SIEMENS

AX

AXCS Network

User Manual Basics

Version: 03

Last Change: 19.11.2010

F. Wicha
AX PLM-E ArC

Document	Doc-Type	Doc-Part	Doc-Version
10046981	ERS	005	03

History

Version	**Date**	**Author Department**	**Changes and Reason / Change Request / CHARM**	**Review-report**
00	17.11.2006	Wicha	First version, the base is the document 'AXCS for beginners'.	n.a.
01	20.07.2007	Wicha	only a new name and update of references	n.a.
02	26.04.2010	Wicha	Ported from FrameMaker to Word General update	n.a.
03	19.11.2010	Wicha	Handling of include files	n.a.

Table of Contents

1 Introduction

1.1 Purpose

This document describes the basis of AXCS. It gives an overview about the network, the communication and the interfaces.

It should help to get into the work with AXCS.

1.2 Scope

AXCS is the network with its interface as well as the protocol and data definitions. The network connects almost all components of an x-ray system. The base is a message passing mechanism within one computer system and on Ethernet between different computer systems.

The benefit is to use standard hardware and software for the communication.

The former versions of AXCS were ACS and XCS used in previous systems.

1.3 Definitions, Acronyms and Abbreviations

ACS	Angio Communication System (used in Polytron TOP systems)
ACSOS	axcs test tool and network monitor
ApId	Application Process Identifier of a logical AXCS unit, which has an own inlist. More than one ApId can be associated to one AxnServer
ARTIS	system family for angiography
AXCS	Angiography X-ray Communication System
Axn	AXCS network software.
AxnClient	software component or application with an own identifier (ApId)
AxnServer	server on each computer platform to connect the AxnClients to the network
INLIST	it defines all objects that the application wants to receive sent by the object INLIST_ENH or INLIST_RED
ARTIS	Angio family
QNX/Neutrino	real time operating system
XCS	Angio Communication System (used in Fluoro systems)

1.4 References

[1] AxnServer and AxnClient interface, Functional Specification (10095636 EPH 01S)

[2] ACSOS fuer NT, Pflichtenheft (5549550 EPH 21S)

and Requirement specification ACSOS version 3.x (10095631 ELH 000)
[3] Data View AXCS Data, this document is individual for each project and version

2 General Description

AXCS is a network that connects different applications running on the same or different subsystems.

AXCS comprises:

- the network, Ethernet in ARTIS and RFP systems
- the basic communication interface AXN (also called AXNROOT)
- the definition of data objects individual in each project and version
- the definition of protocols, data flows and communication rules

Generally, AXCS is independent of the physical network and the platform on which the application is running. However, in the actual projects AXCS runs on Ethernet.

The AXN communication software is available for several operation systems and targets. For more details, please see the release notes. The following list is only an example:

- WinNT
- WinXP
- WIN2000
- WinCE
- QNX
- Neutrino

AXCS is an open network. A new application can announce itself at the network and can right away communicate with all the other applications in the net. To focus the communication on the system there is a list of all possible applications in the project ARTIS. Only these applications are able to communicate using the axcs protocol.

AXCS is not architecturally structured. There is not any master from the network point of view. Only on application level, certain master functions are defined.

AXCS is mainly used

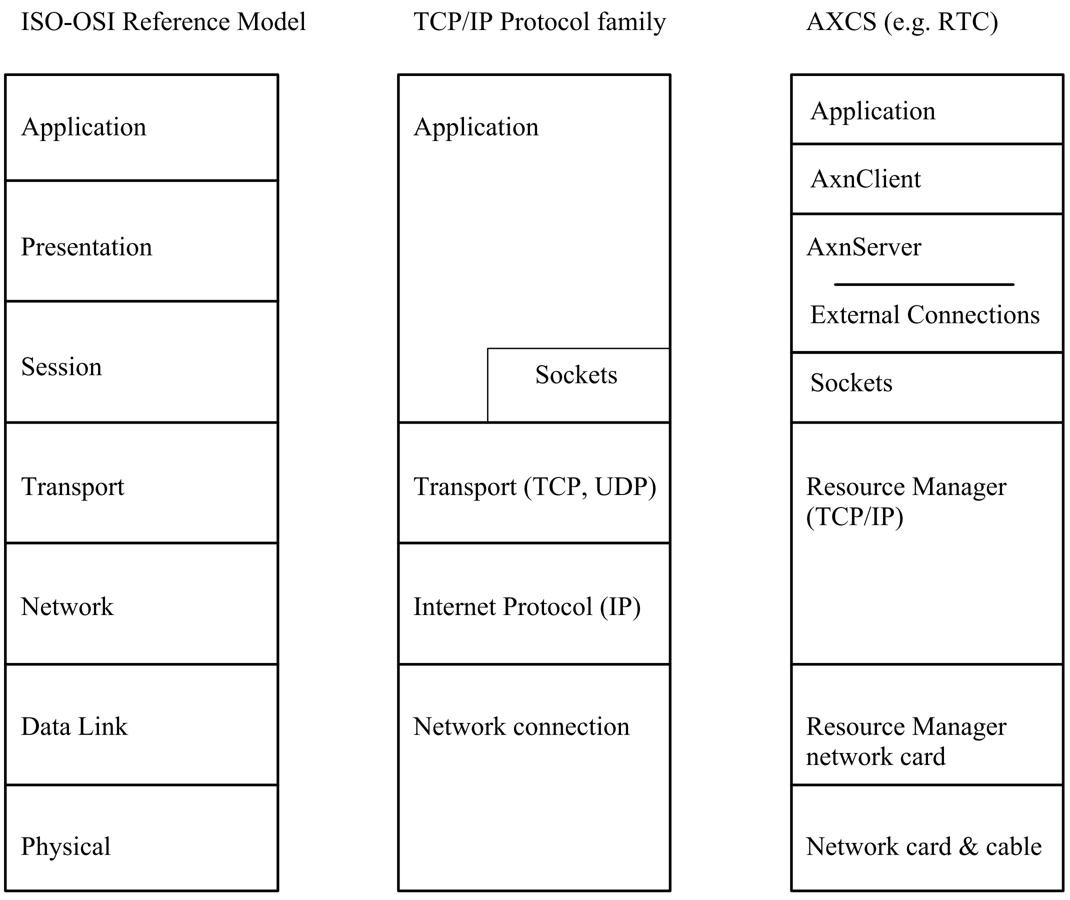
- to control all components in the system
- to distribute all parameters relating to x-ray
- to handle user interface events and data
- to service all components (inclusive software download)

To guarantee a high performance the AXCS network is not used to distribute large amounts of data (like images) during normal operation. Exceptions are possible in case of service or maintenance.

3 Network and basic communication

For more detail, see AxnServer and AxnClient interface, Functional Specification.

