Applying Digital Storytelling to Information System Domains

Vinícius M. Gottin[1] Edirlei Soares de Lima[2] Antonio L. Furtado[1]

[1] PUC-Rio, Departamento de Informática, Brasil
[2]UERJ/IPRJ, Departamento de Modelagem Computacional, Brasil

Abstract

We developed a prototype tool, called **IDB**, to help investigate the characterization of information domains by the stories that emerge from their formal specification. We explore **IDB** as an application of *serious entertainment* where the functionalities of temporal *data*-bases are extended towards an ampler *story*-base scope.

Keywords: Storytelling, Conceptual Specification, Entity-Relationship Model, Relational Databases, Plan Generation, Simulation, Logic Programming, Plot-Mining.

Authors' contact:
vgottin@inf.puc-rio.br, edirlei@iprj.uerj.br, furtado@inf.puc-rio.br

1. Introduction

When elaborating a formal specification, designers always strive to make sure that all data integrity constraints and business rules, regulating the legitimate procedures and barring unauthorized conduct, are enforced. Yet it is hard to predict from a set of rules what situations can result.

This work proposes to view information system domains in terms of the stories emerging from their formal specification. As Schank and Morson [1995] assert, human intelligence is led in a very high degree by the stories in which the person has participated in some role, or has heard from other persons. We start from running conceptual level specifications [Furtado and Ciarlini 2000], expressed in a logic programming format, appropriate to conduct simulation experiments with the help of plan-generation and plan-recognition algorithms. To design and experiment a system, all the way from conceptual specification to implementation in a Database Management System (DBMS), we developed the **IDB** (**I**ntelligent **D**ata**b**ases) prototype tool, now fully operational. Figure 1 is an overview of the architecture.

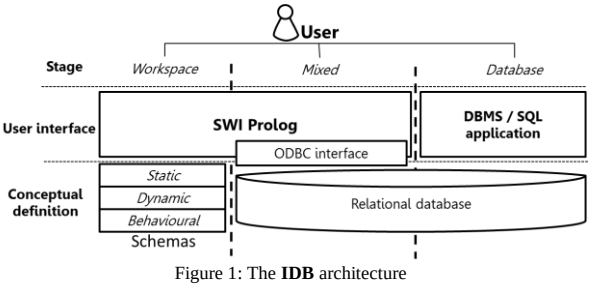

Figure 1: The **IDB** architecture

In this paper, we describe the usage of **IDB** system features towards the goal of transitioning from *data*-bases to *story*-bases as fundamental components of information systems. We claim that such stories are the best way to characterize a system operationally, i.e. showing what it can do as a consequence of the current specification – which should be revised if some desirable stories, corresponding to the users' justifiable expectations are still not deployed, or if transgressive stories, due to loopholes in the complex rule interactions, are detected.

The paper relies, as a running example, on a simplified academic database. Section 2 describes the conceptual specification discipline, which allows to model not only facts, but also events and agents. Section 3 describes the stepwise transition from workspace to DBMS environment. Section 4 explores the plot generation and dramatization features, how recurring patterns are handled by most specific generalization, and how past stories are recovered from an event-oriented LOG. Section 5 contains concluding remarks. A separate document concerning the full conceptual specification and the implementation in the DBMS environment is at http://www.inf.puc-rio.br/~vgottin/ex1_idb.pdf.

2. Three-schemata conceptual modelling

To specify an information system application, it is not enough to define the classes of *facts* that will eventually populate the underlying database. One should also specify in conceptual terms, i.e. in the language of the application domain, a fixed repertoire of *events*, whereby the state of the mini-world would change. And the pragmatic aspect, which has to do with how the *agents* involved would be motivated to reach their goals by bringing about the appropriate events, should also be considered (for a comprehensive discussion, cf. [Ciarlini et al. 2010]). Facts, events and agents are contemplated, respectively, in what we call *static*, *dynamic* and *behavioural* schemas. For brevity, only a few instances of the SWI-Prolog clauses are shown here, as well as in the next sections.

The ***static schema***, wherein *facts* are specified in the Entity-Relationship model [Batini et al. 1991], defines the entity classes (e.g. `student`) and their identifying attributes (e.g. `student_name`). Entities can have additional attributes (e.g. `credits`, for the `course` entity). Relationships associate entities (e.g. `takes` associates `student` and `course` entities; `graduated_in` associates `student` with `program`).

