"Look-out" are described. Here, the concrete model only affects the stock. Similar approaches are also used in systems without serial numbers.
- **StockUnitCategory**: Here, concrete stock units can be built, for example by building batches or tour numbers of the supplier.
- **UnitLoad**: Container 9 describes the unit load which may hold stock units. A unit load may be loaded with mixed or single-type articles.
- **LoadHandler**: The load handler is used for the physical transport and for this purpose temporarily has to take up unit loads. Thus, a load handler is a "bin location" for a short time.
- **Location**: In container 11 the bin locations are recorded. A location may be occupied or free. In addition, a blocking mark can be set like for all other containers (cf. section 2.2).

The containers described up to now represent the typical warehouse data. Further containers can be added to this model, e.g., a container for the grouping of unit loads to classify unit loads with partially similar characteristics. This model is based on a warehouse with few types of unit loads, as it prevails in practice, and represents the differences in an attribute of the container UnitLoad. A real data model does not necessarily use all containers.

9.1.2 Data interrelations

While the containers in Fig. 9.1 describe the data of a WMS, the lines point out their interrelations. Several kinds of uni- or bilateral relations are possible and their multiplicity is marked by numbers or characters at the end of the connecting line.

- A 0 means that there is no relation in that direction. A 0 does not have to be shown to simplify the presentation. Relations with a 0 at both ends are called empty relations, hold no information and are not drawn.
- A 1 at one end of a connecting line shows that at this end exactly one element of the container is assigned to the container at the other side. For example, a telephone number is related to exactly one legal person (customer, manufacturer or supplier).
- An n at an end of a connecting line indicates that there are a maximum of up to n relations to this side. For example, a legal person may have several telephone numbers but there *may* also be *none*.

In case of a multiple relation in both directions, one side is marked with n and the other with m to indicate that the multiplicity may be different. For example, a relation $\overset{n}{\underline{\quad}}\overset{m}{\underline{\quad}}$ between the manufacturer and the supplier container could be shown if it is relevant. This relation could show that a manufacturer uses the services of n suppliers, while a supplier sells the products of m manufacturers. A relation line $\overset{n}{\underline{\quad}}\overset{m}{\underline{\quad}}$, thus, is the symbolic combination of a $\overset{n}{\underline{\quad}}\overset{0}{\underline{\quad}}$ and a $\overset{0}{\underline{\quad}}\overset{m}{\underline{\quad}}$ line.

In the following, all relation lines shown in Fig. 9.1 are listed and described in a table. The numbers of the respective containers are connected by one of the lines mentioned above and marked with a name.

– (1a $\overset{1}{\underline{\quad}}\overset{n}{\underline{\quad}}$ 6) **Delivery:** This relation describes the delivery of articles whereas a supplier delivers several articles ($n \geq 0$), but one article exactly belongs to the delivery of one supplier. Reasonable relations cannot be differentiated from unreasonable ones with the data model alone. This relation, for example, allows empty deliveries but also deliveries without articles. The superordinate logic has to ensure that a supply contains at least one article.
– (1a $\overset{1}{\underline{\quad}}\overset{n}{\underline{\quad}}$ 3) **Order (or notification):** This relation represents an order (or a notification). An order (a notification) is assigned to exactly one supplier while a supplier may be allocated to several orders (notifications)
– (1a $\overset{m}{\underline{\quad}}\overset{n}{\underline{\quad}}$ 4) **Catalogue:** The catalogue of a supplier contains the meta data of different articles but an article may also be bought from different suppliers.
– (1b $\overset{1}{\underline{\quad}}\overset{n}{\underline{\quad}}$ 4) **Item list:** An article which is represented by its meta data is manufactured by exactly one manufacturer who may, however, have several articles in his assortment.
– (1c $\overset{m}{\underline{\quad}}\overset{n}{\underline{\quad}}$ 4) **Purchasing behavior:** The purchasing behavior of a customer can be described by a relation between item data and customer. A customer may buy different articles and an article may be bought by different customers.
– (1c $\overset{1}{\underline{\quad}}\overset{n}{\underline{\quad}}$ 5) **Order:** A customer may submit different orders during the time but an order is always assigned to exactly one customer.
– (1c $\overset{1}{\underline{\quad}}\overset{n}{\underline{\quad}}$ 6) **Serial number tracking:** With this relation the delivered article can be traced back to the customer and vice versa. A warranty claim, for example, can be settled by tracking the relation starting from **delivery** (to the supplier). This relation also closes the chain for a recall.
– (2 $\overset{m}{\underline{\quad}}\overset{n}{\underline{\quad}}$ 4) **Goods classification:** The relation goods classification combines the meta data of articles with the same characteristics. This relation is of interest mainly in shop or merchandize management systems.
– (3 $\overset{m}{\underline{\quad}}\overset{n}{\underline{\quad}}$ 4) **Storage order:** A delivery (or notification) initiates a storage order which is related to a quantity of articles. A delivery is not only related to the article itself but also to its item data.
– (4 $\overset{m}{\underline{\quad}}\overset{n}{\underline{\quad}}$ 4) **Mixed-storage prohibition:** This relation is necessary because some articles must not be stored together (cf. Chapter 2).
– (4 $\overset{m}{\underline{\quad}}\overset{n}{\underline{\quad}}$ 5) **Retrieval order:** Each customer order is assigned to a quantity of articles which is described by this relation to the respective meta data.
– (6 $\overset{m}{\underline{\quad}}\overset{n}{\underline{\quad}}$ 7) **Article classification:** An article may belong to several categories and each category contains a variety of articles.

