Rule-Based Visualization in a Computational Steering Collaboratory

Lian Jiang, Hua Liu, Manish Parashar, and Deborah Silver

Dept of Electrical and Computer Engineering,
Rutgers University, Piscataway, NJ 08854, USA
{lianjian,marialiu,silver,parashar}@caip.rutgers.edu

Abstract. In this paper, we introduce the concept of rule-based visualization for a computational steering collaboratory and show how these rules can be used to steer the behaviors of visualization subsystems. Feature-based visualization allows users to extract regions of interests, and then visualize, track and quantify the evolution of these features. Rules define high level user policies and are used to automatically select and tune the appropriate visualization technique based on application requirements and available computing/network resources. Such an automated management of the visualization subsystem can significantly improve the effectiveness of computational steering collaboratories in wide area Grid environments.

1 Introduction

A computational steering collaboratory is an environment in which geographically distributed scientists can collaboratively investigate complex and multidisciplinary simulations using online monitoring, remote visualization and computational steering techniques. Computational steering in such an environment not only shortens the period between changes to parameters and the viewing of the results, but also enables a what-if analysis which makes cause-effect relationships more evident [1].

The ability to flexibly manipulate the visualization subsystem in a computational steering collaboratory is important for both computational steering and multi-user collaboration as visualization is typically the basis for interactive monitoring and steering. For example, scientists often tend to look at the isosurface of a scalar field. These are specified using thresholds and are generally manipulated interactively using the visualization subsystem. However, for large-scale long-running simulations it may not be feasible to download an entire dataset or even one timestep of the dataset to a visualization platform. Therefore, visualization routines are co-located at the simulation platform. However, this can prevent them from being interactive and thresholds have to be selected a priori, which may not be most effective. Using rule-based control, scientists can enable the visualization subsystem to automatically pick the appropriate threshold. In this paper we present such a rule based visualization system. Rules are decoupled from the system and can be externally injected into a rule base. This

M. Bubak et al. (Eds.): ICCS 2004, LNCS 3038, pp. 58–65, 2004.

allows scientists to conveniently add, delete, modify, disable, and enable rules that will control visualization behavior. Scientists can manipulate rules not only before the simulation, but also during the simulation. For example, while the simulation is running the scientist may specify a rule such as "if the number of the extracted regions is greater than 100, then increase the threshold by 5". The visualization subsystem will automatically adjust the threshold when the rule conditions evaluate to true. However, if the scientist knows from monitoring the first 200 timesteps that the value of x in the second rule is not appropriate and should be changed, she can modify the rule during the simulation and the modified rule will be applied to the rest of the simulation.

Rules can also be used to support collaborative visualization in heterogeneous environments where the collaborators' resources and display capabilities may differ. Some visualization techniques are more computation intensive than others, or may require more powerful graphic capabilities to display. For example, rendering an isosurface with millions of polygons may be too compute/network intensive for a thin client such as a PDA. Either the polygons can be reduced using straightforward triangle decimation techniques, or a more abstract feature-based representation can be displayed [2]. Such automated adaptations can be simply achieved using a rule such as "if there are more than 10k triangles, then display a higher level abstraction".

The rule based visualization subsystem presented in this paper builds on Discover, which is a computational collaboratory for interactive grid applications and provides the infrastructure for enabling rules to be dynamically composed and securely injected into the application, and executed at runtime so as to enable it to autonomically adapt and optimize its behavior [3]. In this paper, we integrate the rule mechanism into a feature-based visualization subsystem and demonstrate how this can be used to improve monitoring, steering and collaboration.

2 The Discover Computational Collaboratory

Discover is a virtual, interactive and collaborative PSE that enables geographically distributed scientists and engineers to collaboratively monitor, and control high performance parallel/distributed applications using web-based portals [3]. As shown in Figure 1, Discover provides a 3-tier architecture composed of detachable thin-clients at the front-end, a network of web servers in the middle, and the Distributed Interactive Object Substrate (DIOS++) [4] at the back-end.

