Implementation of Object Life Styles with noSQL databases, using the example of CouchDB

Amin ABDALLA[1] und Paul WEISER[2]

Institut für Geoinformation und Kartographie · Technische Universität Wien ·
Gusshausstrasse 27-29 1040 Wien
[1] E-Mail: abdalla@geoinfo.tuwien.ac.at
[2] E-Mail: weiser@geoinfo.tuwien.ac.at

1 Introduction

1.1 Change as fundamental concept

If there is one unifying aspect describing the world we live in than it is certainly change. Heraclitus of Ephesus (520BC - 420BC) was probably the first to realize that „Change alone is unchanging“, meaning that nothing is ever static and everything changes constantly. Processes can be understood as the changing of a state and come in many forms. Some are barley noticeable affecting large regions of space (e.g., geological processes), others happen quickly and are locally confined (e.g. landslides).

The representation of continuous change is of particular interest to Physical Geography and can be modeled using Partial Differential Equations (HOFER & FRANK 2009). Human planning can be viewed as a discrete process over multiple levels of detail, involving physical objects (WEISER, SUBMITTED 2012). Social processes involve social objects and result in immediate changes, e.g., marriage (SEARLE 1995). Recent work by (TREIBLMAYR 2011) suggests the benefits of combining business and spatial processes. An important point is that business processes are understood as change per se (e.g. accounting) while today's GIS are mostly concerned with the representation of static phenomena.

1.2 Statement of the Problem

The integration of process models into Geographic Information Systems (GIS) has not made any considerable progress in the past 10 years. Today, GIS are almost exclusively used to retrieve information based on static phenomena. This can help to answer questions related to *Where* something is located but falls short in answering questions *When* something happened or will happen (FRANK, 1998).

Despite several methodologies for dynamic ontologies developed in fundamental research, these have seldom found its way into today's commercial GIS programs. In this work, we propose a schematic implementation of object life styles (see Section 2), using CouchDB, a noSQL database (see Section 3). Object life styles are a way to explicitly represent change in objects. We believe, we can treat geographic objects as documents and represent their state changes as document revisions.

1.3 Organization of research

Section 1 talks about change as a fundamental concept in the Earth Sciences and mentions the shortcomings of representations of change in today's commercial GIS. Section 2 lays the theoretical foundations for this paper and discusses different models of dynamic GIS. Section 3 introduces CouchDB, an example for a noSQL database. In Section 4, we provide a schematic implementation of object life styles using CouchDB and Haskell, a functional programming language. Conclusions and future research are proposed in section 5.

2. The "Evolution" of Dynamic GIS

2.1 Static GIS

Most of today's commercial GIS systems depict representations of static phenomena. Thus, information is always based on one state of the data, i.e., the most recent state the data was collected (WORBOYS AND DUCKHAM 2004). Questions that can be asked in such a static GIS include "Where is feature X located and what are its properties?" (See Table 1). Consequently, static GIS have no sense of time. The "geographic" part in GIS can even be questioned considering the definition of (PARKES AND THRIFT 1980) who define Geography as a discipline "where temporal and spatial aspects are inseparable", as opposed to Geometry, only concerned with space.

2.2 Snapshot GIS

Snapshots, i.e., linearly ordered sequences of data at discrete steps of time, are an attempt to represent dynamic phenomena in GIS. With such time series one can ask questions similar to „What is the relative decrease of wetlands in the US within the last 50 years" (See Table 1)? In today's commercial GIS the notion of time is almost always connected to snapshots. One problem with snapshots, however, is the implicit representation of change. Thus, change can only be inferred by comparing two states of data. The major drawback is that snapshots do not provide a method of explicitly storing change (Worboys and Duckham 2004). Consequently, by comparing two „identical" snapshots we can never be sure if there has really been no change (see Figure 1). We could, for example, assume that nothing has changed between the collection of dataset at t0 and dataset at t1, i.e., both buildings existed the entire time. Another, possible scenario is that between data collection the left building was destroyed and later rebuilt. With snapshots, we can never be sure what scenario is the correct one.

