THE TUXEDO SYSTEM

TUXEDO =
Transaction for UniX Extended for Distributed Operations

A well known middleware product

It provides an
- efficient
- secure
- flexibe
- controlled

operation in distributed system configuration

The development of TUXEDO dates back to the late 1970's

Services of the TUXEDO

The applications using the services of the TUXEDO are running in client-server configuration

services of the TUXEDO:
- it provides site independency, the client needn't know the location of the required server
- it provides a balanced server workload in the whole network
- it provides a standard and open interface to the datasources and to the clients
- it provides several API alternatives to the client components
- it provides security services

Application areas of TUXEDO

The TUXEDO is a useful tool for the following types of applications:

- distributed applications without any database access
- single server type OLTP application
- distributed application with OLTP operations

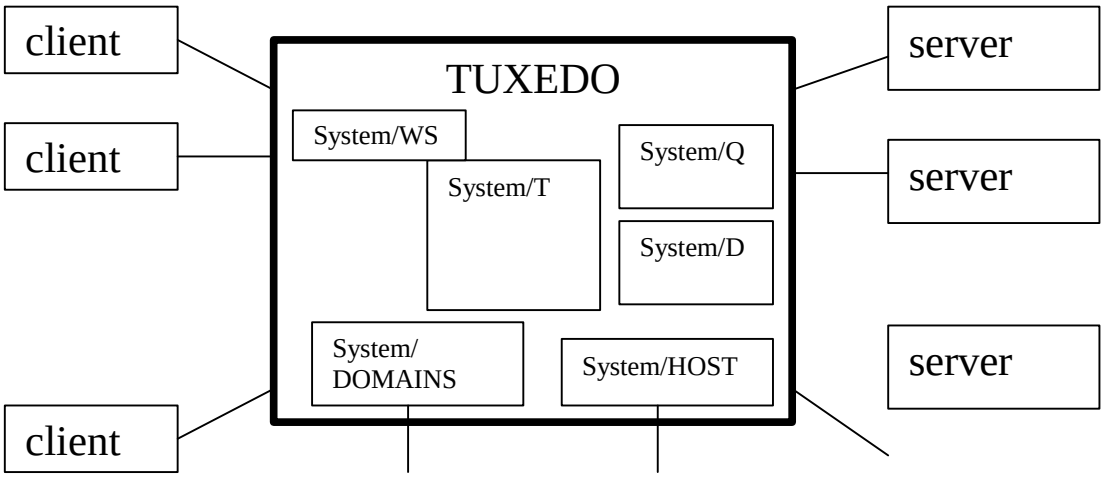
It can manage

- heterogeneous data source types
- heterogeneous display modes
- heterogeneous network protocols
- distributed transactions
- access to the different database management systems

Structure of the TUXEDO

the main TUXEDO components

- System/T : core system
- System/D : data management
- System/Q job management
- System/WS connection to the clients
- system/DOMAIN connection to other TUXEDO



OLTP monitor

OLTP: Online Transaction Processing

OLTP monitor =
DP monitor + DTP monitor

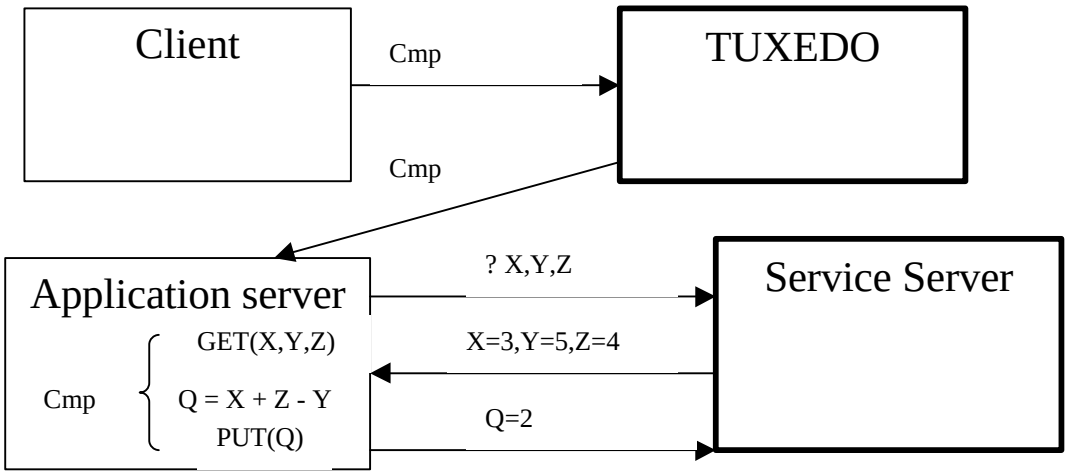
DP monitor =
distributed environment + no data management or only local data management

