TOGA—A Customizable Service for Data-Centric Collaboration

Jürgen Sellentin[1,2], Aiko Frank[2], Bernhard Mitschang[2]

[1] DaimlerChrysler AG, Research & Technology, Dept. CAE-Research (FT3/EK),
P.O. Box 2360, D-89013 Ulm, Germany
Juergen.Sellentin@DaimlerChrysler.COM
[2] Universität Stuttgart, Fakultät für Informatik, IPVR,
Breitwiesenstr. 20-22, D-70565 Stuttgart, Germany
{Aiko.Frank, mitsch}@informatik.uni-stuttgart.de

Abstract. Collaboration in cooperative information systems, like concurrent design and engineering, exploits common work and information spaces. In this paper we introduce the TOGA service (**T**ransaction-**O**riented **G**roup and Coordination Service for Data-Centric **A**pplications), which offers group management facilities and a push model for change propagation w.r.t. shared data, thus allowing for group awareness. Through TOGA's customizability and its layered architecture the service can be adapted to a variety of different collaboration scenarios. Multiple communication protocols (CORBA, UDP/IP, TCP/IP) are supported as well as basic transaction properties. Our approach enables the evolution of a set of separate applications to form a cooperative information system, i.e., it provides a technique towards component-oriented system engineering. In this paper we report on design issues, implementation aspects, and first experiences gained with the TOGA prototype.

1. Introduction

Concurrent and simultaneous engineering (see [2], [3]) is perceived an enabling technology to faster and better design of complex products as e.g. engineered artifacts. From a system point of view this technology refers to a whole spectrum that ranges from multiple and efficient communication protocols [29] to activity coordination (as e.g. provided by workflow systems [17]) and to appropriate support for collaboration techniques (as for example known as computer supported cooperative work [5]).

In contrast to administration and business scenarios these concepts and techniques are not directly applicable to the overly complex area of design, especially engineering design. There, existing system structures, proven processing scenarios, and chosen design methodologies have to be observed and effectively supported. The reason for that is the complexity of the heterogeneous application environment (e.g. CAD systems or FEM systems), its (often) proprietary data structures and data formats that obstruct data exchange, and finally the applied proprietary design process and design methodology itself. As a consequence, effective and practical support for concurrent and simultaneous engineering has to act as a kind of glue to flexibly combine these separate system components to form a cooperative information system, thereby still supporting the existing and well-established design methodology.

M. Jarke, A.Oberweis (Eds.): CAiSE'99, LNCS 1626, pp. 301-316, 1999.
© Springer-Verlag Berlin Heidelberg 1999

1.1 CAD Application Area

Divide and conquer is the underlying technique that allows to cope with the inherent complexity of engineering design. As depicted in Figure 1, this strategy refers to a decomposition of the artifact under design into separate design items and to the delegation of design tasks to separate design teams, with each design team being responsible for the design of the assigned design item.

