Int J Flex Manuf Syst (2006) 17:227–250
DOI 10.1007/s10696-006-8820-8

Reverse simulation for collaborative commerce: A study of integrating object-oriented database technology with object-oriented simulator

Timon C. Du · Eldon Y. Li · Hsin Rau · Guan-Yuan Lian

Abstract Collaborative commerce has been used for communication, design, planning, information sharing, and information discovery in business-to-business (B2B) applications. The collaboration between buyers and sellers enhances product quality and customer satisfaction. However, most effort currently focuses on information sharing with customers and suppliers instead of joint product development or manufacturing. Moreover, traditional analytical methods have limited capability in solving problems. This study presents a framework for doing reverse simulation, where designers can reuse past experiments and change system parameters in manufacturing system for collaborative commerce. The framework integrates the object-oriented simulator and the object-oriented database. In this framework, the object-oriented database records the whole experiment scenarios and allows multiple planners with different expertise to involve concurrently and collaboratively. Then, simulations of advanced planning and scheduling in a product manufacturing environment that involves several planners working collaboratively are used for demonstration.

Keywords Collaborative commerce · Reverse simulation · Object-oriented database

T. C. Du (✉)
Department of Decision Sciences and Managerial Economics,
The Chinese University of Hong Kong, Hong Kong
e-mail: timon@cuhk.edu.hk

E. Y. Li
Department of Management Information Systems,
National Chengchi University, Taipei 11605, Taiwan

H. Rau· G.-Y. Lian
Department of Industrial Engineering,
Chung Yuan Christian University, Taiwan

1. Introduction

Collaborative commerce is a kind of business-to-business (B2B) application being used for communication, design, planning, information sharing, and information discovery (Turban, 2003; Li, 2005). These collaborative activities are usually performed between supply chain partners as well as within an organization. For example, the world largest semiconductor manufacturer, Taiwan Semiconductor Manufacturing Company, adopts "virtual fab" concepts by allowing customers to track all information and business transactions ranging from technology selection and collaborative design to post-sales services (www.tsmc.com/english/default.htm). Other examples include GE (www.geis.com/index.jsp) and Adaptec (www.adaptecconnect.com/). The adoption of collaborative commerce can improve communication between buyers and sellers. In addition, the interactive and collaborative relationship can enhance product quality and customer satisfaction. However, most collaborative effort focuses on sharing information with customers and suppliers. The degree of participation from either buyers or suppliers toward the manufacturer is limited since a production schedule is mostly controlled by the dominant manufacturer. Therefore, it can be considered a breakthrough if the degree of manufacturing planning can be more open to partners. For example, if both parties can perform manufacturing simulation together and propose a better production schedule, customer satisfaction can be further improved.

The complexity of manufacturing systems has increased as the degree of automation has increased. The traditional numerical analysis tools, e.g., operations research and queueing theory may have limited capabilities in solving some complex problems when uncertain factors are involved. Therefore, simulation can be another option to perform system analysis. Simulation has been used in various areas such as scheduling, inventory management, distribution, and forecasting. However, a primary drawback is that simulation experiments are time consuming and costly to develop and run. On the other hand, object-oriented technology is a software technology capable of modeling a one-to-one relationship between real-world objects and system objects. This technology has the advantages of program reusability and extensibility (Du and Wu, 2001). Therefore, if a simulation model could integrate with object-oriented technology, it could be reused and extended. Furthermore, a database system could be used to maintain a significant amount of data generated from simulation experiments. Most simulators use relational databases to store data. This data could be used for statistical analysis and parameter evaluation.

Given the aforementioned advantages of object-oriented technology, this study proposes an object database simulator that integrates an object simulator with an object database. This approach has a unique data structure and the whole simulation scenario can be recorded into a database. Moreover, a version management of an object-oriented database can maintain several versions of simulation experiments. The planners can return to old scenarios to branch new experiments or to involve other experts. In this case, the higher flexibility and better concurrency can be provided to the collaborative partners in uncertain environment.

A designer can set up several different checkpoints and a database can record an experiment at those checkpoints. The checkpoints should be set when (1) significant progress needs to be recorded or backtracked; (2) temporary agreements are reached; (3) acceptable results are found; (4) significant system loading has reached; and/or

(5) periodical system logging is preferred. A designer can return to old scenarios with checkpoints preset and change the parameters or precede new experiments if needed.

The remaining contents of this article are as follows. Section 2 reviews the literature. Section 3 presents the integrated classes of both an object-oriented simulator and an object-oriented database. Reverse simulation is discussed in Section 4. Section 5 demonstrates reverse simulation for advanced planning and scheduling in a collaborative manufacturing environment using an object-oriented database (ObjectStore), an object-oriented language (C++), and a modeling tool (UML).

2. Literature review

This section reviews simulation languages and database technologies. The integration of a database and a simulator is the focal point. Then, collaborative commerce is presented.

2.1. The simulation language

In the 1950s, simulation models were written in third-generation programming language such as Fortran or C (Banks, 1996). In the 1960s, simulation languages with statistical analysis functions were developed, such as GPSS, SIMSCRIPT (Law, 1991), GASP IV (Pritsker, 1986). Most simulation languages provide mechanisms for random number generator, model building, data collection, statistical analysis, and event scheduler. Normally, designers can use block diagrams, similar to flow charts, to assist the development of a simulation model.

In 1967 the concept of objects was introduced in SIMULA67 (Henderson-Sellers, 1991) allowing a simulation model to be composed of objects instead of procedure. This approach increased the speed of building a simulation model. Therefore, SIMULA67 could be considered as a pioneer of object-oriented programming.

In the 1980s, simulation languages had easier and faster functions to transfer between concepts and models. Examples include SLAM and SIMAN (Pegden, 1995). SLAM enhanced GASP IV by having not only an event scheduler, but also procedure-oriented and continuous simulation. SIMAN included modeling in material handling functions, such as conveyor and transporter.

In the 1990s, Windows-based simulators began to prevail. GPSS and SIMAN were modified into ProModel and Arena with graphic-user-interfaces. The interfaces allowed designers to create simulation model in drag-and-drop icons without spending too much time on writing programs.

Recently, object-oriented simulators have appeared. Object-C was used by (McGregor, 1994) to develop a Windows-based, object-oriented simulator. Another popular object-oriented simulator, Simple++, was designed by AESOP GmbH using C++. Simple++ is a simulator with model reuse capability. Mize (1992) compared the differences between conventional simulators and an object-oriented simulator and showed the advantages of an object-oriented simulator. Moreover, object-oriented modeling and simulation are also used for reactive system development (Barcio, 1997) and process planning and production scheduling (Zhang, 1999).

The terminology of reverse simulation was presented by McGreevy (1988). At that time, the reverse simulation used Monte Carlo simulation for chemical, electrochemical, and physical energy engineering. It mainly focused on the observation of the reverse process of materials heat treatment. The same terminology, reverse simulation, can also be found in finding stable system designs (Wild, 1994), where the reverse simulation stands for implementing simulation reversely. That is, the simulation determines outputs of the performance measures first, and then uses the designated performance outputs to find the variable values satisfying the performance measures. It is worth noting that reverse simulation in this study is different from the two earlier approaches. In this study, reverse simulation means that the designer can return to any checkpoint in the simulation process since multiple versions of simulation experiments are maintained by an object-oriented database.

2.2. Object-oriented database

The database system manages data for multiple users. The most implemented database system is the relational data model. The relational database has a simple and uniform data structure that provides advantages of solid theoretical foundation. However, the relational database has the shortcomings of (1) incapability in supporting simple data types, (2) decomposing real-world objects into instance pieces and recomposing them again when needed, (3) maintaining relationships using functions (e.g., triggers and constraints), and (4) having low extensibility after completing the schema design (Du, 1997).

The object-oriented database can be considered as an integration of a database system and the object-oriented technology (Khoshafian, 1993). The object-oriented technology includes the abstract data type, inheritance, polymorphism, and object identity, while the database functions are persistence, concurrent control, recovery, query, integrity, and security (Elmasri, 1994; Kim, 1991). Moreover, because of the integration, the object-oriented database has the functions of version management, schema evolution, and long transaction management (Bertino, 19991; Elmasri, 1994).

