A COMPONENT−BASED AUTOMATION ARCHITECTURE FOR CONTINUOUS PROCESS INDUSTRIES

Guido Urdaneta[1], Juan Colmenares[1], Nelson Arapé[1], Carlos Arévalo[2],
Néstor Queipo[1], Jorge Villalobos[1] and Seida Angel[3]

[1]Applied Computing Institute. University of Zulia. Venezuela.
[2]Novatek Soluciones Tecnológicas, C.A. Venezuela.
[3]Intesa. Venezuela.

ABSTRACT

This paper discusses the design of an automation architecture for continuous process industries based on the Enterprise JavaBeans server−side component model. This architecture has been designed considering criteria such as interoperability, portability, scalability, performance, reliability, security, use of legacy systems and mantainability. The Automation Integrated Environment (AIE) architecture includes the following elements: i) primary data sources, ii) real time data sources, iii) high level data sources, iv) primary event generators, v) server−side applications and vi) client applications. The proposed architecture satisfied the aforementioned criteria and holds promise to be effective in the automation of continuous process industries.

KEYWORDS

Keywords: industrial automation systems, continuous process industries, integration, distributed component model

1. Introduction

Automation systems in continuous process industries (e.g. oil & gas, water/wastewater and power & light) are characterized by having several different SCADA systems, historical databases, corporate databases and other special purpose applications. The evolution of these systems is usually not governed by a standard reference architecture, which results in the following difficulties: i) many of the products use proprietary technology, thus limiting interoperability and portability ii) maintenance problems (e.g. changes in the structure of a database affects many applications), iii) inefficient resource management and iv) incoherent security policies. This situation leads to delays and mistakes in the decision making processes, resulting in considerable losses of labor, time and money.

Typically, continuous process industries have dealt with this problem using techniques such as concentration of data from several information systems in common databases, use of standard APIs and protocols for some operations (e.g. OLE for Process Contro[1]) and use of message oriented middleware or transaction processing monitors for application integration and resource management. These approaches only solve the problem partially since they do not cover all the requirements, such as data abstraction, interoperability and maintainability, among others.

This paper presents an automation architecture for continuous process industries based on the Enterprise JavaBeans server−side component model. This architecture has been designed considering criteria such as interoperability, portability, scalability, performance, reliability, security, use of legacy systems and mantainability. The architecture includes the following elements: i) primary data sources, ii) real time data sources, iii) high level data sources, iv) primary event generators, v) server−side applications and vi) client applications. The remaining sections of the paper discuss the design criteria (Section 2), an overview of the architecture (Section 3), the architecture elements (Section4) and some conclusions and recommendations (Section 5).

  (Refereed-2001)

2. Design criteria

The design of an Automation Integrated Environment (AIE) requires the definition of the system architecture and the selection of the implementation technology. Such tasks demand the existence of certain criteria that ensure the fitness of this environment to the continuous process industries. The criteria imposed on the AIE are:

Interoperability and Portability

It must be based on open and widely supported standards, so that interoperability exists at the following levels:

- It must be possible to choose between several system software and hardware options (e.g. computers, operating systems, database management systems and middleware) so that the user can select the products that best meet its particular requirements.
- It must be possible to incorporate or exchange applications (developed in house or by a third party) without affecting other parts of the system.

Scalability and performance

It must provide a performance level that satisfies all the requests within reasonable response times. Additionally, this platform must be able to increase its capacity without modifying its software by means of improving the underlying hardware.

Reliability

The hardware and software platform of the AIE must provide high availability. This means that the system must ensure the continuous execution of mission critical applications using techniques such as automatic failover and clustering.

Security

The system must include authentication and authorization mechanisms and encrypted communication channels. It must be possible to activate these mechanisms programatically and declaratively. In addition, it is necessary to complement these services with measures concerning network and physical security (e.g. firewalls and restricted access rooms).

Use of legacy systems

It must facilitate the integration of legacy systems, thus preserving previous investments made by the user.

Maintainability

Applications should be developed in terms of business functionality instead of the specific characteristics of the underlying technology. This eases maintenance activities such as bug fixing (corrective maintenance), adaptation to new technology platforms (adaptive maintenance) and improvements to existing applications (perfective maintenance).

