Semantic-based Distributed Version Control Model

Bijian Fan[1, a], Yi Zhuang[1, b]

[1] Department of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China

[a]email: fandy0409@sina.com, [b]email:zy16@nuaa.edu.cn.

Keywords: Version Control, Semantic, Configuration Management

Abstract. Aiming at drawbacks of the poor universality of traditional version control model, a semantic model which can organize unstructured data into structured data is discussed in this paper. Using semantic model can describe the change process of design object dynamically;then this paper further proposed a model called SDVCM (semantic-based distributed version control model) to realize distribute version control. The contrast experiment proved that SDVCM is obviously superior to the existing version model in time and space overhead.

Introduction

Distributed collaborative design platform which is widely used in the field of aircraft design, ship design and other industrial design can make different designers to carry out collaborative design at the same time in different places[1]. Distributed version control is a key technology in collaborative design platform, which not only needs to record the design process, but also provides technical support for the design of event management, control and traceability. The object of version control is the related documents and data in the process of collaborative design.

In the traditional centralized version control system such as SVN[2], all the versions are handled by a shared version center, so it is only need to manage the data objects that can meet the needs of the version management in collaborative design. However, the traditional version control model has the problems of poor generality, consistency and coordination, and it can't meet the requirements of distributed version control[3]. Distributed version control tool, like Git and Mercurial works in the way of point to point. Each copy of distributed version control tool contains all versions of the history and metadata[4]. Compared to centralized version control, distributed version control can reduce the cost of work, improve the efficiency of team cooperation, and support the development of the private sector, and quickly meet the needs of the business[5]. However, due to the differences in time zones, geographical and social culture, the distributed version control will inevitably improve the complexity and difficulty of the development[6].

On the basis of analyzing the advantages and disadvantages of the existing version control model, this paper proposed a model called SDVCM (semantic-based distributed version control model). The model describes the design object by using the modified schema and the version, which describes the design object cannot understood by designers as a structured data that is meaningful and related. On the basis of the schema and version, three kinds of relations (evolutionary, subordinate and dependency) are defined. Schemas, versions, and relationships can dynamically describe the whole evolution of the design object.

Semantic-based Distributed Version Control Model

In the collaborative design, the object of version control is the design information of the design object. Usually, the design information stored in computer is relevant and has special meaning, which can be only understood by design collaborative software or some professional designers. This paper introduces the concept of semantics. The application of semantics gives a practical meaning to the data, and the unstructured and unrelated data is organized into structured and standard data.

In this paper, a semantic model is used to represent the design object. The design object includes

the structure object, data object and constraint relationship between objects. Structure object records the logical structure information of the design object, including the schema and the version. Data object records the actual data of the design object, in the form of data table stored in the database. The schema describes structure, specifications and other standardized information of design object. In the semantic model, the schema can be divided into configuration schema (CS) and data schema (DS). Version is a snapshot of a design object at a time point in the development process. In the semantic model, the version is not only reflected in the design of the data, but also provides the logical structure between the design object. Corresponding to the schema, the version can be divided into the configuration version (CV) and the data version (DV). The specific definition of CS, DS, CV and DV will be given in the following. Figure 1 gives the proposed semantic model of the design object in the version control model.

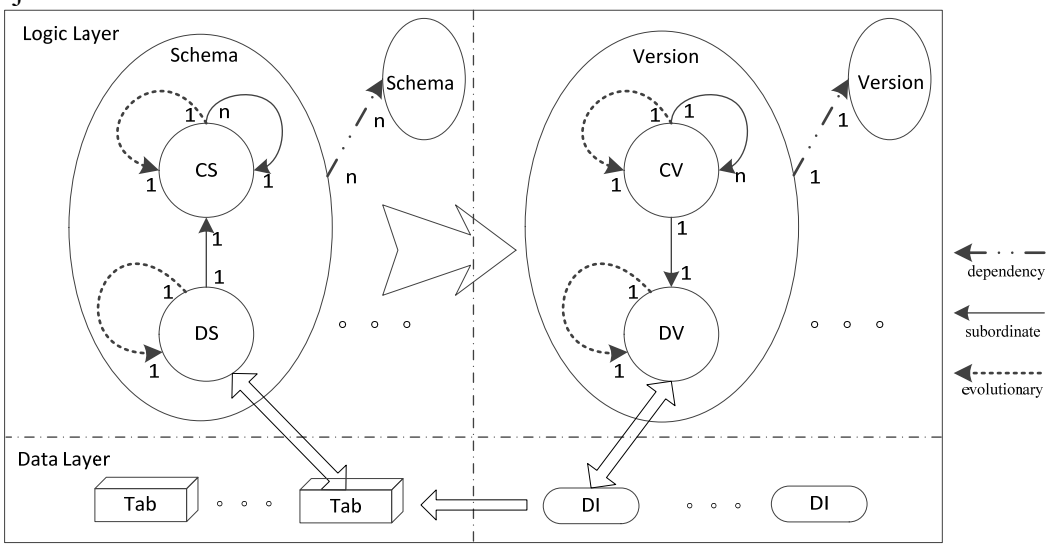


Fig.1. semantic model of the design object in the version control model

In Fig.1. the semantic model is divided into two layers: the logic layer and the data layer. The logic layer is composed of the structure object and its constraint relationship, which can depict the logical structure of the design object. The arrow in the logical layer represents the constraint relationship between the design object information: evolutionary, subordinate and dependency. These relationships exist between schemas and versions, but their meaning is not exactly the same, and the specific definition of the relationship will be given below; Data layer records the data object of the design object, which is stored in the database in the form of data table (Tab), and the data item (DI) is stored in the data table, which is on behalf of a version.

