Examples for the Installation

Document Version 3.6.00

of the
Hierarchical Evaluation Tool

HIT

Version 3.6.036

Editor: J. Mäter

History:

The predecessors of this document were written by M. Schmidt and P. Lengewitz.

Abstract:

One example for each analysis method is provided to test the installation of the HIT-system. All examples are based on the *Office Example*, which is explained in the first section. The second section of this document comments on the differences between the examples. This description is designed to apply for all HIT installations and therefore does not contain any filenames, since they depend on the operating system used.

The title page contains two version numbers: That of this document and the version number of the HIT system to which this document applies. The document version and date is also included in the page footer. Here lower version numbers may appear for unchanged pages.

Address:

Universität Dortmund
Informatik IV
J. Mäter

D-44221 Dortmund

Telefon: (Germany)-(231) 755-2411
Telefax: (Germany)-(231) 755-4730
E-Mail: hit@jojo.informatik.uni-dortmund.de

Contents:

0. Introduction:

One example for each analysis method is provided to test the installation of the HIT-system. All examples are based on the *Office Example*, which is explained in the following section. The second section of this document comments on the differences between the examples.

This description is designed to apply for all HIT installations and therefore does not contain any filenames, since they depend on the operating system used. Still the following naming conventions hold: The (first letters of the) method name is contained in the filename and control files have the suffix *ctl* or *control.* The exact filenames can be found in the different Installation Guides to the HIT-system.

1. Description of the Office Example

As a background for this example, imagine a hypothetical office, in which certain "reports" are to be compiled on a more or less regular basis. Please remember that a hierarchical model structure, i.e., a sequence of model "layers" and corresponding interface "levels" is aimed at.

We describe the activities necessary for furnishing one of our example reports with a service type *report*. Let *report* be based on certain lower-level services, *prepare* (some sub-reports), *join* (these sub-reports), *examine* (the total report) and *archive* (any final activities).

The complete specification of *report* in terms of HI-SLANG also encompasses, in addition to the information given above, the description of how the used services may be influenced (via parameters), of the actual influence imposed and of the temporal structure of using the services (via control constructs).

```
TYPE report SERVICE;                   {naming of service type}

USE SERVICE
     prepare  (time : REAL);           {based on four services}
     join     (time : REAL);           {with one parameter each}
     examine  (time : REAL);           {the time needed}
     archive  (time : REAL);           {in seconds}
END USE;

BEGIN

     LOOP                              {sequence of activities}
          AVERAGE 3 TIMES LOOP         {3 sub-reports}
               prepare(negexp(1/1260));{exponentially distrib.}
                                       {with mean 1260}
          END LOOP;

          join     (negexp (1/1260));
          examine  (negexp (1/1800));
          archive  (negexp (1/ 600));
     END LOOP;

END TYPE report;
```

Two instantiations of individual processes of type *report* are assumed to exist permanently, following the HI-SLANG statement

```
    CREATE 2 PROCESS report;
```

Let us now examine a machine capable of supporting *report*. Our requirements are capacities for preparing and joining the preliminary reports (we deligate these activities to a sub-office, *secretaries*) and for examining and finishing the report (we employ another sub-office, *clerks*).

Sticking, for the moment, to non-hierarchical modelling, our standard component type, *server*, can be directly used for the sub-offices: *server* provides a service, *request*, parameterised with the duration of servicing this request. The HI-SLANG declaration of the two machine components is simply

```
    COMPONENT secretaries,clerks : server;
```

Let us first combine load and machine of a single-layer model: USEd *report.prepare* is linked to PROVIDEd *secretaries.request*, the qualifications (via dot notation) resolving any existing name conflicts.

```
    TYPE office_1 MODEL;                 {HI-SLANG specification}

         TYPE report SERVICE;

         END TYPE report;

         COMPONENT secretaries, clerks : server;

         REFER report TO secretaries, clerks
         EQUATING
              report.prepare     WITH     secretaries.request;
              {join, examine analogously}
              report.archive     WITH     clerks.request;
         END REFER;

    BEGIN
         CREATE 2 PROCESS report;
    END TYPE office_1;
```

Let us now assume that we want to model more precisely what happens within the *secretaries* office. Obviously, the services *prepare* and *join* required from this office, will have to be "refined". Assume that, within secretaries, employees are sitting in front of small office computers and are using the text manipulation and retrieval services of this equipment. Assume also, less abstractly, that both *prepare* and *join* can be explained in terms of the sub-activities *fetch* (a document page from a central site), *change* (the contents of a document page), *think* (between computing-related activities) and *store* (a page at the central site). Let the activity pattern be precisely specified by the following service type *edit*:

```
TYPE edit SERVICE;

     USE SERVICE
          fetch    (time : REAL);
          change   (time : REAL);
          think    (time : REAL);
          store    (time : REAL);
     END USE;

BEGIN

     AVERAGE 10 TIMES LOOP           {10 pages per document}
          fetch (negexp (1/3));      {retrieve page}

          AVERAGE 5 TIMES LOOP       {5 activities per page}
               think (negexp (1/20);
               BRANCH                {"long" or "short" update}
                    PROB 0.75 : change (negexp (1/ 2));
                    PROB 0.25 : change (negexp (1/10));
               END BRANCH;
          END LOOP;

          store (negexp (1/3));      {return page}
     END LOOP;

END TYPE edit;
```

