiences with you.*

What I present in each iteration *more-or-less corresponds to the actual tasks and topics* we covered during the first three iterations and serves as agenda for the remaining presentation.

Those were the aims of the iterations:

1. Get a basic application running – including a UI which allows our consultants to judge whether we have understood the requirements properly
2. Implement basic logic: „Checks“, „Buttons“, „Derivations“
3. Include document management (knowledge provider), change documents and in general: Make it more usable

As of #4: content covered in #1 to #3 in different mixtures

I would like to present some key-elements which we have encountered in each of these iterations and – just like in the project, I would judge that in a short „review“ after each cycle.
The first thing we’ll do is to translate the input from a functional specification to entities in our system.

Specifications usually include a domain model, so design starts with creation of technical representations of them: Classes, structures, database tables.

With BOPF, the business objects processing framework, this is a bit different: After having verified the domain model, the entities can be modeled as so-called Business Objects with almost no transformation in a SAP GUI-based transaction.

- Outside-In-Approach => low representational gap. A domain class becomes a so-called “Business Object Node”
- Representation of the domain models in the system - without designing ABAP-classes and without the need of writing a ”design” it in a text-editor. You can export the models to MS-Word in case you need to publish a design document.
- The necessary runtime-artifacts can be generated from the modeling tool or are implemented generically by BOPF.
BOPF exposes services which provide **generic interaction patterns** with all business objects.

In addition to the well-known CRUD, BOPF also provides **other core-services**

Most important:
- Read ("retrieve" in BOPF language) also includes **retrieve by association** which allows to address related entities based on the model
- A dedicated service allows to read the so-called **properties, which is a definition of visibility, changeability** or execute-ability for various entities. This way, the logic which changes the usability of these entities can be implemented in the business objects and this way **de-couples the user interface**
- **Query** allows to identify a set of instances based on modeled criteria

All those services are **provided without having written a single line of code**.
⇒ „Out of the box“ the developer can operate on the business objects once they have been modeled with full-fledged support for transactional behavior.

BOPF provides a SAP GUI-based generic **test-UI which allows to operate on the objects** – and gaze at the entries on the database afterwards.

This allows **fast development and immediate interactive test** – and shows the separation of the GUI from the backend functionality.
A UI is made for the interaction with a human – a model usually is not. Thus, the next task is to connect the underlying model to a UI.

In order to harmonize the interaction of UI with user and backend, SAP provides the floorplan manager for Webdynpro ABAP (FPM). FPM does not only render pixel-perfect independent of the browser, it also provides a common roundtrip-sequence which includes a sophisticated eventing mechanism.

We have to look at how FPM essentially works in order to understand the integration towards the models:

- FPM is a UI-framework based on building blocks which are controlled using feeders and which are wired using connectors. If a custom development project aims to use FPM, it needs to provide implementations of interfaces defined by FPM which imply well-documented, but not too simple contracts.
- This is where the Floorplan Manager BOPF Integration (FBI) comes into the picture: FBI provides generic set of feeders and connectors which can be bound to the underlying BOPF-models. FBI supports
  - Mapping structures (to form-fields or list-columns including propagation of domain-information such as value-sets)
  - Wiring based on modeled associations
  - Edit-Mode and Lock-Handling
  - Transaction management including data-loss-protection
  - And much more which we did not value in iteration 1

⇒ No code needed in order to create a ready-to-use-UI. We will have a more detailed look into FBI and how it integrates to the other frameworks in iteration 3.
Within the first review, there was only positive feedback by the developers, the customers/consultants and project- and development lead alike:

**Rapid** development
- Much reduced code-effort
- **No UI-Mockups – real applications!**

**Semantical model**
- **Better communication:** Once you have modeled your objects, you precisely address them in daily language ("I create a demand in the request folder")
- Consultants and developers use the **same terminology due to low representational gap**
- **Transparency:** Each model is readable within the modeling environment in a generic manner: Once you have understood the meta-model, you are able to understand every model
- For understanding it, no additional design documents are necessary. The truth about how the system works is in the system.

