

Using database technologies to transform low-level stream data for client analysis

Rebecca Mancy
University of Glasgow
mancyr@dcs.gla.ac.uk

Richard Thomas
University of Western Australia
richard@csse.uwa.edu.au

Steve Draper
University of Glasgow
s.draper@psy.gla.ac.uk

Phil Gray
University of Glasgow
pdg@dcs.gla.ac.uk

Gregor Kennedy
University of Melbourne
gek@unimelb.edu.au

Abstract

GRUMPS, a Generic Remote Usage Measurement Production System, collects streams of computer-generated events representing low-level actions and stores these in a database. We have successfully created repositories of 5 million events in about 2GB of semi-structured data. These generic data represent a valuable source for multiple investigations but require significant transformation before domain-level analysis can begin. This is a difficult and time-consuming stage with this approach, and requires the correct choice of tool. Because of the large volumes of data, we chose to carry out transformations within the database. To benefit from previous experience effectively, techniques and queries need to be optimised for both re-usability and speed of execution with the chosen tool. We have found it useful to group investigations according to the high-level data required, using an intermediate representation from which data for multiple investigations can be generated, and developing techniques using re-usable stored procedures.

1. GRUMPS

The Generic Remote Usage Measurement Production System or GRUMPS [GRUMPS, 2001] is being developed at Glasgow University. The GRUMPS project aims to explore current and emerging technical possibilities for collecting and exploiting usage data e.g. logins, command calls, web page

visits, etc. The project has a number of aspects: for instance developing a novel network infrastructure to support this [Gray 2004], and developing control and management mechanisms to allow an investigator to change dynamically and remotely what is collected.

This new flexibility in collection, along with the recording of generic user actions that lend themselves to a wide range of investigations, allows multiple researchers to adapt collection and data processing as research questions evolve. We call such explorations REDDIs: Rapidly Evolving Digitally-Derived Investigations. These differ both from a traditional hypothesis-testing approach where collection and processing are carefully designed in advance, as well as from data mining, where again collection is fixed in advance, but retrospective analyses are carried out post hoc. In contrast, REDDIs are investigations – as opposed to experiments – that may have no prior hypothesis, do not usually control or manipulate the circumstances being studied, and often involve changes in what is collected in the light of successive examinations of the data.

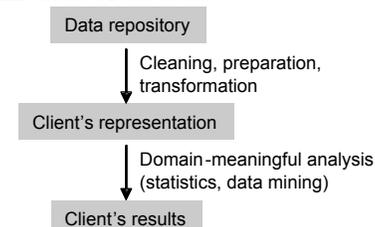


Figure 1. Stages in data preparation

A consequence of this new approach is that the data model most appropriate for each investigation or iteration thereof can not be designed in advance. Instead, the repository schema is optimised for rapid collection, and the transformation of data into a form meaningful and useful to the investigator must be done later. We thus have two major phases of processing: phase 1 transforms the raw data in the repository into the first domain-meaningful form – the *client's representation* – that the client can interpret and use, while phase 2 comprises the further transformations the client directs e.g. statistical or data mining analyses.

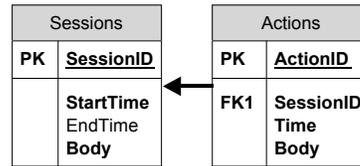
Phase 1 has been noted as a difficult area [Thomas 2003], particularly in the case of large datasets [Hilbert 2000]; this paper describes our experiences in developing phase 1 transformations.

2. Data Set

Several collectors are being developed for the GRUMPS system. The work reported in this paper concentrated on data generated by the User Action Recorder (UAR). This collector uses Windows system calls, gathering large volumes of low-level events representing actions such as window focus and keystrokes.

The UAR was used in the Computing Science Department at the University of Glasgow to collect user interface events from the first year programming laboratories, as students completed programming exercises in Ada, using the Adagide development environment. In spring 2003, 141 students consented to participate, yielding 4.7 million events over 2655 user sessions (periods between login and logout).

The repository is a SQL Server database with a simple schema, the main entities being the Actions (events) and Sessions. Much of the specific information on each action or session is stored as XML in a single attribute. The Actions table has 4.7M rows and takes about 2GB.



<p>gexecute.exe</p><w1>23</w1><wt>54</wt><wr>916</wr><wb>678</wb><ws>nor</ws> ...

Figure 2. (a) The repository schema
(b) Body XML for window focus event

3. The client's data

The first question to be investigated using the UAR data aimed to identify factors that impact on students' persistence with programming tasks. With traditional approaches, a model allowing the investigator to spot patterns in student perseverance would have been determined in advance and used during collection. A consequence of the GRUMPS aim of supporting flexible and evolving data uses is that the analysis leading to such a model is left until after collection, where it is guided by the client's requirements for their particular investigation.