The *edit* service uses a new bottom level consisting of the *request* services provided by three server type components: *link* (for fetching and storing information centrally), *computer* (for local update operations) and *humans* (doing the "thinking"). We can design a new component type *handling*, as providing the *edit*-service:

```
TYPE handling COMPONENT;

     PROVIDE SERVICE
          edit;                      {"exporting" service, edit}
     END PROVIDE;

     TYPE edit SERVICE;

     END TYPE edit;                  {edit, from above}

     COMPONENT link, computer, humans : server;

     REFER edit TO link, computer, humans
          {linking}
     END REFER;

END TYPE handling;
```

Inducing an instance of *handling* into the earlier *office_1* model (replaceing the server *secretaries*) yields a new *office_2* model type, which is hierarchically two-layered. We could obviously go on: Refining *clerks*, or *link*, or *computer*, depending on our interest.

Actually, our models are somewhat unrealistic under the aspect that, with the above specifications, HIT implicitly assumes unlimited resources (for the knowledgeable: "Infinite Server" behaviour), for every component: An unlimited number of employees, in both the *secretaries* and the *clerks* office, each employee with his or her own *computer* and own *link*. To correct this situation, if necessary, recall the mentioned "monitor" property of components. If we declare, in *office_1*:

```
    secretaries : server ( LET SCHEDULE := FCFS (5) )
```

and in *handling*:

```
    link : server ( LET DISPATCH := SHARED ) { PS }
```

we would immediately achieve a limitation of the *secretaries* office to 5 simultaneous activities (i.e. to 5 available employees), waiting situations to be resolved in FCFS order, and a "Processor Sharing" property of the *link* (amounting to a bus instead of the implicitly assumed star-like connection). Regarding the available space, however, we maintain the former declaration in the following.

Let us analyze a model of the initially introduced model type, *office_1*. A possible EXPERIMENT specification is given below:

```
    EXPERIMENT office_analysis_1 METHOD SIMULATIVE;

    BEGIN

        EVALUATE MODEL off_1 : office_1;
            EVALUATIONOBJECT clerks_office VIA off_1.clerks;

        BEGIN
            MEASURE TURNAROUNDTIME AT clerks_office
                ESTIMATOR CONFIDENCE LEVEL 95;

            CONTROL AT clerks_office
                STOP CONFIDENCE LEVEL 95 WIDTH 5
                MEASURE TURNAROUNDTIME;
        END EVALUATE;

    END EXERIMENT office_analysis_1;
```

In this example experiment we are in fact asking for

- a simulative analysis
- of a MODEL, *off_1*, of type *office_1*
- with the sub-office, *clerks*, to be evaluated under the name, *clerks_office*
- where the TURNAROUNDTIME is to be measured, i.e. the response time of services accomplished by the *clerks_office*
- and a 95 confidence interval of the mean value of this performance variable is to be estimated and displayed in a standard table output format.
- the simulation ending when the 95 confidence interval for the *clerks_office* is enclosed by the (mean +/- 5) interval.

As a last step in our illustration, let us now analyze an *office_2* model (the one with the refined version of the *secretaries* sub-office). We shall analyze this model employing pre-analysis of the type *handling* component, resulting in a flow-equivalent aggregate server. This aggregate can subsequently be embedded into the original model, and a simulative analysis of the resulting model may follow.

The introductory pre-analysis step is requested by the EXPERIMENT specification:

```
    EXPERIMENT aggr_hand METHOD ANALYTICAL "separable";
    BEGIN

        AGGREGATE handling OUTPUT "HANDLING";
            CREATE 20 PROCESS edit;
        END AGGREGATE;

    END EXPERIMENT aggr_hand;
```

2. Description of particular Examples

There is one example for each analysis method. Each example evaluates the performance measures POPULATION, THROUGHPUT and TURNAROUNDTIME for the components *secretaries* and *clerks* of the model. The different methods of analysis yield the following differences between the examples:

2.1 DOQ4-Example

First, the component type *handling* is aggregated by calling the HIT-system with the control file for the aggregation (with method name "doq4a" appearing in the filename). Then the aggregated component is used in the experiment (files with method name "doq4b"), which produces the exact results of the Office Example.

2.2 LIN2-Example

The LIN2-method is an approximative method of analysis. The results therefore differ slightly from those of the DOQ4-example.

2.3 MARKOV-Example

In this numerical version of the Office Example, the stop condition ACCURACY 0.1 OR CPUTIME 200 is inserted, and the results will subsequently show small deviations in relation to the DOQ4-example.

2.4 Simulative Example

The distinction of this simulative version of the Office Example is its stop condition. If the stream TURNAROUNDTIME of the component *clerks* has reached a certain accuracy or the simulation of the model reaches more than 7000 units of modeltime, the experiment is stopped. Depending on the performance of the CPU, the results can differ from the exact DOQ4-results.