A description of the interface interaction model of specialized information and software of the distributed automated process control systems

I V Kovalev1,2,3, V V Losev2, M V Saramud2, D I Kovalev2,5* and G D Voskoboinik4

1 Siberian federal university, 79 Svobodny avenue, Krasnoyarsk, 660041, Russia
2 Reshetnev Siberian State University of Science and Technology, 31, Krasnoyarsky Rabochy Av., Krasnoyarsk, 660037, Russia
3 Krasnoyarsk Science and Technology City Hall of the Russian Union of Scientific and Engineering Associations, 61 Uritskogo street, Krasnoyarsk, 660049, Russia
4 Xingtai University, 3 Shizhuan Street, Qiaodong, Xingtai, 54001 Hebei, China

*E-mail: grimm7j@W@gmail.com

Abstract. The analysis of the possibility of a revision of the algorithmic entity interface interaction was completed by authors. The algorithmic entity interface interaction is considered as technologies for organizing specialized information and software for automated process control systems. In the frame of an inertia reduction of interaction of the heterogeneous environment components of an automatic controlling system of the technological process, the paper is suggested for some ways to increase the capacity of the interaction interface of the systems. One of this is the organization of a connections pool based on the iterative calculations of the developed transaction formation approach.

According to the research results of the heterogeneous environment interaction components of an automated process control system [1,2], a perforation character of data recording was found, that can be traced in a single-level and inter-level architectural variant of building system interaction. This circumstance is caused by the inertia that has a place in the interaction of heterogeneous (disparate) media, that value is $T_{in} = 1.22$ sec.

Taking into account the formation (generation) of data of the “computer numerical control” level [3], in an emulation mode of the project of the automated process control system, it was possible to determine the priority tasks of reducing the of the systems interaction inertia. One of the tasks is to increase the bandwidth of the systems interaction interface.

A possible direction of increasing the performance of an interface is the organization of a pool (set) of connections (connectors). This procedure will permit to the application (SCADA-system) repeatedly establishes and terminates the connection to select the pool connectors, excluding the reconnect procedure [4]. There is a possibility of using pool connectors by several components in one process; this circumstance allows separate components to interact with one another within a single process without two-way notification. The pool connectors can be used multiple times by several components. When organizing a pool of connectors, the ODBC driver used [5] must be absolutely thread-safe, which allows simultaneous execution of various calls from different streams (for example, to connect in one stream, a compound the connection in the second stream, and disconnect in the third stream).
For building of an essence of the algorithmic interface model between the heterogeneous systems, we turn to the method of modeling information systems using Unified Modeling Language diagrams (UML) [6].

We suggest constructing a model of the algorithmic essence of the interaction interface of the systems under consideration in the context of UML diagrams (figure 1).

**Figure 1.** An interface model of interaction in the representation of UML diagrams of objects.

The primary diagram for constructing the interaction of the heterogeneous systems is the object diagram, which presents in a component structure. According to an example of the initiators of the Select Class and Insert Class requests are the components of the SCADA system: the archive trend (Arch1: Trend) and the temperature parameter channel (Temp1: Channel). The component, which forms the query the SQL syntax is (: Mng.Query), whose functions also include calling the ODBC driver manager (: Mng.ODBC). The above-mentioned component provides the functions of establishing a connection, authorization with a data source, in accordance with the attributes of the connector, expressed in the version of ODBC (2.0 / 3.0) and the type of DBMS driver [7]. In turn, the driver (: Drv.ODBC) provides procedures for initializing the query structure and their execution to the DBMS. In the case of a remote location of the data source relative to the client application, the driver makes additional calls to ensure the internetwork transport, according to the TCP / IP protocol stack.

The next diagram of building interaction between the heterogeneous systems, which are expressed in the operational behavior of components by calling the ODBC API functions. This is an activity diagram (Figure 2).

In the first step when implementing access to the data source via the ODBC API, the component (: Mng.Query) initializes the procedure for creating an environment descriptor using the SQLAllocHandle function and the SQL_HANDLE_ENV type of the descriptor. That defines to access a various information, including the current settings of all environment attributes, connection descriptors, created for the given environment and diagnostics of the environment level. After allocated memory for the environment descriptor, the component (: Mng.Query) calls the SQLSetEnvAttr function to set the value of the environment descriptor attribute SQL_ATTR_ODBC_VERSION (the ODBC API version number). The next step is to create a connection descriptor using the SQLAllocHandle function and the SQL_HANDLE_DBC descriptor type. As a result, the driver allocates memory to store the connection information and returns the value of the connection descriptor. The SQLConnect function is called for connect to the data source.

After the connection is established, operator descriptors are created to execute the SQL statements [8]. It permits to access information about the executed statement, the cursor name and attributes. A creation a descriptor (identifier) of the operator is implemented by the function SQLAllocHandle and the type of the descriptor SQL_HANDLE_STMT. This procedure initializes the structure in order to obtain information about the current request. In addition, there is a procedure for specifying an explicitly
placed application descriptor to the driver via the SQLSetStmtAttr function and the SQL_ATTR_APP_ROW_DESC descriptor type.

![Diagram](image)

**Figure 2.** Interaction Interface Model represented by UML activity diagram.

In a result of calling the function that executes the SQL statement, which determines the result set, the data selection function becomes available. The ODBC API provides two functions: SQLExecute executes a compiled SQL statement, while first compiling the statement and creating an implementation
plan, and after, its execution, possibly multiple; SQLExecDirect executes the SQL statement specified by the parameter, while compiling the SQL statement and executing it once, when calling this function.