DTP monitor =
distributed environment + distributed transaction management

Simple TUXEDO structure

The configuration contains:

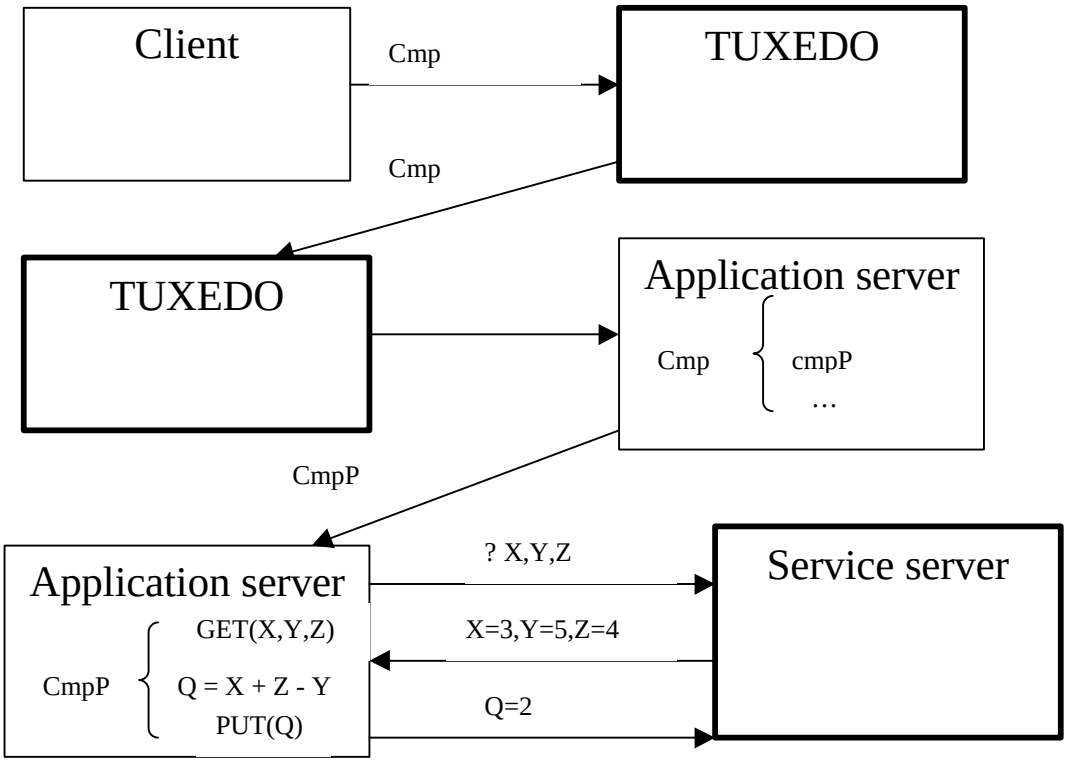
- clients
- application servers
- database or service servers
- one TUXEDO module



Complex TUXEDO structure

The configuration contains:

- clients
- application servers
- database or service servers
- more than one TUXEDO modules