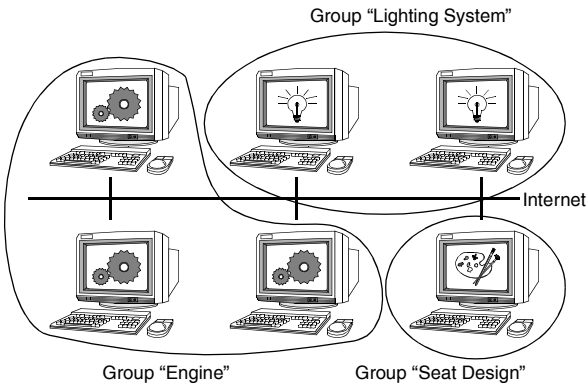


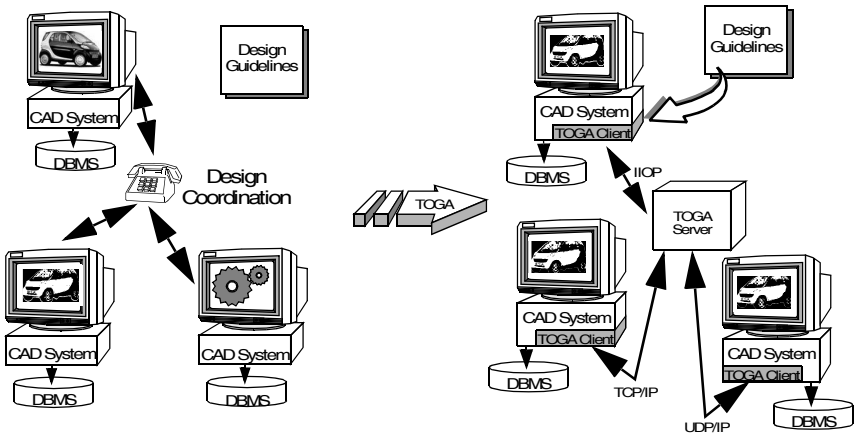
Fig. 1. Design Decomposition: Separate Design Groups and Associated Design Items

To be able to concurrently work within a team on those partitions without interfering or avoiding to interfere with other designers is a demanding task. Therefore it is necessary to organize groups of designers who share a set of design items. Their work should be coordinated automatically to avoid inconsistencies, even if users are spread all over the world (using workstations connected through the Internet).

Unfortunately, current distributed CAD environments often suffer from a lack of coordination among the ongoing design activities. Participants in design have to cooperate mostly by other means than their current design environment; they often have to resort to personal communication methods (like e-mail or telephone) or have to follow certain external design guidelines in order to coordinate their work and to resolve conflicts. Since these rules and design guidelines are specified outside the system, integrity cannot be controlled automatically, leading to manual and error prone design control. This scenario is depicted in Figure 2a.

It is exactly the focus of our work to remedy these grievances. Though there are already a lot of research activities considering similar scenarios, we are not aware of a system that covers all of the properties listed below (see also Section 3). In this paper we report on the design, implementation, and first experiences of our system approach called TOGA. As depicted in Figure 2b, TOGA provides a number of necessary services and system properties. Among the most important ones are:

- TOGA treats participating information systems, as e.g. CAD systems, as separate components within a distributed, collaborative environment, i.e., it supports component-oriented system engineering.



(a) Current Scenario (b) Utilization of TOGA

Fig. 2. Introduction of TOGA

- TOGA is customizable in such a way that the system behaves according to the given design guidelines. These guidelines cover the operations and undo-operations, the object granules, the conflict situations and associated resolution strategies that altogether make up the desired collaboration model.
- TOGA is group-oriented and data-centric as opposed to user- and process-oriented.
- TOGA delivers a synchronous collaboration approach combined with a distributed, shared work and information space.
- TOGA synchronizes (by means of operation coordination) the state of all data objects that are part of the shared information space.
- TOGA automatically controls this integrity of shared data using a transaction-oriented protocol.
- TOGA consists of an application-specific component for customization and a generic (and thus reusable) middleware/server component.
- TOGA supports multiple communication protocols.

1.2 Overview of the Paper

In this paper we introduce the different layers of TOGA and discuss all aspects that have to be decided upon for customization w.r.t. a certain collaboration model. Due to space restrictions, we focus on the system approach, but not on a (formal) model or methodology on how to map particular design processes and their design guidelines onto our system.

The following section covers the design issues of the layered architecture, describing the interfaces of client layers as well as the underlying CORBA service. Readers that are not familiar with CORBA at all should refer to [13], [19] or [25] first. Section 3 contains a discussion on related work covering similar services as well as workflow and CSCW technology. In Section 4 we present decisions and experiences that are related to our prototype implementation. It consists of a generic, CORBA-based TOGA Service as well as a simple test application. A discussion of more complex

application scenarios our implementation might be adopted to, some conclusions and an outlook to additional future work are given in Section 5.