2.3. The integration between simulation and database

Current approaches to storing objects on a database are the object relational database and the object-oriented database. The examples of object relational databases are Oracle 8 and Informix. This approach uses object technology to encapsulate complex data, but the internal data structure is still relational. Therefore, the encapsulated data cannot be queried. The object-oriented databases include O2, GemStone, ONTOS, ORION, Iris, POSTGRES, Versant, and ObjectStore. Regarding the simulators, they can be distinguished into procedure simulators and object-oriented simulators. Currently, three combinations are being used (Mize, 1992; Eum and Minoura, 1996):

(1) Simulator without database. This approach does not have interfaces to store the simulation data into the database. Normally, data are maintained in data files. A designer needs to transfer or input data between simulators and data files. It is a time consuming process, and the designer needs to know the location and format of the data files very well.

Table 1 The comparison of different strategies in integrating simulators and database systems

	Simulator without database	Procedure simulator with RDB	Database language with RDB	OO simulator with RDB	OO simulator with OODB
Model construction	difficult	difficult	very difficult	easier	easier
Model reuse	none	none	none	yes	yes
Model expansion	difficult	difficult	very difficult	easier	easier
Difficulty of integration	very difficult	difficult	median	median	median
Format transform	many times	need interfaces	no interfaces	simulator-defined interfaces	none
Experiment reuse	none	very difficult	difficult	very difficult	easier

(2) Procedure simulator with relational database. This approach stores the data generated in the process of simulation experiments into a relational database. If the simulator is written in a format different from the database, a common interface, i.e. ODBC, is used for transformations. Some simulators provide designated interfaces to transfer data between the simulator and database.
(3) Object-oriented simulator with relational database. This approach can have the advantages of the object-oriented simulator, i.e. object reuse and flexible expansion, and the relational database, i.e., uniform data structure and data management functions. However, the difference of data structure between these two systems requires format transformation back and forth. In addition, the concept of encapsulation in the object-oriented technology conflicts with the data management of the relational database. If this approach uses the object relational database instead of the relational database, the limitation remains the same as that of the relational database . That is, the encapsulated data cannot be accessed other than through designated interfaces, and the objects need to be decomposed into table format.

Considering the drawbacks of the three current approaches, this study combines the object-oriented simulator and object-oriented database by integrating a portion of simulation classes with the object-oriented database. This approach has the common data structure for both simulator and database, and has the advantages of both. The simulator retrieves models from the database without format transforming, and the simulation experiments are recorded by the database. The old scenario can be repeated, using version management. Table 1 shows the comparison of different integration strategies.

2.4. Collaborative commerce

As it has been well known, B2B is different to business-to-consumer (B2C) in the way that both parties are business organizations. Normally, the organizations have formal contract and maintain long-term relationship. The relationship sometimes further extends into a supply-chain relationship. Several entities are involved in the supply-chain relationship, such as sellers, buyers, intermediaries and deliverers. Browne et al. (1995) pointed out that manufacturing systems have faced various sources of stresses and must link through the supplier chain and customer chain. There are various collaborative

formats. For examples, the long-term enterprise relationships across the value chain is considered as extended enterprise (Jagdev, 1998; Childe, 1998) while the virtual enterprise suggests a relatively short-term relationships among companies or departments within a company (Browne, 1999). The main objective of this relationship is to share information with partners. The information sharing can be done through Intranet (Gupta, 1998), Extranet (Turban, 2003), and workflow (Kumar, 2002). Through the information sharing, organizations expect to cut cycle time and cost, reduce administrative errors, increase productivity, and eventually boost revenue. It is clear that this relationship encourages collaboration. The collaboration triggers a new focal point, called collaborative commerce (Li, 2005; Turban, 2003) or collaborative business (Deitel, 2001). The activities of collaborative commerce are varied, ranging from joint design to forecasting. For example, Wal-Mart joins with P&G to form a collaborative forecasting and replenishment (Chopra, 2001) and Zacharia designs a collaborative reputation mechanism to predict reliabilities (Zacharia, 2000). Other studies include content management (Gupta, 2001), collaborative design (Huang, 2002), collaborative engineering (Contero, 2002), collaborative decision-making (Raghu, 2001), and collaborative forecasting (Aviv, 2001).

There are many collaborative models can be adopted in the collaborative commerce. For example, sales force automation (SFA), continuous replenishment programs (CRP), and vendor-managed inventory (VMI) that allows the suppliers to manage the inventory for the buyers (Chopra, 2001). Advanced planning and scheduling develops the detailed production schedules on what, where, when, how and who to make products by considering materials availability, plant capacity, and other business objectives (Kuroda, 2002). Moreover, collaborative planning, forecasting, replenishment (CPFR) model further integrates production, purchasing planning, demand forecasting, and inventory replenishment (Fliedner, 2003). Although the collaboration can happen in any stages of business transactions, the proposed reverse simulation for collaborative commerce focuses on the collaborative planning of manufacturing process. In the design, both parties from collaborating companies can interact with each other to generate a mutually agreed plan. The plan can produce the most satisfactory profits while balancing the manufacturing capacity. This provides the possibility of building profound relationship among supply chain partners.

3. An integrated architecture of object database simulator

The object-oriented simulation language develops models in the structure of classes, which generate program objects rapidly. Similar to the object-oriented simulator, the object-oriented database maintains data and applications in classes. Classes are the common ground of both object-oriented simulator and object-oriented database. Therefore, a plausible approach is to build an integrated architecture that uses object-oriented simulator to proceed simulation operation and an object-oriented database to maintain both data and experiments. Figure 1 presents the architecture. The integrated class library stands between the object-oriented simulator and the object-oriented database. The simulator generates program objects from the integrated class library, and the database system creates database schemas based on the class library. Other

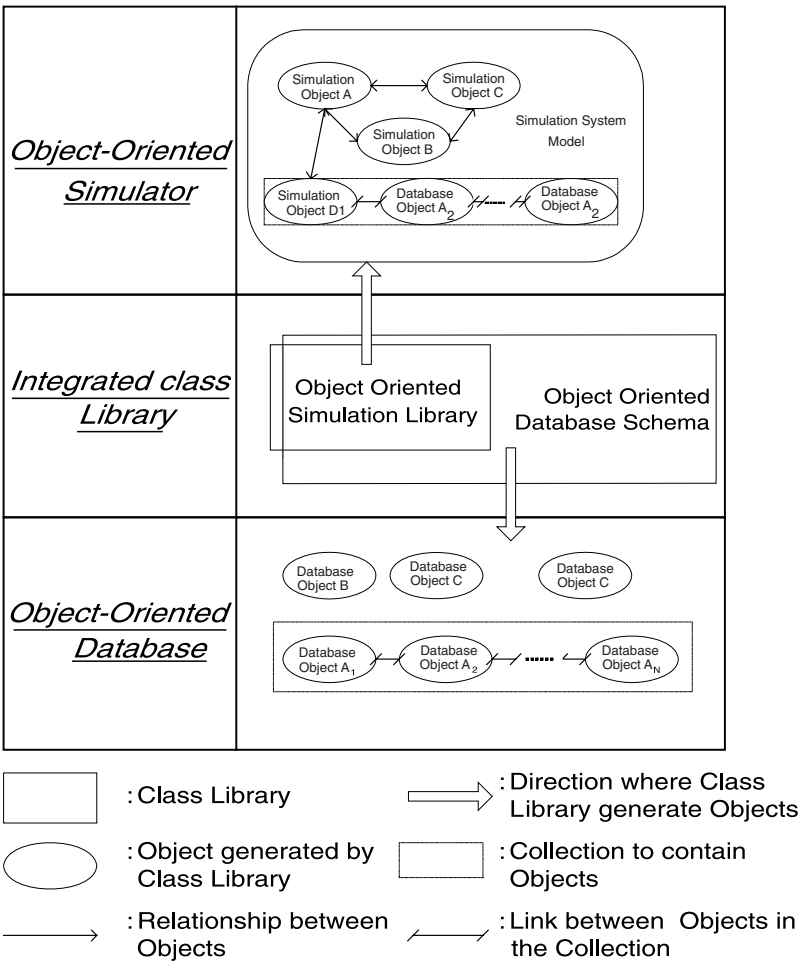

Fig. 1 The architecture of an object database simulator

than the schemas for simulation purpose, the object-oriented database also maintains external data, which are discussed in Section 3.1.

