Historizing of Application Objects and Relations

– Versioning Tools Compared

Dr. Oskar v. Dungern
2014-10-22

Keeping the History of Application Objects and Relations

When conceiving a system to support objects and relations *with their revisions*, the question arises how to implement the versioning, more precisely the historizing of data.

- First, file versioning tools known from software development come into mind. A given state of data is kept in a ReqIF file (which is a text file in a special XML format) and can thus be easily submitted to a file versioning tool such as Subversion or MS Sharepoint. Thus, a state of data (similar to a snapshot, baseline or a backup) is stored from time to time. Changes between baselines are not traceable and it needs quite some effort to extract a list of changes to an object, as all baselines must be loaded and searched. These limitations are fully acceptable for many simple applications.
- There are also software libraries supporting data historizing. The solution's data model keeps the actual data and for every update an entry is made in a 'change log'. Here, every change to an object is traceable and it is easy to extract a history of changes from the change log. However, it is virtually impossible to recreate the relations that have been in effect at a given point in time.
- For the ReqIF Server we have decided to weave the revisions into the data model. Every transaction creates a new revision of one or more objects and/or relations. Consequently, the full state of data of any point in time is readily accessible. No baselining is needed. The complexity is hidden in the server and a web-app may or may not consider the revisions.

Keeping the History of Application Objects and Relations using …

	File Versioning (Subversion etc.)	ReqIF Server
Concept	• Very efficient for storing a „baseline“, „snapshot“ or „backup“, i.e. the state of application data at a given point in time contained in one or more files.	• Stores revisions of every single object and relation in a database. • Revisions are known to the data model; they are not stored in a separate ‚change trail‘ or ‚change history‘.
Get historic state of data	• Reload a file revision (baseline) to access a historic state of data. • An application cannot recreate a state of data between baselines (unless it keeps its own revisions).	• Quickly get the state of data of any point in time without ever storing or reloading a baseline.
Get list of changes	• Must first inspect all stored baselines to deliver a list of object revisions.	• Easily and quickly obtain a list of object revisions right from the database.

Keeping the History of Application Objects and Relations using ...

	File Versioning (Subversion etc.)	ReqIF Server
Purpose	Versioning of text files. Focus on *how* a file is different from its previous revision.	Versioning of objects and relations. Focus on *dependency management* between revisions of objects.
Concept	Work with the *files* themselves, store the text differences.	Work with a *model*, where objects may • either be purely conceptual • or represent real files.
	Manage file configurations (file sets) and their versions (‚tags').	Objects may reference • any web-accessible file • or imported file .. ready for delivery.
	Based on a static data model aimed at software development.	Based on a dynamic object-relational data model (can be changed at runtime) for a wide range of applications.

Keeping the History of Application Objects and Relations using …

	File Versioning (Subversion etc.)	ReqIF Server
Revision	Store the delta of a file.	Create new revision of the representing object.
Tag	Remember a certain set of file revisions … and give it a name.	Remember a transaction number which exactly defines the complete state at that moment … and give it a name.
Branch	Copy a set of file revisions	Copy objects with all their relations
Merge	Compare content of files … and let the user choose whether a difference in text is carried forward or not.	Compare objects with all their attributes and relations … and let the user choose which ones are carried forward.
		Please note: • A client ‚web-app‘ is needed to perform these operations, which is not part of the ReqIF Server. • The ReqIF Server provides the base services to read/update objects as well as relations.

Keeping the History of Application Objects and Relations using ...

	File Versioning (Subversion etc.)	ReqIF Server
Typing	• Untyped (free) content	• Objects and relations have types defining their attributes. • Attributes have a defined data type allowing for input validation.
Relations	• No references/relations between files.	• Two relationship types with different versioning strategies: ‚fixed‘ or ‚floating‘ pointers to object revisions.
Navigation	• Files grouped in hierarchical folders, no navigation support.	• Navigate/query using the relations, e.g. „Get all objects being related to XY“.
Access	• Get a file revision as a whole • Get a file set of a specified tag	• Web-services to Create, Read, Update and Delete any object and relation.
Permissions	• ‚Read‘ or ‚Write‘ permission for a whole file.	• Roles with fine grained permissions at attribute level, i.e. it can be decided per attribute, whether a given role can read (=view) or update.

ReqIF Server

Is this interesting for you?

Any questions?

Contact
Dr.-Ing. Oskar v. Dungern
+49 173 670 9958
od@enso-managers.com

Information
www.reqf.de