A reference software architecture for the development of industrial automation high-level applications in the petroleum industry

Guido Urdaneta *,1, Juan A. Colmenares, Néstor V. Queipo, Nelson Arapé, Carlos Arévalo, Mirché Ruz, Héctor Corzo, Andreína Romero

Instituto de Cálculo Aplicado, Facultad de Ingeniería, Universidad del Zulia, Maracaibo, Venezuela

Received 10 March 2005; accepted 10 April 2006

Available online 7 August 2006

Abstract

This paper presents a reference software architecture for the development of enterprise industrial automation applications for the oil industry. Its design accounts for criteria such as interoperability, portability, scalability, availability, security, use of legacy systems and maintainability. The architecture includes a technological platform that consists of a J2EE application server, a failover management system and, optionally, a server farm with an IP redirection-based load balancer. Also part of the architecture are infrastructure elements such as: (i) process data sources (PDSs) that offer an uniform interface for the synchronous and asynchronous access to SCADA or similar systems, (ii) field event generators (FEGs) that produce asynchronous notifications corresponding to the occurrence of pre-established conditions in the industrial processes, and (iii) business entities (BEs) that allow the handling of persistent information of real business entities, independently of the persistence mechanism used. Finally, its effectiveness is verified through the development of a prototype application for the optimization of the duration of well production tests.

Keywords: Industrial automation; Software architecture; Enterprise applications

1. Introduction

A significant part of the cost associated with the development of oil reserves is related to process and data management [1–2]. For example, the Petroleum Open Standards Consortium (POSC) has stated that the shortcomings of the traditional architecture that supports the execution of high-level applications in the oil industry may represent up to $3 dollars in the oil prices. Furthermore, Norsok, a Norway initiative has established that there may be room for up to a 50% reduction of the cost of developing oil reserves by optimizing key processes. In this context, automation systems play a vital role because they allow for the supervision, analysis and control of complex oil production processes in such a way that safe and nearly optimal operational values can be achieved. Of particular interest are the so called Industrial Automation High-Level Applications (IAHLAs), which are software applications employed by specialized users to manage production processes. These applications permit the evaluation of operational conditions, forecasting, scenario analysis and adjustment of operational parameters, among other tasks.

In the oil industry and other continuous processes industries (e.g., electric and water/wastewater companies), a standardized reference software architecture rarely rules the evolution and integration of automation systems, and IAHLAs. Therefore, specific solutions are commonly developed as needs arise. This situation usually leads to:

- interoperability and portability problems due to the use of proprietary products based on non-standard technologies,
- maintenance problems because of the needless complexity of the system,
- scalability problems caused by inefficient resource management (e.g., use of a two-tier client-server architecture when a multi-tier one is more appropriate), and

* Corresponding author. MCO-459 11010 NW 30TH ST STE 104 Miami, FL 33172-5032 USA.

E-mail addresses: guidox@ica.org.ve (G. Urdaneta), juancol@ica.org.ve (J.A. Colmenares), nqueipo@ica.org.ve (N.V. Queipo), narape@ica.org.ve (N. Arapé), carevalo@ica.org.ve (C. Arévalo), mruz@ica.org.ve (M. Ruz), hcorzo@ica.org.ve (H. Corzo), aromero@ica.org.ve (A. Romero).

1 Present address: Dept. of Computer Science. Vrije Universiteit. De Boelelaan 1081. 1081HV Amsterdam, Netherlands

0166-3615/$ – see front matter © 2006 Elsevier B.V. All rights reserved.
• incoherent security policies (e.g., each application has its own security scheme, which results in either the creation of multiple accounts for the same user in multiple systems or the lack of authentication).

Several techniques have been used to solve these problems, for example: concentration of data from multiple systems in common databases, use of repositories based on standard data models (e.g., PPDM [3] and Epicentre/POSC [2]), use of standardized protocols and interfaces for accessing systems that provide production data (e.g., OPC [4], CORBA DAIS [5], MODBUS [6]), use of message-oriented middleware [7] or transaction processing monitors [8] to standardize the access to certain resources, and use of distributed object collections based on standard repositories (e.g., OpenSpirit [9]).

