AspectTAZ : a new approach based on Aspect Oriented Programming for Object Oriented Industrial Messaging Services design

L. Teboul[*+], R. Pawlak[*], L. Seinturier[+], E. Gressier-Soudan[*], E. Becquet[*]
[*]CEDRIC-CNAM,
292, rue St Martin,
75 141 Paris Cedex 03 France,
{teboul, pawlak, gressier, becquet}@cnam.fr

[+]LIP6, Université Pierre et Marie Curie
4, place Jussieu
75 252 Paris Cedex 05, France
Lionel.Seinturier@lip6.fr

Abstract

Since 1994, we have designed and prototyped several industrial messaging services over different platforms. Our experiments have been based on ISO-MMS standard and TASE.2. This paper describes the lessons learned from these past projects, and some of our activities in designing a next generation of industrial messaging services. Faced with a fast changing world of communication protocols, a stringent requirement for these services is to be easily adaptable to new protocols (yesterday ONC-RPC and CORBA, today SOAP/XML). Recent software engineering researches introduced the notion of aspect-oriented programming (AOP) to cope with such a need in adaptability. This paper demonstrates the viability of this approach in addressing our requirement.

1. Introduction

In the past decade, we designed and prototyped different generations of industrial messaging services [9]. We used different distributed system platforms:

- ASN.1/BER together with Sun's ONC-RPC (implemented in C) [7],
- CORBA object request brokers (ORB) :
  - COOL, over the real-time micro-kernel ChorusOS, (implemented in C++) [10],
  - Jonathan, ORBacus over Linux, and Windows, (implemented in Java) [8] [23] [4],
  - MICO, over Linux (implemented in C++) [3].

The level of reuse between two implementations was low. Lessons learned from one prototype enhanced the design of the next one, but the old implementation was discarded. Other technologies (e.g. Java-RMI, OPC/DCOM) could have been also investigated. .NET or SOAP/XML are also promising for factory communication systems, and we'd like to evaluate them. Yet we want to avoid the "yet another version of an object oriented industrial messaging service over ORB Z" syndrome.

New challenges on factory communication system imply to be able to build distributed system services independently of the platforms that can support them. To face this software engineering problem, we decided to stress the Aspect Oriented Programming (AOP) [14] approach using JAC a framework (Java Aspect Components) [20] that some of us are developing. Earlier studies on AOP underlined its advantages for distributed system services engineering.

This paper describes AspectTAZ, our AOP based industrial messaging service. Its aim is to build an object oriented industrial messaging service easily adaptable to middleware technologies such as RMI, CORBA, SOAP/XML, or any other to come. This service is based on TASE.2 as specified in [3], uses the JAC framework, and is developed in Java. Only one core code of the industrial messaging services is specified and will be ready to run over the three platforms. One can notice that our project shares many goals with approaches such as the OMG Model Driven Architecture [18].

As far as we know, there is no equivalent project in the field of factory communication systems. This is probably due to the youth of the AOP approach. More generally, existing applications that are built with AOP are demonstrators or toy systems.

Furthermore, our work is of interest for power utilities. First, TASE.2, our application service model is used to support production data exchanges, and results from the standardization process driven by experts of the utility domain. Also, the European open Power market is under construction and could rely on TASE.2 services. Indeed, a close analysis of TASE.2 functions shows that it can be the basis of a real-time peer-to-peer data exchange service for the trading of energy. As a consequence, providing TASE.2 functionalities independently of any middleware solution is a key goal in order to reduce development costs.

This paper is organized as follows. Section 2 presents the principles of AOP and JAC. Section 3 expresses the needs for a new software engineering approach in factory communication systems. Section 4 recalls the key features of our object oriented TASE.2 based industrial messaging service and presents the design of

AspectTAZ. It gives the current performances. Section 4 also presents new trends for AspectTAZ. Section 5 concludes.