The investigator (client) was familiar with studies using special purpose audit trails [Kennedy 2003]. Investigations had been carried out with an iterative process over three levels from general to specific and the client asked for a similar output in the form of reports. The first, most general report was a list of all sessions, with statistics such as pause times and number of compiles for each session. The second report shows the sequence of applications visited, plus a further subdivision of any activity in Ada, broken down into essentially the edit-compile-run cycle. The third level was the raw GRUMPS action data.

Action ID	What	Time Offset
1250975	UAR.exe	2
1250978	Explorer.EXE	8

1251807	Ada	1147
1251808	Activity	1154
1251810	No activity	1156
1251811	Activity	1161
1251836	Build	1177
1251838	Run	1187
1251840	leave gexecute	1187
1251841	Ada	1193

Figure 3. Client's representation - second report

3.1. Defining the intermediate representation

To convert the low-level UAR data into the higher-level reports described, it was necessary to perform significant transformations. Studying the requirements for the two reports outlined above, it became apparent that there was a subset of information that would have to be computed for both reports.

For example, both reports required knowledge of the *context* (or application) in which activities took place, and calculations of *durations*. Neither of these was directly available in the UAR data but had to be calculated using filtering and abstraction [Hilbert 2000].

We thus decided on an *intermediate representation* or recoding of the data containing most of the required information for the reports. This was a recoded stream where certain fields were the same as in the raw UAR data, whilst others were transformed to a form that would more easily allow us to generate the reports.

3.2. Generating the Intermediate Representation

We decided to carry out the data transformations in the database because of the large volumes of data and the flexibility that this offered. This prevented us from having to extract and transfer the full dataset, a slow process on 2GB of data. The use of indexes and parallel processing capacities make databases particularly appropriate for dealing

with large volumes of data. We used Microsoft's SQL Server, programming with SQL, and the extension of this language, Transact-SQL (T-SQL).

To generate the intermediate representation, data transformations were carried out at various levels, from transformations that are essentially data "cleaning" to higher level interpretations and abstractions such as context and durations.

One of the low-level transformations consisted in ensuring that each action was represented by exactly one line in the Actions table. For example, changes in window focus and use of the function keys F1-F12 generated two events (Windows systems calls) at collection, both stored in the repository.

Similarly, dates and times are stored in the database as recorded by the collector. In order for these to be meaningful for the client, they were transformed into a common intermediate form, in this case, dd/mm/yyyy.

At a slightly higher level, the XML data, storing much of the information about the events, needed to be readily available; for example, the name of the application, in the case of window focus changes. Aspects of the XML were expanded out, essentially by partial normalisation of the application name into a new table in the database. (Note that this would probably not have been necessary had we used a later version of SQL Server with native XML support.)

However, the most important and initially challenging transforms were to compute the context (application) and duration of each action. Had we chosen to use flat files and sequential processing, such transformations would simply have been a question of looping over the file, holding, for example, the last application name in a variable. With set-based, parallel processing, implicit sequence is lost and new techniques are required.

Taking the example of durations, the resources available suggested using a technique that calculated a partial cross product [Nielsen 2003]. Predictably, this scaled very poorly and we eventually opted for a solution based on a self equijoin. By joining the table to itself "one line down" and selecting the relevant columns,

we were able to obtain start and end times on the same line; subtracting one from the other gave durations. A T-SQL procedure was written that included code to account for the last ActionID.

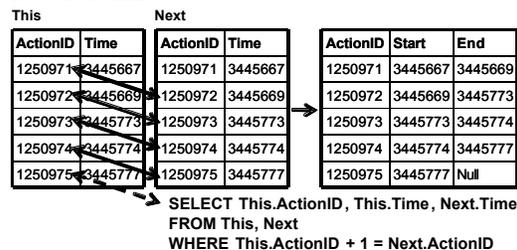


Figure 4. Calculating end times

The above method obviously relies on ActionIDs being contiguous. With respect to context, a similar self-join technique was used to generate entry and exit times to an application. However, this calculation concerned a subset of the actions – those that represented a change of application – and as such the ActionIDs were no longer contiguous. We found that the time taken to copy this subset into a temporary table with an automatically generated, contiguous primary key was negligible; this table could then be joined with itself as required. Further manipulations allowed the current application to be calculated for each action.

The above techniques were developed in T-SQL for SQL Server, to run in an acceptable time on the large dataset of 4.7 million actions. Because of the volumes of data that we were dealing with, indexing was important, and allowed us to reduce the time taken to generate the intermediate representation from 12 hours to 9 minutes.