These techniques usually offer good solutions for particular problems; however, they cannot be regarded as reference architectures for IAHLAs because they:

• Do not provide a practical abstraction mechanism of business elements (e.g., wells, flow stations) that takes into account real-time data coming from the field (e.g., signals of pressure and flow). This makes it hard to develop IAHLAs that require the integration of enterprise information (relatively static data), and real-time data (e.g., signal samples from a SCADA system).
• May not offer scalability or high availability features, compromising the reliability of the system.
• May not provide a unified model for security management.
• May present maintainability problems, since they require applications to be developed in terms of systems programming interfaces and not in terms of business functionality.
• May not offer support for multiple hardware and software platforms, which seriously affects portability and interoperability.

Table 1 presents a summary of recent work regarding software architectures for industrial automation and makes a comparison with the present work (Table 1). Dujmović [10] shows the benefits of object-oriented concepts in industrial automation applications by developing a framework based on JavaBeans components. The framework provides facilities for field bus communication, error detection, status monitoring, graphical user interface and Internet services. Abstraction of business elements is achieved using JavaBeans components. This framework is tested in an elevator control application. Chisholm and Smith [11] use ActiveX components to implement a distributed architecture for industrial environments. Their work focuses on the communication of personal computers on an Ethernet network and PLCs using a fieldbus. Chacón et al. [12] present a conceptual automation architecture based on a three-axe integration model which allows the interaction between support areas, production processes and decision systems. This work focuses on the modeling of continuous production complexes as a composition of components using an object-oriented approach based on the use of the Unified Modeling Language (UML). Furthermore, Lüders et al. [13] describe the componentization of an industrial control system. They use COM as the component model for their system. The system architecture includes components for I/O access, communication management and user interfaces. They analyze the following quality attributes: performance, reliability, maintainability and scalability. García et al. [14] propose an architecture that uses COM and ActiveX components for defining domain components, XML for defining structural dependencies for these components and OPC for communication with plant devices. The architecture is tested on a robot cells project. Finally, Gupta et al. [15] propose a software architecture for power distribution automation. Their architecture is based on the concept of a centralized database for the integration of software elements. This work emphasizes a number of both high-level and low-level power distribution applications.

The analysis of these studies shows the following limitations:

• the use of technologies specific to a hardware/operating system platform, thus limiting interoperability and portability [11,13,14],
• lack of support for distributed computing [10],
• high availability, scalability and security issues are not addressed [10–12,14],
• lack of an abstract model for business elements that takes into account process data and persistent information [10–15], and
• lack of a well defined framework for the development of new applications [15].

Table 1
Summary of recent work in the area of software architectures for industrial automation applications

<table>
<thead>
<tr>
<th>Authors</th>
<th>Underlying technology</th>
<th>Portability</th>
<th>Distributed computing</th>
<th>Facilities for high availability, security and scalability</th>
<th>Well defined prog. interfaces</th>
<th>Mechanism for abstraction of persistent/real-time data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dujmović [10]</td>
<td>JavaBeans</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Chisholm and Smith [11]</td>
<td>ActiveX</td>
<td>Windows only</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Not specified</td>
</tr>
<tr>
<td>Chacón et al. [12]</td>
<td>Not specified</td>
<td>Not specified</td>
<td>Not specified</td>
<td>No</td>
<td>Yes</td>
<td>Not specified</td>
</tr>
<tr>
<td>Lüders et al. [13]</td>
<td>COM</td>
<td>Windows only</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Not specified</td>
</tr>
<tr>
<td>García et al. [14]</td>
<td>ActiveX/XML</td>
<td>Windows only</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Not specified</td>
</tr>
<tr>
<td>Gupta and Varma [15]</td>
<td>Not specified</td>
<td>Central database</td>
<td>Yes</td>
<td>Not specified</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Urdaneta et al. (present work)</td>
<td>J2EE/FMS</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
This paper presents INAHLAR (INdustrial Automation High-Level Application aRchitecture), a reference software architecture for the development of IAHLAs that overcomes the aforementioned limitations. Its design considers criteria of interoperability, portability, scalability, availability, security, use of existing systems and maintainability. Additionally, its effectiveness is verified through the development of a prototype application for the optimization of the duration of well production tests. The rest of the paper is structured as follows: Section 2 describes IAHLAs, Section 3 establishes the design criteria for the proposed architecture, Section 4 presents the technological platform, Section 5 describes the proposed architecture, Section 6 presents the case study, and the conclusions are given in Section 7.