**Powerful modeling environment**
- Reuse of known tools (DDIC)
- Good integration (DDIC, class-builder, Test-Tool)

**High motivation**
• **Presentable** results
• **Immediate feedback**
• **Early validation of requirements**

⇒ **High stability** of the cores of BOPF, FPM and FBI enabled a **kick-start** and everybody was enthusiastic about „BOPF“ - even if only the developers knew that BOPF itself as only responsible for one part as this.
CRUD is hardly what a customer is happy about and why he decides to invest into an SAP-based development project.

In iteration 2 we focused on the dynamic aspects: How can we actually implement business logic and bring it to the UI.

Apart from that, we also modeled the most complex domain as business objects („hardest second strategy”), but this shall not be the focus for us now as the tasks and judgments are the same as in iteration 1.
We are speaking of „Consumption of BOPF“ when thinking of an interaction with a business object via BOPF.

**Major consumer is the UI**, but it could also be a classical report which is executed in a batch-job. *As the consumption from the UI is generically implemented by FBI, we did not use those interfaces, but as the patterns are also applicable for the provisioning side, I’d like to start with that.*

Via the **service manager** interface, BOPF provides access to a single business object.

Interactions via its interface-methods all follow common principles:

- Data transported in the signature is typed generically ⇒ **No type-check while coding**
- Each method represents a command: *The entities addressed are part of the method signature* (e.g. „retrieve( request folder root)“).
- All interfaces are mass-enabled (**operate on multiple instances**) for **performance-reasons**.
- **Model entities addressed** in the commands are coded as globally unique identifiers (GUID)s. The design-time environment generates constant-interfaces for allowing to use them in code.
- The actual **business data is transported in internal tables** based upon structures (no instance of an ABAP object). **Instances themself are referred to via GUIDs** as well (you can think of them as serialized pointer)

By the **transaction manager** interface, it offers methods for **handling and monitoring the state of the transaction** as whole at which all instantiated service
managers take part.
Also while implementing business logic, BOPF remained a black box. We knew it somehow manages buffering and database-access, but we did not have to care about that. We could just focus on implementing ABAP classes containing business logic.

All that's been said for the consumption also applies for provisioning of logic: BOPF has defined ABAP interfaces which establish a contract, they follow the command pattern, address model entities by GUIDs from the same constant interface and are mass-enabled.

BOPF differentiates between changing logic (which can be triggered implicitly as a side-effect (determination) or explicitly by a consumer (action)) and checks (validations). Each kind requires a different interfaces to be implemented and offers a different set of configuration which makes the framework trigger the logic (inversion of control). This way, BOPF enforces developers to modularize semantically (based on the single responsibility pattern).

Into those interfaces, BOPF injects objects which provide the operations which are necessary in the context of the logic (e. g. each of the interfaces has a read-accessor, but a validation has no modify-object in the method’s signatures as checks must not modify the state of an instance).
Let's start with the state we left in iteration 1. As you surely remember, the UI was built on building blocks which are referring to nodes of the business object and by interconnecting them based on associations.

The logic added in iteration 2 is only reflected indirectly on the UI: Validations and determinations are executed after data manipulation (updates) issued by the user and only become “visible” by the messages they create. Only actions have a representation on the UI: They are usually visualized as buttons.

But how is this being integrated?

→ Let’s have a look at an example of some checking logic which prevents the execution of an action:

• User pushes the button
• FPM receives the interaction from the Webdynpro and creates an event and passes it to the feeder.
• FBI issues the command of executing the action submit_for_checking of the demand selected to BOPF.
• Before executing the action, BOPF triggers a configured validation.
• This validation refuses the action and creates error messages.
• These messages are translated by FBI and are propagated to FPM
• FPM publishes the texts of the messages to the message-area and highlights the fields which represent the attributes which have been referred to in the message.
For all this interaction, no code in the UI had to be done. The whole UI-roundtrip is handled generically. Only artifacts for the validation- and action-logic were implemented - in the backend.