2. Oriented Programming and the JAC framework

2.1. Key principles of AOP

Separation of concerns in software engineering has always been a very natural means to handle complexity of software developments [19]. However, the design of a modular code can be a very tricky task for the programmer. It raises some issues such as performance, crosscutting, or redesigning when the software is used in a context that is quite different from the overseen one. By handling crosscutting within the language or system, the recent approach of Aspect-Oriented Programming (AOP) [15] seems to be a very promising way for helping developers to handle separation of concerns and to overcome the drawbacks of traditional design approaches.

Aspect orientation and aspect oriented programming [16] is an approach studied in the software engineering research field that was raised for the first time in 1997 by Gregor Kiczales from XEROX PARC. The field inherits results from both the separation of concerns ideas of design and analysis methods, and the reflection ideas of programming languages. An aspect-oriented application is a collection of aspects and of a program called a base program. An aspect is a piece of software that implements a requested feature. But, unlike a class, an aspect also defines the way it will modify the base program. This last point is important as it allows to leave the base program clear of any intrusion from the aspects that will be added later on. Kiczales' team designed an extension of Java called AspectJ [16], with dedicated keywords to write aspects, and implemented a compiler to generate executable code. Many other AOP languages or frameworks exist (e.g. AspectC [21], Apostole [22], AspectS [2], JAC [20]). However, if AOP introduces a new programming paradigm that complements existing ones, it is clear that it brings a new bunch of difficult problems such as aspects composition.

The base program and the aspects are glued together during an integration process called aspect weaving. Aspect developers specify with a syntactic contract called join points definition, the elements of the base program that will be modified by the introduction of the aspect. Depending on the AOP platform the granularity of these elements may vary (classes, methods, variables, method calls, exceptions). If the elements specified are missing in the base program, the contract is not fulfilled and the aspect will not be woven. In the AOP approach, the base program implements the business logic of the application, and the aspects are dedicated to non-functional properties. Nevertheless, as the frontier between functional and non-functional properties may be moving depending on the application field, aspects (e.g. time constraints) may be part of the functional requirements in some domains (e.g. real-time control), and of non-functional ones in other domains (e.g. word processing).

Another interesting point about AOP is that it leaves behind the traditional use relationship between software entities. In the procedural or object-oriented approach, a "client code" uses the functionalities provided by a "server code" (a method or a procedure). The "client code" developer must thus master the syntax and the semantics of the invoked "server code". In the AOP approach, the "server code" developer specifies the way his service, implemented as an aspect, modifies the "client code". As "server codes" are written less often and by more skilled developers, AOP is an interesting way of reducing development costs.

2.2. The JAC framework

JAC [20] is an ongoing research project whose goal is to build an environment for aspect oriented application development in Java. Fully working releases of the product exist and can be downloaded from our Web site [1]. JAC defines an AO programming model and implements an application framework that supports it. It is a general purpose application framework but our target domain and main interest lies in middleware systems and applications. Thus we put the emphasis on dynamicity that is a major requirement in this domain. An aspect with JAC is a set of aspect objects that are to be deployed on top of application objects. An innovative feature of JAC is that the link between aspects and application objects can be dynamically set and removed at run time.

JAC provides a standard library of aspects for remote communication (currently JavaRMI and CORBA, SOAP is under development), distributed deployment of applications, distributed naming and binding with name repositories, data caching, memory consistency (currently sequential consistency with strong replication and update of data, entry consistency is under development), persistency (with Java JDBC), GUI (with Java/Swing), logging. This set of aspects can of course be enriched by developer defined aspects.

One of the point worth mentioning about JAC is that, when several flavors of the same aspect exist (e.g. JavaRMI and CORBA for remote communication), the framework allows the undifferentiated use of one of them: the developer selects either RMI or CORBA and the same (unchanged) application takes advantage of the specified aspect (e.g. the application objects remotely communicate through JavaRMI or CORBA). Furthermore, and except for the remote communication aspect where the choice is to be made at configuration time, an aspect can be dynamically deployed,

undeployed and replaced by another one on top of a running application.

