As information technology provides industry with the means to identify new product and service opportunities, the increasing demand for these products and services places new demands on information technology. This cycle has created unprecedented requirements for integrated enterprise systems and applications, which are capable of modeling rapid changes in business policies, workflow and information processing. A new breed of systems we refer to as object-oriented enterprise frameworks (OOEFs) has emerged to address these requirements.

Our research has studied the trends in OOEFs [5, 6], the guidelines for developing OOEFs, and the economic benefits of enterprise frameworks [7]. OOEFs are designed to reduce the complexity and cost of enterprise systems, and therefore have become strategic assets for organizations across all business sectors. Evidence of this is reflected in the recent entry of flexible and extensible products for enterprise resource planning (ERP), manufacturing execution systems (MES), product data management (PDM), customer service/sales order processing, and other commercially available, framework-based applications. While the Object Management Group’s (OMG) Web site identifies several successes in mission-critical enterprise applications leveraging CORBA, it is not our intent to take a position in the CORBA versus DCOM debate—as we will see, interoperability is only one of many characteristics of an OOEF. Rather, we hope to discuss the pragmatic issues that face decision-makers, and to establish a formal set of criteria to be used in building or selecting an enterprise framework.

Framework requirements are generally defined by an independent standards body (CORBA is one example), a Software Vendor (Microsoft’s MFC, COM and DCOM, FASTech’s FACTORYworks, IBM’s San Francisco Project, Philips’ New York Project) or a Systems Engineering Group (Motorola’s CIM Baseline). Frameworks combine the best features of state-of-the-art programming languages, development environments, and tools. In addition, frameworks provide an extensive library of business objects supporting the intended application domain.

At one time or other, every developer or development team has created a framework. Such frameworks often

Identifying critical issues for selecting or building enterprise frameworks.

MOHAMED E. FAYAD, DAVID S. HAMU, AND DAVIDE BRUGALI
manifest themselves as undocumented or loosely documented standards for developing an application (or applications.) However, these and most of the current generation of enterprise frameworks do not satisfy the requirements for an OOEF. Our studies of enterprise frameworks suggest the characteristics, criteria, and challenges common in OOEFs, which we explore here.

**Mature Runtime Functionality**

An important characteristic of a good enterprise framework is that it provides mature runtime functionality within the specific domain in which it is to be applied. Any enterprise framework follows an evolution in time, which is called “the framework life span” [2]. In this life span, the basic architectural elements, which are independent from any specific enterprise application, are implemented first. Through its adoption in an increasing number of applications, the framework matures, yielding more concrete components, providing design and implementation solutions for increasingly difficult problems.

High-level objects, representing the major abstractions found in the problem domain may also mature within the framework. The framework application development methodology is often only crudely present in early instantiations of the framework. As the framework is applied to a wider array of enterprise challenges, the relationships between objects are clarified. This maturation proceeds along many parallel layers of the framework.

Early in the life cycle, a framework is less likely to reflect the level of completeness required for the application domain. Extensive customization is required, and business logic is created to fill the gaps left in the framework. The reason this happens is that a framework is flexible by design, yet flexibility works in opposition to the concept of validation and rule enforcement. Frequently, the rules that enforce the myriad relationships between objects are not fully understood early in the framework life cycle. Many rules of an enterprise are not understood by the people operating the business. Still, many more rules are not applied, because of a lack of systems support for changing policies or because the policies are exceptionally complex. OOEFs demonstrate in practice the ability to apply enforcement (or, when needed, to relax enforcement) of both complex policies and changing policies. However, this is an area where frameworks can be immature in the early stages of the OOEF life cycle.

Thus, it is important to consider goodness-of-fit and anticipated usage when selecting or developing an enterprise framework. As the framework matures, if the framework is a good fit in conception and is well suited for the anticipated application domain, it is likely to return dividends throughout the life cycle.

A suitable framework requires relatively little code to meet the user requirements for new enterprise applications—this is the challenge for the framework team. Framework developers or providers must have a clear understanding of the application domain in which the framework will be applied, and the design and implementation of the framework must reflect that understanding. It is not cost effective to select a framework offering significantly less functionality than that provided by an existing system. Therefore, mature runtime functionality is not only an essential characteristic of a good framework, but it is essential to the selection process. Therefore, mature runtime functionality is the first of our criteria for enterprise frameworks.

