A Rule-Based Shell to Hierarchically Organize HST Observations

Ashim Bose and AndrewGerb

bose@stsci.edu and gerb@stsci.edu
Space Telescope Science Institute
Baltimore, MD 21218

1. Abstract

An observing program on the Hubble Space Telescope (HST) is described in terms of exposures that are obtained by one or more of the instruments onboard the HST. These exposures are organized into a hierarchy of structures for purposes of efficient scheduling of observations. The process by which exposures get organized into the higher-level structures is called *merging*. This process relies on rules to determine which observations can be "merged" into the same higher level structure, and which cannot.

The TRANSformation expert system converts proposals for astronomical observations with HST into detailed observing plans. The conversion process includes the task of merging. Within TRANS, we have implemented a declarative shell to facilitate merging. This shell offers the following features: a) an easy way of specifying rules on when to merge and when not to merge, b) a straightforward priority mechanism for resolving conflicts among rules, c) an explanation facility for recording the merging history, d) a report generating mechanism to help users understand the reasons for merging, and e) a self-documenting mechanism that documents all the merging rules that have been defined in the shell, ordered by priority.

The merging shell is implemented using an object-oriented paradigm in CLOS. It has been a part of operational TRANS (after extensive testing) since July 1993. It has fulfilled all performance expectations, and has considerably simplified the process of implementing new or changed requirements for merging. The users are pleased with its report-generating and self-documenting features.

2. Introduction

2.1. Planning and Scheduling HST Observations

Once a proposal for observing with the HST has been approved, the astronomer submits a detailed observing plan. This plan contains specific exposures, instrument configurations, and constraints on exposures. There are a variety of scientific reasons why an astronomer might place additional constraints on exposures and between exposures. For example, exposures may be designated as acquisition or calibration exposures. Some exposures might be executed at particular times, specific orientations on the sky, or within a designated time interval. In the case of time-variable phenomena (e.g. binary stars, Cepheid variable stars) the proposer may require repeated observations at specific time intervals (Miller and Johnston 1991).

2.2. TRANSformation - The Big Picture

The process of converting a proposer's specification into a form suitable for scheduling is called *transformation*. Transformation involves several tasks including determining the ordering of the observations, grouping to minimize telescope

movement, instrument reconfiguration and other overheads, providing extra observations and instrument activities necessary to obtain the requested data, and organizing observations into a hierarchy of higher-level structures for purposes of scheduling observations.

The TRANSformation expert system (TRANS) converts proposals for astronomical observations with the HST into detailed observing plans. In other words it performs all the tasks associated with the process of transformation, as described in the previous paragraph. For a detailed description of its workings, see (Gerb 1991). An extension of Common LISP (Steele 1990) was used for its implementation.

3. "Merging" Observations into a Hierarchy

Merging is defined as the process of organizing observations into a four-level hierarchy for purposes of efficient scheduling (Fig. 1).

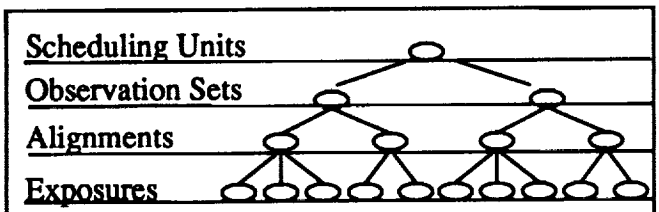

Figure 1: The Four Merging Levels

At the lowest level are the exposures that are obtained from the observing plans. During the process of merging, the ordered exposures are grouped into contiguous disjoint sets called alignments. Alignments are then grouped into observations sets (obsets) and finally, obsets are grouped into scheduling units.

All exposures in an alignment must use the same HST pointing and orientation, must generate science and engineering data at the same rate, and must have only small time gaps between successive exposure members. Grouping of exposures into alignments is important for two reasons. First, exposures in the same alignment can be commanded for much more efficient use of spacecraft time. Second, alignments are the basic units of planning for the downstream scheduling system.

Obsets are groups of alignments that can be executed without a change in the operating mode of the pointing control system. HST usually depends on positional monitoring of pairs of stars (called guide stars) to maintain its pointing. A series of alignments can be in the same obset if they all can use the same guide star pair, or if they do not use guide stars.

Scheduling Units are groups of obsets that are scheduled together. When scheduling an obset requires the next obset to be scheduled immdiately afterwards, both are placed in the same Scheduling Unit.