2.3. Programming with JAC

Programming an application with JAC is a three step process:

- base application development: this is the business process development (this task should stay clear of any intrusion from non functional properties),
- aspect development: each non functional property is to be implemented in a dedicated aspect,
- integration: the software architect defines which non functional (i.e. aspect) properties should be added (and where) on top of the application.

For the first task, JAC provides an application framework. For the second one, aspects must respect a programming model. Finally, once specified, the integration is automatically handled by JAC.

3. AOP for factory communications

The underlying idea of this paper is that one of the requirements for next generation open factory control systems is to be able to achieve an obvious separation between functional and non-functional properties. By functional properties, we refer to the business logic that can be caught in a process control application such as supervision or remote control. By non-functional properties we refer to the underlying platform that provides the core services (system, network or middleware services) to run and manage a process.

This requirement becomes unambiguous when solutions are analyzed. The issue here is to be able to precisely modularize what is relevant to the application and what is relevant to the process itself. The preferred property is of course, the ability to reuse all or most of the business process whenever the architecture changes, but also to leave these two pieces of code (business and architecture) as much independent as possible. Object-oriented approaches (based on ORB middleware) addressed the first property but in our opinion, failed to reach the second one. Our goal is thus to make a proof of concepts and to show how this independence exists with AOP.

The need of AOP for factory communication systems appears in two different ways, in fact at two different levels of factory architectures. Initially, our team were looking for a new approach to spend less efforts on developing new versions of functionally identical industrial messaging services. Related to this, we can assert that AOP is a very good answer to our requirement ant it suits to factory communication system design and implementation. Also, as a side effect of what AOP is provided for, we believe that it is an appropriate approach to design and implement the overall factory system. One can consider that protocol designers are aware of the problem that a protocol should circumvent. When a protocol is specified, it answers some true users' requirements, it is domain centric: semantic is relevant, not syntax. From this viewpoint, ISO-MMS history is significant. Experts provided a very interesting application protocol for manufacturing systems but it did not become as successful it should have been. Its most important drawback is that it was completely dependant from the ISO stack. We could think about what could have happened if the same ISO-MMS software would have been able to run in a flexible way over PROFIBUS, FIP, MODBUS, Internet at the same time. This sort of challenge is still existing in the automation community where fieldbus providers are concerned by migrating their proprietary high level communication services towards switched Ethernet and TCP/IP based solutions [11]. AOP clearly allows these services to be implemented once and to run on different communication systems if the deployment aspect includes the targeted platform sub-systems.

This is an open approach because emerging technologies can be added smoothly to the AOP environment. Also, it allows co-existing versions of a service over different platforms if gateways are defined. Gateways are easier to implement because the protocol is identical: exactly the same semantic everywhere, only communications in a broad sense change.

4. Design and Implementation of AspectTAZ

Our AOP based industrial messaging prototype is implemented in Java using the JAC framework [20].

4.1. An object oriented TASE.2 based industrial messaging service

The TASE.2 protocol is a companion standard of the popular MMS [24] designed to support the exchange of data between utility control centers and production units [12] [13] [14] [6].

TASE.2 is explicitly described as Client/Server based. Two types of interactions are defined: "operations" are initiated by clients and correspond to a classical reliable method invocation (they usually return a result), and, "actions" are initiated by servers and correspond to a notification with data. Four data transfer semantics are provided: "once" (classical client/server request), "periodic" (periodic transfer), "exception" (state change based transfer), "event" (event condition based transfer).

TASE.2 functions are separated in nine conformance blocks. As far as we are concerned, our work deals with blocks 1, 2. Block 1 defines a minimal set of services related to data management and periodic data exchanges. Block 2 extends block 1, it provides exception semantics often referred as Report By Exception semantics.

