DISTRIBUTED OBJECT VISUALIZATION FOR SENSOR-DRIVEN SYSTEMS

Christopher D. Gill, Washington University, St. Louis, MO

David L. Levine, Washington University, St. Louis, MO

Carlos O'Ryan, Washington University, St. Louis, MO

Douglas C. Schmidt, Washington University, St. Louis, MO

Abstract

Many sensor-driven systems, such as those for avionics mission computing and for manufacturing process control, have stringent timing requirements for processing sensor data. Furthermore, many of these systems must manage multiple sources of sensor data simultaneously. Our previous work has shown that sensor-driven systems can be implemented efficiently and predictably using a real-time CORBA Event Service. This approach allows designers of real-time systems to leverage the benefits of flexible and open distributed computing architectures, such as those defined in the CORBA specification, while still meeting real-time requirements for efficiency, scalability, and predictability. To build and manage these types of systems, application developers and test engineers must be able to monitor and visualize the systems' real-time behavior.

This paper describes how we have extended our distributed object visualization environment (DOVE) framework to monitor the timing behavior of a real-time application that generates and processes two separate streams of simulated sensor data events. The principal contributions of this paper are: 1) applying the DOVE framework to a simulated sensor-driven application, 2) extending the DOVE framework to support new application requirements, and 3) demonstrating and visualizing quality of service (QoS) control for multiple event streams within a real-time CORBA Event Service.

Introduction

Many real-time systems, particularly hard real-time systems, are not correct unless they can meet their deadlines. Rate monotonic analysis and other scheduling strategies have been developed to help ensure that real-time systems achieve this goal. However, scheduling strategies alone offer little guidance during the debugging and testing phases. Traditional debugging techniques do not help either because they can change the behavior of the system. Therefore, a distributed object visualization environment (DOVE)[1] framework can offer an alternative, less obtrusive, way to observe how a real-time system works at run-time. Thus, it can be a powerful tool in the real-time system development cycle.

Key design forces in the sensor-driven systems domain impose significant constraints on visualizing the real-time behavior of these systems, however. Our initial DOVE framework lacked the extensions described in

[1] M. Kircher and D. Schmidt, "DOVE: A Distributed Object Visualization Environment", *C++ Report*, March 1999.

this paper; it simply provided capabilities to visualize distributed object computing (DOC) systems. However, the design forces stemming from the real-time nature of sensor-driven systems motivated us to extend the DOVE framework to resolve these forces.

The remainder of this paper is structured as follows. The Design Forces section describes key design forces for real-time sensor-driven systems. The DOVE Architecture section describes the architecture of the original DOVE framework. The Design Forces Resolved section describes how key design forces are left unresolved by the original DOVE architecture, and explores how our extensions to the framework resolve these key design forces. The Demonstration Architecture section describes an integrated architecture suitable for demonstrating this technology. Finally, the Concluding Remarks section summarizes the work and describes our plans for future development and measurement using DOVE.

Key Design Forces and Requirements

Software for visualizing real-time system behavior must address the following significant challenges that are not faced when visualizing the behavior of non-real-time systems. First, the visualization framework must not interfere with the correct timing behavior of the real-time system. Second, the framework must be flexible to address diverse system behaviors, particularly when sources of non-determinism appear. Finally, the framework must support both independent and correlated visualizations of distinct event streams. This section examines each of these key design forces.

Unobtrusive Visualization: As noted above, visualizing the behavior of a real-time sensor-driven system is a valuable engineering tool, particularly in the validation phase of the system lifecycle. However, information about real-time system behavior must be collected and displayed without interfering with the overall timing behavior of the system. For example, in dynamically scheduled avionics mission-computing systems[2], monitoring and displaying behavioral information must not cause critical operations to miss their deadlines. Furthermore, excessive impact of the visualization on non-critical operations should be avoided, as well. The visualization mechanisms used to instrument the sensor-driven system must be efficient and relatively deterministic.

Visualizing Non-deterministic Behavior: Dynamically scheduled sensor-driven systems can produce non-deterministic behavior for certain operations when they are overloaded. For some dynamically scheduled sensor-driven systems[3], a low level of overload is an acceptable operating characteristic to maximize utilization of system resources. Therefore, visualization software should be able to detect and provide a reasonable visualization of the load on the system, as well as the effects of overload on system behavior.