- (6 $\overset{n}{\underline{\quad}}\overset{1}{\underline{\quad}}$ 9) **Load:** With this relation the articles are assigned to the unit loads. An article can be on just one unit load[2], but this *can* hold several articles.
- (8 $\overset{1}{\underline{\quad}}\overset{0}{\underline{\quad}}$ 13) **Work order,**
- (9 $\overset{1}{\underline{\quad}}\overset{0}{\underline{\quad}}$ 13) **Transport object,**
- (10 $\overset{1}{\underline{\quad}}\overset{0}{\underline{\quad}}$ 13) **Transport medium,**
- (11 $\overset{1}{\underline{\quad}}\overset{0}{\underline{\quad}}$ 13) **Transport source,**
- (11 $\overset{1}{\underline{\quad}}\overset{0}{\underline{\quad}}$ 13) **Transport destination:** These relations all start from *one* transport order and are unilateral. The relation *work order* assigns just one *worker* to the transport order, e.g., a stacker driver for a driving order or a picker for a picking order. In a fully automatic warehouse the relation *work order* describes the responsibilities. The relation *transport object* indicates which unit load is moved while the relations *transport medium* indicates the relation to the executing load handler. The relations *transport source* and *transport destination* are self-descriptive.
- (9 $\overset{n}{\underline{\quad}}\overset{1}{\underline{\quad}}$ 11) **Bin occupation:** Each unit load has to be assigned to exactly one bin. A storage bin may take up several unit loads.
- (10 $\overset{m}{\underline{\quad}}\overset{n}{\underline{\quad}}$ 11) **Working area:** A certain bin location may be accessed by several machines, but one load handler may also serve several bin locations.
- (11 $\overset{m}{\underline{\quad}}\overset{n}{\underline{\quad}}$ 12) **Zoning:** Different locations can be grouped in zones and each zone may consist of many bin locations.

Further relations can be generated, but are seldom required. The model may be extended, for example, by a $\overset{n}{\underline{\quad}}\overset{m}{\underline{\quad}}$ relation between worker and load handler, which informs users about the skills and authorization to operate certain machines. The respective characteristics then have to be removed from the container *worker*.

The existing relations may also appear in another multiplicity, e.g., the relation *working area* as a $\overset{1}{\underline{\quad}}\overset{n}{\underline{\quad}}$ relation when the bin locations are accessed by exactly one load handler, e.g., a rack feeder. Similar restrictions may also be made concerning the *article classification* when the single category is built by the batch.

The relations may also be interpreted in another way. The relation *purchasing behavior*, for example, may occur as *purchasing prohibition*[3]. If *purchasing behavior* as well as *purchasing prohibition* are wanted as a relation between customer and item master both are shown by a separate line.

[2] Articles which are transported or stored without a unit load are assigned to a "virtual unit load", which is physically nonexistent.

[3] Such a purchasing prohibition for single articles and countries was stipulated, for example, in the CoCom–list (Coordinating Committee on East-West Trade Policy).

9.1.3 Interfaces

The containers which in Fig. 9.1 are crossed by the dotted line build the interface of the WMS. The data in these containers are relevant for the WMS as well as for the adjacent systems such as

– Personnel management
– Quality assurance,
– Forwarding agencies
– Client management and
– Merchandize management

They can be implemented in two different ways: A joint database helps to avoid redundant data stocks and the related disadvantages (cf. section 7.2). Alternatively, the principle of multiple data management can be used, i.e., the data are stored as copies at different locations.

A disadvantage of this method is the high expenditure for the synchronization of data stocks and the short-term infringement of the integrity by unilateral data changes. Nevertheless, this method is often used in practice because it can easily be implemented and allows for the intermittent autonomous operation of sub-systems.

Figure 9.2 demonstrates the method of a multiple data management at the example of the article item data. These data are required by a WMS as well as a merchandize management system (MMS). In addition to the part which is commonly used by both systems – shaded in grey in the center of Fig. 9.2 – the article master also hold specific data for the WMS and the MMS.

Only the mutually used part builds the interface and has to be synchronized. Generally, the data are synchronized cyclically in times of low activity, mainly at night. Other containers with highly dynamical data have to be synchronized much more frequently, in the extreme case even after each change.

9.2 Classical implementation of a WMS

This section describes some aspects of an implementation by means of a relational database. This method to implement the logical concept of a WMS is typical for currently offered systems. Because of the suggestive structure of relational databases and their plausible query language systems can be implemented within a short time.

9.2.1 Functional structure

A WMS can be implemented in two ways:

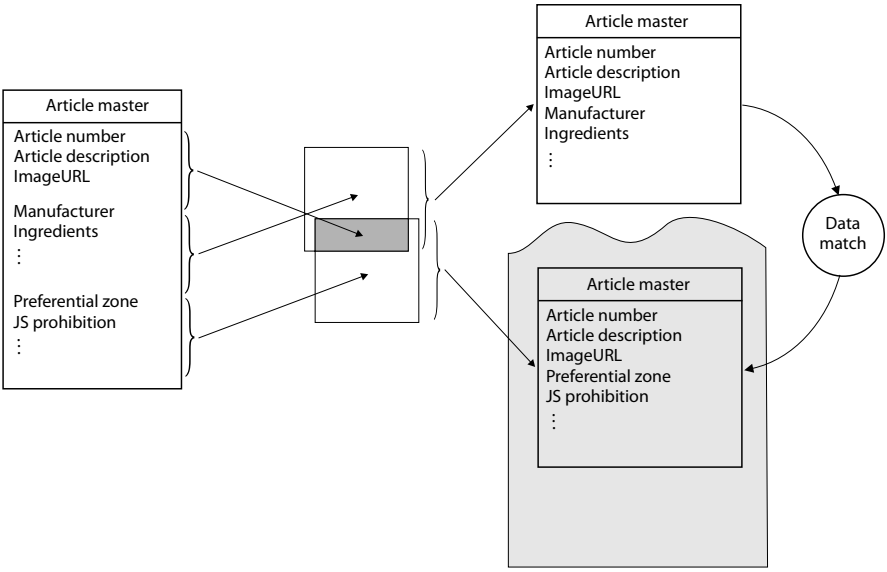

Figure 9.2. Implementation of interfaces by a partially redundant data management

– Mainframe-based architectures (cf. Fig. 9.3), where the WMS runs on a central computer with a database server[4].
– Client-server systems (cf. Fig. 9.4) which bind distributed work stations with specific functions to a database server.