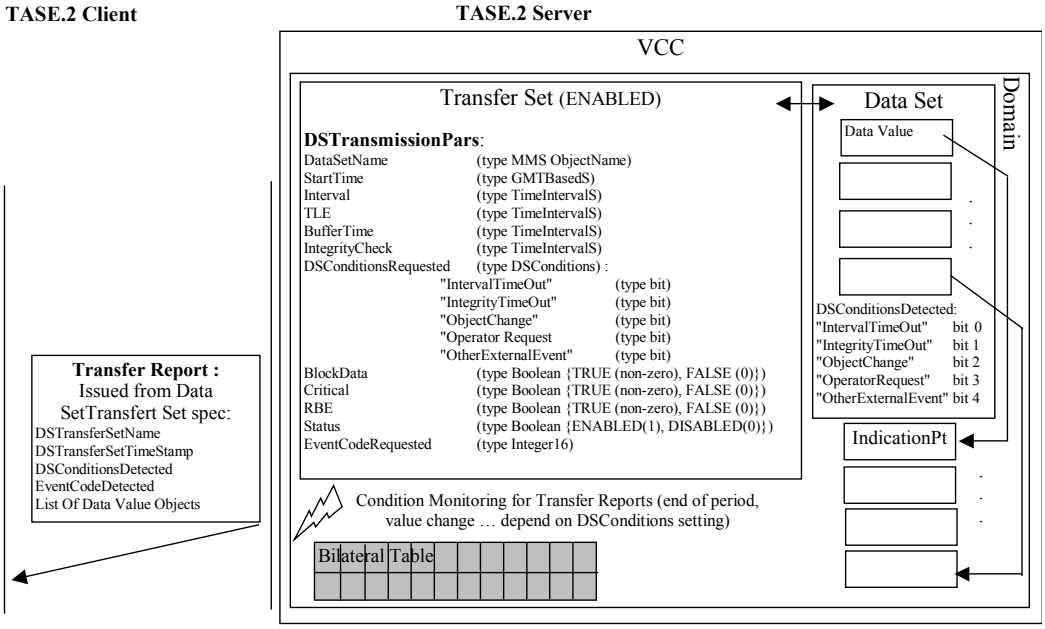

Figure 1. Overview of TASE.2 Variable Management.

TASE.2 defines "Data Value" objects and "Data Set" objects (supported by MMS named variables or list of named variables) managed by the server. Data Value management and Data Set management functions [14] deal with the look up of existing Data Values and Data Sets, their creation, their destruction, etc. Data values reference in practice specific information such as "Indication Points" (status information, analog values, attributes, etc.). Indication points can have attributes like timestamp, quality class (VALID, HELD, SUSPECT, NOTVALID), change of value counter… Data Set Transfer Sets describe the way Transfer Reports must be pushed toward the client and contain parameters defining under which conditions data values related to data sets are transmitted. Figure 1 gives an overview of variable management in TASE.2.

AspectTAZ offers TASE.2 functions on an object oriented basis over different platforms: RMI and CORBA currently, SOAP is planned for soon. Its design is based on ObjectTAZ [3], that objectified TASE.2 features on top of CORBA. Opposed to AOP benefits, ObjectTAZ is monolithic: access to the CORBA ORB is plugged into TASE.2 functions. Its interface has been specified using the CORBA IDL and is freely available [5]. This specification drove AspectTAZ implementation in Java. ObjectTAZ fits utilities requirement: UCA (Utility Communication Architecture) 2.0 specification [6](p9) mentioned that CORBA is a candidate platform to support TASE.2 services. AspectTAZ enhances previous results by allowing TASE.2 functions over CORBA, RMI and SOAP/XML.

4.2. AspectTAZ

The design of AspectTAZ is split in two phases: implementation of the TASE.2 client and server as one standalone Java program, specification and use of the deployment aspect, which is brand new.

