Analysis, design, and implementation of PHENIX on-line computing systems software using Shlaer-Mellor Object-Oriented Analyis and Recursive Design

Thomas Kozlowski [a], Ed Desmond [b], John Haggerty [b], Hyon-Joo Kehayias [b], Martin Purschke [b], Chris Witzig [b]

[a] *Physics Division, Los Alamos National Laboratory, Los Alamos, NM 87545*

[b] *Physics Department, Brookhaven National Laboratory, Upton, NY, 11973*

An early prototype of the core software for on-line computing systems for the PHENIX detector at RHIC has been developed using the Shlaer-Mellor OOA/RD method, including the automatic generation of C++ source code using a commerical translation engine and "architecture".

Key words: OOA; OOD; Object-Oriented; Shlaer-Mellor; C++; PHENIX.

1 Shaler-Mellor OOA/RD

The Shlaer-Mellor method is centered on "Information Models" that precisely define the entities, rules and policies of a domain. Domains are the highest level components of a Shlaer-Mellor system, identified according to their specific "subject matter", and can be developed relatively independently. The life-cycle of each "active" object in an Information Model is modeled in a State Transition Diagram. Actions are associated with each state and are specified by Data Flow Diagrams or a high level Action Specification Language. Transitions between states result from "events" (messages) sent from objects or external domains. The resulting set of models is a description of the system as a set of interacting Finite State Machines embedded in object instances. Since the models are rigorous and complete, the system can be simulated by executing them, or they can be transformed directly into source code by a translation engine according to a specific "architecture" (translation rules, code templates and supporting software libraries). Performance and resource constraints are addressed in the architecture, not the OOA models. Different architectures can be developed for different implementation requirements. While the OOA models precisely define each domain in an implementation-free way, a Shlaer-Mellor architecture is used to translate the models to an implementation in an application-free way. In this sense an architecture is very analoguous to a high level language compiler and associated run-time libraries (thus the alternative term, "model compiler"). Recently it has been possible to purchase architectures "off-the-shelf", offering a fast implementation path, especially for prototype projects. More details about the method can be found in books by Shlaer and Mellor[1,2] and from the company founded by the method's authors, Project Technology Inc. [1] Our reasons for selecting this particular method were reported at a past conference[3].

2 Application to PHENIX On-line Computing

The On-line Computing Systems (ONCS) group has completed the first prototype of the on-line computing software. This prototype is a skeleton system that implements only essential data acquisition capabilities (primarily run control and configuration of down-loadable frontend hardware). A primary goal of this prototype was to gain experience with the method through a complete development cycle. In support of this effort, we purchased from Project Technology the BridgePoint Model Builder tool for creating the OOA models. We also contracted for several weeks of a Shlaer-Mellor OOA consultant. Members of the group have also received various levels (from none to three weeks) of formal Shlaer-Mellor training courses.

ONCS Shlaer-Mellor Domains. The domains for the prototype system are diagrammed in Figure 1. The arrows do not show data or control flow, but "dependencies". For example, the PHENIX On-line Operations domain assumes there is a User Interface domain that provides input from and output to users and user processes. PHENIX On-line Operations is the main or "application domain"; it represents the system from the point of view of the user. Various service domains provide services to other domains: User Interface (including access control), Ancillary Systems Management (access to high voltage, etc.), Event Analysis (processes built events), Custom Hardware Management (access to data acquisition hardware), Event Data Logging, and Archive and Log Management (non-event data and logs). Architecture is the Shlaer Mellor architecture domain. The remaining domains are implementation domains which are used by the architecture or independently to implement the other domains. The domains in the dashed box are domains which may be implemented in whole or in part using Shlaer-Mellor. The others are imported software or are implemented using "traditional methods". In

