STAR

detector simulations in OO era

- *GSTAR framework design*
  - **Basic requirements**
  - **Software design**
  - **Specification language**
  - **Geometery Data Base**
  - **Multy-standard Input/Output**
  - **ROOT interface**
- *Implementation details*
  - **Program functionality**
  - **Simulated geomeries**
  - **Accuracy vs production time**
  - **Event size vs event history details**
  - **Supported platforms**

GSTAR basic requirments

- *GSTAR Design requirments*
  - **Long term maitenance and development**
  - **Open system (to work with ROOT, GEANT4 etc)**
  - **Code modularity (ie test setup etc)**
  - **Code readability and easy to learn**
  - **Code robustness (to human errors)**
  - **Flexible, detailed geometry (over many years)**
  - **Physics tuning independent of geometry**
- *Data structuring*
  - **Multi-standard event I/O (FZ,XDR,ROOT,Objectivity)**
  - **multiple version support**
  - **automatic documention**
  - **Geometry Data Base interface**
  - **basis for other analysis code**
- *Control tools (interactive)*
  - **C++ interface classes**
  - **access to detector description**
  - **access to generated data (express-analyse)**
  - **access to documentation**
  - **material scan and other Geant tools**

GSTAR software design

- *ierarchical system - no user code addresses low level directly*

basic level	**platform dependent code** **system libraries**
low level **(cernlib)**	**Zebra, Kuip** **Geant, Paw** **Comis, Dzdoc**
system level	**Rebank as Zebra interface** geant **parser** **AGI-library** **ROOT**
user level	**modules in** geant **language** **(fortran++), C (C++)**

- *user code communicates only with the system level, which :*
  - **verifies parameter consistence**
  - **adds dependent and inherited parameters**
  - **resolves numbering conflicts**
  - **accepts polymorphism of Geant objects**
  - **provides and controls structural data access**
- *user code may be written in* geant *specification language*
  - **syntaxis independent of low level implementation**
  - **control at preprocessing and compilation time**
  - **still a code (Fortran or C -like)**
- *user code is loaded dynamically and can be modified at run time*

Geometry specification language

The **geant** language is a OO Fortran extension oriented to the GEANT application.

Apart from Fortran statements, it contains a number of **geant statements** in the form:

$$\text{OPERATOR NAME [} keyword_1{=}value \text{ ... } keyword_n{=}value \text{]}$$

- OPERATOR defines a service to be performed by the *Advance Geant Interface*
- there are
  - 3 code structuring operators,
  - 9 GEANT dedicated operators,
  - 3 data handling operators
- NAME is a name of a GEANT volume, or a volume shape, or a material or medium name
- Keywords are variables, **used in the GEANT3 manual**, to describe parameters of the GEANT routines
- Their values (right parts of assignment) are any legal Fortran expression.

Volume description

- A detector subsystem is described as a separate **module**.
- Any GEANT volume in a module is a GSTAR object and is described as a **block**.
- BLOCK, ENDBLOCK and CREATE are structuring operators because they affect the code execution order.
- CREATE is executed as a jump to the requested BLOCK code and the return back when its ENDBLOCK is reached.
- A block consists of two parts - the description of its own *properties* and the description of its *content*
- All GEANT dedicated operators are memeber functions of a AGI class and they execute one or few GEANT routines.

	BLOCK NAME comment	
	MATERIAL name definition	⇒GSMATE
or few	COMPONENT name definition	⇓
followed by	MIXTURE name definition	⇒GSMIXT
	MEDIUM name definition	⇒GSTMED
	ATTRIBUTE name definition	⇒GSATT
	SHAPE name definition	⇒GSVOLU
		⇒GSDV(*)
a number of	**CREATE** name	
followed by	POSITION name definition	⇒GSPOS(P)
	HITS name definition	⇒GSDET(*)
	DIGI name definition	⇒GSDET(*)
	ENDBLOCK	

Data Structuring

A group of logically linked variables, declared in a STRUCTURE operator, gets their values defined using the FILL statement:

FILL NAME		! bank comment
	$variable_1 = value_1$	! explanation of $variable_1$
	. . .	
	$variable_n = value_n$	! explanation of $variable_n$

- Structure name is used as the bank name and as the prefix of its variables in the Fortran code.
- When the FILL statement is executed, not only the data are saved as a bank, but the variable names and comments are saved as well.
- $value_i$ are Fortran expressions in case of a simple *variable* or a vector in the form $\{val_1, ...val_k\}$ for an array.
- The order of assignments is irrelevant, but comments and explanations are mandatory.