4.2.1. Design of AspectTAZ

The implementation of AspectTAZ follows mostly the design of ObjectTAZ written in C++. An adaptation has been made. The IDL specification guides the implementation of TASE.2 standard objects in Java. The differences are the following. There is some adaptation between the two implementations due to the differences between Java and C++, and to supported libraries. In the context of AOP, a standalone program represents both TASE.2 client and server. As a consequence, there is no need to code the so called object server implementations needed by CORBA. This feature will be taken into account by the deployment aspect automatically as specified in aspect configuration. The code is organized as follow in a class Run that initializes the application:

```
public class Run {
  public static void main (String[] args) {
    ...
    Server serverAspectTAZ = new Server();
    Client clientAspectTAZ = new Client();
    ...
    //specific for running aspects
    serverAspectTAZ.init();
    serverAspectTAZ.start("serverAspectTAZ",serverAspectTAZ);
    clientAspectTAZ.start("clientAspectTAZ", clientAspectTAZ);
  }
}
```

As this class is only concerned with the business logic of the application, its programming is much more like developing a centralized application: no details related to distribution disturbs the code. Rather, distribution is

purely a non-functional concern, it will be added during the deployment phase of the application.

4.2.2. Applying the deployment aspect to AspectTAZ

One of the main features of JAC is the ability to deploy remotely and dynamically an application from an administration console. Although developers can write their own customized deployment aspect, we felt the need to provide a standard implementation for this recurring task. The deployment aspect provided with JAC is based on the notion of JAC server. A JAC server, much like other application servers (e.g. Sun Enterprise Java Beans EJB or .Net from Microsoft), runs on a network node and hosts JAC objects within a so-called container. The server provides a remotely accessible interface to deploy and invoke objects. Several flavors of JAC servers exist depending on the underlying communication protocol used. Currently JAC servers for Java RMI and CORBA exist, and a JAC server for SOAP is under implementation. Each server is identified by an URI string associated to the protocol used (e.g. rmi://some.host/anId for RMI or an IOR string for CORBA).

The grain of deployment is the object. Like in any other OO approach, a JAC application is a set of interoperating objects (either business objects, or aspect objects implementing a non-functional concern). JAC objects are named, either explicitly by developers who choose an unique name, or implicitly by JAC with the object class name extended by an unique instance number. The deployment policy is defined in a text-based descriptor giving:

- the localization of each object of the application, i.e. giving the relationship between the set of object names and the set of running JAC server names,
- the initial localization of client stubs for JAC objects.

If the first point is rather classical, the second one allows to pre-deploy access stub on known client hosts. Client stubs can be later on dynamically and transparently downloaded, but this initialization step speeds up the execution of the application. For application using numerous objects and client stubs, the exhaustive definition of all localizations is hard to handle; the syntax of the deployment descriptor allows to specify regular expression to summarize the policy. For instance, the file deployment.acc contains each aspect involved in the application:

```
deploy s0 clientAspectTAZ s1;
deploy s0 serverAspectTAZ s2;
createStubsFor clientAspectTAZ s1 s2;
createStubsFor ServerAspectTAZ s2 s1;
```

The following descriptor: specifies that the object named clientAspectTAZ (resp. serverAspectTAZ) from node s0 is to be deployed on node s1 (resp. s2) and that a stub for clientAspectTAZ hosted on s1 (resp. serverAspectTAZ on node s2) is to be created on nodes s1 (resp. s2). s0, s1 and s2 are logical hosts, they can be located on the same host or on different hosts. Remember that a JAC server runs on each node able to receive deployable JAC objects.

The deployment phase in JAC is depicted in Figure 2. JAC follows the specification described in the deployment.acc file: it creates stubs and migrates java objects. Deployment uses RMI and serialization.

Figure 3. illustrates a TASE.2 client and a TASE.2 server while interacting through their implementations clientAspectTAZ and serverAspectTAZ in Java using RMI.

4.3. Performance of AspectTAZ

This section gives performance measurements on the simple client/server program of Figure 3. Three operations are measured in Table 1 and Table 2. Table 1 presents results for a get and a set from a client to a server. Table 2 gives the time taken to provide values from the server to the client periodically. These three operations are benchmarked under different configurations. Column one: the client and the server are a centralized application (without the deployment). Column two: the client and the server are a distributed application on two logical sites but on one computer. Column three: the client and the server are a distributed application on two computers. The results shown in the next tables are in milliseconds.