3.1. Class library of object database simulator

In this study, the class library of an object database simulator includes three levels of classes: (1) object-oriented simulator, (2) object-oriented database, and (3) integrated class library. The object-oriented simulator has several fundamental functions such as clocks of simulation process, random number generators, and subject components in order to create simulation environments. These functions can be grouped into two packages: *Simulation_Control* and *Simulation_Component*. A package is an UML terminology that groups similar classes for clear identification. The *Simulation_Control* package has simulation clocks and distribution generators. Since these functions in-

teract with other class modules at all times, they should defined as global variables. Two classes are defined in this package: *Simulation_Event* and *Random_Variable*. The other package, *Simulation_Component,* contains classes of subject components such as parts, buffers, operation centers, machines, etc. Two basic classes are provided: *Active* and *End*. If simulation objects are related to the time factor, e.g., machines, parts, etc., they are inherited from the *Active* class. On the other hand, if objects are irrelevant to time, e.g., buffer size, they are inherited from the *End* class.

The object-oriented database is a database that maintains not only external data, e.g., information of products and machines, but also operational data, which are generated during the simulation process. In this study, a package, *Simulation_Data_Stored,* has classes *A*, *B*, and *C* to maintain information of both external data and operational data. The schemas are defined in Schema.cpp, as follows in ObjectStore format:

```
#include "A.hh"
#include "B.hh"
#include "C.hh"
void dummy() {
        OS_MARK_SCHEMA_TYPE(A);
        OS_MARK_SCHEMA_TYPE(B);
        OS_MARK_SCHEMA_TYPE(C);}
```

Note that the attributes and behaviors of classes A, B, and C are defined in file *A.hh*, *B.hh*, and *C.hh*, respectively, and *OS_MARK_SCHEMA_TYPE()* is the predefined keyword for defining schemas in ObjectStore.

The integrated class library is the connector between the simulator and database. The connection between the simulator and database is the package *Simulation_Component*, since it contains the data of simulation subjects. In this case, the database has packages of *Simulation_Data_Stored* and *Simulation_Component*. The package *Simulation_Control* is the collection of temporary objects. A temporary object will be removed after completing simulation.

3.2. The object-oriented database and simulation

The object-oriented database management system can both store and manage objects. The following operations can assist the construction of simulation models:

(1) Collection. A collection is an object that is composed of other objects. For example, a buffer can be a collection if the buffer retains work-in-process parts. On the other hand, the parts are the elements of a collection. The elements in a collection should be of the same type but have different identification numbers. There are four kinds of collections in the ObjectStore: *set, bag, list,* and *array*. The *set* collection has unordered and unrepeated elements while the *bag* collection allows repeated elements. The *list* and *array* collections are similar to *set* and *bag*, respectively, but the sequence of elements is maintained. Note that, the declaration and operations (insertion and removal) of collections are predefined in the object-oriented

database system. In simulation, the collection can be used to model buffers, or machine groups.

(2) Query. A database management system not only stores data but also manages data for querying. Theoretically speaking, the object-oriented technology encapsulates data into objects, and only through designated predefined methods can the data be accessed. This constraint decreases the query performance. A commercial object-oriented database such as ObjectStore provides structured query language (SQL) to create, delete, query, and modify objects. This hastens the simulation operations. For example, a planner can query the sequence of machining parts in the input queue to decide whether or not to change the dispatching rules.

(3) Index. In the relational database, indices are set on attributes to enhance the query operations. Similarly, the object-oriented database can have indices on the same type of objects to increase query performance. Applying indexing to the simulation environment, we can have an index on parts in queue based on the dispatching rule. This approach enables quick retrieval of next part for machining.

3.3. Reuse of simulation

In the object-oriented simulation, the system is modeled in objects. Since the object-oriented database is the storage for keeping objects, the whole simulation environment can be retained. Unlike the conventional database, which records data only, the simulation environment is also recorded in this study. Note that the maintenance of environmental information by the object-oriented database has the advantage of scenario reuse.

As stated before, the package *Simulation_Data_Stored* is used to store external data and the objects that are generated from experiments. In the package,

- class *Simulation_Data_Stored_Schema* is created to keep object schemas and class *Simulation_Data_Object_Stored* stores the objects.
- *Simulation_Component* retains the simulation objects using classes *Simulation_Component_Schema* and *Simulation_Component_Object_Stored*. Class *Simulation_Component_Schema* defines the schemas of simulation objects and all the scenario objects are kept in the class, S*imulation_Component_Object_Stored*.
- Package *Simulation_Control* is used for simulation control and generates objects when needed. Therefore, it is not stored in the database.

Note that the objects in the database can be retrieved repetitively. When the objects are retrieved, the simulation returns to the old scenario. This can be considered as simulation experiment reuse.

4. Using object-oriented database in reverse simulation

This study uses the version management technique of the object-oriented database to build reverse simulation model. Section 4.1 discusses the issues of version management and the reverse simulation is presented in Section 4.2. The concern of setting checkpoints is covered in Section 4.3.

4.1. Version management

In the transaction processing, the database needs to maintain the atomicity, consistency, isolation, and durability of data. Normally, placing locks on data is a common approach for sharing data among multiple users. However, data cannot be locked for too long, or else system performance will decrease. The object-oriented database uses version management to process long transactions so that multiple users can share the data and have independent controls. The version management is implemented through two functions: configuration and workspace.

Since the object-oriented database represents data as objects, a configuration can be considered as a container to store objects. The configuration is the basic unit of versioning. That is, when different versions of objects are recorded, the object-oriented database uses different configurations to keep track of these versions.

The workplace is a working area where a designer checks out objects from the database and modifies them. The checkout process is a long transaction that uses the configuration as a unit. The workspace consists of the global workspace and the personal workspace. The personal workspace is a private place, and will not be accessed by other users. On the other hand, the global workspace is a public place from which users can share information with each other. In the example of this study, a global workspace is created to share information, and three personal workspaces are used to operate simulation. The configuration of objects is checked out from the global workspace to the personal workspace. When the given experiment is finished, objects are checked back into the global workspace and new versions are created.

4.2. Reverse simulation

In the traditional simulation process, experiments begin after the simulation models are constructed. The experiments change the seed of random variables to average the effect of abnormal values. Each experiment selects a set of parameters of the model and repeats several runs. Each run changes the seed and collects observation values. After each run, the experiment selects another set of parameters, repeating the process until satisfied parameters are found. In this situation, many simulation runs can be executed only on the same set of parameters. It is difficult to adjust the parameters dynamically. In contrast, the integrated architecture in this study uses the object-oriented database to record the entire simulation environment, and applies the function of version management to maintain different scenarios. This is achieved by setting different checkpoints (the setting of checkpoints is discussed later). In each checkpoint, one version of the most current scenario is recorded. In the future, the recorded scenario is traced back to observe the simulation results of a specific parameter set. The advantage is that the parameters can be changed at any checkpoint, when necessary, and a new simulation experiment can proceed from that checkpoint. After checking the experiment back to the global workspace, a new version (a new branch) of the simulation is created. The newly-created branch can also have different versions and branches. Since the simulation experiment can return to any past scenario, it is called the reverse simulation.