Visualizing Distinct Streams: Multi-sensor systems may produce distinct streams of sensor data. The real-time behavior of these systems often depends on the interactions between multiple streams. For example, two data streams may be processed at different priorities, e.g., processing for the higher priority stream may preempt processing for the lower priority stream. Likewise, there may be dependencies between the streams. Therefore, the visualization framework must consider data streams both individually and in the aggregate.

DOVE Architecture

The DOVE framework developed at Washington University in St. Louis, Missouri, in the Center for Distributed Object Computing (DOC) consists of the following major components: a DOVE-enabled application, an optional DOVE management information base

[2] D. Levine, C. Gill, and D. Schmidt, “Dynamic Scheduling Strategies for Avionics Mission Computing”, 17th IEEE/AIAA DASC, 1998.
[3] B. Doerr, T. Venturella, R. Jha, C. Gill, and D. Schmidt, “Adaptive Scheduling for Real-Time, Embedded Information Systems”, 18th IEEE/AIAA DASC, 1999.

(MIB), a DOVE agent, and a DOVE-enabled browser. The DOVE-enabled application provides visualization information to a DOVE agent, which can log the information to a DOVE MIB. A DOVE agent can also pass the information to a DOVE-enabled browser, which displays various visualizations of the information it receives from one or more DOVE agents.

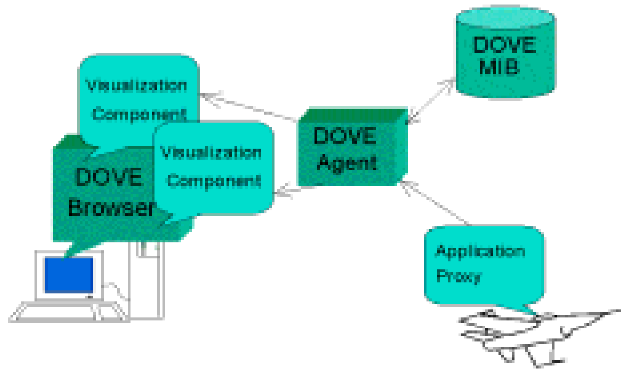


Figure 1. DOVE Components

The components in DOVE and their high-level relationships are shown in Figure 1. The DOVE application contains a proxy that communicates with the DOVE agent. The proxy augments the application; it is not necessary to modify the application to make it DOVE-enabled. The DOVE agent performs service advertisement, change notification, data reduction and correlation, visualization configuration, and MIB management tasks. DOVE agents reside on the target processors, and must therefore be very lightweight in terms of their resource consumption.

To reduce the load on the operational system, the DOVE browser and the visualization components usually run on a host monitoring processor. To facilitate development and deployment of visualization components in such diverse client environments as monitoring workstations or web browsers, the DOVE browser and visualization components are written in Java^{TM}. Thus, the DOVE framework must decouple the timing behavior of the browser and visualization components from the timing behavior of the application, so that the application's real-time behavior is not adversely affected by the behavior of the Java Virtual Machine^{TM}.

The DOVE framework described in this section is implemented in the context of *The ACE ORB* (TAO)[4]. TAO is open source CORBA-compliant real-time ORB middleware developed at Washington University in St. Louis, Missouri, in the Center for Distributed Object Computing (DOC). The *Simulator* example distributed with TAO[5] provides the DOVE application and DOVE browser portions of this framework. These components interact with TAO's Real-Time Event Service[6], which acts as the DOVE agent.