	1	2	3
Get	6	471	508
Set	3	831	863

Table 1. get and set operations.

	1	2	3
Meas. 1	11	849	876
Meas. 2	10	298	420
Meas. 3	11	201	361
Meas. 4	10	258	153

Table 2. Periodic transfers.

In Table 1. column 1, when AspectTAZ runs as a centralized application, get and set take between 3 and 6 ms. When AspectTAZ is performed as a distributed application (columns 2, 3) get and set take between 470 and 865 ms which shows an important overhead. Around 50% of the time is devoted to perform RMI. The other 50% comes from the JAC runtime. It adds code to allow the dynamic management of aspects. This states the cost of the flexibility introduced by the JAC framework and how it handles aspects dynamically.

Table 2. states the same phenomenon with periodic transfers. In lines 2, 3, 4, Table 2. shows also a well-

known effect of Java programs. Execution times decrease significantly after the first utilization of Java Classes.

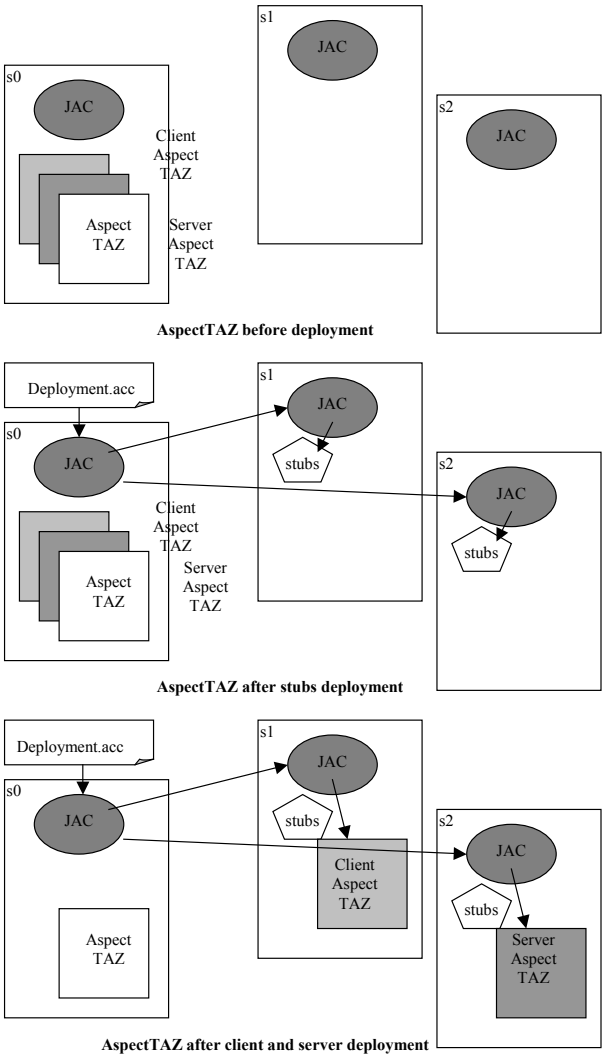

Figure 2. Phases of AspectTAZ deployment.

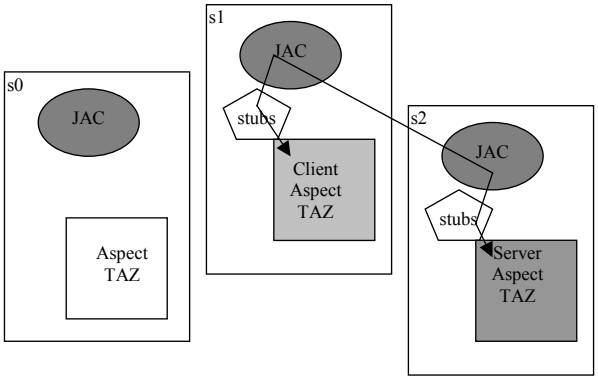

Figure 3. AspectTAZ running.