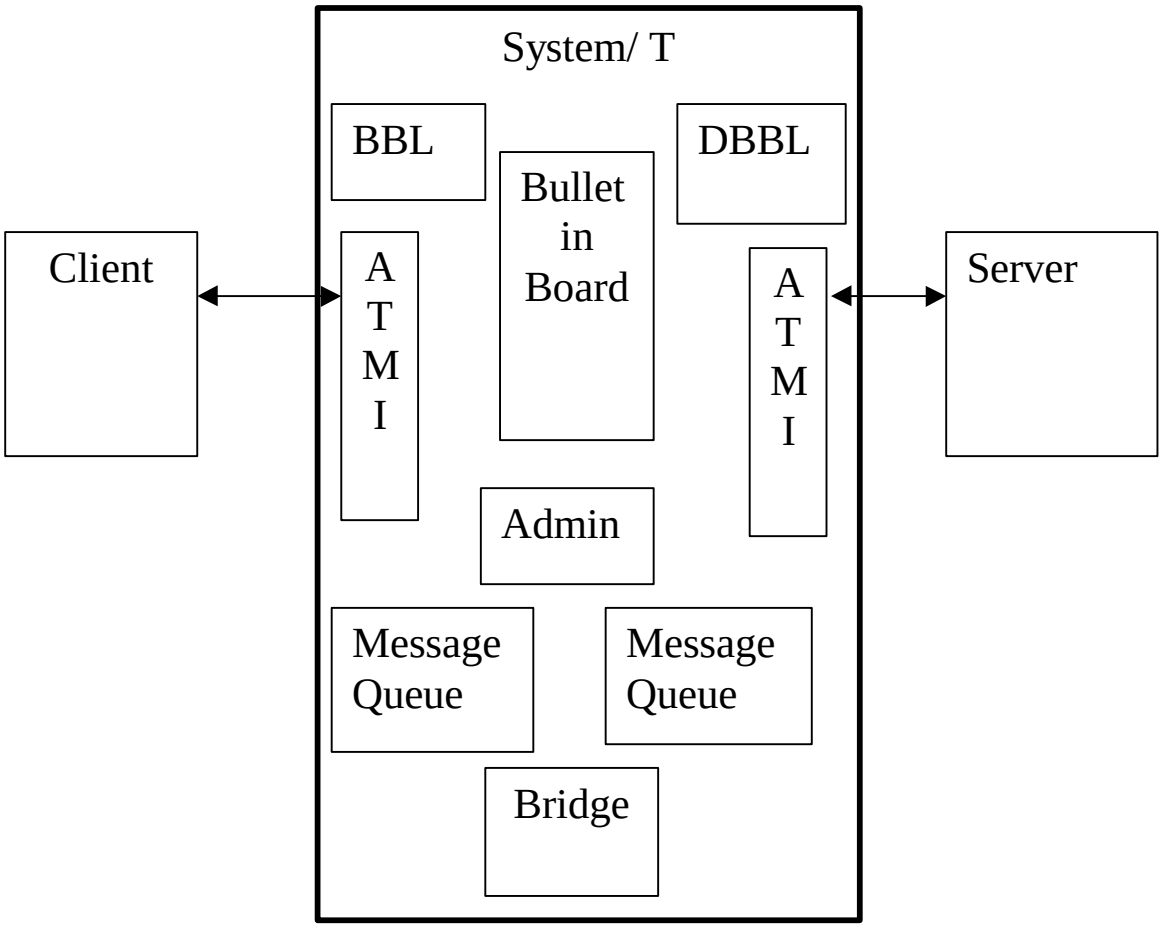
The functionality of the SYSTEM/T module

The SYSTEM/T module manage the the following information:

- clients with active connections
- the list of services requested by the clients
- access path to the servers
- list of services provided by the servers
- work load of the services
- message passing
- data passing
- message form conversation