3. Overview of the architecture

A technology that has the potential to satisfy the previously mentioned criteria is Component Transaction Monitors (CTM) [2]. CTMs implement robust server−side component models that make it easier for developers to create, use, and deploy business systems. This technology combines the benefits of distributed objects and transaction processing monitors. It provides an infrastructure that can automatically manage resources, transactions, object distribution, concurrency, security and persistence.

The proposed AIE architecture is currently based on J2EE (Java 2 Enterprise Edition)[3] which is a de facto standard[1]. The core of J2EE is the Enterprise JavaBeans component model (EJB). It specifies the following component types:

- *Stateless Session Beans*: non persistent components that implement business processes without conversational

1This architecture is also being defined in terms of the CORBA Component Model (CCM)[4]. CCM is a superset of EJB and allows the development of components in different programming languages.

(Refereed-2001)

state.
- *Stateful Session Beans*: non persistent components that implement business processes with conversational state.
- *Entity Beans*: persistent components that implement abstractions of business data.
- *Message Driven Beans*: non persistent components activated asynchronously.

The definition of an EJB component includes:

- *Remote interface*: defines the business methods.
- *Home interface*: defines the creational and finder methods.
- *Bean class*: implements the business logic.

For Entity Beans, a primary key class must be defined in order to uniquely identify Entity Bean instances.

The AIE architecture includes the following elements (Figure 1):

- *Primary Data Sources (PDS)*: they are specialized existing or future systems like SCADA systems and corporate databases.
- *Real Time Data Sources (RTDS)*: provide a unified interface for accessing PDS without creating HLDS. These components are less expensive to create and manage than HLDS and are used in applications that require real time data updates (e.g. certain visualization applications).
- *High Level Data Sources (HLDS)*: provide a business oriented object view of information from PDS. The goal of these components is isolating the data logic from its source, so that applications do not have to search through several relatively low level sources.
- *Primary Event Generators (PEG)*: provide asynchronous notifications to other components under pre−established conditions in the PDS.
- *Server−side Applications*: implement business logic. Applications obtain data from HLDS, unless they require real time data updates.
- *Client Applications*: these applications reside outside of the application server and are generally used for data visualization and administration of applications that reside in the application server. Most of these applications should be based on Web technology.

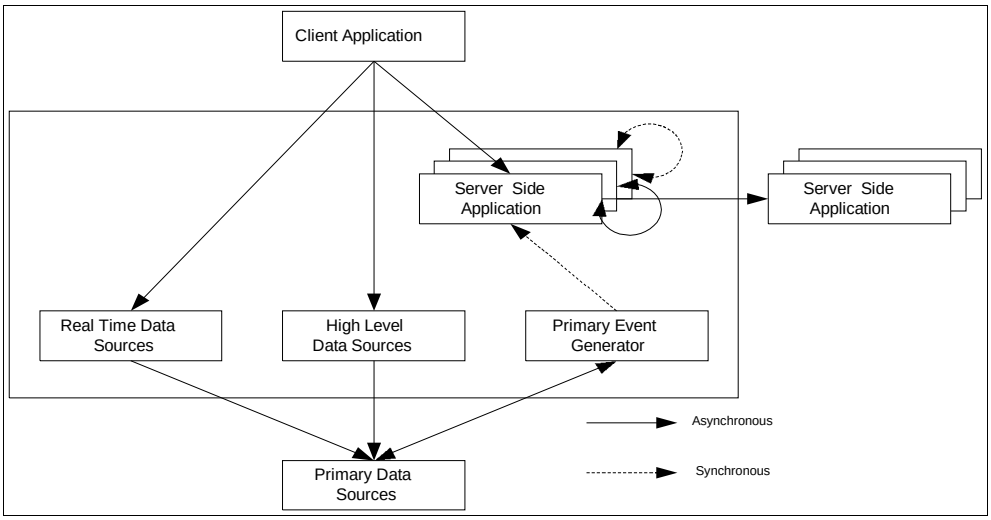

Figure 1. AIE architecture.

3. Architecture elements

With reference to Figure 1, the architecture elements are now described.

Primary Data Sources

Primary Data Sources (PDS) are specialized systems that provide data to be used by AIE applications. Examples of PDS are SCADA, relational database and tag database systems. PDS reside outside the CTM and usually have proprietary

programming interfaces.

Real Time Data Sources

Real Time Data Sources (RTDS) define a standard interface to PDS that provide access to constantly updated data identified with tags (e.g. SCADA systems) and a class to represent the values they provide.

