Paper ID: 34454
oral

MICROGRAVITY SCIENCES AND PROCESSES SYMPOSIUM (A2)
Facilities and Operations of Microgravity Experiments (5)

Author: Mr. Benjamin Weps
DLR (German Aerospace Center), Germany

Mr. Daniel Lüdtke
German Aerospace Center (DLR), Germany
Dr. Olaf Maibaum
German Aerospace Center (DLR), Simulation and Software Technology, Germany
Dr. Andreas Gerndt
German Aerospace Center (DLR), Simulation and Software Technology, Germany

MODEL-BASED SOFTWARE ARCHITECTURE FOR A COLD GAS EXPERIMENT ON A SOUNDING ROCKET

Abstract

The MAIUS-1 mission aims to bring atom-optical experiments on a sounding rocket that performs a parabolic trajectory with an apogee of 250 km above the surface. This mission allows performing experiments with ultra-cold gases for around 6 minutes in a microgravity environment. Besides the scientific goals in the area of quantum-optics, other important objectives of the mission are miniaturization and further development of laser systems, vacuum components, quantum sensors, and other related technologies. This paper presents the software architecture of the main payload computer for the MAIUS mission which controls the experiments autonomously during flight and provides a telemetry and telecommand interface to monitor experiments and influence experiment sequences if necessary. The on-board computer runs the control software which makes use of DLR's Tasking Framework to concurrently execute the different software modules. Besides the modules for data collection, filtering and evaluation, a sequence player is developed to control the experiment apparatus. The sequence player controls at which point in time which parameterized experiment will be conducted. The sequences describe all necessary commands to the experimental hardware (shutter, lasers currents, camera triggers, etc.). The autonomous control of the experiment requires a non-linear execution of sequences. There are branches needed to optimize several parameters, for example laser currents. This requirement is implemented with an experiment execution graph. To enable an intuitive way to edit the experiment flow a graphical domain specific language (DSL) has been developed. Also to reduce sources of errors in the driver development, we designed textual DSLs in combination with code generators. This increased the level of automation in the software development significantly. Four different DSLs were designed in a layered architecture: Card DSL, Stack DSL, Subsequence DSL, and Sequence DSL. Each of these DSLs has its own syntax and semantics, which uses references to the layer below. Code generators transform the abstract hardware descriptions and experiment sequences to C++ code. In addition to syntactical validation of the DSL models, semantic and logical checks of the cards, stacks, sequences and the experiment flow graph are performed to ensure consistency and to prevent errors. Our approach has the benefit of an increased performance at runtime due to generated and compiled code, compared to interpreted driver descriptions and experiment sequences. More importantly, runtime errors can be prevented since the descriptions are validated during generation and the resulting code only depends on code generation templates.