HSII Data Replication Triggers

This document is intended to provide both a Summary / Executive Overview and a technical overview of HSII's Data Replication Trigger process.

The "Summary / Executive Overview" is presented immediately. Following this, a "Quick Table of Contents" outlines the remaining contents of this document. Please refer to the "Guide for Readers" section for how to proceed.

Summary/Executive Overview

Concerns raised about the stability and robustness of HSII's Data Replication Triggers, in particular as they pertain to MC3, are valid concerns. After an in-depth review, we firmly believe this part of the system is not ready for heavy production use in its current form (beyond a very small HMO beta-test site).

Recently, HSII has indicated that they have made extensive enhancements to the overall transaction processing mechanism as well as the data replication trigger process (these are closely intertwined). These system changes are now in beta testing by HSII. We are eager to receive these enhancements and examine them on our system as soon as HSII releases them to us.

In the meantime, we have devised two preliminary approaches, either of which will improve the performance, stability, and robustness of the existing replication trigger process.

The first approach relies upon HSII overall trigger mechanism and enhances the processing which takes place after the data comes out of the transaction processing queue. All of the code here is outside of the HSII core system and modifications here are appropriate, from HSII viewpoint, to individual client / customer sites. Resource and time requirements for this approach are modest. Furthermore, this approach is considered "in synch" with HSII overall strategic direction.

The second approach, a kind of renegade approach, bypasses HSII's transaction process all together. It would require us to build our own, independent replication trigger queue / processor. An advantage is that it avoids the already complex and overburdened general transaction processors. Two disadvantages are that (1) it is not a trivial undertaking and (2) it is unclear whether it is in HSII's overall strategic direction.

Factors which would indicate one approach over the other include our ongoing relationship / alliance with HSII, potential client demands, available resources and time. These are not technical factors; rather, they require executive resolve.

Independent of these, we feel we have sufficient knowledge of the existing system and expertise to pursue the final design and implementation of either approach.

A Quick Table of Contents

Guide for Readers

This document is intended as a general purpose, summary description of the HSII Data Replication Trigger process for both non-technical and technical readers alike. However, each audience will have a different focus on its contents. Please refer to the appropriate sub-section below.

For Non-Technical Readers: This documents may contain references to internals of the HSII application environment which may be unfamiliar. This should not prove to be a barrier to understanding the overall process. You may not find the whole document useful, but you should at least skim through the whole document. You should pay particular attention to the following sections:

- Summary / Executive Overview.
- The Data Replication Trigger "Map" — a brief review of this will give you a good idea of the innate complexity of the replication trigger system.
- Concerns in Light of Current Knowledge — to provide more detail on our basic technical concerns.
- Alternatives — to provide some pros and cons for possible decision-making purposes.

Additionally, some memos are attached to provide the rationale for our study and this document.

For Technical Readers: This document is intended to provide the larger framework of the trigger process and a starting point for further study / analysis. You should read and understand the entire document. Pay particular attention to the following sections:

- A Re-Examination of Replication Triggers
- The Data Replication Trigger "Map" — pay close attention to circuit names and global references.
- Concerns in the Light of Current Knowledge.
- Result of Conference Call with HSII Tools Group (Rick Alm) — to provide some insight on HSII's background and directions on the replication trigger process.
- Recommendations — to provide the "next steps" action plan and a reference for future documentation.

Copyright © 1995, KHP Services Inc.

Prepared by: Jeff Szuhay
KHP Services, Inc.
P. O. Box 898837
Camp Hill, PA 17089-8837
(717) 730-1827

A Re-Examination of Replication Triggers

As a result of Jim Tripp's testing of HSII's replication triggers and his concerns expressed in his memo of February 3,1995 (see attachments), a team was formed to more closely examine the trigger process. The team consisted of Bill Hanna, Jeff Szuhay, Larry Faraci, Trevis Becker, and Dave Nebinger.