The SYSTEM/T module contains several sub modules

The structure of SYSTEM/T



SYSTEM/T components

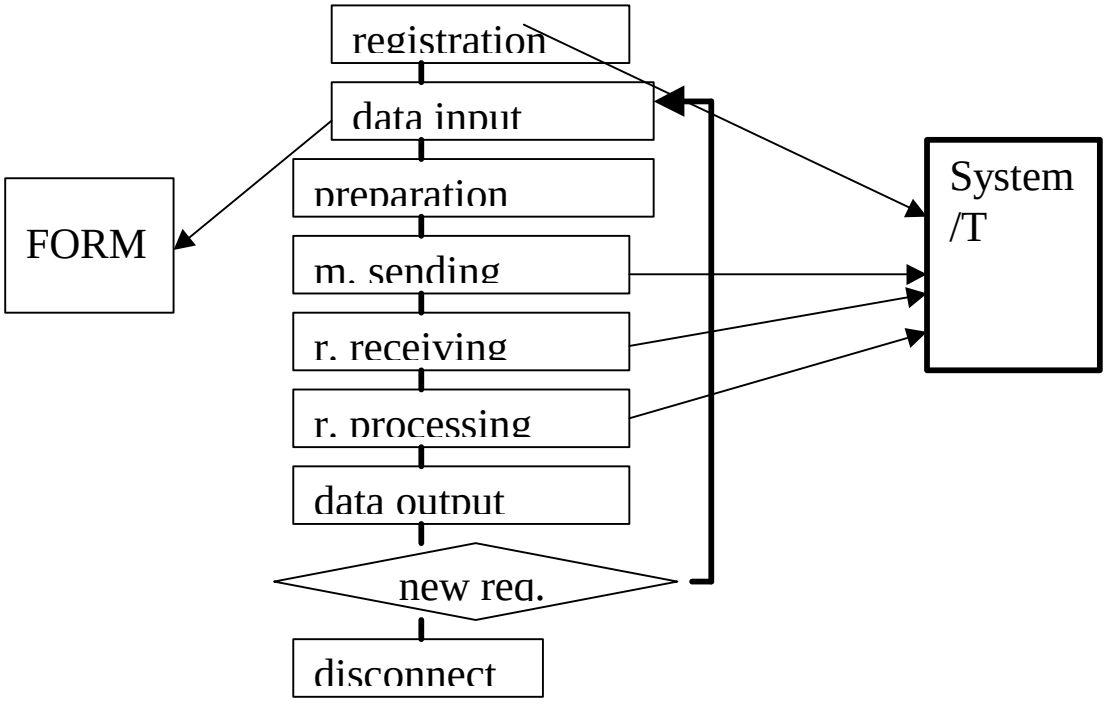
- Bulletin Board
  repository, description of the system state
- Message Queue
  list of messages sent by the clients or servers

Bridge
  communication service between the System/T moduls
- BBL
  administration of the BB structure
- DBBL:
  distributed administration of the cooperating BB structures
- ATMI
  API interface library for the client application

Communication Process

The main steps of the communication process at the client site:

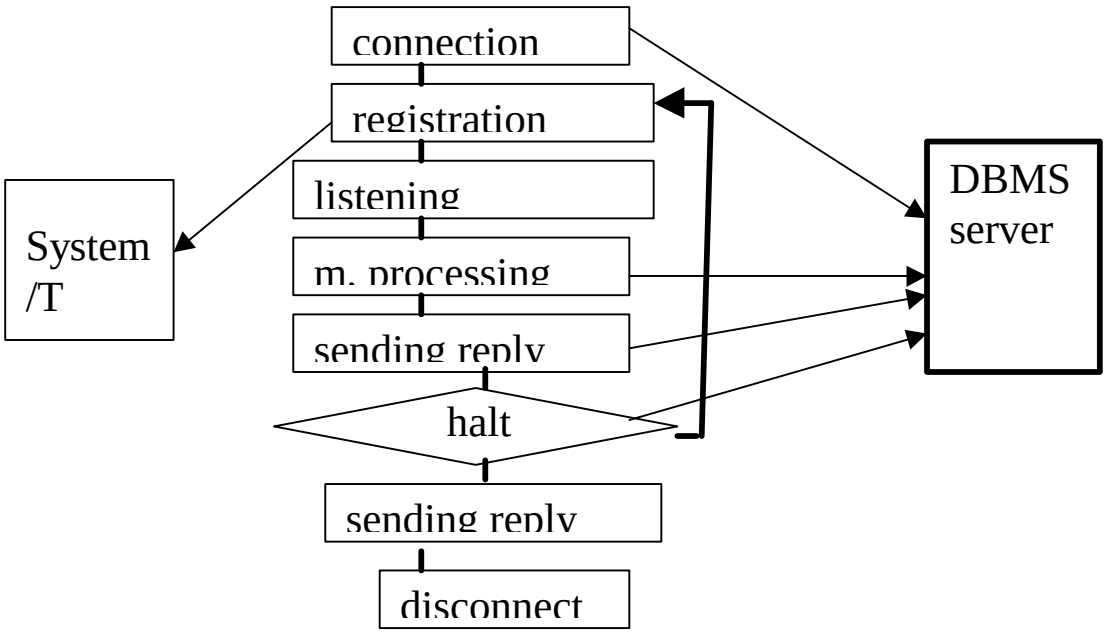
- registration of the client
- client decides to use a service
- message generation
- sending the message to the TUXEDO
- receiving the reply message
- release the registration



Communication Process

The main steps of the communication process at the server site:

- registration of the server
- registration of the available services
- listening to the service requests from the TUXEDO
- message processing
- sending reply to the TUXEDO
- stopping the listening cycle
- release the registration



ATMI function calls

tpinit	registration
tpalloc	allocation of the communication buffer
tpfree	release the communication buffer
tpacall	sending an asynchron service request
tpcall	sending a synchron service request
tpgetrply	waiting for the reply
tpconnect	opening a communicatiob channel
tpsend	sending data
tprecv	receving data
tpreturn	sending a reply of the server after processing a request
tpbegin	starting a transaction
tpcommit	committing the transaction
tpabort	aborting the transaction

Communication models

There is an intensive information and data exchange between the clients and the servers

The server should receive the following data elements:

- data and services to be processed
- process parameters
- authentication data

Two alternatives of the communication:

- RPC : remote procedure call
  - lower network traffic
  - higher autonomy
- Messsage based

  More flexible service processing

RPC communication

The RPC can be work in synchron or in asynchron mode

In synchron RPC, the client gets in a waiting state, it will be continue only after receiving the reply from the server

The execution steps of a tpcall call:

- look up the servers providing the requested service
- selecting an available server from the list above
- generating a message to the selected server
- sending this message to the inbox of the selected server