DIOS++ enables rule based autonomic adaptation and control of distributed scientific applications. It is composed of 3 key components: (1) autonomic objects that extend computational objects with sensors (to monitor the state of an object), actuators (to modify the state of an object), access policies (to control accesses to sensors and actuators) and rule agents (to enable rule-based autonomic self-management), (2) mechanisms for dynamically and securely composing, deploying, modifying and deleting rules, and (3) a hierarchical control network that is dynamically configured to enable runtime accesses to and management

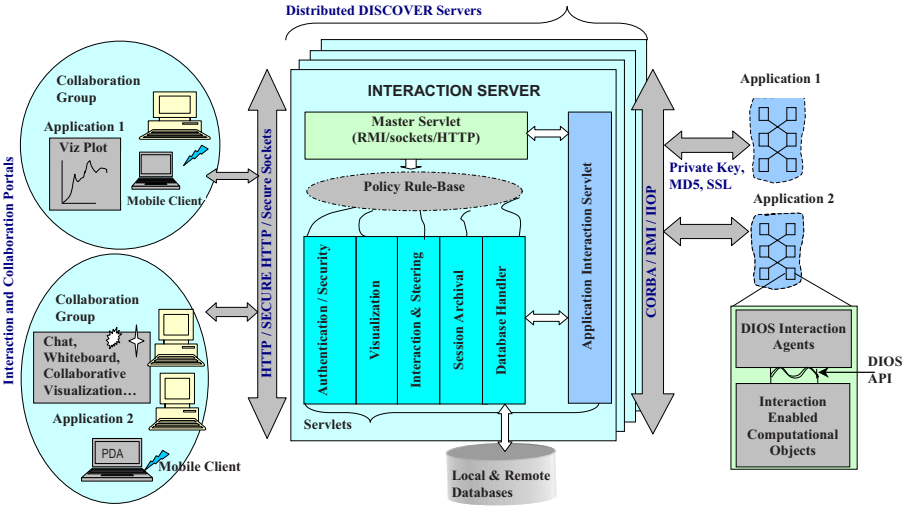

Fig. 1. Architectural Schematic of the DISCOVER Computational Collaboratory

of the autonomic objects and their sensors, actuators, access policies and rules. The rule engine responds to the users' actions (e.g. rule creation, deletion, modification, activation, deactivation) by decomposing and distributing the rules to corresponding rule agents, collecting rule execution results from the rule agents and reporting to users. The rule engine also controls and coordinates rule agents. Rules are evaluated and executed in parallel by the distributed rule agents. Priority and dynamic locking mechanisms are used at each rule agent to solve rule conflicts. The Discover collaboratory allows users to collaboratively interact with, interrogate, control, and steer remote simulations using web based pervasive portals, and has been used to enable remote feature extraction and tracking for the 3D Ritchmyer Meshkof compressible turbulence kernel (RM3D) [5,6].

3 Feature Extraction and Feature Tracking

Feature-based visualization allows scientists to extract regions of interests, and then visualize, track and quantify the evolution of these features. The first step in feature-based framework is defining the feature of interests. In [2], features are defined as connected regions which satisfy some threshold since this is a basic definition for most visualization routines (such as isosurfaces or volume rendering).

Features are tracked from one timestep to the next to capture how the features evolve [2]. Feature events can be classified into the following categories: continuation (a feature continues from one timestep t_i to the next t_{i+1}), creation (a new feature appears in t_{i+1}), dissipation (a feature in t_i does not appear in t_{i+1}), bifurcation (a feature in t_i splits into one or more features in t_{i+1}) and

amalgamation (two or more features from t_i merge into one in t_{i+1}). Feature tracking allows events to be catalogued and enables complex queries to be performed on the dataset. Queries include data-mining type exploration, such as "how many new regions appear in timestep t_i?", "in which timesteps do large regions merge?", or "where does a large region break up?". The framework of feature based visualization is shown in Figure 2. First, the features (shown as the isosurfaces in this figure) are extracted from the time-varying data. Then the evolution of these features are tracked and the features are abstracted for different levels, one example of which shown in this figure is ellipsoid. Finally, the quantification, such as volume, mass, centroid, of the features is computed. The tracking, abstraction and quantification results can be used for enhanced visualization and event querying in a heterogeneous collaboratory.