2. Design

Our architecture enables the evolution of a set of separate applications to form a cooperative information system. The cooperation concept is realized by a powerful collaboration approach that exploits a common work and information space. The set of applications that want to collaborate through a common information space have to decide on a number of critical aspects in order to use TOGA effectively. First of all, the contents, i.e. the shared data objects, of the common information space have to be defined. Second, the object granule has to be decided upon, third, the relevant operations (and corresponding undo operations) to change the information space are to be defined as well, and fourth, strategies to decide on conflicts among these operations have to be specified (conflicts are treated by aborts, see below). These decisions clearly depend on the application scenario, but are necessary in order to determine the particular collaboration support TOGA is asked for. For example, considering object rotation in several dimensions, each rotation in a single dimension or only the final state may be synchronized, or, creating a new module, the creation of each (partial) item or only the creation of the (final) module may be synchronized. Choosing the right granule of data objects and operations is the key issue in defining the appropriate level of collaboration.

Once the common work and information space is defined, its consistency is automatically controlled by transaction-oriented processing: Each operation that effects the common information space is encoded into an event and sent to the TOGA server, which utilizes a 2-Phase-Commit protocol (2PC, see [6]) to distribute this event to all group members and to ensure that they all have performed the encoded operation successfully. If an application is not able to perform such an operation, it votes “abort” and TOGA notifies all group members to undo this operation.

In order to support various environments, our service has a layered architecture as displayed in Figure 3. The top level interface is written in C++ (since most CAD applications are written in C++)[1] and offers the integrated functionality needed by the application. It hides the underlying implementation, which may in principal consist of one or several components that utilize other services and communicate via CORBA, or directly via TCP/IP or UDP/IP etc. The prototype implementation described in this paper is based upon a single, integrated CORBA service and several CORBA clients.

The application layer (Section 2.2) contains all information specific to a particular application scenario and has to be developed as part of the customization process. E.g., it is responsible for the encoding of application-specific operations into generic events (and vice versa). Next comes the session layer (Section 2.3), which has a slim, but powerful interface. It will propagate (generic) event data received from the application

[1] We are aware of the fact that not all CAD systems provide suitable APIs for their integration into a TOGA environment, yet. Nevertheless, considering our experiences at DaimlerChrysler, we assume that the vendors of such systems will offer those APIs once proof of concept is done and a strong demand for this technology arises among participating enterprises.

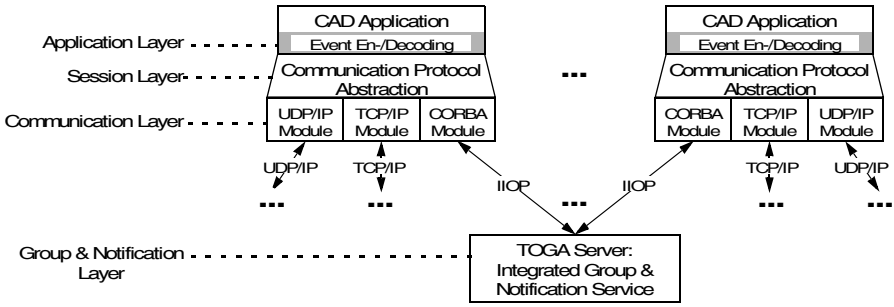


Fig. 3.:Overall Architecture of TOGA

layer to the actual communication layer (and vice versa). This layer is necessary since we want to transparently support direct usage of low level communication protocols like TCP/IP and UDP/IP, but also more powerful middleware like CORBA. The underlying communication layer (Section 2.4) is split into three different modules with specific interfaces. Since this paper is focussed on a CORBA-based server for TOGA, we will not describe interfaces to the TCP/IP and UDP/IP modules in detail. The interface of the group and notification layer (Section 2.5) is defined using the Interface Definition Language (IDL) of CORBA. This enables a component-based architecture and thus the integration of existing software (that is wrapped by specific CORBA skeletons) or the development of new services (like the TOGA server described here).

A simple processing example is presented in the next subsection. It is followed by a more detailed description of each layer, using this example to illustrate the realized functionality. Finally two figures are presented that describe the flow of events through all layers. Please note that these figures are specific to the example as well.

2.1 Overall Processing Example

Let us assume two applications working on the same set of data (e.g. the two applications working on the "lighting system" in Figure 1). According to the design decomposition, they shall become members of the same group. Furthermore, application 1 wants to perform an object rotation that is, according to the customization, an operation to be controlled by TOGA. How does it work?

