ACS AS A DEPENDABLE DISTRIBUTED SYSTEM[*]

K. Žagar[†], I. Verstovšek, G. Pajor, M. Šekoranja, Cosylab, Ljubljana, Slovenia
G. Chiozzi, ESO, München, Germany

Abstract

ACS (*Advanced Control System*) is a component-oriented infrastructure for distributed control systems. Components representing controlled devices or control logic can be deployed across host computers throughout the network. A central entity called the Manager is responsible for determining on which host a given component will reside. This centralized approach to deployment allows dynamic reconfiguration of the system, e.g., due to changes in requirements or as an automated response to failures within the system. In this article, the dependability and availability of an ACS based system is discussed, and future developments in the context of the European Union's 6th Framework project *Dependable Distributed Systems* (*DeDiSys*) are outlined.

INTRODUCTION

The key element for achieving scalable and maintainable distributed software systems is dependability, because otherwise the complexity of distribution would leave the system uncontrollable.

Data in distributed systems are not stored at a single location, nor is data processing performed by only one computer. Such an interconnected system is much more susceptible to failure than a non-distributed one: if only one of the many computers fails, or if a single network link is down, the system as a whole may become unavailable.

ACS

ACS is a set of application frameworks built on top of CORBA [1]. It provides a common software infrastructure for development of distributed control systems as well as other distributed applications.

Principal driver of ACS development is the Atacama Large Millimeter Array (ALMA), in the context of which the ACS acronym stands for *ALMA Common Software.*

The purpose of ACS is twofold:

- from a system perspective, it provides the implementation of a coherent set of design patterns and services,
- from the perspective of a developer, it provides a friendly programming environment in which the complexity of the CORBA middleware and other libraries is hidden and coding is drastically reduced.

[*] The work described herein is supported by the European Community under Framework Programme 6 of the IST Programme within the project Dependable Distributed Systems (DeDiSys, Contract No. 004152)

[†] klemen.zagar@cosylab.com

The ACS allows development in C++, Java and Python. This gives developers enough flexibility to choose the right tools for the right tasks – the code that interacts with the hardware is typically written in C++, user-interfaces and high-level applications are covered by Java whereas Python is useful for quick *ad hoc* scripting, testing and prototyping.

Deployment Management

One of the core elements of ACS is a *Component-Container Model* of deployment (Figure 1). In this model, the application-specific code is packaged in *components.* Physically, components are shared libraries or similar dynamically loadable code, such as Java classes and Python scripts.

Components typically implement one or more interfaces through which other components and clients (user-interfaces, scripts, ...) interact with them. The interface is defined using CORBA's *Interface Definition Language* (IDL).

Container is a process that hosts components. Physically, the container is an executable that provides environment to its components. It also directly manages component's lifecycle (loading of code, activation, destruction and unloading of code).

ACS features a *Manager,* which is essentially a broker or a naming service that issues clients references to components they request. The Manager communicates with containers to perform the following tasks:

- Manager instructs the container to activate a component. The Manager specifies the code that will have to be loaded (e.g., the name of the dynamic library) and the name of the component. Container loads the code, creates an instance of the component, and handles all the complex overhead activities, such as making the component accessible through CORBA. The container also ensures that the component receives its configuration data from the CDB.
- Use a heart-beat to periodically check the health of a container.

In ACS, the deployment of components into containers is configurable through the ACS' *Configuration Database* (CDB).

Fault Management

So far, ACS made a limited effort in the direction of detecting faults and mitigating their negative effects.

Every container is capable of self diagnostics. When a container detects a critical error condition, it tries to notify

