DOWNLOAD PDF FOURTH INTERNATIONAL WORKSHOP ON OBJECT-ORIENTED REAL-TIME DEPENDABLE SYSTEMS

Chapter 1 : Selected Publications

The object-oriented paradigm has been used extensively both in CASE tools for modeling and design of distributed real-time systems as well as in software tools for developing supervisory control systems.

These components may extend the useful lifetime of a legacy system by making a subset of its data and functionality available to new applications, including intranets and extranets. However, the availability of the legacy systems often does not meet the requirements of these new applications. This paper presents a strategy for migration to enterprise components that improves the effective availability of legacy functionality and data by storing component data in native, object form. We propose the use of persistence services embedded in the component execution environment to store persistent enterprise data. We compare our approach to alternatives and relate our initial experience in applying this strategy to command and control systems. Introduction Object-oriented approaches have long been investigated for building complex software systems. More recently, objects have been applied to wider-ranging software systems, including real-time systems [13 and systems with challenging scalability and availability requirements, like intranets and extranets [2]. Object-oriented software design and implementation addresses many difficulties, including modeling complex behavior and data, specifying precise yet abstract system descriptions, and separating concerns in an implementation. Objects have also been used to reengineer or encapsulate legacy systems [3]. Such systems are generally hard to reengineer, for the following reasons: It is difficult to understand what the system does and to separate what it does well from what it does poorly. In the United States Department of Defense DoD environment, the organization that built the system is often no longer under contract. The tools used to develop the system may be unavailable. For example, there are many systems in use today for which no compiler is available. The system may use outdated technologies e. Often, these systems were designed according to the organizational needs known when the system was built. For example, a legacy order processing system may process orders at night local time for an organization with one office. If the organization later expands to include offices world-wide, the system may need to process orders continuously, perhaps with allowances for infrequent situations. This paper considers techniques for improving the availability of such legacy systems, without necessarily reengineering them. We are specifically concerned with timely access to current data values, where values are updated sporadically. Of course, this approach can only give the illusion of a continuously available legacy system, since we do not propose to reengineer those systems. CORBA greatly simplifies the construction of a distributed object system, allowing the use of a heterogeneous mix of programming languages and machines. This approach can improve the availability of a system of systems by allowing clients to reconnect to a wrapped system, e. Software For these technology collaborate wrapped legacy systems to improve components and other reasons, software component was developed to group objects which closely and to allow users to compose I I 1. Software component technology has been a vision for complex systems development for at least 30 years [IO]. While the author is specifically interested in command and control systems, the current commercial impetus for these technologies is business information systems, which manage payroll, accounting, and the like for almost every kind of organization. Our conceptual architecture for component- based software systems is given in Figure 1. Component Framework Dlsmbuted Object Management 0. Conceptual architecture of component-based software As shown in Figure 1, component frameworks improve application development by providing a significantly more abstract virtual machine. This abstraction makes certain services available and allows components to collaborate e. Programmers can now build software that relies on framework functionality for important but domain-independent tasks, such as persistence or reliable inter-component communication. More importantly, new components can be built to work closely with existing components thanks to the explicitness of the framework- imposed requirements. Users can gain the ability to configure software to address unanticipated needs, blurring the distinction between users and programmers. The notion of applications is replaced by the

more flexible notion of configurations of components. This can allow the use of smaller configurations that require less memory, disk space, and most importantly supporting software. Many component technologies utilize a container that gives components access to framework services. This paper primarily considers frameworks that support data- managing components rather than graphical user interfaces GUIs. Because data is shared and accessed by potentially many clients, execution of these components must be robust and scalable. The container is typically responsible for services like durable persistence of component data, flexible and reliable communication with remote objects, etc. As illustrated in Figure 1, component technologies use a distributed object communication mechanism, which 96 should be optimized for local. For example, the same programming mechanism with different optimizations might be used for intra-thread, intra-process, and inter-process calls. Such components have interfaces described in, e. Thus, the container approach, coupled with appropriate services, is useful for the data availability concerns of an intranet application. A software component technology formally defines a component model, which describes the form of component interfaces and features. While interfaces are familiar from prior programming models, the features define aspects of a component that can be changed configured and the messages that can be used to connect the component to others combine. To compose a component-based system, one configures and combines components. A component will typically encapsulate multiple objects with method definitions. Components are the implicit units of administration, system accounting, loading into the environment , and fault tolerance or fault containment [9]. We are primarily interested in enterprise components that implement object-oriented data models and business logic. Enterprise components often must be kept consistent with legacy information sources systems and databases. Enterprise components present data to client user interface components, respond to user updates from client components, utilize data management and related services provided by containers , and interact with existing systems and databases. Figure 2 depicts these enterprise components. Enterprise components and containers In Figure 2, enterprise components shown at the middle are deployed within a container. We have used EJB and Java to implement these in our experiments. Client components word processing, map display, etc. Availability concerns Let us continue the running example of an order processing system that is being migrated to provide data from legacy databases to an intranet. We can build enterprise components to model the data of interest, and if desired can incorporate new business rules as object methods implemented by these components. Technologies like EJB allow many possible forms of connections between an enterprise component and a legacy database. Some of the options are: The enterprise component manages a read-only view of the legacy data. Updates can only be achieved through the legacy interface. The component manages a live view of the legacy data. Updates to component data are immediately forwarded to the legacy database. Systems connected directly to the legacy database can rely on the currency of the data they access. This is only possible if the legacy database can support a continuous stream of updates, which is unlikely in our example. The component manages a view of the data and adds new data elements as needed. Updates to new elements are not reflected back to the legacy database but clients that are unaware of the new elements can continue to interact directly with the legacy database. The databases must be reengineered new columns, tables and indexes as new elements are added. Option 1 gives a quick way to publish legacy data on an intranet, but does little to improve the availability of the data unless the desired access is read-only. In the remair option 5 for mal extranet applica preclude the us e. Availabili Enterprise CO intranets and e models of data a can be adapted t that supports thc available additic transactions and context in whi presents altern component data. With options , the availability legacy databases are reflected in the mts. For example, consider an update to. Option 2 cannot make the new her legacy systems connected directly il after the nightly updates. With option 3, framework to store this new data but legacy relied upon for legacy data. As with often insufficient for the availability -anets and extranets. Option 4 reflects this and 1 step to reengineer legacy databases to t already present in some form. Even if le resulting systems will still reflect the ons for access to legacy data of the r of this paper, we discuss the use of g legacy data available to intranet and Ins. Note that this option does not of the existing database technologies 3MS. We do assume that the legacy