A timeout method is used to avoide an infinite waiting loop

RPC communication

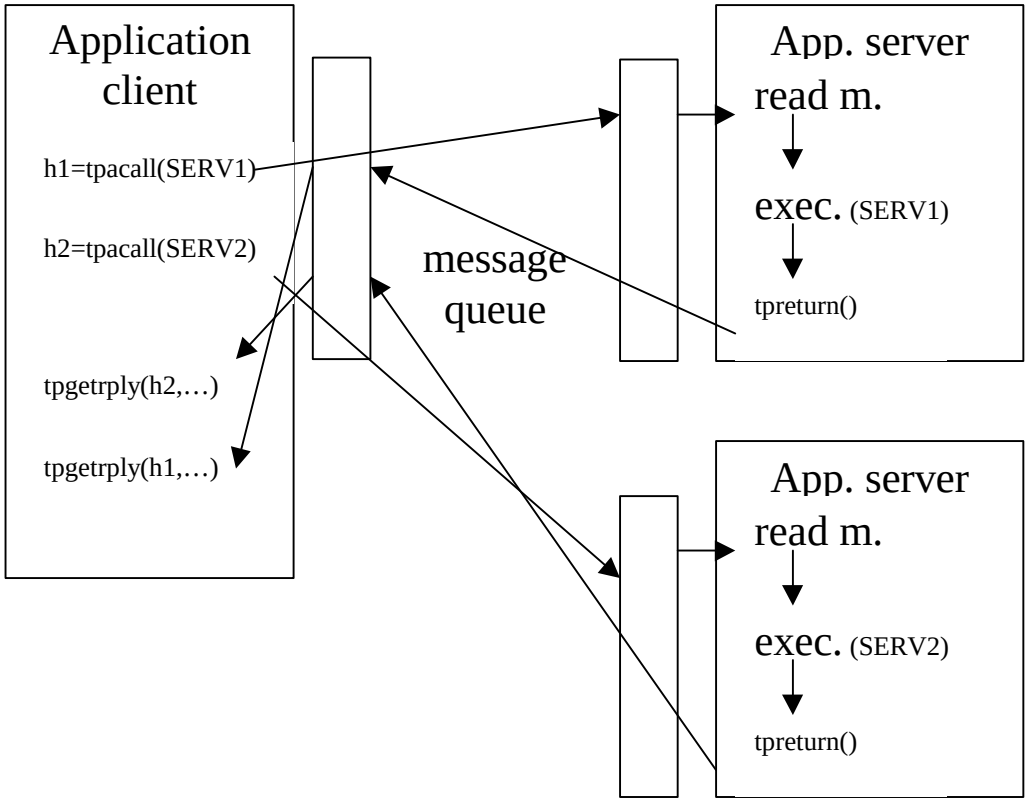
In synchron mode, the server processes only one service request at the same time.

The execution steps of a synchron tpacall call is similar to the asynchron case except the following aspects:

- In asynchron mode, the server processes only one service request at the same time.
- the clients can go on to work other parts of the program
- the clients should request the reply from the servers

RPC communication

The servers can forward the service request to an another server for processing



Message based communication

The client and server are making a conversation

The client can ask for information about execution details to control or fine tune the execution algorithm

The connection between the client and server is established by the tpconnect ATMI call

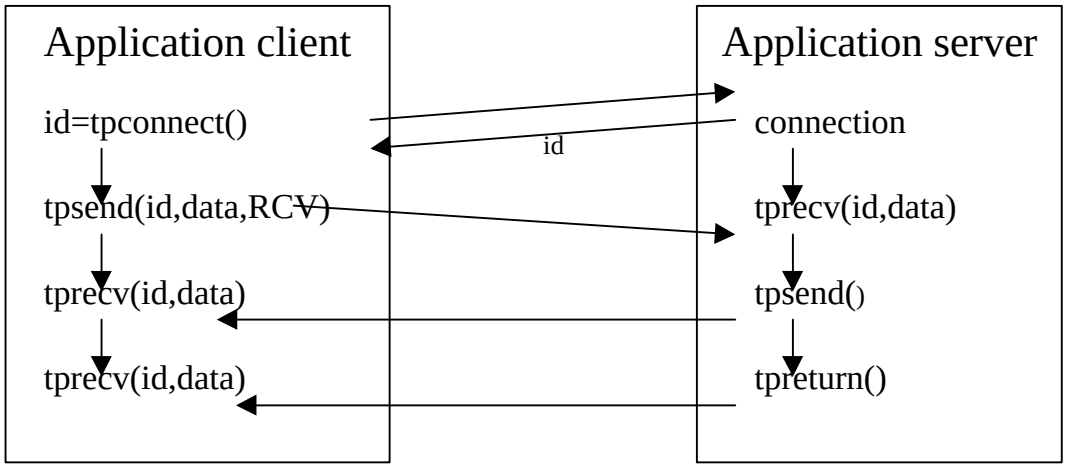
A client can have several connection at the same time

The reply message is stored in the message buffer

Message based communication

During the conversion, a flag is used to mark the speaking, active partner. This flag can be set in the messages sent by the active module.

