Data Integration Patterns

Alexander Schwinn, Joachim Schelp
Institute of Information Management, University of St. Gallen
{alexander.schwinn,joachim.schelp}@unisg.ch

Abstract

The application landscapes of major companies all have their own complex structure. Data has to be exchanged between or distributed to the various applications. In this paper different types of data integration are identified and categorized. Advantages and disadvantages as well as usage scenarios are discussed for each identified integration type. This paper also tries to answer the question "Where does redundancy make sense?", not "How to avoid redundancy?".

1. Heterogeneous Application Landscapes lead to Data Redundancy

The companies' evolving application landscapes are becoming more and more complex as long as no standardization takes place. Newer technologies introduced by electronic business increase the complexity. They require a higher degree of integration between intraorganizational applications than previous technologies, when the evolution of applications took place within departmental borders. Stovepipe application types were the result [Lint00, 4].

But the increased need for integration is not just a result of new requirements, induced by new businesses: Mergers and acquisitions result in similar requirements. Similar application systems have to be run for some time after a merger or acquisition has taken place [KrSt02]. But to dig up any synergy potential e.g. customer data has to be integrated or exchanged between these parallel running applications. Several integration concepts are currently discussed under the label of "Enterprise Application Integration" (EAI) [SchWin02, 12-17]. They can be reduced to data, function or event-oriented integration (e.g. [SMFS02, BFGH02]). In this paper the further discussion focuses on data integration. To analyze the different data-oriented integration types, redundancy should be considered. In this section, redundancy has to be discussed before further data integration types can be identified in the next section.

For this paper we define redundancy as storing the same data multiple times. This may cause problems when changes require the modification of stored data. All copies of the original data have to be modified as well to avoid inconsistencies causing problems with further data processing or prohibiting it completely. Consider, for example, changing customer address data: When the data is changed in the sales department only, tracking the invoice will be problematic in the accounting department etc. Consistently avoiding data redundancy is recommended in literature (e.g. [Dit99]).

But why should data redundancy be applied systematically and managed, when avoiding it is recommended? In some areas, e.g. data warehousing, data redundancy is required to increase query performance. Complex queries are not executed in the operational environment, but in a data warehouse holding copies of operational data, structured for analytical purposes [Inm96, Inm99]. Newer concepts like active warehousing still require data redundancy, but ask for a quicker propagation of changes. The vision of a real time enterprise demands current data (copies) to avoid any delays in the execution of business processes [DRFA02]. The concept of active warehousing seeks for near real-time updates of data warehouse data [Bro02]. These near real-time updates have to be done with an EAI infrastructure.

The following example illustrates in which cases avoiding data redundancy can result in drawbacks. Our fictitious telecommunications company has several business units offering different services. To support the business processes each unit has its own transactional systems. To avoid inconsistencies and to make data changes easier there is one central shared database for customer data which is used by all business units. Customer data can be created, changed and deleted centrally. Any customer data transaction has to be executed once only. This high level of data integration—every query for customer data runs against the central database—may result in some drawbacks:

- The availability of all components—departmental applications, central database, EAI infrastructure, network etc.— has to be ensured to allow operation. A failure in one component brings the whole system down.
- All components must have a high capacity. If, for example, a large number of queries are made from the

Witold Abramowicz, Gary Klein (eds.), Business Information Systems, Proceedings of BIS 2003, Colorado Springs, USA

internet portal application during nightly backups of the central database, internet customers are not willing to wait for long. Accordingly, the overall system capacity has to cover the combined load of all systems.

- Maintenance, further development, and tests become more complex because of the higher requirements concerning availability, capacity, performance, etc. Maintenance cycles of the individual systems have to be coordinated to avoid interference of operation.
- Splitting up the company and selling a business line is less difficult if there is a central database because the acquiring company has access to the selling company's data.

Another reason to stick to data redundancy is country-specific legislation. In Europe there are strong rules concerning changing and exchanging individual customer data. Complete customer data can be proliferated within a group of companies if the individual customer agrees to it [Bül02]. After mergers and acquisitions customers would have to agree to a group-wide usage of their data.