In the reverse simulation, a simulator may change corresponding parameters at each checkpoint or between the checkpoints. When parameter changing is at the checkpoint, the simulation creates a new version. However, if the parameter is changed between the checkpoints, the closest scenario of the checkpoint right before it is adopted, and the simulation experiment runs from the checkpoint and changes the parameters at the designated checkpoint. Figure 2 presents the differences between conventional simulation and reversed simulation. In Fig. 2(b), the reverse simulation has several checkpoints. The object-oriented database records scenarios as different versions, i.e. **v1_t1**, **v1_t2**, **v1_t3**, **v1_t4**, **v1_end**, for the parameter set *k*. If the experiment cannot produce satisfactory results, the designer can trace back to any scenario, e.g. checkpoint *t1*, change the parameter set into $k + k1$, and a new branch is created. Similarly, branches $k + k2$ and $k + k1 + k3$ are created at checkpoints *t2* and *t3*, respectively. Notice that the reverse simulation can return to any checkpoint, the parameters are changed at that checkpoint when needed, and a new branch is created accordingly. In addition, a branch can have new branches. Therefore, the version with the best performance is adopted.

It is worth noticing that simulation experiments are time consuming. The version management has another advantage in processing long transactions in different personal workspaces. The personal workspaces are independent and can share experiment results through the global workspace. The reverse simulation takes advantage of the version management to allow experiments in different workspaces. Specifically, designers can perform experiment in their own workspaces by checking out objects from the global workspace and then checking the objects back into the global workspace after completing the analysis. This approach increases the system performance. Figure 3 shows the reverse simulation in three workspaces: *User1_workspace*, *User2_workspace*, and *User3_workspace*. The version **v1_t2** is checked-out from *User1_workspace* to *User2_workspace*, and *User3_workspace* and the parameter set is changed from *k* to $k + k2$ and $k + k4$. Each workspace proceeds with its own experiment and generates different versions. All versions are checked back into the global workspace (not shown) and shared between workspaces.

Based on the previous discussions, the reverse simulation has several advantages:

(1) Through the reverse simulation, a designer can return to previous scenarios, and change the parameters if original parameters are not satisfied.
(2) New branches can adopt the past experiment scenario, resulting in a shortened computation time, i.e. the computation time before the checkpoint is not needed.
(3) The reverse simulation provides better flexibility in that the designer can choose a set of parameters at a specific checkpoint and proceed with new experiments.
(4) The experiment results can be operated in different workspaces and shared among designers through the global workspace. This approach allows designers to work collaboratively.

4.3. The setting of checkpoints

Since the reverse simulation can return to the scenarios from the checkpoints, the setting of checkpoints is very important. A simulation experiment can return to more

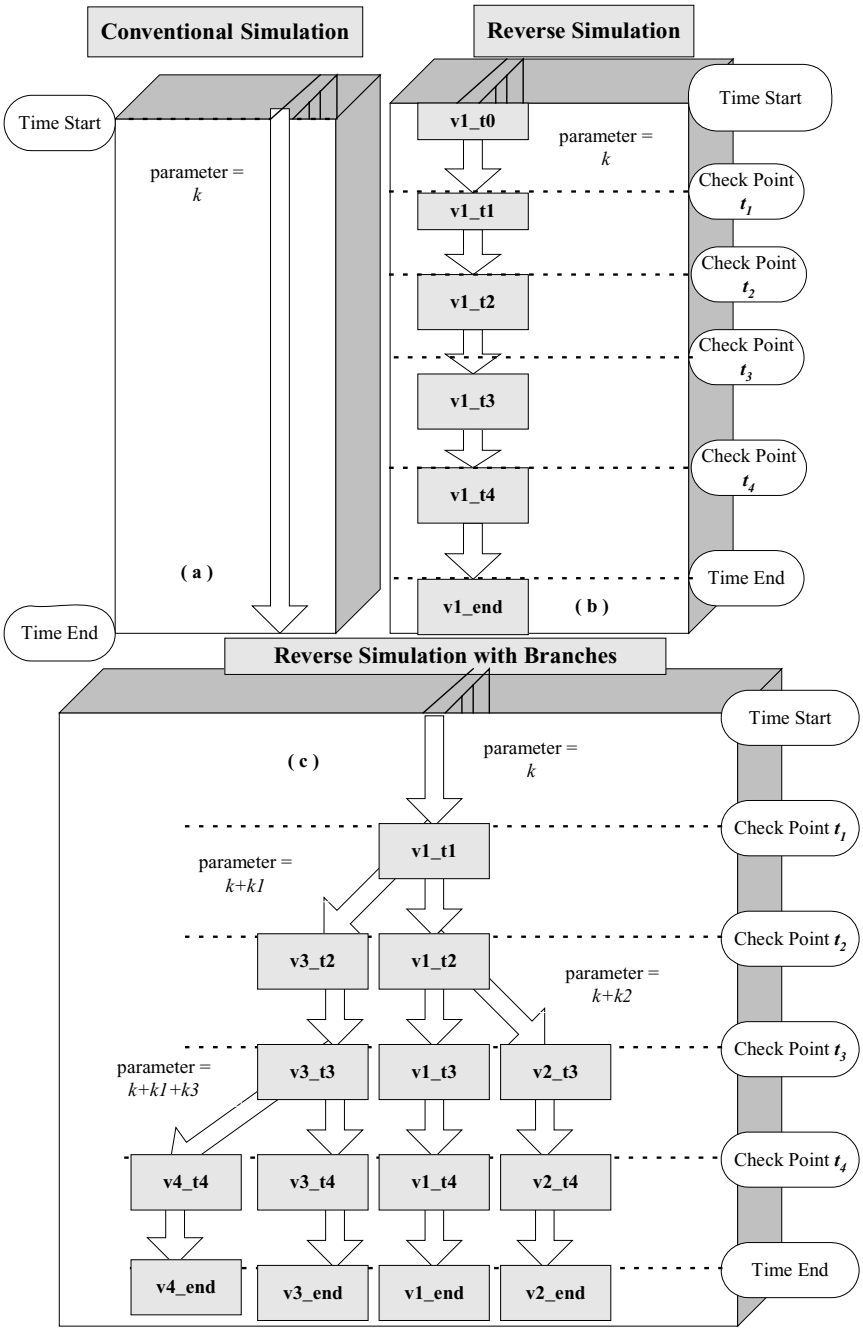

Fig. 2 The comparison between conventional simulation and reverse simulation

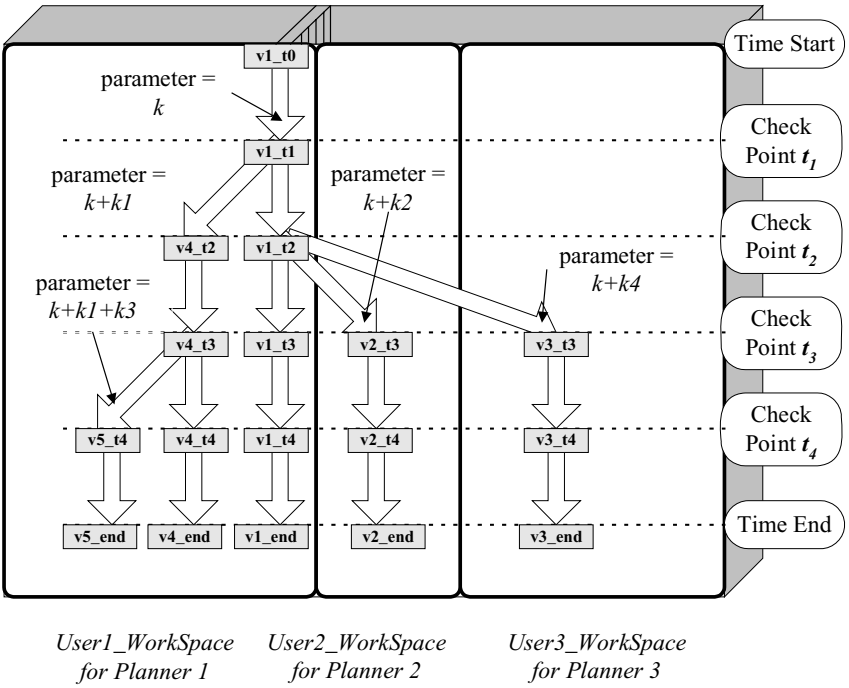

Fig. 3 Apply reverse simulation in different workspaces

scenarios if more checkpoints are set originally. However, the more checkpoints, the more database storage is required. In addition, the checkout and check-in operations are costly. On the other hand, too few checkpoints also decrease the flexibility of adjusting parameters.