3.1 Protocol Layers

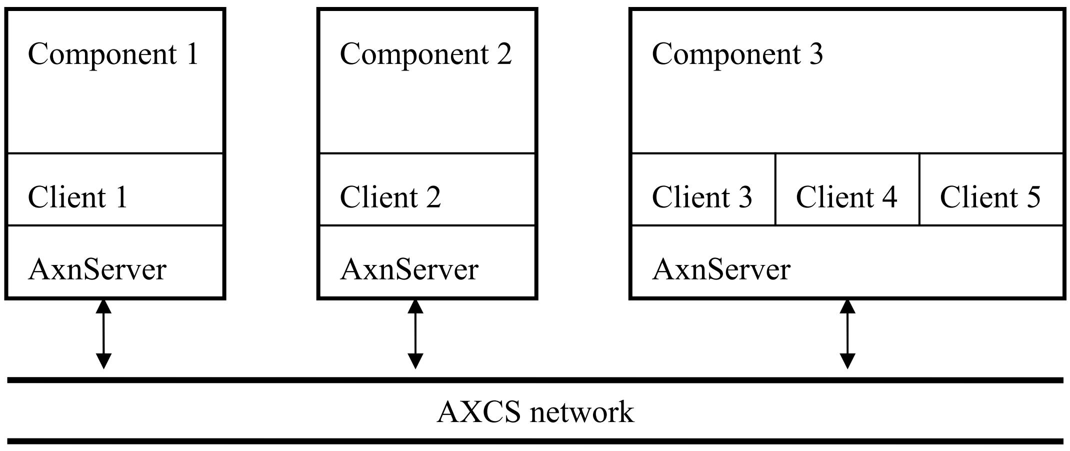


AXCS in the ISO/OSI-Model

All the layers below the application are also called AXN software. The network together with the lower application layers is called **AXCS network**.

In the projects ARTIS and RFP **Ethernet** with **TCP/IP** is used.

3.2 Topology



Basic local network with several systems

Different subsystems are connected to a local network. Each subsystem may contain one ore more applications. They can communicate with each other using AXCS. From the application point of view, it does not make any difference if they communicate internally or externally.

The AXCS functionality is represented by the so-called **AxnServer**. For the AXN software, a participant with an ApId is an **AxnClient**. Every physical machine can run several AxnClients but there is only one AxnServer. Several AxnClients and one AxnServer on one machine build a so-called virtual network. An AXCS network consists of several virtual networks.

The only way for an AxnClient to send an AXCS object to another AxnClient is, to send the object to the AxnServer, which forwards the object to the target AxnClient. Only AxnClients that are logged on to the AxnServer can receive objects.

AxnServer and AxnClient communicate via message queues.

3.3 AxnServer

An AxnServer is the interface to the axcs network. It is used to send and receive objects and to monitor the network with all the components.

The AxnServer is running in an own process and provides the following functionalities:

– Set up the basic communication in the network
– Pass objects to and from the AxnClients from/to the network
– Supervise the network and all AxnClients (alive check).

Each subsystem (that means computer system) has to have exactly one AxnServer with a unique identifier. See TCP/IP addresses.

The AxnServer provides the following 3methods:

- ConnectToServer is the initialization.
- SendObjToServer is used to send objects to the net. It can be used from different places of the application software.

- RecvObject is used to receive objects. It’s recommended to have a separate thread or process, that is calling RecvObject because
  - o it is blocked until an object is received or the time-out is over
  - o it has to receive the dummy messages if there is no other communication

3.4 AxnClient

An AxnClient is strictly speaking the application itself. However, here only the interface between the application and AxnServer is meant, which is provided with the AXN package and which is necessary to send and receive axcs objects.

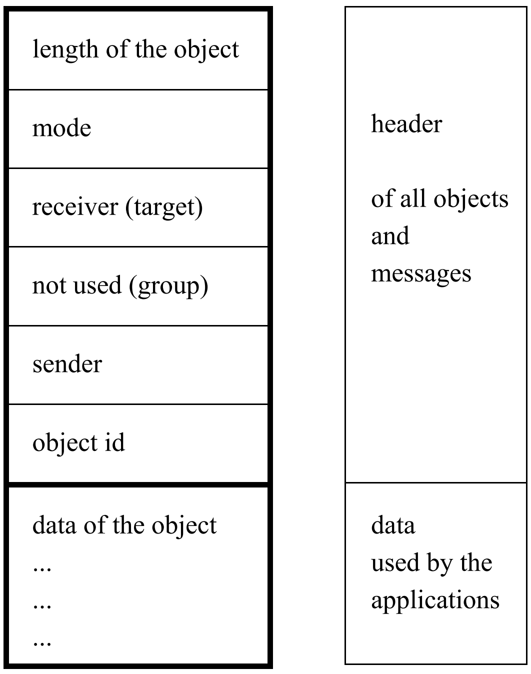
Each application has a unique ApId. See TCP/IP addresses.

3.5 AXCS Data Transfer

The axcs data transfer is based on data objects transferred over the network from one client to one or more other clients.

3.5.1 AXCS objects

Each object has a similar structure. It contains a header with a fixed structure and length and data structure with variable structure and length. The complete length of an object is 1460 bytes (until VB versions of the header file only 508 bytes).



General structure of an object

All axcs objects are described in [Functional Specification Data view AXCS data] of the corresponding project.

3.5.2 AXCS messages

An (AXCS) message is an object that is created and sent only by the AxnServer and received from one or more client. Messages contain information about the network itself. Messages should not be included in the INLIST. The clients receive them automatically if necessary.

All axcs objects are described in [[3] Data View AXCS Data, this document is individual for each project and version].

3.6 Addressing modes

Objects can be sent directly or functionally addressed.

- Directly addressed: The sender of an object determines the address of the receiver. Only this application gets the object.
- Functionally addressed: The sender of an object does not determine the address of the receiver and sends the object without any address to the network. All applications that want to receive this object get the object. The objects that should be received are defined in the inlist.
- Mode message: With this mode, the messages are sent from the AxnServer to the AxnClient.
- There are some more modes. However, they are only used internally by the AXN servers.

Objects can be sent and received only if the application is connected to the network.

The functional address mode should be preferred whenever it is possible.

3.7 Safety

There is an alive mechanism for

– the connection between AxnClient and AxnServer

– the connection between two AxnServer

Whenever there is not any communication between the applications for a certain time, dummy messages are sent between all the AxnServers and to the AxnClients. By this mechanism, the application is able to supervise the axcs network and the AxnServer.

The AxnServer is also able to supervise the application. If one dummy message is not taken out of the message queue, the application is automatically removed from the network.

To guarantee the data transfer each AxnServer is connected to each other by sockets (point-to-point connection), and TCP/IP is used.

3.8 TCP/IP addresses

We are using a class A network but with the network mask 255.255.255.0 and with a general address range from 10.1.1.1 to 10.1.1.254.

Within this range, we are using for the axcs communication the range 10.1.1.1 to 10.1.1.99.

The file broadcast.ini defines the used addresses.

However, we are also running other communications based on different protocols on this physical network within the range 10.1.1.100 to 10.1.1.254.

Each subsystem (that means computer system) has to have exactly one AxnServer. The identifiers are individually defined in each project, but all TCP/IP addresses and ApIds are harmonized over all projects.

