

# A pattern language for reconciliation

Lotte De Rore, Monique Snoeck, Guido Dedene  
LIRIS Group, Faculty of Economics and Applied Economics, K.U.Leuven  
Naamsestraat 69, 3000 Leuven, Belgium  
{Lotte.DeRore, Monique.Snoeck, Guido.Dedene}@econ.kuleuven.be

## Abstract

*Although it is a generally accepted principle in ICT to manage an information record representing a specific real world object by one and only one ICT application, for several reasons, it can be allowed to store this information in different applications and/or databases. Often the coherence between these different records can be disrupted after replication or duplication of the data. Therefore, a reconciliation process should be implemented in order to detect and report the inconsistencies between the master and slave application. This pattern language documents the decision whether or not to set-up a reconciliation process, and the alternatives to implement that process.*

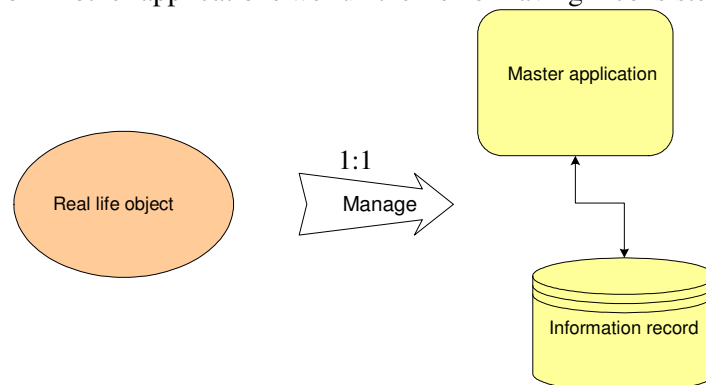
## 1. Introduction

### Definition reconciliation

Several definitions can be given to the term ‘reconciliation’. In this context, a reconciliation process is a process implemented to detect and report the inconsistencies that can appear between two datasets that represent the same information, in particular between a master and a slave application.

### Master-slave principle

It is a generally accepted principle in ICT to have an information record representing a specific object out of the real world managed by one and only one ICT application. For example: We will manage the information concerning a specific customer John Doe in one application. If we would also manage this customer’s information in other applications we run the risk of having inconsistencies.



However, for several reasons (functional as well as technical), information can be allowed to be replicated to other applications so they can access these information records. In that case, they are not allowed to manage (change, delete or add) this information. Such a setup is typically known as a **master-slave model**.

For example: We bought an application which has its own component to store customer information. This application expects all customer information to be stored in that specific component as at run time it only accesses this component. To guarantee consistency we’ll disable all management facilities concerning customer information of the application and keep its customer information component synchronised with the master application for customer information.

In a basic master slave model the slave contains the same view on the real life object. The technical features of the representation can differ between master and slave.

For example: If we look at customer information it is possible that the slave will contain less information elements concerning a customer or that some elements refer to another semantically equivalent list of allowed values, ....

But it could also be necessary to have a specific view on a real life object rather than the general view of the master application. For several reasons it can be necessary to create that view in another application which is therefore also a slave application. A slave can be more than just a 'one on one copy'.

It's a slave application with respect to the real life object, but as this application is the only application that can create this specific view, the application will, at the same time, also be a master application, namely with respect to that specific view.

For example: The financial accounting application of a company should provide an accurate overview of all debts, claims, rights and obligations of that company. The analytic accounting application wants to offer a detailed view on the profitability which is yet another view on the operations and agreements with the customers managed by a master application. These slave applications are the master application with respect to these specific views.

## **When can a master-slave construction be used?**

Besides functional reasons, there can also be technical reasons to allow the use of a master and slave construction. For example, when one works with third party applications, packages or when data is needed on different technical platforms or in different compartments, it might be necessary to duplicate the data. Another possibility is when data from the operational environment is replicated in the information layer. In each of these scenarios, one has to decide who is the master and who is the slave in the construction.

## **Avoid inconsistencies**

Information about one business reality is stored in different applications and/or data bases. Often the coherence between these different records can be disrupted after replication or duplication of the data. There are several ways to avoid inconsistencies between the master and the slave application:

1. consistency by design:

If slave applications can be avoided we achieve consistency by design. As real life objects are managed, referred and viewed by one single application, there can't be any inconsistency as there are no other representations of a specific real life object.

2. master-slave interaction models:

- a. State oriented interaction model:

The master application of real life objects will translate events or requests out of the real live world to transactions that will change the state of the representation of the real life object. One way to feed the slave application is to forward the new state of the object.

- b. Event oriented interaction model:

Another possible way to feed the slave application is to forward the events to the slave. The latter should then translate these events to an equivalent transaction that should be applied to the slave information records.

- c. Mix of state and event oriented interaction model:

In between state and event oriented interaction models some intermediary variants are possible where a mix of event and state information flows from the master to the slave.