The objective of the team was to rapidly gain in-depth technical knowledge of the existing system. From that analysis, we hoped to understand current concerns, and find possible solutions to the concerns (as outlined in the memo of 3/2/95—see attachments).

This document is the result of that analysis and a telephone conference call focusing specifically on the replication triggers process with Rick Alm of the HSII Tools Group and Karen Bostwick of HSII (agenda of this conversation can be found in the memo of 3/13/95—see attachments).

A Brief Analysis of the Replication Trigger Process

Our analysis focused primarily upon processes and routines. While some emphasis was given to data structures, we did not do a complete entity-relationship analysis (this is, however, recommended for a later date).

Trigger Setup. Replication triggers can be set for pre- and post-processing data operations (insert, modify, delete). In practice, only post-processing triggers are normally used. The trigger setup procedures consist of a set of screens which correlate a given file/field on the HSII system to a SQL table/field in the target relational system. The results of the setup process are a set of interrelated globals which internally describe (parameterize) the trigger process.

The mapping of between HSII databases and any other database is strictly a one-to-one mapping. That is, all the fields in one HSII file can be mapped to one and only one database/file/table in the target system. The target database may have a different name and any or all of the fields may have different names. However, with replication triggers an HSII database cannot be split among multiple target tables nor can multiple HSII databases be merged into one target table.

Also, there is a site-setup flag which indicates whether *any* replication triggers should be used on the system (a system trigger flag). This flag turns on/off all replication triggers. It must be set to "on" for any replication trigger to be used.

Trigger Use. Once a replication trigger is set up, it is invoked implicitly by the following code generator processes: [`INSERT`], [`MODIFY`], [`DELETE`], [`GROUP MODIFY`]. These are typically activated by the user upon a "File" operation and occur at such a low level the user is typically unaware of their use. They can also be explicitly invoked by the programmer and given points in a program by the [`ENQUEUE`

TRANSACTION] process; such use is solely at the programmer's discretion and entirely out of the user's control.

Replication triggers consist of three distinct phases:

1. Building the replication trigger and enqueuing it to the transaction processor.
2. Processing / managing the transaction queues.
3. Dequeuing the replication trigger and processing it.

Building & Enqueuing the Replication Request. Examination of the low-level decomposition of trigger activation processes (given above) shows how the trigger mechanism works in MUMPS. This is shown in the following code outline.

```
•
•       (interactive screen processing logic here)
•
```

User Hits "file" key (usually [F6]). This "permits" the execution of one of [INSERT], [MODIFY], or [GROUP MODIFY].

Low level process of user program does the following:

```
1. Is system trigger flag set?
   yes (system triggers is on):
        call BUILD^ZTRIG to build triggers.
        do any fields have replication pre-triggers?
        yes (1 or more fields are to be replicated):
            enqueue replication request/data: RUN^ZTRIG
        no (no fields to be replicated):
            do nothing.
       continue user processing.
   no, system triggers off:
        ignore triggers/do nothing.

2. Update/write HSII database & associated indexes.

3. Is system trigger flag set?
   yes (system triggers is on):
        (Note: BUILD^ZTRIG only called once, above)
        do any fields have replication post-triggers?
        yes (1 or more fields are to be replicated):
            enqueue replication request/data: RUN^ZTRIG
        no (no fields to be replicated):
            do nothing.
       continue user processing.
   no (system triggers is off):
        ignore trigger/do nothing.
```

Interactive control of user program is returned to user.

```
•
•       (interactive screen processing logic here)
•
```

Please note that for repeated group processing ([GROUP MODIFY] process) the process is complicated by additional "up-front" triggers whose purpose is not obvious at this time. Therefore, repeated group records which have replicated fields cause even more overhead / burden upon the transaction processing mechanism. This will have to be explored further.

Also, because the trigger mechanism is "built-in" at the low-level of code generator processes, it is typically beyond the usual consideration of the application programmer (except the [ENQUEUE TRANSACTION] process, which is explicitly invoked and controlled by the programmer with a high-level process).