Nevertheless, some immature frameworks may yield a superior architecture and better support for the other characteristics of enterprise frameworks. Thus, mature runtime functionality is properly considered as but one of the criterion to be weighed during the design or selection process.

**Support for Extensibility, Tailorability, and Customizability**

The notion of extensibility is frequently prescribed within the OO paradigm. In fact, software is frequently extended in ways its developers have not anticipated [11]. Although it is implied by the fundamental notion of a framework, an important characteristic of an OOEF is that it must be extensible, tailorable, and customizable if it is to be useful. A framework is not intended to be used like an application. Instead, a framework is the basis for constructing and delivering applications, which are highly tailored and configured to a specific domain. Extensibility, tailorability, and customizability ensure the framework may adapt new constructs and more accurately model a specific domain.

Framework developers are challenged to provide support for extensibility through broad support for OO constructs. OOEFs should support extensibility through one or more OO programming languages. For a framework to be a truly OOEF, it should provide support for common OO constructs, such as polymorphism, inheritance, encapsulation, reuse, and persistence. Leveraging these constructs ensures the framework maximizes extensibility, tailorability, and customizability.

Customization is done through parameterization or by encoding functional specifications. Frequently, the two views, referred to as white box and black box approaches to reuse, may be simultaneously present
in an enterprise framework [8].

A framework enhances extensibility and customizability by providing explicit hook methods [6] and hotspots—architectural elements that allow an application to extend its stable interface. Framework extensibility is essential to ensure timely customization of new application services and features. Therefore, extensibility is the second factor of our evaluation criteria for enterprise frameworks.

A Catalog of Business Objects and Enduring Business Themes

The business objects catalog provides ready-to-use objects, which can be used to extend the framework. These objects should also be available as templates for constructing new business objects (BOs) based on specified operating scenarios captured in the functional requirements.

Whereas an enduring business theme (EBT) is the structure for the sentences that are found in the language of business, the BOs are instantiations of EBTs. They fill in the blanks, attributes or semantics in these sentence structures. The EBT is concerned with a business issue; namely, “what is being done”, not “how it is being done.” For example, in the transportation industry, the EBT is concerned with moving material from one location to another. The best way to identify EBTs is to look at the organization from its customer’s point of view. The customer cares most about what is being accomplished, and only to a lesser extent, how it is being done.

A truly enduring business theme is one that survives over time. This enduring quality can only be determined by examining the underlying business issues, rather than a narrow focus on some abstract technical notion of goodness [4]. Hence EBTs are related to the purpose of the business and BOs are instantiations detailing how the business is run at a discrete point in time. Thus, a good enterprise framework reflects the concise understanding of the enduring qualities of the business through EBTs plus discrete implementations reflecting the best practices known today.

OOEF developers must understand the purpose of the business as well as the current practices of the business. Therefore, the quality of an enterprise framework is measured in part on its ability to model both the enduring business themes and the current business practices.

A Workflow Management Metaphor and Enduring Business Processes

Workflow management and enduring business processes are key to an OOEF. Workflow management streamlines the complex interactions between objects that are found in large-scale OO applications. Proponents of frameworks go so far as to suggest that workflow mechanisms should eliminate the need for most application programming in the workplace [10].

Knowledge workers will be empowered to solve business problems in the context of a workflow leveraging the increasingly powerful tools available on the desktop. Modern workflow management tools provide a graphical design palate for workflow definition (see the figure appearing here). Nested state diagrams are well suited for the task of dynamic modeling of application workflows [10], although other process flow representations are equally applicable.

A workflow management metaphor provides the necessary modeling capabilities for constructing business processes. Well-designed frameworks should capture the enduring business processes (EBPs) or workflows that are fundamental to the target application domain. These enduring processes capture workflows that do not change over time. They are concerned with the high-level sequence of what is done such as the top-level state diagram. The same workflow metaphor also provides a means for representing detailed and dynamic business processes (the nested diagrams), that are concerned with the step-by-step sequence of activities used to complete the task today. For instance, the manufacture and assembly of cellular telephones involves the integration of a molded plastic chassis with the internal electronic components. The assembly process is relatively similar.
whether you are assembling a bulkier first-generation cellular telephone or the newer subpalm-size flip-phones. In the figure here, it is the nested diagram for “Perform Manufacturing Operation” that would change to fit the process for the new model wireless telephone assembly, suggesting a workflow for a manufacturing process. Each step in the process represents a sequence of business rules or a subprocess consisting of embedded workflows. For instance, “Perform Manufacturing Operation” may consist of a sequence of manufacturing operations, which are defined by a nested workflow. Similarly, the “Statistical Process Control” step may be constructed as a sequence of business rules or methods executed against the catalog of BOs residing within the framework.

