S E S 2 0 1 5
Eleventh Scientific Conference with International Participation
SPACE, ECOLOGY, SAFETY
4 - 6 November 2015, Sofia, Bulgaria

DEVELOPMENT CLASSES OF OBJECTS' DESCRIPTORS FOR SPACE MISSIONS SIMULATION

Atanas Atanassov

Space Research and Technology Institute – Bulgarian Academy of Sciences
e-mail: At_M_Atanassov@yahoo.com

Keywords: *parallel calculations, pool of threads model; multi-physic models simulations.*

Abstract: *Object-oriented programming is powerfull modern approach for development of flexible programming tools. Some classes of objects applied in program system for space mission and simulation of experiments are presented. The aim of development of such classes of objects is approaching flexibility related to calculation's organization. Every class represents pattern for creation of objects' descriptors. Code fragments and application of developed classes of objects are shown. Classes of objects for description of ordinary differential equation systems integrators, situation problems solvers, initial values problems union of parallel tools and other are developed on the present stage.*

РАЗРАБОТКА НА КЛАСОВЕ ОТ ДЕСКРИПТОРИ НА ОБЕКТИ ЗА СИМУЛИРАНЕ НА КОСМИЧЕСКИ МИСИИ

Атанас Атанасов

Институт за космически изследвания и технологии – Българска академия на науките
e-mail: At_M_Atanassov@yahoo.com

Резюме: *Обектното програмиране е мощен съвременен подход за разработка на гъвкави програмни средства. Разгледани са няколко класа обекти използвани в рамките на рограмна система за симулации на космически мисии и експерименти. Целта на разработката на тези обекти е да се постигне гъвкавост по отношение на организацията на изчисленията. В случая всеки клас представлява шаблон за създаване на дескриптори на обекти . Показани са кодови фрагменти и реализации за използване на разгледаните обекти. На този етап са разработени класове за описание на интегратори на системи от диференциални уравнения, процесори за решаване на ситуационни задачи, обединения на паралелни инструменти и други.*

Introduction

Technological developments of computers provide more calculation powers for scientist-designer in field of space investigation. This allows development of more complex models and execution in details of simulations without necessaries from special deduced computer architectures.

One modern concept for complex and reusable software development is based on object oriented programing approach. Object programing offers possibilities for broader abstractions related to new user-defined data types and applied appropriate data processing methods. Every object has specific properties which distinguishe it from other objects. These properties could be described through complex user-defined type. A simulation model formed on the base of some types of objects is possible to be executed numerous times through different changeable scenarios.

The better using of growing calculation power could be achieved through increasing flexibility of developing software and development of possibilities for easily definition of new tasks in the frames of appropriated objects field.

Algorithms and program system for multi-satellite missions simulation is under development in STIL-BAS, branch in Stara Zagora. The recently improving of the system flexibility and possibilities for simulations based on complex physical-mathematical models are shown in the present report.

Basic tasks in the frame of the program system for space experiments' simulation

The basic tasks provided for solving was [1]:

- Numerical integration of satellites motion equations.

- Calculation of different geometrical and physical parameters of the environment along the orbits.
- Situation analysis - calculation of time intervals appropriated for satellite measurements according to specific constraints.
- Active satellite experiments and physical processes simulation at appropriate parts of orbits, according to previously executed situation analysis.
- Satellite operations scheduling.
- Visualization of results and simulated scenes.
- Writing of obtained results.

The organization of calculations comprising different tasks from listed above types was based on static scheme, connected with consecutively execution of these one.

Two parallel tools- ordinary differential equations systems integrator [2] and situation problems solver [3] was developed in the course of space missions' simulation system development. Motion equation systems of one or more classes of space objects (satellites, space debris, charged particles and neutral or charged dust particles) could be solved through starting numerous of actual integrators.

For instance, a set of situation problems could be solved with group of satellite and other set of situation problems with space debris. The both sets of situation problems could be solved simultaneously in parallel trough starting more than one actual situation problem solver. These actual integrators and solvers could be executed simultaneously through "union of pools of threads" program model [4]. The applying of this model demands from application of more flexible schemes for calculation scenarios definition and control of their execution.

State of the problem

The aim of the present work is to present some user-defined types, which could be used for flexible definition of calculation tasks and their execution. Classes of objects- descriptors heaving such user-defined types could be created. Definition of complex and various versions of simulations are achieved via these classes of objects.