4 Communication on application level

4.1 AXCS objects and parameters

All the objects and parameters are described individually in each project and version.

The data area of each object has a defined structure. A definition of the structure is available in ANSI C.

4.1.1 Data types and representation

Several types of data are defined. The value ’xyz_NIL’ is not considered in the range, but it is additionally valid for every data type. It means that the value does not exist now or it is not known.

All data types correspond to the Intel data format:

- all the data are byte orientated; one byte has 8 bit
- Bit-0 is the least significant bit. It is the most right one in this paper
- the low order byte is sent first, the high order byte last
- values can be displayed in (B) binary, (D) decimal or (H) hexadecimal.

All objects have

- to fulfill the byte alignment up to a 4 byte value

4.1.2 DUMMY bytes

To fulfill the byte alignment ’dummy bytes’ are added to the objects. The ’dummy byte’ has the type ’byte’ and the value ’0’. It is not allowed to evaluate this value; ACS_DNIL = 0 should be used. These dummy bytes are also used to extent the object without changing the length of the structure.

4.1.3 Objects with variable length

It is possible to repeat a part of an object (let us call it substructure) several times within an object. Each substructure is defined separately. A parameter of type DCount represents the number of repetitions. It has to be followed immediately by the substructure. The number of repetitions is zero to a max. value defined for each object. In this case, the object has a variable length.

This mechanism is also used to handle text string with variable length: The substructure consists of only one Unicode parameter.

4.1.4 Extension of objects

If it is necessary to add new parameters to an object they will be added somewhere in the

middle instead of dummies or at the end of an object. However, the last item means that an object gets longer.
This may lead to problems in the communication if different components run with different include files. Therefore, a component has to be able to receive an object even if it is longer than the structure defined in its own axcs.h include file.
If an object is shorter than expected the receiver has to provide an adequate error handling. The receiver may discard the object with or without an error message.

4.1.5 Version of objects

If a new parameter has to be added to an object, a dummy parameter is replaced if there is a dummy available. If not the new parameter is attached to the end. In this case, the object gets longer. To be compatible to future releases every component must be able to receive longer objects than defined in the axcs.h at compile time.

To make it easier to handle different versions of objects some object contains a parameter that defines its version. It has to be filled out by the sender of the object. The include file axcs.h defines the actual version for each of these objects. Whenever the object is changed, it will be increased automatically.

Example: Sending the object XYZ the component has to fill out the parameter 'version_xyz' in the following way:

e.g. ST_STPAR:	st_stpar.version_ststpar = ACS_V_STSTPAR;
e.g. SH_STPAR:	sh_stpar.version_shstpar = ACS_V_SHSTPAR;

4.1.6 Object representation

Some objects may be sent with different meanings depending on the situation:

- ACS_ACTUAL_SETTINGS
  These are actual values. They are normally sent cyclically (e.g. x-ray parameter during x-ray, parameter of the stand during movement, ...)
- ACS_PREVIEW
  These are forecast values or parameters that are expected for the next run, movement etc.
- ACS_REVIEW
  Something has been finished. These parameters show the final result (e.g. end position, sum of x-ray parameters after a run)

4.2 Simple data transfer

4.2.1 The Immediate Transfer

Immediate Transfer means the transmission of data without any synchronization or mechanism to provide consistency of data.

Getting an object, the receiver has to install the data immediately.

4.2.2 Request of objects

It is also possible to request an object from other applications. The objects that can be requested have to be defined individually for each system. They are defined in the documentation of the projects.

4.2.3 Use of OBJ_COUNT

If the amount of data may exceeds the maximum length of one object the object is provided with the parameter OBJ_COUNT. If necessary the object may be sent several times, and they are consecutively numbered by OBJ_COUNT. In the last object of a sequence, OBJ_COUNT is negated.

It is not sure that a component receives the objects in uprising order.

4.3 P-transfer (Artis only)

The protocol-transfer (p-transfer) is a predefined synchronized mechanism for distribution of data, which provides consistency of the data in the whole system at any time. It is additionally a mechanism to synchronize the data transfer with x-ray runs. P-transfers and x-ray are mutually synchronized.

For the p-transfer there are only virtual stations defined. The applications play these roles. An application may have more than one role.

- Initiator station: It is an application where something is initiated or operated.
- Target station: It's an application where something is set or displayed
- Arbitration master: This application synchronizes all p-transfers with the release of x-ray.
- Data master: This application distributes all data necessary for acquisition and fluoro.

It is not possible that there is more than one p-transfer at the same time. During a p-transfer, any additional AMR is answered by an AMA (busy).

All clients, which have a ST_object of a p-transfer in their inlist, have to send the corresponding SA-object, also if the ST-object does not contain data for this component. They need to have one buffer for each ST_object of a p-transfer (except for ST_PARAM, because two objects are send within the same p-transfer). The clients always have to store the last ST_object. Older objects with a different index can be replaced. With AMC (data valid) all temporarily stored ST_objects with the same index get valid, objects with a different index not.

If a component sends a SA-object (not ok) or if the data master sends a DMA (nok), the arbitration master completes the p-transfer with AMC (abort) and AMA (abort).

There is not any automatic repetition of a p-transfer, if a p-transfer is rejected. It has to be done by the application, which sends the AMR, if it is required.