Thus, a suitable enterprise framework reflects the understanding of the enduring business processes. Workflow management is commonly found in a variety of frameworks and traditional software products. Increasingly, workflow mechanisms have been incorporated into enterprise frameworks. Workflow provides a means to model both the enduring processes and the nested workflows. Therefore, we believe that workflow management and enduring business processes is a fundamental piece of our criteria for enterprise frameworks [4].

Achieving Software Stability
With software becoming an increasingly integral part of our lives, the field of software engineering is increasingly important. Unlike other engineering fields, however, the products produced through software engineering are often intangible. Also, unlike the products of other engineering fields, software products are unlikely to remain stable over a long period of time.

In hardware areas, the failure rates of products often start high, then drop low, and then go high again. Early in a hardware product’s life cycle, there are some problems with the system. As these problems are fixed, the failure rate of the hardware products drops. However, as hardware ages, physical deterioration causes the hardware to fail. In other words, the hardware wears out and the failure rate rises again.

Software, on the other hand, is not subject to the same wear and tear that hardware is. Seldom are there environmental factors that cause software to break. Software is a set of instructions, a recipe, for a piece of hardware to follow. There are no moving parts in software. There is nothing that can physically deteriorate. Software should not wear out; unfortunately, it does.

Countless authors in the field of software engineering have identified this problem. However, the software engineering techniques outlined by many software engineering authors have not achieved a good amount of stability in software projects.

This problem is more than just an inconvenience for software engineers and software users. The reengineering that is required for these software products does not come without a price. It is not uncommon to hear of these reengineering projects costing hundreds of thousands to millions of dollars. This does not take into account the opportunity cost of the reengineering process.

Software defects and deterioration are caused by changes in software. Many of these changes cannot be avoided. These changes can be minimized, however. When changes must be made to a software program, it is common that the entire program must be reengineered. It does not matter if the change required is due to new technology or a change in clientele. This reengineering process is ridiculous. The core purpose of the software product has not changed. Why, then, must the entire project be reengineered to incorporate a change?

If a desired characteristic of an enterprise framework is long-term stability, then the challenge to OOEF developers is to deliver a product that supports the notion of stability. The idea in this case is to identify those aspects of the environment in which the software will operate that will not change and cater the software to those areas.

The bulk of the engineering performed during a software project should be focused on those areas that will remain stable. Such an approach ensures a stable core design and, thus, a stable software product. Those changes that are introduced to the software project will then be in the periphery, since the core
was based on something that has remained and will remain stable. Since whatever changes must be made to the software in the future will be in this “periphery,” it will only be these small external modules that will have to be engineered. And, thus, we avoid the endless circle of reengineering entire software projects for minor changes [4].

The preceding discussion of enduring business themes and business objects provides a foundation for software stability. Probably the most important criterion for identifying EBTs and BOs is that they must be enduring. Both must be stable over time. Where they differ is in how they are stable.

Enduring business themes are completely stable both internally and externally. They have never and will never change. All the EBTs identified in the case studies mentioned previously are completely stable. Warehouses must always store goods in an efficient manner and serve their customers if they expect to stay in business. Similarly, kitchens must always facilitate meal preparation.

Business objects can change internally. Externally, however, they must remain the same. The warehouse case study has a perfect example of this [4]. In the warehouse study, the storage schema is a business object. Externally, it must always be a method of storing goods in the warehouse. Internally, however, it is an aggregate of industrial objects. Any of these industrial objects can change or be replaced. As long as the storage schema is still externally the same, the model is still valid. Item is another BO in this model. Internally, it does not matter if the item is a bathroom slipper or a telephone directory. Externally, it is a good that can be stored and that is all that matters to the system.

Industrial objects are those objects that can be changed at will. If the system is designed correctly, changing these objects should not cause changes to ripple through the rest of the system. For example, in a kitchen, the range can be replaced with either a barbecue or a food replicator. As long as it still facilitates the theme of meal preparation, the model remains intact. Do not take this to the extreme, however. In the warehouse example, one cannot replace the loading dock with a banana and expect this to work. The industrial objects must continue to fit with the BOs and EBTs to which they are attached [4]. Therefore, support for software stability is another factor in our criteria for design or selection of a suitable enterprise framework.