An instantiation of the schema, representing *facts* in clause format, expresses an *initial state*. E.g., `course('Art')` states the existence of a course entity with `course_name` "Art" as identifying attribute. A separate attribute clause, `credits('Art',2)`, further characterizes "Art" as a 2-credit course.

The ***dynamic schema*** deals with *events* able to change the state of the mini-world of the application domain. They are limited to a pre-defined repertoire of operations, specified by their pre-conditions and post-conditions (effects), following the STRIPS formalism [Fikes and Nilsson 1971]. In our example, one such operation is `change_cr`, that changes the value of attribute `credits`:

```
operation(change_cr(C,N1,N2)).
deleted(credits(C,N1),change_cr(C,N1,N2)).
added(credits(C,N2),change_cr(C,N1,N2)).
precond(change_cr(C,N1,N2),credits(C,N1)).
```

The **IDB** planning algorithm enforces a discipline that in most cases simplifies the definition of the operations. For example, it is not necessary to add a pre-condition to guarantee that `credits(C,N2)` does not already hold.

The planner interprets the pre-conditions both as tests for the applicability of an operation `Op` and, in case of failure, as sub-goals to be fulfilled by operations to be introduced before `Op` in the plan. This recursive treatment of pre-conditions as sub-goals constitutes the *backward chaining* strategy, on which many planning algorithms (including ours) are based. An especially powerful feature that extends the planner's ability to pursue goals or sub-goals involving numerical expressions is *constraint programming* [Rossi and Beek 2006], to which we had access through the SWI-Prolog clpfd library module.

The ***behavioural schema*** concerns the authorized *agents*, who are motivated to perform events in order to achieve goals induced by certain situations, as expressed in situation-objective (`sit_obj`) rules. As the planner is applied to these rules, alternative future stories are composed, as illustrated in section 4. In a separate work we considered an extension to the schema, which purports to model the agents' personality traits in terms of drives, attitudes and emotional profile [Barbosa et al. 2015].

3. From workspace to database environment

Once the schemas have been specified, and a (possibly empty) initial state has been provided in workspace memory, one can perform state transitions through the `execute` predicate. A call to `execute(enroll('Bea', 'Art'))` would have the double effect of enrolling Bea in the Art course and, since this would be her first enrollment, of registering her as student, with zero credits.

The `execute` predicate also works on plans, but another predicate, `goal_exec`, has the advantage of exhibiting the alternative plans that have been found to achieve the given goal expression, and allowing the user to choose the alternative to be executed.

At a second stage, one can still issue the commands in the Prolog environment, while actually handling an Oracle database, instead of main memory workspace. The first step towards this stage is to call the `gen_tab` predicate to generate a script (i.e. a ".sql" program) to translate into relational tables the entities, attributes and binary relationships defined in the static schema, and running the script in order to create and install the tables in the DBMS environment.

This is done by invoking our predicate `run_sqlplus_script`, which takes the name of the script as its only parameter. Tables are created for each of the specified entities and relationships, with columns for each of their attributes. Indeed, for any existing one-to-n relationship no table need be created, being enough to add a column to the table of the participating entity for which the other entity represents yet another (single-valued) attribute. In our example, however, all relationships are m-to-n, and adoptiing this optimization strategy would violate the normalization principle, famously imposed by relational model practice. So, `STUDENT(STUDENT_NAME, CREDITS_WON)` is created for the `student` entity, and `TAKES(STUDENT_NAME,COURSE_NAME)` for the `takes` relationship that relates students and courses.

In addition, there are two tables that are used by every **IDB** application, for data administration purposes: the `REF` table and the `LOG` table. The `REF` table stores the granted references, which are distinct positive integers that will serve as identifiers for the *IDB-transactions*. This special type of transactions controlled by **IDB** allows to characterize a potentially long process, which may be resumed in future sessions. The `LOG` table registers the execution of the operations, giving, for each execution, the reference of the transaction to which it belongs, the timestamp read from the system's clock, and the event (name and parameter list of the operation executed).