[1] http://www.projtech.com/

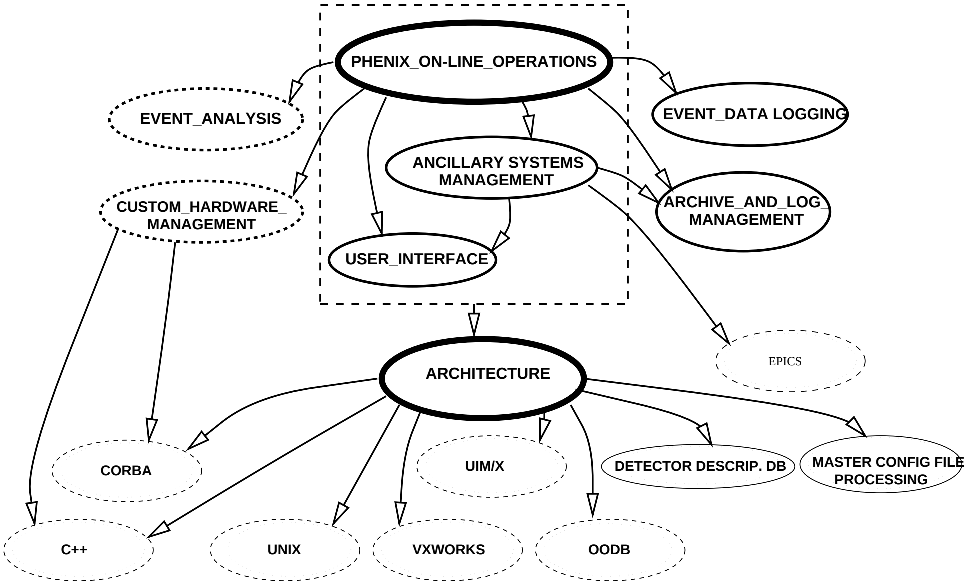

Fig. 1. ONCS Domain Chart.

the case of these latter domains, a layer of inter-domain "bridge" software is added to interface them to client domains. For the first prototypes, some of these domains are very simple. Their capabilities will be expanded as needed in later releases. The Shlaer-Mellor bridge concept provides a clean interface which can simplify upgrades.

Information Models. In the first prototype, only the PHENIX On-line Operations domain was developed using OOA. A large domain can be divided into more manageable sized "subsystems", which are collections of closely related objects, loosely connected to objects in other subsystems. The prototype subsystems are: the Detector Subsystem (PHENIX detector in terms of subdetectors, magnets, gas systems, and electronics racks); the Subdetector Subsystem (a subdetector in terms of major components such as a drift chambers and ancillary system facilities like high voltage channels); the Data Acquisition Subsystem (configurable data acquisition and trigger components, runs, detector "partitions", etc.); and the Bridge Objects Subsystem (handles interfacing to external domains - bridges). Figure 2 is a simplified version of the Information Model for the Subdetector Subsystem. The boxes represent objects and list their attributes. It shows a Subdetector (an object in the Detector Subsystem), its Subdetector Components and associated Ancillary System Facilities, and the relations between them. Attributes flagged by asterisks are "identifiers" used to uniquely identify a particular instance, and those qualified by "(Rnnnn)" expressions are referential attributes used to formalize relationships with the identifying attributes of related objects. The arrows show relations. A doubled-headed arrow points at an object which may have

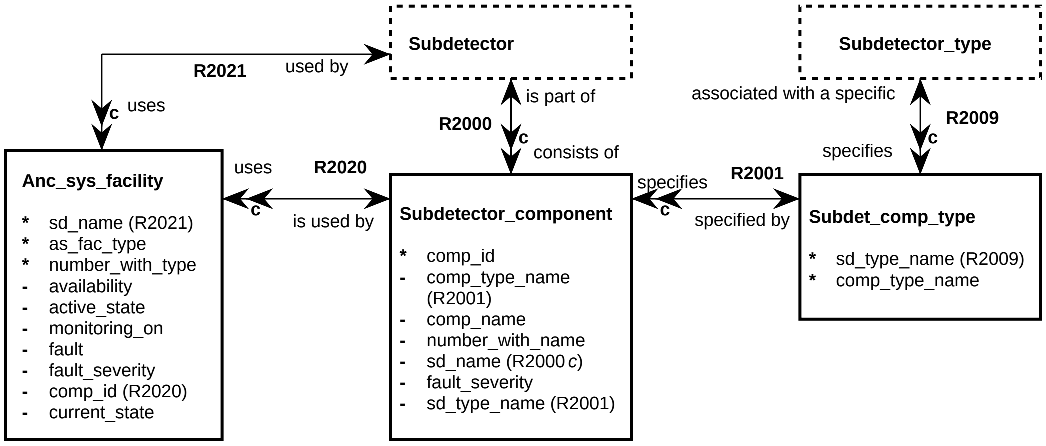

Fig. 2. Simplified Subdetector Subsystem Information Model.