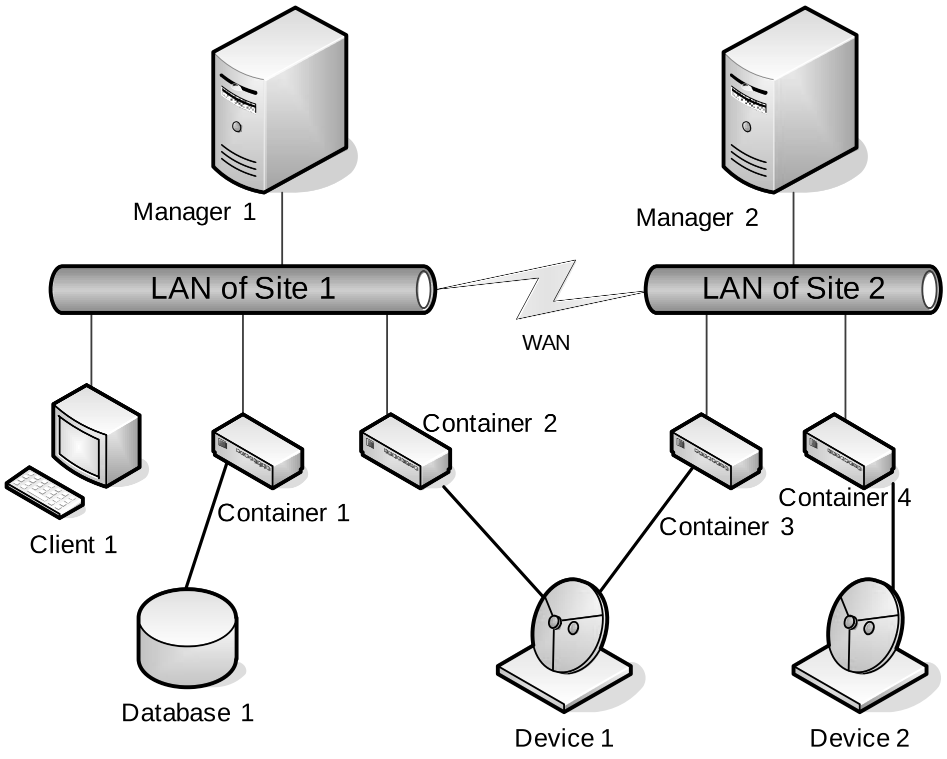


Figure 1: An example deployment of an ACS-based system. Every site has its own Manager responsible for supervision of containers. Containers host components representing devices attached to the host computers.

the Manager and restarts. This allows the Manager to notify the clients of the components hosted by the affected container, and also to activate these components at a later time.

A more frequent failure is when the container can not notify the Manager of its condition, e.g., when its host computer crashes, or when the network link is down. This class of failures is detected by the Manager's heart-beat mechanism: upon receiving no reply during a repeated heart-beat, the Manager assumes that the container, and consequentially all of its components, are unavailable.

Recovery scenarios in the case of node failures and link failures may be different (Figure 2). For example, if only a network link had failed, the Manager need not take any action. However, if a node has crashed, it could be more benefical to activate the crashed components in other containers, thus increasing availability. Unfortunately, link and node failures can not be reliably distinguished yet.

ACS Users

ACS is used in various distributed control applications as well as in the field of *Geographical Information Systems* (GIS). Current applications include:

- The control system of the *Angströmquelle Karlsruhe* (ANKA) synchrotron light source [4].
- The *Atacama Large Millimeter Array* (ALMA), under development by *European Southern Observatory* (ESO) and *National Radio Astronomy Observatory* (NRAO) [2].
- The *Atacama Pathfinder Experiment* (APEX) of the *Max-Planck-Institut für Radioastronomie.*
- The *1.5m Hexapod Telescope* of the *Ruhr-University Bochum.*
- Cosylab's GIS framework *Giselle*, with an application deployed at the Slovenian *Ministry for Agriculture, Forestry and Food.*

DEDISYS

DeDiSys [5] is a European research project with the focus on dependability of distributed systems in order to improve availability, reliability, and safety. The project focuses on a highly innovative method – trading consistency against availability – as the means to enhance dependability in the presence of node and link failures.

There are 8 members in the project consortium: Vienna University of Technology (Austria), Wroclaw University of Technology (Poland), Linköpings Universitet (Sweden), Universidad Politecnica de Valencia (Spain), Frequentis (Austria), Etra (Spain), Cosylab (Slovenia) and XLAB (Slovenia). The project started in September 2004, and is scheduled to end in July 2007. At the time of this writing, the requirements analysis phase of the project is nearing completion.