In our approach, any kind of data and event processing has to be started by opening a new session at the client site. We propose to attach this step to the initialization phase of each application. The desired communication type (CORBA, TCP/IP, or UDP/IP) has to be specified as a parameter. After that, the chosen protocol is not visible any more. It is encapsulated by the session layer. Though each client chooses a single communication protocol, a server might support multiple ones.

In a next step, the application will join a group. Since the application layer is responsible for the relationship between data objects and group definitions (thus it can decide which operation influences which groups' information space), an application may in principal be part of several groups. Defining our architecture, we decided that an application has to be part of exactly one group, though. If it is necessary to access

data objects that are related to different groups at the same time, the design could be extended to hierarchical groups.

By joining a group that is already controlled by TOGA, the session layer will automatically forward the status of all data objects of the common information space to the application layer of the new member. Once the application is a member of a group, it may create, access or modify all data objects related to this group. Those actions are typically encoded into events and forwarded to the session layer. All events are shared between all members of an application group, guaranteeing consistency of the information space. E.g., the rotation of a 3D object might be encoded in an event that contains the object ID, the operation code and the angle to rotate. Afterwards, TOGA ensures that either all or none of the group members have performed this operation.

2.2 Application Layer

The application layer realizes the customization. It has to bridge the gap between the application and the generic TOGA components presented in this paper. Thus it has to be aware of the current design decomposition, related data objects, operations and corresponding undo operations. All this information is neither available to the TOGA server nor to the session or communication layer (since these components have a generic interface).

Knowing the design decomposition, the application layer defines groups and related sets of data objects. In addition, it has to define which operations have to be synchronized immediately and which (sequence of) operations can be performed locally before the next synchronization step is necessary.

Once the application has joined a group already known and controlled by the TOGA server, or it has created (and implicitly joined) a new group using the session layer's operations[2], the application layer realizes a (virtual) information space for that particular group. Any changes to the information space are propagated to all group members via TOGA.

Vice versa, the application layer has to decode each event received from the session layer (method `messageUp`, see next section) and to check if the encoded operation is in conflict with other operations that were already performed locally. In case of no conflict, the application layer initiates the execution of the encoded operation and sends a "vote commit" message to the session layer, otherwise the operation is not performed and the application layer votes to abort. For each (sequence of) operation(s) that is to be synchronized, the application layer has to define undo operations, which have to be performed if any other group member initiates an abort.

Since all these tasks are very application-specific, they can not be moved to or controlled by the TOGA server. Thus, all semantic aspects of the collaboration and its customization are encapsulated within the application layer, whereas the underlying client layers and the TOGA server offer technical, generic collaboration support (that is

[2] The session layer can only be used to query for groups that are currently known to and processed by the TOGA server, but the server does not know about the original design decomposition. Since the application layer is aware of the design decomposition, it might initiate the activation of a group (within the TOGA server) that is responsible for another design task.

described in the following subsections). In order to exploit this support, it is very important that each client application contains an application layer with the same (or a compatible) customization model! Please keep in mind that a formal description of this model is out of scope of this paper. However, for our test application (see Section 4.1) we hand-crafted a particular customization.

2.3 Session Layer

The session layer has a slim interface that covers three different issues: The propagation of event data (received from the application layer) to the communication layer, processing and forwarding of group operations, and upcalls (to the application layer) due to event data received from the communication layer. The session layer's interface comprises the C++ class `Session` and two global methods `makeSession` and `destroySession` to start and end a session of work. The class `Session` defines generic data structures and methods for sending events (`sendData`), propagating context information (`sendAllData`), group administration (`AddToNewGroup`, `addToGroup`), voting (`voteAbort`, `voteCommit`), receiving events by upcalls (`messageUp`, see below), and additional convenience functions. `makeSession` initializes a global context (see Section 2.1) whereas `destroySession` ensures that the application will be removed from its group. In addition, it notifies the communication layer to cancel all connections in a consistent manner. Since our architecture focuses on a generic approach that should also support low-level protocols, there is no need for a more complex object-oriented design. Even more, the introduction of additional classes might lead to additional IDL interfaces at the CORBA level that will reduce performance of communication [24].