Design Forces Resolved

The original DOVE framework addresses many of the common design forces encountered when visualizing the behavior of DOC systems. These forces include *monitoring distributed components, monitoring components written in different programming languages,* and *monitoring components running on different end-system platforms.*

The original DOVE framework also addresses some of the design forces for real-time sensor-driven systems. For instance, using a DOVE agent to relay information from the DOVE application to a DOVE-enabled browser offloads processing from the application and may reduce the impact of a DOVE-enabled browser on the DOVE application's real-time behavior. Likewise, using a CORBA Event Service as the DOVE agent[4] allows the visualization framework to scale between collections of components ranging in size from

[4] http://www.cs.wustl.edu/~schmidt/TAO.html
[5] The Simulator example is found in the {$TAO_ROOT}/examples/Simulator/ directory
[6] T. Harrison, C. O'Ryan, D. Levine, and D. Schmidt, "The Design and Performance of a Real-time CORBA Event Service", *IEEE JSAC*, 1998.

small to large. Furthermore, the Real-time Event Service provided with TAO[5] allows filtering and correlation of events to reduce overall distributed system load.

However, the original DOVE framework does not address several key design forces for real-time systems. In particular, aspects of the DOVE architecture had to be extended to balance the following design forces: 1) unobtrusive visualization, 2) visualizing non-deterministic behavior, and 3) visualizing distinct event streams. This section describes how we extended the DOVE framework to address each of these key design forces.

Unobtrusive Visualization

Three features of the original DOVE architecture facilitate our extensions to support unobtrusive visualization. First, the DOVE architecture's flexibility allows us to add lightweight *decorators*[7] to the application components. Thus, existing application code need not be modified to use DOVE, or to provide customized forms of real-time monitoring. Second, the DOVE agent and its proxies are themselves lightweight, which means that unnecessary overhead is avoided in our extensions. Third, the DOVE architecture loosely couples visualization timing to application timing via the Active Object pattern[8], which we extend by controlling thread priorities to further reduce the impact of visualization on the application's real-time behavior.

As shown in Figure 1, a DOVE application contains the unmodified target application and a proxy. The proxy adapts the application to the DOVE agent API. This minimizes the impact on the application by allowing a suitable, application-specific data collection interface. For example to visualize the timing behavior of an application component, we can extend the DOVE framework via a thin decorator around the component. This decorator collects the time of entry and exit of a call to the component, and then packages up the times in an event that it forwards to the DOVE agent.

The DOVE agent and its proxies must provide deterministic and non-intrusive behavior because they reside on the embedded target system. For example, packaging monitored information in a proxy must not consume excessive CPU cycles at the expense of critical application processing. By reducing the overhead for collecting monitor information to a few simple steps, DOVE takes only a small number of CPU cycles away from the application for each monitoring event.

The DOVE architecture also uses the Active Object pattern to decouple the thread of control in the application from the thread of control in the browser. This pattern prevents timing delays in the application due to blocking while Java™–based visualization components are executing. We can reduce this impact even further by configuring thread priorities in the Real-time Event Service so that monitored events are forwarded at a lower priority than the application's threads of control.

Visualizing Non-deterministic Behavior

Non-critical operations in sensor-driven systems using hybrid static/dynamic scheduling techniques[9,10] may exhibit non-deterministic behavior. For example, a built-in system test operation for a manufacturing process control system may run periodically. Moreover, it may

[7] E. Gamma, R. Helm, R. Johnson and J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, 1995.

[8] R. Lavender D. Schmidt, "Active Object: an Object Behavioral Pattern for Concurrent Programming", Pattern Languages of Program Design, J. Coplien, J. Vlissides and N. Kerth, eds., Addison-Wesley 1996.

[9] C. Gill, D. Levine, and D. Schmidt, "The Design and Performance of a Real-Time CORBA Scheduling Service", International Journal of Time-Critical Computing Systems, special issue on Real-Time Middleware, Wei Zhao, ed., Kluwer, 1999.

[10] D. Stewart and P. Khosla, "Real-Time Scheduling of Sensor-Based Control Systems", *Real-Time Programming*, Halang W. and Ramamritham, K. eds., Pergamon Press, Tarrytown, NY, 1992.

be preempted by higher priority operations, such as system alarm processing. When the test operation is preempted, no test data will be logged until the higher priority operations complete. Therefore, timing of test operation data updates will be non-deterministic.

Even in the face of such non-determinism, the work done by such low priority operations is often still useful to the system. Below, we describe two classes of real-time algorithms that can withstand a moderate level of timing non-determinism: 1) those that can withstand missing data but not timing delays, and 2) those that can withstand timing delays but not missing data.