In reality, a high level of integration can be found within a wide range of companies. But [KrSt02] shows that in some 35% of the integration projects due to mergers and acquisitions, similar functional applications of the individual companies still run in parallel and are not standardized—e.g. to gain from specialization of the applications shaped for different businesses. Accordingly, data has to be exchanged and is stored several times, because these applications have individually developed databases, which are difficult to merge.

The next chapter discusses how data can be stored redundantly. Different types of data integration build the framework to identify data integration patterns. The patterns presented here reflect data-oriented integration only. A discussion of other integration types—e.g. process or object integration—can be found in [DLPR02], amongst others.

2. Data Integration Patterns

The following section gives an overview of different data integration types. Subsequently, the individual variants are presented in detail. Usage scenarios, as well as advantages and disadvantages are given for each of the solutions presented.

2.1 Overview

The classification of data access types presented here is based on the assumption that an application needs specific data at a given time, in a specific format and in a required quality. The classification points out how these requirements can be met. The classification is based on how the application gets the data (e.g. in which time intervals, kind of communication, etc.), and on whether the application accesses the original data source or a copy of the data. A goal of this approach is to model an application landscape in which temporal and technical dependencies as well as redundancy appearance can be discovered. This is especially important when implementing new applications or replacing old applications by new ones. Fig. 1 shows the identified data integration patterns:

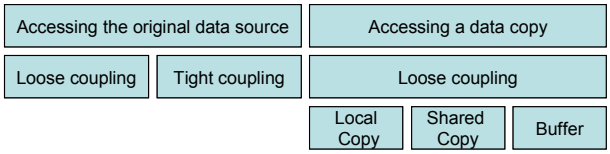

Figure 1. Types of data integration

The types identified here can also be understood in the sense of design patterns as they are known in the object-oriented world. A pattern describes a problem which recurs regularly in our environment, and the core of the solution for this problem, so that the solution can be reused at any time [AISJ77, 10].

Below redundancy-free patterns are presented (accessing the original data), differentiating between loose and tight coupled variants. Subsequently, alternatives for access to data copies are presented, differentiating between three different types of data copies: local data copies, shared data copies and buffers. When creating data copies, the time dimension is also considered, that is how quickly the data copy becomes available for the application, resp. how up to date the copy is.

2.2 Redundancy-free Solutions

In order to avoid redundancies, the original data source must be accessed. This is unproblematic and makes sense in some scenarios. Two different variants of how the original data can be accessed are shown here. The application can be accessed directly (tight coupling) or via a mediator (loose coupling). The different concepts and usage scenarios are presented below, with their respective advantages and disadvantages.

2.2.1 Direct Access

Accessing a data source directly is only possible under certain conditions. The direct call is usually made either by one of the data base management systems (DBMS) involved, or by means of an API (Application Programming Interface) call. A call initiated by the database management system can only be implemented if the system is accessible and the application is not a "Black Box". Direct access to database systems is often

impossible. Packaged applications usually provide an API for accessing their database systems. However, an API call only makes sense if it meets the exact requirements of the potential initiator, and the initiator application can be manipulated. An extension of APIs is not possible in most cases because the application code is not accessible. Usually this is only possible, if the software has been developed in-house. Table 1 shows usage scenarios and gives an overview of advantages and disadvantages of the direct access integration pattern:

Table 1. Usage scenarios, advantages and disadvantages of the direct data integration pattern

Direct Data Integration Pattern
Usage Scenarios
• Industry standards are used • Software was developed in-house (full access to source code is given)
Advantages
• Easy to implement, lowest complexity • No overheads
Disadvantages
• Software components often cannot be manipulated and accessed • Tight coupling (lower availability, difficult change management) • Locking-problems within complex transactions • Cross platform communication usually not possible or difficult to implement

2.2.2 Data Access via Mediator

If it is not possible or desired to access the original data directly, an application can access a source of original data through a mediator. The integration logic is transferred to the mediator, because no access to the source applications is granted—neither on the application nor on the database level. The mediator essentially takes on the following tasks:

- **Transformation:** A transformation can take place on a semantic level (mapping data contents, e.g. transformation of country codes or currency codes) and on a syntactic level (transformation of different data formats or message formats).