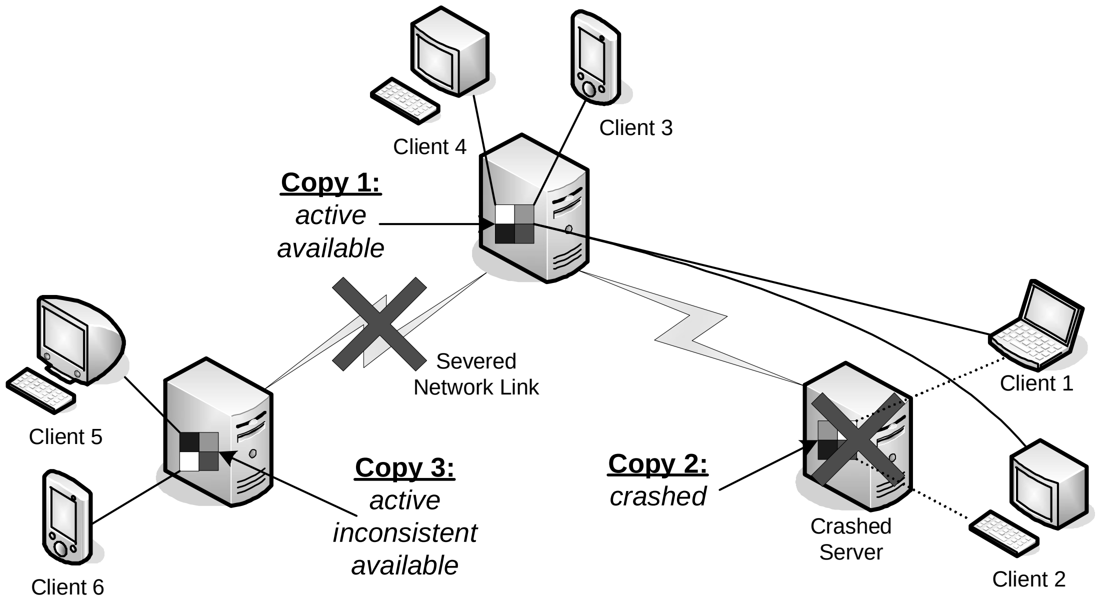


Figure 2: Clients 1 and 2 could switch to using copy 1 if their dedicated server crashes. If a network link is down, copy 3 is still available, but it could become inconsistent.

Consistency/Availability Trade-Off

The most commonly used approach to improve availability is to replicate services and data to several locations in the network, making at least one copy available while failures are present.

However, this introduces additional issues:

- How to ensure that modifications of the data are propagated to all replicas?
- How to minimize deteriorating impacts on performance due to such widespread updating?

And what if update propagation can not succeed due to the presence of failures?

Simply disallowing modification would reduce availability, whereas allowing them would involve a risk of introducing inconsistencies among replicas. The aim of the Dependable Distributed Systems (DeDiSys) project is to investigate the possible optimum between the two extremes (Figure 3):

- can one give up some of the data and service integrity constraints, and
- yield improved availability of the distributed system in return?

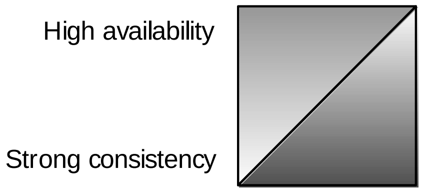


Figure 3: Depiction of the trade-off between consistency or availability: one can be gained only at the expense of the other.

PLANNED ACS IMPROVEMENTS

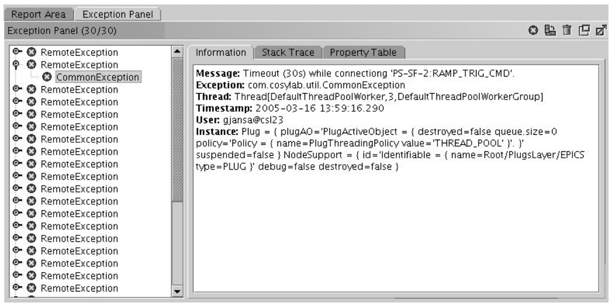
Alarm System and Diagnostics

According to a survey of existing control system users and operators peformed in the context of the DeDiSys project, one of the most important tools that would facilitate operations and improve availability is detailed diagnostics that would accurately and quickly pin-point the fault 4.

ACS already features a distributed *logging system* which allows for high-performance, reliable, asynchronous delivery of log messages to one or more interested parties (e.g., a relational database). The logging system allows for accurate pin-pointing of a problem, but analysis of logs is both time-consuming and requires in-depth knowledge of the workings of the control system and its parts.

Another useful tool for fault detection is the *alarm system*. ACS currently has a synchronous implementation of the alarm system, which requires interested clients to connect to all possible alarm sources. Theory forecasts that such approach does not scale, and practice has confirmed this to be true.