A checkpoint is a snapshot in the continuous process of simulation experiments. In a simulation, the time span is continuous and the elapse is based on either next-event or fixed-time span. Because an observation is a system change during time movement, a meaningful scenario should be based on some event happening rather than being located at a fixed-time span.

The setting of checkpoints could be determined by specific quantities, performance index, or output quantities. When the designated event appears (e.g., manufacture every 500 parts), a checkpoint operation is triggered. The checkpoint holds the simulation process, downloads the whole environment into a database, and then proceeds with the experiment until the next event appears. As has been stated, the operation of checkpoint is costly, but provides high system flexibility. Unfortunately, an appropriate number of checkpoints is determined by specific problems, systems, and performance requirements and cannot be pre-determined.

5. System demonstration

This section demonstrates how collaborative commerce is possible by using reverse simulation in a collaborative manufacturing environment. A simple version for an

advanced planning and scheduling function is built to demonstrate reverse simulation using classes as discussed in session 3. The demonstration is a simplified model of collaborative commerce for illustrating the proposed architecture. The scenario simulates the planning process involved three parties: a final assembler (planner 1) and two suppliers (planners 2 and 3). The manufacturing environment has six machine groups, and each machine group has several machines of the same type. Three planners work together to simulate the manufacturing process. Note that in the proposed scenario, the planners can be belonged to different companies in the supply chain or an expended enterprise that involves various partnerships among departments. By using the reverse simulation, the simulation process is allowed to go back to old simulation parameters and results since different versions of simulation runs are maintained by the object-oriented database. In this case, the planners can choose different designs based their needs.

5.1. Object-oriented classes for the simulation of manufacturing planning

The manufacturing classes include class packages *Simulation_Control*, *Simulaition_Component* and *Simulation_Data_Stored*. Since the *Siumlation_Control* package should be accessed by other classes, it is defined as global variables.

Siumlation_Control has classes for moving time series (*SimulaEvent*), generating random variables (*RandomVar*), and assigning dispatching rules (*DispatchRule*). The classes are only used during experiments, i.e., the information is created only for temporary purposes. Therefore, the object-oriented database does not contain these classes.

Simulaition_Component has classes of facilities such as queues (*Queue*), machines (*Machine*), machine groups (*MachineGroup*), machine group set (*MDB*), materials handling machines (*MHS*), plants (*Plant*), and arrival parts (*Arrival*) (please refer to Fig. 4). Since each simulation experiment has several runs, another class (*Run*) is created to contain several identical plants. Each run is equal to the generated simulated data for a plant. The relationships between classes, i.e. generation, aggregation, and association are included. Classes *Machine*, *MHS*, and *Arrival* are related to time and are therefore inherited from class *Active*. Since class *Queue* is irrelevant to time but is relevant to quantities, it is inherited from class *End*. Class *Machine* and *Queue* are aggregated into class *MachineGroup*, and machine-group set *MDB* has several machine groups. Also, a plant class, *Plant,* is aggregated from classes *Arrival*, *MDB*, and *MHS*. Note that the relationships between classes can be considered as the message passing. For example, when class *Arrival* generates a product, the input queue of one machine group should increase. Since different types of products have different sequences of operations, the *Arrival* informs *MDB*, and the predefined method of *MDB* increases the input queue of the machine group based on the product type and operational sequence.

In Fig. 4, some classes have function parameters such as os_database* and os_configuration*. The function parameters are the declaration of configuration objects of version management in ObjectStore. Based on the grammar of ObjectStore, they are defined in classes. The attribute os_Set<A*>B is the definition of collection provided by ObjectStore, where the object information generated from class A* is retained in collection B, to hasten the query operation.

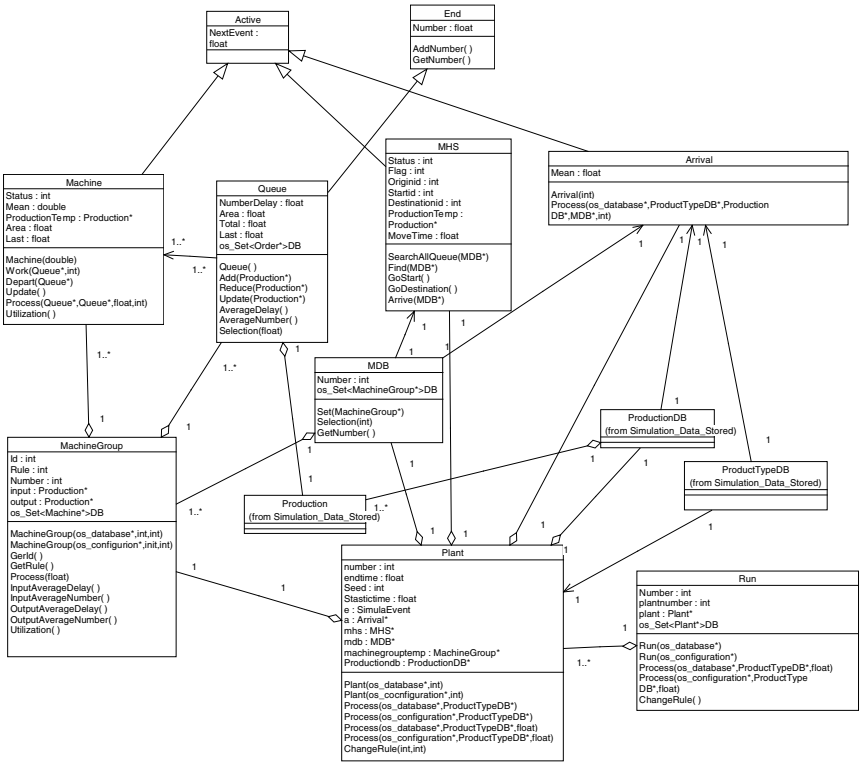

Fig. 4 The class diagram of *Simulation_Component* class package

Simulation_Data_Stored has classes of external data such as products (*ProductType*), product types collection (*ProductTypeDB*), production information (*Production*), and production information for analysis (*ProductionDB*) (please refer to Fig. 5). The relationship between classes *Production* and *ProductType* is 1 to 1. Classes *ProductTypeDB* and *ProductionDB* are the collection of classes *ProductType* and *Production*, respectively.

The previous classes are defined in the class library *Simula.hh*. ObjectStore can include the head file and declare the schema directly. To coordinate the simulation operation of classes, a common interface function *Process()* is designed to perform simulation. For example, in machine class, obtaining part is done by calling *Work()* and returning is done by *Release()*. In C++, the method is defined as:

Machine::Process()

{

Machine::Release();

Mahcine::Work();