While merging, the ordering of objects is preserved, i.e. there is no change in the ordering of observations due to merging. So, if there are n exposures in an observing plan, numbered 1 thru n, and they are in ascending order, we first create a new alignment (numbered 1) for exposure 1. Then we see if exposure 2 can be put in alignment 1:

- If yes, exposure 2 gets included into alignment 1. Now, we see if exposure 3 can be included in alignment 1. If yes, alignment 1 now has three exposures. If no, we create a new alignment (numbered 2), and exposure 2 goes into alignment 2.
- If no, alignment 1 has exposure 1 and we create a new alignment (numbered 2) that has exposure 2. Now, we see if exposure 3 can be put in alignment 2. If yes, alignment 2 now has two exposures, exposures 2 and 3. If no, exposure 3 goes into a new alignment (numbered 3).

We repeat this process for exposures 4 until n, always considering the latest new alignment formed, for exposure inclusion. Once we have all the alignments, we step up a level, and go through the alignments in order, grouping them into obsets. Once this is done, we group the obsets into scheduling-units. The exposure->alignment merging process is also illustrated in Fig. 2

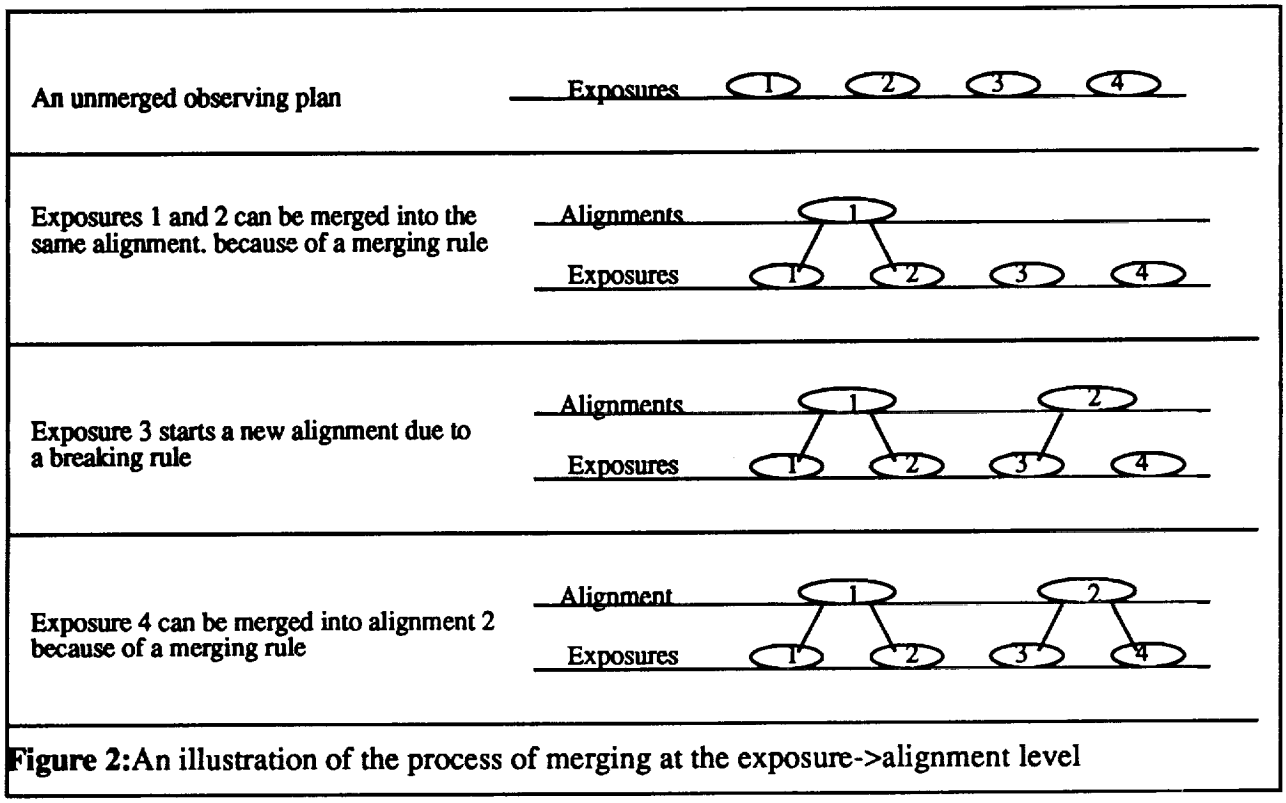

Figure 2: An illustration of the process of merging at the exposure->alignment level