Processing/Managing the Transaction Queues. The transaction queue collectively consists of one master queue, a suspend queue and several other auxiliary queues whose precise function is yet to be determined. These are all managed by a separate "transaction processor" process. It is essentially a background, batch, or phantom MUMPS process which perpetually examines the state of the queue and directs processing depending upon the transaction request type. There will be at least one such transaction processor running on a production system; additional background processors may be run depending upon the number and type of transaction requests (these are "tunable" by the site system administrator).

Historically, the transaction process was built for a specific module—Membership. Consequently it is known as ^MBT077 and has been considered a part of that application module (subject to application release schedules). Efforts are currently underway to move the transaction processor out of its original application module and make it an official part of system software (therefore subject to system release schedules). As a system process, it will become ^ZTRAN. At this writing, ^ZTRAN is in beta-testing by HSII and will certainly be an integral part of system release 4.0 or possibly an interim patch to the current system release.

Replication triggers are recognized by the transaction processor from their identifying type of "ZTRIG_REPL" (other transaction types were not considered).

If a transaction request for a given transaction type (i.e. "ZTRIG_REPL" type) fails more than a specified (configurable) number of times, all transaction requests of that type will be suspended (moved to the suspend queue) until the error is resolved and various counters / flag are manually reset.

Dequeuing & Processing the Replication Request. Once the transaction processor determines that a transaction request is not suspended, it processes that request (in this case ZTRIG_REPL requests) to its completion. Regardless of the number of transaction processors, transaction requests of type ZTRIG_REPL are "single threaded." That is, a single request is processed in strict chronological sequence; other ZTRIG_REPL requests are held in suspended state until the current

request completes. This design ensures that replication requests are performed as they occur thereby eliminating some nasty synchronization issues.