A Model for Distributed Objects and Scalability
Object distribution is a common feature for enterprise frameworks as today enterprise information systems integrate and coordinate geographically (often worldwide) distributed activities. A challenge for enterprise framework developers is the definition of distribution models that support interoperability among heterogeneous information systems. In fact, it is more often the case that groups of enterprises interconnect their information systems to create the so-called “Virtual Enterprises” [1]. Therefore, an important characteristic of enterprise frameworks is that they provide an integrated model for distributed objects, such as:

- Remote Procedure Calls/Remote Method Invocation

The distribution model must support transparent communication between objects over a distributed computing environment. In this way the distribution model supports scalability; as more users (clients) are added to the system, additional server instances can be added to improve throughput. Another important aspect of the distribution model is that object definitions or structure are preserved across the transport eliminating the need for the numerous data translation layers found in legacy systems.

Integration of Multiple Application Frameworks and Legacy Components
Enterprise frameworks are, by definition, the cornerstone of a system architecture. Therefore, they must provide the structure and tools for easy integration of multiple application frameworks and legacy components. Several problems are encountered when integrating multiple frameworks. An enterprise framework cannot easily put aside each of these problems. However it is important that the framework developers have considered each of these issues and that tools are provided to reduce the impact of these problems. The delivered framework should provide mechanisms for dealing with the more common and more challenging integration problems. We have classified some of the more fundamental problems as follows:

- Hollywood Principle: Concerns the issue of inversion of control (“Don’t call us, we’ll call you.”) A framework’s control loop invokes the application code or methods when appropriate. The problem occurs when two or more frameworks call the application code and each assumes ownership of the main event loop of the application.
- Capitalization Principle: In the migration to
enterprise frameworks, we would like to exercise the policy “Don’t throw anything away, use and reuse as much as you can.” However, integrating existing tools and legacy components as classes with the framework is not a trivial task and may cause typing conflicts.

- Overlapping Principle: This problem occurs when two or more frameworks have the same real-world components with different representations and services [9].
- Composition Principle: This problem occurs when a real-world component has to be modeled by integrating parts of functionality from different frameworks.
- Gap Principle: This problem occurs when integrating two or more frameworks and the resulting architecture does not cover the desired application’s requirements. This problem is one of framework gap [12].
- Impedance Principle: This problem occurs when two or more integrated frameworks with different architectures fail to interoperate. This is due to the fact that many framework components exist in multiple forms which are not readily shared or communicated with another framework.

Although this is not a complete list, it serves as a starting point for some of the issues encountered when integrating multiple frameworks. This characteristic is closely related to framework adequacy, which requires frameworks to support a sufficient, yet narrow focus. When frameworks are narrow in focus, support for integration is critical, because it is often necessary to integrate many best-of-breed frameworks or legacy components to produce a top-to-bottom solution.

An open application programming interface (API) provides a ready means of integrating best-of-breed applications to the framework. The API allows these integrated applications to participate within the framework and provides a means for exposing the implied or actual object models of integrated applications to the framework. However, an open API is only a part of the solution to the problem of integrating multiple frameworks.

Mature frameworks will reflect an understanding and consideration of the problems described previously. In addition, they will provide recommended solutions to these integration problems. Framework providers must possess a working knowledge of the business issues that fall outside of the framework domain and design frameworks with an awareness of the types of applications that will be integrated in the overall architecture.

**Platform Independence or Portability**

Platform independence and portability ensure that the framework supports all of the platforms in use by an organization. In addition, platform independence is important because it ensures that other applications can interact with applications built from the framework.

Platform independence is closely related to the concern for open APIs and support for distributed objects. However, it is not sufficient to specify that frameworks should be open or provide support for distributed computing. Platform independence is essential for framework developers working in large organizations supporting multiple platforms. Generally the framework team will be required to provide support for the major platforms operating in an organization. Thus, failing to provide platform independence will diminish the scope of use and acceptance of the framework. Platform independence is especially important for framework vendors; if a framework is not open it will certainly result in lost sales opportunities when customers are not willing to be limited to the platforms supported by the framework.