Within TRANS, merging can be achieved in two ways. Under *manual-merging*, the user prescribes to TRANS how an observing plan should be merged. This is done through a "merging-file" that the user sets up before running TRANS on an observing plan. Under *automatic-merging*, the entire merging process is left to the software. Decisions on whether to merge an object into a higher-level-object are based on rules. Rules are of two types: merging, or breaking. A rule of type merging (henceforth referred to as a *merging rule*) specifies a set of conditions under which a lower level object can be included in a higher level object (that may already contain other lower level objects). A rule of type breaking (henceforth referred to as a *breaking rule*) specifies a set of conditions under which a lower-level-object cannot be included in a higher-level-object that already contains other lower-level-objects. Conflicts might arise when both merging and breaking rules may be applicable. So, a scheme for *conflict resolution* is important.

The following requirements were defined for the implementation of the TRANS merging shell:

- an easy way of specifying merging and breaking rules
- a priority mechansim for resolving conflicts among rules
- an explanation facility to document the merging history for an observing plan under automatic merging
- a facility to validate an observing plan that is being manually merged
- a self-documenting mechanism that documents all the merging rules that have been defined in the shell, ordered by priority.

4. A Declarative Merging Shell

Automatic merging, as described in the previous section is implemented using the TRANS Merging Shell. This shell provides mechanisms for encoding the rules, and defining and using any associated data structures. It also provides explanation and self-documenting facilities. In describing the shell, a notation very close to LISP (and

CLOS) syntax will be used because of the need to include implementation details, where necessary.

4.1. The Mechanisms for Declaring Knowledge

An object-oriented approach is taken to implement the merging shell. There are two kinds of objects: *data* objects, and *declarative* objects. The data objects are of the following type: exposure, alignment, obset, and scheduling-unit. These objects are doubly-linked to preserve ordering and facilitate object-traversal in both directions. The declarative objects are the rules and the slots. The rules are used to encode the merging criteria. The slot objects are used to encode knowledge used to populate some of the slots in the data objects.

Rules are the primary means of encoding the criteria for merging. A rule is defined using the construct *define-rule*. Each define-rule declaration results in the creation a rule object. The template for a rule is described in Fig. 3.

```
(define-rule
  :type <rule-type>
  :name <rule-name>
  :level <rule-levels>
  :instrument <rule-instruments>
  :priority <rule-priority>
  :test <rule-test>
  :description <rule-description>
)
```

Figure 3: The Rule Template

A description of the various parameters in the rule template is in order.

1. *<rule-type>* can be either :merging or :breaking.

2. *<rule-name>* is a short descriptive string that succintly conveys the meaning of the rule. It is used for identifying the rule in the explanation and self-documenting mechanisms, and so has to be unique.

3. *<rule-levels>* are the one or more merging levels the rule is applicable at, i.e. 'ex->al, or 'al->ob, or 'ob->su. The keyword :all may be used if the rule is applicable to all merging levels.

4. *<rule-instruments>* are the one or more instruments the rule applies to. Again, the keyword :all may be used if the rule is applicable to all instruments.

5. *<rule-priority>* is a real number. It establishes the priority of a rule, to aid in conflict resolution. The larger the number, the higher the priority of the rule. While resolving conflicts, a rule with higher priority takes precedence over a rule with lower priority.

6. *<rule-test>* is the symbolic expression that determines if the rule should fire. This expression is encoded in LISP. References to the current higher-level-object under consideration (*self*), and lower-level-object under consideration (*obj*), can be made in the symbolic expression.

7. *<description>* is a string that contains the detailed description of the rule in english. It is used by the self-documenting mechanism.

Note that the *<rule-test>* corresponds to the antecedent part of a traditional production rule. The symbolic expression that is *<rule-test>* can be arbitrarily complex and can refer to any of the properties of the lower level objects being considered for merging. It is evaluated by the inference engine. If the result is a non-null value, the rule is considered to have fired or activated. The role of consequent is played by the *<rule-type>*, which indicates the action to take in case the *<rule-test>* is "true".