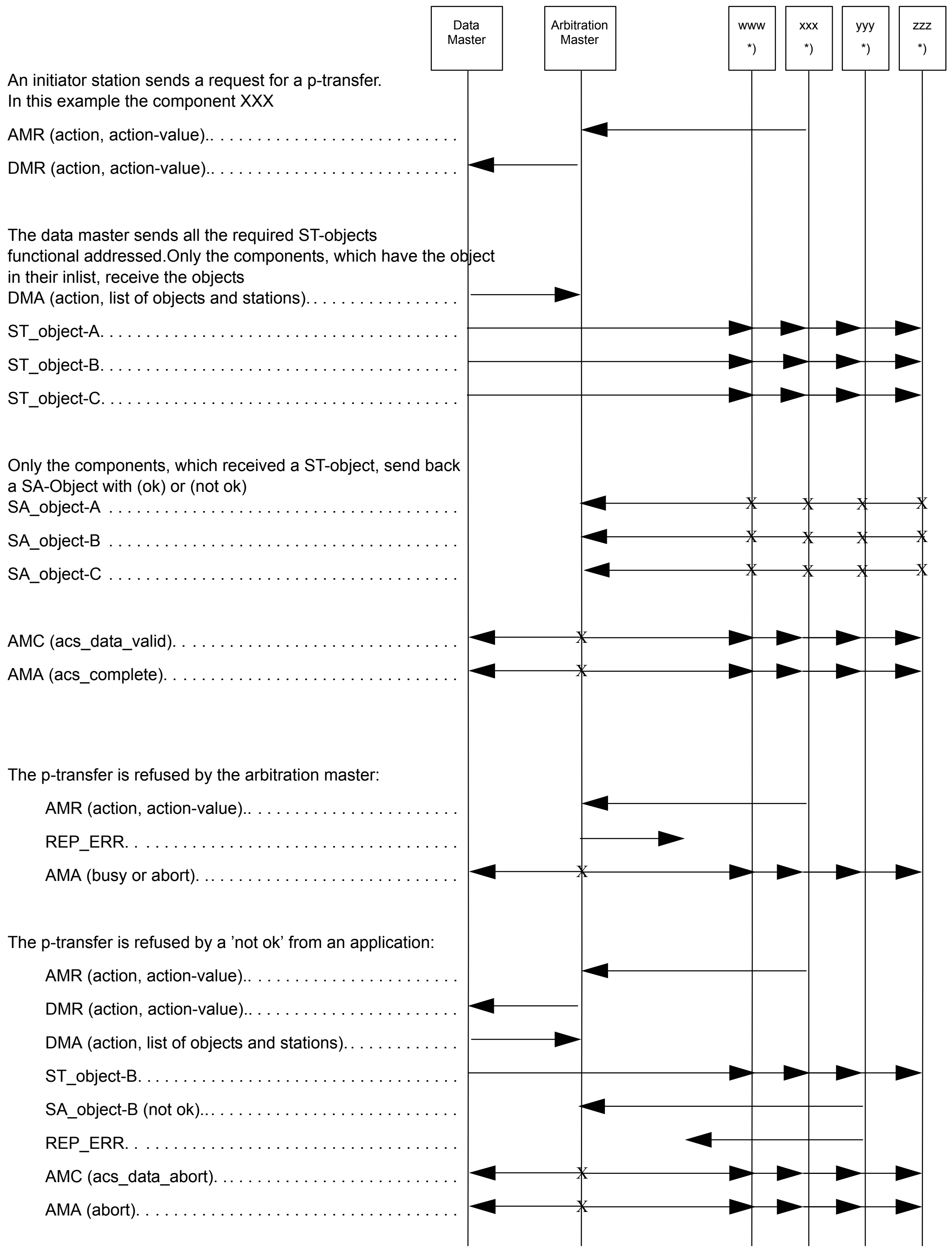
This is the general flow of a p-transfer:

- AMR is the request for a p-transfer. It is sent by the task that wants something; it is normally an UI event and it is only received by the arbitration master. It should not be necessary that any other component have to receive it.

- If a p-transfer is not allowed, the arbitration master sends an AMA (abort) and the p-transfer is finished. If a p-transfer is allowed the arbitration master defines the parameter IDX for the new p-transfer and sends a DMR.
- The data master receives DMR. It sends
  - DMA to the arbitration master containing a list of all SA_objects from all components. If a subsystem needs to evaluate different objects together it has to take the objects with the same index. If a subsystem needs to know which objects are sent together in the p-transfer it has to receive DMA. There is the parameter PTRANS_STOBJ. It is a 'bit set' and defines all parameters that will be sent in the p-transfer.
  - all ST_objects that should be installed in this p-transfer with STTYPE = ACS_CHANGE_BY_VALID and the parameter IDX.
- If a subsystem receives a ST_xyz object with STTYPE = ACS_CHANGE_BY_VALID and IDX between 1 and 254, it has to answer in any case with SA_xyz. If all the parameters are valid and acceptable it has to set ACK = ACS_SETTINGS_OK else ACK = ACS_SETTINGS_NOT_OK. The ST_objects have to be stored temporarily.
- The arbitration master receives the DMA and collects all SA-objects with this IDX.
- If the arbitration master has received the DMA and all the SA_objects listed in the DMA, it sends the AMC. If the arbitration master has send a DMR, it always sends an AMC too (if the p-transfer is valid or not).
- Every component which has received a ST_object, has to receive the AMC too:
- With AMC (valid), all the temporarily stored ST_objects with the same parameter IDX get valid. The target stations have to check this.
- With AMC, (abort) all the temporarily stored ST_objects with the same parameter IDX get not valid. They can be 'deleted'.
- AMA is the final answer to the AMR and sent after a delay (typically 150ms). Between AMC and AMA, the data in the system are inconsistent, because all the target stations have to make the currently distributed data valid. Therefore, the arbitration master delays the release of x-ray between AMC and AMA.
- The UI action is finished, successfully or not. AMA should be used by the task that has sent the AMR to finish its action. To make the software easier it is recommended to each component not to send a new AMR while an own p-transfer is running. Nevertheless, the arbitration master refuses a second one anyway.
- An AMA (complete) normally follows an AMC (valid). Only if the arbitration master gets in between a software error, it will send an AMA (abort). The target stations have to make the data valid. The UI should be unlocked (if it was locked). The UI should always take the data that get valid with AMC.
- An AMA(abort) always follows an AMC(abort)

Every object distributed within a p-transfer has the parameter IDX with the identical value differently defined by the arbitration master for each individual p-transfer.

4.3.1 General flowchart of a p-transfer (Artis only)



An initiator station sends a request for a p-transfer.
In this example the component XXX

AMR (action, action-value)..

DMR (action, action-value)..

The data master sends all the required ST-objects functional addressed.Only the components, which have the object in their inlist, receive the objects

DMA (action, list of objects and stations).

ST_object-A.

ST_object-B.

ST_object-C.

Only the components, which received a ST-object, send back a SA-Object with (ok) or (not ok)

SA_object-A

SA_object-B

SA_object-C

AMC (acs_data_valid).

AMA (acs_complete).

The p-transfer is refused by the arbitration master:

AMR (action, action-value)..

REP_ERR.

AMA (busy or abort).

The p-transfer is refused by a 'not ok' from an application:

AMR (action, action-value)..

DMR (action, action-value)..

DMA (action, list of objects and stations).

ST_object-B.

SA_object-B (not ok)..

REP_ERR.

AMC (acs_data_abort).