In a mainframe architecture all processes run on the central computer and are operated at terminals with no WMS functions of their own. In client-server systems these processes are distributed completely or partially to the intelligent work stations. The informational tasks are divided as follows:

– Masks can be configured and data represented locally without database access, i.e., by a HTML browser.
– Plausibility checks can be carried out by a client. The correct entry of zip codes or e-mail addresses, for example, can be checked locally. A data transfer from the client to the server and back is an unnecessary communication.
– Integrity checks can be carried out locally when they refer completely to a current operation. For example, when the serial number for an article is entered it can be checked locally that the field for the number of articles

[4] The database operations can also be encapsuled by means of an application server to offer services on a WMS level.

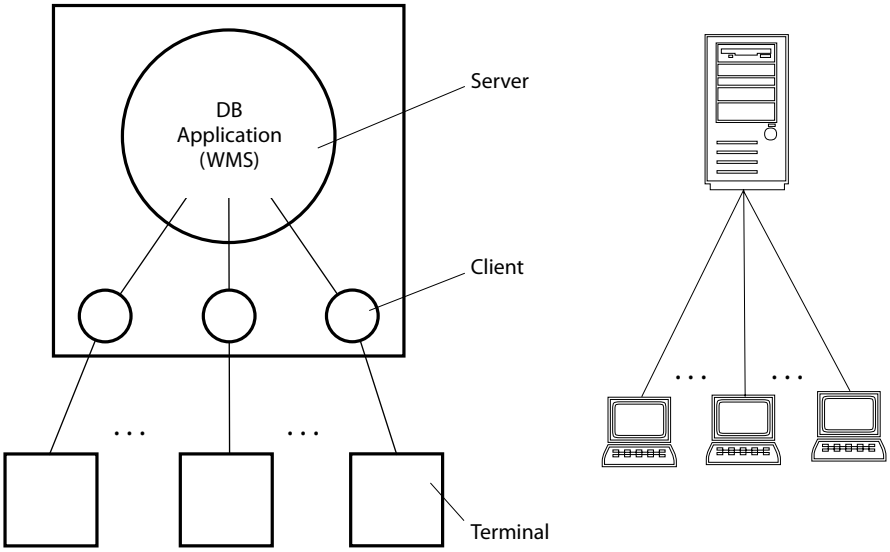

Figure 9.3. Example of a mainframe-based architecture

is not unequal 1. The *uniqueness* of the serial number, however, cannot be checked locally[5].
- Depending on whether data are requested from the database server or not computations can be made locally or just on the server.
- Generally, database operations have to be carried out on the server with the exception of distributed databases which are partially realized by clients (*peer-to-peer solution*).

In practice, it is not always easy to define if a system is based on a mainframe architecture or a client-server solution.

9.2.2 Table structure

The database allows for the structured storage of all data which are necessary for or generated during the operation. This structuring is facilitated by the table concept. Generally, each container shown in Fig. 9.1 can be represented in a database table of its own.

[5] Unless a "serial number" consists of parameters which guarantee for uniqueness like, for example, the millisecond which since 1st January 1970 concatenates with a primary key describing the input terminal.

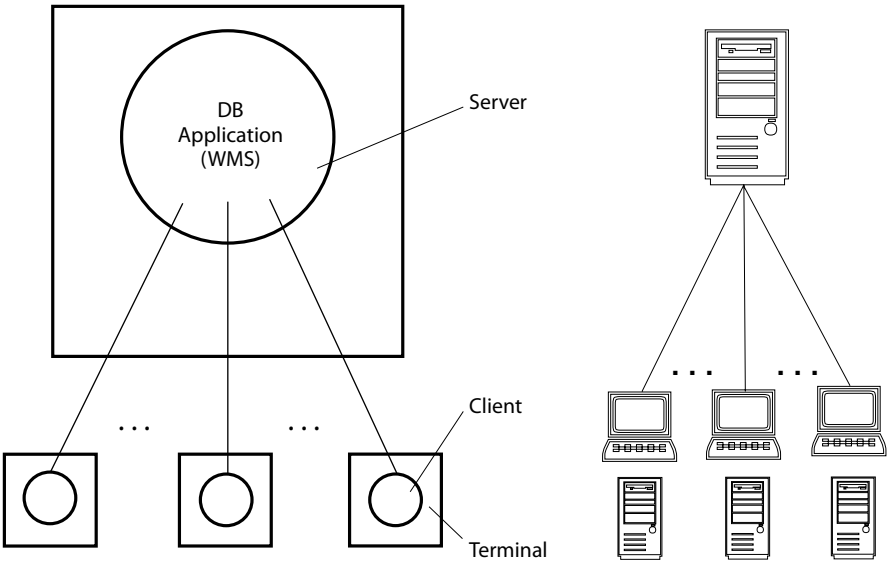

Figure 9.4. Example of a client-server architecture

Before the data can be stored, however, each table has to be structured, i.e., the following storage parameters have to be specified

– what (which)
– where (which table)
– how (which accuracy)
– whereby (which data type)

It makes sense, for example, to store a telephone number as an *integer number*[6] However, the zero as first digit of the area code or special symbols like a plus or a hyphen cannot be stored as an integer data type. For this reason, a telephone number should be stored as a character string where a sufficient number of characters (accuracy) has to be chosen as possible length.