An example of a rule is shown in Fig. 4.

```
(define-rule
    :name "BREAK EXPOSURES THAT
           DO NOT HAVE IDENTICAL
           ORIENTATION"
    :type :breaking
    :level :all
    :instrument :all
    :priority 2
    :test '(not
            (identical-orientation-p
              (first-ex-in-self self)
              (first-ex-in-obj obj)
            )
    :description
        "Break an exposure into a new SU if
         it does not have identical upper
         and lower limits for absolute and
         nominal orientations as the
         exposures in the previous SU."
)
```

Figure 4: An Example of a Rule Declaration

This rule enforces the condition that observations that do not have identical spacecraft orientations*, should not be grouped into the same higher level object. Hence, this rule is of type "breaking", and applies to all merging levels (ex->al, al->ob, and ob->su). It is applicable to all instruments, and has a low priority. It activates when the "identical-orientation-p" test fails. This test is performed using the first lower-level object in the latest higher-level object (denoted by *"self"*) and the current lower-level object under consideration (denoted by *"obj"*).† So, if we are at the lowest level of merging (exposure->alignment), and the latest alignment to be created is alignment 3, with exposures 7, 8, and 9, and the object under consideration is exposure 10, we merge exposure 10 into alignment 3 only if it has the identical orientation as exposure 7, otherwise exposure 10, starts a new alignment (alignment 4). Considering another case, let us assume we are merging at the intermediate level (alignment->obset), and the latest obset to be created is obset 2, with alignments 2 and 3, and the alignment under consideration is alignment 4. Further assume alignment 2 has exposure 3 as its first exposure, and alignment 4 has exposure 10 as its first exposure. If the orientations of exposures 3 and 10 are identical, alignment 4 gets merged into obset 2, if not it starts a new obset (obset 3).

The properties of a higher-level-object keep changing as new lower-level-objects are included within it by the inferencing mechanism. These properties are defined using the construct *define-slot* (Fig. 5).

```
(define-slot
   :object <object-types>
   :name <slot-name>
   :initialize-with <initital-value>
   :update-with <update-expression>
   :update-after <other-slots>
 )
```

Figure 5: The Slot Template

A description of the various parameters of the slot template is next.

*For a detailed description of how orientation constraints are dealt with in TRANS, see (Bose and Gerb 1994).

†The use of the word "self" to refer to the current higher-level-object is not without significance. A rule can be considered a method for the higher-level-objects for the merging level of a rule. "Self" would then refer to the current higher-level-object for which the rule was being executed. For slot updates, "self" appears to be a good choice for obvious reasons.

1. *<object-type>* is a symbol representing alignment, obset, or scheduling-unit.

2. *<slot-name>* is a symbol that serves as a unique identifier for this slot.

3. *<initial-value>* is the value with which the slot in the relevant data object is initialized when the object is created.

4. *<update-expression>* is a symbolic expression that, when evaluated, yields the value associated with the slot. This expression can be arbitrarily complex, and can include references to other objects.

5. *<other-slots>* are the slots that should be evaluated before this one. This feature enables an ordering in the evaluation of the slots.

Both rule and slot declarations result in the creation of objects of the corresponding types. These are in addition to the exposure, alignment, obset, and scheduling-unit objects (data objects) that are created as needed. Note that slot objects contain information on attributes of the data objects. Since the *<update-expression>* may contain a reference to another slot in the same object, it is important to specify the slot-dependencies through *<other-slots>*.

An example of a slot declaration is shown in Fig. 6.

```
(define-slot
   :name 'primary-priority
   :object 'alignment
   :initialize-with -1
   :update-after 'primary-exposure
   :update-with
     '(cond
        ((equalp obj (primary-exposure   self))
           (primary-exposure-priority obj))
        (t (primary-priority self)))

)
```

Figure 6: An Example of a Slot Declaration

This slot is defined for data objects of type alignment. Whenever an alignment object is created, a slot called "primary-priority" is automatically created and initialized to -1 for the alignment. Whenever a new lower level exposure object is added to the alignment object, all the slots in the alignment object are updated with the result of the evaluation of the "update-with" expression. In this case, since the update-with expression contains a reference to another slot called "primary-exposure", the "primary-exposure" slot needs to be populated before the "primary-priority" slot.