- **Routing:** The routing is responsible for the distribution of messages to the applications involved. Besides, mechanisms for buffering messages are provided by the routing component.

- **Composition/Decomposition:** The composition component merges several messages into one, the decomposition component divides a message into several.

- **Controlling:** The controlling component controls the chronology and resolves dependencies.

These basic functions also appear—partly implicitly—in other approaches - although there are various terms and delimitations (see [RieVog96, SMFS02] for details).

Table 2. Usage scenarios, advantages and disadvantages of data integration via mediator.

Data Integration via Mediator
Usage Scenarios
• Complex transformation, routing, composition/decomposition or controlling is required • Integration of packaged applications
Advantages
• No (code-)manipulation within the affected applications is necessary • Controlling complexity by encapsulation through the mediator
Disadvantages
• Mediator can become very complex • Increased overheads • An additional component in the application landscape must be operated/managed

Further general advantages and disadvantages of tight and loose coupling can be found in [RMB01, 20-21; Cum02, 48]. The coupling measures the level of interdependency between two components. It also measures the impact that changes in one component will have on the other. In loose coupling, the integration is dependent on some interfaces. Loosely coupled components have the advantage of fewer dependencies.

2.3 Redundancy Based Solutions

Compared with a redundancy-free solution, solutions which create redundancies always incur additional expenditure because the data must be synchronized. There is the danger of evolving inconsistencies, whereby the data quality suffers. Detailed information about data quality can be found in [Hel02]. However, for various reasons (see section 1) redundancy solutions are often implemented. We differentiate between three types of redundancy solutions: Local data copies, shared data copies and buffers. We do not make a distinction between integrations with or without a mediator because in theses redundancy based scenarios a mediator is almost always used. Complex transformations, routing, composition/decomposition and controlling of the data is usually needed. For usage scenarios, as well as

advantages and disadvantages of using mediators see section 2.2.2. In the literature (e.g. [AKVG01, 209-234]) a distinction is made between data copies on the application level and data copies on the database level. This distinction is not considered in this paper because it is not relevant for the recognition of redundancy and the illustration of redundancy relationships.

2.3.1 Local Data Copy

Local data copies exist whenever an application keeps the data copy locally, i.e. the necessary data is supplied by a central database and it is stored by the application locally. The application, which keeps the copy, always works with the copy and not with the original data. The example of the Telco in section 1 is a typical local data copy scenario. If the autonomy of the application, which needs data from a central database, has to be ensured, a local data copy must be used. Thus the application can work autonomously and is not affected by master database failures. Besides, a local data copy is usually more efficient and can satisfy requirements concerning transaction processing. The following table summarizes advantages, disadvantages and usage scenarios of the local data copy pattern.

Table 3. Usage scenarios, advantages and disadvantages of the local data copy pattern

Local Data Copy Pattern
Usage Scenarios
• High availability necessary • Autonomy of application desired • Examples: Data warehouse (DWH), channel and sales applications, packaged applications
Advantages
• High performance • High availability • Stand-alone solution • Single-source for analyzing data (optimal data view for analytic system/analyses; no distributed queries over several data sources necessary)
Disadvantages
• Lower data timeliness (depends on refresh period) • More overheads because the data has to be synchronized to avoid inconsistency • Costs for redundancy (memory, management of redundancy, change management) • Inconsistencies may arise

2.3.2 Shared Data Copy

The shared data copy is a data source which is used by several applications. The shared data copy usually contains data from several sources, whereby data from heterogeneous databases is merged. A typical example of a shared data copy would be an operational data store (ODS), which integrates operational data from multiple sources. Different applications can access this data. The access is transparent for the application, e.g., it only accesses the ODS and does not know the origin of the data. In comparison with the local data copy, the shared data copy results in fewer redundancies because it is not necessary to duplicate the whole database for each application, but only the needed data. A further example for using the shared data copy is the coupling of computing centers (middle to large geographical distance), so that the application logic can run distributed in several computing centers. This scenario also increases the reuse of application logic because client access is transparent. As most standard software packages do not provide direct access to their database systems it is not always possible to create a shared data copy. Table 4 summarizes advantages, disadvantages and usage scenarios of the shared data copy pattern.