2.3 Snapshots, extended with Object Life Styles

A way to overcome the limitations of snapshots is to explicitly store change using object life styles. The term object life refers to all aspects while an object is in existence. Object life styles are a coherent set of operations describing the life of an object (FRANK, 2001). For example, during its existence an object can be *created* or *destroyed.*

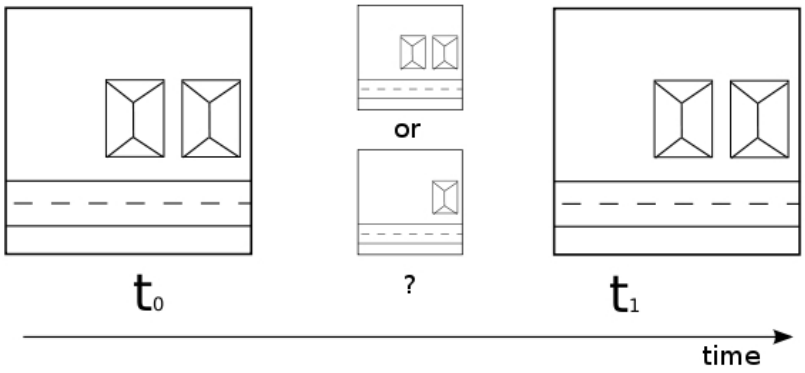

Fig. 1: Problems arising from an implicit representation of change

There are many more object life styles, either referring to one or multiple objects. For an overview see Figure 2. The notion of object life styles goes back to research by (AL-TAHA AND BARRERA, 1994) based on temporal constructs of identities and has subsequently been extended by (HORNSBY AND EGENHOFER, 2000). With a GIS that is capable of representing change in form of object life styles one could ask "When has feature X been destroyed"(See Table 1)? In this work we concentrate on the representation of change in objects using the before mentioned object-life styles. A schematic implementation is proposed using CouchDB (see Section 4).

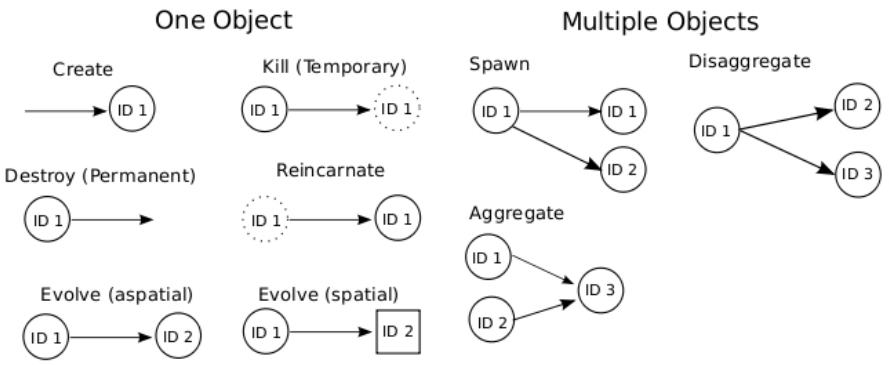

Fig. 2: Several possible object life styles (adapted from AL-TAHA AND BARRERA, 1994)

2.4 Dynamic GIS

To achieve the vision of a fully dynamic GIS the processes, events, and actions that change the state of objects need to be modeled. Unfortunately, dynamic models are far less advanced than their static counterparts. Notable contributions include (WORBOYS, 2005; GRENON AND SMITH 2004; GALTON AND WORBOYS 2004). In a dynamic GIS one could ask questions related to cause and effect of change in objects. See Table 1 for a juxtaposition of different types of GIS and possible queries.

