# Archiving System States by Persistent Objects

Holger Vogelsang Uwe Brinkschulte Marios Siormanolakis
Institute for Microcomputer and Automation, University of Karlsruhe
Haid-und-Neu-Str. 7, 76131 Karlsruhe, Germany
{vogelsang|brinks|sior}@ira.uka.de

## Abstract

*This paper describes one specific aspect of the software component construction in the life cycles of computer based systems. The construction is located following the requirements analysis, conception and design.*

*Systems are designed as within the ECBS process so called services which consist of a set of objects working together, keeping the states of the system. To ensure an efficient and rapid construction of systems an easy to use mechanism to store and retrieve objects together with their relations is necessary. This demands an embedded method of keeping objects in a database — the persistence.*

*The described mechanism is implemented using C++ and verified in some projects. In this paper a man machine service is used as an example to show the application of this approach.*

## 1 Overview

Booch [2] writes, that object persistence is the property of on object through which its existence transcends time and/or space. This first allows an object to survive the lifetime of an application, the second enables an object to move from one address space to an other. This paper describes an approach for persistence in time together with a subset of persistence in space: Objects can be located somewhere on a distributed system, but there is no mechanism realized to move them around.

Object persistence is necessary for several reasons:

- **Reusability**

  After the restart of a system it could be necessary to reuse precalculated or entered states.

- **Memory restrictions**

  Many systems consists of very large state values. Due to memory restrictions it is sometimes not possible or significant to hold the entire state in memory.

- **Safety**

  A fatal system error often causes a total lost of information. This is in most applications not tolerable.

- **Relations**

  In addition to Booch we demand the possibility to keep the relations between objects persistent.

- **Data sharing**

  Data sharing allows different applications to use the same information. Together with a suitable access management this results in an object oriented database.

To assure this functionality in traditional systems, often an application dependent store-and-load mechanism is used to write and restore the object's contents. This can be done by placing objects into a special container object, which itself is responsible for the object management. This has one major disadvantage: Persistent objects cannot be treated like other non-persistent objects. A much better approach is to *request* an object to save its own state or to restore it, because of the following main advantages:

- **Embedding**

  The access of persistent objects is identical to this of non-persistent objects. There are only additional functions to control the load/save mechanism.

- **State dependence**

  An object is able to request itself to save its own state after major or important changes.

- **Platform independence**

  A major problem in the above mentioned man machine service was the reuse of user interfaces on different hardware platforms. As a result, there was no guarantee that saved objects on one system are usable on other systems due to alignment and byte-order problems. This requires the availability of full type information at runtime, which could not be derived from most programming languages. Every object can provide the persistent object system with its own type description, if platform independence is required. This allows the packing into a unique network format, readable and writable on every supported hardware platform.

## 2 Persistent objects

Persistent objects are realized by declaring the corresponding classes as persistent. This is done by deriving these classes from the internal class *Persistent* on the applications side. On the systems internal side every persistent object is connected with one (of several possible) database objects. These acts as stubs for the real local or remote database server in a heterogeneous net of computers. The connection to a stub is dynamically changeable so that an object's content can be loaded from or stored to different databases. The advantage of this design is that a persistent object is placeable anywhere on a net. No object has to know its own storage place, only the reference to the local stub is kept. All database stubs share the same machine-dependent hash table for two purposes: First, to determine whether an object is already in memory and second, to find the memory address for a given object identification. Figure 1 shows the internal structure together with the application interface. The diagram uses the OMT notation [10] for data structures.