Development of some user-defined types

a). Type "parallel solvers"

This user-defined type serves as object-descriptors for creation of parallel calculation tools based on "pool of threads" program model. The definition of this type is shown on figure 1a.

```
type     pool_par
 sequence
  UNION
    MAP
      integer   num_threads
      integer   ha_race
      integer   counter_adr
      integer   pool_par_adr
      integer   granulation
    END MAP
    MAP
      ! integer   union_atr(2)
    END MAP
  END UNION
end type  pool_par                    (a)
```

```
type     IVP_par
  character   name*20
  integer   integ_index ! serial order in the class
  integer   num_objects
  integer   t_adr,dt_adr
  integer   xvn_adr,xvk_adr,eps_adr
  integer   adr_Grv_model,len_Grv_model
  integer   transfer_data_adr,work_data_adr
end type  IVP_par
                                                    (b)
```

Fig. 1. (a) parallel solvers type definition; (b) Initial values problem type definition

The components of user-defined type **pool_par** are: **num_threads -** containing number of the threads, **ha_race** - handled of the event for synchronization between threads when tasks are got from input task queue, **counter_adr** - counter address for countering solved tasks, **pool_par_adr** - address of pool of threads parameters and **granulation** - control parameter pointing the rate of breaking up entire task into smaller tasks.

This **pool_par** type could be used for descriptors of different solvers – on this stage these are integrators of ordinary differential equation systems and processors for situation problems' solvers.

b). Type "initial values problems"

The type IVP_par contains various attributes describing an initial values problem (fig. 1b). Character type attribute "name" contains the name of IVP. The attribute **integ_index** contains serial order of pool of threads which represents ordinary differential equation systems integrator among all objects in the class. The next variable **num_objects** indicate the number of all objects which motions could be integrated. Attributes **t_adr** and **dt_adr** contain addresses of variables, where time and step of time are stored. Analogously the next lines contain addresses of coordinates and tolerances data about all objects, address of array containing information about perturbations for each object and length of element of the array. The last line contains addresses of working arrays which are necessary for integrator.

c). Type "situation problems"

The type **SitProblems** contains different attributes related to situation problems solving (fig. 2). The first two attributes contain order numbers of situation processor and initial value problem as members in respective classes. **num_objects** present the number of objects (satellites), **max_num_sit** and **num_sci_prob** determine the size of array, **sit_prob** contains situation problems having address in **addr_sit_prob**.

```
type      SitProblems
 integer  pool_index       ! index of the pool in a descriptor - class
 integer  IVPs_index
 integer  num_objects      ! number of objects in conected IVP
 integer  max_num_sit      ! maximal number of situation conditions for all situation problems
 integer  num_sci_prob     ! number of situation problems
 integer  addr_sit_prob    ! address of situation 2D array containing situation problem
                           ! definitions- each column contains situation problem
 integer  addr_xvn,addr_xvk
 integer  TrParam_adr     ! TrParam- contains calculated parameters along the orbit
end type  SitProblems
```

Fig. 2. User-defined type **SitProblems** contains attributes for description of situation problems

d). Type "union of pools of threads"

The user defined type **PoolThUnion** (fig. 3a) represents template for descriptor of pools of threads union. The first attribute **num_threads** contains sum of threads for all pools. The second attribute **union_atr** contains address of array which contains all control parameters for union of pools [4].

```
type      PoolThUnion
  integer  num_threads
  integer  union_atr(2)
end type  PoolThUnion
```

(a)

```
type      TrajParam
 integer   num_objects
  integer   trj_par_
end type  TrajParam
```

(b)

Fig. 3. (a) this user-defined type describe "pools of threads union" objects; (b) type for objects "*trajectory calculations*"

e). Type "trajectory calculations"

This type (fig. 3b) provides calculation of various quantities from geometric and physical nature along the orbit. These quantities are calculated on every step in the time, after objects motion integration. The obtained results could be used for situation analysis or simulations. The type **TrajParam** contains information about number of objects, address of calculations control structure and address of structure containing calculated quantities from models.

The access to classes' descriptors from random point of the program is ensured trough common named areas. Every area contains current size and address of the respective class (fig. 4a, 4b).

Creation of classes of objects

When one structural variable from given user defined type receives values of his components we can accept that object is created. The members of given class are objects with same types.