Resilience to Missing Data: Algorithms that are time-sensitive, but resilient to missing data points may simply drop data that arrives after its deadline. For example, a non-critical video streaming operation may drop an occasional frame without noticeable degradation in video quality.

Resilience to Timing Delays: Other low priority algorithms may not be able to tolerate missing data points, but may be able to withstand timing delays for data. For example, the "Persian" Recursion[11] (PR) algorithm, which performs successive refinement of a drawn image, cannot tolerate missing data in a sequence but can tolerate delays in the arrival of data itself. It provides an effective way to visualize the real-time behavior of time insensitive algorithms, as the completeness of the pattern corresponds to the degree of refinement of the algorithm. Real-time algorithms, such as built-in system tests that are timing-insensitive, can be visualized using DOVE framework extensions that are similar to the PR algorithm extensions described below.

As Figure 2 illustrates, the PR algorithm subdivides a square by drawing two perpendicular bisectors of the square, and then performing this step recursively on each of the four quadrants of the original square, halting each recursive descent when the remaining squares are only a pixel wide. The colors of the bisecting lines are a function of the colors of the sides of the square being subdivided in the current recursive step. Different color mappings supplied to this algorithm will produce different resulting patterns, some of which are suggestive of the pattern in a Persian rug (thus the name of the algorithm).

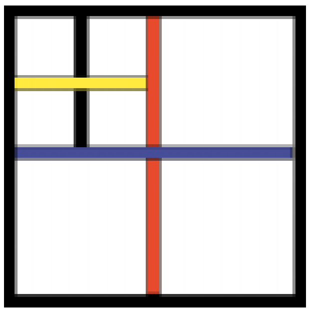


Figure 2. "Persian" Recursion

Because the color assignment at each recursive step of the PR algorithm depends on the colors assigned in the previous step, dropping the data for one recursive step and proceeding to the next one could produce an incorrect pattern. However, delaying the data will simply retain the pattern at its current level of refinement without affecting the correctness of subsequent refinements.

Extending the DOVE Framework: Applications such as video streaming, which were previously common only in non-real-time systems, are increasingly useful for real-time systems. For example, the ability to stream terrain footage from a ground-based tracking center to a number of remote search aircraft can greatly assist in search and rescue operations.

Video streaming is also a representative example an algorithm that is resilient to missing data. The stream itself often provides adequate visualization of the real-time behavior of the application. Therefore, we focus on extending the DOVE framework to algorithms that can withstand data delays but not missing data.

The PR algorithm is a reasonable surrogate for this second class of algorithms. The rate of

[11] A. Burns, ""Persian" Recursion", Mathematics Magazine, vol. 70, no. 3, MAA, 1997.

update also provides an animated indicator of the rate of arrival of updates, as well as conveying any extended delays. Extending the DOVE visualization framework to support the PR algorithm consists of adding a new application component, a new browser data handler, and a new browser display component.

The application component calculates the colors and endpoints of the horizontal and vertical bisectors at each recursive step. The application component packages up the coordinates and colors for each pair of bisectors in a data event, and passes it, via the Application Proxy to the DOVE agent. In this case, the DOVE agent is TAO's Real-time Event Service, as shown in Figure 3.