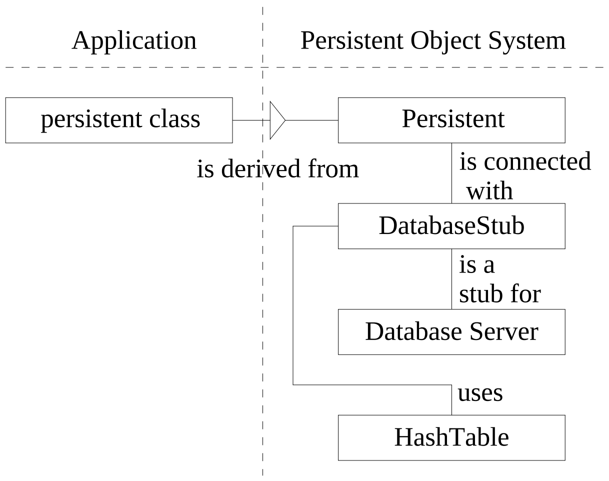

Figure 1: Internal system structure

## 3 Persistent object relations

System states are not only defined by object values, they also consist of relations between objects, which are expressed in object oriented programming languages by pointers. To keep the relations between persistent objects after a program termination alive the special pointer class *PersistentPtr* is introduced. In addition to the memory mapped reference to the destination object it contains the objects unique identification. This class either allows the automatic reloading of the destination object during its own creation (*AutoPersistentPtr*) or a program controlled recreation (*PersistentPtr*). This two mechanisms enable either a total rebuild of the entire data structure only by reloading the base object or a partial loading of huge structures at a given point of time. *PersistentPtr* is totally embedded in the host language (here C++) using overloading of operators to hide most of the functionality. Figure 2 shows the internal structure.

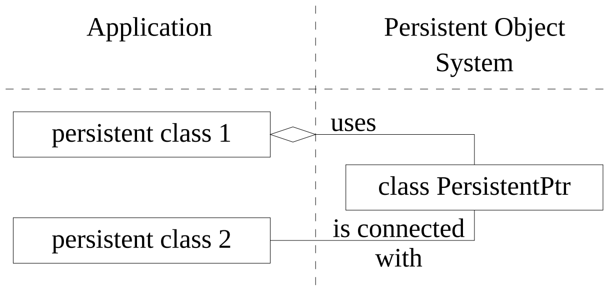

Figure 2: Persistent relation

Figure 3 shows a single linked list of persistent objects as an example for persistent relations. Every object in this list contains a persistent pointer to hold the successor relation to an other object. Even very large data structures can be described an kept persistent using this kind of pointer. The overhead when moving from non-persistent to persistent relations in an application is nearly zero.

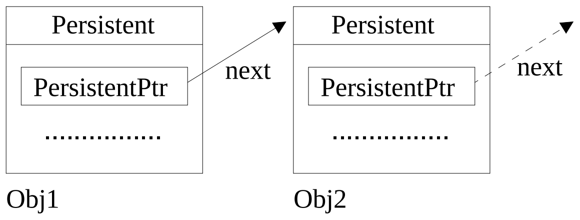

Figure 3: Persistent single linked list

## 4 Realization

The realization is based on the process database service *Merlin*. *Merlin* is one of the basic services developed at the institute to ease the computer based system engineering process.

An attempt using a database server has the following advantages:

- **Simplicity**

  The language dependent part of the component is very small.

- **Cost-efficiency**

  Users are able to use their existing (often expensive) relational database server.

- **Mixed operation**

  Traditional programs with direct access to a relational database can be used together with new systems without the need of two database systems.

- **Common access**

  System are able to share information in the same database because of a build-in protection scheme and synchronization function.

- **Distribution**

  Persistent objects can be distributed on several databases on different computers.

- **Security**

  A crash of a traditional system leads in the worst case to a total or partial loss of information or inconsistent states. A database server with a build-in security mechanism allows the restart on previously defined points using a rollback functionality.

### 4.1 The database server "Merlin"

The process database service *Merlin* is a general service for data management. It was designed to meet the requirements of modern information and automation systems. Its main properties are:

- a powerful but simple interface to the application program
- cooperative and distributed network database management
- real-time system operation
- 24 hour on-line availability
- configurable data security
- a small amount of needed systems resources
- portability and platform independence

For data storage, Merlin uses relational data structures. As a special feature, large unformatted objects up to 4 Gbyte size can be stored as part of the database relations. Merlin offers an easy set oriented interface containing operations for high speed data access and manipulation, data security and data control. This interface is embedded in the language C/C++. Using multiple client server architectures, database operations can be distributed in a heterogeneous network. This distribution is mostly hidden to the application program. To provide platform independence and portability, Merlin can operate on a various number of hardware and software platforms. These attributes make Merlin suitable for storing the persistent objects. A more detailed description of Merlin, it's aims and features can be found in [3], [4] and [5].