**Mature Framework Documentation**

Documentation is key to the success of the framework. It is the single most distinguishing characteristic of a high quality software product. Many mature frameworks are lacking in one or more of the qualifications described previously. Mature documentation ensures that a framework gets used. In addition, high quality documentation ensures that design and implementation standards are reflected in all of the application content built with the framework. Therefore, mature framework documentation ensures reuse and maintainability.

Framework providers are challenged to deliver mature documentation with their products. Framework documentation must describe the purpose of the framework, how to use the framework, and the detailed design of the framework. In addition, Enterprise frameworks include concise documentation of the framework architecture, configuration and development tools, and object and API references. These documents are best deployed online and employ a standard hypertext presentation. Furthermore, mature framework documentation provides references and links to numerous examples including sample source code and templates to be used in developing specific application components.

A recommended approach for incorporating all of these features and benefits into the framework documentation is to structure the documentation as a set of patterns. Patterns are a design documentation tech-
The application domain for which the framework has been conceived; this is necessary to allow users to choose the right framework for their needs.

- The design of the framework in terms of objects and their relationships; this helps the designer to extend the framework, explaining previous design decisions and their motivations.

- The specification and implementation of the framework classes; this provides the starting point for the implementation of concrete applications.

In general, the application developer need not understand all aspects of the framework, but should be able to search for patterns that reveal techniques for solving a specific class of problems using the framework. Framework documentation is extensible allowing the framework team to insert details about framework extensions directly into the framework documentation. In addition, the framework documentation should be closely tied to the modeling tools provided within the framework, so that documented analysis and design elements are easily translated into functional models.

**Support for the Role Object Pattern and Ease of Use**

The enterprise framework must be intuitive and easy to understand. Anyone (technical or non-technical) should be able to understand it and use it. Therefore the enterprise framework must be user friendly and to do so, the role object pattern must be utilized in the enterprise framework. Enterprise frameworks are adapted to different client's needs through transparently attached role objects, each one representing a role that the framework objects play in that client's context. The object manages its role set dynamically. By representing roles as individual objects, different contexts are kept separate and system configuration is simplified.

A role is a client-specific view of an object playing that role. One object may play several roles, and the same role can be played by different objects. An instance of a core concept belonging to the accounting domain may play several roles. For example, 'an accountant' can play the role of 'a controller' or 'an auditor'. In the financial domain, 'an accountant' can play the role of 'financial advisor'. These three roles can also be played by the same accountant—in real life as well as in the system model. A combination of design patterns can be used [6], collectively called the Role Object Pattern, to attach accounting and financial domains' roles to core concept objects.

**Web- and E-business-Ready**

As the Internet is used increasingly for commerce, enterprises will find ways to work more effectively through electronic data transfer and through online transactions over either identical or disparate frameworks. In either case frameworks will be an enabling technology, without which the benefits of the Internet will not be fully realized. Frameworks will empower cooperative supply chains and enable business partners to work together more efficiently and manage complex business transactions in minutes or even seconds with little or no human intervention. The end result is a faster response to a customer's needs and faster time to market. Thus, another characteristic of an enterprise framework is that it is Web- and e-business-ready. Framework developers and providers need to challenge themselves to build solutions that are designed from the start to be Web-ready.

**Support for Separation of Concerns**

Separation of concerns is at the heart of framework development. The OO paradigm works well only if the problem at hand can be described with relatively simple interface among objects. The core complexity is that concurrent/distributed systems have more than one dimension. Features such as scheduling, synchronization, fault tolerance, security, testing and verification and validation are all expressed in such a way that they crosscut different objects. Hence, a simple objects interface is violated and traditional OO benefits no longer hold.

A current attempt to resolve the issue is the aspect-oriented architecture. Aspect-oriented architectures are intended to address the multidimension structure of any framework. Therefore, we need to distinguish between components and aspects. Aspects [3] are defined as properties of a system (framework) that do not necessarily align with the system's functional components. These properties include communication, performance, load balancing, synchronization, and scheduling. Aspect properties tend to cut across functional components, increasing their interdependencies, and resulting in a code-tangling problem. The aspect-oriented paradigm (AOP) helps to retain the
advantages of OOP and helps to avoid the tyranny of
dominant decomposition. The goal is to achieve an
improved separation of concerns in both design and
implementation. We suggest that the application of
aspect-oriented architecture is an important charac-
teristic of OOEFs and it helps to overcome the chal-
lenege of separation of concerns.