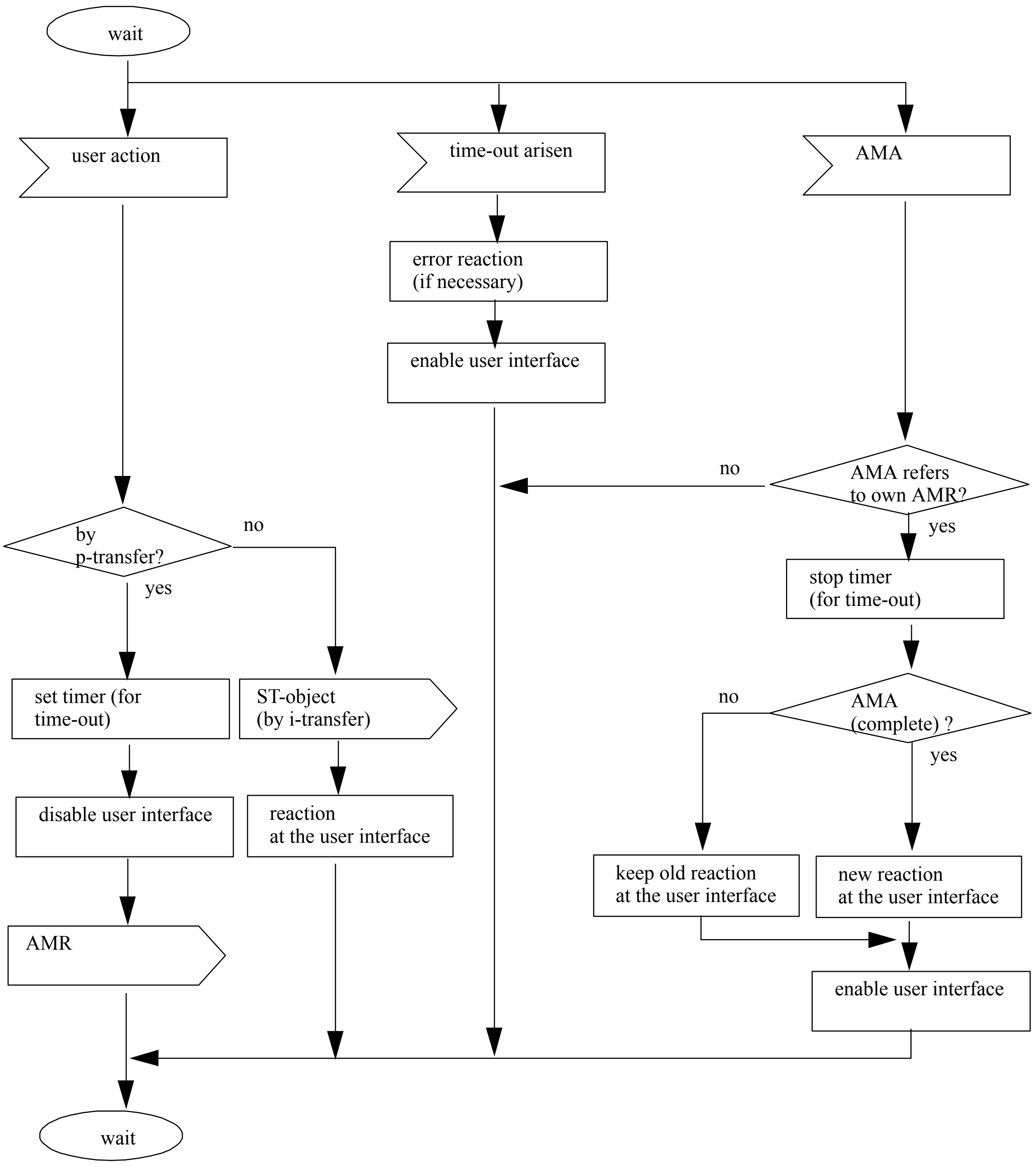
AMA (abort).

*) This means any other application

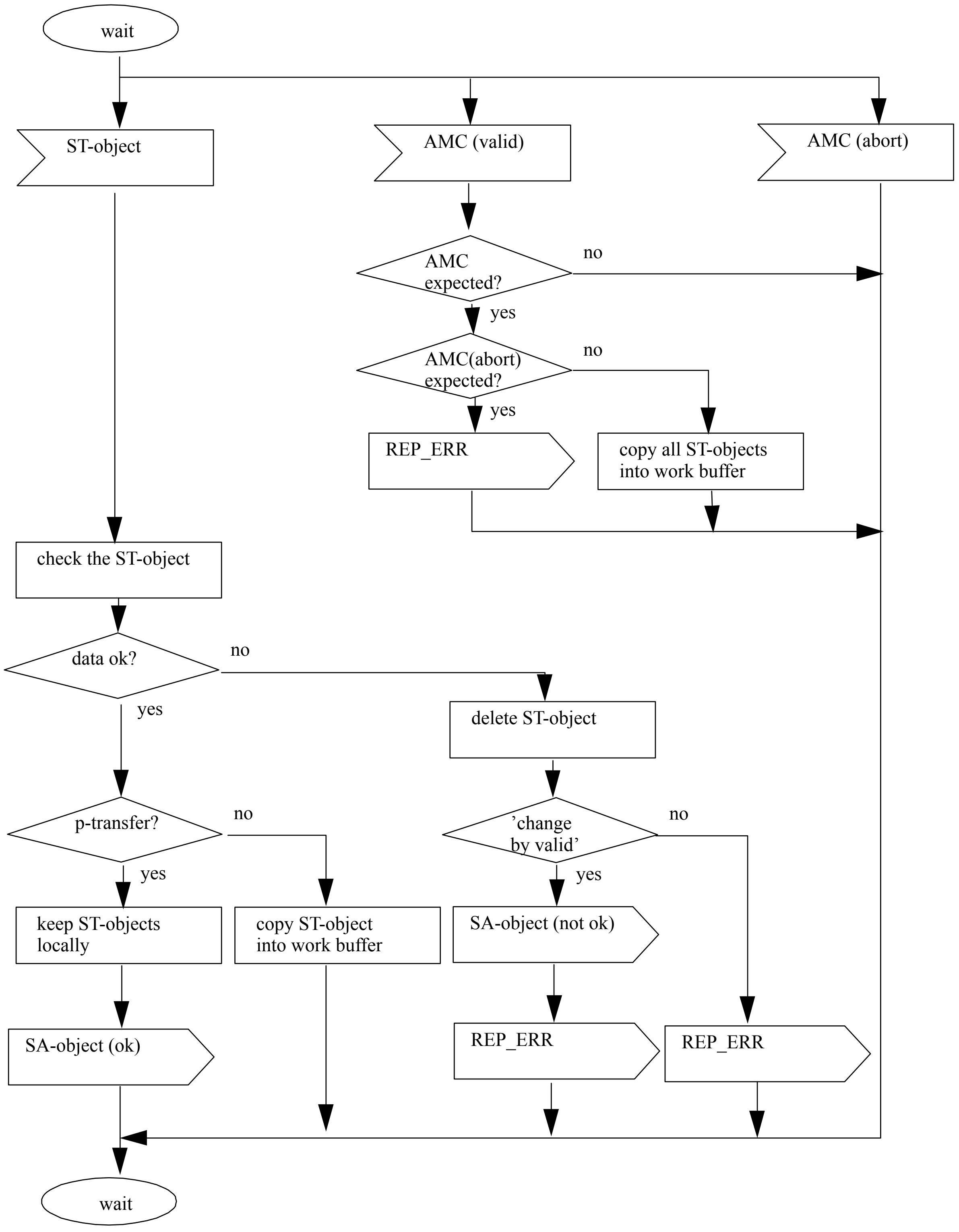
In AXIOM Artis the roles 'data master' and 'arbitration master' are implemented in the subsystem System Controller (SYC).