neet availability requirements for the 1. We also consider how and where to added as the organization evolves. The framework enterprise components can also make 1 data management services, such as ersistence. This section discusses the persistence services are used, and les for persistence of enterprise 3. Context for enterprise component persistence Object-oriented approaches are the central context for enterprise components and the frameworks and services that support them. An organization adopting a component- based approach will often make a significant investment in skills development and planning. For these pragmatic reasons, an organization may wish to use object-oriented techniques for new software, and gradually reduce its reliance on legacy software development techniques. Object persistence services are favored if they can support a seamless connection between objects in execution environments and on disk, without negatively impacting performance or reliability. As with all information systems, a boundary must be drawn between persistent and transient data. More generally, simple rules are needed to describe the lifetime of data managed by enterprise components. The availability of this enterprise data is affected by the way in which the boundary is established. First, a persistence mechanism that requires the enterprise component developer to explicitly identify persistent objects puts availability decisions in the hands of the developer. In general, the enterprise component developer should not need to be aware of the way that the component is used, including the data availability requirements that its interfaces expose. Second, a persistence mechanism that requires the developer to use additional data models such as serialized streams or strings for storage interrupts the desired seamless representation of objects. In this case the developer must make decisions about how to store enterprise component data, and these decisions naturally affect how the data can be used, which includes the resulting availability of that data. In a sense, such mechanisms build availability decisions into components in much the same way that these decisions are built into legacy systems. We are primarily concerned with the use of enterprise components in large systems. Organizations with multiple sites, using multiple databases and technologies, are likely to implement operational models as software artifacts here, enterprise components. For these systems, scalability is of utmost importance. Although these aspects give rise to the availability requirements we have assumed, they also can justify the costs of tools and techniques for addressing scalability.

Chapter 2 : CiteSeerX â€” Citation Query Valuedriven assignment in object-oriented real-time dependable

The following topics were dealt with: real-time CORBA; real-time object-oriented modeling and analysis; real-time distributed objects and frameworks; real- Fourth International Workshop on Object-Oriented Real-Time Dependable Systems (Cat.

Chapter 3 : Rob Pettit - Publications

Get this from a library! Fourth International Workshop on Object-Oriented Real-Time Dependable Systems: proceedings, January , , Santa Barbara, California, USA.

Chapter 4 : Phil Koopman: Historical Projects

Fourth International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'99) () Real-Time Object-Oriented Modeling and Analysis.

Chapter 5 : George Wang - Publication

[IEEE Fourth International Workshop on Object-Oriented Real-Time Dependable Systems - Santa Barbara, CA, USA ()] Engineering Solutions for the Next Millennium IEEE Canadian Conference on Electrical and Computer Engineering (Cat

DOWNLOAD PDF FOURTH INTERNATIONAL WORKSHOP ON OBJECT-ORIENTED REAL-TIME DEPENDABLE SYSTEMS

No 99TH) CCECE - Persistent enterprise components: improving the availability of.

Chapter 6 : Publications â€“ DEPEND Research Group

Fourth International Workshop on Object-Oriented Real-Time Dependable Systems: proceedings, January , , Santa Barbara, California, USA / sponsored by IEEE Computer Society.

Chapter 7 : [WORDS] IEEE Workshop on Object-oriented Real-time Dependable Systems

This workshop, which is the eighth in the highly successful WORDS series, continues its theme in integrating three computer system engineering technologies (CSETs): Object-oriented CSET, Real-time CSET, and Dependable CSET. The workshop is intended to be a forum for exchange of newly recognized.

Chapter 8 : Phil Koopman Publications & Patents

[PDF Download] Scientific Engineering of Distributed Java Applications: 4th International Workshop.

Chapter 9 : Welcome to SEG Research Group

Fourth International Workshop on Object-Oriented Real-Time Dependable Systems, Santa Barbara, California, USA, January , Proceedings. IEEE Computer Society , ISBN X [contents].