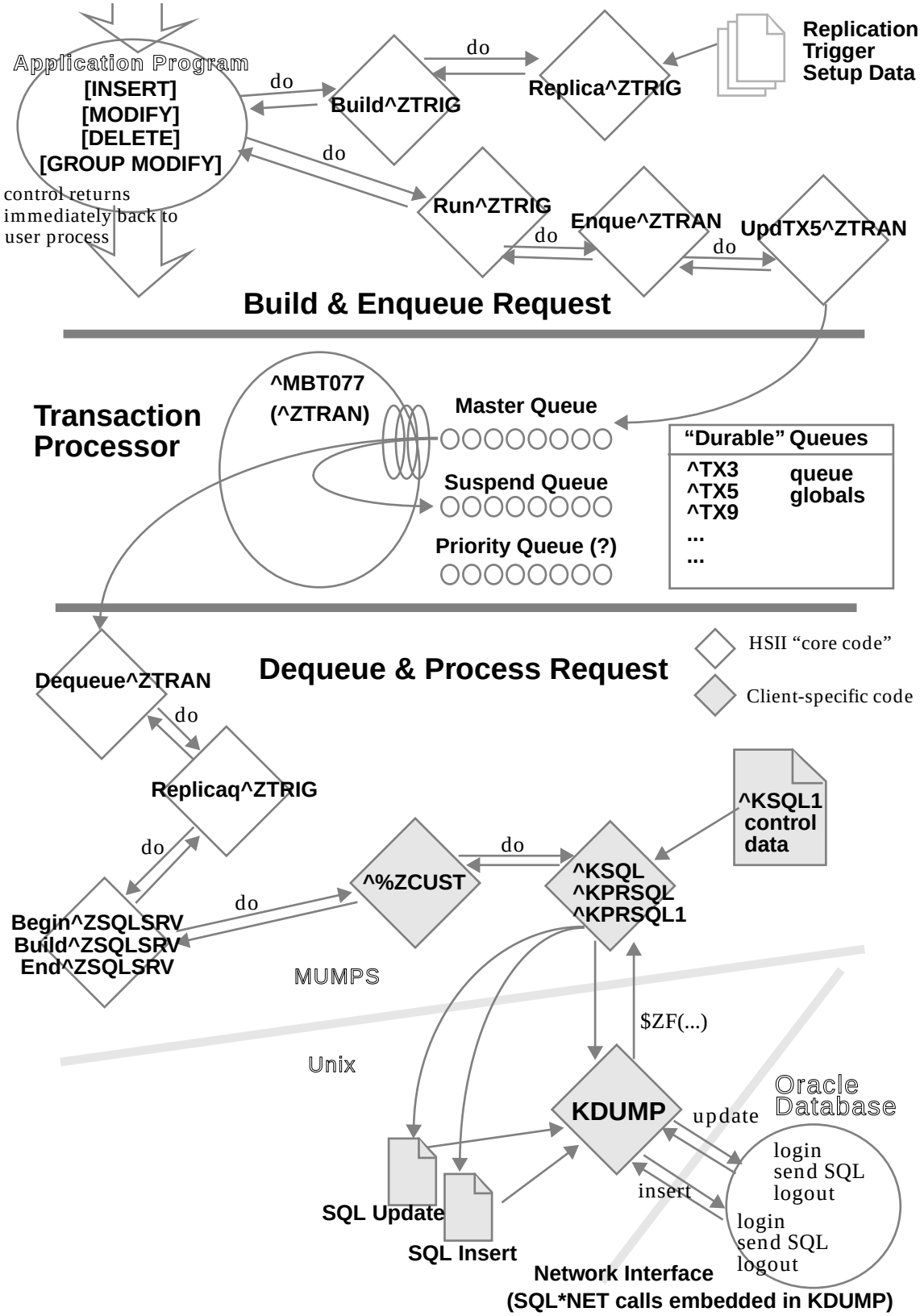
Such a design seems to negate the advantage of having multiple transaction processors. If each transaction processor dequeues a `ZTRIG_REPL` request, only one of them may process; the other processors must wait until the one processor is done. Then, again, only one processor allowed to run with the next chronological request. It seems likely this can cause a dramatic bottleneck if there are an equal number of other kinds of transactions request types (not just replication transaction requests).

HSII “core” processing routines versus Custom Site processing routines
(what the pieces do, in general. How the pieces fit together.)

(creation of both SQL insert and update statements)

(Update done first, if that fails, assume record doesn’t exist and send insert. If that fails, return error... Why this is so, future enhancements possible, etc.)

The Data Replication Trigger "Map"



This map is described in the text on previous pages.

Concerns in the Light of Current Knowledge

As a result of our examination of the overall trigger process, a number of concerns became obvious. Some of these have already been mentioned in Jim Tripp's memos and elsewhere.

Our concerns will be presented in no specific order; this is intended to illuminate observed anomalies and failures of the existing system with the light of our new knowledge in some of the "inner" technical details.

Performance under load. Five areas of greatest concern regarding performance are the following:

1. The ability of the general transaction processor to handle the increased traffic due to replication. It is not known to us what the transaction process rate is *without* triggers on an HSII system. Therefore, we have no basis for speculation as to the impact of the added transactions from triggers to the HSII system. This is an area of great concern because we cannot begin to estimate its ultimate impact in a production environment.
2. Related to item (1) is the inability of the replication trigger process to determine if any triggered fields have changed or not. Apparently, a replication request will be generated regardless of any change in data, thus creating redundant, unnecessary SQL transactions.
3. The burdensome overhead of logging in, processing, and logging out of Oracle for *each* SQL transaction that is sent (both for insert and update). This design not only takes time for the login/logout process but also redundant re-allocation of system resources associated with the login process.
4. Currently both update and insert SQL statements are generated. Because the trigger process knows whether an insert or update is being done by HSII, generation of both is redundant and causes unnecessary overhead.
5. The ability of Oracle to rapidly process updates to tables as the number of records in its tables increases. Unclear to us at this time is how the "mirrored" HSII tables are indexed (for performance) and how SQL statements are dynamically optimized by Oracle (again, for performance). Lack of either ability will dramatically impact Oracle's ability to process the SQL statement; it is likely that processing and time requirements will increase *exponentially* with the number of records in the mirrored tables.