All of above described user defined types serve for objects- descriptors creation, each of them belonging to respective class. An essential parts of descriptor's attributes contain meta-data addresses of the real data and their dimensions. These meta-data are determined in the course of tasks definition which will be solved and preceded inserting of particular data. When the values of all attributes of one structured variable are defined, these variables are submitted to subroutine for object creation and adding to corresponding class.

```
INTERFACE
 SUBROUTINE    add_object(dimension, AIs_descriptor_adr,  AIs_descriptor_adr_new,   AI_param, &
                           IVPs_descriptor_adr, IVPs_descriptor_adr_new,  IVP_param, &
                          TrPas_descriptor_adr,TrPas_descriptor_adr_new,TrPar_param, &
                          StPrb_descriptor_adr,StPrb_descriptor_adr_new,StPrb_param, &
                         UsPTh_descriptor_adr,UsPTh_descriptor_adr_new,Union_param)
  integer                           dimension
  integer,            optional ::  AIs_descriptor_adr,  AIs_descriptor_adr_new, &
                                  IVPs_descriptor_adr, IVPs_descriptor_adr_new, &
                                 TrPas_descriptor_adr,TrPas_descriptor_adr_new, &
                                 StPrb_descriptor_adr,StPrb_descriptor_adr_new, &
                                UsPTh_descriptor_adr,UsPTh_descriptor_adr_new
   type        pool_par
    integer   num_threads,ha_race,counter_adr,thread_par_adr granulation
   end type  pool_par
   type       (pool_par),                optional :: AI_param

   type        IVP_par
    character    name*20
    integer       integ_index,num_objects,t_adt,dt_adr,
    integer       xvn_adr,xvk_adr,eps_adr,adr_Grv_model,len_Grv_model
    integer       transfer_data_adr,work_data_adr
   end type IVP_par
   type      (IVP_par),                  optional :: IVP_param

   type        TrajParam
    integer   num_objects      ! number of obects
    integer   trj_par_adr        ! address of trajectory parameters arry for one IVP
   end type  TrajParam
   type       (TrajParam),               optional :: TrPar_param

   type        SitProblems
    integer   sit_solv_index,IVPs_index
    integer   num_objects,max_num_sit,num_sci_task,addr_sit_prob
    integer   addr_xvn,addr_xvk,TrParam_adr
   end type  SitProblems
   type       (SitProblems),             optional :: StPrb_param

   type        PoolThUnion
    integer  num_threads
    integer  union_atr(2)
   end type  PoolThUnion
   type       (PoolThUnion),             optional :: Union_param
 END SUBROUTINE  add_object
END INTERFACE
                                                                                              (a)
```

```
common   /c_IVPs/num_IVPs,IVPs_descriptor_adr1
        …
      AI_1%num_threads    = num_threads;                    AI_1%thread_par_adr= thread_par_adr;
      AI_1%    ha_race       = ha_1;                             AI_1%counter_adr      = LOC(AI_1_glb_counter)
                                                                   AI_1%granulation       = 1
CALL  add_object(num_AIs,AIs_descriptor_adr,AIs_descriptor_adr,AI_1)
                                                                                              (b)
```

```
common   /c_StPrs /num_StPrs,StPrs_descriptor_adr1
        …
CALL  add_object(num_StPrs,StPrb_descriptor_adr=StPrs_descriptor_adr, &
                              StPrb_descriptor_adr_new=StPrs_descriptor_adr,StPrb_param=StPrb_param);
                                                                                              (c)
```

Fig. 4. a). Interface of subroutine; b) and c). Variants for calling the subroutine according to object type. Illustration of polymorphism is shown.

The subroutine **add_object** (fig. 4b, c) accepts objects and inserts them in respective class. One new class of descriptors is created during the first call of the subroutine add_object with actual parameters - object of given type. The subroutine has polymorphic abilities and accepts all **different** objects according to their user-defined types. This is approached trough appropriate interface and description of actual parameters shown on figure 4.

Different relations and possible connections between separate/particular classes are shown on figure 5. For example, object-descriptors from class of parallel tools (integrators, situation solvers) are in connection with object-descriptors of initial value problems. Descriptors of integrators and situation solvers are connected too.