}

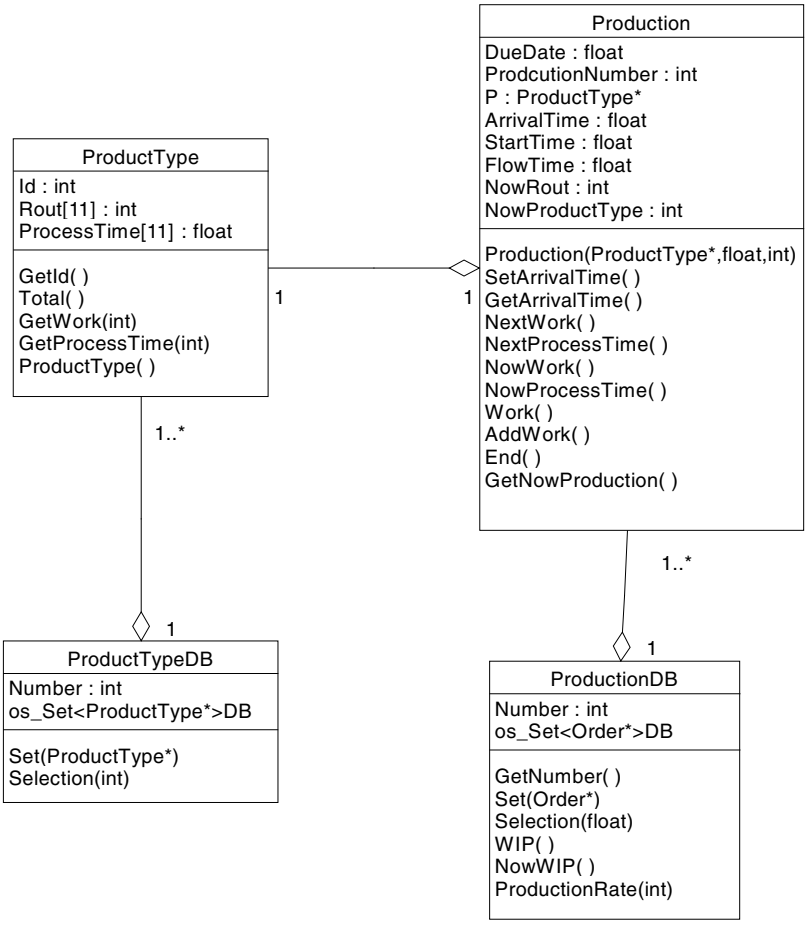

Fig. 5 The class diagram of *Simulation_Data_Stored* class package

5.2. The demonstration of reverse simulation

To demonstrate the object database simulator, this section develops a simple advanced planning and scheduling process for collaborative manufacturing environment using ObjectStore 4.0.2 and Visual C++. The advanced planning and scheduling schedules N jobs on M machines where finding an optimal sequence is considered as a complex search procedure, and therefore is suitable to solve with simulation. The manufacturing environment has six machine groups and one handling vehicle to deliver parts between machine groups. The operational settings are: (1) each machine group represents one machining process; (2) each machine group has several machines of the same type; (3) each machine group has both input and output queues; (4) each machine group has its own dispatching rules to assign jobs; and (5) each machine can manufacture one part at a time. In the environment, each part has 6 or less operations and will not go through the same machine group twice. The operational sequence and time of parts

Table 2 The machining procedure of six products for demonstration case

Product		Operation 1	Operation 2	Operation 3	Operation 4	Operation 5	Operation 6
1	Sequence	1	2	3	4	5	–
	Time*	900	450	300	300	750	–
2	Sequence	2	5	6	1	–	–
	Time	450	600	450	300	–	–
3	Sequence	6	4	1	3	–	–
	Time	450	600	150	1200	–	–
4	Sequence	4	5	–	–	–	–
	Time	1200	300	–	–	–	–
5	Sequence	5	4	1	–	–	–
	Time	300	450	300	–	–	–
6	Sequence	2	3	5	4	6	1
	Time	300	600	300	600	300	450
7	Sequence	2	4	5	3	1	–
	Time	600	150	300	150	900	–
8	Sequence	4	3	2	1	–	–
	Time	1200	300	450	1200	–	–
9	Sequence	5	6	3	–	–	–
	Time	150	300	300	–	–	–
10	Sequence	6	4	–	–	–	–
	Time	600	900	–	–	–	–

*Unit of time is one second

are fixed (shown at Table 2). For example, the machining sequence of product 6 is operations 2, 3, 5, 4, 6, 1, and the machining time are 300, 600, 300, 600, 300, 450 sec, respectively. This information is stored in the database.

The part enters the machine groups one at a time; each part has the same probability to enter the system. The arrival of parts is at the exponential distribution and the mean value is 150 sec. The materials handling vehicle takes 30 sec to move parts from one station to another. In the experiment, several performance indices are observed: (1) the average utilization of each machine in the machine group; (2) the average waiting time of parts in queue; (3) the average number of parts in queue; (4) the average work-in-process (WIP); and (5) the average output rate of parts (output amount/input amount). Table 3 summarizes the variables of the experiment environment.

During the simulation process, using an advanced planning and scheduling scenarios for illustration, the final assembler (planner 1) first selects checkpoints. Checkpoints are set based on both time and quantity perspectives. Experiment A uses time frame for the checkpoints, and Experiment B uses quantity as the checkpoints. Although more checkpoints can provide more choices for planners to return from current versions to previous versions, each experiment was assigned with four checkpoints to prevent system overload. To simplify the problems, the checkpoints have equal duration. Each experiment has ten runs. The collection of simulation data starts after 30 min.

In Experiment A, the total observation duration is 24 h, i.e. 86400 sec. Since the span between checkpoints is equal, the checkpoints are set at 21600, 43200, 64800, and 86400 sec. Including the initial setting at 0 sec, five recording points are located. In Experiment B, the total monitor quantity is 200. Therefore, the checkpoints are set at

Table 3 The variables of the experiment environment

Variable type	Descriptions
Performance measures	The average utilization of each machine in the machine group
	The average waiting time of parts in queue
	The average number of parts in queue
	The average work-in-process (WIP)
	The average output rate of parts (output amount/input amount)
Input parameters	Each part has the same possibility to be processed
	The due date is the current system time plus the exponential distribution with mean value equals to total machining time
	The arrival of parts is at the exponent distribution and the mean value is 150 sec
	The materials handling vehicle takes 30 sec to move parts from one station to another
Checkpoints of:	
Experiment A	0, 21600, 43200, 64800
Checkpoints of	
Experiment B	50, 100, 150, and 200

50, 100, 150, and 200. Figure 6 presents this idea. Planner 1 simulates the experiments and returns the versions **v0_t0**, **v0_t21600**, **v0_t43200**, **v0_t64800**, and **v0_t86400**.

Assuming the initial dispatching rule for machine groups is shortest processing time (SPT), which has the characteristics of high machine utilization, low part waiting time, and low average waiting quantities. However, when the bottleneck machine applies the SPT rule, the high utilization will cause a long waiting line if the machine capacity is insufficient for the load. In this case, other dispatching rules are expected. Table 4 shows the simulation results and throughput of machine group 4. From the high utilization, it is noticed that machine group 4 is the bottleneck machine group, and the average delay and average number (quantity) in the input queue are increasing. In addition, the output of product 4 and 8 are much lower than other product types. This indication reveals that the dispatching rule SPT is not appropriate in this machine group.

Since there are several combinations of parameters, and the same parameter may have a different performance at different checkpoints, the simulation experiments can have versions like a tree. Each branch represents experiments of a set of parameters. From the database point of view, every experiment is a long transaction. That means that the data have to be locked by the transaction for long periods.

The version management of the object-oriented database shows another advantage from this perspective. A version of objects can be checked out from global workspace to a personal workspace and experiments can be conducted independently. This allows planners of different organizations in the collaborative network to work together. After the experiment, the objects can be checked back into the global workspace, and a new version of objects is created. This behavior can also be observed in Fig. 6.