Apart from business logic, structural enhancements of the existing objects have been made during iteration 2. E. g., we added a new node into a business object. Just like in iteration 1, we simply had to configure a UIBB to bind to this node and connect it using the composition association.
No code was involved there as well.
There have been very **differentiated opinions** about what we experienced in iteration 2:

- **Developer's were racking their brains** about the **technological details:**
  - GUIDs: Having understood that the methods can be understood as commands issued and getting used to using the constant interface for coding those commands was quite easy: **You don't see the GUIDs while coding – but while debugging.** As also the instances are addressed by GUIDs, some of us were **sick of GUID-overflow.**
  - **Mass-enabling** is crucial for performance, but also sometimes really **tricky to implement.**
  - **Mis-use of generically typed interfaces** led to many **runtime errors.** Also, those dumps which were raised intentionally by BOPF (e.g. if a structure which should be used as target for a data retrieval did not match the node's structure) were **not too verbose.**
  - While **analyzing mis-uses of the framework,** developers can’t avoid to **debug the framework** – and this has quite some challenging code which is highly optimized but still generic. It’s great to know how BOPF works internally, but it **takes some time to understand how BOPF operates** and understand the own fault consuming it.

Apart from the brainy challenges, everybody was more than happy about the intermediate results:

Since every part of logic and the configuration about what triggers the logic is
readable in the design-time, transparency increased a lot – particularly, if you adhere to the single responsibility principle.

Due to the programming paradigm, adding and changing atomic parts of business logic does not have a negative impact on performance and stability of the overall system.

The generic interfaces are a challenge, but the way of “talking” to business objects is universal. Once you’ve reached a basic level of comprehension, developers could work in each business area and did not have to read a (potentially obsolete) design-paper upfront and every part of the application feels familiar.

From a customer’s point of view, adaptability was a major benefit. As there is less code involved, especially changes to the UI (e.g. moving fields to another tabstrip, changing labels, ...) were done quickly.

All these features actually enabled an iterative model:

- People were not bound to a particular area of knowledge
- Short feedback cycles with the customer enabled fast feedback and resulted in high motivation
- Atomic enhancements enabled a flexible scoping of each iteration and consistent results at its end.
In iteration 3, we majorly went on with tasks similar to the first two iterations. But in addition to that, we had to accept that Business Objects – even though modeled nicely – are not always consumable by an end-user without transformation. Also, there is logic in an application which is hard to be classified as “business logic” (e.g. raising confirmation dialogs with dynamic texts) – so it was time to look at the features the toolset has in stall for that.

Furthermore, we scoped requirements which included the connection of two technical „SAP standard components“ to our objects.

→ I’d like to show to you how this can be managed with the help of the BOPF re-usables
Up to now, I always stated that the UI was just being „bound to the models“.

If we have a closer look at the business object models we’ll notice, that it’s a good idea with respect to consistent processing to adhere to some normalization principles. In real-project-life, it rarely happens that users are happy with a normalized UI. Mostly, users would like to see everything that’s relevant for their tasks on one screen.

Our escape from this dilemma was a component which is part of the BOPF re-usables: The Floorplan Manager BOPF Integration (FBI). This component is – as you might remember – responsible for feeding data into the UIBBs and for controlling the floorplan.

Let’s have a look at how FBI works. It’s a framework and as such it never operates on its own – It needs configuration which it interprets at runtime just like the other frameworks: BOPF interprets business object definitions, FPM interprets UIBB-component configurations and just like that, FBI also has a configuration object. This so-called FBI-View „wraps“ a business object (precisely a node) and can be used as source for the feeder. Via this entity

- de-normalization can be modeled
- mapping-instructions can be defined (such as translating a timestamp to date & time)
- UI-specific action-behavior (such as confirmation dialogs for actions) and UI-texts can be bound to backend-entities

If the declarative approach provided at designtime is not sufficient, FBI offers exits
at multiple levels which can be used (in order to e. g. execute UI-specific actions which trigger multiple backend actions).

→ This way, we were fully flexible to adapt the user interfaces to our needs.
While there are many business components which we also address via the usual interfaces, there are quite some technical components which are also consumed in SAP products which provide a high value for custom developments as well.

To some of them, the BOPF re-usables provide an integration with which they can be consumed as if they had been implemented with BOPF.

→ For us, storage of attachments which are linked to our object’s instances and the documentation of changes were a major requirement.
For both components, SAP provides business objects which wrap the Netweaver-functionality of the Knowledge Provider and the Netweaver Change Documents. Thus, the existing behavior and customizing of those components can be re-used completely with the additional benefit of a seamless integration into the world of business objects.