4.2. The Inference Engine

The Inference Engine (IE) uses the knowledge encoded in the declared objects alongwith the data in the data objects to accomplish the process of merging. The algorithm used is shown in pseudo-english in Fig. 7.

```
Algorithm Merge:

From lowest to the highest merging level  {
  set obj to first lower-level-object          ;
  while lower-level-objects remaining  {
      deduce decision based on self and obj
      if decision is to merge  {
            merge obj into self
            update slots in self }
      else
            set self to new higher-level-object;
      set obj to next lower-level-object
  }
}
```

Figure 7: Algorithm Merge Used in the Inference Engine

Merging commences at the lowest (exposure->alignment level), and then proceeds to the alignment->obset level, and finally obset->scheduling-unit level. The lower-level objects are merged into the higher-level objects in order, at each level. The decision to merge a lower-level-object into a higher-level-object is based on the priority of the rule that was activated. Rule activation is

based on the result of the evaluation of the *<rule-test>*, given the current values of *slot* and *obj*. Algorithm deduce-decision which performs rule-activation is shown in Fig. 8.

```
Algorithm Deduce-Decision:
 set decision to "break";
 set current-priority to highest rule-priority;
 while no rule has been activated and there
   are rules remaining {
       if all current-priority rules have been
         exhausted
           set current-priority to
             next lower rule-priority;
       set rule to next unconsidered rule with
         current priority;
       evaluate <rule-test> using self and obj;
       if result is "true" {
           ;rule has been activated
           record <rule-name>;
           if <rule-type> is "merging"
             set decision to "merge"
       }
   }
   return decision
}
```

Figure 8: Algorithm Deduce-Decision Activates Rules

It operates by attempting to activate rules in descending order of priority. As soon as a rule activates, it returns the decsion based on the type of the rule. The restriction that rules with the same priority have to be of the same type simplifies rule-ordering (before activation) and conflict-resolutiuon (after activation). Rule-ordering for rules with the same priority is no longer important because all rules have the same consequence, i.e. merging or breaking. Conflict-resolution for rules with the same priority does not arise, again because all rules have the same consequence, and so there are no conflicts. If no rules are activated, the default decision is "not to merge", or to "break".

5. Examples of Merging Shell Usage

Merging shell usage will be illustrated with the help of two examples.

5.1. Example 1

In this example, we make slot and rule declarations to ensure that observations that use the Faint Guidance Sensors (FGS) should be grouped into different obsets if they have different spacecraft pointings. In order to implement this requirement at the alignment->obset merging level, and keeping in mind that an alignment may have several exposures, we make use of the concept of primary-exposure within an alignment, which determines the pointing of the alignment. We declare a slot called primary-fgs for obsets, that should contain a reference to the first primary exposure within it if the exposure happens to be an FGS observation (Fig. 9).

```
(define-slot
  :name 'primary-fgs
  :object 'obset
  :initialize-with nil
  :update-with
     '(or (primary-fgs self)
          (let ((ex (first-ex obj)
                (pri (primary-exp obj))
                (fgs nil))

                (while (and (not fgs) ex)
                    (when
                      (equalp (si-used ex) )
                      (set fgs t))
                    (setq ex (next-ex-in-al ex))
                )
                (when fgs pri)
          )
     )
  )
```

Figure 9: A Slot Definition for Primary-FGS

The rule declaration is shown in Fig. 10. The *<rule-test>* essentially states that if *self* contains a primary FGS exposure, and *obj* also has a primary FGS exposure, and the two primaries do not have the same pointing, then *obj* should start a new obset.

```
(define-rule
  :name "FGS ALIGNMENTS WITH
         DIFFERENT POINTINGS"

  :type :breaking
  :level 'al->ob
  :priority 4
  :test  '(let
          ((primary-fgs-self (primary-fgs self))
           (primary-obj (primary-exp obj))

          (and
            primary-fgs-self
            (fgs-observation-p primary-obj)
            (not (same-ex-pointing
                    primary-fgs-self
                    primary-obj)
            )
          )
        )
    :description
      "Each alignment contains an FGS
       observation and the alignments have
       different pointings."
)
```