Since the session layer realizes only a pure push model for events (see e.g. [14]), there is only a single (asynchronously called) method `messageUp` for forwarding events and related data. Though this method is declared as part of the session layer's interface, it has to be defined (and thus implemented) by the application layer. Actually, it is only called by the session layer to forward events to the application layer. Considering the application scenario mentioned above, we do not expect that there is a need for a pull model. More precisely, we assume that an application will not wait for an event, but it will perform operations that initiate events (using the push model).

If the application layer generates a new event due to an operation of the application, it will call the method `sendData` to forward this event to all other group members. Thereby it will pass all necessary data (e.g. the operation ID for object rotation, the object ID and the rotation angle - see Section 2.1) encoded in an instance of `dataType`.

2.4 Communication Layer

The communication layer consists of three modules that establish a connection to the server component based on the actual protocol (CORBA, TCP/IP, or UDP/IP) selected. All modules have the same interface that is very similar to the session layer's (and is therefore omitted here). During an open session, only one communication module is active at a time (as specified in Section 2.1).

The CORBA module consists of a CORBA client stub for propagating events to the TOGA server (based upon `interface c2s` of Figure 4), a CORBA server object

to receive events from the TOGA server (implementing `interface s2c` of Figure 4) and some methods that map between those objects' interfaces and the interface of the communication layer. E.g. the methods `sendData`, `sendAllData`, `voteCommit` and `voteAbort` are mapped to the single method `sendData` of the TOGA server.

2.5 TOGA Server

The TOGA server (**T**ransaction-**O**riented **G**roup and Coordination Service for Data-Centric **A**pplications) is an integrated CORBA service that manages transaction-oriented event processing as well as related group administration. Its interface is presented in Figure 4. The server itself implements `interface c2s` (client to server), whereas `interface s2c` (server to client) is implemented by the communication layer of each participating application.

Data structures presented in Figure 4 are used in both components. Since the TOGA server is a CORBA-based service (see Figure 3), we can assume reliable communication. The other modules will have more complex protocols.

Interface `c2s` is very slim and reflects the generic architecture: It has only one method for event propagation and three methods for group management. We decided to define only a single, generic method for event operations since the main part of processing will be the same for all kind of events. Thus it is much easier to handle threads of control and related transaction processing within the service.

```
interface c2s {
  short addToNewGroup(in long commNr,
                      in short groupId,
                      in string20 groupName,
                      in short memberId,
                      in string20 memberName);
  short addToGroup( in long commNr,
                    in short groupId,
                    in string20 groupName,
                    in short memberId,
                    in string20 memberName);
  void removeFromGroup( in long commNr,
                        in short memberId);
  short sendData( in long commNr,
                  in long msgNr,
                  in dataMap data);
  // ... additional methods for initialization
};
```

```
interface s2c {
  void addNewGroup  ( in short groupId,
                      in string20 groupName,
                      in short memberId,
                      in string20 memberName);
  void addToGroup   ( in short groupId,
                      in short memberId,
                      in string20 memberName);
  void removeFromGroup ( in short memberId);
  short sendData    ( in long commNr,
                      in long msgNr,
                      in dataMap data);
};
```

```
// Data Structures used by TOGA

typedef string<20> string20;

enum flagMap{ SHORT_TAG, USHORT_TAG,
              INT_TAG, UINT_TAG,
              LONG_TAG, ULONG_TAG,
              CHAR_TAG, FLOAT_TAG,
              DOUBLE_TAG };
union partMap switch (flagMap) {
  case SHORT_TAG  : short dataShort;
  case USHORT_TAG : unsigned short
                    dataUShort;
  case INT_TAG    : short dataInt;
  case UINT_TAG   : unsigned short
                    dataUInt;
  case LONG_TAG   : long dataLong;
  case ULONG_TAG  : unsigned long
                    dataULong;
  case CHAR_TAG   : char dataChar;
  case FLOAT_TAG  : float dataFloat;
  case DOUBLE_TAG : double dataDouble; };
typedef sequence<partMap> dataMap;

struct memberMap{ short id;
                  string20 name; };
typedef sequence<memberMap> memberList;
struct groupMap { short id;
                  string20 name;
                  memberList members; };
typedef sequence<groupMap> groupList;
```

Fig. 4. IDL Interface of the TOGA Service (server and client components)

Interface `s2c` is similar to `c2s` since events will be directly propagated from the communication layer to the application layer, except for group events (related to changes of a group composition) that will be processed within the session layer.