Planner 1 completes the experiment and generates versions of **v0_t0**, **v0_t21600**, **v0_t43200**, **v0_t64800**, and **v0_t86400**. The planner can change the dispatching rules from shortest processing time (SPT) to first-come first-serve (FCFS), or earliest due

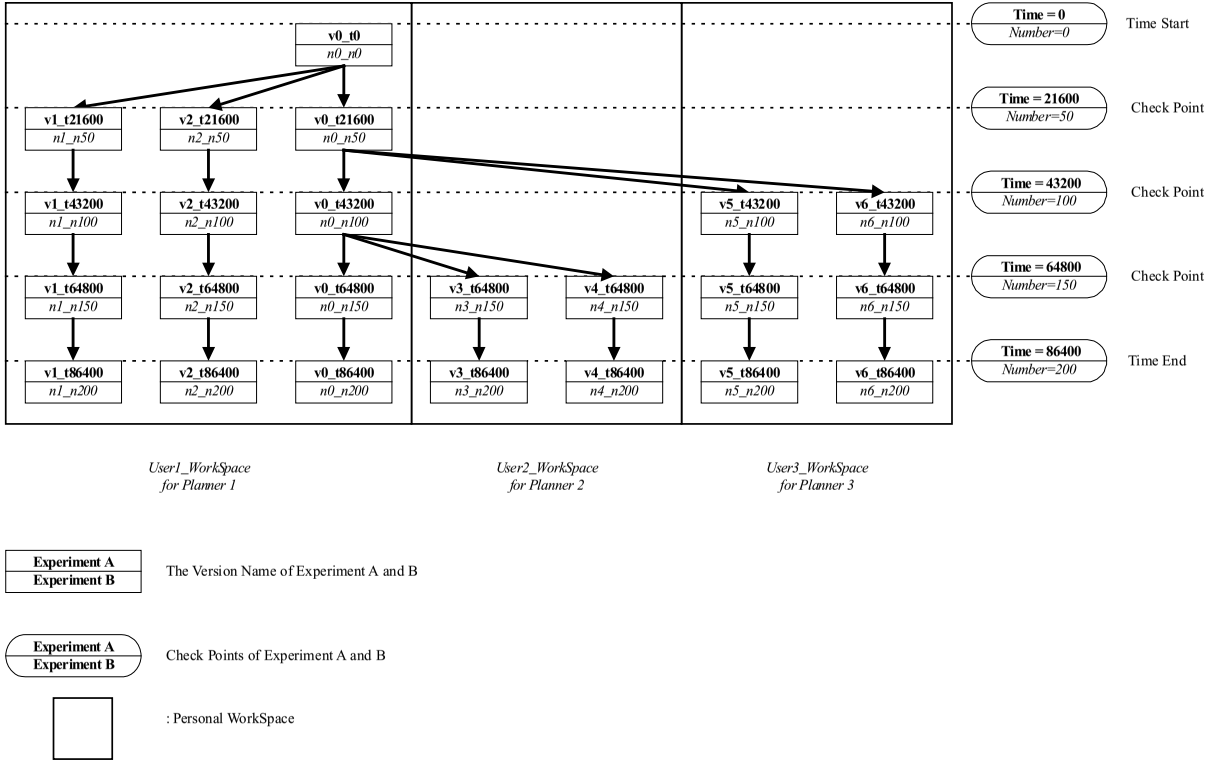

Fig. 6 The simulation versions and workspace in the reverse simulation

Table 4 The simulation results and throughput of machine group 4 in Experiment A

Version	**v0_t 21600**	**v0_t 43200**	**v0_t 64800**	**v0_t 86400**	**v1_t 86400**	**v2_t 86400**	**V3_t 86400**	**V4_t 86400**	**v5_t 86400**	**v6_t 86400**
Run time	21600	43200	64800	86400						
Average utilization	0.986	0.993	0.995	0.996	0.99	0.99	0.99	**0.99**	0.99	0.99
Average delay in input queue	2964	5820	8877	11873	19610	18480	16005	**15553**	18702	17457
Average number in input queue	6.5	13.3	20.8	27.9	50.2	46.8	41.5	**39.3**	48.4	44.7
Average delay in output queue	1189	2458	3717	5062	2562	2821	3323	**3546**	2730	3051
Average number in output queue	1.7	3.9	6.1	8.4	3.2	3.7	4.1	**4.7**	3.4	4.1
Average WIP	21.3	39.2	57.6	75.3	78.2	75.7	80.0	**77.5**	77.6	76.4
Average yield :										
Rate of product 1	0.41	0.55	0.54	0.56	0.38	0.48	0.32	**0.46**	0.37	0.46
Rate of product 2	0.64	0.70	0.60	0.70	0.82	0.82	0.78	**0.79**	0.78	0.79
Rate of product 3	0.54	0.54	0.63	0.65	0.40	0.40	0.40	**0.40**	0.41	0.40
Rate of product 4	0.23	0.19	0.16	0.14	0.51	0.49	0.55	**0.49**	0.52	0.49
Rate of product 5	0.68	0.79	0.72	0.80	0.49	0.51	0.45	**0.79**	0.48	0.50
Rate of product 6	0.41	0.41	0.46	0.50	0.33	0.42	0.27	**0.39**	0.35	0.41
Rate of product 7	0.45	0.61	0.63	0.55	0.42	0.42	0.42	**0.42**	0.43	0.45
Rate of product 8	0.16	0.10	0.08	0.065	0.41	0.36	0.44	**0.39**	0.42	0.35
Rate of product 9	0.79	0.80	0.74	0.75	0.87	0.86	0.89	**0.87**	0.84	0.83
Rate of product 10	0.64	0.69	0.79	0.81	0.52	0.54	0.51	**0.53**	0.48	0.50

date (EDD) for machine group 4, creating two new branches, such as branch of **v1_t21600**, **v1_t43200**, **v1_t64800**, **v1_t86400**, and branch of **v2_t21600**, **v2_t43200**, **v2_t64800**, **v2_t86400**. In contrast, in the conventional simulation approach, the planner goes back to 0 sec, changes parameters, and proceeds with another experiment.

Supplier 1 (planner 2) can check out objects to a personal workspace, User2_Workspace, at 43200 sec, and can change parameters thereafter. Similarly, supplier 2 (planner 3) can checkout objects to User3_Workspace at 21600 sec. In this case, the planners are allowed to determine the production plans together by considering various material availabilities, plant capacities, inventory planning, and available-to-promise (ATP). Table 4 shows the comparison among versions **v1_t86400**, **v2_t86400**, **v3_t86400**, **v4_t86400**, **v5_t86400**, and **v6_t86400**. If the requirements of the average product type outputs cannot lower than 0.35, lowest average delay time, and lowest delay quantities, version **v4_t86400** is a satisfied setting. Therefore, the answer is the branch of **v0_t0**, **v0_t21600**, **v0_t43200**, **v4_t64800**, and **v4_t86400**. SPT is used before 43200 sec, and EDD is used after it.

Experiment set B uses quantities as the checkpoints. The mainstream of versions is **n0_n0**, **n0_n50**, **n0_n100**, **n0_n150**, and **n0_n200**. Similar to Table 4, the simulation results and throughput of machine group 4 is presented in Table 5. Since machine group 4 is the bottleneck machine group, the SPT dispatching rule causes the output of product types 4 and 8 to be much lower than other product types. Similar to the approach in experiment set A, dispatching rules of FCFS and EDD are tested, and

Table 5 The simulation results and throughput of machine group 4 in Experiment B

Version	**n0_n50**	**n0_n100**	**n0_n150**	**n0_n200**	**n1_n200**	**N2_n200**	**n3_n200**	**n4_n200**	**n5_n200**	**n6_n200**
Run time	27316	51511	77412	101935	108197	107284	110174	**106178**	109971	107061
Average utilization	0.99	0.99	0.99	0.99	0.99	0.98	0.99	**0.99**	0.99	0.99
Average delay in input queue	3788	7420	10598	13998	24680	24456	20159	**18878**	23602	20804
Average number in input queue	10.1	17.5	26.1	34.4	65.4	61.0	56.7	**52.7**	67.3	61.5
Average delay in output queue	1038	2553	4008	5489	2890	3414	3736	**3804**	2681	3064
Average number in output queue	1.7	4.1	7.1	9.6	3.7	4.3	5.0	**5.3**	3.7	4.6
Average WIP	24.4	44.3	65.6	86.11	95.0	95.8	95.6	**90.6**	94.9	90.0
Average yield:										
Rate of product 1	0.47	0.51	0.53	0.53	0.39	0.43	0.31	**0.45**	0.37	0.42
Rate of product 2	0.67	0.68	0.70	0.67	0.83	0.78	0.78	**0.80**	0.79	0.82
Rate of product 3	0.66	0.61	0.64	0.69	0.44	0.45	0.43	**0.41**	0.42	0.38
Rate of Product 4	0.16	0.12	0.11	0.11	0.48	0.50	0.54	**0.46**	0.50	0.46
Rate of Product 5	0.72	0.82	0.77	0.77	0.45	0.52	0.47	**0.52**	0.48	0.52
Rate of Product 6	0.53	0.54	0.52	0.56	0.37	0.44	0.34	**0.44**	0.37	0.47
Rate of Product 7	0.57	0.64	0.58	0.59	0.43	0.41	0.41	**0.47**	0.41	0.44
Rate of product 8	0.08	0.10	0.08	0.07	0.43	0.36	0.45	**0.37**	0.46	0.37
Rate of Product 9	0.76	0.76	0.81	0.82	0.88	0.90	0.88	**0.88**	0.90	0.88
Rate of product 10	0.62	0.74	0.77	0.79	0.52	0.48	0.50	**0.53**	0.47	0.50

results are shown in Table 5. The conclusion can be that the satisfactory version is located at **n4_n200**, and branch **n0_n0**, **n0_n50**, **n0_n100**, **n4_n150**, and **n4_n200** is the answer. The SPT is adopted before quantities 100, and EDD is used after that.