The next step is to produce a mechanism capable of updating the relational tables, which is done by the command `compile_ops`, whereby a *procedural* version of the operations is compiled from the previously introduced declarative specification. Their main components are predicates that have been implemented to execute, via ODBC, the basic SQL data manipulation commands. The pre-conditions are now checked by `select` calls, and the post-conditions (effects) take the form of `insert`, `delete` and `update` calls. The planner is still available to work upon the information now stored in the Oracle database.

The first two stages are intended for the specification tasks and for the performance of simulation runs, with the help of the plan-generator. To facilitate the transition to a third stage, a second compiler is provided to translate from the procedural version of the operations, produced by the first compiler, into Oracle *stored procedures*. The Oracle `create` statements to install in the database the compiled procedures are generated by the `gen_procs` predicate, whose output is recorded in a script file, to be transferred and processed by `run_sqlplus_script`, exactly as done to create the relational tables.

At this point, the database is ready for routine operational usage, employing some commercially available DBMS, such as Oracle, possibly together with some suitable host language. Even after reaching the final database stage, however, we find advisable to keep the logic programming specification, since it serves several practical purposes, including: training the prospective users; simulation and continuous testing; documentation; redesign; monitoring; plot mining [Furtado et al 2007], and all sorts of opportunities to employ Artificial Intelligence methods of analysis (see for example [Barbosa et al. 2007]).

4. Application domains as story genres

While plan generation opens the way to explore alternative future stories, the LOG of events that the **IDB** prototype maintains as a relational database table works as a repository of past stories, from which *typical plans* and sequence patterns resulting from actual usage can be extracted, to be profitably employed later for recognizing and predicting what can be expected from the current users' observed actions.The LOG – repository of past

stories – can be inspected at any time, either in the Prolog environment, by entering:

```
:- select_log(R,Ts,Ev).
```

or over the **Oracle Database XE** shell, by the command:

```
select ref,ts,event from log
order by ref,ts,event;
```

The LOG contains the answer to temporal queries, such as: how many total credits had Joe in July 15th 2011, and what event produced that value? To reply, one searches the LOG for an event prior to the indicated date that would be able to alter the value, and then makes sure that no event able to modify it has occurred between the two timestamps. Also, it is possible to revert to a past state, by determining the net effects of the events occurring since the system started until the given date.

By calling `show_plot`, one can visualize the contents of the LOG in a storyboard-like comic strip aspect, with playful images and colloquial template-driven natural language text. This plot-dramatization feature is provided by **Plot Viewer**, an additional module of **IDB**, implemented in C#. The compositing process uses the event description to create graphical illustrations according to the event parameters. Gender is established by inspecting extensive first name tables, which are updated by querying the user whenever a still unknown name comes up, so that male and female actors are drawn differently. More details on the generation of comic strips can be found in our previous work on interactive comics [Lima et al. 2013].

Figure 2 ilustrates `show_log` when applied to a selected part of the LOG that concentrates on "Bea", a model student with praiseworthy performance, once `involved('Bea',Story)` is indicated.

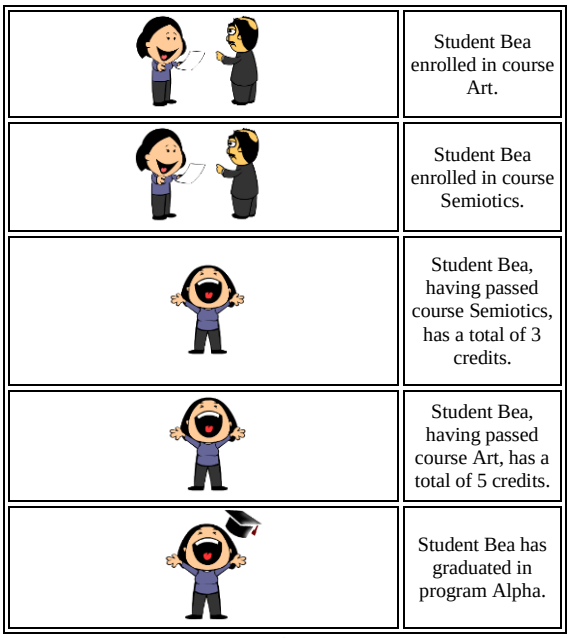

Figure 2: Bea's academic life in storyboard presentation

Besides visualizing the existing stories, **IDB** allows the hypothetical consideration of future events. For the goals supplied by the situation-objective rules, one may look for some *plan* (also amenable to storyboard display).