The overall architecture of TOGA ensures that participating applications will act as event supplier and event consumer. Thus it realizes a symmetric event model. Furthermore, from a logical point of view, the TOGA server is a central server that is responsible for all known applications based upon the architecture displayed in Figure 3. Nevertheless, the actual implementation may consist of several, distributed servers that use additional protocols to achieve consistency of operations (i.e. defining a global state of group compositions).

2.6 Processing Example

A processing example of a possible initialization phase is presented in Figure 5: Application 1 (see left hand side) initializes its session (thereby choosing the TOGA communication module), retrieves all information describing current groups and members and requests to become a member of one group. Processing this request, the TOGA server retrieves all data related to this group from application 2 (which is already a member of that group), propagates it to application 1, notifies all members of that group (including the new member) that the group composition has changed and replies that the operation has succeeded.

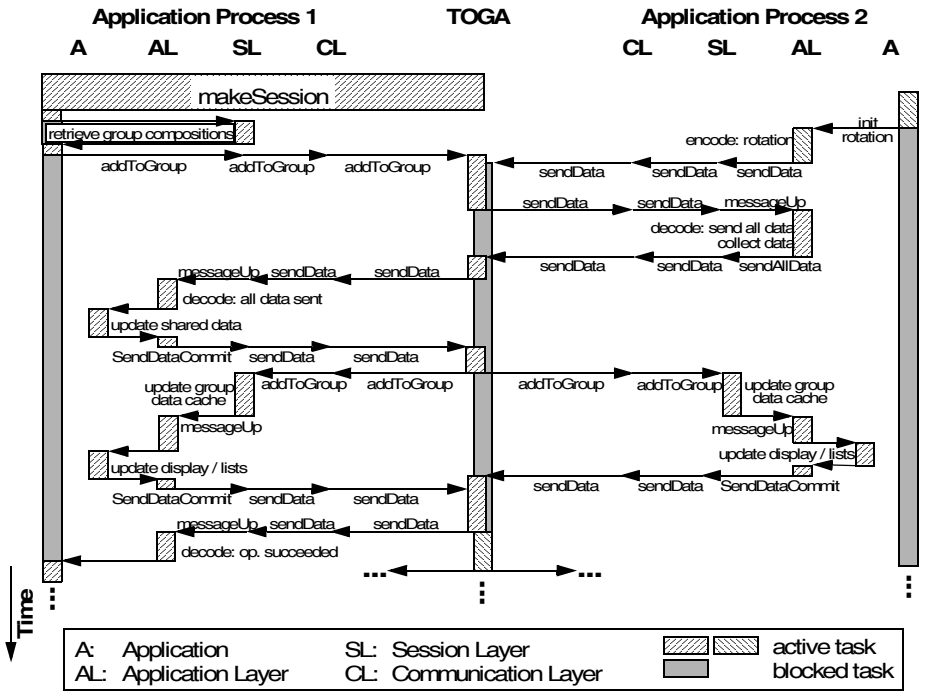


Fig. 5. Processing example of the initialization phase

Figure 5 contains also a request for a rotation operation of application 2 (see right hand side). Since the request of application 1 refers to the same group, the operation request of application 2 is automatically deferred until the first request is processed. If both operations would have been processed in parallel, and the rotation would have been executed by application 2 before it replies to the request for all data, and application 1 would have received the rotation request after having received all data, then the rotation would have been performed twice on the data of application 1, but only once on the data of application 2. Of course, more detailed synchronization protocols (preventing this problem) are possible within our architecture. E.g. the TOGA server might "know" that the rotation has already been performed on the data, and therefore the rotation event must not be propagated to application 1. As one can see, the granularity of shared data has to be well-defined in order to ensure a scalable architecture.