Work in progress targets performance licks. Current optimization deals with the code added to handle dynamic management of aspects. If performance becomes a more stringent requirement than flexibility, statically compiled code can be produced to reduce overhead. This code can embed the deployment and the distribution aspects. In such a case, the original JAC framework could be used as a rapid prototyping environment in the design phase of a factory communication system.

4.4. Alternate Trends for AspectTAZ Design

We built AspectTAZ like an industrial messaging service, quite in a "classical way". But, TASE.2 can be seen as the specification of a variable based distributed shared memory. Indication points are replicated both on the client and on the server, and TASE.2 defines consistency management rules to update them. We could have chosen the replication aspect supported by JAC to replicate TASE.2 Indication Points. JAC offers the well-understood and widespread Sequential Consistency model [17] (replicas contain the same value, programs perform totally ordered read or write accesses to data) but it is too poor to address a time based consistency model required by TASE.2. However, this suggest to enrich consistency aspects of JAC with the model provided by TASE.2 Such a feature would help application programmers.

Access Control is a general requirement for exchanges that cross WANs like production data exchanged between control centers and power plants. Access control is taken into account within TASE.2 standards, but its implementation is let to suppliers in the version of the standards we worked with. With JAC we have the opportunity to handle access control as an aspect independently. Access control is a built-in aspect of JAC. This ability eases architecture design: access control is handled outside the industrial messaging protocol without disturbing the functional code making the programming task much more simpler.

5. Conclusion

The use of AOP is efficient but not something immediate for developers. There is some kind of revolution to achieve to be able to design correctly your application on a per-aspect oriented programming basis. It is more difficult than coming from classical programming to object oriented programming. Aspect composition is a key issue of a successful AOP based implementation. In our experiment aspect composition was the most difficult task. We saw also that aspects need to be written by skilled developers. And finally, some expertise is required to integrate aspects and application code gracefully and efficiently. Authors of JAC are involved in the project and it was a big help. Despite these difficulties, we believe that AOP is a promising approach; it only needs a lot of training.

Our results demonstrate the viability of our AOP for factory communications. AOP matches users' needs

allowing the same industrial messaging service to run over CORBA and RMI. Future works in this area will extend the JAC deployment aspect with a SOAP/XML personality (results due for end of September). Our project addresses software engineering issues, at a different level, its goals match the same users' requirements than the OMG Model Driven Architecture [18]: design once, be able to work with different middlewares independently.

Our work is more generic than only a new way to develop industrial messaging services. Our industrial messaging service can become an aspect itself inside the JAC framework. Such a new feature would ease factory application design.

Finally, a close analysis to TASE.2 functions shows that it can be the basis of a real-time peer-to-peer data sharing service for the trading of energy. AspectTAZ could be the next generation: a real-time platform independent peer-to-peer data sharing service for energy trading

References

[1] AOPSYS, JAC. http://jac.aopsys.com, April 2002.

[2] AspectS, http://www.prakinf.tu-ilmenau.de/~hirsch/Projects/Squeak/AspectS/, April 2002.

[3] E. Becquet, M. Abdallah, E. Gressier-Soudan, F. Horn, L. Bacon, "Object Oriented Timed Messaging Service for Industrial Ethernet: a Fieldbus like Architecture for Power Plant Control and Factory Automation", *Fourth IFAC Conference on Fieldbus Systems and Their Applications (FeT'2001)*, pages 14-16 November 2001. Nancy, France.

[4] R. Boissier, E. Gressier-Soudan, A. Laurent, L. Seinturier, "Enhancing Numerical Controllers using MMS Concepts and a CORBA based software bus", *International Journal of Computer-Integrated Manufacturing (IJCIM)*, 14(6), 2001, Taylor and Francis Editor.