Table 4. Usage scenarios, advantages and disadvantages of the local data copy pattern

Shared Data Copy Pattern
Usage Scenarios
• Distributed computing centers (middle to large geographical distance) • Merge data from several heterogeneous database systems • Example: Operational data store (ODS)
Advantages
• Transparent access for clients • Loose coupling between distributed computing centers • Higher degree of application logic reuse • Fewer redundancies compared with the local data copy
Disadvantages
• Data replication is not always possible within packaged applications

2.3.3 Buffer

Buffers are generally used for processing optimization. A typical example in the financial services sector is printing account statements according to their deadlines. If you printed all account statements on the deadline without using a buffer, the operational systems would not be able to handle the amount of data without loosing performance. Another example is printing call detail records (CDR) on phone bills where huge amounts of data is required from the operational systems. Table 5 gives a

summary of usage scenarios, advantages and disadvantages of the buffer pattern.

Table 5. Usage scenarios, advantages and disadvantages of the buffer pattern

Buffer Pattern
Usage Scenarios
• Processing optimization • Batch run/deadline-oriented processing • Example: Account statement/phone bill printing
Advantages
• High performance • Processing can be done separately from the operational system • Operational system load is decreased
Disadvantages
• Dependencies have to be considered (between the buffer and the operational system) • Not a redundancy-free solution

2.3.4 Creation Time of the Data Copy

If a redundant solution is selected for data integration the time at which the copy is created (e.g. how up-to-date is the data in my ODS) is crucial. Three creation time categories have been identified: Unknown/manual, periodic and near real-time copies. These three categories will be presented in the following. Again, advantages, disadvantages and usage scenarios of the individual solutions are presented.

Unknown Creation Time/Manual Creation

The simplest form of data copy creation time, which is usually neglected, is the manual one. The copy is created manually at an unspecified point of time (for example by user input). Manual data integration makes sense whenever the integration cannot take place automatically, i.e. if no implementation is available or an implementation would not be cost-effective. This is only be appropriate in those cases where data changes take place very rarely, and the amount of data to be integrated, is relatively small. Typical examples are changes of country codes or language codes. If a country code changes it has to be updated manually within the appropriate applications. The only advantage of this solution is its cost-effectiveness. No integration tools (e.g. a mediator) needs to be purchased or developed. The disadvantages of a manual solution are the danger of data inconsistencies, and a higher error rate. The data timeliness depends on the employee responsible for the data integration. Inconsistencies can arise if, for example, some systems have a new data copy and others do not. Referring to our example, an application would not recognize a language code any more. Finally, the fault rate is higher because manual data input can result in errors (see table 6).

Table 6. Usage scenarios, advantages and disadvantages of manual data integration pattern

Manual Data Copy Pattern
Usage Scenarios
• No implementation is available and an implementation is not cost-effective • Small data sets with a static character must be integrated • Lack of standardization (e.g. language codes, country codes)
Advantages
• Cost effective solution (because implementation costs are low)
Disadvantages
• Data timeliness depends on the discipline of the person responsible • Data administration dialogues/functions are necessary • Danger of inconsistencies • Error-prone (e.g. typing errors)

Periodic Data Integration

In the periodic integration scenario the data gets integrated in predefined periods (e.g. once a day, once an hour, etc.). A typical usage scenario of the periodic data copy is batch processing, which is activated by a scheduler. The periodic data integration makes sense whenever the data timeliness requirements are not very high and large data sets have to be integrated. In this case, a near real-time integration could overload the operational systems, so that the availability of the operational systems is reduced. By applying periodic data integration, dates can be selected where the system load is low. Furthermore, the periodic data integration has the advantage that the system can usually be debugged more easily and rollbacks are possible. Table 7 summarized usage scenarios as well as advantages and disadvantages of the periodic data integration pattern.