4.3.2 Roles in a p-transfer (Artis only)

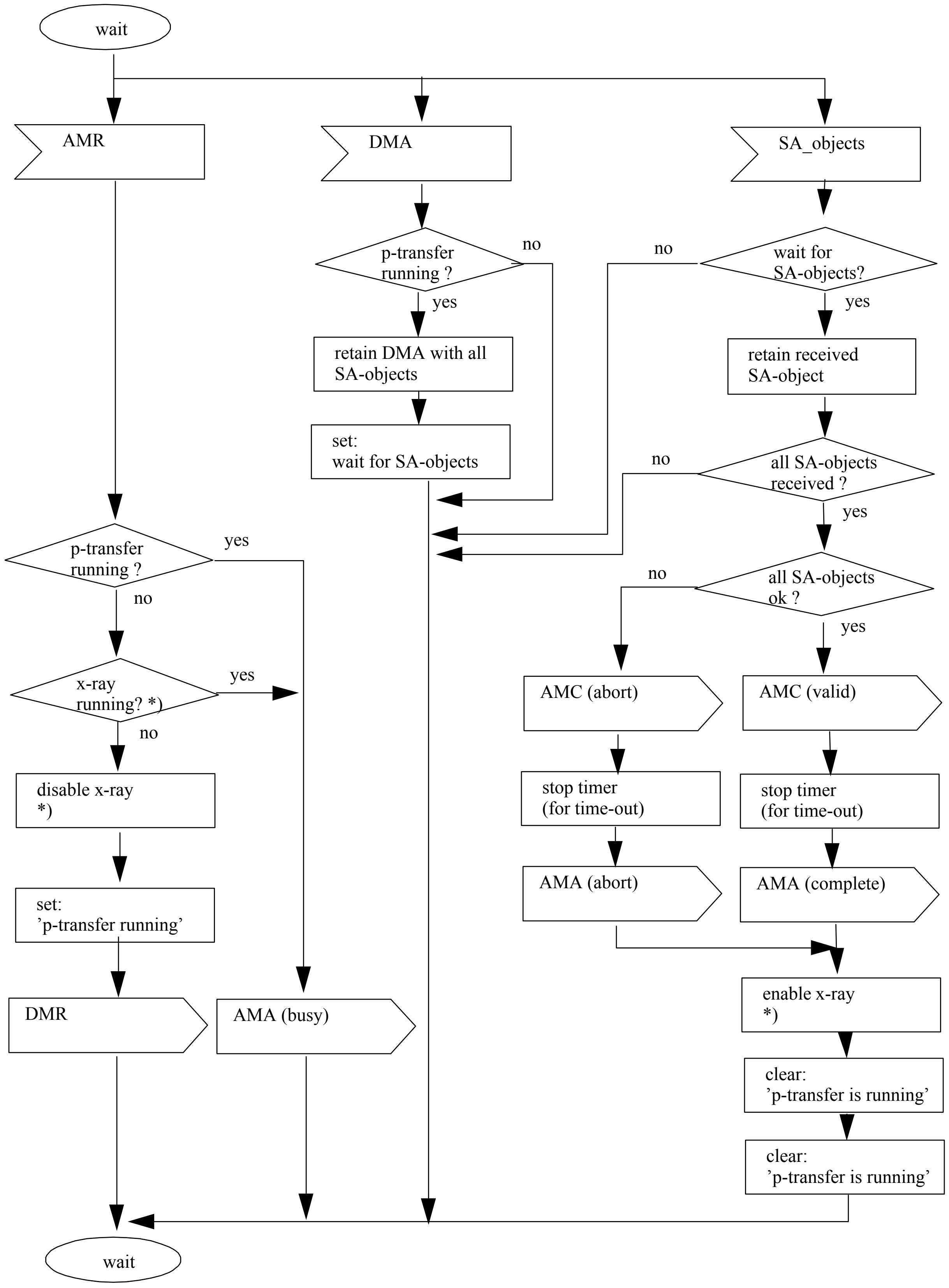
4.3.2.1 Initiator station



4.3.2.2 Target station

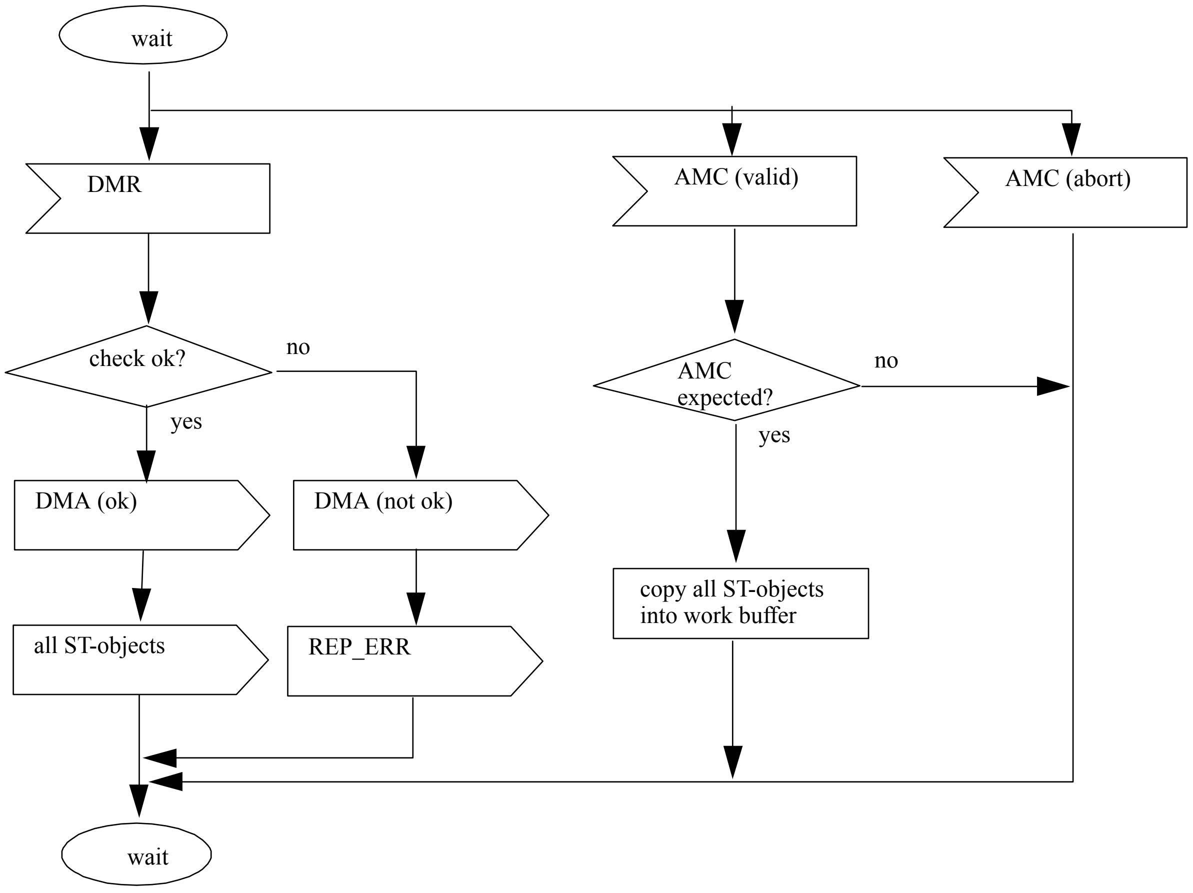


4.3.2.3 Arbitration master



***) Each system individually defines the mutual exclusion of x-ray and p-transfer. The whole diagram does not show any time-out handling**