2. Industrial automation high-level applications

Continuous processes industries, and in particular the oil industry, typically have automation systems for controlling and supervising their production processes. These systems fundamentally include (Fig. 1):

- **Field instrumentation**, which consists of sensors and actuators that allow to obtain process variable values and manipulate these variables, respectively.
- **Field computers**, which read data from the sensors and control the actuators; PLCs (Programmable Logic Controllers) [16] and RTUs (Remote Terminal Units) [17] are currently the most widely used.
- **Supervisory control and data acquisition (SCADA) systems** [17], which make the data from the sensors and the control capacity of the actuators available to remote operators, applications and other systems. These systems usually store samples of process variables as quadruples \((t, v, ts, q)\), where \(t\) is the identification (tag) of the variable, \(v\) is the value of variable \(t\) at time \(ts\) (usually \(v\) a floating point, integer or boolean value), \(ts\) is a timestamp and \(q\) indicates the quality of the datum (e.g., correct, bad, uncertain). Some systems store process information corresponding to long periods, and they are called Time Series Archives (e.g., PI DataStorage [18]).

In the last years there has been a trend in the oil industry to incorporate software applications for the effective management of production processes (called in this paper *Industrial Automation High-Level Applications*—IAHLAs). These applications allow specialized users (e.g., process engineers and technical managers) to know the management indicators of production processes in order to: (i) evaluate their performance, (ii) predict future situations, (iii) identify abnormal behavior, and (iv) analyze possible scenarios. On the other hand, IAHLAs are usually part of heterogeneous environments, since they require production process data from several sources (e.g., SCADA systems, relational databases and enterprise information systems) and interact with other specific purpose applications. Examples of IAHLAs in the oil industry are software applications that allow for:

- the optimization of duration of well production tests,
- the online estimation of well productivity index and detection of substantial variations of it,
- the estimation of permeability in producing wells using historical records of bottom hole flowing pressure and oil production rate, and well test interpretation models, and
- the determination of operational parameters for production optimization in contingencies (e.g., restrictions in gas-lift supply)

According to their operation mode, the IAHLAs most frequently found in the oil industry (and other continuous process industries) can be classified as:

1. **Non-continuous execution server applications**, which execute actions of finite (typically short) duration in response to a client request; in many cases the action defines or is part of a transaction. Among numerous examples are applications for: oil field production forecasting, tank volume computation, and calculation of optimal set-points for artificial gas-lift.

2. **Non-continuous execution client applications**, which usually run on personal computers or workstations activated by end users and typically interact with server applications for a finite time. By their nature, these applications do not have special reliability, scalability and security requirements. Production data visualization applications fall under this category (e.g., production reports and process displays).

3. **Unique instance continuous execution applications**, which execute an operation for an indefinite amount of time and
require that only one instance of the application is executing at any time. This type of applications is very common in industrial automation environments (e.g., control loops and continuous supervision of process variables). These applications usually are not servers; that is, they do not process requests issued by client applications.

4. Multiple instance continuous execution applications, which are similar to the previous ones since they execute an operation for an indefinite amount of time, but in this case there is no requirement of only one instance running at any time. A typical example of these applications is a highly available programmed tasks scheduler for server applications (e.g., storage of daily production reports in a database) that uses a database synchronization mechanism to avoid activating duplicate tasks.