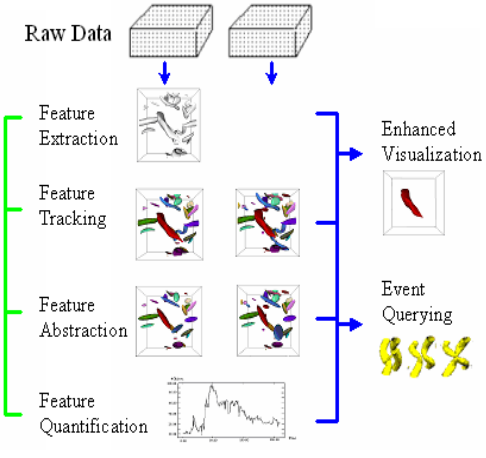

Fig. 2. Feature based visualization

After the features are extracted, the feature attributes, such as isosurface, mass and volume, can be computed. The features can also be abstracted using a simpler shape. One such reduced representation is an ellipsoid that provides an iconic abstraction to blob-like regions. An ellipsoid can capture the essential shape and orientation of the region with a limited number of parameters (center + axes) as opposed to an isosurface which may contain thousands of triangles. For huge datasets which may contain hundreds or thousands of evolving regions, and ellipsoid may provide a suitable first abstraction. Other types of abstractions include more line-like regions like skeletons [7] and vortex cores [8], or for vector fields, critical points and critical curves [9]. For scalar fields, contour trees [10] can also provide an abstraction. A contour tree is a graph which describes the topological changes of the isosurface with respect to the threshold.

An example of ellipsoid abstraction is shown in Figure 2. The scientist can choose between displaying ellipsoids or isosurfaces depending on the requirements and computing/network resources.

In addition to the standard feature extraction and tracking implemented within AVS, in [5], a distributed AMR version of feature extraction and tracking is described. AMR (Adaptive Mesh Refinement) is a technique used in computational simulations to concentrate grid points in areas where the errors are large. This results in a set of nested grids with varying resolutions. In distributed AMR grids, features can span multiple refinement levels and multiple proces-

sors; therefore, tracking must be performed across time, across levels, and across processors. The distributed algorithm [5], called *Ftrack*, is implemented within Discover and run in-situ, i.e., together with the simulation so that the data does not have to be copied or stored for visualization processing.

4 Rule Definitions

Ftrack, the feature-based visualization system integrated with Discover [3], is capable of self-management based on runtime user-defined rules. The rules are categorized into: (1) *Steering rules* apply intra-function management (e.g. changing the runtime behaviors of the visualization functions by dynamically altering their parameters). (2) *Configuration rules* apply inter-function management (e.g. organizing the visualization functions by selecting the appropriate functions to be executed). Examples of rules and the related sensors, actuators applicable to *Ftrack* are presented below:

Steering Rule: The rule for choosing an appropriate threshold mentioned in Section 1 is an example of steering rule.

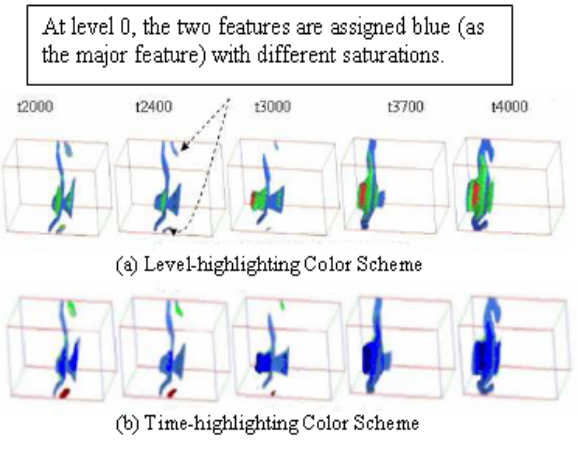

Fig. 3. Two color schemes used in AMR tracking for RM3D: (a) a level-highlighting color scheme is used. (b) a time-highlighting color scheme is used.

Another example given here is a rule for changing the color schemes. The level-highlighting color scheme gives each level a different hue and assign different saturation to each feature, as shown in figure 3(a). On the other hand, the time-highlighting color scheme assigns a different hue to the feature in different timestep. If a feature spans several levels, then each part of the feature falling in a different level is assigned a different saturation of the feature's hue, as shown in figure 3(b).