## **Still inconsistencies possible**

Due to various reasons, the slave information created by a slave application can differ from the one managed by the master application. Some possible causes are

- Human errors in functional design or specifications
- Coding errors

- Human errors in manipulation of technical objects
- Hardware or system software failures
- ...

So, one can not presume that a slave application will always be a 100% accurate replica. Therefore, a reconciliation process should be implemented in order to detect and report the inconsistencies between the master and the slave application.

## 2. The Pattern Language

### Audience

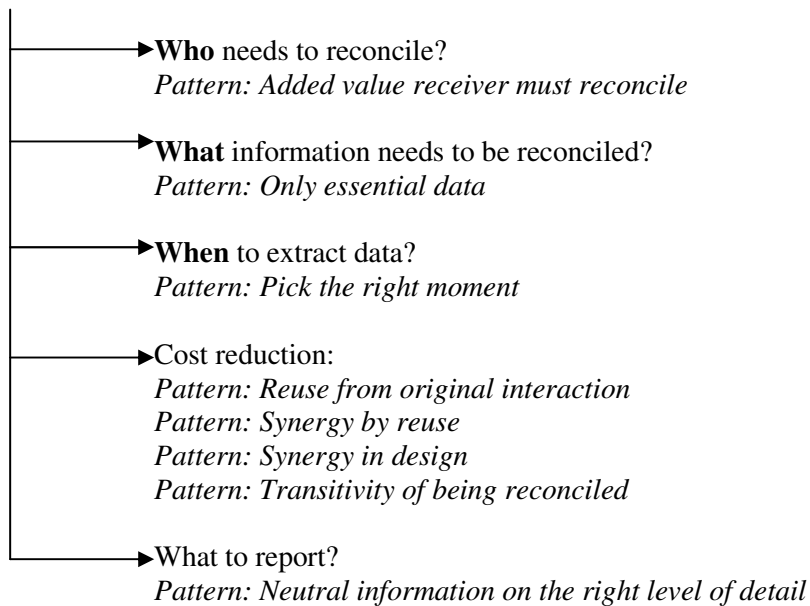
Business-analysts, analysts, architects and work preparation managers

### Overview

Do I need to reconcile? *Pattern: Balance costs*

↓ YES

How do I set up the reconciliation process? *Pattern: Reconciliation process*



Remark: Most of the patterns are not only applicable in the master-slave construct situation, but also in the more general situation where you have to compare two datasets about one business reality. The context of each pattern describes which situation is applicable for the specific pattern.

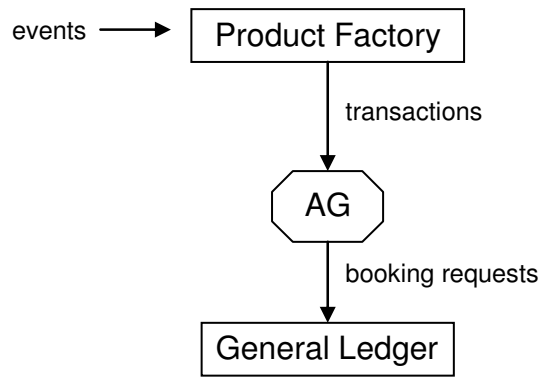
### Example

The General Ledger (GL) is an application for financial accounting. It forms and shows an accurate view of all debts, claims, rights and obligations of a company and all events leading to this state.

For that purpose, GL will be fed by various applications of core, e.g. product factories, or non core nature, e.g. personnel administration. These applications are the master application and GL is their slave. By means of the inventory control, the consistency between the inventory of the product factory and GL balances of the accounting domain is investigated.

As a running example through several patterns of the pattern language, we will investigate a loan product factory as master application that manages agreements with the customers and contains on its own all relevant information concerning the agreements such as information on the full amount a customer can lend, the current amount, interest and various charges the customer dues to the bank,...

All this information and related events should be reflected in the GL. So the loans product factory is the master and GL is the slave. In this case we have an event-based interaction model where the product factory forwards the transactions it executes based on the real life events. A specific accounting generator (AG) translates these transactions to booking requests for the GL.



## **Pattern: Balance costs**

### **Context**

Information about one business reality is stored in different applications and/or data bases. The coherence between these different records can be disrupted after replication or duplication of the data.