3. Software architecture design criteria

An architecture for the development of IAHLAs, especially in continuous processes industries, should fulfill the following extra-functional design criteria:

- **Interoperability.** The architecture must be based on standards supported by multiple vendors, so that an organization can choose between several products (e.g., middleware, databases, operating systems) without affecting their applications.

- **Portability.** It must be possible to implement the architecture in different hardware and operating system platforms, so that an organization can choose the combination that best suits its needs.

- **Scalability.** The architecture must allow server applications to increase their capacity by incorporating additional hardware, without any software modifications beyond configuration settings.

- **Availability.** The architecture must have mechanisms that guarantee the high availability of the applications (e.g., hardware redundancy [19], diverse design software [20]).

- **Security.** The architecture must provide authentication, authorization and encrypted communications services.

- **Use of existing systems.** The architecture must provide mechanisms that allow the applications to access existing enterprise information systems in the organization (e.g., message-oriented middleware, transaction processing monitors, ERP systems).

- **Maintainability.** Applications should be developed in terms of business functionality and not in terms of systems programming interfaces specific to underlying technologies. This eases maintenance labors such as bug fixing, application improvements and technological platform changes.

4. Technological platform

A technological platform has been chosen in order to support the types of applications described in Section 2 and satisfy the criteria established in Section 3. The platform includes the following elements:

1. **J2EE (Java 2 Enterprise Edition) Application Server** [21]. It defines a set of programming interfaces for enterprise-class distributed applications. Typical J2EE implementations provide implicit access to middleware services such as security, load balancing and high availability. For our purpose, the most relevant J2EE interfaces are:

   - **Enterprise Java Beans (EJB)** [22], which define serverside components with declarative middleware services (e.g., remote invocation, persistence, resource management, security, and transactions). There are three types of components: (i) session beans, which allow the modeling of business processes, (ii) message-driven beans, which allow the implementation of asynchronously activated business processes, and (iii) entity beans, which allow the modeling of business objects with persistent information.

   - **Java Message Service (JMS)** [23], which provides distributed and transactional asynchronous messaging services.

   - **Servlets and JavaServer Pages** [24,25] are components that allow the development of web applications.

   - **J2EE Connector Architecture**[26], which provides standard interfaces for accessing existing enterprise information systems.

   J2EE Application Servers provide scalability and high availability capabilities for non-continuous execution server applications, the ability to run on multiple operating systems and hardware architectures, and independence from the middleware vendor. The benefits of using this technology are well documented. See, for example [27–33].

2. **Failover Management Software.** The programming models offered by J2EE support only non-continuous execution server applications. In order to achieve high availability and scalability in continuous execution applications, it might be possible to run several copies of the applications, but many times this is not possible. For example, if an application that continuously monitors field conditions and provides asynchronous notifications based on them is run at the same time on multiple computers, unwanted duplicate notifications will be produced. In this case it is necessary to guarantee that only one instance of the application is running at a given time and that it is able to survive failures. Failover management software provides the capability to automatically migrate a continuously running application from one computer to another one in case of failures, thus providing the necessary high availability for this type of application. Advantages of this technology are extensively discussed in [34].

3. **Server Farm with IP Redirection.** Server applications can improve their scalability using a farm with IP redirection. The IP redirector allows client applications to access multiple computers using the same IP address. Note that this infrastructure element is only necessary when it is required to improve the scalability of server applications that can be replicated in multiple computers. The benefits of IP redirection are also presented in [34].
5. INAHLAR—software architecture

This section describes the elements of INAHLAR and their implementation details. Fig. 2 shows a UML diagram that includes the constituents of INAHLAR.

5.1. Process data sources (PDSs)

Process data sources consist of a set of classes and interfaces that provide uniform access to process data from SCADA or similar systems. Their main objective is to decouple IAHLAs from the proprietary application programming interfaces (APIs) of those systems. Their importance lies in the fact that they are the link between low level industrial automation systems and IAHLAs.