Figure 10: A Breaking Rule for Faint Guidance Sensor Observations

5.2. Example 2

This example is more complex, and demonstrates how higher level macros can be defined in LISP that use the merging shell facilities. Fig. 11 is an example of the use of such a macro, whose purpose is to prevent grouping of exposures that do not satisfy the "homogeneity criteria" into the same higher-level-object. The "homogeneity criteria" is defined to be the comparison of the values returned by a test that has, as its argument, an

exposure from the set of exposures being evaluated. If the values returned by the test for all exposures in the set are identical (a value of nil is considered to be identical with any other value), then the set is said to pass the homogeneity criteria, else it fails. The implementation of the macro itself involves implementation details that are beyond the scope of this paper.

```
(define-homogeneity-breaking-rule
   :name "BREAK PURE PARALLELS WITH INCOMPATIBLE TARGETS"
 :type :breaking
 :level '(al->ob)
 :priority 5
 :test (let ..<details intentionally left out>)
:description
 "Pure parallel exposures (s.r. PARALLEL (but not PARALLEL WITH) or s.r. EXTERN PARALLEL WITH) should not be merged into the same obset with exposures with incompatible targets. Two exposures have targets incompatible for parallel merging if:
1. Either is a solar system target.
2. Either is an external target and the target names are different.
3. One is pure parallel and the other is not."
)
```

Figure 11: An Example of the Use of the Homogeneity Breaking Rule

6. Output Products Generated

The merging shell enables the generation of two reports that have been found to be extremely useful by both the Users and Developers. The first is the "Merging Reasons Report". An excerpt from this report is shown in Fig. 12. This report enumerates the "reasons" why the observations were grouped in a certain way. The "reasons" are the names of the rules that were activated by the inferencing mechanism.

The second is due to the self-documenting feature of the shell, that creates a listing of all the rules that have been defined alongwith their explanations. An excerpt from such a listing is shown in Fig. 13. This listing is an integral part of the TRANS Scripting Guide, that contains exhaustive documentation on the requirements implemented within TRANS. The listiing of the merging rules has been found to be very useful by the Configuration Management/Quality Assurance personnel charged with keeping the TSG up-to-date.

7. Validation of Manual Merging

As was pointed out earlier, an observation plan may be "manually merged" within TRANS by explicitly specifying the observation hierarchy that TRANS should use. This feature allows the user to circumvent the automatic merging mechanism within TRANS. Even while using "manual-merging" however, the user wants to be informed of all merging and breaking rules that may have been violated in selecting the specific manual-merge hierarchy. This task, which is called "validation of manual merging", is achieved within TRANS by first merging automatically using the merging and breaking rules, and then comparing the results to the specified manual-merging hierarchy. In case of conflict, the appropriate merging or breaking rule is output alongwith a diagnostic informing the user of a rule violation. An example of this diagnostic is shown in Fig. 14.

8. Implementation and Experience

As has already been mentioned, the TRANS Merging Shell was implemented in CLOS using an object-oriented paradigm. It should be pointed out, however, that TRANS is implemented in an extension of LISP called the transformation command language (XCL). XCL is implemented using the LISP macro facility, and supports a procedural rule syntax and allows abstraction for underlying data structures (Johnston and Gerb 1992). Hence, in order to

```
TRANSFORMATION VERSION DEVELOP 66.0
MERGING REASONS REPORT
GENERATED 12-7-1994 13:32:21
PROPOSAL 274 VERSION C
TRANSFORMED USING FULL-TRANS

SU 0027401:

  OBSET 01:
  BREAK REASON: NO MERGING RULE

    ALIGNMENT 01:
    BREAK REASON: NO MERGING RULE
      EXPOSURE 01 (1.0000000):
      BREAK REASON: NO MERGING RULE

    ALIGNMENT 02:
    MERGE REASON: MERGE ALIGNMENTS INTO OBSETS, PRIORITY 1
      EXPOSURE 01 (2.0000000):
      BREAK REASON: NO MERGING RULE

SU 0027402:

  OBSET 02:
  BREAK REASON: NO MERGING RULE

    ALIGNMENT 01:
    BREAK REASON: DONT CASUALLY MERGE DIFFERENT CONFIGS OR PRIORITIES, PRIORITY 2
      EXPOSURE 01 (3.0000000):
      BREAK REASON: NO MERGING RULE
<report truncated>
```

Figure 12: An Excerpt from a Merging Reasons Report

Merging Exposures into Alignments

This section of the Transformation Scripting Guide was created by the self documenting TRANS Merging Mechanism on 04/15/94, Version #21.

Rules for MERGING, Priority:6
Two exposures should be merged into the same alignment if:

1. MERGE PARALLEL WITH AND PRIMARY
   Coordinated parallel exposures (exposures with the PARALLEL WITH sr) must be in the same alignment as their primary. This is not the case when coordinated parallels are being treated as pure (which does not apply to exposures with config=S/C).