Logically related data like the *ItemData* (cf. Fig. 9.1) are entered line by line into the specified cells of a table. To use relational databases and tables the primary keys for the identification of data lines have to be assigned in a definite way.

A database table *ItemData* could be structured as shown in Table 9.1. In this structure the cell `photo` has a length of 255 characters to enter a URL (see page 233) for the photo. The database type *blob* (Binary Large Object) is one possibility for storing such binary data. The advantage of the URL compared to the blob is that the storage location for the photo data can be freely chosen. Furthermore, multiple references can be made and the database

[6] A data type integer represents a positive or negative integer number.

Table 9.1. Structure of the database table itemData

Name	Type	Length
rId	int	
artno	varChar	20
name	varChar	42
desciption	varChar	255
photo	varChar	255
size	varChar	80
weight	int	
pref.zone	varChar	12
reor.level	int	
act.stock	int	

Table 9.2. Structure of the table stockUnit

Name	Type	Length
rId	int	
serial no	varChar	20
number	int	
itemDataId	int	

does not have to handle large data volumes. A similar method is to describe the data in a separate text file where they can be accessed via a URL. With regard to internationalized descriptions the URL offers many possibilities to manage and represent the data.

In Table 9.1 the field ID `rId` defines a clear rawID as primary key. The article number could not be used as primary key because it would lead to ambiguity at least at the implementation of a client warehouse. Present database systems allow for the automatic generation of primary keys for the single data sets and thus relieve the programmer.

There is a relation $\overset{1}{\underline{\quad}}\overset{n}{\underline{\quad}}$ between ItemData and StockUnit. The ItemData may refer to no, one or several StockUnit tables. This relation is defined by the field `itemDataId` (cf. Table 9.2) that assigns exactly one `rId` of ItemData to *each* line of StockUnit while an `rId` of ItemData can appear in several lines of StockUnit (cf. Fig. 9.5).

The $\overset{n}{\underline{\quad}}\overset{m}{\underline{\quad}}$ association between location and locationCategory can only be built with an auxiliary table, the reference table, also called *map table* (cf. Fig. 9.6).

Each line of the reference table connects a rawID of the table location to a rawID of the table locationCategory and contains a primary key which can be built from the two rawIDs of the respective line.

Without such a reference table the data stock would be highly redundant what would lead to additional memory and runtime requirements.

9.2.3 Securing the logical integrity

Fig. 9.6 shows the structure of the database table locationCategory where the field `characteristic` describes the single zones. The reasonable assignment of a location to a locationCategory has to be ensured by a logistic integrity. Depending on the application each location should or has to be assigned to *exactly one* zone A, B or C. From the logistical point of view a location must not be assigned to several zones but may be allocated to a weight category which is represented by the same field.

Alternatively, the contents of locationCategory can be distributed to several dedicated database tables. This problem is avoided by a distribution with only $\overset{1}{\underline{\quad}}\overset{n}{\underline{\quad}}$ relations. This would impede a statical structure and additional categories *cannot* be generated during operation without having to change the data model.

Another problem regarding the safe logical integrity is described on page 314. When in the database table stockUnit the field `serial number` is occupied the value in the field `number` *has to* be 1.

9.2.4 Generation and query of master data

Data stored in a database are accessed and manipulated by the user in a specific database language (DML[7]). Most database systems support SQL[8] as standardized query language. A user has to know the data model to use SQL for queries and changes of a data stock. The examples describe some typical database operations carried out in SQL.