### **Problem**

When data about one business reality is stored at different places (e.g. working with a master slave construction), the risk of inconsistency is inevitable. **Is action needed to solve these inconsistencies?**

### **Forces**

The two main forces here are Business Importance and Costs:

Business Importance:

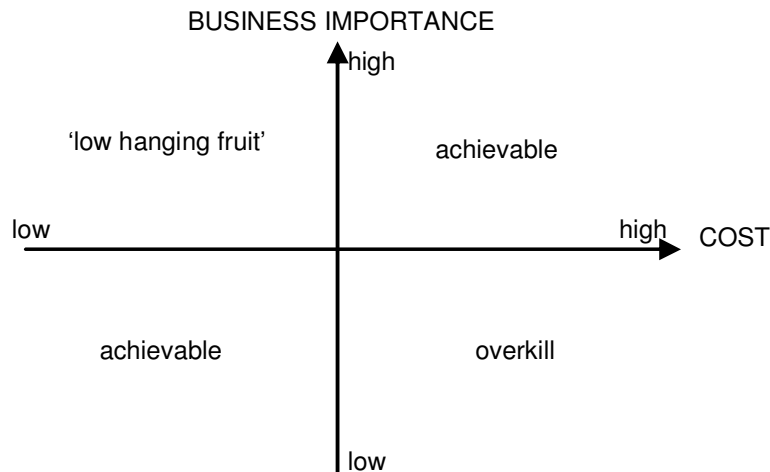
- Obligation: Sometimes there is an explicit order to use reconciliation processes:
  - o At request of the company revisers
  - o At request of audit
  - o At request of the business or end users
  - o At request of marketing department
  - o ...
- Quality: The applications need to be as correct as possible. As many as possible of the inconsistencies between the different applications need to be resolved.
- Risk: An undetected inconsistency involves a risk and a possible damage.

Cost:

- Development and exploitation costs of applications need to be kept as low as possible

### **Solution**

The decision to build a reconciliation process depends on the combination of the costs for the reconciliation process and the importance for the business.



**As the reconciliation process should limit the inconsistency risks, the cost of it must balance the potential cost of inconsistencies.** This cost can be expressed in terms of business importance.

If the cost for reconciliation (development and exploitation) is high, reconciliation can only be justified when there is a big business importance. This importance may be due to an obligation to perform a reconciliation, or due to a need for very high quality of the process (all inconsistencies need to be resolved) or due to the fact that an undetected inconsistency involves a great risk and possible

damage. If business importance is not high and an expensive reconciliation process is implemented, there will be 'overkill' and money will be wasted.

In the case of less important processes that involve little risk and no damage the only achievable solution is a reconciliation process with a low cost.

In the case of low costs for the reconciliation process and high importance for business, we speak of 'low hanging fruit'. It is a bonus as the goal can be reached with little effort.

REUSE FROM ORIGINAL INTERACTION, SYNERGY BY REUSE, SYNERGY IN DESIGN or TRANSITIVITY OF BEING RECONCILED can be used to reduce costs.

## **Consequences**

A reconciliation process will only be implemented when this can be justified according to the business importance.

## **Example**

Inventory control is the legal requirement to compare the inventory of the product factories with the relevant accounting positions. As a company is legally liable to record all transactions in a general ledger system, business needs certainty about the correctness of these transactions in GL. So the business importance force in this case is high and a reconciliation process will be implemented.

## **Pattern: Reconciliation process**

### **Context**

Information about one business reality is stored in different applications and/or data bases. The coherence between these different records can be disrupted after replication or duplication of the data. A reconciliation process is needed to detect the inconsistencies.

### **Problem**

How can the data about the same business reality that is stored in different application and/or data bases be compared and how can the inconsistencies between these be reported? **How do you set up a reconciliation process?**

### **Forces**

- Every source of error can need a different approach to be detected. There are several possible causes why the information stored in one application can differ from the one managed by the other application:
  - o Human errors in functional design or specifications
  - o Coding errors
  - o Human errors in manipulation of technical objects
  - o Hardware or system software failures that affect data
  - o ...
- The quality of the reconciliation process needs to be as high as possible, this means a reduction of the FP (false positives) and the FN (false negatives) of the errors that are detected with the reconciliation process:
  - o The applications need to be as correct as possible. As many as possible of the inconsistencies between the two applications need to be resolved.
  - o An undetected inconsistency involves a risk and a possible damage.
- Development and exploitation costs of reconciliation applications need to be kept as low as possible.

### **Solution**

ADDED VALUE RECEIVER MUST RECONCILE defines **who** needs to implement the reconciliation process.

The reconciliation **process** can be set up as follows and can be seen schematically in figure 1:

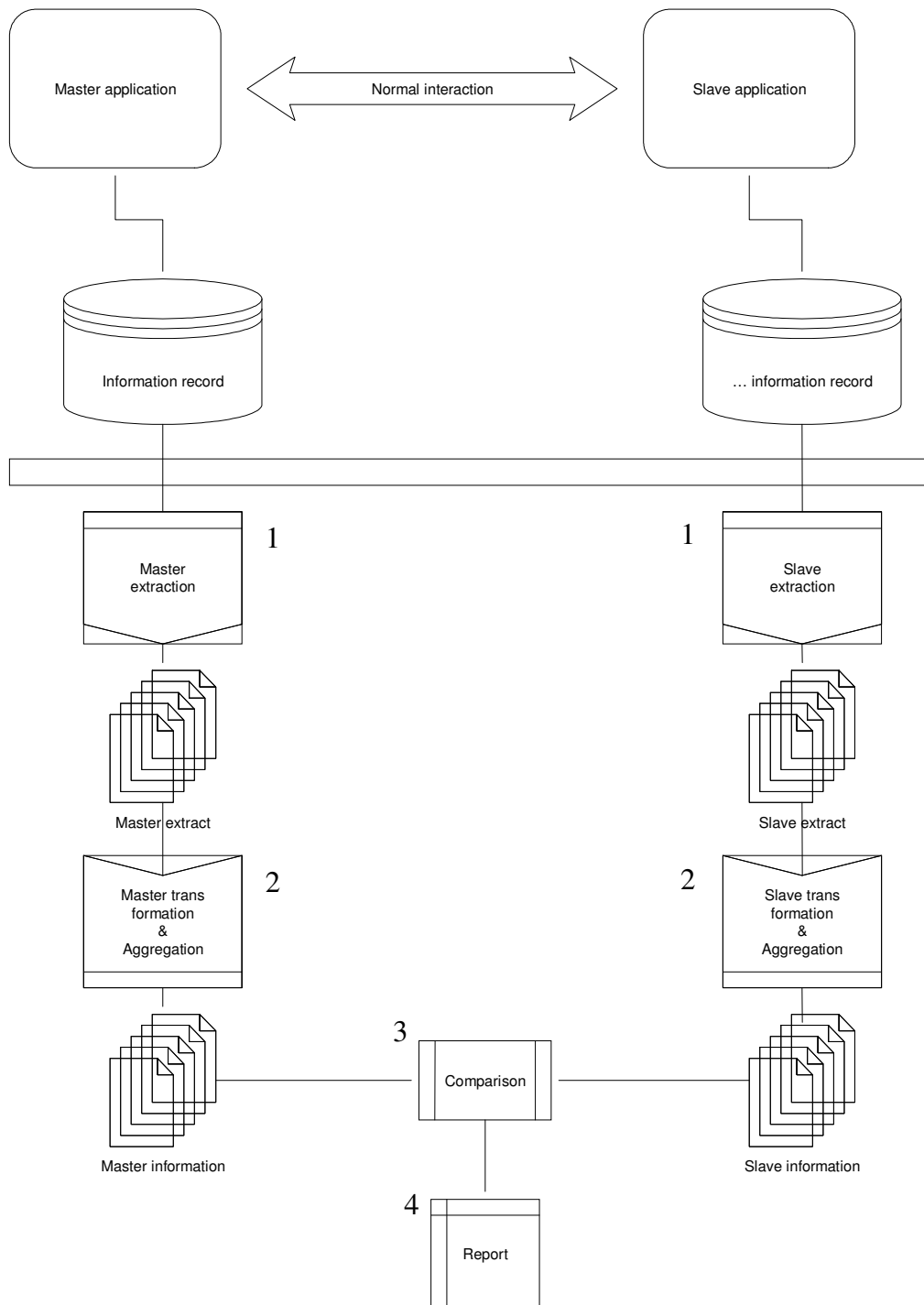
#### 1. Extractions

Everything starts with the extraction of the information out of the two applications (e.g. master and slave application). ONLY ESSENTIAL DATA defines **which data** to extract. PICK THE RIGHT MOMENT defines the moment **when** to extract the data.

This process must extract all necessary information to establish a relevant reconciliation:

- The appropriate list of occurrences
- The appropriate set of descriptive information concerning these occurrences or related other objects.

In some cases there's no explicit extraction of the application needed. Consequently, this will reduce the cost of the reconciliation process. For example SYNERGY BY REUSE or, in the case of a master and slave construction, when the interaction between master and slave already contains an information flow with the relevant information. For example, in the case of current accounts or client accounts the statements forwarded by the financial institution are also a basis for reconciliation.



**Figure 1: reconciliation process for master slave construct**

**2. Transformation and aggregation**

To make the data from the two applications comparable, transform these data to a common information model.

The information you extract can be delivered on several levels of detail. As it is not always necessary to have the most detailed view of the information, also choose during this phase the level of detail you need or want for the reconciliation.

**3. Comparison**

Finally you have two information sets in a common information model and with the same level of detail. Compare both information sets and detect the inconsistencies.

Next to SYNERGY BY REUSE, also REUSE FROM ORIGINAL INTERACTION, SYNERGY IN DESIGN and/or TRANSITIVITY OF BEING RECONCILED can be used to reduce the costs of the reconciliation process.

#### 4. Reporting

The detected inconsistencies can be reported in various ways to the ICT and/or Business responsible. NEUTRAL INFORMATION ON THE RIGHT LEVEL OF DETAIL defines what to report. The specifications – and thus the complexity – will depend on technical and functional requirements:

- Functional
  - Is a drill down to base information out of the reported inconsistencies needed?
  - Is a workflow to allocate inconsistencies to specific roles needed?
  - Should there be support for the error tracking and handling processes?
  - ...
- Technical
  - What's the typical amount of inconsistencies that will be reported?
  - What's the volume of base information that should be presented together with the inconsistencies?
  - How heterogeneous is the current technical environment of the user workplace in which the report should be presented?
  - ...