Table 7. Usage scenarios, advantages and disadvantages of periodic data integration

Periodic Data Integration Pattern
Usage Scenarios
• Batch processing (account statement/phone bill shipping) • Load processes in data warehouses (DWH) • Data timeliness requirements are low • Deadline-oriented processing • Large amount of data has to be integrated (bulk updates, initial loads)
Advantages
• High throughput • Reduces operational system load (data integration is done when system load is low) • Easy rollback possible, if errors occur
Disadvantages
• Data timeliness is lower compared to near real-time integration • Long processing times, because of large amounts of data, which have to be integrated • Higher availability of operational systems (integration is done when operational system load is low)

Near Real-Time Data Integration

The near real-time integration is the most difficult pattern to implement. This kind of integration is, however, the only one which guarantees a high data timeliness. It is used whenever up-to-date data is required (e.g. getting cash at an ATM). If near real-time integration is applied, very efficient systems are necessary which must process the load even at peak times. This implies high costs for powerful systems. A further disadvantage of near real-time integration is the fact that the data is always integrated immediately after creation and not only when it is needed. The main characteristics of near real-time integration are summarized in table 8.

Table 8. Usage scenarios, advantages and disadvantages of near real-time data integration pattern

Near Real-Time Data Integration
Usage Scenarios
• High data timeliness is required (e.g. ATM)
Advantages
• Data timeliness is high
Disadvantages
• Performance of the system is impaired because data is integrated not only when it is needed, but always immediately after creation • Expensive

The following matrix gives a rough overview of when which type of data integration pattern (manually, periodically or near real-time) should be used. It can support decision making when new data integration requirements arise.

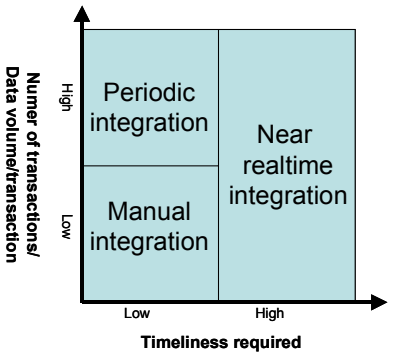

Figure 2. Usage criteria for data integration patterns

On the one hand the data timeliness requirements are differentiated, on the other hand the number of expected transactions, and/or the expected volume of data per transaction are distinguished. If the data timeliness requirements are high only near real-time integration is appropriate. If the data timeliness requirements are rather small and the data sets which have to be integrated are large, the periodic variant can be selected. At low data timeliness requirements and few transactions the manual integration type should be considered. Due to the disadvantages presented above, the manual type should, however, be generally avoided.

3. Conclusions and Further Research

The data integration patterns identified in the previous section are helpful when integrating application systems via data integration. The presented advantages and disadvantages, and the usage scenarios were identified in a research project with a Swiss IT solution provider who develops and runs banking applications. The patterns were tested in two major companies in the Swiss financial sector. The same patterns were found within their application landscapes. The next step will be to test these patterns in a wider environment to verify their reliability and robustness.

To identify data dependencies and to conduct further investigations, a consistent methodology for the description of application landscapes has to be developed. It would be easier to consider data dependencies at a time when applications have to be replaced and/or additional applications have to be implemented. Such a methodology is currently being developed in a research project at the Institute of Information Management of the University of

St. Gallen. The findings presented here result from one of the first steps in this research project.

4. References

[AKVG01] Adams, J.; Koushik, S.; Vasudeva, G; Galambos, G.: Pattern for e-business- A Strategy for Reuse, IBM Press, Double Oak 2001.

