Abstract
This paper describes how object-oriented concepts can be used throughout system development for integration purposes. Based on the distinction of physical and conceptual integration the concept of object wrapping is discussed for the integration of non-object-oriented systems. By regarding applications as high-level objects, i.e. wrapped applications, integration is achieved by modelling so-called integration relationships between these wrapped applications. While in conceptual integration redundancy and message passing relationships are specified, in physical integration message passing relationships only are implemented. The paper finally outlines the activities of the conceptual integration process.
The emergence of the Common Object Request Broker Architecture [OMG 92] promises to simplify the integration of technically heterogeneous object-oriented applications. Furthermore, the concept of object wrappers is applied to technically integrate non-object-oriented legacy systems and commercial software packages. But integration is not only a technical problem. Technical integration must be viewed as an enabling condition for optimal support of the data processing requirements of the business. Business requirements, not information technology, determine why and how applications must be integrated. These business requirements must be understood and modelled in the analysis and design phase.

We will show how the concept of object wrapping can be used throughout system development for integration purposes. Based on the distinction of physical and conceptual integration in section 2, section 3 briefly describes object wrappers. Section 4 discusses how the same concepts can be applied in the analysis and design phase. Furthermore, applications are regarded as high-level objects, i.e. wrapped applications. Integration is then achieved by modelling so-called integration relationships between these wrapped applications. Then, section 5 outlines the activities necessary for conceptual integration based on wrapped applications. Finally, section 6 gives an overview of our further research.

2 Physical and conceptual integration

By integration we refer to a process (or a result) whereby mutually complementary parts are combined to create a comprehensive whole, i.e. applications are linked to an integrated information system. Integration relates to linking applications, often residing on heterogeneous computer systems. Therefore, interoperability between software components of different applications is required. Today, research and industrial attention is mainly applied to overcome different hardware platforms, programming languages, database systems, networking protocols, etc. Throughout the paper we call this area physical or system-level integration. Physical integration has to cope with system heterogeneity issues of the underlying hard- and software platforms.

Physical integration is not by itself sufficient to successfully integrate multiple applications. Rather business requirements, not information technology, determine which applications are to be integrated with each other. Physical integration is only a prerequisite for optimal support of the data processing requirements of an enterprise. These business requirements are compiled in the analysis and design phase. We refer to this process as conceptual integration. In conceptual integration we must deal with the issues of meaning and use of data as used by different applications, with multiple (possibly changing) interpretations of data by different users in different contexts, data inconsistencies and incomplete information, i.e. semantic heterogeneity issues [Sheth 91].

During the analysis phase, logical connections between applications are identified. We call any logical connection between different applications integration relationship (note: throughout the paper, we consider only data and functionality which is part of an integration relationship). Integration relationships are then specified in detail in the design phase. Therefore, conceptual integration refers to the identification and system independent specification of integration relationships between different applications. These resulting prescriptive
statements are then implemented in the coding phase by means of physical integration (see also the summary in figure 2).

3 Object wrappers for physical integration

In object-orientation each object is autonomous and encapsulates both structure and behaviour. Objects are loosely coupled components which communicate by message passing. When integrating two object-oriented applications the interaction between them is achieved by message passing between objects of these applications, i.e. the same modelling construct is used as for integrating objects within one application.

But organisations do not start from a clean slate. A lot of non-object-oriented legacy systems and commercial software packages already exist and must be integrated with each other. These applications provide a special integration challenge since they are often closed and do not provide an adequate API. Nevertheless, we must build on these existing investments because there are not enough development resources for a complete object-oriented redevelopment of the whole information system [Dietrich 89, Graham 94, Jacobson 91]. The existing applications must be integrated and gradually migrated to an object-oriented environment. The concept of object wrapping is used for the following purposes [Dietrich 89, Graham 94, Jacobson 91, Katz 93, Mowbray 95, PacBell 95, Winsberg 95]: Unification of legacy applications under a single graphical interface, integration of object-oriented applications with non-object-oriented ones, re-engineering of legacy systems with objects, gradual migration to an object-oriented architecture and data migration.

An object wrapper encapsulates a conventional piece of code and the data accessed by it. It enables a new, object-oriented part of an application to interact with the wrapped code by message passing [Graham 94]. I.e., object wrappers are pseudo-objects which serve as an interoperability bridge between non-object-oriented and object-oriented systems. They are well suited for the integration of non-object-oriented systems for several reasons [Mowbray 95]:

- Object wrapping allows for information hiding.
- Object wrappers present a clean API that provides abstract services.
- Object wrappers isolate implementation details, such as programming languages and operating systems, and support heterogeneity because the messages passed among object wrappers depend only on their interfaces, not on the internals of the object wrappers themselves.
- Object wrappers provide technology migration paths for legacy systems, allowing component upgrade without affecting the rest of the application.
- Object wrappers permit the reuse of legacy code.