### **Consequences**

With this kind of reconciliation process, not all the possible types of errors can be detected. For example, you will not be able to detect errors due to human errors in functional design or specifications with this kind of reconciliation process where you compare two data sets.

There are several opportunities while developing the reconciliation process to reduce the implementation and exploitation costs.

### **Example**

Master and slave extraction: If General Ledger wants to reconcile with a loans product factory we must extract all financial accounts affected by loans and their balances. The loan product factory makes an inventory picture (selects all loans + amounts + all information which is necessary to identify the GL accounts for the amounts).

Master and slave transformation and aggregation: The GL balances and the inventory events have to be aggregated on the level of which the reconciliation takes place. Different levels of aggregation can be possible. The lowest aggregation level is the GL-accounting key (= accounting number + all dimensions)

Match GL balances and inventory: In this process-step the actual inventory control takes place. Lines with the same key-field (dependent on the level of control) will be matched against each other.

## **Pattern: Added value receiver must reconcile**

### **Context**

Information about one business reality is stored in different applications and/or data bases. A reconciliation process will be implemented to detect the possible inconsistencies between the two applications.

### **Problem**

**Who has the responsibility to implement a reconciliation process?**

### **Forces**

- Awareness of the need for reconciliation:
  - o One application (e.g. the master) can have the desire to have control over the quality of its data stored in the other application.
  - o An application wants its data to be correct
- Knowledge of the kind of process and logic that needs to be reconciled
- Cost allocation

### **Solution**

Assume that one of the applications (let's call it A) has no idea about the logic in the other application (say B). If there is no need from A's side to be reconciled with B, **allocate the responsibility to implement a reconciliation process to application B.**

When application B is constructed and the appropriate level of reconciliation is defined, the owner of application B should also initiate the implementation of a reconciliation process in co-operation with application A.

When there is a specific desire from one application (A) to be reconciled with one or more other applications, the cost allocation force will be decisive to determine who has the responsibility. If application A always has the responsibility for the reconciliation, all costs will be central, namely for application A. In some cases an application A can have several other applications containing datasets that represent the same information as in A, and A does not always have control over which applications these are. This happens for example in the case of a master application with multiple slaves. In that case, these costs can be very high for A if the responsibility of the reconciliation is always assigned to A.

This principle is independent of...

- the possible correctness of master and slave applications
- the factors that contribute to the business importance and the reconciliation responsibilities at business level
- the scheduling of jobs implementing the reconciliation

### **Consequences**

By placing responsibility with the application that desires to be reconciled, the costs for the reconciliation process are not centralized in one application, but locally distribute over all applications that desire to be reconciled.

## **Pattern: Only essential data**

### **Context**

Information about one business reality is stored in different applications and/or data bases. A reconciliation process will be implemented to detect the possible inconsistencies between the different applications.

### **Problem**

**Which data need to be extracted from the applications to perform the reconciliation?**

### **Forces**

- In the case of a master slave construct, several master-slave interaction models are possible. In a state-oriented interaction model, the slave application will be fed by forwarding the new state of an object. In an event-oriented interaction model, the slave application will be fed by forwarding the events that change the state of an object.
- There are several causes possible why the information stored in one application can differ from the one managed by the other application:
  - o Human errors in functional design or specifications
  - o Coding errors
  - o Human errors in manipulation of technical objects
  - o Hardware or system software failures that affect data
  - o ...
- Development and exploitation costs of applications need to be kept as low as possible.
- Obligation: Sometimes there is an explicit order to use reconciliation processes:
  - o At request of the company revisers
  - o At request of audit
  - o At request of the business or end users
  - o At request of marketing department
  - o ...
- Quality: The applications need to be as correct as possible. As many as possible of the inconsistencies between different applications managing the same business reality need to be resolved.
- Risk: An undetected inconsistency involves a risk and a possible damage.

### **Solution**

**Extract only the essential data necessary for the reconciliation.**

First, decide **which processes/data** you want to check for inconsistencies. Two things may influence this decision. Firstly, how critical is the data for the company or what is the business importance of each process or data element. For example: is there any obligation? what is the risk involved?

Secondly, in the case of a master and slave construct, the kind of interaction influences the decision too. For an event-oriented interaction model, the focus will be on the logic the slave uses to interpret this event. In that case, check mainly data/processes where the event has an effect on. In a state-oriented interaction where the new state of an object is pushed to the slave, no reconciliation is needed, because no errors are possible there.

Secondly, decide **which type of errors** you want to trace, keeping in mind the risk that an error involves and the type of error that can occur mostly.

And finally, check which data occurrences are needed to be able to reconcile, which data occurrences are available and which ones will need to be created. This process must extract all necessary information to establish a relevant reconciliation:

- The appropriate list of occurrences
- The appropriate set of descriptive information concerning these occurrences or related other objects.