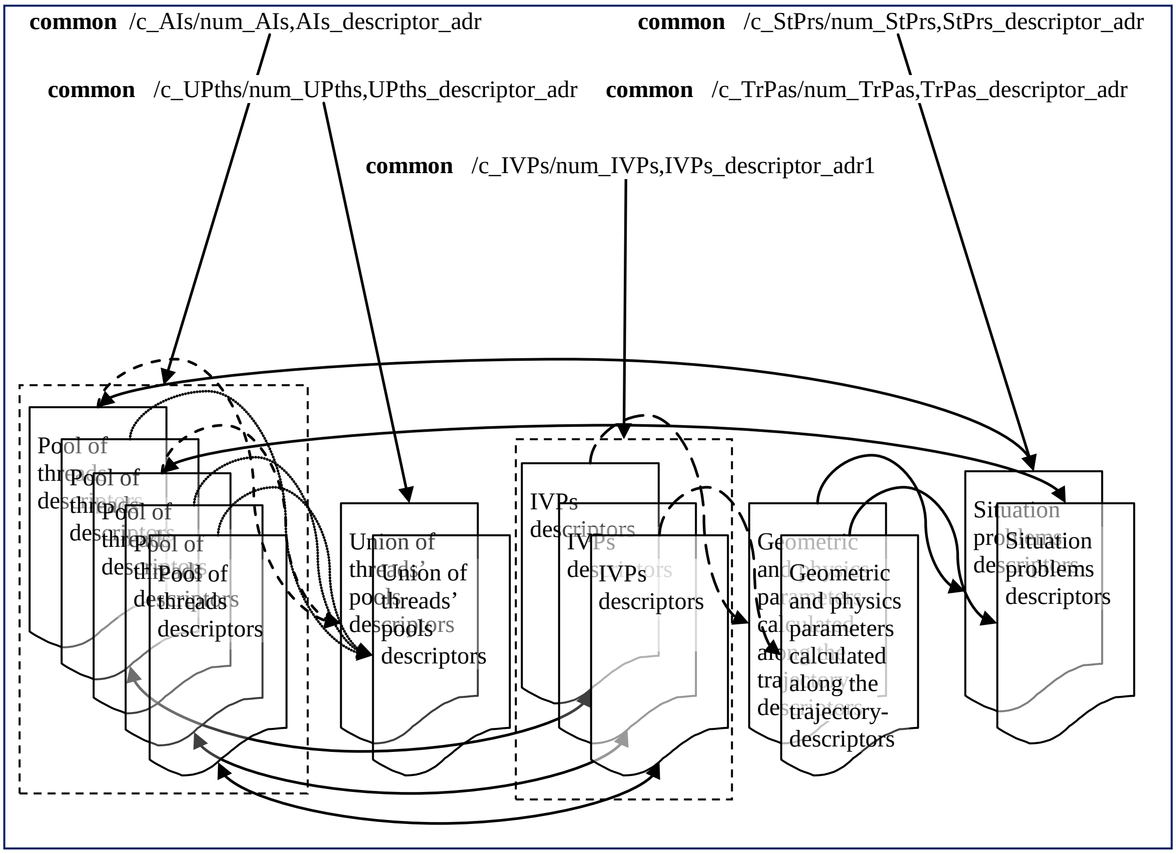

Fig. 5. Semantic model presenting relations between different classes

Conclusion and future work

Only five user-defined types are developed on this stage and some number of simulation problems could be defined. These types contain basic meta-data (address of the real data in the storage and dimensions) about described from them object. Two of explained types - **pool_par** and **PoolThUnion** describe abstract models for parallel calculations execution.

Developed types are used for development of new control of calculations and achieving a flexibility and freedom about definition and execution of the simulation tasks in the frame of Program System for Space Missions Simulation [5].

Reflection of relations and description of properties in given object field is the aim of the development of above explained and other user-defined types in future.

Explained approach for development of object-classes is different from these one which are used in object-oriented programing via fortran 95/2003.

References:

1. Atanassov, A., Program System for Space Missions Simulation – First Stages of Projecting and Realization, In Proceedings of SES 2012, 2013, pp. 209-214.
2. Atanassov, A.M., Parallel, adaptive, multi-object trajectory integrator for space simulation applications. Advances in Space Research 54, 2014, pp. 1581–1589.
3. Atanassov, A.M., Parallel Solving of Situational Problems for Space Mission Analysis and Design, proceedings of 9th scientific conference Space Ecology Safety, 2013, 2014, pp. 283–288.
4. Atanassov, A.M., Method of Thread Management in a Multi-Pool of Threads Environments, proceedings of 9th scientific conference Space Ecology Safety, 2014, 2015, pp. 241-246.
5. Atanassov, A.M., Approach and Development of Tools for Different Variants of Space Missions Simulation Definition and Execution, proceedings of 9th scientific conference Space Ecology Safety, 2015, 2016, pp.