The `Sample` class is their central element. It represents a sample of a process signal and has the following features:

- It provides methods that allow getting the value, quality and timestamp of the sample.
- It supports serializable Java type values, although it is optimized for numeric and boolean values, which are more frequent in industrial automation applications.
- It supports multiple quality values such as: invalid, uncertain, bad, stale, OK, computed, in alarm and manual.
- The timestamp is a 64-bit integer value which represents the number of milliseconds since 1 January 1970 at midnight (UTC).
- It is immutable [35], thus it is safe to be used in multithreaded environments without using special synchronization primitives.

On the other hand, PDSs define interfaces with operations commonly supported by SCADA systems to access: (i) real-time data (e.g., getting the most recent sample of a signal), (ii) historical data (e.g., getting all samples from last month) and (iii) configuration data (e.g., signal tags). They must be implemented as specific stateless session EJB components for each SCADA (or similar) system. The interfaces are:

- `ProcessDataSource`. It is the entry point for the other interfaces.
- `RealTimePds`. It offers operations for synchronous access to real-time data.
- `HistoricalPds`. It allows synchronous access to historical data.
- `PdsConfiguration`. It allows synchronous access to configuration data.
- `AsyncRealTimePds`. It allows asynchronous access to real-time data.
- `AsyncHistoricalPds`. It allows asynchronous access to historical data.
- `AsyncPdsConfiguration`. It allows asynchronous access to configuration data.

Fig. 3 shows a UML Diagram of PDS interfaces.
Additionally, as part of the architecture, a generic PDS was developed as starting point for other implementations. The generic PDS provides:

- A class implementing the `ProcessDataSource` interface, which can be used without modification by any other specific implementation.
- Classes implementing the asynchronous interfaces that delegate the work on the corresponding synchronous interfaces.
- Skeletal abstract classes for synchronous interfaces implementing operations with multiple signals in terms of single signal operations. The synchronous single signal operations have to be implemented for each SCADA system using their respective proprietary API.

Furthermore, a Universal PDS was developed, which can be used as a unique access point for all the other PDSs. The Universal PDS does not access any SCADA system directly, but delegates on other PDS implementations. Fig. 4 shows the relationship between the Universal PDS and other PDS implementations.

Finally, to verify the effectiveness of the PDS design, a specific PDS for the PI DataStorage system [18] was developed and it was called PDS-PI.

5.2. Field event generators (FEGs)

FEGs are applications that produce asynchronous notifications related to pre-established conditions in field variables under a Publish-Subscribe scheme [36]. This is a common requirement for many IAHLAs in the oil industry. JMS was selected to implement this scheme, which frees FEGs from the responsibility of providing logic for subscription management, high availability, load balancing and security. This implies that FEGs and subscriber applications are loosely coupled from a programming interface perspective and therefore, the contract between them must be established using the message format.

The FEG Message Format (FEG-MF) is based on the sample class. A large part of the message information is set by JMS properties (Table 2), which allow subscribers to filter incoming messages. This is very important since most subscriber applications are designed to process only a subset of the messages (e.g., messages from a specific signal). The body of the message has a list where the \( n \)th element contains information related to the signal whose tag is specified by the property \( \text{Tag}_N \).

INAHLAR includes the implementation of a FEG application that can generate messages from conditions detected through PDS implementations. This FEG implementation is called FEG-PDS and its modular design facilitates possible modifications intended to support other data sources. FEG-PDS is an example of a unique instance, continuous execution application whose high availability requirements are provided by a Failover Management System.

FEG-PDS comprises the following modules (their interaction is illustrated in Fig. 5):