[7] Data Manipulating Language
[8] Structured Query Language

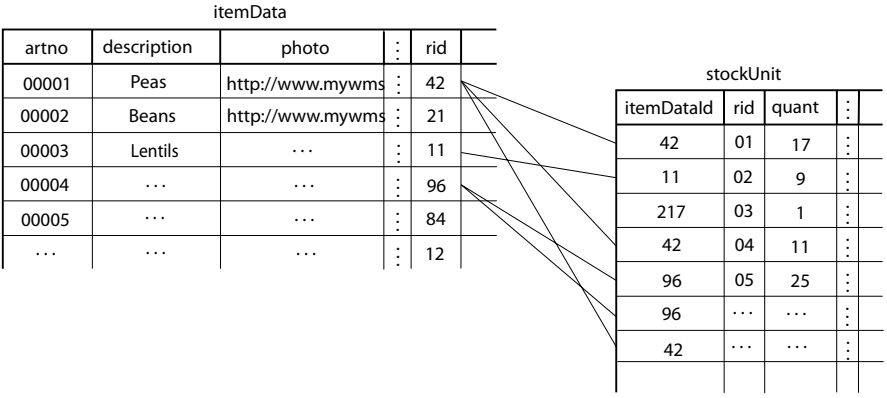

Figure 9.5. Example of a 1 to n relation

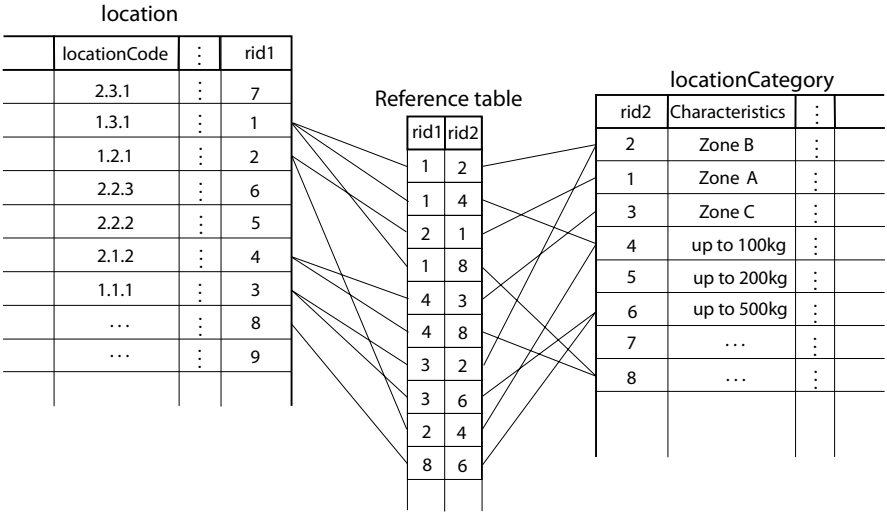

Figure 9.6. Example of a n to m relation

A new data set is entered into the table stockUnit withh the following command where the cell contents are defined by variables. In the example the `stockUnitCount` represents a *sequence* which contains the number of data sets in the table stockUnit:

```
insert into stockUnit
            (rId, serial number, number, itemDataId)
       values
            (stockUnitCount.nextVal,'',11,42);
```

The names of all articles with a weight fewer than ten weight units can be retrieved with

```
select name
       from itemData
       where weight < 10;
```

If all shelves in zone A should be selected this is achieved with the command:

```
select count (*)
       from reference table
       where locationCategory.characteristic = 'Zone A'
       and rid2 = locationCategory.rid2;
```

In the example above the number of selected lines in the reference table corresponds to the number of bin locations which fulfill the desired criterion.

With the function *Stored Procedures* many relational databases offer the possibility to store frequent queries on the server together with the transfer parameters.

9.3 myWMS

The object-oriented programming (OOP) offers many advantages also for the implementation of a WMS. For this reason often object-oriented programming languages[9] are used for more advanced systems. This method has the best effect when not only the programming, but also the analysis and the design of a software are based on it. The focus does not lie on the representation of data structures in a database table, but on an object-oriented analysis. During this phase of the analysis not only the data and their interrelations are described, but also possible *methods* referring to these data. The data are represented in software structures and realized by means of a OOP later on.

Such an OOP solution for WMS was developed in the project myWMS[10]. This section describes the technical concept of myWMS as an example for the consistent use of the OOP technique in WMS. From now on, knowledge about object-orientation and Java is required. But owing to the suggestive structure of this chapter and the chosen examples, the basic principles can be understood without such know-how. Section 9.4 shows the use of this frame for the implementation of a simple WMS.

9.3.1 The basic structure of myWMS

Like an operating system, myWMS provides several basic functions and elementary services for WMS applications. It is a platform and, thus, cannot be operated without specific applications. myWMS creates well-defined communication interfaces to external systems and internal interfaces to the exchangeable modules, the so-called *plug-ins*, which are specified by Java interfaces and allow third parties to integrate own products.

In addition to the pure warehouse management, the myWMS framework also includes a material flow control. myWMS does not deal with aspects of merchandize management, production planning and production control although the framework includes interfaces to such systems.

Since myWMS is not limited to special types of warehouses and logistic processes, its range of application is wide, from manually operated warehouses and automatic warehouses to physically distributed warehouses with heterogeneous structures. Thanks to suitable plug-ins like routing or optimization algorithms the system can easily be adapted to existing warehouses.

[9] According to the general terminology "object-oriented" is abbreviated with OOP.
[10] see http://www.mywms.net

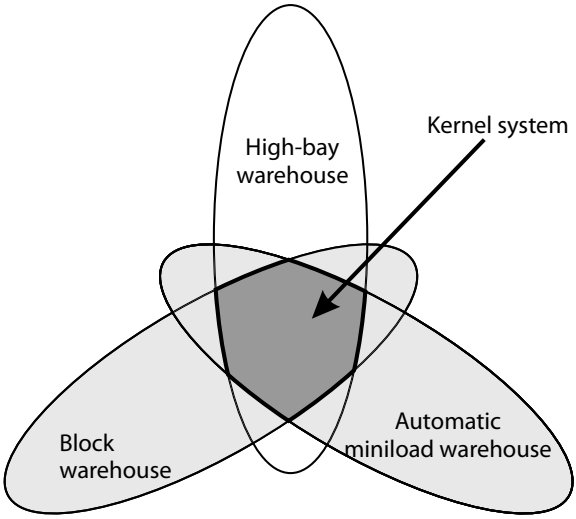

Figure 9.7. As intersection of functions and data structures the core system *simplifies* and *idealizes* different types of warehouses

A plug-in has to be bound to a process – later called *kernel*. A kernel provides basic functions independent of the type of warehouse (cf. Fig. 9.7). myWMS offers a library for frequently used plug-ins and the user can optionally use the original or a modified version or develop his own plug-ins. Thus, a system can have the USP of the user but still benefit from the advantages of a standardized system.

A modular and up-gradable warehouse management system, which is not yet specialized for a certain application, should at least meet the following requirements:

- Independent of platforms: A programming language which allows for the generation of portable codes assures that the system runs on different computer architectures and operating systems.
- Distributability: It should be possible to concurrently run the modules of a WMS on several cooperating computers. The material flow control, for example, could run on another computer than the warehouse management system itself. Since the system is independent of a platform it can be operated on heterogenous computer systems.
- Scalability: Increasing requirements, like a growing throughput rate, for example, or more clients should be met by a WMS by means like distribution, adjustment of modules and parametrization.
- Parametrization: A WMS should allow for the continuous parametrization of its modules. For this purpose the configuration concept should facilitate the persisting of the parameters.