## **Consequences**

As not all the data is extracted for the reconciliation, but only the essential data, the cost for the reconciliation process can be reduced.

However, sometimes there is no extra cost involved when you extract more information than needed. For example, when all information is stored in one data base, there is no (cost) benefit by extracting only some rows or columns. The extra cost will be caused when a join or merge necessary between several data bases.

## **Example**

Inventory control is the legal requirement to compare the inventory of the product factories with the relevant accounting positions. It will be this legal requirement that decides which information needs to be included in the reconciliation process.

## **Pattern: Pick the right moment!**

### **Context**

Information about one business reality is stored in different applications and/or data bases. A reconciliation process will be implemented to detect the possible inconsistencies between the different applications.

A similar state with respect to a point in time of the information records of both applications (for example: end of the month, end of the batch process) is needed as a basis for the reconciliation process.

### **Example**

A similar state in time seems trivial, but in practice a lot of factors can influence this:

- Information can be delivered from outside the company or can originate from different compartments or information environment-domains.
- Product factories sometimes deliver their 'end of the month'-state only after 3 work days.
- Information can originate from different time-zones (for example, when originated from a foreign branch).
- The 'last working day' of the month can differ from country to country, depending on official holidays.

### **Problem**

**When, at which moment do we need to extract the information out of the different applications?**

### **Forces**

- Functional timing aspects need to be taken into account. For example: closing of a financial year.
- Technical timing aspects need to be taken into account. For example: when will the information be available, how fresh is the information,...

### **Solution**

**Concentrate first on the functional timing aspects** that are business related and take care there is an agreement between master and slave on the functional timing:

- If you want to reconcile the state of an information record representing a real life object, define the moment in time of this state.
- If you want to reconcile a list of information records that originated in a certain time span, define this time span.

**Then, focus on the technical aspects** by answering following questions:

- When will all the necessary data be available?
- When will we be able to extract the information?

### **Consequences**

This can become very complex if an application needs to be reconciled with several other applications. The combination of all the individual timing constraints can lead to a complex set-up.

### **Example resolved**

For the examples given, first look at the business related requirement, for example: 'end of the month'-state is required for a specific object. Then concentrate on the technical aspects; this state will only be represented in the data after 3 work days or there is a time difference between Europe and Asia.

## **Pattern: Reuse from original interaction**

### **Context**

You have a master slave construct, where information about one business reality is stored in different applications and/or data bases. A reconciliation process will be implemented to detect the possible inconsistencies between master and slave.

The same transformations on the data are needed for the reconciliation process as in the original interaction components between master and slave.

### **Problem**

**Which of the original interaction components should we reuse for the reconciliation process?**

### **Forces**

- The quality of the reconciliation process needs to be as high as possible, this means reduce the FP (false positives) and the FN (false negatives) of the errors that are detected with the reconciliation process:
  - o The applications need to be as correct as possible. As many as possible of the inconsistencies between master and slave need to be resolved.
  - o An undetected inconsistency involves a risk and possible damage.
- By reusing components of the original interaction in the reconciliation process, we will not tackle all the possible errors and the correspondent risks of inconsistencies in that specific component we reuse.
- The development and exploitation costs of the reconciliation process needs to be kept as low as possible. By reusing components of the original interaction, the implementation cost of the reconciliation process can be reduced.
- Sometimes, the existing software, when it is a black box, cannot be rebuild and hence needs to be reused for the reconciliation, in order to transform the data records into a similar format to reconcile.

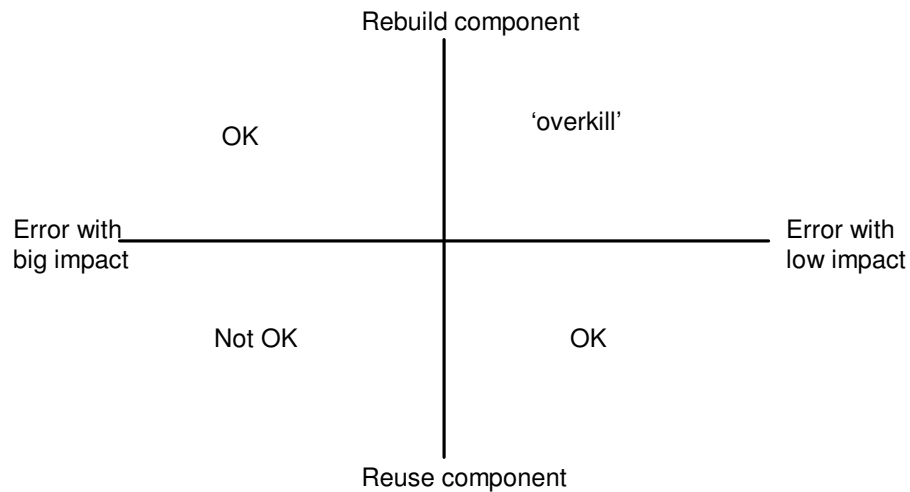
### **Solution**

Use a mix of original interaction components and newly developed components for the reconciliation process. Keep in mind what processes you want to control and which processes don't involve a risk or possible damage when not controlled. **Find the match between the errors you want to detect and the processes you are controlling with the reconciliation.**

In order to reduce the costs of the reconciliation process, reuse all components that do not deliver a great benefit when rebuilt for the reconciliation process.