"many" instances that participate in a relation. The small "c" by an arrow head, indicates a conditional relation, i.e., there may be no object instances that participate. The textual tags further specify the relation. They can be read in either direction: "a Subdetector consists of none or more Subdetector Components" or "a Subdetector Component is part of one and only one Subdetector".

State Models. The dynamics of an active object are specified by a State Transition Diagram (STD), using the Moore form in which actions are associated with the destination states (not with transitions). A transition may occur on receipt of an event by an object instance from the same or another object instance or from an external domain. Events are given a unique event name based on the destination object type and state. An event can carry "supplemental" data for use by the State Action. Figure 3 shows the STD for the Ancillary System Facility object. An Ancillary System Facility is a generic object representing supporting equipment or a sensor. This STD specifies the dynamics of an Ancillary System Facility, including the posting of significant state changes and fault conditions. The detailed equipment operations are not modeled here, since they are handled by the Ancillary System Systems Management domain which has knowledge of the capabilities and operational details of each specific type of equipement or sensor. The textual tags associated with transition arrows specify the events that cause the transition.

Process Models. The method uses an "Action Data Flow Diagram" (a specific form of a traditional Data Flow Diagram) to specify the processing that occurs on a transition to a state. Most CASE tool vendors do not use them, but use instead a high level "Action Specification Language" (ASL).

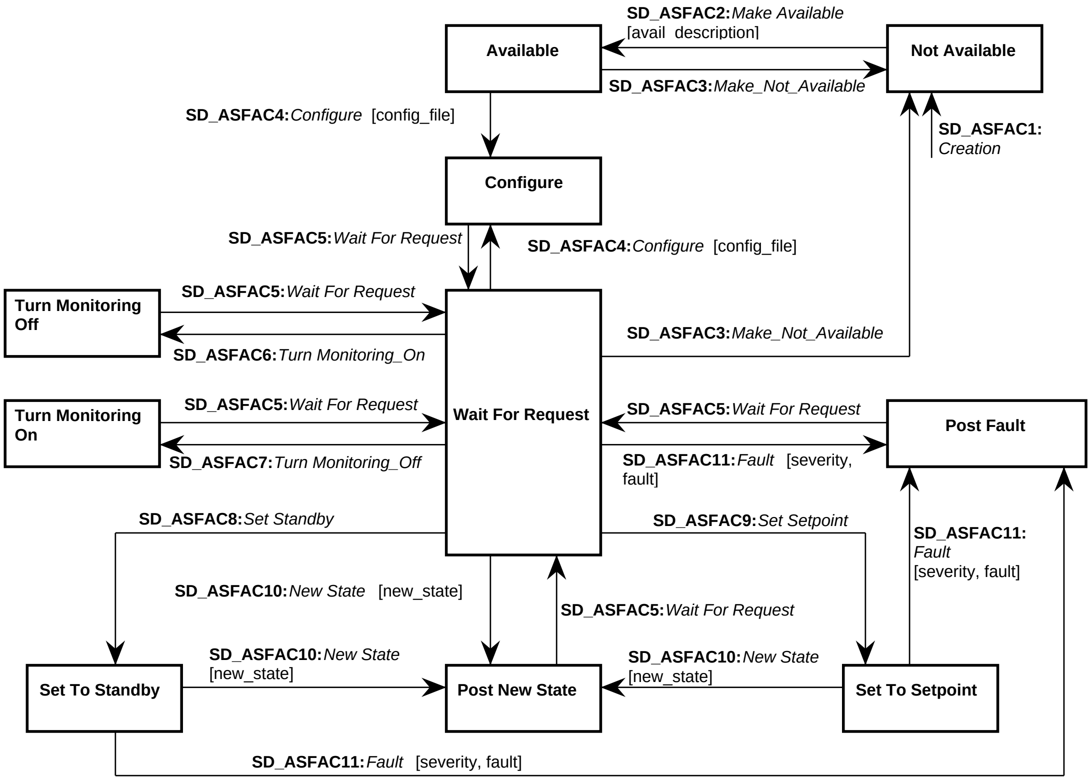

Fig. 3. Ancillary System Facility State Transition Diagram.