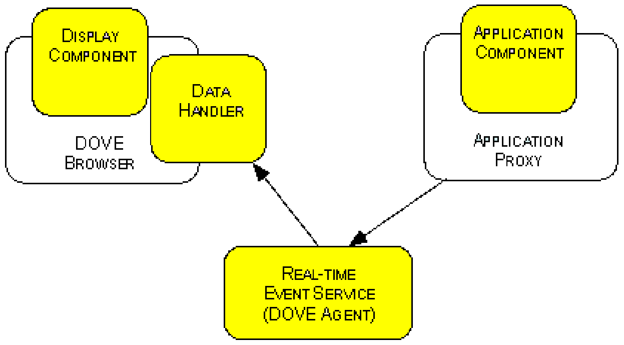


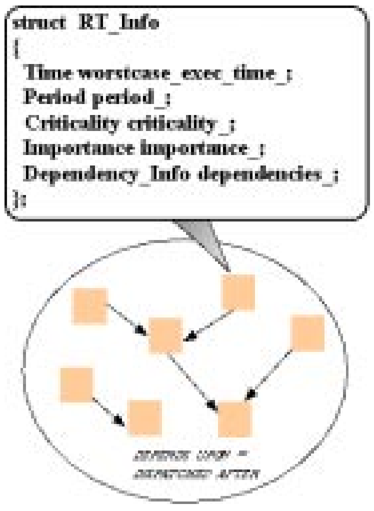
Figure 3. Extended DOVE Framework

The DOVE agent passes the data event to a DOVE-enabled browser, which invokes the new data handler that was registered for that type of event. The new data handler unpacks the color and endpoint data for each bisector, and updates an appropriate data structure, e.g., adds to a linked list of bisector structures, or draws the new bisectors on a bitmap image. The new browser display component, which is an *observer* of the data structure, updates its display by drawing the new line segment, or displaying the updated bitmap.

Visualizing Distinct Streams: Multi-sensor systems often produce distinct streams of data. These streams may be filtered or correlated prior to further processing in the application. The original DOVE framework allows visualization of the functional behavior of distinct or correlated event streams using the Real-time Event Service as the DOVE agent. For example, a DOVE-enabled browser can subscribe with the Real-time Event Service to receive the event types associated with the streams, so they can be visualized individually. In addition a DOVE-enabled browser can register to receive conjunctions and disjunctions of events in the streams, e.g., to visualize dependencies between the streams.

While the original DOVE architecture allowed us to visualize the functional aspects of distinct event streams, several aspects of real-time behavior are not supported directly. In particular, priority inheritance policies for correlation of events may be needed. For example, if two streams are at different priorities, then correlating the events in order to visualize them may need to be performed at a lower priority than the streams themselves.

The real-time behavior of distinct event streams depends on operation QoS characteristics, specifically those specified in the real-time information descriptors[12] (RT_Info) used by TAO's Real-time Scheduling Service. The primary fields in the RT_Info descriptor are shown in Figure 4.



```
struct RT_Info
{
  Time worstcase_exec_time_;
  Period period_;
  Criticality criticality_;
  Importance importance_;
  Dependency_Info dependencies_;
};
```

[12] D. Schmidt, D. Levine, and S. Mungee, "The Design and Performance of Real-Time Object Request Brokers", *Computer Communications*, Elsevier, Vol. 21, No. 4, April 1998, pp. 294—324.

Figure 4. Application QoS Data

Based on the values in the RT_Info descriptors, and the dependencies between them, TAO's real-time scheduler assigns static priority and sub-priority values, and configures the dynamic run-time operation-dispatching behavior. Some of these field values are determined automatically through configuration runs. Application developers supply others. By specifying the characteristics of visualization operations as well as application operations, and explicitly defining the dependencies between them, the impact of visualization on distinct application event streams, even at different priorities, can be reduced**.**

Concluding Remarks

In this work, we have identified key design forces for visualizing sensor-driven systems. The forces can be resolved by extending the DOVE framework to avoid imposing constraints on the application programming model or real-time system behavior in order to visualize the system.

We have described two classes of operations that are resilient to non-determinism in operation scheduling. The first are resilient to missing data; the second to timing delays. For the time insensitive class, we have shown how a simulated sensor-driven application, i.e., the "Persian" Recursion algorithm, can be used to model the real-time behavior of a class of successive refinement algorithms in the face of non-deterministic scheduling of operations.

We have also described techniques for specifying QoS characteristics for visualization and application operations. This preserves the real-time behavior of distinct event streams at different priority levels, while correlating events for visualization.

The extensions to the DOVE framework described in this paper provide a basis for implementing monitoring and visualization capabilities in distributed real-time systems, using standards-based COTS OO middleware, such as a real-time implementation of. CORBA. This framework and the sample application are distributed with The ACE ORB (TAO), which is freely available via the open source software model. For more information about TAO and DOVE, please see the following URL: http://www.cs.wustl.edu/~schmidt/TAO.html.