Another example describing the overall processing for a successful rotation operation is presented in Figure 6. Any operation is performed due to a received event, even within the initiating application process. First, this enables the usage of multicast protocols instead of multiple operation invocations. Second, it simplifies the application (otherwise the application code would have to distinguish between user-requests and events).

Considering a third application that could not perform the requested operation, and thus votes abort, application 1 and 2 would have to undo the already performed operation. This would happen instead of the "no operation" displayed in the lower right part of Figure 6.

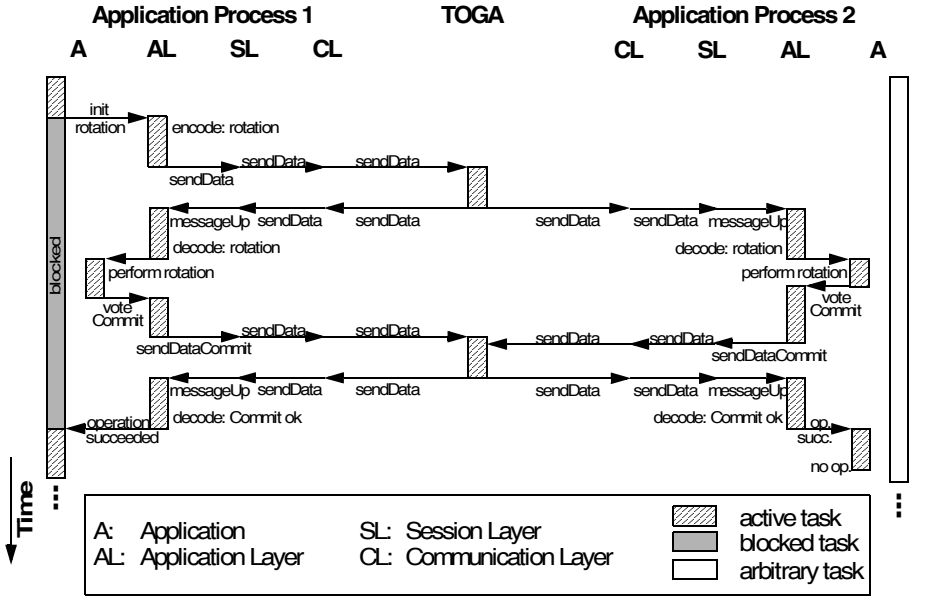


Fig. 6. Processing example of a successful rotation operation

3. Relationship to Existing Technology

TOGA was developed since other approaches don't fulfil the full set of requirements that are associated with a consistent and distributed collaboration in a heterogeneous environment by means of common information spaces, especially for concurrent engineering (see Section 1). TOGA can manage shared data objects consistently among the possibly heterogeneous applications that belong to a group. This leads to a component-based approach showing tight coupling, high group awareness, and consistent work data. Nevertheless, since there exist many different mechanisms for communication and collaboration in distributed environments, we want to show how they relate to our approach.

3.1 Comparison to Existing Communication Services

The simplest approach of communication is defined by the primitives send/receive [29]. It can be used to build higher level communication systems, as e.g. the TOGA service.

Messaging and queuing systems offer functionality to handle messages between two parties in a distributed environment, without being linked by a private, dedicated, logical connection [18]. Thus this approach enables asynchronous communication as used for e.g. bulk message processing or mobile computing. In contrast, our approach ensures communication for synchronous work among a given set of parties and in dedicated Intranets.

Persistent queues (usually) offer reliable asynchronous message exchange and sometimes they can also ensure the order in which messages are received (FIFO, etc., see e.g. JPMQ [26]). This is actually more closely related to our approach which offers guarantees in event and data handling as well as synchronization of events. Message queues could be one way to implement TOGA, with high level functionality like event and group management still to be added.