### 4.2 Mapping between Merlin and persistent object system

The implementation of persistent objects has the class hierarchy described in figure 4. Only the grey shaded components are visible to the programmer. All other components are part of the persistent object system, which are hidden from the user.

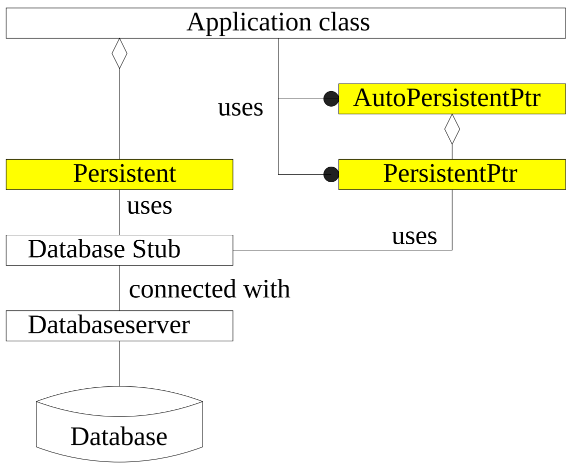

Figure 4: Class hierarchy

## 5 Example: Man machine service

This realization of persistent objects was validated using it for an implementation of a Man Machine Service

(MMS) for distributed and heterogeneous systems.

Most systems are not able to work without the influence of a human user because they need decisions they can't make themselves. Influence in this case means changes of a systems internal state due to an interactive access. On the other hand users must be informed of system states or parts of it to allow the inspection of the system.

The man machine service is triggered by input events due to internal state changes of the aggregate system. First, an interactive access of a user causes changes in the internal states of the man machine service. Second, if the affected states are observable outside the MMS, new events are generated to trigger other subsystems.

Due to the fact that man can perceive and handle information fastest in a visual way, this is the best channel to inform about complex system states. The optical channel is also a good choice to support human interaction. This is achieved in feeding back the user's actions. A MMS has to provide two general services: The visualization of structured information and the modification of structured information. Based upon these services any communication between user and aggregated system can be realized.

A symbol is a graphical representation of a structured data type. The user can define symbols very flexible with a tool called *Symboleditor*. The symbols are realized as persistent objects for the usage within the application. Symbols can be defined hierarchically, i.e. a symbol can contain other symbols or base symbols. Changing a object connected to a symbol leads to a different graphical representation. Changing the graphical representation (e.g. the user moves a symbol interactive) leads to a different object value. The relations between data type values and the resulting images can be defined. This relation is either continuous where we provide linear or logarithmic functions or discrete.

Normal symbols can be used to visualize a big amount of user defined data types. The *presentation object*, a new type of symbol is introduced to offer the developer the facility to group symbols together and to create images of complex data type with a special semantic. There are different types of presentation objects predefined: A **picture** is a set of symbols as an image of a set of objects. A **menu** is an image for a variable of an enumeration type, each button shows a selectable value. A **mask** is an image for an object or a structure of an application. Modifiable components of the object can be changed by the manipulation of the corresponding symbols (sliders, buttons, textfields, ...). A **table** is an image of an array of objects or structures.

Presentation objects can be build automatically by the service if the type of the corresponding object is known. Because every presentation object is derived from the same common class, they share the same (small) set of operations. The presentation objects them self are used to build higher level objects like text editors, hierarchical graphs and help systems.

All symbols are arranged and positioned in planes. Each plane defines a unit of measurement respectively a scale. One symbol can only be assigned to one plane. A plane allows the grouping of symbols. The user can define rectangular areas on the screen. We call such areas windows. Windows can superpose each other and the sequence in the window stack can be changed. Planes with all their symbols can be displayed in windows. A window can hold multiple planes simultaneously and a plane can be displayed with different scales in multiple windows. Each window holds a stack of the assigned planes and their scales. The stack sequence is changeable.