The Signal class represents a value obtained from an RTDS and supports the following concepts:

- Time stamp: represents the instant when the datum was captured. It is an integer that holds number of milliseconds since January 1st, 1970 at 00:00:00 UTC.
- Value: represents the datum value. It is a IEEE 754 double precision floating point number. In case of a logical signal, the Signal.TRUE and Signal.FALSE constants are used to represent the logical values. In case of an integer signal, it is represented by the nearest floating point number.
- Quality: represents the datum quality (e.g. OK, questionable).

The RTDS are implemented as Stateless Session Beans which use a common interface as the remote interface.

The RTDS interface provides operations that can be divided in three groups:

- Current data access: provides operations to manipulate current state of variables.
- Historical data access: provides operations to manipulate previous states of variables.
- Tag management: provides operations to manage variables (e.g. add and remove).

High Level Data Sources

AIE applications should be designed in terms of business objects instead of programatic interfaces for a specific data source (e.g. the APIs of various SCADA systems, SQL).

Designing applications in terms of business objects provides the following benefits:

- Data logic is separated from the data source, which lets the developer change the data source without affecting the applications. Furthermore, it is possible to define several implementations (mappings) for the same business object. For example, it might be possible to make an implementation of a Pump business object that represents a fully automated pump and another implementation which represents a partially automated pump.
- There is no need to access data using product specific APIs.
- Code complexity and development time of applications are reduced considerably.

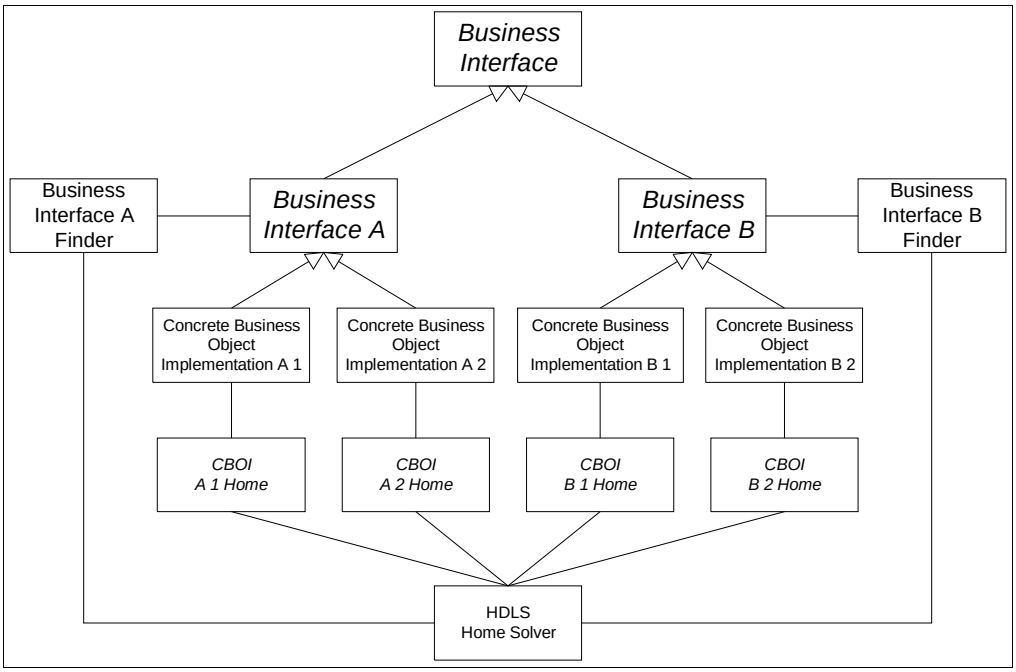

Figure 2. HDLS class diagram.