   At issue here is how Oracle performs an update when the "where" clause is used. It is currently felt that using "where" with any non-indexed fields will cause the *entire* table to searched before the update takes place, even if the result is only one record.

Please note that items 1 and 2 fall within HSII core system code (not under KHPS control). Items 3 and 4 are determined by site-customizable code (under KHPS control). Item 5 is an Oracle tuning issue.

Robustness. Currently inadequate real-time feedback mechanisms exist to ensure the continued operation and / or rapid problem resolution of the transaction processor(s).

Overall robustness of the system goes hand in hand with error reporting. One cannot easily be isolated from the other.

Specific trigger types may be halted because of some problem but no notification is given to system administrators (a message *can* be sent to user's group but users must then be trained to properly respond to message if sent). The transaction processors must be continually watched by an operator (a manual sequence of steps).

At this time, the impact of robustness considerations are less apparent in comparison to the above-mentioned performance issues (when those are resolved robustness will likely come to the fore).

Error Reporting. Currently very sparse. There is an apparent fixed (configurable) limit as to how many unsuccessful transactions may suspend a given transaction type. Errors are not permanent, however, and are likely to be overwritten by the next error message.

A design issue to be immediately addressed is the way in which the failure of one type of SQL insert request automatically assumes that the record exists and therefore attempts an update. The replication trigger request should *not* generate both an insert request *and* an update request; rather it should know from the data whether new or revised data is being presented to Oracle and generate only one appropriate request (either only an insert or only an update). If that transaction fails, it should then be reported to system / database administrators. Error reporting then could provide information whether failure was due to system or network failure or whether a database synchronization problem had occurred.

At this time, the impact of error-reporting considerations are less apparent in comparison to the above-mentioned performance issues (when those are resolved error reporting will come to the fore).

Incomplete SQL transactions. We learned from Rick Alm that this is an application coding logic flaw. When we determine precisely where this occurs, we will notify HSII, and they will make a fix.

Alternatives

At this time, there are only two alternatives available to us:

1. Bypass HSII's general transaction processor.
2. Enhance existing process.

Each of these alternatives is described below with pros and cons (as seen from a technical standpoint). Both alternatives can be implemented to accommodate the concerns described in a previous section.

Bypass HSII's General Transaction Processors

Instead of calling HSII's standard routines for replication triggers, we replace those calls and call our own routines.

These routines would have to handle the extraction of data from the MUMPS database, queuing of the requests to Oracle, managing continual sessions and dialog with Oracle, and handling / posting error messages. This solution will require that various pieces be in MUMPS, some in Unix, and possibly some in Oracle. This would constitute a significant change in the overall system architecture.

Pros

- single function process—just for replication triggers
- avoids traffic jams encountered in general transaction processor
- moves queuing mechanism out of M and into Unix (for better performance tuning)
- KHPS "owns" the process—we can optimize / change at will; not dependent upon HSII for upgrading; large added value by KHPS.
- less dependence upon customer's system software level.
- provides possibility of "batching" of flat-file transactions.
- provides possibility of using SQL statements or of using internal protocols between our custom HSII routines and Oracle procedures to handle them.
- provides possibility of replicating data on many target databases (not just one)

Cons

- extensive re-write of both M portions and Unix portions.
- duration of project: 1 month (optimistic) to 9 months (pessimistic) of concerted development & testing effort.
- future maintenance and support issues to be considered.
- no guarantee that new process will be *dramatically* better than HSII's process in a production environment.