It is possible to create *bindings* between symbol events and operations, or between events on presentation objects and operations: Several internal operations of the man machine service can be bound to events, so that typical interactions can be created by the GUI tool without writing any line of code. Hierarchical menus, masks and tables are created by bindings between presentation objects. Furthermore, user defined operations are bound to events to create callback functions. An application is able to catch an event using this technique.

The graphical user interface is a group of presentation objects and windows, which are needed at the same time to solve a given task. It is realized using persistent objects to separate the application from the GUI. This allows the reuse of the entire GUI or parts of it in other applications on different hardware platforms and the on-line modification of the GUI through the application itself.

The man machine interface requires tools to allow an interactive and comfortable way to create symbols and GUI's. These tools are the symboleditor to construct symbols as images of objects and the GUI-editor to build graphical user interfaces as a set of presentation objects and windows.

The resulting environment allows pleasant and comfortable development of user interfaces for distributed systems. The major features of this man machine service are:

- Portability
- Flexibility
- Complex graphical objects
- Tools and editor services
- Separation of GUI database and application
- Resolution independence
- Client/Server-Concept

## 6 Conclusion

The persistent objects have been recently used with success in different applications such as a control of a production cell. Since the MMS is implemented using persistent objects many of the above mentioned features such as the separation of GUI and application were realized without much afford.

## 7 Acknowledgment

This paper is based on research done at the Institute for Microcomputers and Automation, Prof. Schweizer and Prof. Brinkschulte.

## References

[1] O. Bantleon et.al., “Streams++: Portable Bibliothek für persistente Objekte in C++”, *Articles in iX 3-4/1994*

[2] G. Booch, *Object oriented design with applications*, The Benjamin-Cummings Publishing Company, 1991

[3] U. Brinkschulte, “MERLIN — Ein Prozeßdatenhaltungssystem für Echtzeitanwendungen”, *in conference proceedings, Echtzeit 93*, Karlsruhe, Germany, 1993

[4] U. Brinkschulte, “Architektur eines Datenhaltungsdienstes”, *in conference proceedings, 39. Wissenschaftliches Kolloqium, TU Illmenau*, Illmenau, Germany, September 1994

[5] U. Brinkschulte, “Database Services”, *in conference proceedings, KEOOA 95, Knowledge Engineering and Object Oriented Automation Workshop*, Strasbourg, France, May 1995

[6] U. Brinkschulte, M. Siormanolakis, H. Vogelsang, “Graphical User Interfaces for Heterogeneous Distributed Systems”, *in: proceedings of EI’96*, San Jose, USA, 1996

[7] U. Brinkschulte, M. Siormanolakis, H. Vogelsang, “Visualization and Manipulation of Structured Information”, *in: proceedings of Visual96*, Melbourne, Australia, 1996

[8] A. Dearle, R. Bona, J. Farrow, F. Henskens, A. Lindström, J. Rosenberg, F. Vaughan, *Grasshopper: An orthogonally persistent operating system*, Department of Computer Science, University of Adelaide and Sydney, Austria

[9] O. Hammerschmidt, H. Vogelsang, “Design of Distributed Real Time Systems in Process Control Applications”, *in proceedings of CIMPRO’96*, Eindhoven, Netherlands, 1996

[10] J. Rumbaugh et.al., *Object-Oriented Modeling and Design*, Prentice Hall, 1991

[11] V. Singhal, S. V. Kakkad, P. R. Wilson, “Texas: An Efficient, Portable Persistent Store”, *in proceedings of the Fifth International Workshop on Persistent Object Systems*, San Miniato, Italy, September 1992

[12] A. Stevens, “Persistent Objects in C++”, *Dr. Dobb’s Journal*, December 1992

[13] H. Vogelsang, U. Brinkschulte, “Persistent Objects in a Relational Database”, *Submitted paper to ECOOP’96*, Linz, Austria, 1996

[14] M. Voss, “System Theories for an Engineering Discipline of Computer-Based Systems”, *in proceedings of EMCSR96/Session C*