All what’s been said up to now with respect to programming paradigm and UI-integration applies to those wrapped components as well, this particularly includes

⇒ Consumption via the same interfaces
⇒ Generic integration into the UI via FBI

It’s always the same paradigms, so once you’ve understood the mechanisms, you’re easily able to understand the interaction with the wrapped components without knowing how to actually talk to the foundation-components.
FBI is **yet another framework**: Generic interfaces enable powerful generic implementations and re-usable components which are quite tricky to develop if you have to do that on your own.

If such an implementation is provided by someone else, one only gets in touch with **complexity for analyzing errors**. Therefore, it is crucial for SAP-provided components to check the contract of their usage and clearly **prevent mis-use**. This can be done via designtime-checks, good upfront-documentation or latest with assertions at runtime.

→ For each component, at least one developer understood it thoroughly. After a knowledge transfer, the rest was (at least partially) enlightened.

When we’ve been using the infrastructure, each of the **components had a different maturity** and **different functional gaps which we closed via generic code**. This was tricky, but once mastered, **significantly reduces development efforts**.

For consultants, the BOPF **re-usables provide integration to known components which also increased confidence in the solution**. If errors occur there, we can address them via OSS-messages.
Finally, we have seen many effects but I have also told you about the boundary conditions and challenges.

As a developer, I’d say that the usage of the frameworks is defined by an interface: You have to provide some parameters which influence the output of the frameworks’ implementation.

Boundary conditions which are defined by the project (importing)

ARCHITECTURAL_UNDERSTANDING: The team’s capability of understanding architectural principles, object orientation, design patterns
LEARNING_MINDSET: Whether the team likes to learn new stuff or prefers sticking to known technology
ABAP_SKILL: Are the developers able to use advanced features of ABAP, such as generic interfaces, RTTI, class-based exceptions/messages or at least programming to an interface
ATTITUDE_TOWARDS_MODELING: Are the developers able to abstract from code and to think in objects and responsibilities? Do they want to “code” or to “develop”?

For our project the following parameters were applicable

• Architectural understanding
  Most developers could think in objects. Having a tool (BOPF) which makes the objects become “alive” in the system got the whole team thinking in an object-oriented sense within some days. Actually, speaking of abstracted entities was easier for the developers than for the consultants (who sometimes went on talking about “tables”).
• **Learning Mindset**
  
  As said, it was a quite **young and not too experienced team of developers.**
  Everybody was eager to learn and to apply this knowledge

• **ABAP skill**
  
  In our project, developers at least knew how to program to an interface and what a factory (service locator) is. Before starting with the actual development, a **five days lasting training which included pair programming** built a sufficient foundation.

• **Attitude towards modeling**
  
  Not too many developers had thought of intensive configuration before this project – due to the additional effort. After the initial training, everybody **enjoyed it because of the ease-of-change.**

**We got from that**

• **Speed and effort**
  
  We were in fact **much faster** than we’d have been with an own architecture. One major reason for that was that we could start with the implementation of the actual business logic on day 1 – **without the need of building technological layers upfront.** The generic implementations done in FBI and FPM accelerated UI development tremendously. This closely relates to the much reduced effort. I sincerely can’t tell you how must less effort it exactly was, as instead of delivering earlier, the scope was much increased during the iterations.

• **Architectural quality**
  
  Due to the concepts which are implied by the frameworks, **many architectural principles** (single responsibility, loose coupling, separation of logic from the UI) were **enforced.** This improves maintainability as well.

• **Agility**
  
  Being able to configure many aspects (instead of coding it) and the stable technological foundation enabled us to adapt to changed and enhanced requirements in a much easier way. This benefit can actually not be overrated. It enables state-of-the-art project management and significantly increases satisfaction in the daily work.
Further information:

- FPM on SCN: http://scn.sap.com/community/web-dynpro-abap/floorplan-manager/
- BOPF and the BOPF reuables incl. FBI on SCN: http://scn.sap.com/community/abap/bopf
- Classroom-training for BOPF overview at SAP university: https://training.sap.com/v2/course/wdebof-nutzen-des-sap-business-object-processing-framework-bopf-classroom-001-de-de/