Clearly, other solutions are possible; the above is given as an illustration. Golab and Ozsü (2003) discuss stream management more fully.

3.3. Generating the Client's Representation

The intermediate representation proved to be extremely useful for generating both of the reports described above. The first report, a summary of the activities of each session, uses

a case statement and a selection of built-in aggregate functions.

In order to generate the second report shown in Figure 3, a case statement generated the report according to a state table defining the type of high-level action (activity, compile, run, etc.). Furthermore, some of the techniques developed earlier were re-used. For example, the duration technique was used for higher level groups of actions such as compiling and editing.

4. Other Investigations

4.1. Typing and setup times

Following the success in generating the reports outlined above, the second author worked on his own research questions concerning typing speeds and setup (chunking) times. The idea was to establish whether there was a link between typing speeds in Ada and student success on the course.

The client's data for this investigation was clearly different, but it was found that the intermediate representation was a useful starting point for transformations. These investigations required information on individual keystrokes, and the intermediate representation was refined to take account of this.

Similarly, these investigations re-used many of the techniques developed during the creation of the first two reports. For example, the durations technique was used for the inter-key time, but as an onset (time before) rather than offset (time after). Furthermore, when these queries were established for typing activity in general, the techniques were partially applicable for some lexical analysis, to give Ada reserved words, for example. In turn this capability gave rise to the opportunity to recreate commands issued as keyboard shortcuts, such as CTRL-ALT-Down, which gives the next warning in the compiler listing when using the Adagide environment.

4.2. Further investigations

As anticipated, we are beginning to find that the UAR and repository have more general applicability than the two investigations

mentioned above, particularly for researchers who want to see what users do in the long term or across applications.

For instance, indicators of flow [Draper 1999] are being investigated as users perform tasks, with a thinkaloud screen where every 15 minutes they are prompted to type what they are thinking [Karahasanovic 2004]. Analyses of the UAR and thinkaloud data aim to determine whether flow can be detected by looking at activity; both the rate of clicks and keystrokes, and the variation of these with time. This required a report similar to that shown in Figure 3.

The same researchers believed it would be useful to know how long a user spends in each document. A report was generated from a simple extension of the stored procedures, using the intermediate representation and building on an understanding of how to extract durations and context.

Another researcher suspects that data generated by the UAR could give information about whether male and female computer users organise windows into different screen layouts. It is expected that the work already carried out will form a strong basis for the required analysis, adapting the high level session report, with its calculation of statistics for each session but using other fields in the XML, such as window minimised/maximised/normal and coordinates.

5. Discussion

Although far from trivial, transforming the low-level raw data into domain-meaningful client's data, is nonetheless feasible. As we tackled each new transformation, we gained experience and were able to benefit from the strategies and knowledge we had developed previously.

Three partially conflicting objectives are noted in the data preparation stage: (1) addressing the client's needs, (2) re-usability and (3) speed of query execution. The latter is important at the development stage, and equally to avoid slowing the investigation process.

The first investigation required significant time and effort, and we note that this approach,

where data is collected without prior knowledge of its use and later transformed, can be seen as feasible only if multiple investigations are to be carried out, either on the same data set or very similar data.

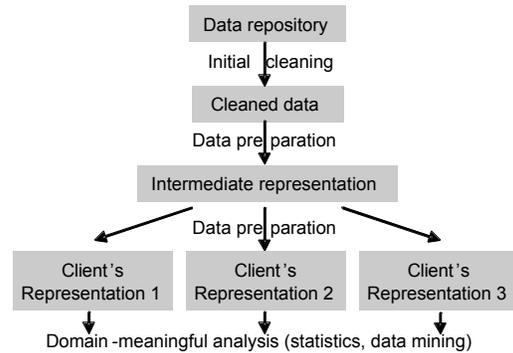


Figure 5. Stages in data preparation with intermediate representation.

It has become apparent that different investigations based on the similar often share concerns, such as the need for durations in several of the investigations outlined above. Because of this, we found it extremely useful to develop an intermediate representation from which several different clients' data could be generated. It seems that investigations can be grouped according to the data that they require, and either one or several intermediate representations developed. These can be considered as relatively flexible, being added to or adjusted as and when necessary.

For the same reasons, techniques developed for one investigation can often be re-used. It is therefore worth spending the time and effort necessary to optimise these, and developing them with a view to re-use.

Although certain of the optimisations carried out are DBMS-independent (for example, calculating a cross-product will always scale badly), it is worth noting that the way in which techniques are optimised and developed depends in part on the database in use. Different DBMS have different behaviour, use different versions of SQL and various extensions to this language, such as T-SQL, PLSQL, etc. A technique that is optimised for

one DBMS may therefore be inappropriate for another.