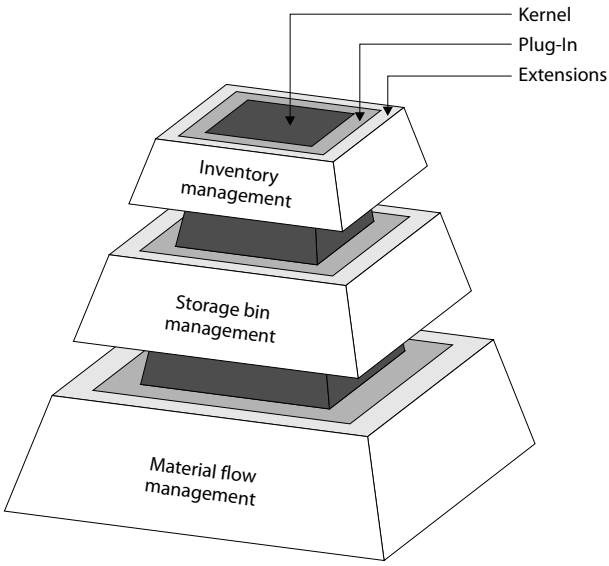

Figure 9.8. The hierarchical model of myWMS

- Releasability: The long-term constancy of the interfaces is an important aspect for the selection of a WMS. The basic software can be exchanged retaining the existing modules and their application-specific extensions as these modules can be exchanged retaining the basic software.
- Web-based operation: The system can be operated with any web browser and is thus independent of platforms. This allows for the integration of the existing IT-infrastructure.
- Independence: No limitation to certain warehouse types, operating means or strategies.
- Upgradability: The system can be adapted to the logistic requirements by generating additional or upgrading existing modules.

myWMS fulfills these requirements by means of the kernel/plug-in technology, a conceptional distribution, the configuration capability and an implementation in Java. myWMS is based on a three-level hierarchy consisting of the inventory management, the location management and the material flow control (cf. Fig. 9.8).

The inventory manager is responsible for the management of item data and article-related operations independent of the location as well as of available and existing quantities of stock units (cf. Fig. 9.1).

The *location manager* manages the current occupancy of the *locations* by unit loads as well as the attributes of the locations.

The *equipment manager* calculates the transport routes, orders the operating means and supervises the operative execution of the transport orders.

Not all systems have to use all three levels. For example, an inventory and location manager can be operated while the bottom level is replaced by a separate material flow computer. In a non-automated warehouse the bottom level, i.e., the equipment manager, may print transport instructions or control radio terminals.

Since the bottom level is operated separately an independent material flow computer without a warehouse function can be built on a myWMS basis.

Each level can also be operated in several instances. A multi-client warehouse, i.e., the provision of several logical warehouse in one warehouse, can be built, for example, by instantiating the two top levels. By means of a multiple instantiation of the bottom level a multiple warehouse, i.e., a warehouse with several physical locations, can be operated. This also allows for the building of multi-client multiple warehouses.

9.3.2 Business objects

The single levels are responsible for managing the respective data stored in the *business objects*. Each business object exactly corresponds to a container in the data model shown in Fig. 9.1. myWMS includes the following business objects:

- **Supply**
- **ItemData**
- **Order**
- **StockUnit**
- **StockUnitCategory**
- **UnitLoad**
- **LoadHandler**
- **Location**
- **LocationCategory**
- **OrderChain**

Thanks to the hierarchical structure single objects can be modelled in detail, e.g., the physical location is defined as **StorageLocation** and the logical one as **Location**. Generally, each logical location has a physical counterpart. The space on the load take-up of a machine is a physical location without a logical equivalent (cf. page 310).

The business objects have been generated consequently according to the OOP principle, i.e., analysis, implementation and test. No attribute can directly be accessed: All attributes are encapsulated and only accessible by so-called accessor methods. Each business object communicates in the abovementioned way via interfaces. Other methods are encapsulated and not available.

Business objects can be specialized by inheritage. If new attributes are added which were unknown or not considered during the implementation they

are realized by key-value codes. This method is generally known as the design pattern *property* and implemented in Java among others as *hashtable*. It is also possible to combine both methods, inheritage and property technique.

The myWMS business object *ItemData* should be adapted to the current application just by derivation because in the basic category only the attributes *itemNumber* and *name* (plain text description) exist, as in every warehouse[11]. Real warehouses may need other attributes like the location of a photo or a minimum inventory (cf. section 2.2). These would then be declared and defined in an *ItemData* upgraded class together with their accessor methods. They are unknown to the kernel and are used by other objects or plug-ins.

For the business object *LocationCategory* can be reasonable to use properties because this allows for the dynamical extension of location groups.

The *comObject*, a category implemented by myWMS, uses both methods. The basic category provides a protected hash table which is filled with every derivation phase. Each derivation provides the corresponding accessor method.

Adequate business processes have to be modelled and implemented to ensure the functioning of a WMS. myWMS can be notified about these business processes by the *PluginInterface*. Another important component is the *SupplyStrategyInterface* as an extension of the *PluginInterface* and serves the *Locations*. The coupling of business processes and business objects is event-driven and based on the *Listener concept*. The business processes do not exchange any status information and and are considered as stateless by myWMS. Typical business processes are

- Storage
- Order-picking
- Retrieval

When a business object is changed all business processes registered as listeners are notified. A business process which receives such a message checks if additional frame conditions are met and, if necessary, performs the next process step. This may require a change of business objects.

myWMS contains some pre-modelled business processes but generally they have to be adjusted to the actual logistic requirements.

9.3.3 Kernel concept

The three levels – inventory management, location management and equipment management – consist of at least one IT process, a so-called kernel. A WMS consists of the kernels of the three levels, the respective plug-ins and the application extensions. The basic principles of myWMS are:

[11] *ItemData* can also use plug-ins via interfaces, such as *ItemDataLockInterface* to manage article blocks.

