Data Servers in Parallel Processes and Caper Language Paradigms

Sergey Vartanov
The Institute for Informatics and
Automation Problems of NAS RA
Yerevan, RA
e-mail: s.vartanov@iiap.sci.am

***Abstract*—The paper is devoted to the issues of constructing data servers by means of the Caper language for individual applications. The problems arising in this case and schemes for their solutions by creating data attributes and forming reentrant procedures are considered. The results of studies on implanting the mechanism for assigning attributes at the level of the compiler and virtual machine are presented.**

***Keywords*—Parallel computing, application data servers, data engineering.**

I. Introduction

In [1], a virtual machine (VM) of the Caper language was described with the ability to distribute it across processor cores and OS threads, allowing over a million parallel processes to be executed simultaneously in one application (estimated at up to 1.5 million with two-gigabyte RAM addressing on one computer with 8 32-bit cores (16 threads)). The article is devoted to the language paradigms for creating internal application data servers that provide simultaneous requests from different processes for placing, using, and deleting data. The problems of implementing controlled access and data transformations by means of object-oriented programming methods and ways of organizing controlling actions at the stage of program compilation.

II. Problems of Managing Common Data

Due to the ability of applications created on Caper to work with different data structures simultaneously, the task of organizing internal data servers with controlled access to them has arisen. Traditional solutions in modern OSes, as a rule, rely on the creation of monitors/mutexes in applications in other languages, registering them in the OS environment, with the help of which the problems of monopoly access to data are solved. Addressing a mutex that manages a shared resource causes suspension of processes, including those from other applications (in cases when the mutex belongs to the OS), that address already executed mutexes, which, of course, delays their execution due to delayed calls. A certain solution for control and access to data has been present in the Caper language since the first versions [2] and is supported by the mechanism of so-called variables with status, called “place variables”, which are formed by a pair (status, regular variable). Due to certain inconveniences and limitations of variable locations, difficulties in compilation and control by the virtual machine, the topic of sharing data from multiple parallel procedures, and even those located in different modules, had to be addressed again, especially in the context of significantly expanded language capabilities. A special place here belongs to composite data, which require control over the descriptors of its entire composition as well as its separate elements. These include lists containing heterogeneous data whose elements can be created and added to the list by different procedures from different parallel processes.

III. Solution Schemes

The basic idea to resolve common data usage issues is to create proprietary and application internal data servers that are out of the control of the OS. The Caper language, whose VM has its own mechanisms of generating parallel processes “lighter than light” and also the property of modularity with possibilities of dynamic loading and unloading of modules in the process of calculations, seems to be a particularly convenient tool for solving the mentioned problems. Moreover, Caper modules can only be data carriers, acting as their own structured pages of application memory. That is, creating one or more modules that service multiple parallel processes within a single application is a fairly natural structure that does not affect the OS mechanisms, and, therefore, excludes any indirect impact on other applications. Caper's modularity allows you to create multiple copies of the same data server to service selected groups of processes. In Caper servers, the problem of servicing multiple requests is removed by the actual reentrancy of the servicing procedures, since they are implemented by receiving parameters and asynchronously launching a parallel process for implementing actions.

```
block CreateData ( parm1, . . . ) ;* Procedure with parameters created with each call
do asynch CreateDataProc( parm1, . . . ) ;* Call of parallel process of creating data
endblock ;* Logical bracket of procedure description termination
```

That is, in this case, each call will correspond to its own process of creating data. Such a procedure is trivially reentrant. A similar thing (using “wrapper” procedures as in

the examples) is organized for other data actions, in particular, for calling processing procedures:

```
// Search procedure with parameters
block SearchData ( parm1, . . . )
do asynch SearchDataProc( parm1, . . . ) ;* Call of parallel search process with the next call
endblock    ;* Logical bracket of procedure description termination etc.
```

The bottleneck here is the direct transformation of the data, which requires special attention. Such problematic ones include list modification: insertion of an element, deletion and addition of an element, since structural changes imply the formation of pointers of links between elements. Simultaneity in this case is simply excluded and solved by monopolizing actions. Caper has the appropriate tools in the form of critical sections and special-type procedures. Let us emphasize: reentrancy of calls in the Caper language is solved by calling parallel processes, which is much simpler, "lighter" than lightweight processes in the OS.

IV. Data Attributes

Among the most integrated in terms of access controllability and transformations are list structures of heterogeneous data, whose elements are carriers of values or references to values, and generated by different concurrent procedures (the most capacious type of data in terms of variety is marked). The solutions for them extend to other data types in truncated forms as well. Thus, to control the data, a mechanism of attributing the following attributes to each element of the list, as well as to the whole list (partly, it is a variant of file attribute assignments accepted in OS file systems) has been investigated and proposed:

- data owner/creator indication;
- indication of status: access to other procedures and parallel processes is allowed or not;
- indication of permissible actions for other procedures and parallel processes: read only, write only, read/write.

The controlling link of the list (structure - list descriptor) is supplemented with attributes that indicate the possibilities of applying certain procedures, such as searching for values, adding, deleting, or inserting elements into the list, sorting, merging, creating list maps (map), and others. All of the above are represented by describing structures of both the list and its elements, whose members are carriers of attribute values, together with members that are pointers to the next element (and to the previous one in the case of bidirectionality), of the element's value. Such an organization is currently implemented in a separate module - a server for creating, managing access to lists and their elements, various types of search procedures and transformations. Here, a similar, but greatly simplified solution suggests itself for creating a server for managing arrays with fixed and floating boundaries. This was also implemented in a separate module. Let us note another property that allows for full control: the language virtual machine can provide any procedure with information about who called it (from which procedure and which process), in which module the calling procedure is located. This allows for access to data and procedures to be differentiated based on these characteristics: each procedure call can begin with a request to the VM "Who called it and from which module?"

V. Implementation of Variable Attribute Assignment Systems

During the research, the question was raised about the possibilities of providing variables of all types and kinds (or some of them) with control attributes. Two approaches were considered: the traditional one, based on long-accepted and tested methods of object-oriented languages, and the second one, associated with an attempt to provide the operators of data construction [3] of the Caper language with the assignment of responsibilities for the formation of the above attributes and the implementation of control procedures to the compiler and virtual machine. The reasons are clear: at the compiler level, there is a possibility to control the correctness of variable (data) usage, at the virtual machine level, there is a possibility to create a uniform support of data control and transformation by VM means, or, at least, the introduction of restrictions on the procedural maintenance of data. The analysis has shown that such means do not bring any special advantages, and in some cases narrow the possibilities of programming.

References

[1] С. Р. Вартанов, "Распределение вычислений для многоядерных/многопроцессорных вычислителей на основе системы виртуальных машин языка параллельного программирования Caper", *Параллельные вычислительные технологии – XV международная конференция*, ПаВТ'2021, г. Волгоград, с. 176–185, 2021.

[2] С. Р. Вартанов, "Язык программирования CAPER", Препр. 97-5, Национальная Академия Наук Украины, Институт Кибернетики им. Глушкова, Киев, 29 с., 1997.

[3] С. Р. Вартанов, "Методы конструирования и сопровождения данных средствами языка параллельного программирования Caper", *Труды 4-ой международной конференции "Параллельные вычисления и задачи управления"*, PACO-2008, Москва, с. 1268-1288, 2008.