As it was demonstrated, through the use of reverse simulation, the collaborative parties (one final assembler and two suppliers in this scenario) can work together on determining various production factors based on preset performance indexes. This allows collaborators to participate and contribute to various stages of collaborative commerce.

6. Conclusion and future study

This study presented the process of using reverse simulation for advanced planning and scheduling in a collaborative commerce environment by developing an integrated architecture of the object-oriented database and the object-oriented simulator. The integrated architecture has a unique data structure between database and simulator. In this architecture, the version management records experimental parameters and results for doing reverse simulation. Specifically, the database manages objects and provides collection, query, and indexing functions. In addition, the simulation models can be reused, and returning to the old scenario becomes possible. The scenarios are recorded as branches in the database when the checkpoints are set. The reverse simulation uses different parameters at different checkpoints. This approach provides a mechanism to modify parameters easily and dynamically. Each branch has new versions of objects maintained in checkpoints. A branch can have another branch going out; thus the simulation experiment looks like a tree. Instead of restarting from the beginning at each experiment, the reverse simulation allows planner to begin with any existing versions. This approach also has the advantage of the reuse experiment scenario, which decreases the computation time. Given all these features, multiple planners can do the experiment independently, interactively, and collaboratively in different personal workspaces, and the experiment results can be shared through global workspace.

As the information technology advances and the global competition intensifies, more and more collaborations between businesses are emerging. This study presented a concept of collaborative planning using the technology of reverse simulation. It should be noted that collaboration could also appear in many other applications such as design, engineering, knowledge sharing, decision-making, and supply chains. Moreover, issues such as security, legality, and ethics are worth further exploiting. For example, how could the product demand data be shared among the planners who work on the same advanced and planning project but belonged to different companies? Especially, the demand may include sensitive information of the competitors of the collaborators. How could a company take advantage of knowing the proprietary knowledge of a collaborator and use it to improve it's own product. These issues remain to be further explored.

Acknowledgements This work is supported in part by National Science Council of Republic of China under the grant NSC 89-2213-E-033-028.

References

Aviv Y (2001) The effect of collaborative forecasting on supply chain performance. Man Sci 47(10):1326–1343

Banks J, Aviles E, McLaughlin JR, Yuan RC (1991) The simulator: New member of the simulation family. Interfaces 21(2):76–86

Banks J, Carson JS, Nelson BL, Nicol DM (2000) Discrete-event system simulation, 3rd edn. Prentice-Hall, Englewood Cliffs, NJ

Barcio B, Ramaswamy S, Barber S (1997) An object-oriented modeling and simulation environment for reactive systems development. Int J Flex Manuf Syst 9(1):51–80

Bertino E, Martino L (1991) Object-oriented database management systems: Concepts and issues. IEEE Comput 24(4):33–47

Browne J, Zhang J (1999) Extended and virtual enterprises—Similarities and differences. Int J Agile Manag Syst 1(1):30–44

Browne J, Sackett PJ, Wortmann JC (1995) Future manufacturing systems—Towards the extended enterprise. Comput Ind 25:235–254

Chopra S, Meindl P (2001) Supply chain management: Strategy, planning and operation. Prentice-Hall, New Jersey

Childe SJ (1998) The extended concept of co-operation. Prod Plan & Cont 9(4):320–327

Contero M, Company P, Vila C, Aleixos N (2002) Product data quality and collaborative engineering. IEEE Comp Graph Appli 22(3):32–42

Deitel HM, Deitel PJ, Steinbuhler K (2001) e-Business and e-commerce for managers. Prentice-Hall, New Jersey

Eum DD, Minoura T (1996) Structural active object system for mixed-mode simulation. IEICE Trans Inform Syst E79-D(6):855–865

Du T, Wolfe P (1997) An implementation perspective of applying object-oriented database technologies. IIE Trans 29:733–742

Du T, Wu JL (2001) Developing an evolutional vehicle routing system in object-oriented paradigm. Comp Ind 44:229–249

Elmasri R, Navathe SB (1994) 'Object-oriented database,' in Fundamentals of database systems, 2 edn. Benjamin-Cummings, California pp.663–701

Fliedner G, (2003) CPFR: An emerging supply chain tool. Industrial Man Data Syst 103(1):14–21

Gupta A, Stahl DO, Whinston AB (1998) Managing computing resources in intranets: An electronic commerce perspective. Dec Supp Syst 24(1):55–69

Gupta VK, Govindarajan S, Johnson TM (2001) Overview of content management approaches and strategies. Electro Mkts 11(4):281–287

Henderson-Sellers B (1991) A book of object-oriented knowledge. Prentice-Hall, New Jersey

Huang GQ (2002) Web-based support for collaborative product design. Comp Ind 48:71–88

Jagdev HS, Browne J (1998) The extended enterprise-a context for manufacturing. Prod Plan Contr 9(3):216–229

Kim W (1991) Object-oriented database systems: strengths and weaknesses. J Obj-Ori Progr 4(4):21–29

Kumar A, Zhao JL (2002) Workflow support for electronic commerce applications. Dec Supp Syst 32(3):265–278

Kuroda M, Shin H, Zinnohara A (2002) Robust scheduling in an advanced planning and scheduling environment. I J Prod Res 40(15):3655–3668

Law AM, Kelton WD (2000) Simulation modeling and analysis, 3rd edn. McGraw-Hill, New York

Li Eldon, Du T (2005) Advances in electronic business, vol I: Collaborative Commerce. IGI Press, Hershey, Pennsylvania

McGreevy RL, Pusztai L (1982) Reverse Monte Carlo Simulation: A new technique for the determination of disordered structure. Mol Simu 1(6):359–367

McGregor DR, Randhawa SU (1994) ENTs: An interactive object-oriented system for discrete simulation modeling. J Obj-Ori Prog 5(8):21–29

Mize JH, Bhuskute, Pratt, HCDB, Kamath M (1992) Modeling of integrated manufacturing systems using an object-oriented approach. IIE Trans 24(3):14–26

Pegden CD, Shannon RE, Sadowski RP (1995) Introduction to simulation using SIMAN, 2nd edn. McGraw-Hil, New York

Pritsker AAB (1986) An introduction to simulation and SLAM II, 3rd edn. Wiley-Interscience, New York

Raghu TS, Ramesh R, Chang A-M, Whinston A (2001) A collaborative decision making: A connectionist paradigm for dialectical support. Inform Syst Rest 12(4):363–383
Turban E, King D (2003) Introduction to e-commerce. Prentice-Hall, New Jersey
Wild R, Pignatiello J (1994) Finding stable system designs: A reverse simulation technique. Comm ACM 37(10):87–98
Zacharia G, Moukas A, Maes P (2000) Collaborative reputation mechanisms for electronic marketplaces. Dec Supp Syst 29(4):371–388
Zhang D, Zhang HC (1999) A simulation study of an object-oriented integration testbed for process planning and production scheduling. Int J Flex Manuf Syst 11:19–35