The client can sand more than one request, message without checking for the reply message.



Communication buffer

The problem of message and data passing between the different components requires a common data and message format.

The communication buffer is used to store the data and message in a common format within the SYSTEM/T module.

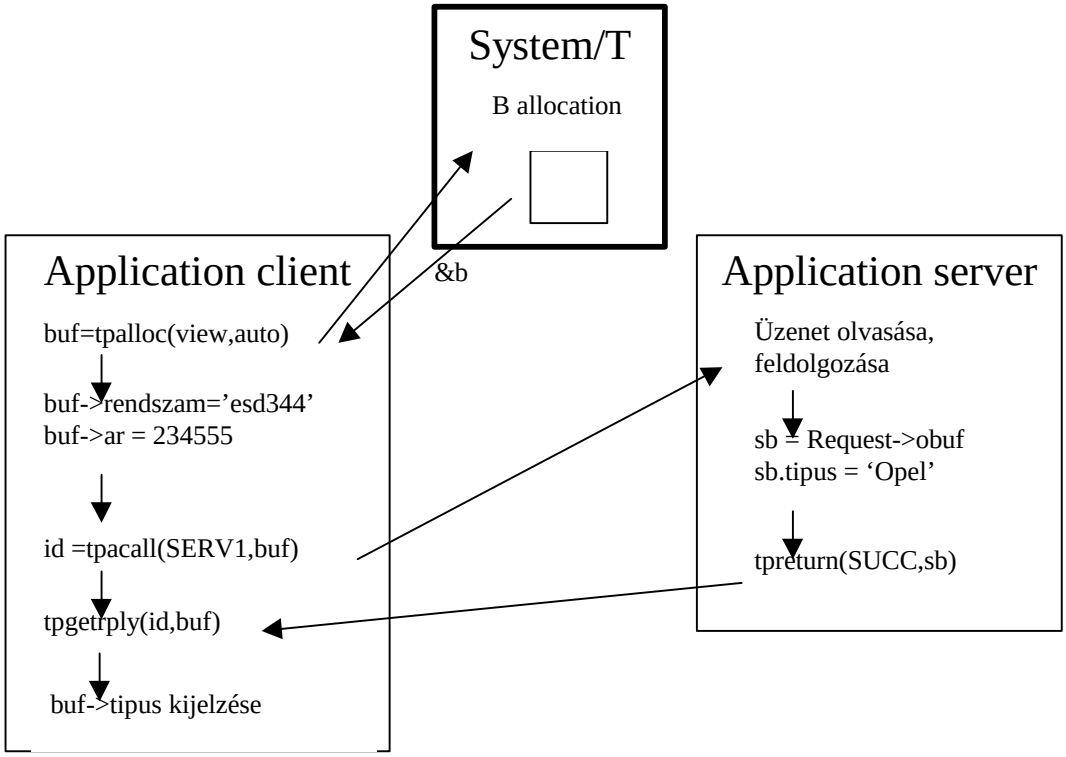
The communication buffer is allocated by the client using the tpalloc ATMI call.

The basic communication buffer types:

- string
- C array
- structure, view
- FML, flexible record structure

Communication buffer

The size of the communication buffer can be dynamic changed



Workload balancing

To achieve a balanced server workload, the actual workload for every server is kept in the Bulletin Board.

The calculation of the actual workload is based on the number of waiting services at the server.

Calculation method:

- real-time, exact value
- round robin, estimation value

To achieve a better approximation for the expected workload the different services may be assigned to different weights

The weights are proportional to to the average execution time

Real Time Method

The real time methos is used only when only one Bulletin Board exists in the system. This BB contains all information about all servers.

The calculation method:

- Initial workload value is equal to zero
- New incomming service increases the workload value by the weigth
- When the service request leaves the list, the workload value is decreased by the corresponding weight value

A new incomming service request is assigned by the SYSTEM/T to the server with the lowest workload value

Round Robin Method

If there are several Bulletin Boards in the system, the round robin method can be used.
All af the BBs stores the workload status of the every servers, but it takes only the local assignments into account to minimize the network traffic cost.