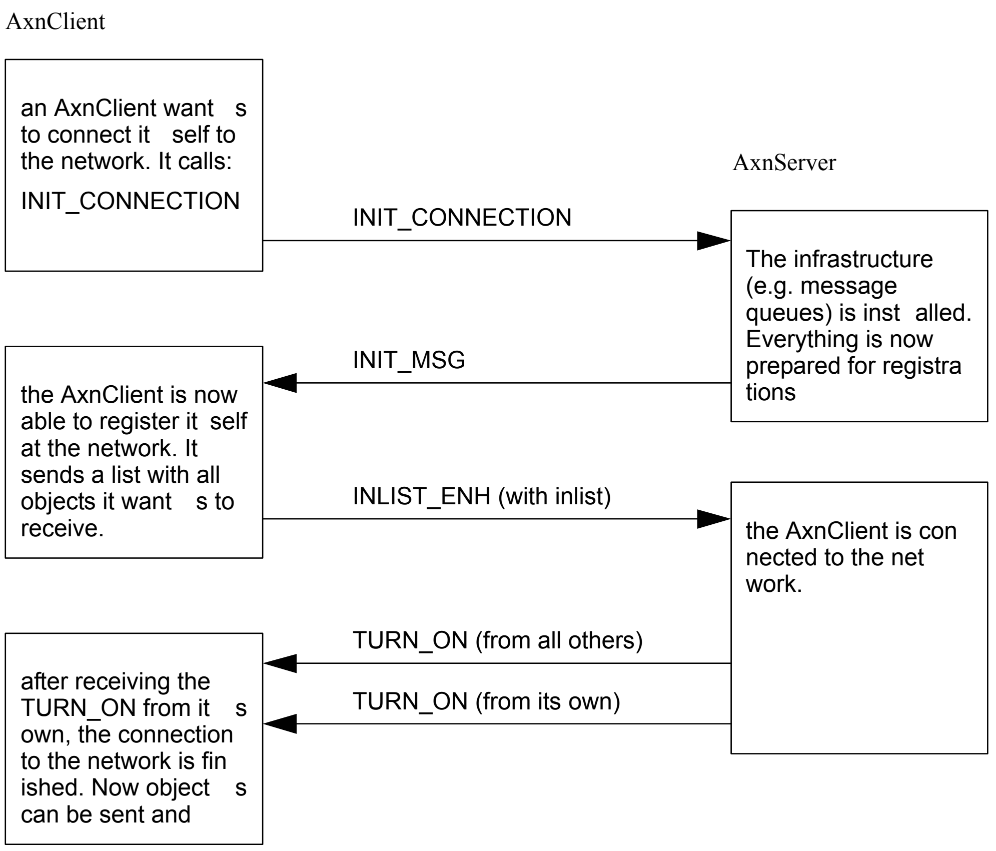
4.3.3 Data master



4.4 Network connection

By this mechanism, every AxnClient gets the information to which other AxnClient the communication can be performed.

4.4.1 Get connected to the AxnServer and network


AxnClient
an AxnClient want s to connect it self to the network. It calls:
INIT_CONNECTION
AxnServer
INIT_CONNECTION
The infrastructure (e.g. message queues) is inst alled. Everything is now prepared for registra tions
INIT_MSG
the AxnClient is now able to register it self at the network. It sends a list with all objects it want s to receive.
INLIST_ENH (with inlist)
the AxnClient is con nected to the net work.
TURN_ON (from all others)
TURN_ON (from its own)
after receiving the TURN_ON from it s own, the connection to the network is fin ished. Now object s can be sent and

Before an AxnClient (application) can receive objects, it has to be connected to the network. This is done in two steps.

First, the infrastructure has to be built up. The AxnClient sends INIT_CONNECTION to its AxnServer. When the communication path is ready, it gets back the INIT_MSG: Now an AxnClient can be connected to the network by sending INLIST_ENH. From this point, its own AxnServer knows the AxnClient and the objects it wants to receive. Moreover, the AxnServer hands on this information to all other AxnServers in the net, and all the others pass it over to their clients by TURN_ON. Therefore, every AxnClient knows that there is a new one connected. Moreover, the newly connected AxnClient gets from its AxnServer the TURN_ON objects from all the other AxnClients in the net too, and at least its own one.

There is a similar behavior if the connection between AxnServers is interrupted and again reconnected (e.g. if a cable between two servers is pulled out and plugged in again). The AxnServers are exchanging the information about there clients and all the AxnClients get the TURN_ON messages from the others.

4.4.2 Get disconnected from the AxnServer and network

An AxnClient (application) can disconnect itself from the network by sending the object INLIST_RED (total list) or the AxnServer is able to remove an AxnClient (see Safety). In both cases, a TURN_OFF is generated by AxnServer and sent to all other clients. If a cable is disconnected, the AxnServers at both sides generate these TURN_OFF objects of the lost connections and send them to their clients.

4.5 TURN_ON and TURN_OFF

Both objects are generated and sent by the server. They can be sent by the server running on your system or by any other server. They are representing the knowledge of the server(s) concerning the clients in the network.

If the server sends a TURN_ON it has a connection to the client indicated in this message. If the server sends a TURN_OFF it has lost the connection to the client indicated in this message. In both cases, the client can run on the same or different system.

A client establishes the connection to its server by sending INLIST_ENH. From this point, its own server knows the client. In addition, the server hands on this information to all other server in the net. In addition, all other servers pass it over to their clients by TURN_ON. Therefore, every client knows that there is a new one connected. Moreover, the newly connected client gets from its server the TURN_ON messages from all the other clients in the net too, and at least its own TURN_ON. If you reconnect a cable between two servers both are exchanging the information about there clients and all the clients get the TURN_ON messages from the others.

A client can disconnect itself from the net by sending the message INLIST_RED. In addition, if a client does not pick up its dummy or regular messages within the defined timeout (it is set with INIT_CONNECTION) it is removed from the net by its server. In both cases, a TURN_OFF is generated and sent to all other clients. If you disconnect a cable, the servers at both sides generate these TURN_OFF messages of the lost connections and send them to their clients. By this mechanism, every client gets the information to which client the communication can be performed.

5 How to start

The software consists of 3 part:

- AxnServer: Interface to the physical network and basic communication software
- AxnClient: That part of the application software that establish together with the methods of the Axn library the direct interface between the application and AxnServer.
- Application using axcs data and structures.

5.1 AxnServer

The following files are necessary to run the AxnServer. An installation is not necessary. Only the executable must be started.

• axnroot.cfg	TCP/IP address of the axcs net (optional, see below)
• broadcast.ini	List of all TCP/IP addresses of the axcs network
• axnroot.exe	Executable of the AxnServer

5.1.1 axnroot.cfg

The axnroot.exe takes the (first) TCP/IP address it finds in the system. If there is only one Ethernet channel the communication will work fine and the file axnroot.cfg is not really necessary.

If more than 2 Ethernet channels are available, the configuration file 'axnroot.cfg' is needed and has to contain the address that has to be used.

If the AXNROOT should run locally without any external connection, the configuration file 'axnroot.cfg' has to contain the string 'local’.