The relationship between design data also need to record. The evolutionary relationship describes the change of a configuration of the design object. The dependency relationship records the corresponding relationship between the configurations and between the configuration and the data. Dependency relationship represents the driver relationship between different design object information. The schema and the subordinate relationship statically describe the characteristics of a design object. The schema, the version, evolutionary relationship and dependency relationship can dynamically record the change of the design object. The above information can make the design object have the semantics, they are the management object of version control.

Semantic-based Distributed Version Control Model and Consistency Constraints

The semantic-based design object has the information of schema and version. In order to manage and store the information effectively and conveniently, a semantic-based distributed version control model is established, and corresponding consistency constraint rules are given.

Define 1 Semantic-based distributed version control model SDVCM is represented by multi-tuple, as follows:

$$SDVCM=(CS, DS, CV, DV, \Theta(s), \Psi(v), \Delta(v)) \quad (1)$$

DS, CV, DV, CS are the definition of semantic model of design object. As described above CS

represents configuration schema, DS represents data schema, CV represents configuration version, DV represents data version. Three consistency constraints are required to be satisfied in the version control process, respectively: $\Theta(s)$, $\Psi(v)$, $\Delta(v)$. $\Theta(s)$ represents evolutionary consistency. $\Psi(v)$ represents subordinate consistency. $\Delta(v)$ represents dependency consistency. Specific definitions are as follows:

Evolutionary consistency means operations of schema and version should meet the requirements for serialization. In SDVCM, the configuration version, data version and actual data should be used in sequential version marking method. And once a version is formed, the version information is only allowed to read operations, and the operation is not allowed to delete, so the operation of the version information will not affect the consistency of the whole version.

Define 2 Let WL(s) donate adding a write lock for schema s. $\Theta(s)$ represents schema s meet the evolutionary consistency. The necessary and sufficient condition for the operation of schema s satisfying the evolutionary consistency is to meet formula (2).

$$\Theta(s) \Leftrightarrow \mathrm{WL}_s(\{s\} \cup \bullet\bullet s) \tag{2}$$

Formula (2) indicates that when creating schema s, all ancestors of schema s should also be added to the write lock.

Subordinate consistency refers to each submitted version is fully consistent with the constraints of schema, and matches the full sub schema tree.

Define 3 Let $\Psi(v)$ donate version v meets the subordinate consistency. The necessary and sufficient condition for the operation of version v satisfying the subordinate consistency is to meet formula (3).

$$\Psi(cv) \Leftrightarrow (\forall v_i \in cv\bullet : \mathrm{E}(v_i) \vee \Psi(v_i)) \wedge (\forall s \in (\Gamma cv)\bullet : (\exists v_i \in cv : s = \Gamma v_i)) \tag{3}$$

Formula (3) requires that the configuration version satisfying the subordinate consistency requires its sub version exists or satisfied with the subordinate consistency, and the sub version of configuration version should be in one-to-one correspondence with the sub schema of corresponding configuration schema.

Dependent consistency of version requires that there is no ambiguity about the dependency of the version and its sub-version, that is, all the data in one version cannot be dependent by two different versions corresponding to a certain schema.

Define 4 Let $\Delta(v)$ donate version v meets the dependency consistency. The necessary and sufficient condition for the operation of version v satisfying the dependency consistency is to meet formula (4).

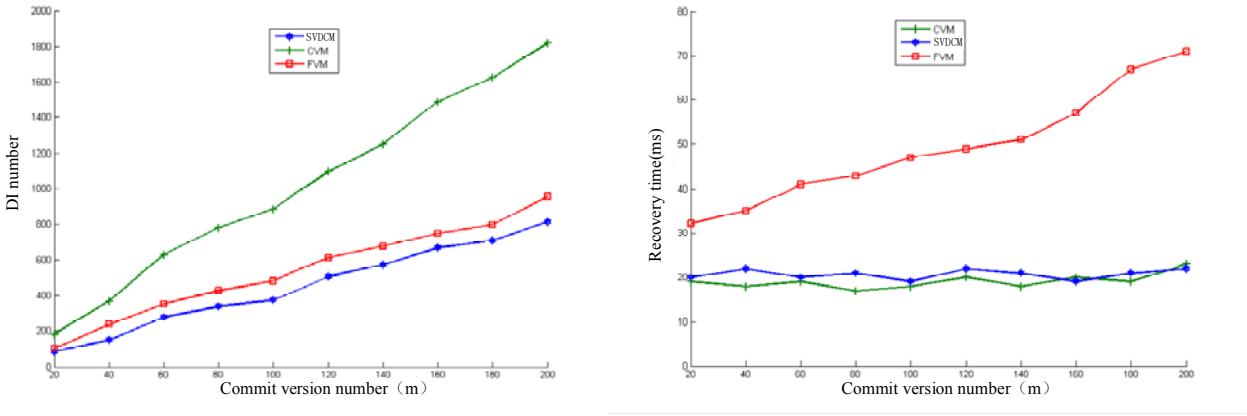
$$\Delta(v) \Leftrightarrow \forall v_1, v_2 \in \mapsto (\forall v' \in \{v\} \cup v\bullet\bullet) : v_1 = v_2 \tag{4}$$