In our example, one of the rules states: the Chairman, when noting that a course was created two or more years ago and no student has passed it until now, would be inclined to remove the course. This rule involves an access to the LOG: are there courses initiated two or more years ago that no student has up to now been able to pass?

Suppose that this is the case with the Design course, and that the present Chairman wants to find what would happen if two students, say Ken and Laura, are still taking it. How can the course be cancelled? The simulation can be run either against the LOG or by placing these facts in the workspace. The Chairman will observe that one generated plan would cause Ken to drop the course, and Laura to transfer to another course that she has not yet taken, since `cancel` requires as precondition that there should be no student currently taking the course:

```
P = start=>drop(Ken, Design)=> transfer(Laura,
Design, Art)=>cancel(Design)
```

But other plans are also presented, as the user keeps entering the Prolog directive to search for alternatives. In one of these, the plan-generator responds in a strikingly different way in Laura's case, suggesting that Laura be automatically approved!

```
P = start=>drop(Ken, Design, Semiotics)=>
pass(Laura, Design, 0, 1)=>cancel(Design)
```

This comes much to the Chairman's surprise, as well as to ours, authors of the specification, who failed to recall – but the "system" would not – that *ceasing to take a course had been declared as one of the effects of the* `pass` *operation*!

Once this has been recognized as a possibility, it is up to the Chairman to decide whether or not it is acceptable. In the negative case, a written note might be sent to the application manager, demanding a change in the specification (to be pursued all the way down to the implementation).

Note, anyway, that the goals generated by the `sit_obj` rules are no more than *recommendations*, which the agents are not compelled to accept. For example, we have another rule that states that a student who dropped a course and is not currently taking any course ought to be interested in some course with a smaller number of credits (and therefore presumably easier). Suppose however that Zoe, who failed Art (2 credits) and is recommended by the planner to take Design (1 credit), ends up enrolling in Semiotics (3 credits, and by assumption more difficult).

Suppose further that a recurring *pattern* can be extracted from the LOG, revealing that the choices of several students quite often coincided with Bea's choices. To formulate a pattern, one computes the *most specific generalization* of the detected similar lists of events, keeping the constants that identically fill corresponding positions, and consistently introducing variables wherever different constants occur.

We may imagine that, reputed to be a model student, Bea's actions tend to be imitated by her colleagues. Successful completion of the courses involved would provide an even more influential pattern. At this point, we are perhaps in a position to risk an explanation for Zoe's peculiar conduct: having observed her choice of

Semiotics, which Bea took successfully (as registered in the LOG), we express the conjecture that this is no mere coincidence, provided that Zoe's enrollment, if in fact influenced by Bea's enrollment, did occur *after* hers. The pattern-recognition algorithm would then instantiate the pattern with the observation of Zoe's enrollment, in an extended *plot format* [Karlsson et al. 2009], indicating explicitly the partial-order constraints.

Lastly, there is, so to speak, a *meta-story* to consider, concerning how the system itself evolves during its lifetime, in response to the demands of dissatisfied users. The Chairman, in our example, might demand a mechanism for determining whether a student can reasonably be approved when a course that the student is taking is to be cancelled. A common sense extension to our oversimplified specification would be to add a `performance` attribute to the `takes` relationship, whose current value would be taken into consideration whenever the precondition of `pass` is tested. Also, recalling the questionable criteria that, in our example, a student like Zoe would use to choose a substitute course, we included in that precondition a call to predicate `eval_min`, whereby an online multiple-choice exam checks whether the student meets the minimum requirements for approval.

6. Concluding remarks

The distinctive features of our **IDB** project, namely conceptual modeling not only of facts but also of events and agents, the availability of a LOG to register past events, and of a plan generator to project alternative futures, enabled us to achieve an effective transition from *data*-bases to *story*-bases as the fundamental component of information systems. To implement the proposed architecture, logic programming, complemented by constraint programming, Oracle, C#, and R (that we are now trying for statistical analysis) proved to be adequate.