Object wrapping involves different implementation techniques depending on the accessible elements of the application. Examples are wrapping with RPC, files or sockets [Mowbray 95].

Wrapping legacy systems allows to physically integrate basically any application by means of message passing. The only kind of integration relationship which must be modelled and implemented on the physical integration level is the so-called message passing relationship.
4 Applying object-oriented concepts to conceptual integration

The results of analysis, design and coding differ mainly in their levels of abstraction. From analysis to design and coding, objects are specified in more detail and new objects are introduced which reflect lower levels of implementation detail.

4.1 Objects and object wrappers

For the above reason, integration of object-oriented applications is achieved by modelling message passing relationships between objects of different applications throughout the development process. Integrating non-object-oriented applications using the concept of object wrapping in all development phases allows to model integration by means of message passing relationships, too.

Object wrapping on the physical level encapsulates code and thereby hides physical implementation details. Therefore, object wrapping at this level can be used to overcome system heterogeneity issues. In contrast, object wrapping on the conceptual level must specify which data must be exchanged and what the meaning of this data is. Thereby, we have to cope with semantic heterogeneity issues and an object wrapper encapsulates the meaning of the data and transforms it for message passing. Conceptual integration involves two basic semantic issues [Sheth 91]:

• Determine if and how two or more (data) objects are related. Furthermore we must identify possibly different interpretations of the same stored (data) object accessed by different applications.

• Resolve possible semantic heterogeneity, identify semantic discrepancy and support multiple interpretations of the same stored (data) object.

After determining which data must be exchanged between applications, we must define the functionality responsible for message passing. This functionality is encapsulated as a method in an object or object wrapper respectively. Methods must translate internal data and processing of each application to the conventions established for message passing. The data exchanged in a message is ideally encapsulated in business object documents. Each business object document covers information about the sending as well as the receiving application, the requirements on the method called and the data to be processed [Killer 95]. All these decisions are modelled in message passing relationships which must be implemented in physical integration.

To allow for significantly loose binding between applications information about, among others, the sequence of data items, the size of each item, the type of each item, and its semantic value must appear in the business object document [Sims 96].

4.2 Wrapped applications

Applications can be regarded as high-level objects [Burkart 92]. Originally we intended to call such a high-level object "application object". But this term is used by the OMG for objects (not applications) which are specific to particular industries or end-user applications and are not subject to standardization [OMG 92]. Therefore, we use the term (conceptually) wrapped applications.

A wrapped application receives and sends messages and, on receipt, delegates responsibilities to objects
or object wrappers which it encapsulates. Therefore it is different from the notion of the application wrapper used by [Winsberg 95]. An application wrapper "surrounds a complete legacy system, both code and data, by emulating an end-user sitting at a character terminal [only]."

Integration of applications in our understanding means defining a set of external message passing relationships for each wrapped application and the necessary functionality by means of methods of objects or object wrappers. By default, the interface of a wrapped application is the union of all its methods involved in message passing relationships.

Applying the concepts of wrapped applications and business object documents for integration has the following advantages:

- Each wrapped application is responsible for the management and for the meaning of its own data. It is a loosely coupled, autonomous component of the information system which can be changed, enhanced or replaced independently of other wrapped applications because its data and functionality is encapsulated.

- In analogy to EDI business object documents for message passing will (hopefully) be standardised in the long run. The vision of an enterprise-wide data or object model with standardised semantics and formats of all (data) objects is replaced by a message passing relationship model. The semantics and formats of (data) objects within an application may reside application specific and will be transformed to the standard semantics and formats for message passing only (see figure 1). Standardising objects for new object-oriented applications is an extraordinary challenging task but seems to be impossible for legacy systems. Therefore, we suggest to concentrate on standard semantics for message passing. Applying this concept rigorously is a more promising attempt to integrate legacy systems and currently available commercial software packages with minimum effort.

4.3 Dealing with redundancy

The concepts applied so far suggest that the message passing relationship is the only logical connection (i.e. integration relationship) between

---

**Figure 1: Wrapped Applications and Business Object Documents**
applications. The existence of redundant data has been neglected. Apart from more technical reasons for storing data replications like increased data availability, improved performance and fault tolerance other important reasons for the existence of redundant data are:

• Applications have been independently evolved during different time frames by different people. Decentralised application development often increases the problem of avoiding redundant data. Especially when dealing with legacy systems and commercial software packages, we must cope with redundant data residing in closed applications.

• The modelling effort for a sufficiently detailed and integrated enterprise-wide model is too time-consuming, expensive and complex. Furthermore, modern organisations deal with increasing complexity while remaining responsive to fast business change. Investment in existing systems is vast and cannot be abandoned overnight.