Rules can be used to select the color schemes. For example, when the number of features is very small (below 3), there is no need to highlight time-tracking. A scientist may prefer to see the tracking in refinement levels. For this purpose, *Ftrack* exposes one sensor *getAverageNumberofFeatures()* and two actuators *useLevelHighlightingScheme()* and *useTimeHighlightingScheme()*. A corresponding rule could be:

```
IF getAverageNumberofFeatures()<3 THEN useLevelHighlightingScheme()
                                  ELSE useTimeHighlightingScheme()
```

Configuration Rule: Scientists may choose to vary the visualization techniques based upon the computing/network resources available at the time of visualization. For example, when the number of grid cells in the dataset exceeds a threshold, a scientist at a thin client may want to display ellipsoids instead of isosurfaces. Thus, *Ftrack* should expose a sensor getNumCells() (to get the number of the cells in the input volume) and four actuators: *enableEllipsoidFitting()*, *disableEllipsoidFitting()*, *enableIsosurface()*, *disableIsosurface()*. Corresponding rules could be:

```
Rule 1: IF getNumCells()>10K
        THEN {enableEllipsoidFitting(); disableIsosurface()}
        ELSE {disableEllipsoidFitting(); enableIsosurface()}
```

The scientist can modify this rule with operations like "change 10K to 50K" or "switch THEN and ELSE statements". If a scientist is working on a PDA, which typically has poor graphics resolution as well as limited memory capacity, the rule specified could be:

```
Rule 2: IF isPDA()=TRUE THEN {enableEllipsoidFitting();
                              disableIsosurface()}
```

5 Rule-Based Visualization Using Discover

The visualization subsystem, *Ftrack*, which performs the distributed feature extraction and tracking algorithms, is designed as a DIOS++ object. As shown in Figure 4, the *Ftrack* object consists of three actors, each managing a part of the visualization task:

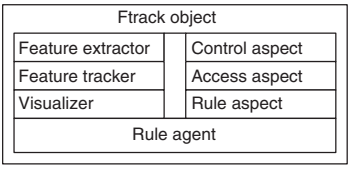

Fig. 4. Ftrack Object Structure

Feature extractor: this actor extracts interesting features, and computes the geometry attributes(e.g. isosurface) and quantification attributes (e.g. volume, mass, tensor, etc.) for each feature. Furthermore, it can calculate global statistics such as the average number of features, the variation of feature numbers, etc.

Feature tracker: this actor tracks the extracted features. The input is the feature information coming from feature extractor and the user steering commands from portals or rule operations from the rule engine. The output is the tracking information which can be represented as a set of graphs. A graph can be a feature tree [5] describing the feature correlation tracked on different levels, or a direct acyclic graph (DAG) [2] describing the feature correlation tracked on different timesteps.

Visualizer: this actor includes various visualization modules supported by the feature extractor and feature tracker. It utilizes the results coming from the other two actors to create visualizations of the features (for example, an isosurface rendering displaying quantification) and sends the image or data to the Discover portals.

The rule agent (RA), which is dynamically generated by the rule engine, is shared by the three actors. The rule engine and the RA form the interface between the actors and rules. Control aspects in the rule handler define sensors and actuators for the three actors and allow the state of the visualization subsystem to be externally monitored and controlled. The access aspect controls access to these sensors/actuators based on user privileges and capabilities. The rule aspect contains the rules that are used to automatically monitor, adapt and control the object. The rule-based visualization process is summarized below. Detail of the operation and evaluation of rule-based control in DIOS++/Discover can be found in [4].

Initialization and Interaction: During initialization, the application uses the DIOS++ APIs to register its objects, export their aspects, interfaces and access policies to the local computational nodes, which will further export the information to the Gateway. The Gateway then updates its registry. Since the rule engine is co-located with Gateway, it has access to the Gateway's registry. The Gateway interacts with the external access environment and coordinates accesses to the application's sensor/actuators, policies and rules.

At runtime the Gateway may receive incoming interaction or rule requests from users. The Gateway first checks the user's privileges based on the user's role, and refuses any invalid access. It then forwards valid interaction requests to *Ftrack* and forwards valid rule requests to the rule engine. Finally, the responses to the user's requests or from rules executions are combined, collated and forwarded to the user.