1. **Condition detection.** It consists of a set of classes and interfaces that contain the logic for the detection of conditions, and the classes to be used in the body of the message. Some conditions that can be detected are: changes in the value and/or quality of the signal, the signal value goes out or comes into a given interval, and the signal value surpasses a defined threshold.
2. **Configuration data.** It establishes the way in which the conditions to be detected are configured.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>String</td>
<td>Name of the source that produced the message. The rule for determining this</td>
</tr>
<tr>
<td>Tag(N)</td>
<td>String</td>
<td>Tag of the (N)th signal referenced by the message ((N = 0, 1, \ldots, n))</td>
</tr>
<tr>
<td>Value(N)</td>
<td>Primitive</td>
<td>Value of the (N)th signal at the time of the message generation</td>
</tr>
<tr>
<td>Quality(N)</td>
<td>Int</td>
<td>Quality of the (N)th signal at the time of the message generation</td>
</tr>
<tr>
<td>Timestamp(N)</td>
<td>Long</td>
<td>Time at which the (N)th signal was analyzed</td>
</tr>
<tr>
<td>Condition(N)</td>
<td>String</td>
<td>Condition of the (N)th signal that triggered the message generation</td>
</tr>
</tbody>
</table>
3. **Process data acquisition.** It offers a set of classes and interfaces that allow the acquisition of process data from their source in a decoupled way. Its implementation uses the data access interfaces defined by the PDS.

4. **Processing.** It contains the logic of the loop for data reading, condition detection and message generation.

5. **Hot reconfiguration.** It contains logic that supports the reconfiguration of FEG-PDS while it is running.

6. **Connection, session and destination management.** It includes utility classes for efficient management of JMS session and destination references, and automatic reconnection to JMS servers.

7. **Main program.** It initializes the modules just described.

### 5.3. Business entities (BEs)

BEs consist of a set of components that abstract real business objects with persistent information, such as reservoirs, wells and flow stations, among others. BEs can potentially have multiple implementations; for example, different divisions of the same company can implement the well concept using different data repositories (e.g., PPDM [3] and Epicentre/POSC [2]).

BEs define the following concepts:

- **Business entity data (BED):** is a class that contains data. These objects do not contain any data access logic. BED objects are implemented as simple classes with default constructors and getter/setter methods for accessing the data.

- **Business entity factory (BEF):** is a component that provides operations related to BED objects such as creation, searching, updating and deletion. A BEF component contains all the logic required to interact with the systems that provide the persistence for BED objects (e.g., relational databases). The BEFs are implemented as stateless session EJB components.

The main advantage of BEs is that they simplify access to data that is common to many IAHLAs. Fig. 6 shows a UML diagram with the general structure of a BE.

### 5.4. Implementation details

INAHLAR was implemented using the J2EE application server BEA Weblogic 8.1 [37] and the failover management system Linux Failsafe Version 1.0.4 [38]. PostgreSQL 7.4.2 [39] was the relational database management system used for managing the persistence of business entities. On the other hand, a PDS implementation, called PDS-PI, was developed for accessing PI DataStorage Version 3 [18]. PDS-PI was written in Java and C++, and employs the PI-API Version 1.3.4. Additionally, a FEG was implemented as a unique instance continuous execution application with high availability.

### 6. Case study: well production test optimization

In order to evaluate the effectiveness of the proposed software architecture, a prototype application for minimizing the duration of well production tests in separators [40] was developed.

Well production tests are very important to evaluate the performance of individual wells, diagnose well problems, and manage reserves properly. In a typical scenario (e.g., consolidated oil production fields) several tens of wells are connected to a flow station with up to three test separators (handling different wells according to their output pressure). For testing a well, its flow is sent to a test separator, which is filled and emptied several times. The well oil production is calculated by multiplying the number of times the separator is emptied by its known volume and dividing this product by the time. A test may last from 8 to 16 h with wells typically tested twice a month. Note that 2 weeks (on average) is a long period considering that the engineer responsible for a well is committed to deliver a monthly production quota. Consequently, reducing the production test duration helps to timely detect anomalies in production wells to minimize deferred production.

The application associated with the case study is called well production test optimization (WPTO) and includes the following modules: (i) business entities, (ii) test scheduler, (iii) test termination evaluator and (iv) test result saver. These modules were developed employing the technological platform and architecture elements described in Sections 4 and 5, respectively.

### 6.1. Business entities

WPTO uses the following BEs:

- **BusinessUnit.** It represents a business unit of the organization. Each business unit has associated a set of facilities (Facility).
• **Facility.** It represents a physical facility with a geographical location (e.g., flow station, well). A facility may contain Equipments.