The solution would be a cross-over of the above: an alarm system that would deliver alarm notifications asynchronously using a publisher-subscriber paradigm. The development of such a system is currently nearing completion at CERN [6], and there is an ongoing investigation into whether that same solution would also be applicable for the ACS alarm system.



```
RemoteException
RemoteException
    CommonException
RemoteException
RemoteException
RemoteException
RemoteException
RemoteException
RemoteException
RemoteException
RemoteException
RemoteException
RemoteException
RemoteException
RemoteException
RemoteException
RemoteException
RemoteException
RemoteException
RemoteException
```

```
Message: Timeout (30s) while connectiong 'PS-SF-2:RAMP_TRIG_CMD'.
Exception: com.cosylab.util.CommonException
Thread: Thread[DefaultThreadPoolWorker,3,DefaultThreadPoolWorkerGroup]
Timestamp: 2005-03-16 13:59:16.290
User: gjansa@csl23
Instance: Plug = { plugAO='PlugActiveObject = { destroyed=false queue.size=0
policy='Policy = { name=PlugThreadingPolicy value='THREAD_POOL' }'. }'
suspended=false } NodeSupport = { id='Identifiable = { name=Root/PlugsLayer/EPICS
type=PLUG }' debug=false destroyed=false }
```

Figure 4: A screen-shot of Abeans [7] exception panel, which shows detailed information about a failure in the system.

Manager Replication

Currently, the ACS' Manager is a *single point of failure* – if the Manager becomes unavailable, no client will be able to retrieve a reference to a component it requires, effectively rendering all subsequently requested components unavailable.

Fortunately, the Manager is not required much during operation because most references are established upon strartup of the distributed control system. Therefore, it is very likely that a temporary unavailability of the Manager will not have any negative effects.

Manager maintains rich information regarding deployment state. This information must not get out-of-sync with the actual state of deployment, as otherwise components could be created more than once or destroyed through ACS' distributed garbage collection pre-maturely. Currently, the Manager handles this issue by keeping a transaction log of all the operations on the deployment state, which it can replay during recovery from a crash.

The next step would be to allow for a secondary Manager. Whenever the deployment state of the primary Manager would change, the primary Manager would not only persist the transaction to persistant storage, but also transmit it to the secondary Manager. The method of transmission (synchronous/asynchronous) is yet to be determined: choosing the first could cause unavailability of the primary Manager due to secondary Manager's failure, and choosing the second could result in inconsistancies of deployment state.

Component Replication

Component replication is either a very easy or very difficult task, depending on whether components have state that needs to be synchronized among replicas (*state-full components*) or not (*state-less components*).

Benefits of component replication can be two-fold:

- Resilience to faults. If one component fails, one of its replicas could take over.
- Load balancing. The replicas utilize resources of several machines.

The case with the state-less components is already supported by ACS and used in practice for over a year in GIS applications. There, the Manager may be configured with a strategy that helps the Manager decide in which container to host a component. For example, the strategy might keep track of CPU utilization of container's hosts, and always decide to activate a component in the container where most resources are available.

CONCLUSION

Douglas Adams once wrote: "*The major difference between a thing that might go wrong and a thing that cannot possibly go wrong is when a thing that cannot possibly go wrong goes wrong it usually turns out to be impossible to get at or repair*" [8].

Following this advice, distributed control systems must anticipate failures, and assist as much as possible to either circumvent failures automatically, or at least help operators to reduce the down-time.

ACS already provides some features that make it easier to survive through failures (e.g., recovery of the Manager, containers under heart-beat surveillance, ...). In the next few years, we intend to further improve on this through experience of others and incorporation of DeDiSys project's research findings.

REFERENCES

[1] G. Chiozzi et al., "The ALMA common software: a developer friendly CORBA-based framework", SPIE Astronomical Telescopes and Instrumentation, July 2004, Glasgow

[2] G. Chiozzi et al., "The ALMA Common Software (ACS): Status and Developments", ICALEPCS 2003, October 2003, Gyeongju, South Korea

[3] K. Žagar et al., "ACS – Overview of Technical Features", ICALEPCS 2003, October 2003, Gyeongju, South Korea

[4] I. Kriznar et al., "The Upgrade of the ANKA Control System to ACS (Advanced Control System)", ICALEPCS 2003, October 2003, Gyeongju, South Korea

[5] DeDiSys consortium, "Dependable Distributed Systems", European Community's Framework Programme 6 project (IST Programme), `http://www.dedisys.org`

[6] M. W. Tyrrell et al., "Moving Towards a Common Alarm Service for the LHC Era", ICALEPCS 2003, October 2003, Gyeongju, South Korea

[7] I. Verstovsek et al., "Abeans: Application Development Framework for Java", ICALEPCS 2003, October 2003, Gyeongju, South Korea

[8] Douglas Adams, "Mostly Harmless", Henemann, London 1992