Table 1: Different types of GIS and possible queries (WEISER, submitted 2012)

Type of GIS	**Possible queries**
Static GIS	Where is feature X located and what are its properties?
Snapshot GIS	What is the relative increase of Y in area X over a period of ten years?
Snapshot GIS, extended with Object Life Styles	When has feature X been destroyed?
Dynamic GIS	What has caused the change in feature X?

3. CouchDB

3.1 Principles

CouchDB is a schema-free and document-based database management system (DBMS). Schema-free means that there is no need define the structure of the database beforehand, e.g., using an Entity-Relationship Diagram. Consequently, there are no tables, columns, primary keys, foreign keys, joins, or relationships as they are in relational databases. In CouchDB data are stored in self-contained documents ("Everything is in one place"). Data can be aggregated and reported using a JavaScript engine and are represented using the JSON (JavaScript Object Notation) format (LENNON 2009).

It is important to mention that in a CouchDB database, documents can widely differ in syntax! Thus, not every document requires the same number of fields. In the example of "Joe the Plumber" (see Figure 3) we have fields for name, job description, telephone, fax, etc. but it causes no problem to have other documents in the same database that only have a field for telephone number but none for fax. In a relational database it is necessary to specify all fields that could possibly be needed, even if they later have the value *null*.

Also, in CouchDB it is not possible to just change one value of some field in the database. Changes always result in a completely new document with a new revision. The concept of revisions is deeply ingrained in CouchDB, i.e., every change results in a new revision. Thus, every change of the data stored can be kept and traced back, even deletion.

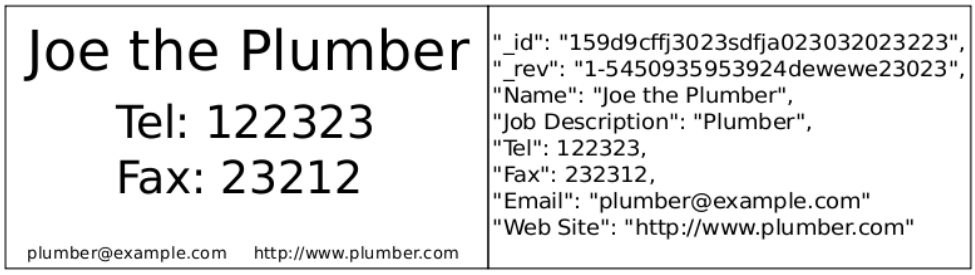

Fig. 3: Example Business Card and corresponding representation in CouchDB (JSON)

3.1 Communicating with CouchDB

CouchDB uses the HTTP REST API (FIELDING 2000) in order to create, update, retrieve, and delete documents. The corresponding HTTP REST commands are PUT, GET, and DELETE, respectively. The following request creates a new database "business_cards":

curl -X PUT http://127.0.0.1:5984/business_cards

We can also create a document representing a business card by issuing another command:

curl -X PUT http://127.0.0.1:5984/business_cards/JoeThePlumber -d \ '{"Name":"Joe the Plumber", "Tel":"122323"}'

The response by CouchDB looks like this:

{"ok":true,"id":"JoeThePlumber","rev":"1-4a12XXXX"}

We can retrieve a document by specifying the database, the document ID, and its revision number:

curl -X GET http://127.0.0.1:5984/business_cards/JoeThePlumber?rev=1-4a12XXXX

CouchDB returns the JSON object we defined before:

{"_id":"JoeThePlumber","_rev":"1-4a12d1cd7990f8ee946f5c1f53d20d26","Name":"Joe the Plumber","Tel":"122323"}

We can also make changes to the document, though we have to be aware that we cannot just change a value but have to provide a full JSON object as argument:

curl -X PUT http://127.0.0.1:5984/business_cards/JoeThePlumber -d \ '{"Name":"Joe the Plumber", "Tel":"122323", "Email":"plumber@example.com"}'

This results in a new revision of the document (rev=2-XXXX).