- Logistic orientation: In contrast to universal databases or application servers myWMS is tailored to the requirements of logistics, especially the warehouse management.
- Flexibility: Since it is limited to core functions and the use of specific plug-ins the system is highly flexible[12].
- Distributions of kernels: The kernels of the different levels can run on different computers. Furthermore, the single levels may consist of several distributed kernels. Section 9.3.1 describes this principle at the example of a multi-client and a multiple warehouse.
- Event-control: To achieve short response times, the event control is stringently used according to the design pattern *listener*. Here, an object registers by callback with one or several other objects.
- Exception handling: Exceptions are unpredictable events during the runtime like the accidental triggering of a sensor signal by the operating staff. Usually, the result of a radio request is returned to the requester in the form of a feedback which describes the success or failure of the action. When the implementation is correct this feedback is handled adequately. The exception handling enforces a transparent further handling what directly improves the quality of the software.
- Integrating active components: The plug-ins are called by the kernel via their methods. They can concurrently carry out actions as independent threads and thus increase the performance.
- Realization of associations: The business objects do not manage any associations but are realized in the kernel. This is one of the prerequisites for the distribution capability.
- Distribution of business objects: If a kernel temporarily distributes a business process to a concurrent process this object is copied with the same object number. Business objects are capable for serialization and are distributed via different computers.
- Assurance of integrity: Although each business object makes elementary integrity checks additional checks can be made with plug-ins. StockUnit, for example, checks if the serial numbers are correctly completed with allowed characters and blanks. With a plug-in it can be checked if these serial numbers already exist.
- Persisting of data: The kernel requests persisting methods which have to be provided by means of a plug-in interface. Some persisting plug-ins are part of myWMS so that data can be secured via the database and the file system can be used by means of serialization. For teaching and training the data can be stored via the same interface without using a database.

According to the above-mentioned concepts the myWMS kernels and the respective plug-ins should best be implemented in Java because it offers:

[12] This limitation does not imply that the kernels are trivial. The open source code reveals the complexity and efficiency of this concept.

– Event and exception handling as well as multithread
– Interfaces to describe abstract classes
– Basic serialization concepts
– Remote Message Invocation (RMI),
– Extensive network capability
– Extensive class libraries for all kinds of problems

Furthermore, Java is the most commonly taught language used by most programmers and software engineers and widely accepted.

9.3.4 Runtime environment

In addition to the kernel and the plug-in library myWMS has a runtime environment, the "standard environment", which offers many useful functions and can also be used for other projects. This environment is divided into the following sections:

– Logging: The logging records different messages which are transferred to exactly one logserver according to their *logLevel* and category. myWMS provides such a logserver as singleton. Thanks to a categorization into
  – executing computer
  – current user
  – date and time
  – object
  – method
  – thread

  the messages can efficiently be analyzed offline.
  – Statistics: Rough statistical data are collected on a separate server, the statistics server, analogous to the recording of log messages.
  – Configuration: In addition to the parametrization the configuration determines which plug-ins are used. When a user logs in at a configuration provider the registered objects are informed about changed configurations. Thus, the configuration can be changed also during the runtime.
  – Clearing: Exceptions which cannot independently be handled by the system are transferred to a *clearing singleton*[13] with a given number of responses or instructions. If a clearing receives such an exception the correct response or acknowledgement of an instruction has to be chosen from outside.
  – Communication: Thanks to the *channel concept* of myWMS communication objects can be transmitted via different physical and logical lines. myWMS implements the transport via the layer of the ISO/OSI protocol stack (cf. section 7.1.1) like the communication via RS232. The protocols

[13] A class which corresponds to the design pattern of the singleton ensures that it is initiated only once.

in the form of plug-ins allow for the serialization and deserialization of the communication objects.
- Event multicasting: Different listener objects can register at a *panel object* which is a listener and thus build a listener group. Events which arrive at the panel object are sent as a copy to all registered listeners. The event multicasting implemented in myWMS can manage up to two listener groups.

The included runtime environment does not necessarily have to be used; only the interfaces have to be considered. Thus, the system can be integrated into existing systems like an own logging.

9.4 Example of a distribution system using myWMS

This section describes a virtual warehouse with typical warehouse management structures, its topology, operating means and logistic processes.

9.4.1 Description of the example

The functions are described at the example of a distribution system for small household appliances. The system consists of a goods receipt and goods issue and the warehouse with an order-picking zone. (cf. Fig. 9.9).

Arriving units are unloaded by stackers and buffered in reserved pick-up zones. After they have been checked and entered into the system, they are brought by stackers to the I-point of the warehouse system (input spur) or into a separate blocking zone for an extensive quality check. On the one hand, complete units are taken from the storage zone and brought with stackers from the output spur to the buffer zones in the goods issue. On the other hand, picking Us are built in front of the high-bay rack and transported analogously to the goods output after completion.

Description of the warehouse system The warehouse (cf. Fig. 9.10) is a simple, automatic high-bay warehouse. On the one hand, it represents typical components and warehouse processes to meet the requirements of the reference system; on the other hand, it is simplified at many points to make it less complex and to outline the most important aspects.

Topology The presented warehouse has 4000 bins and uses Euro pallets (800 mm×1200 mm) as loading aids where the articles are transported and stored homogeneously. It consists of the function areas high-bay rack, pre-storage zone and order-picking terminals.

The high-bay warehouse consists of four aisles with two high-bay racks each. Each aisle is served by a rack feeder. The pre-storage area connects the different function areas. Traversing cars are used for cross-transports.