Sound Investment: Framework Economics
An OOEF project represents a formidable invest-
ment, whether building the framework or integrating
a commercially acquired framework. A characteristic of that investment is that it should be a
sound investment. Return on investment can take
many forms. A framework vendor should realize the
high level of flexibility and functionality necessary to
ensure a high volume of product sales. In addition,
we have seen that framework vendors often realize a
reduction in overall engineering resources required to
support the core elements of the framework once the
framework has reached the marketplace.

Framework developers also realize efficiencies in the
development of application content and collateral
products by exploiting the framework’s tools and mod-
eling facilities. Frameworks generate significantly
higher revenues for those technology and service firms
competing for a piece of the framework technology
marketplace. Framework vendors and service providers
are highly specialized in an industry or commercial
domain with teams of well-compensated experts.

Early adopters of enterprise frameworks pay a pre-
mium for products and services. However, the frame-
work economy will ensure that even very complex
solutions will be deployed in a turnkey fashion and
while hourly and capital costs may increase, lengthy
implementation schedules have been compressed from
years to months and months to weeks or even days.

Life-cycle costs are also reduced due to the empha-
sis on standards, which are exploited in an enterprise
framework development environment. The emphasis
on standards results in formal processes for reuse and
accelerates the time to deliver new application con-
tent. This means that it will cost less to produce or
extend applications for the enterprise. It is a funda-
mental challenge for framework developers to deliver a
solution that offers an aggressive return on invest-
ment over conventional products, and prospective
customers of enterprise framework vendors seek
assurances that the return on investment is tangible.

Conclusion
While the notion is in its infancy today, it is our
belief that OOEFs will be at the core of software
engineering technology in the 21st century. There-
fore, a proper context and understanding of the con-
cepts, characteristics, challenges and evaluation
criteria are needed for decision-makers. We have
shown that good OOEFs are characterized by sup-
port for the essential tools used in systems analysis,
modeling, design, development, testing, mainte-
nance, and administration. Therefore, although this
is an emerging area in software engineering, it
reflects the natural evolution in the development of
software engineering tools and practice. We have
also provided a taxonomy of characteristics and cri-
tera for evaluating enterprise frameworks for suit-
ability and have described these characteristics and
criteria with a focus on the common enterprise com-
puting challenges systems engineers commonly
struggle with today. In this sense, we believe we have
identified a reliable set of criteria that can be used in
the building or selecting enterprise frameworks.

References
1. Brugali, D., Dragomirescu, D., Galarraga, S., and Menga, G. A cus-
tomizable software infrastructure for Virtual Factories development. In
Proceedings of the International Conference on Robotics and Automation,
Detroit, MI, May 1999.
3. Constantinides, C.A., Bader, A., Elrad, T.Z., Fayad, M.E., and Neti-
nant, P. Designing an aspect-oriented framework in an object-oriented
5. Fayad, M.E. and Johnson, R. Domain-Specific Application Frameworks:
6. Fayad, M.E., Schmidt, D., and Johnson, R. Building Application
Frameworks: Object-Oriented Foundations of Framework Design. Wiley,
NY, 1999.
7. Hamu, D.S., Fayad, M.E. Achieving bottom-line improvements with
8. Johnson, R.E. and Foote, B. Designing reusable classes. Journal of
9. Mattsson, M., Bosch, J., and Fayad, M.E. Framework integration:
Problems, causes, and solutions. Commun. ACM 42, 10 (Oct. 1999),
81–87.
11. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W.
12. Sparks, S. Benner, K. and Faris. C. Managing object-oriented framework
reuse. IEEE Computer, theme issue on managing object-oriented soft-

Mohamed E. Fayad (fayad@cse.unl.edu) is J.D. Edwards
Professor in the Computer Science and Engineering Department at
the University of Nebraska, Lincoln; www.cse.unl.edu/~fayad
David S. Hamu (dlhamu@tx.net.com) is manager of systems
integration at TRW Global Enterprise Solutions in Phoenix, AZ.
Davide Brugali (brugali@polito.it) is a postdoctoral research
fellow in the Department Automatica e Informatica at the Politec-
nico di Torino in Turin, Italy; www.polito.it/~brugali

© 2000 ACM 0002-0782/00/1000 $5.00