Using T-SQL, we found it extremely useful to integrate queries into stored procedures. This allowed us to develop techniques incrementally and execute them with a single command, storing the queries rather than the intermediate data created.

Gray et al. [Gray 2002] discuss the issue of creating multiple copies of data that lead to rapid *data inflation*, suggesting that it is often preferable to store the metadata – in our case, the stored procedures – that allow the generation of this data when necessary.

Although we have found it feasible to transform generic data for different investigations, it is nonetheless true that certain transformations are more difficult to achieve than others. This depends, to some extent, on the tool used for the transformations. Our choice to manipulate the data within the database was determined essentially by the volumes of data to be manipulated. With this choice came the difficulties associated with loss of implicit sequence (e.g. calculating durations and context). Other successful transformations have been carried out on smaller datasets, extracting the data and transforming using Java [Renaud, 2004].

It is therefore worth spending time designing the collector in such a way as to alleviate these difficulties as far as is practical, both in general, but possibly also in view of a specific transformation tool. When using the database for this stage, it was important to have sequence numbers for the actions as timestamps were not always unique and so could not be used to sort reliably. Because we were using Windows system calls, one action sometimes generated several events, which complicated transformations. In some situations it may be possible to avoid this, in a trade-off between processing during collection and after storage, although data is likely to be lost.

In addition, it is worth evaluating different tools for transforming the data, according to the specific constraints of the collected data and the investigation itself. The problems of

expanding the XML would have been alleviated by using a later version of SQL Server with native XML support. However, this issue is independent of others encountered, such as calculating durations.

6. Conclusion

We have described our experience of transforming low-level usage data into forms that can be used directly by multiple investigators. The raw data requires significant manipulation and we have shown that it is both possible and efficient to do much of this within the database itself. We suggest that investigations based on the same or similar raw data will inevitably share concerns and that research questions can be grouped according to these.

For each group of investigations, it is useful to develop an intermediate representation from which the data for several studies can be generated. Techniques and queries should be developed using stored procedures and need to be optimised for re-usability and execution times, both partially dependent on the DBMS used. This way of working significantly reduces effort required with each iteration or new investigation.

Acknowledgements

The GRUMPS team gratefully acknowledges the funding provided by the UK's EPSRC (GR/N381141). We also thank Malcolm Atkinson, Gregor Kennedy, Murray Crease, Karen Renaud and the Level 1 Computing Science students at the University of Glasgow.

References

- Draper, S. (1999) *Analysing fun as a candidate software requirement* Personal Technology, vol.3 pp.117-122
- Golab, L. & Ozsu, M.T. (2003) *Issues in data stream management*. SIGMOD Record, 32 (22), 5-14, June.
- Gray, J., Szalay, A. S., Thakar, A. R., Stoughton, C., vanderBerg, J. (2002) *Online Scientific Data Curation, Publication, and Archiving*. MSR-TR-2002-74: <http://arxiv.org/abs/cs.DL/0208012>
- Gray, P., Mcleod, I., Draper, S., Crease, M., Thomas, R. (2004) *A Distributed Usage Monitoring*

System. In Proc CADUI (Computer Aided Design of User Interfaces). Kluwer, 2004. pp. 121-132.

GRUMPS (2001) The GRUMPS Research Project. [Online]. <http://grumps.dcs.gla.ac.uk>

Hilbert, D. M. & Redmiles, D. F. (2000) Extracting Usability Information from User Interface Events ACM Computing Surveys (CSUR) archive 32(4), 384 - 421.

Karahasanovic, A., Fjuk, A., Sjøberg, D., Thomas, R. *A Controlled Experiment to Evaluate the Reactivity and Usefulness of Data Collection with the Think-Aloud Tool versus Classical Think-Aloud Method*. IRMA 2004 Conference, accepted.

Kennedy, G.E. & Judd, T.S. (2003) *Iterative analysis and interpretation of audit trail data*. In Crisp, G. & Thiele D. (eds) Proc ASCILITE Conference, Adelaide, Australia, 1, pp 273-282, December 7-10, 2003.

Nielsen, P. *Microsoft SQL Server 2000 Bible* (2003), Wiley Publishing, Inc., New York. P260-264.

Renaud, K., Gray, P. (2004) *Making sense of low-level usage data to understand user activities*, TR-2004-159, Department of Computing Science, University of Glasgow.

Thomas, R., Kennedy, G., Draper, S., Mancy, R., Crease, M., Evans, H., Gray, P. (2003) *Generic usage monitoring of programming students*. In Crisp, G. & Thiele D. (eds) Proc ASCILITE Conference, Adelaide, Australia, 1, pp 715-719, December 7-10, 2003.