{"ok":true,"id":"JoeThePlumber","rev":"2-4a12XXXX"}

4. Implementation

To implement an object life style we chose the example of a house undergoing several states during its existence (Figure 4). In our fictional example the following storyline is assumed:

At some point in history a house was built. A misjudged artist spent a number of years living in it. Unfortunately, the artist died under mysterious circumstances in a fire that rendered the house unusable. Luckily, a private donor helped to rebuild the house and refurbish it into its prior state. As the years went by, the work produced by the unfortunate artist became honoured by society and internationally known. Due to this development, the city sees a fortune in turning the house into a museum. Therefore an entrance and shopping area is attached to it.

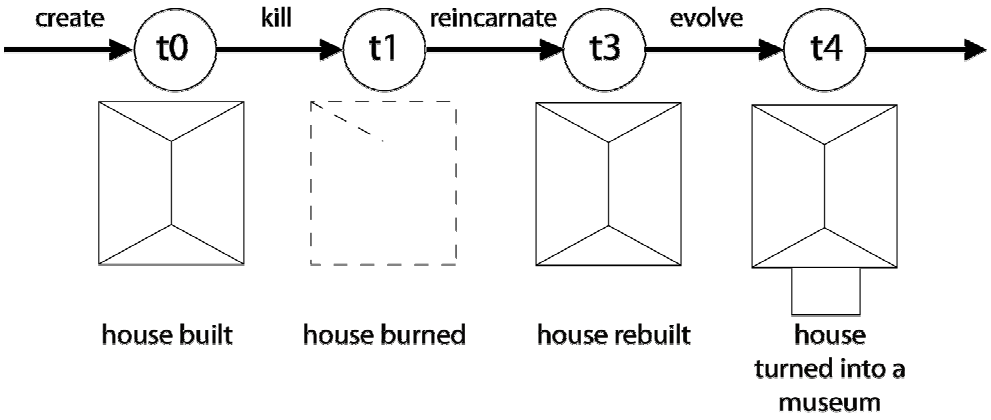

Fig. 4: Graphical Representation of the object life, describing the storyline.

To describe the storyline of the house in terms of object life styles, we can say that the object was *created, killed, reincarnated* and finally *evolved* into a different one.

4.1 Functions

Our assumption is that object life styles can be relatively easy represented in document based databases, such that each document stands for an object and each revision for the state of the object.

For our implementation we used Haskell, a functional programming language, and the CouchDB.0.10.0 package, as an interface to handle and manipulate the database.

To represent a simple geographic object we employed the following definition:

```
data Object = Object { name,time :: String,
                       geometry :: [String],
                       ancestor :: (Doc,Rev,String) }
```

An object therefore has a name, time, geometry and ancestor field attached to it. The information stored in the ancestor field is basically the ID and revision number that identifies it and the life-operation that produced the current object.

In the next step we translated the above mentioned life-operations into functions. See Figure 5 for a graphical representation.

(1) (create-operation): `createObject :: Object -> IO (Doc, Rev)`

The createObject function takes an object as parameter. It then passes the information to the database, along with the instruction to store it. CouchDB then returns the document representation of it, that is a tuple in the form of ID and revision number.

(2) (kill-operation): `killObject :: Doc -> Rev -> IO ()`

This function requires the ID of the document and its revision number, hence its identifier and state. The specified document is then deleted by the database. As described previously, deletion does not imply the total loss of it.

(3) (reincarnate-operation): `reinObject :: Doc -> Rev -> IO()`

In CouchDB, deletion means no more than creating another version of a document that is marked as "deleted". Hence we are able to access the previous versions of it. In terms of the reinObject-function we want to re-establish the state of the object before it was killed. The fact that this state is still available, enables us to copy its content and create a new document using the same ID. By passing an already existing ID, CouchDB automatically creates a new revision number for that document, what fits exactly our intention.