Formula (4) indicates that the version satisfying the dependency consistency and all descendants should rely on the same version.

Test results

In order to test the proposed SDVCM's performance, we made two contrast experiments. In the experiments, the comparison objects of SDVCM are Complete Version Management method (CVM) and Forward Version Management method (FVM). We implemented these three models in the Oracle database and the aircraft collaborative design software, and test version operation of query, commit and merge through the design platform of aircraft.

The space utilization ratio of the three version control models was tested in experiment 1. In the experiment, we use the aircraft design software to submit data to the three version control models (implementing the version operation of commit and merge), and compare the occupied storage space after the m times submission of the version. The experiment was performed 10 times; each experiment was carried out with different values. The design object of the experiment contains 18 schemas, the value range is 20 to 200, and the step size is 20.



(a) space utilization ratio (b) query efficiency

Fig.2. Comparison experiment of version control models

Fig.2(a) gives the experimental results of the experiment. The abscissa is shown in the version number m of each experiment; the ordinate is the number of data item DI occupied database. From the graph, it can be seen that the space utilization of FVM and SDVCM has a greater advantage than that of CVM. In contrast, the SDVCM is slightly superior to the FVM, because FCM records the changes between the version and the previous version, while SDVCM stores the only index values that point to the previous version (or sub version).

Experiment 2 is based on Experiment 1. Experiment 2 tested the query efficiency of the three version control models by comparing the speed of recovering version graph of experimental 1. Fig.2(b) is the result of the query efficiency of experiment 2. The abscissa is shown in the version number m of each experiment; the ordinate is the time of recovering version graph, and the unit is milliseconds (ms). From the figure, we can see that SDVCM and CVM are better than FVM. This is because FVM needs to compare the difference between each version to recover the full version. The query efficiency of SDVCM and CVM is similar and CVM is slightly better than SDVCM. This is because every time of SDVCM's query needs to get the data of sub version from the index, but the time of the operation is very small, and it is totally acceptable.

Experimental results show that SDVCM is superior to the other version control model FVM and CVM in time and space. The retrieval efficiency and space utilization of SDVCM are well. In SDVCM, three kinds of relationships are introduced, which enrich the semantic description of the design object, and the constraint conditions can ensure the consistency of the process. Different from the previous version control model, which statically records the information of version, SDVCM dynamically stores the relevant information and structure of the whole design process through the dependence relationship.

Conclusion

On the basis of analyzing the advantages and disadvantages of the existing version control model, a semantic-based distributed version control model SDVCM is proposed. In SDVCM, the modified schema and the version are used to describe the design object which cannot be understood by designers. Further, the consistency constraint conditions are defined for the version control model: evolutionary consistency, subordinate consistency and dependency consistency.Experimental results show that SDVCM is superior to the other version control model FVM and CVM in time and space.

Acknowledgement

This work is supported by the National Natural Science Foundation of China (General Program) under Grant No. 61572253, the National Natural Science Foundation for Youth of China under Grant No. 61202351, the National Postdoctoral Fund under Grant No. 2011M500124, and the Fundamental Research of Nanjing University of Aeronautics and Astronautics under Grant No. NS2012133.

References

[1] Costa C, Murta L. Version Control in Distributed Software Development: a Systematic Mapping Study[C]//Global Software Engineering (ICGSE), 2013 IEEE 8th International Conference on. IEEE, 2013: 90-99.

[2] Kemp L E, Yamamoto M, Soldati‐Favre D. Subversion of host cellular functions by the apicomplexan parasites.[J]. Fems Microbiology Reviews, 2013, 37(4):607–631.

[3] O'Sullivan B. Making sense of revision-control systems[J]. Communications of the ACM, 2009, 52(9): 56-62.

[4] Knott M. Version Control with Git[J]. Beginning Xcode, 2014:379-341.

[5] BOHM M R, STONE R B, SIMPSON T W, et al. Introduction of a data schema to support a design repository [J]. Computer-Aided Design, 2008, 40(7): 801–811.

[6] Jean Carlo Binder. Global project management: communication, collaboration and management across borders[M]. Gower Publishing, Ltd., 2007.