Figure 2 shows a class diagram for HLDS. Their structure is described as follows:

- There are interfaces that represent business objects (e.g. Pump and Flowstation) and provide the corresponding business methods (e.g. Pump.isOn(), Pump.isOff(), Flowstation.getPumps()). These interfaces are called Business Interfaces (BIs) and extend a more abstract interface (BusinessInterface) that provides basic methods common to all business objects.
- There are several concrete implementations of each interface (e.g. AutomatedPump, ManualPump, BasicSupervisionFlowstation). They are called Concrete Business Object Implementations (CBOIs), are implemented as Entity Beans and use a BI as the remote interface. Each instance of a CBOI has an ID, called Business Object ID (BOID) which allows locating and referencing the business object throughout the system.
  Each CBOI stores in a persistent repository the information needed to implement the business methods of the BI. This information can be tag IDs , BOIDs of other business objects or any other value needed to get data from a PDS.
- For each BI, there is a Stateless Session Bean that permits the location of any business object that implements that interface, given the BOID. This component is called Business Object Finder (BOF).
- There is an Entity Bean, called HDLS Home Solver (HHS), that keeps track of every business object in the system and permits the location of the home interface of any CBOI, given a BOID. The HHS also allows the location of any business object, but returns it as an abstract interface (BusinessInterface) that requires an explicit type conversion (cast) by the caller.

It is necessary to design a BI collection specific to the industry where the AIE is going to be implemented. For example, in a petroleum company the BIs would be well, manifold, flowstation, among others.

Primary Event Generators

Primary Event Generators (PEGs) are daemons responsible to detect certain conditions in PDS that result in asynchronous notifications to other components of the AIE. The detection mechanisms of those conditions are PDS dependant. The notification service used is JMS (Java Message Service), which is included in all standard J2EE implementations.

An important feature of PEGs is that they must be based on a framework designed to achieve high availability, since many mission critical application would depend on them.

Server−side Applications

Server−side applications are groups of components (e.g. Stateless or Stateful Session Beans) that implement business logic. For example, a report application can be implemented as a Stateless Session Bean, while an optimization application might require a Stateless and a Stateful Session Bean.

Applications should never access the PDS directly. Most applications will only have to access HLDS or other applications, although there could be applications that will require access to RTDS for performance reasons (e.g. production data visualization).

There may be cases where an existing application cannot be implemented in terms of Enterprise JavaBeans (e.g. legacy applications, ERP systems); in such cases it will be necessary to create a group of components that act as a wrapper to the original application, so that other elements in the AIE access it as if it were an AIE application.

Client Applications

Client applications consist fundamentally of interactive applications for data processing and visualization, for example, data visualization and parameter settings for server−side applications. It is recommended that client applications be based on web technology, so there is no need to administer configuration parameters in client computers.

Most client applications will not access data sources directly (HDLS or RTDS), but through server−side applications. However, there may be certain applications requiring real time data updates (e.g. process data visualization) that will access RTDS.

4. Conclusions and recommendations

This paper discussed the design of an automation architecture for continuous process industries based on the Enterprise JavaBeans server−side component model. This architecture has been designed considering criteria such as

interoperability, portability, scalability, performance, reliability, security, use of legacy systems and mantainability. The AIE architecture includes the following elements:

- *Primary Data Sources (PDS)*: they are specialized existing or future systems like SCADA systems and corporate databases.
- *High Level Data Sources (HLDS)*: provide a business oriented object view of information from PDS. The goal of these components is isolating the data logic from its source, so that applications do not have to search in several relatively low level sources.
- *Real Time Data Sources (RTDS)*: provide a unified interface for accessing PDS without creating HLDS. These components are less expensive to create and manage than HLDS and used in applications that require real time data updates (e.g. certain visualization applications).
- *Primary Event Generators (PEG)*: provide asynchronous notifications to other components under certain conditions in the PDS.
- *Server−side applications*: implement business logic. Applications obtain data from HLDS, unless they require real time data updates.
- *Client applications*: these applications live outside of the application server and are generally used for data visualization and administration of applications that reside in the application server. Most of these applications should be based on web technology.

The authors are evaluating the implementation of the AIE for the Oil Industry. Its implementation in other specific continuous process industries would be an exciting area of research.

The upcoming specification of the CORBA Component Model will allow the implementation of the proposed architecture in a variety of programming languages.

REFERENCES

[1] OPC Foundation. OLE for Process Control Overview. Accessed online at http://www.opcfoundation.org
[2] Monson−Haefel, Richard (2000). Enterprise JavaBeans. 2nd Edition. O' Reilly & Associates. 4−5.
[3] Sun Microsystems, Inc. (2001). Java 2 Platform, Enterprise Edition Specification. Accessed online at http://java.sun.com.
[4] Object Management Group (1999). CORBA Components. Accessed online at http://www.omg.org

  (Refereed-2001)