Enhance Existing Process

Such enhancements involves two parts:

1. Upgrade HSII system software (now in beta) which promises increased performance and functionality.
2. Optimize the custom code (notably the `KSQL` routine in M and the `KDUMP` routine in Unix) for faster throughput and logging.

All queuing mechanisms and data extraction mechanisms would remain as they are (supported by HSII). Apart from changes which HSII would have to make, we would be altering two specific areas of the overall process. This would not constitute any major architectural change in the overall process.

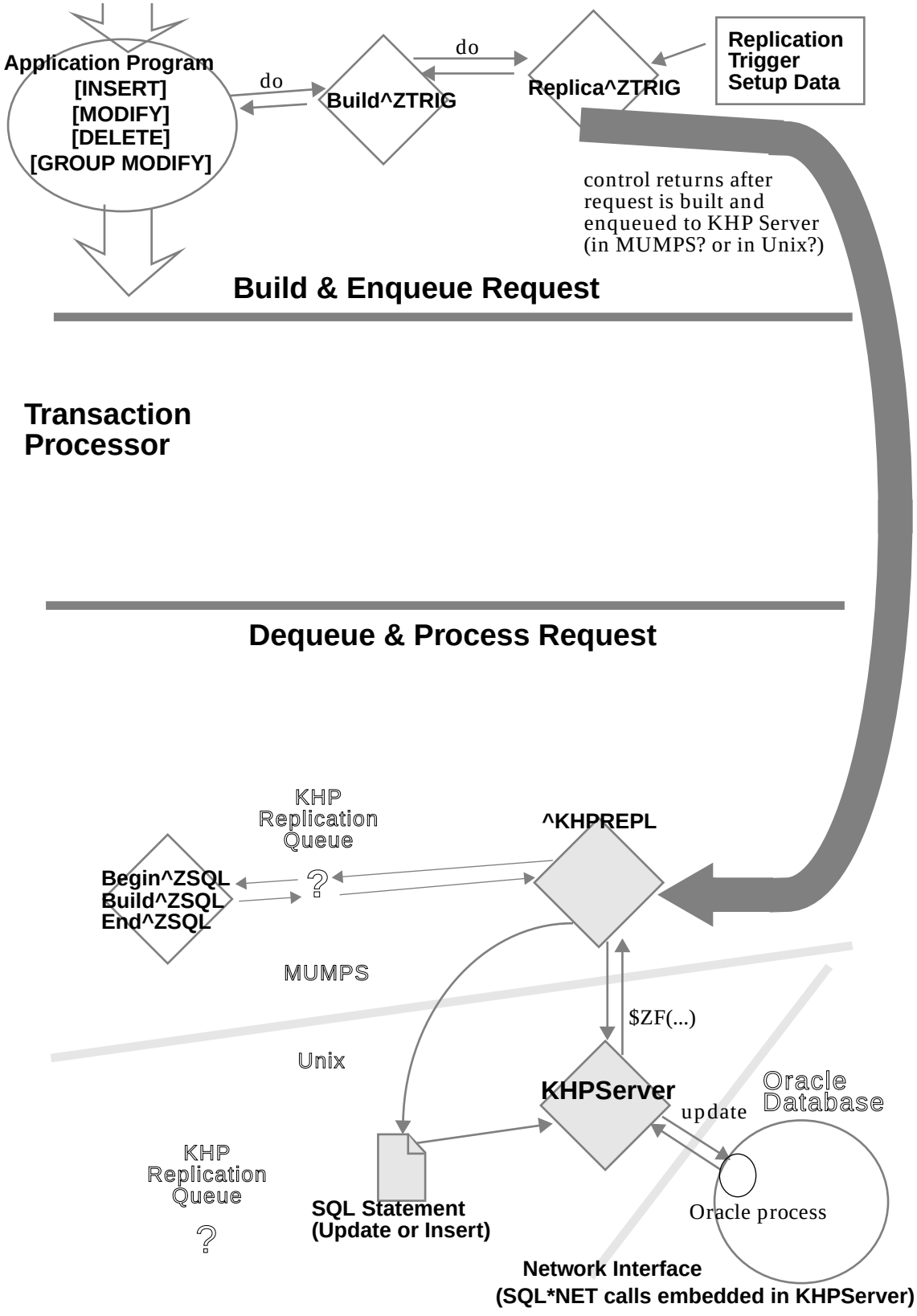
Pros

- knowledge & maintenance of transaction process remains with HSII.
- shorter time to completion: 1 week (optimistic) to 2 months (pessimistic) of focused (less staffing requirements) development & testing effort.

Cons

- ownership remains with HSII (small added value from KHPS).
- dependence upon traffic flow of general transaction processor.
- one-and-only one target replication database.

Map of Transaction Process "Bypass" Solution



This diagram is explained in the text on previous pages.

Result of Conference Call with HSII Tools Group (Rick Alm)

Based upon our telephone conference call with Rick Alm and Karen Bostwick of 3/15/95 where we learned the following:

- our analysis was confirmed by Rick Alm's description of the process;
- it was reinforced that the processing of replication triggers is "single threaded"—that is, each SQL update is processed one-at-time in time-sequential order;
- we first learned of HSII notification system, a feature to possible better track the operation of the transaction processor;
- HSII has been working to enhance both the performance and functionality of the transaction processing mechanism;
- HSII has been working to make the trigger process an integral part of *system* software;
- HSII *requires* its customers to adopt the latest system software version (as opposed to application software version);
- that the system upgrade process is far more backward compatible;
- that the system version is (relatively) independent of the application version; and
- that the upgrade process is less painful than an application upgrade (and therefore, much more likely to occur at all, or most, client sites).

Outcomes/actions as a result of this conversation are the following:

- We will shortly receive either a patch to transaction system or a complete system upgrade (which includes the patch). This system upgrade is now in beta-test stage.
- We will receive documentation/information concerning the notification system, so we can learn more about it in relation to monitoring the transaction process (especially for replication triggers).

Recommendations

Recommendation One: Perform a thorough entity-relationship analysis. This will be necessary regardless of how we process from here.

An E-R analysis will provide us with the following:

- a complete data map of *all* data elements involved in the process.
- the availability and inter-relationship of data elements so that we may focus on *precisely* the data elements we need and those we can ignore.
- a tool to provide information to possibly streamline the SQL transactions thereby "tuning" the replication triggers to our Oracle database.

Recommendation Two: From what we learned from our phone conversation with Rick Alm, I believe it is in our best interests to continue with HSII's standard transaction processing mechanisms.

- We will be receiving either a patch or a system release with enhanced trigger functionality; this should be thoroughly tested.
- We should enhance our custom M and Unix programs for faster throughput.
- We should only consider bypassing HSII's standard transaction processing mechanism after testing both their and our enhancements to the existing systems and determining that they not reliable or robust.
- Alternatively, if batch processing of flat-file transactions (such as exports directly to Oracle) cannot be processed any other way and this is a firm business requirement, bypassing HSII's standard transaction processing mechanism becomes a valid, perhaps necessary, solution. It is not yet clear whether this reason alone is enough justification to bypass HSII's mechanism; other solutions may require less time and resource.

Attachments

Jim Tripp's "Trigger Testing" memo of 2/3/95

Jim Tripp's "Trigger Testing" memo of 2/27/95

Jeff Szuhay's "HSII Trigger Process Optimization" memo of 3/2/95
This memo served as the basis for analysis which results are this document.

Jeff Szuhay's "HSII Trigger Process—In-depth Review" memo of 3/13/95
This memo served as an outline/agenda for a conference call between Rick Alm and Karen Bostwick of HSII and Ken Nissley, Bill Hanna, Jeff Szuhay, and Larry Faraci of KHPS on 3/15/95.