The SQLExecDirect function passes the extracted SQL statement to the data source through the ODBC driver and follows the logic of the Insert Class query. At the same time, returning data in the form of a result set is excluded. However, it entails calling the SQLRowCount function returning the number of rows affected by the SQL statement.

When generate a query of the Select Class of the SQLExecute function, it is preceded by the SQLPrepare function, whose task is to initialize the compilation of the SQL statement. Upon completion of the compilation, the SQL statement is executed by the SQLExecute function. The result set in the allocated memory area is formed after a result of processing. The next step is to allocate this set from the specified memory to the variables of the programming language using the SQLFetch function, which performs simultaneous data retrieval only if the result set fields were previously associated with the variables by calling the SQLBindCol function for each associated field.

The SQL query can be a transaction, if the connector is not set to AutoCommit, and the transaction has not been explicitly initialized before executing the SQL statement, the driver automatically opens the transaction. In this case, for successful (Commit) or abnormal (Rollback) completion of the transaction, the SQLEndTran function is used.

The final step in accessing the data source is to release the operator descriptors, a function of SQLFreeHandle, break the connection and release the connection descriptors and the environment.

Connection pooling organization.

The projects of distributed process control systems involve hundreds of information I/O points, which becomes the potential data sources and with a high percentage of archiving. The ability to pool connections exists. It is necessary to ensure an adequate number of one-time connections to the database. And as a result, it exerts intensive load on DBMS via SQL queries.

The ODBC technology, by means of which addresses to a DBMS are considered, involves a multi-step process of establishing and breaking a connection within a single request. However, there is the possibility of organizing a pool of connections (connectors) necessary to ensure an adequate number of one-time connections to the DBMS.

The connection pool is managed by the driver manager. The connection is selected from the pool when the application calls the SQLConnect function. And it is returned to the connector pool when the SQLDisconnect function is executed. The size of the pool changes dynamically: if the connection has not been used for a certain period of time, then it is removed from the pool.

Using the SQL_ATTR_CONNECTION_DEAD attribute in the ODBC API 3.x version, the driver manager can determine the connection status from the pool: the SQL_CD_TRUE value determines that the connection is broken, and the SQL_CD_FALSE value means that the connection remains active.

The inclusion of the connection pool mode is implemented by the SQLSetEnvAttr function with the SQL_ATTR_CONNECTION_POOLING environment attribute equal to the SQL_CP_ONE_PER_DRIVER value or the SQL_CP_ONE_PER_HENV value. When the SQLSetEnvAttr function is called, the environment descriptor is set to NULL, which defines the SQL_ATTR_CONNECTION_POOLING attribute as a process level attribute. The value of SQL_CP_ONE_PER_DRIVER determines that a separate connection pool is supported for each driver.

The next step creates the environment descriptor by calling the SQLAllocHandle function with the descriptor type parameter value equal to SQL_HANDLE_ENV. The created environment will be implicitly shared.

There is exist to need for creating a connection descriptor by calling the SQLAllocHandle function with the descriptor type parameter value equal to SQL_HANDLE_DBC. The driver manager will search for the existing shared environment with the corresponding attributes of the environment (when the required environment is found, it is returned to the application and the driver manager increases the counter value by 1). If there is no such environment, the driver manager creates it and sets the counter value to 1.
For getting the connection from the connector pool has to call the SQLConnect function. The driver manager uses the parameter values and connection attribute values to determine the desired connection from the connector pool. This takes into account the value of the SQL_ATTR_CP_MATCH attribute (the correspondence of the required connector to the connection from the pool).

![Diagram](image)

**Figure 3.** A connection establishment model via interaction interface in UML sequence diagram view.

There has to call the SQL Disconnect function for breaking a connection. When this connection is returned to the pool and provided for further using.

In Figure 3 depicts a sequence diagram reflecting the interaction of components of a heterogeneous process control system in time, in the form of a procedure for establishing and terminating a connection with a DBMS, by calling ODBC API functions. The organization of a pool of connectors will permit to close the connection and return it to the pool after completing work with the DBMS. Though the connection and environment descriptors will not be released, this will reduce the time for engaging the interface of interaction of heterogeneous systems.

As part of reducing the inertia of interaction between the heterogeneous environment components of the automated process control system, the work is suggested some ways to increase the interface capacity of the interaction of the systems under consideration, one of which is the organization of a pool of connections based on iterative calculations of the developed transaction formation approach.

**References**

[1] Losev V V, Kovalev I V and Karetnikov K V 2010 On the issue of improving the efficiency of information support of the automated process control system *Industrial ACS and Controllers* 8 1-4

[2] Losev V V 2009 Research of heterogeneous systems in projects of automation of technological processes *Vestnik «KRO RIA»* 1(27) 13-25

[3] Demenkov N P 2004 *SCADA- as a design tool for automated process control systems* (Moscow: Moscow State Technical University named after N.E. Bauman Publ)
[4] Kuznetsov P A, Kovalev I V, Losev V V, Kalinin A O and Murygin A V 2016 *IOP Conf. Ser.: Mater. Sci. Eng.* **122**

[5] Broydo V L 2006 *Computer systems, networks and telecommunications* (SPb.: Piter)

[6] Booch G, Jacobson I and Rumbaugh J 1999 *The unified modeling language user guide* (Addison Wesley)

[7] Haught D, Ferguson J 1996 *Microsoft Jet Database Engine Programmer's Guide* (Microsoft Press)

[8] ANSI X3.135-1992 American National Standard for Information Systems (Database Language SQL)