In some cases, the weight value is increased if the server is located at a remote node.

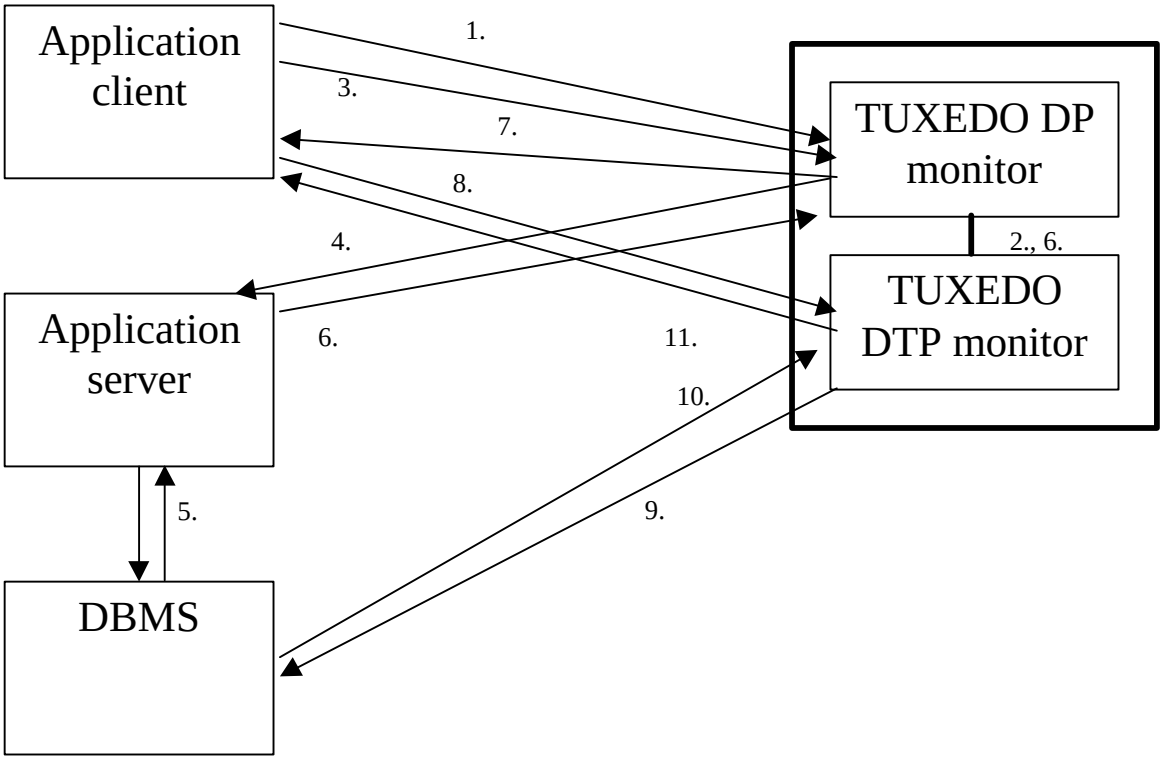
A new incomming service request is assigned by the SYSTEM/T to the server with the lowest workload value based on workload status in the local BB.

DTP System

DTP: Distributed Transaction Processing

The DTP is based on the two-phase commit protocoll

It communicates with the clients as with the data sources.
ATMI: interface to the clients
XA: interface to the data sources



XA function calls

xa_open	connecting to a datasource in the frame of a 2PC protocoll
xa_close	disconnect from the datasource
xa_start	starting a global t ransaction
xa_end	end of a global transaction
xa_prepare	set the datasource to a ’ready to commit’ state
xa_commit	commit a transaction
xa_rollback	rollback a transaction
xa_forget	release transaction description data
xa_recover	recover a failed transaction