(4) (evolve-operation) : `evolveObject :: Doc -> Object -> IO(Doc,Rev)`

The last function we implemented, stands for the evolve-operation and is the first that demands additional programming effort. The evolve operation describes the creation of a new object originating from another. Talking in documents, it means that a new document with a new ID is stored, based on the contents of another. To achieve our aim of traceability, the ancestor field in the newly created object needs to be filled. The part CouchDB cannot resolve for us, is the change of the object itself, hence the content of the document. The change happening within the object properties requires manual adjustment. This can be achieved by the defining the following function:

```
createEvoBody :: Doc -> Rev -> Object -> Object
```

This function takes the document-ID, revision number of the evolving object and the properties of the evolved Object. It then stores the ID, revision number and operation in the ancestor field of the evolved object and returns it. This new Object (new ID) is then stored in a new document while the object it has emerged from (old ID) is killed.

4.2 Object-life

Finally we put the functions into chronological order to simulate the object life style:

```
objectLife = do
             (id,rev) <- createObject Object
              killObject id rev
              reinObject id rev
             (e_id,e_rev) <- evolveObject id evObject
              return()
```

We start off with creating the object and assign the output to a tuple representing the containing document. In the second step we kill the object by passing the identifier and version of the document that represents the object. To reincarnate we use the same ID and revision number, we employed to specify the object to kill. By passing the ID of the document and the new evolved object to the evolveObject-function we create a new document that represents the evolved object and delete the old one.

5. Conclusions & Future Research

In this paper we argued that object life styles help to store the history of an object more efficiently. Therefore, we came up with an example storyline that we described using object-life operations. Our assumption was that document based databases offer a convenient interface for modeling object life styles, due to the well suited paradigm they are based on.

By implementing our example, we concluded that document-based databases are an promising tool to represent object life styles. We particularly found that documents and their revision numbers form a suitable analogy to objects and their states. Thus, the automated versioning function of CouchDB simplifies the implementation tremendously. It appeared that most of the life-operations employed, were realized by simple HTTP-requests. There was almost no need for additional programming effort. The only exception was the “evolve” life-operation, since it required the adjustment of the object itself.

For the future, we plan to implement more life-operations using CouchDB, particularly the ones involving multiple objects, such as spawn or aggregate. Furthermore, we want to find out how relational databases handle object life styles. Thus, we intend to draw conclusions about the advantages and drawbacks of document based-databases, in contrast to traditional relational DBMS.