• For the same (data) object each user brings his or her own perspective, problems and requirements. These perspectives may often conflict with each other and can be considered in one model only with considerable effort. Therefore, different types of applications have different requirements (e.g. detailed data for day-to-day business vs. summary data in executive information systems and decision-support systems in order to identify trends, challenges and opportunities).

For these reasons we believe that redundancy must be dealt with even in pure object-oriented applications. Therefore, two types of integration relationships exist when integrating wrapped applications: *message passing relationships* and *redundancy relationships*.

We use the term redundancy to refer to the situation where the same fact is recorded more than once. In other words, data is redundant if it is identical with other data or can be derived from other data. For redundant data we define which data is the original and which is the copy. The original may be altered and the copy has to adopt these alterations. For this purpose we specify a *redundancy relationship*. Each redundancy relationship relates two (data) objects to each other and must define (among other things) [Gassner 96]:

• **Original and copy**
  In general, there should be only one original and only alterations to the original are transmitted to the copy. Between the original and the copy exists a coordinate-subordinate relationship. In case the distinction between the original and the copy is not possible, both data objects share equal status (i.e. they are peers) and alterations to the original and the copy must be passed on to the other data. This situation is represented by treating each (data) object as original as well as copy. Therefore, we define two redundancy relationships and rules to avoid and/or resolve conflicting values in the original and the copy.

• **Coherency condition**
  The central requirement for the copy is that it must be up-to-date, i.e. there must be consistency between the original and the copy. What deviations between the original and the copy are permitted is defined by using coherency conditions [Alonso 88]. Examples for coherency conditions are [Barbará 90, Sheth 90, Wiederhold 90]:

- Time intervals (e.g. the redundant copies will be made consistent every hour).

- Time points (e.g. the redundant copies will be made consistent at 6 p.m. every day).

- Version Condition: This condition specifies a window of allowable values in terms of versions.

- Event Condition: This condition specifies an event where the copy must be consistent (e.g. the copy must be made consistent before running a specified logical transaction).

  • **Coherency responsibility**
    The coherency responsibility determines whether the original or the copy in a redundancy relationship is responsible that the deviations between original and copy do not violate the coherency condition. If the original is responsible we use the notification principle (the original notifies the copy), otherwise the request principle (the copy requests the original whether the coherency condition is violated).

  • **Resolution of schema conflicts**
    As each wrapped application uses its own database, schema conflicts may result. These schema conflicts must be resolved in the redundancy relationship whereas the schemata of the wrapped applications usually remain unchanged. The two basic causes of schema conflicts are the use of different structures for the same information (e.g. storing the address of a customer in one attribute, in several attributes or in an own address-object) and the use of different specifications for the same structure (e.g. different names, data types or constraints) [Kim 91].

  • **Selection-/transformation condition**
    The selection condition defines which instances of a (data) object are redundant. The transformation condition establishes how the semantics of an original are translated into the semantics of a copy [Sciore 94] and thereby handles the semantic heterogeneity of redundant data. Simple examples of transformation conditions are recoding (e.g. changing the currency code "13" into the currency code "US$") and calculation (e.g. calculating the revenue for a period from the sales).

    In case a coherency condition is violated the copy must be updated by message passing. Therefore, we must specify a message passing relationship for each redundancy relationship. Redundancy relationships influence the specification of the corresponding message passing relationships in the following ways [Gassner 96]:

    • The coherency condition determines when the copy must be updated and therefore when the message must be passed.

    • The coherency responsibility determines which wrapped application must initiate the message passing to update the copy.

    • Schema conflicts must be resolved. This is achieved by transformation of the application specific format (schema) to/from the standard format of business object documents.

    • The selection-/transformation conditions must be applied for message passing too.

4.4 **Summary**

Even when applying object-oriented concepts for application integration on a conceptual level, we
must deal with redundancy. This means that on the conceptual level the principle of information hiding must be violated in order to find redundant data and specify redundancy relationships. The principle of encapsulation must not be violated and the functionality responsible for any data exchange by message passing must be encapsulated either in objects or object wrappers within wrapped applications.

A redundancy relationship is a logical connection between two pieces of data, whereas a message passing relationship is a logical connection between two pieces of functionality (methods of objects or object wrappers) defining which business object documents are exchanged.

When integrating applications we must identify redundancy and message passing relationships. For each redundancy relationship a corresponding message passing relationship must be derived. Then, these relationships are specified in detail and, finally, the message passing relationships must be implemented. Therefore, in conceptual integration we have to cope with redundancy and message passing relationships, whereas physical integration deals with message passing relationships only (see figure 2). In conceptual integration we must define what and how semantic heterogeneity must be overcome; in physical integration we implement how semantic and system-level heterogeneity is overcome.