Data Base Interface

- **loading geometry data creates a relational data base in the GSTAR memory**

⇒example

- **selection of the actual copy of each structure to be used by the program (instantiation), is done by the USE statement:**

USE NAME *variable = value*

- **any variable from the structure can be used as the selection key**
- **USE processor in AGI interface not only finds the the requested instance of a structure, but also checks its consistence with the code (number of variable and their types).**
- **once a top level bank is selected with USE operator, other branches are dropped and the lower level banks are selected within the same branch only.**

Geometry Data Base

- a set of close versions of the same subsystem is described by one geometry module.
- a single data file per subsystem contains all versions of parameters, selectable by version number, date or other key(s).
- major modifications of subsystems are described by different codes (different files).
- version selection is done at the GSTAR run time with interactive commands, after geometry is dynamically loaded (not necesserely to run simulations).
- selected versions are written into the output together with their specifications.
- gstar prodices IDL description of the parameter sets to use with STIC compiler
- data consistence is checked at the run time by input module.

event INPUT/OUTPUT

- **Input facilties**
  - **data set library (XDR) format**
  - **Any ZEBRA format (user controlled)**
  - **text files (user control possible)**
  - **previously generated data**
- **Output facilities**
  - **GEANT standard (ZEBRA)**
  - **data set library (XDR) format**
  - **ROOT Tobjects**

Supported platforms

- **GSTAR is fully supported on the following UNIX platforms :**
  - **IRIX (Silicon Graphics)**
  - **SunOS (Sun Microsystem)**
  - **Sun Solaris (Sun Microsystem)**
  - **ALPHA OSF (DEC)** (∗)
  - **HPUX 9 (Hewlett-Packard)**
  - **HPUX 10 (Hewlett-Packard)**
  - **AIX RS/6000 (IBM)**
  - **PentiumPro under Sun Solaris**
  - **Linux 4, Linux RedHat 5**
  - **CRAY T3E (multi-processor machine)**
- **Work is progress for Windows NT version**
- **Multiple platform support is extremly useful for debbuging and robust running !**

(*) 64 bit machine

Implemented STAR subsustems

- **Beam Pipe**
- **Silicon Vertex Tracker**
- **Time-Projection Chamber**
- **Multi-Wire Chamber (inside TPC)**
- **Forward TPC**
- **Central Trigger Barrel**
- **e.m. Calorimeter Barrel**
- **e.m. End-cap Calorimeter**
- **Vertex Position Detector**
- **Magnet and magnetic field**
- **Photon Multiplicity detector (*)**
- **Zero degree calorimeter (*)**
- **Time of flight system (*)**

(*) not in the baseline design

Program performance

- **Full rapidity coverage is used for simulations**
- **Geant cuts selected at 1 MeV for most of the detectors**
  - **lower cuts are used in e.m. calorimeters to better reproduce shower width**
  - **lower cuts are recomended for dedicated TPC tracking accuracy study (close δ - electrons affect cluster size)**
- **Photo-absorption ionization model is used** *locally* **for TPC gas and some thin sensitive layers (Si, Shower Max detector)**
  - **very good agreement with experimental data (R.Bossingham, STAR note 249)**
- **Typical simulation time for a complete $Au - Au$ event (above 10K primary particles) on Pentium Pro is about 30 minutes**

It is often desirable to determine the origin of hits from secondary tracks, produced during the simulations.

In practice, it is hard to save all secondary tracks because of the event record size.

GSTAR may append to the output bank only those secondary tracks, which have produced a hit in a specified detector.

The vertex entry of secondary tracks keep their closest ancestor as well as the originating process and the volume where it happend. This information is retrieved "post factum" from GEANT.

In this way the total size of an output event is always determined by the amount of hits (about 20 Mb for a Au-Au event), but most of the history information is avalable to the analysis modules.

Future plans

GSTAR is already a robust production tool.

Future development is foreseen in the following directions :

- **GSTAR/ROOT full integration**
- **GSTAR/GEANT4 interface**
- **fast parametrization options**
- **further geometry detalization**