Our approach helps application administrators and the various classes of prospective users to grasp the functionality of the specified system, by being told in what kinds of stories they are invited to participate. And, with the intent to come closer to the seemingly contradictory ideal of *serious entertainment*, we equipped **IDB** with a storyboard facility for narrating the stories by images and template-driven natural language text.

From the viewpoint of the **IDB** machinery, much remains to be done, among other concerns to extend the LOG to include already scheduled future transactions (an AGENDA facility), and provide friendly user interfaces that may totally hide the clausal notation interactions displayed on the SWI-Prolog screen. Also it must be recognized that, until now, we have been working with very small examples. For scaling-up the use of **IDB** to real-life business applications an even greater effort should be invested, both by enhancing the implemented algorithms and by adopting a modular divide-and-conquer design strategy [Casanova et al. 1991, Graefe et al. 2014] to help reducing the size and complexity of the various tasks involved at each stage.

From the viewpoint of **IDB** utilization, two lines for future research seem particularly relevant to our group. The first is to adapt it for training purposes. Users would be exposed to the stories that emerge from the specification, with pauses between the successive "chapters", during which they would be called to interact by considering the current state of the mini-world, and making decisions to affect in what direction the story would branch in continuation. By transposing **IDB** to a client-server architecture, multiuser participation could be enabled, eventually permitting to add a stimulating *game* feature to training, with criteria to grade the participants, who would compete and/or collaborate to reach goals, subject to the limitations of scarce resources.

The other line, which can be termed *plot-mining* or *story-mining*, looks even more promising, as evidenced by research in the field of *process mining* [Aalst 2011]. We believe that having a time-stamped LOG to register transactions – composed of conceptually meaningful events – is a major asset towards a semantically and pragmatically richer approach to perform knowledge discovery over the processes that may occur in a given information system.

References

AALST, W. M. P. VAN DER, 2011. Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer-Verlag, Berlin.

BARBOSA, S. D. J., BREITMAN, K. K., FURTADO, A. L., CASANOVA, M. A., 2007. Similarity and analogy over application domains. In: *Proceedings of the Simpósio and Brasileiro Banco Dados*, João Pessoa.

BARBOSA, S. D. J. ET AL, 2015, Plot Generation with Character-Based Decisions. Computers in Entertainment, v. 12, pp. 1-21.

BATINI, C., CERI, S. AND NAVATHE, S., 1991. *Conceptual Design: an Entity-Relationship Approach*. Addison-Wesley.

CASANOVA, M.A. ET AL., 1991. A software tool for modular database design. ACM Transactions on Database Systems, vol. 16, no. 2, pp. 209-234.

CIARLINI, A.E.M. ET AL, 2010. Modeling Interactive Storytelling Genres as Application Domains. Journal of Intelligent Information Systems, v. 35, pp. 347-381.

FURTADO, A.L., CIARLINI, A.E.M., 2000. Generating Narratives from Plots using Schema Information. In: *Proceedings of the 5th International Workshop on Applications of Natural Language for Information Systems*. Springer-Verlag.

FURTADO, A.L., CASANOVA, M.A., BARBOSA, S.D.J., BREITMAN, K.K., 2007. *Plot mining as an aid to characterization and planning*. Technical Report MCC07, PUC-Rio.

FIKES, R. E. AND NILSSON, N. J., 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* , 2(3-4).

GRAEFE, G. ET AL., 2014. In-memory performance for big data. In: *Proceedings of the VLDB Endowment*, vol. 8, no 1.

KARLSSON, B. ET AL, 2009. A plot-manipulation algebra to support digital storytelling. In: *Proceedings of IFIP International Federation for Information Processing (ICEC)*.

LIMA, E.S., FEIJÓ, B., FURTADO, A.L., BARBOSA, S.D.J., POZZER, C.T., CIARLINI, A.E.M., 2013. Non-Branching Interactive Comics. In: *Proceedings of the 10th International Conference on Advances in Computer Entertainment Technology*, pp. 230-245.

ROSSI, F., AND BEEK, P. 2006. Handbook of Constraint Programming, Elsevier Science.

SCHANK, R. AND MORSON, G.S., 1995. Tell Me A Story. Northwestern University Press.