[AISJ77] Alexander, C.; Ishikawa, S.; Silverstein, M.; Jacobson, M.; Fiksdahl-King, I.; Angel, S.: A Pattern Language, Oxford University Press, New York 1977.

[Bro02] Brobst, S. A.: Enterprise Application Integration and Active Data Warehousing, in: von Maur, E.; Winter, R.: Vom Data Warehouse zum Corporate Knowledge Center, Proceedings of Data Warehousing 2002, Physica, Berlin et al. 2002, p. 15-22.

[BFGH02] Bunjes, B.; Friebe, J.; Götze, R.; Harren, A.: Integration von Daten, Anwendungen und Prozessen am Beispiel des Telekommunikationsunternehmens EWE TEL, in: Wirtschaftsinformatik, Vol. 44., No. 5, p. 415-423.

[Büll02] Büllesbach, A.: Datenschutz bei Data Warehouses und Data Mining, in: von Maur, E.; Winter, R.: Vom Data Warehouse zum Corporate Knowledge Center, Proceedings of Data Warehousing 2002, Physica, Berlin et al. 2002, p.1-13.

[Cum02] Cummins, F. A.: Enterprise Integration – An Architecture for Enterprise Application and System Integration, OMG Press, 2002.

[DLPR02] Dangelmaier, W.; Lessing, H.; Pape, U.; Rüther, M.: Klassifikation von EAI-Systemen, in: HMD – Praxis der Wirtschaftinformatik, Vol. 39, No. 225, 2002, p. 61-71.

[Dit99] Dittrich, K.: Datenbanksysteme, in: Rechenberger, P.; Pomberger, G.: Informatik-Handbuch, 2nd Ed., Hanser, München, Wien 1999, p. 875-908.

[DRFA02] Drobik, A.; Raskino, M.; Flint, D.; Austin T.; MacDonald, N.; McGee, K.: The Gartner Definition of Real-Time Enterprise, Note Number COM-18-3057, Gartner Group, 2002.

[Hel02] Helfert, M.: Planung und Messung von Datenqualität in Data-Warehouse-Systemen. St. Gallen, Dissertation 2002, No. 2648, Difo-Druck, Bamberg 2002.

[Inm96] Inmon, W. H.: Building the Data Warehouse, 2nd ed., John Wiley, New York et al. 1996.

[Inm99] Inmon, W. H.: Building the Operational Data Store, 2nd ed., John Wiley, New York et al. 1999.

[KrSt02] Kromer, G.; Stucky, W.: Die Integration von Informationsverarbeitungsressourcen im Rahmen von Mergers & Acquisitions, in: Wirtschaftsinformatik, Vol. 44 , No. 6, 2002, p. 523-533.

[Lint00] Linthicum, D. S.: Enterprise Application Integration, Addison-Wesley, Harlow et al. 2000.

[RMB01] Ruh, W. A.; Maginnis, F. X.; Brown, W. J.: Enterprise Application Integration, Wiley & Sons, Inc., 2001.

[RieVog96] Riehm, R.; Vogler, P.: Middleware – Infrastruktur für die Integration, in: Österle, H; Riehm, R.; Vogler, P.: Middleware – Grundlagen, Produkte und Anwendungsbeispiele für die Integration heterogener Welten, Vieweg, Braunschweig et al. 1996, p. 25-135.

[SchWin02] Schelp, J.; Winter, R.: Enterprise Portals und Enterprise Application Integration – Begriffsbestimmung und Integrationskonzeptionen, in: HMD-Praxis der Wirtschaftsinformatik, Vol. 39, No. 225, 2002, p. 6-20.

[SMFS02] Schissler, M.; Mantel, S; Ferstl, O. K.; Sinz, E. J.: Kopplungsarchitekturen zur überbetrieblichen Integration von Anwendungssystemen und ihre Realisierung mit SAP R/3, in: Wirtschaftsinformatik, Vol. 44, No. 5, 2002, p. 459-468.