As all the possible errors and the corresponding risks of inconsistencies of the components that are reused will not be tackled by the reconciliation process, don't reuse components where the risk of an undetected inconsistency and the possible damage is high.

Components that are black box need to be reused and the errors in these components will not be tackled by the reconciliation process.



### Consequences

- The components that are reused are not checked and the number of errors that can be detected is lower.
- By reusing components, the development cost is reduced.
- By not reusing components, all the intermediate logic is checked also.

### Example

For the reconciliation of the inventory of a product factory with the GL, reconciliation schemes are constructed (different from the schemes used in the operational environment) due to the fact that we want/need to check the correctness of the operational schemes.

## **Pattern: Synergy by reuse**

### **Context**

Information about one business reality is stored in different applications and/or data bases. A reconciliation process will be implemented to detect the possible inconsistencies between the different applications.

Several information flows can be in place with respect to these applications for various purposes.

### **Problem**

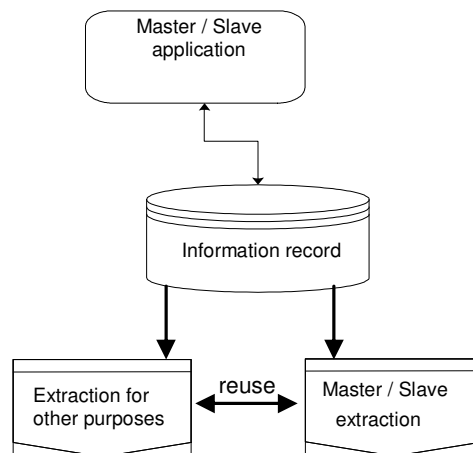
**Can the cost for the reconciliation process be reduced knowing there are already information flows in place?**

### **Forces**

- Reduce the cost for reconciliation as much as possible
- Reduce implementation work for the reconciliation process as much as possible
- The quality of the reconciliation process and in particular the quality of the extraction of information needs to be as high as possible.

### **Solution**

Before constructing the extraction of the information for the reconciliation process, **check if a similar information flow is or will be constructed for other purposes** different from the flows in the operational process between the different applications. If such information flows are available for one of the applications, these can be reused as the extraction in the reconciliation process.



### **Consequences**

The reconciliation cost can be a lot smaller if you grab synergy opportunities by reusing information flows. As you only reuse similar information flows, there will be no loss in quality.

By reusing an existing information flow as extraction for the reconciliation process, it is possible that more information will be extracted than necessary, which is in conflict with ONLY ESSENTIAL DATA. A filter that drops the not relevant information for the reconciliation can be a solution.

### **Example**

The extraction of all necessary information for the master in the example of the loan product factory and GL isn't solely used for reconciliation purposes only. It is the regular extraction towards the information layer (IL) of the corresponding domain. The extracted information will be stored as an inventory in the IL so that it can be reused for other purposes too, for example for the reconciliation process.

## **Pattern: Synergy in design**

### **Context**

An application needs to reconcile with several other applications.

### **Problem**

**Can the cost for the reconciliation processes be reduced with respect to the first application?**

### **Forces**

- Reduce the cost for reconciliation as much as possible
- Reduce implementation work for the reconciliation process as much as possible
- The quality of the different reconciliation processes need to be as high as possible.

### **Solution**

If an application has to reconcile with different other applications, **consider the synergy between all these implementations**. If these other applications are of the same nature, it's obvious that an infrastructural approach will be relevant. You can for example set up only one extraction process to feed the various reconciliations. Or you can set up the comparison process and/or report process common for all reconciliations.

### **Consequences**

The reconciliation cost can be a lot smaller if you grab such synergy opportunities. As the other applications are of the same nature, there will no loss in quality.

By setting up only one extraction process to feed the various reconciliations, it is possible that more information than needed will be extracted, which is in conflict with ONLY ESSENTIAL DATA. A separate filter that drops the not relevant information for each reconciliation can be a solution.

### **Example**

The extraction of all necessary information out of the GL for the several product factories isn't dedicated to reconciliation purposes only but is also the regular feed to IL Accounting. The extract is transformed to the information model 'GL Outstanding' which is semantically comparable to 'Account Outstanding'. GL outstanding gives the balance on an account and an 'account outstanding' is an outstanding agreement on an account, where an 'agreement' is a transaction (e.g. an option) or a position (e.g. current account). An agreement can have more than one outstanding agreement on accounts (e.g. the nominal amount, the accrued amount, the MTM value ...)