• **FlowStation.** It represents a flow station and extends Facility. In the context of WPTO, a flow station has ProductionSeparator and TestSeparators.

• **Well.** It represents a well that produces crude oil. This BE extends Facility. A well has one or more completions (WellCompletion).

• **WellCompletion.** It represents a well completion.

• **Equipment.** It represents an equipment in a Facility (e.g., separator, pump, tank).

• **ProductionSeparator.** It represents a production separator in a flow station. This BE extends Equipment.

• **TestSeparator.** It represents a test separator in a flow station. This BE extends Equipment.

• **WellTestResult.** It represents the result of a production test for a specified well.

• **WellTestSpecification.** It includes the specification parameters for the execution of a production test for a given well (e.g., number of initial separator discharges to ignore, tolerance, maximum duration).

• **WellTestSchedule.** It represents the schedule of tests of a specified test separator. It contains a list of WellTestSpecifications related to a test separator.

Note that several of these BE are useful in the implementation of other IAHLAs in the oil industry, so they could be reused. Also note that many of the attributes of these components are actually tags whose real-time and historical values are accessed through PDS interfaces.

Fig. 7 shows a UML diagram with the data model implemented by the WPTO business entities.

6.2. Test scheduler

It includes a client application that invokes a stateless session bean, called WellTestScheduler, that is responsible for generating the schedule of well tests of a test separator. The schedule is generated by sorting the well tests in ascending order according to their maximum duration. A human operator may modify the schedule before its definitive approval.

6.3. Test termination evaluator

Every time a test separator is emptied, the process variables of the current well test must be evaluated to determine whether the termination conditions are met. For that, the FEG is configured so that it generates a message every time a test separator is emptied. The message is processed by an EJB component (WellTestTerminator) that contains the logic for terminating the test. To indicate the end of a test, the value of a process variable is changed through the PDS.

6.4. Result saver

When a well test ends, the results must be stored in a database. For that, the FEG is configured to generate a message
at the end of a test. Such message is processed by an EJB component (WellTestResultSaver) that reads the results through the PDS and stores them in the database.

6.5. Operation mode

An operator accesses the client application that invokes the WellTestScheduler component in order to generate a well test schedule corresponding to a business unit. This schedule may be modified by the operator before its final approval. The schedule (WellTestSchedule) and the test specification (WellTestSpecification) are sent to field computers (e.g., PLCs), through the PDS, for their execution.

Every time a test separator is emptied, WellTestTerminator receives a message from the FEG and determines whether the test must end. After the test is finished (by an order from WellTestTerminator, by exceeding a time limit or by an order from an operator), WellTestResultSaver receives a message from the FEG and stores the result in a database.

Fig. 8 shows a message sequence chart that illustrates the interaction between the components of WPTO and INAHAR.

WPTO was executed using simulated data and proved INAHAR to be effective to develop IAHLAs in the oil industry while satisfying extra-functional criteria.

7. Conclusions

This paper has described INAHAR, a reference software architecture for the development of industrial automation high-level applications (IAHLAs) in the oil industry. IAHLAs are characterized by being part of heterogeneous environments and executing operations for the effective management of oil production processes.

INAHLAR overcomes the following limitations found in other efforts:

- the use of technologies specific to a hardware/operating system platform, thus limiting interoperability and portability,
- lack of support for distributed computing,
- high availability, scalability and security issues are not addressed,
- lack of a well defined framework for the development of new applications, and
- lack of an abstract model for business elements that takes into account process data and persistent information.

The first three limitations are overcome by means of a technological platform that includes a J2EE Application Server, a Failover Management Software and an IP Redirection system. The last two limitations are addressed with the creation of the following infrastructure elements: (i) process data sources (PDS), which offer a uniform interface for accessing SCADA or similar systems, (ii) field event generators (FEG), which provide asynchronous notifications related to the occurrence of pre-established conditions in field processes, and (iii) business entities (BE), which define a high-level mechanism for managing persistent information related to real business entities. These infrastructure elements allow IAHLAs to be independent of data sources (e.g., SCADA systems, corporate databases), so that they can be easily reused and ported.