[5] COM-RT, http://comrt.sourceforge.net, April 2002.

[6] EPRI, "Utility Communications Architecture Version 2.0, Introduction to UCA Version 2.0, *Editorial Draft 1.0*. September, 1998.

[7] E. Gressier, M. Lefebvre, S. Natkin, "TCP/IP Manufacturing Applications: an experiment with MMS over RPC", *ULPAA'95. Sidney 1995*, ftp://ftp.cs.su.oz.au/bob/ULPAA/101-GRE.ps.gz.

[8] E. Gressier-Soudan, M. Epivent, A. Laurent, R. Boissier, D. Razafindramary, M. Raddadi, "Component oriented control architecture, the COCA project", *Special Issue on Manufacturing, Microprocessors and Microsystems Journal*, 23(2):95-102, September 1999, Elsevier Science.

[9] E. Gressier-Soudan, "Prototyping a CORBA based MMS -Industrial Communications with CORBA", *OMG Technical Meeting. Burlingame. California USA, September 10-15 2000*, ftp://ftp.omg.org/pub/doc/mfg/00-09-16.pdf.

[10] G.Guyonnet, E. Gressier-Soudan, F. Weis, "COOL-MMS: a CORBA approach to ISO-MMS", *ECOOP'97. Workshop: CORBA: Implementation, Use and Evaluation,* Jyvaskyla, Finland, June 1997.

[11] IAONA. Ethernet in Automation. http://www.iaona-eu.com/home/home.php. December 2001.

[12] IEC, Utility Communications Specification Working Group, "TASE.2 Services and Protocol", Version 1996-08, IEC870-6-503, ICCP Inter-Control Centre Communications Protocol Version 6.1, August 1996.

[13] IEC, Utility Communications Specification Working Group, "TASE.2 Object Models", Version 1996-08, IEC870-6-802, ICCP Inter-Control Centre Communications Protocol Version 6.1, August 1996.

[14] KEMA-ECC, "ICCP User Guide", *Final Draft.* Mineapolis, October 8, 1996.

[15] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.M. Loingtier, J. Irwin, "Aspect-oriented programming", *In Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP'97)*, volume 1241 of Lecture Notes in Computer Science, pages 220{242. Springer, June 1997.

[16] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, W. Griswold, "An Overview of AspectJ", *ECOOP 2001, LNCS 2072*, June 2001.

[17] L. Lamport. "How to make a Multiprocessor Computer that correctly executes Multiprocess Programs", *IEEE TOC, 28(9)*, 1989.

[18] OMG Architecture Board MDA Drafting Team, "Model Driven Architecture (MDA)", *Document number ormsc/2001-07-01,* July 9, 2001. ftp://ftp.omg.org/pub/docs/ormsc/01-07-01.pdf.

[19] D. Parnas, "On the criteria to be used in decomposing systems into modules", *Communications of the ACM*, 15(12):1053-1058, 1972.

[20] R. Pawlak, L. Seinturier, L. Duchien, G. Florin, "JAC: A Flexible and Efficient Solution for Aspect-Oriented Programming in Java", *Reflection 2001. LNCS 2192,* Pages 1-24, September 2001.

[21] UBC, "Exploring an Aspect-Oriented Approach to Operating System Code", http://www.cs.ubc.ca/~ycoady/aspectc.html, April 2002.

[22] UBC, Apostle: Aspect Programming in Smalltalk, http://www.cs.ubc.ca/labs/spl/projects/apostle/, April 2002.

[23] L. Seinturier, A. Laurent, B. Dumant, E. Gressier-Soudan, F. Horn, "A framework for Real-Time Communication Based Object Oriented Industrial Messaging Services", *7th IEEE International Conference on Emerging Technologies and Factory Automation. ETFA'99*, Barcelona, Spain, October 1999.

[24] Valenzano, Demartini, Ciminiera, "MAP and TOP Communications", *Addison Wesley*, 1992.