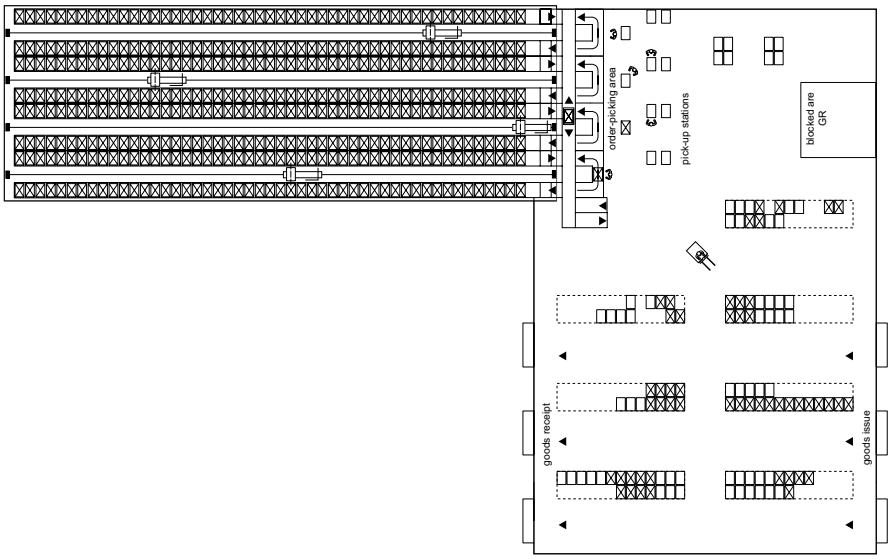

Figure 9.9. The distribution system

Furthermore, the pre-storage area includes the input spur with an I-point, the output spur and two transfer terminals per aisle where the units are transferred between stacker and traversing car. Each of the four order-picking stations consists of the "order-picking U", the space for pickers and pallets to be picked on collecting units. The order-picking U consists of the buffer to be picked from as well as an upstream and downstream transfer terminal as a connection to the traversing cars.

Warehouse technology This form of high-bay warehouse is widely used in practice and represents a classical warehouse technology (cf. Chapter 4). All racks have the same dimensions and are designed for the single-depth lengthwise take-up of standard Euro-pallets. Each of the eight racks consists of 50 sheds in horizontal and 10 sheds in vertical direction, i.e., 500 sheds per rack and a total of 4000 racks. This warehouse does not consider specialities like double-deep storage or racks of different dimensions.

Conveyor technology Each aisle is served by an automatic, rail-bound overhead rack feeder (cf. section 4.2.2). The vehicle cannot change from one aisle into another. All drives are electrical, the energy and control signals are transmitted over contact lines. The load is taken up by a telescopic fork. The most important parameters of the rack feeder are shown in Table 9.3.

The traversing car in the pre-storage area is also rail-bound. However, the energy and control signals are transmitted over trailing cables and the load take-up is a chain conveyor. Since it is a central and decisive element

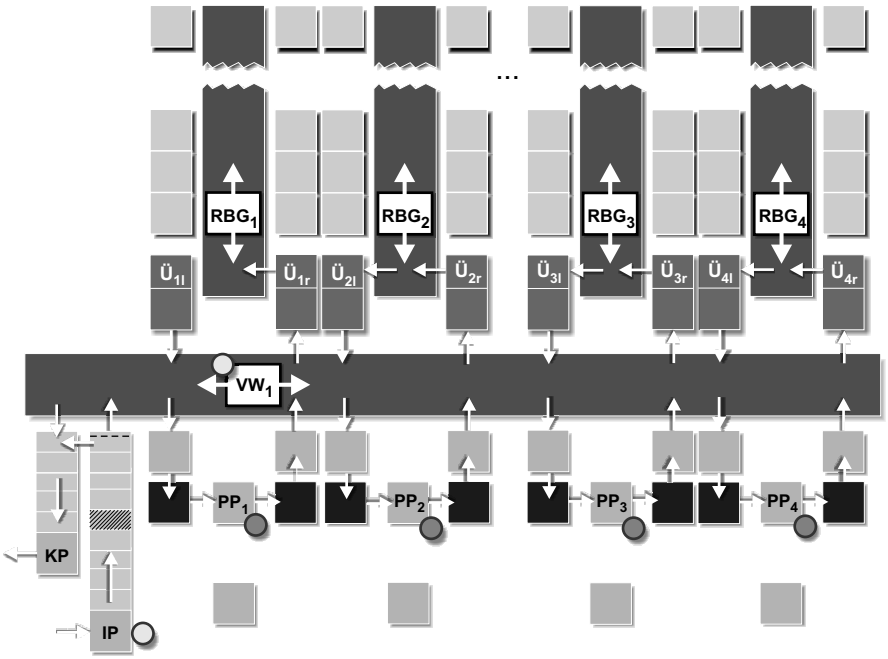

Figure 9.10. The sample warehouse

the traversing car has a high acceleration and speed rate to meet the high requirements.

Two chain conveyors set up in series act as transfer station between pre-storage zone and high-bay rack and offer a buffering capacity of two bins. Each aisle is equipped with two such pairs of chain conveyors: One pair for the input, one for the output. Together with two edge converters the chain conveyor modules build the "order-picking Us". The input and output belts are two driven accumulating roller conveyors. A converter is integrated at the end of the input line which transfers pallets (e.g., in case of a negative contour check) directly to the output line or to the traversing car. Conventional front-end fork lift trucks put pallets on the input line, retrieve pallets from the output line or pick complete pallets. They receive their orders from a separate stacker control system which is no fixed part of this system.

Control technology First, the information for the control of an automatic warehouse is given by the sensors and identification devices. One of these identification media is a barcode label on the loading aid (cf. Chapter 6). This label is applied during the goods receipt process, which takes place outside the system boundary of the reference warehouse and, thus, is not described here. Inside the warehouse each pallet can be identified by two scanning stations. One of these is a stationary scanner at the beginning of

http://www.springer.com/978-3-540-35218-1

Warehouse Management
Automation and Organisation of Warehouse and Order Picking Systems
Hompel, M.; Schmidt, T.
2007, XII, 356 p., Hardcover
ISBN: 978-3-540-35218-1