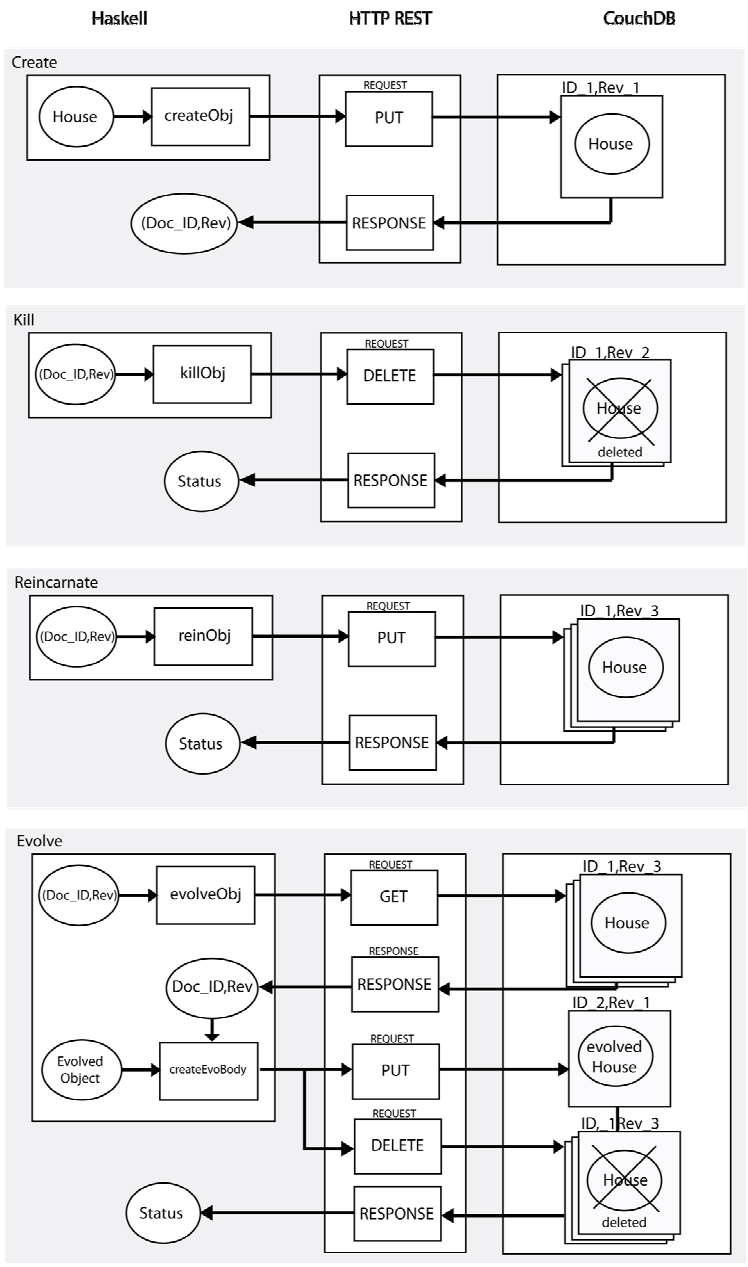

Fig. 5: The graphical abstraction of the functions illustrates the interaction that happens between the programming part, the HTTP REST API, and CouchDB. It becomes apparent that the only additional function required for the object life style is the "createEvoBody"-function to support the evolution-operation.

References

AL-TAHA K. & BARRERA R. (1994), Identities through time, In Proceedings of International Workshop on Requirements for Integrated Geographic Information Systems, New Orleans, Lousiana.

GALTON A., WORBOYS M., (2005), Processes and Events in Dynamic Geo-Networks, In Geospatial Semantics:45-59

GRENON, P., SMITH, B. (2004), SNAP and SPAN: Towards Dynamic Spatial Ontology, In Spatial Cognition & Computation: An Interdisciplinary Journal, Vol. 4, No. 1. (2004), pp. 69-104.

HOFER B. & FRANK A.U. (2009), Composing Models of Geographic Physical Processes, In Spatial Information Theory, Springer Berlin / Heidelberg, 5756, 421-435

HORNSBY K. & EGENHOFER M. (2000), Identity-Based Change: A foundation for spatio-temporal Knowledge Representation, In International Journal of Geographic Information Science, 14:3, 207-224

FIELDING R. T. (2000), Architectural styles and the design of network-based software architectures, PhD Thesis, University of California, Irvine, USA

FRANK A. U. (1998), GIS for Politics, Paper presented at the GIS Planet '98 Portugal, Lisbon, Portugal

FRANK A. U. (2001), Socio-Economic Units: Their Life and Motion, In Life and Motion of Socio-Economic Units, Taylor & Francis, London

LENNON J. (2009), Beginning CouchDB, APress

PARKES, D., THRIFT, N., (1980), Times, spaces, and places. New York, John Wiley and Sons.

SEARLE J. (1995), The Construction of Social Reality. Free Press, New York

TREIBLMAYR M. K. (2011) Geospatial enablement for enterprise information processing. PhD Thesis, Westfälische Wilhelms-Universität, Münster, Germany

WEISER P. (submitted 2012), Modeling discrete processes over multiple levels of detail using partial function application, Short Paper submitted to GI Zeitgeist 2012, Münster, Germany

WORBOYS M. AND DUCKHAM M. (2004), GIS: A Computing Perspective, 2nd edition, CRC Press