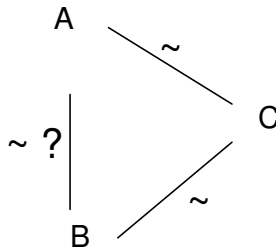
## Pattern: Transitivity of being reconciled

### Context

Application A and application B are reconciled with the application C. The information in applications A and B is comparable and there is the need to compare them.

### Problem

Is it necessary to implement a reconciliation process between applications A and B?



### Forces

- The quality of the reconciliation process and the exhaustivity of the comparison needs to be as high as possible:
  - o The applications need to be as correct as possible. As many as possible of the inconsistencies between two applications need to be resolved.
  - o An undetected inconsistency involves a risk and a possible damage.
- The cost reduction by using transitivity
- The work reduction by using transitivity

### Solution

Although applications A and B are reconciled with the same application C, inconsistencies between applications A and B can still occur because of a difference in the level of granularity of the both reconciliation processes with application C or because of the possible transformations that are made with the data in these applications. **If the relationship ‘being reconciled’ (denoted by ~) does not differ a lot from the relationship ‘being equal’ and the level of granularity of both reconciliation processes are similar, then we can state that A and B are also reconciled.** And no reconciliation implementation is needed between applications A and B.

If the level of detail of both reconciliations differs a lot, it would be wrong to state that A and B are reconciled. In this case, you can wonder if that not very detailed reconciliation really makes sense! And one could consider dropping the not very detailed reconciliation after implementing the reconciliation process between A and B.

### Consequences

By deducing that applications A and B are reconciled from the fact that A and B are both reconciled with the same application C, the cost can be reduced significantly as no extra reconciliation process needs to be implemented or executed.

As we only state that applications A and B can be seen as reconciled when the reconciliations with application C have the same level of granularity, the quality of the statement ‘being reconciled’ will not be affected much.

However, in order to have the same level of granularity and the same relevant data to be reconciled, probably the most and only interesting kind of transitivity will be on record or object level.

## **Pattern: Neutral information on the right level of detail**

### **Context**

Information about one business reality is stored in different applications and/or data bases. Often the coherence between these different records can be disrupted after replication or duplication of the data. A reconciliation process will be implemented to detect these inconsistencies.

### **Problem**

In case of the master slave construct, by only reporting the inconsistency, the user (business) will be inclined to see the master as 'true', but this can be wrong. The master is not by default right. **What should the reconciliation process report in order to enable the user to decide about right and wrong?**

### **Forces**

- The reports need to be useful for the business users, enough detail is necessary so they can decide for each inconsistency about who is right and who is wrong.
- The information in the reports needs to be correct:
  - o The reconciliation process can be a possible cause of reported inconsistencies; these are the false positives of the reconciliation process.
  - o In case of the master slave construct, although you might think that the master application will always be right for being the source of all information forwarded to the slave application, this is not always the case:
    - If the master and slave application interact by exchanging the new state of the master information records, then only the master can be right. After all, the slave hasn't got knowledge of the event or request who initiate the state change.
    - If the interaction is based on the propagation of events and requests then both master and slave translate these to interactions they individually apply to their information records. So both translation processes can contain mistakes.
    - If the interaction between master and slave is based on both events and states, then both translation processes can contain mistakes.

### **Solution**

Every inconsistency detected by the reconciliation process should be treated individually without making any assumption on the application being correct or wrong. **The reconciliation process only reports neutral information**, namely the inconsistencies between the information sources without stating who was right or wrong.

By a detailed investigation of the information records of the different applications and the information exchanged between them, the error causing the inconsistency should be detected. **The reconciliation process delivers information on the right level of detail to make it possible for the user (business) to trace the starting point of the error.** Providing the kind of interaction model used in the master-slave construction can be useful information to make the decision about correctness more easily for the user: the master will be right in a state-oriented interaction model.

The specifications for the report will depend on the functional requirements from the user:

- Is a drill down to base information out of the reported inconsistencies needed?
- Is a workflow to allocate inconsistencies to specific roles needed?
- Should there be support for the error tracking and handling processes?
- ...

## **Consequences**

Because the reconciliation process does not state who is right or wrong when reporting the inconsistencies, no wrong decisions about the correctness can be made by the reconciliation process. Still, enough information is delivered to the user (business) to treat each inconsistency individually and to be able to resolve the inconsistency.

However, if one would want to automate the reconciliation process and more particularly resolving the inconsistencies after the reconciliation, some educated guesses about who is right and who is wrong would be necessary. But resolving inconsistencies is out of scope for this pattern language.

## **Acknowledgements**

The authors would like to thank Arno Schmidmeier for his very useful remarks during the shepherding process. We also would like to thank several people from KBC for proposing this interesting case.

This paper has been written as part of the KBC-research chair on 'Managing efficiency aspects of software factory systems' sponsored by KBC Bank en Verzekeringen.