2. MERGE PARALLEL WITH SAME PRIMARY
   Exposures that are PARALLEL WITH the same primary should be merged into the same alignment. This is not the case when coordinated parallels are being treated as pure (which does not apply to exposures with config=S/C).

Rules for BREAKING, Priority: 5
Two EXPOSUREs should be broken into different alignments if:

1. BREAK CONDS UNLESS DATA IS IDENTICAL
   Exposures with the COND sr should never be merged with other exposures in the same SU unless they are conditional on exactly the same lines with exactly the same conditions (ignoring spaces).

2. BREAK COSTAR/AFM/POM UPLINKS FROM OTHER EXPOSURES
   COSTAR exposures with the REQ UPLINK sr should be in their own obset. This is because mechanism history keeping cannot be performed with these exposures.

3. BREAK DIFFERENT WFPC/WFPC2 MODES
   WFPC/WFPC2 exposures with different modes should be in different alignments.

4. BREAK EXPOSURES REQUIRING DIFFERENT POINTINGS
   Exposures requiring different spacecraft pointings should be broken into separate alignments. Exposures are said to have the same pointing if the following two apply:

   (a) They point to the same target. (Name must be the same. Same coordinates is not sufficient.

   (b) They have the same V2/V3 pointing. V2/V3 pointing is derived from the qaapertures table and the POS TARG offsets (see section 2.1 for details on computing V2/V3 pointing).

Figure 13: An Excerpt from the TRANS Scripting Guide Illustrating the Self Documenting Feature of the Merging Shell.

ERROR ENCOUNTERED

Exposure 15.0000000 was merged into the previous scheduling unit due to manual merging.

This violates a BREAKING rule called

'BREAK ALIGNMENTS USING DIFFERENT GUIDE STARS' (priority 5).

DIAGNOSTIC TYPE: MERGING RULE VIOLATED DURING MANUAL MERGING

IN PROPOSAL 305

IN OBJECT EXPOSURE

Figure 14: An Example of a Manual Merging Validation Diagnostic

interface with the rest of TRANS, extensive use was made of XCL facilities for object creation and indexing. The examples in this paper all use CLOS syntax to refer to object slots and methods for ease of understanding. It was felt that introducing new syntax (XCL) in the examples was unnecessary.

A brief note on the necessity of having a separate mechanism for declaring slots for the data objects (separate from the actaul class definitions of the data objects) is in order. The slot defintion mechanism enables the following:

- use of arbitrarily complex symbolic expressions, the evaluations of which yield values for slot-value updates,
- specification of ordering in the updating of slot-values (important when there are interdependencies between slots),
- ease of adding new slots and modifying old ones, without having to deal with the implementation of the class definitions,
- separation of the declarative component (rules and slots) from the implementation.

The Merging Shell has been an integral part of the TRANS expert system for over 17 months. It has greatly simplified the modification of old rules and the declaration of new ones. The users are happy with its facilities for validation of manual-merging and report generation.

9. Conclusions

We have described here a declarative shell to facilitate organization of space observations into a hierarchy based on pre-specified rules for hierarchical organization. Even though the underlying principles of the shell are relatively simple, it offers a powerful way of expressing and dealing with knowledge related to the process of hierarchical organization. Its object-oriented implementation in CLOS provides for a seamless integration with a large expert system implemented in LISP. The concepts behind the merging shell can be applied to any space observation program where observations have to be organized into a hierarchical structure for purposes of planning and scheduling to satisfy resource constraints.

REFERENCES

(Bose and Gerb 1994) Bose, A. and Gerb, A. Propagating Orientation Constraints for the Hubble Space Telescope, (to appear in) *J. of Robotics and Computer Integrated Manufacturing.*

(Gerb 1991) Gerb, A. Transformation Reborn: A New Generation Expert System for Planning HST Operations, *J. of Telematics and Informatics, v.8 n.4.*

(Johnston and Gerb 1992) Johnston, M.D. and Gerb, A. *Transformation Script Programmer Manual (Rev. K)*

(Miller & Johnston 1991) Miller, G.E. and Johnston. M.D. A Case Study of Hubble Space Telescope Proposal Processing, Planning, and Long-Range Scheduling, *Proceedings AIAA-91.*

(Steele 1990) Steele, G. *Common LISP, The Language (2nd ed.).* Bedford, MA: Digital Press.