By means of a prototype application for optimizing the duration of well production tests, INAHAR proved to be effective to develop IAHLAs for the oil industry. And considering its design features, it also promises to be equally useful in other continuous production industries (e.g., electrical power, gas and water supply companies) and become a reference software architecture for IAHLAs in those industries.

Future work should include extensions to ease the use of programming languages different than Java, and the creation of models for assessing the cost, time, and effort to implement INHALAR under field conditions.

Acknowledgement

The authors wish to thank the economic support offered by FONACIT (Venezuela) through projects G-97003899 and G-97000824. Also, we thank Proytek, S.A. and OSI Software Inc., for providing a license for the PI DataStorage system.

References


Guido Urdaneta received the Computer Engineering degree from the Rafael Belloso Chacin University (Venezuela) and the M.Sc. degree in Applied Computing from the University of Zulia (Venezuela). He is currently a Ph.D. student in the Department of Computer Science of the Vrije Universiteit Amsterdam (Netherlands). His research interests include distributed systems and their application to industrial automation. He is also faculty member of the Applied Computing Institute at the University of Zulia.

Juan A. Colmenares received the B.Sc. degree in electrical engineering and the M.Sc. degree in computer science from the University of Zulia (Venezuela) in 1997 and 2001, respectively. He is currently a Ph.D. Candidate in the Department of Electrical Engineering and Computer Science at the University of California, Irvine (USA). His research interests include real-time distributed systems, control and automation systems, and modeling and optimization. He is also faculty member of the Applied Computing Institute at the University of Zulia, and he serves as reviewer for the IEEE Latin America Transactions.

Nestor V. Queipo is a Professor and Director of the Applied Computing Institute, at the Faculty of Engineering of the University of Zulia, and has been a visiting professor at the Aerospace and Mechanical Engineering Departments of the University of Arizona, Tucson, and University of Florida, Gainesville, U.S.A. Sponsored by the Venezuelan government and oil industry, he leads national research efforts in the area of information systems for the oil and gas industry, reservoir characterization and simulation, surrogate-based analysis and optimization of oil recovery processes, and production forecasting with uncertainty quantification. He received his Ph.D. in Mechanical Engineering from the University of California at Berkeley, U.S.A.
Nelson Arape is Assistant Professor of the Applied Computing Institute, at the Faculty of Engineering of the University of Zulia, Venezuela. His research interests include distributed computing, load balancing, and automation and control. He received his M.Sc. in Applied Computing from the same research institute where he is now a professor.

Carlos Arevalo is Associate Professor of the Applied Computing Institute, at the School of Engineering of the University of Zulia, Venezuela. His research interests include distributed computing and its application in continuous production environments. He received his M.Sc. degree in Applied Computing from the same research institute where he is now a professor.

Mirche Ruz is a Sales Engineering Consultant at the Sybven Corporation in Maracaibo, Venezuela. Her research interests include distributed systems. She is a graduate student at the Applied Computing Institute, Faculty of Engineering, University of Zulia, Maracaibo, Venezuela. She received his B.Sc. in Computer Science from the Faculty of Sciences, University of Zulia.

Hector Corzo works as a Consultant in the area of Web applications at the Sybven Corporation in Caracas, Venezuela. His research interests include distributed systems, control and automation systems. He is a graduate student at the Applied Computing Institute, Faculty of Engineering, University of Zulia, Maracaibo, Venezuela. He received his B.Sc. in Electrical Engineering from the Faculty of Engineering, University of Zulia.

Andreina Romero is a System Analyst at the Sybven Corporation in Caracas, Venezuela. Her research interests include distributed systems. She is a graduate student at the Applied Computing Institute, Faculty of Engineering, University of Zulia, Maracaibo, Venezuela. She received her B.Sc. in Computer Science from the Faculty of Sciences, University of Zulia.