The following is the BridgePoint ASL that specifies the "Configure" state action (See Figure 3):

```
// ** update active state attribute
  assign self.active_state = "CONFIGURING";
// ** send event to Ancillary Systems Service
  generate AS3:Configure_Facility_Using_Config_File (
    config_file:rcvd_evt.config_file, sd:self.sd_name,
    as_fac_type:self.as_fac_type, number:self.number_with_type,
    to AS;
```

This very simple state action does two things: it sets the object's "active_state" attribute to the new "CONFIGURING" state, and sends an event to the Ancillary System Management domain (AS), with instance identifiers and configuration file name for use by the service. When the service completes it will send a "New_State" event back to this instance.

Architecture and Code Generation. The ONCS group purchased the BridgePoint code generator and an "off-the-shelf" architecture (MC-2010) from Project Technology. MC-2010 is a simple but complete architecture compatible with the BridgePoint toolset. While MC-2010 does not meet our final needs, it was adequate for the prototype. It is somewhat restricted: OOA analyzed domains all reside in an single task; non-OOA domains may reside in the OOA task or an external task; and all tasks are on the same node. It generates C++ code in a UNIX environment, including: C++ classes for every object; a member function for each state action; header and source files for events (they can be used by non-OOA domains for sending events an OOA-domain); and header files for events directed to non-OOA domains (the non-OOA domain supplies the implementation). Inter-task communication uses UNIX message queues. User control of the actual generation process is simple: configuration information is specified such as which domains are in which tasks, and then a top level make file is invoked. The following code was generated for the preceding "Configure":

```
void PHLOPSAS_Anc_sys_facility_c::asyncActionConfigure_state(
                         stda_string_c const &p_Config_file ) {
  stda_benchmarkActionStarted();
  this->setm_Active_state( "CONFIGURING" );
  PHLOPSAS_ANCSYS_AS_bridge_c::
      Configure_Facility_Using_Config_File(
        this->getm_As_fac_type(), p_Config_file,
        this->getm_Number_with_type(), this->getm_Sd_name());
}
```

Since even our prototype system is a distributed one, we added a Shlaer-Mellor-event-like intertask and internode communication mechanism based on CORBA.

3 Experiences and Future Plans

Experiences. Our experiences have been mostly positive, although there is concern about payback from the investment in tools, training, learning on the job, and the inevitable unease in developing software in a way that is quite different for most of us. In particular, the large analysis effort expended early on in a project in getting the models right (and understanding the system) tends to make those used to "traditional methods", in which emphasis is concentrated on developing code, uncomfortable. Formal training in the method is definitely important and results in a net saving of time. But is it is not enough, since developing analysis skills takes time and experience. The use of a consultant has been valuable in helping us make progress and giving us encouragement during the often difficult learning period. The learning curve problem practically guarantees that the initial projects will take significantly longer than with more traditional methods. The investment should result in better productivity in subsequent development and in better understood and documented systems which match requirements and are more maintainable (It is much easier to work at the level of the graphical models than C++ source code.). It is important that the entire team be involved in learning and understanding the method at appropriate levels, so that all efforts will fit in with the Shlaer-Mellor way of organizing and doing things. This has been somewhat of a problem for us because of more than normal staffing changes.

Future. After the completion of the early prototypes, we will evaluate our experiences and our options for subsequent ONCS releases, including final system architecture requirements and extending OOA analysis to other domains. As mentioned earlier, the purchased architecture is not adequate beyond the first prototypes. The size of the PHENIX system requires a distributed system because of the large number of processors and expected performance gains from parallelism. We will either have to modify the current architecture or provide our own. We have already had to port the architecture to new platforms, which was not a simple effort. Although it will not be easy to make major modifications, the current architecture probably is a good basis and sufficient framework for modifications (adding CORBA distributed objects, for example) for the next level of prototype.

Acknowledgement

This work has been funded by the U.S. Department of Energy.

References

[1] Sally Shlaer and Stephen J. Mellor, *Object Oriented Systems Analysis, Modeling the World in Data* Yourdon Press, 1988.

[2] Sally Shlaer and Stephen J. Mellor, *Object Oriented Systems Analysis, Modeling the World in States* Yourdon Press, 1992.

[3] T. Kozlowski et al., Shlaer-Mellor Object-Oriented Analyis and Recursive Design, an effective modern software development method for development of computing systems for a large physics detector, in: *Proceedings of the International Conference on Computing in High Energy Physics, Rio de Janeiro, Brazil, 1995* (World Scientific, 1996) 687-695.