Notation: a.b.c.x (in Artis systems only 10.1.1.x) without CR/LF at the end of this line.

5.1.2 broadcast.ini

It contains a list of all TCP/IP addresses of the axcs network. Only these addresses can be reached by the communication.

Notation: a.b.c.x (in Artis and RFP systems only 10.1.1.x)

5.1.3 Start of AxnServer

The AxnServer may run as a process or a service (on systems with Windows). It does not matter if the AxnServer or the AxnClient is started first. However, normally the AxnServer runs first.

- If AxnServer is started without parameters, the files axcroot.exe, axnroot.cfg and braodcast.ini have to be on the same directory.
- Alternatively AxnServer may be started with parameters:

axnroot.exe [[path1\]broadcast.ini [-ip [path2\]axnroot.cfg]]
(path1/2 may be relative or absolute)

5.2 AxnClient

The interface, the implementation and examples are described in AxnServer and AxnClient interface, Functional Specification. These are only the main issues.

The following files are needed:

- Library with header files
- Include files for the application objects; they depend on the version of the product

5.2.1 Dynamic link library / shared object

The dynamic library is supported from Axn version VC14E.

The interface is implemented in C++ with different classes for the initialization and to send and to receive axcs objects. The following files are needed:

- Library linked together with the client; they belong to the version of the AXN server
  - libaxnclientif.dll dll
  - libaxnclientif.lib is additionally needed with WinXP even if the dll is used
  - libaxnclientif.so shared object (QNX')
  - debug versions also exist
- header files used by the client implementation; they depend on the version of AXN server
  - AxnClientIF.h
  - AxnClientIFCreator.h

5.2.2 Static library (no longer supported)

In the past (till Axn version VC12x), the interface between AxnClient and AxnServer was provided by a static library. The interface is implemented in C++ with different classes for the initialization and to send and to receive axcs objects. The following files are needed:

- Library linked together with the client; they belong to the version of the AXN server
  - libclientrecvLib.lib Library
  - (libclientrecvdLib.lib debug version of the library)
- header files used by the client implementation; they depend on the version of AXN server
  - libclientrecv.h
  - axnclientrecv.h
  - axnglobeldefines.h
  - axnportab.h
  - axnportabce.h

- axnportabnt.h
- axnportabnto.h

How to use the include files:

- The part of the application that uses the methods to send and/receive objects

  #include "axcscom.h"

  #include libclientrecv.h
- The part of the application that uses only the structures and definitions of the objects

  #include "axcscom.h"

5.2.3 Include files for the application objects

• axcs_typedef.h	this file defines only the types of data used in the axcs communication
• axcs.h	this file defines the structures of all axcs objects
• axcs_xyz	this files define the structures of all axcs objects used by a certain subsystem. It is only available on demand.
• axcscom.h	this is the combination of axcs_typedef.h and axcs.h (both are internally included).

5.3 How to use the include files

We have to distinguish between 2 cases with 2 possibilities:

- The application uses the common file with all definitions of all objects
  - The part of the application that uses the methods to send and/receive objects

    #include CAxnClientIFCreator.h

  - The part of the application that uses only the structures and definitions of the objects

    #include "axcscom.h"

- The application uses a specific include file for its objects
  - The part of the application that uses the methods to send and/receive objects

    #include axcs-typedef.h

    #include axcs-xyz.h

    #include CAxnClientIFCreator.h

  - The part of the application that uses only the structures and definitions of the objects

```
#include  axcs-typedef.h
#include  axcs-xyz.h
```

5.4 Additional include files

If these files are used the standard files have to be included before as described above.

• axcs_comp.h	This file contains the names of all subsystem as strings in ASCII and Unicode.
• axcs_obl.h	This file contains the structures and definitions of the "object description list" and the trace files of the tool ACSOS.

6 Tools

6.1 ACSOS network monitor

6.1.1 Introduction

ACSOS is a monitor and simulation tool for the axcs network. The main features are:

- All objects transferred on axcs can be stored in a trace independently of the sender or receiver.
- To simulate the behavior of a certain application only predefined objects can be stored in a trace.
- Every single object can be filled out and sent.
- Macros are supported:
  - a certain reaction can be defined
  - a sequence of objects can be defined
  - Events to trigger a macro are a key, a mouse click, an axcs object or a parameter of an object.
- The objects with all parameters are displayed with predefined names (the same names as in the documentation).
- All traces may be stored. External traces may be opened and evaluated.

6.1.2 Used files

These files belong to ACSOS:

• AcsosNT.exe	ACSOS executable
• AcsosNT.HLP	ACSOS help file

Additional application and user specific files:

• default.inl	an empty INLIST (normally used for ACSOS)
• default.mac	default macro file
• default.flt	default filter file

Files specific for the version of the project or system, in which ACSOS is used

• axcs.obl	the object description file
• station.cfg	list of all applications of the axcs network; may be redefined by the user

Every user may define its own files for the INLIST, the macros and the filters. They can be separately opened by ACSOS.

It is recommended that all these files are located in a separate directory.

Help menus (in English) are available. For more details see ACSOS fuer NT, Pflichtenheft (5549550 EPH 21S)
an.

6.1.3 Installation

ACSOS is able to run on every PC with Win2000, WinNT or WinXP (at least Pentium II with 200 MHz).

ACSOS must not be installed. It is sufficient to copy all the files onto the PC. However, a separate directory is recommended.

Additionally AXNROOT must be available on the PC. It is stored in the same folder ACSOS starts it automatically, if it is not yet running.

The IP address of the ACSOS PC should be 10.1.1.13 (or 10.1.1.14).

ACSOS can also be used to evaluate traces from other ACSOS stations without being connected to the axcs network. In this case, AXNROOT has to run locally. For more details, see readme-axnroot.txt

6.1.3.1 Till version 3.8

ACSOS stores its complete configuration in the registry. Therefore, additional inputs of the user are necessary the first time ACSOS is started. The details are described in the help file of ACSOS.

The actual version of axcs.obl can be shown with ACSOS. It is display in the name of a parameter of the pseudo object _GEN_DEFINITION.

6.1.3.2 From version 4.0

ACSOS stores its complete configuration in the file acsos.ini. If all necessary files are stored in the same folder or a subfolder, where the executable is stored, relative paths are used in the configuration file. Therefore, ACSOS can also run stored at a memory stick without further configuration. The details are described in the help file of ACSOS.

6.2 AXN Data Base

In this database, all objects with all parameters are defined and described. Out of this database, the following documents can be generated:

- a documentation
- an include file (ansi C) with the structures of all objects
- a description file for ACSOS with all structures and names

7 Tips and tricks

- A subsystem has to leave the emergency or service mode, if it receives AMC(ACS_DATA_VALID) with at least one ST_PARAM distribute in this p-transfer.
- Do only use symbolic constants (#define ...) of the header file, if you are really using the parameter of an object, where it is defined.

Version of this document template is V1.2 from 10-Mai-2007
Version dieser Documentvorlage ist V1.2 vom 10-Mai-2007