Visiting Vehicle Ground Trajectory Tool

The International Space Station (ISS) Visiting Vehicle Group needed a targeting tool for vehicles that rendezvous with the ISS. The Visiting Vehicle Ground Trajectory targeting tool provides the ability to perform both real-time and planning operations for the Visiting Vehicle Group. This tool provides a highly reconfigurable base, which allows the Visiting Vehicle Group to perform their work. The application is composed of a telemetry processing function, a relative motion function, a targeting function, a vector view, and 2D/3D world map type graphics.

The software tool provides the ability to plan a rendezvous trajectory for vehicles that visit the ISS. It models these relative trajectories using planned and real-time data from the vehicle. The tool monitors ongoing rendezvous trajectory relative motion, and ensures visiting vehicles stay within agreed corridors.

The software provides the ability to update or re-plan a rendezvous to support contingency operations. Adding new parameters and incorporating them into the system was previously not available on-the-fly. If an unanticipated capability wasn’t discovered until the vehicle was flying, there was no way to update things.

This work was done by Dustin Hamm of Johnson Space Center. Further information is contained in a TSP (see page 1). MSC-24763-1

Workflow-Based Software Development Environment

The Software Developer’s Assistant (SDA) helps software teams more efficiently and accurately conduct or execute software processes associated with NASA mission-critical software. SDA is a process enactment platform that guides software teams through project-specific standards, processes, and procedures. Software projects are decomposed into all of their required process steps or tasks, and each task is assigned to project personnel. SDA orchestrates the performance of work required to complete all process tasks in the correct sequence. The software then notifies team members when they may begin work on their assigned tasks and provides the tools, instructions, reference materials, and supportive artifacts that allow users to compliantly perform the work.

A combination of technology components captures and enacts any software process use to support the software lifecycle. It creates an adaptive workflow environment that can be modified as needed. SDA achieves software process automation through a Business Process Management (BPM) approach to managing the software lifecycle for mission-critical projects. It contains five main parts: TieFlow (workflow engine), Business Rules (rules to alter process flow), Common Repository (storage for project artifacts, versions, history, schedules, etc.), SOA (interface to allow internal, GFE, or COTS tools integration), and the Web Portal Interface (collaborative web environment).

SDA achieves software process automation using the Business Process Management (BPM) approach to managing the software lifecycle for mission-critical projects. It contains five main parts: TieFlow (workflow engine), Business Rules (rules to alter process flow), Common Repository (storage for project artifacts, versions, history, schedules, etc.), SOA (interface to allow internal, GFE, or COTS tools integration), and the Web Portal Interface (collaborative web environment).

SDA efficiently processes tasks in parallel across multiple threads. The MTTM efficiently processes tasks in parallel across multiple threads. The MTTM architecture wraps access to data with thread management to move threads to the home processor for that data so that the computation follows the data in an attempt to avoid L2 cache misses. Cache homeing is also handled by a memory manager that translates identifiers to processor IDs where the data will be homed (according to rules defined by the user). The user can also specify the number of threads and processors separately, which is important for tuning performance for different patterns of computation and memory access.

SDA efficiently processes tasks in parallel across multiple threads. The MTTM efficiently processes tasks in parallel across multiple threads. The MTTM architecture wraps access to data with thread management to move threads to the home processor for that data so that the computation follows the data in an attempt to avoid L2 cache misses. Cache homeing is also handled by a memory manager that translates identifiers to processor IDs where the data will be homed (according to rules defined by the user). The user can also specify the number of threads and processors separately, which is important for tuning performance for different patterns of computation and memory access.

SDA efficiently processes tasks in parallel across multiple threads. The MTTM efficiently processes tasks in parallel across multiple threads. The MTTM architecture wraps access to data with thread management to move threads to the home processor for that data so that the computation follows the data in an attempt to avoid L2 cache misses. Cache homeing is also handled by a memory manager that translates identifiers to processor IDs where the data will be homed (according to rules defined by the user). The user can also specify the number of threads and processors separately, which is important for tuning performance for different patterns of computation and memory access.

SDA efficiently processes tasks in parallel across multiple threads. The MTTM efficiently processes tasks in parallel across multiple threads. The MTTM architecture wraps access to data with thread management to move threads to the home processor for that data so that the computation follows the data in an attempt to avoid L2 cache misses. Cache homeing is also handled by a memory manager that translates identifiers to processor IDs where the data will be homed (according to rules defined by the user). The user can also specify the number of threads and processors separately, which is important for tuning performance for different patterns of computation and memory access.

SDA efficiently processes tasks in parallel across multiple threads. The MTTM efficiently processes tasks in parallel across multiple threads. The MTTM architecture wraps access to data with thread management to move threads to the home processor for that data so that the computation follows the data in an attempt to avoid L2 cache misses. Cache homeing is also handled by a memory manager that translates identifiers to processor IDs where the data will be homed (according to rules defined by the user). The user can also specify the number of threads and processors separately, which is important for tuning performance for different patterns of computation and memory access.

SDA efficiently processes tasks in parallel across multiple threads. The MTTM efficiently processes tasks in parallel across multiple threads. The MTTM architecture wraps access to data with thread management to move threads to the home processor for that data so that the computation follows the data in an attempt to avoid L2 cache misses. Cache homeing is also handled by a memory manager that translates identifiers to processor IDs where the data will be homed (according to rules defined by the user). The user can also specify the number of threads and processors separately, which is important for tuning performance for different patterns of computation and memory access.

SDA efficiently processes tasks in parallel across multiple threads. The MTTM efficiently processes tasks in parallel across multiple threads. The MTTM architecture wraps access to data with thread management to move threads to the home processor for that data so that the computation follows the data in an attempt to avoid L2 cache misses. Cache homeing is also handled by a memory manager that translates identifiers to processor IDs where the data will be homed (according to rules defined by the user). The user can also specify the number of threads and processors separately, which is important for tuning performance for different patterns of computation and memory access.

SDA efficiently processes tasks in parallel across multiple threads. The MTTM efficiently processes tasks in parallel across multiple threads. The MTTM architecture wraps access to data with thread management to move threads to the home processor for that data so that the computation follows the data in an attempt to avoid L2 cache misses. Cache homeing is also handled by a memory manager that translates identifiers to processor IDs where the data will be homed (according to rules defined by the user). The user can also specify the number of threads and processors separately, which is important for tuning performance for different patterns of computation and memory access.

SDA efficiently processes tasks in parallel across multiple threads. The MTTM efficiently processes tasks in parallel across multiple threads. The MTTM architecture wraps access to data with thread management to move threads to the home processor for that data so that the computation follows the data in an attempt to avoid L2 cache misses. Cache homeing is also handled by a memory manager that translates identifiers to processor IDs where the data will be homed (according to rules defined by the user). The user can also specify the number of threads and processors separately, which is important for tuning performance for different patterns of computation and memory access.

SDA efficiently processes tasks in parallel across multiple threads. The MTTM efficiently processes tasks in parallel across multiple threads. The MTTM architecture wraps access to data with thread management to move threads to the home processor for that data so that the computation follows the data in an attempt to avoid L2 cache misses. Cache homeing is also handled by a memory manager that translates identifiers to processor IDs where the data will be homed (according to rules defined by the user). The user can also specify the number of threads and processors separately, which is important for tuning performance for different patterns of computation and memory access.