5 Conceptual integration process

Based on the above insights, we outline the most important activities necessary to integrate wrapped applications. The integration process model shown in figure 3 is generic in the sense that its activity must be carried out independent of whether object-oriented and/or non-object-oriented applications are to be integrated with each other. Therefore, these activities can be incorporated in any object-oriented and structured development and re-engineering method to cover conceptual integration.

<table>
<thead>
<tr>
<th>Integration level</th>
<th>Conceptual Integration</th>
<th>Physical integration</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Purpose</strong></td>
<td>identify business requirements for integration and overcome semantic heterogeneity</td>
<td>overcome system-level heterogeneity</td>
</tr>
<tr>
<td><strong>Development phase</strong></td>
<td>Analysis</td>
<td>Design</td>
</tr>
<tr>
<td><strong>Task</strong></td>
<td>identification of integration relationships</td>
<td>system independent prescriptive definition of integration relationships</td>
</tr>
<tr>
<td><strong>Relevant redundancy relationships</strong></td>
<td>redundancy relationships</td>
<td>redundancy relationships</td>
</tr>
<tr>
<td></td>
<td>message passing relationships</td>
<td>message passing relationships</td>
</tr>
</tbody>
</table>

*Figure 2: Summary of conceptual and physical integration*
The integration process model distinguishes the following phases:

- **Analysis**
  In the integration context, the analysis phase calls for examination and study of the existing state of affairs concerning integration in a given business area of the enterprise. Based on the analysis of the objectives of integration, the scope of integration is defined, i.e. all wrapped applications which are to be integrated are identified. Then existing and planned integration relationships must be identified and documented. When dealing with existing applications reverse engineering may be necessary to find these integration relationships (note: the integration process model currently does not cover any activity for reverse engineering, finding redundant data etc.).

- **Design**
  The wrapped applications and integration relationships are analysed of whether they are appropriate to fulfil the integration objectives. A re-

```
Identify applications which are to be integrated (and wrapped)

Identify integration relationships

Analyze wrapped applications and integration relationships and possibly re-cluster objects, data and functions

Specify "ordinary" message passing relationships

Specify redundancy relationships

Specify message passing relationships for the redundancy relationships (see section 4.3)

Select for each application

Specify objects and methods which are responsible for providing the functionality for message passing

Specify object wrappers for non-object-oriented applications

Coding of objects, methods and object wrappers
```

*Figure 3: Basic activities of conceptual integration (simplified version of [Gassner 96])*
clustering of objects and, in non-object-oriented applications, data and functions into other wrapped applications may be beneficial. This re-clustering may produce new and/or different integration relationships. Then each integration relationship must be modelled in detail. Finally, the functionality responsible for message passing is specified. In object-oriented applications this functionality is provided by methods of objects, whereas in non-object-oriented applications object wrappers must be determined.

• Coding
In this phase the specified objects, methods and object wrappers are implemented. Coding is part of physical integration.

6 Further Research

In our research we concentrate on the integration of administration systems (transaction systems) which are storage and retrieval systems with large databases in the centre. They support strongly structured administration tasks such as wage and salary accounting, production planning, payments and dispatch.

For this purpose we have specified different types of message passing relationships. Furthermore, based on the above model and a (non-object-oriented) meta-model for conceptual integration [Gassner 95], we have defined a documentation model covering all deliverables (documents) necessary for conceptual integration. Finally, we have detailed the integration process model outlined in section 5 [Gassner 96]. It comprises all activities for defining integration objectives from a business point of view and identifying and specifying integration relationships. For each activity we have defined design guidelines and which deliverables are used (as input-documents) and which are produced (as output-documents).

In the next step we will concentrate on physical integration and investigate how different types of (conceptual) message passing relationships can be implemented with different object-oriented and non-object-oriented middleware products.

Acknowledgements

The described approach is based on the research in the Competence Centre Process and System Integration (CC PSI) at the University of St. Gallen, Switzerland, and was formulated at the University of Tsukuba, Japan. We would like to thank the CC PSI working group, without whose support and commitment this work would not have been possible. The CC PSI working group was made up of experts from the following eight Swiss companies: AGI, Bühler AG, Ciba, Credit Suisse, PTT, Swiss Bank Corporation, Union Bank of Switzerland, and Winterthur Insurance.

References

[Alonso 88]

[Barbará 90]

[Booch 94]


[PacBell 95] PacBell Calls on Wrapper Technology, http://www.datamation.com/PlugIn/issues/may15/05bsws1.html


[Winsberg 95] Winsberg, P., Legacy Code: Don't Bag It, Wrap It, http://www.datamation.com/PlugIn/issues/may15/05bsws100.html