Rule Deployment and Execution: The Gateway transfers valid rules to the rule engine. The rule engine dynamically creates rule agents for *Ftrack* and other objects if they do not already exist. It then composes a script for each agent that defines the rule agent's lifetime and rule execution sequence based on rule priorities. For example, the script for the rule agent of *Ftrack* may specify that this agent will terminate itself when it has no rules, or that Rule 1 of the Configuration Rules (see Section 4) has higher priority than Rule 2.

While typical rule execution is straightforward (actions are issued when their required conditions are fulfilled), the application dynamics and user interactions make things unpredictable. As a result, rule conflicts must be detected at runtime. In DIOS++, rule conflicts are detected at runtime and are handled by simply disabling the conflicting rules with lower priorities. This is done by locking the required sensors/actuators. For example, configuration rules Rule 1 and Rule 2 conflict if *getNumCells()* is less than 10K (*disableEllipsoidFitting()* and *enableIsosurface()* should be used) while *isPDA()* is TRUE (*enableEllipsoidFitting()* and *disableIsosurface()* should be used). Assuming Rule 1 has higher priority, the script will inform the rule agent to fire Rule 1 first. After Rule 1 is executed, interfaces *enableEllipsoidFitting()* and *disableIsosurface()* are locked during the period when getNumCells() is less than 10K. When Rule 2 is issued, it cannot be executed as its required interfaces are locked. The two interfaces will be unlocked when *getNumCells()* becomes greater than 10K. By modifying the rules in the rule base through the Discover portal, the scientist can promote

the priority of Rule 2 to be higher than that of Rule 1 so that the visualization subsystem always displays ellipsoids if *isPDA()* is TRUE.

The rule agent at *Ftrack* will continue to exist according to the lifetime specified in its script. If the scientist modifies the rules, for example, promoting the priority of Rule 2, the rule engine can dynamically modify the script to change the behavior of the rule agent.

6 Conclusion

This paper presented a rule based visualization system that improves the flexibility of visualization in a WAN-based computational steering collaboratory. Rules can be used to steer in-situ visualization and aid in data mining. In a heterogeneous environment, rules can help the scientists specify the type of interaction and visualization desired based on system capabilities.

This work was done at the Vizlab and TASSL, Rutgers University. This material is based upon work supported by the National Science Foundation under Grant Nos. 0082634, 9984357, 0103674, and 0120934. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

1. Mulder, J.D., Wijk, J.J.v., Liere, R.v.: A survey of computional steering environments. Future Generation Computer Systems (1999)
2. Silver, D., Wang, X.: Tracking and visualizing turbulent 3d features. IEEE Trans. on Visualizatin and Computer Graphics (1997)
3. Mann, V., Matossian, V., Muralidhar, R., Parashar, M.: Discover: An enviroment for web-based interaction and steering of high-performance scientific applications. Concurrency-Practice and experience (2000)
4. Liu, H., Parashar, M.: Dios++: A framework for rule-based autonomic management of distributed scientific applications. In: Proc. of the 9th International Euro-Par Conference (Euro-Par 2003). (2003)
5. Chen, J., Silver, D., Jiang, L.: The feature tree: Visualizing feature tracking in distributed amr datasets. In: IEEE Symposium on Parallel and Large-Data Visualization and Graphics. (2003)
6. Chen, J., Silver, D., Parashar, M.: Real-time feature extraction and tracking in a computational steering environment. In: Proc. of Advanced Simulations Technologies Conference (ASTC). (2003)
7. Reinders, F., Jacobson, M.E.D., Post, F.H.: Skeleton graph generation for feature shape description. Data Visualization (2000)
8. Banks, D., Singer, B.: A predictor-corrector technique for visualizing unsteady flow. IEEE Trans. Visualization and Computer Graphics (1995)
9. Helman, J., Hesselink, L.: Representation and display of vector field topology in fluid flow data sets. IEEE Computer (1989)
10. Sural, S., Qian, G., Pramanik, S.: Segmentation and histogram generation using the hsv color space for image retrieval. In: Int. Conf. on Image Processing. (2002)