Since TOGA was implemented using CORBA, we want to show how it relates to existing *CORBA Common Object Services*. The *CORBA Event Service* offers event channels to suppliers and consumers of events. Both can subscribe to such a channel in order to receive or initiate events, either using the push or the pull model. Since the pull model is somehow comparable to polling, and thus inappropriate for our application scenario, we will look only at the push-based event channels. There are generic events which have to be interpreted by the application, and typed events which are defined via IDL. A generic event channel with push consumers and suppliers is similar to our approach. However, in TOGA the group composition is known and accessible to all group members and additional group functionality is available (see Section 2), which is not the case for the *CORBA Event Service*, where suppliers and consumers are connected to the event channel without knowing of each others existence [14], [19]. Furthermore TOGA clients are both suppliers and consumers (per definition), whereas clients connected to an event channel can be suppliers, consumer, or both. Last but not least, it would take considerably more effort to implement a comparable service with event channels, since more service logic has to be implemented. In particular, it is necessary to provide implementations for `PushConsumer`, `ProxyPushConsumer`, `ConsumerAdmin`, `EventChannel`, `PushSupplier`, and `SupplierAdmin`. All in all

the *Event Service* shows less transparency of event processing and much more administration overhead. Finally, the *Event Service* doesn't (necessarily) support transaction-like behavior, order of events, and lacks further guarantees (see e.g. [19]).

Transaction-like handling of objects in a CORBA environment could be achieved by deploying the *CORBA Event*, *Object Transaction* and *Concurrency Control Services*. One major drawback is that the objects to be handled, have to be first class CORBA objects, which causes a lot of overhead (see [19], [24]). In addition, the relationship between applications, events, event channels and transactions is difficult to model. We prefer TOGA as a low level service, which might be used to build an advanced object model adapted to a particular design scenario on top of it.

3.2 Comparison to CSCW Technologies

Since we aim at supporting distributed collaborative work environments, it is obvious that we offer functionality that can be found in CSCW (*Computer Supported Cooperative Work)*.

Workflow Management Systems realize process coordination and enactment especially for asynchronous work. Our approach enacts (ad hoc) coordination of synchronous work on shared data objects, but not directly the flow of work activities, since it is not process-oriented.

TOGA more closely resembles a CSCW service for synchronous distributed applications such as e.g. multi-user editors ([11], [22]). CSCW focuses on issues like groups, group awareness, human interaction and common information spaces in order to support cooperation. TOGA does not include support for groups of people like a generic group service would, but groups of applications, where one group is characterized by sharing a common set of data objects. This is due to the data-centric approach of this service and thus it is not user-centric. So we clearly focus on common work and information spaces. In doing 2PC combined with a push model, we also allow for an accurate group awareness, since all actions of a member are directly reflected in all group member applications (see Section 2). This is one of the major points where we differ from approaches like [10] and [27], which support cooperation of asynchronously and possibly disconnected users. To support these users, techniques like versioning, replication, inconsistency detection, and merging are being applied. Since we support synchronous work, such mechanisms become obsolete. We have to cope only with short periods of inconsistency while a commit phase is going on. Afterwards all common data objects are again synchronized, which is the so-called quiescent phase as described in [4]. But in contrast to TOGA, the algorithm presented in [4] relies on transforming conflicting operations to reach the convergence of the shared objects, while TOGA shows a more generic and transaction-oriented behavior. There exist other approaches, which developed formal models to handle concurrency control in synchronous applications (mainly multi user editors), e.g. [16],[21]. Especially [28] seems to be a very comprising work. Numerous criteria have been defined and corresponding algorithms presented in order to support controlled concurrent work. TOGA only offers the basic services to couple different applications by customizing information spaces and the necessary operations. Though it would be a good idea to implement the before-mentioned algorithms into the application layer. For example, one could implement the

functionality of the ORESTE-Algorithm (Optimal RESponse TimE, [1]), which is used in synchronous groupware tools (i.e. GroupDesign, [1]).

Another service for distributed collaborative work is realized by the Corona Server [9] and the DistView [20] package. DistView is a toolkit for synchronizing views at the granule of single GUI windows, which is generally coarser than what can be achieved with our architecture. The Corona server was built to support publish/subscribe (P/S) and peer group (PG) communication. The P/S paradigm is related to the CORBA event channel notion (see above). The PG functionality supports group communication and can distinguish between different roles for users (principals and observers), whereas there is no difference for application users in TOGA, since it focuses on symmetrically propagating work activities which are relevant to all applications and doesn't implement direct inter-person communication support (e.g. for email or chat). Furthermore in [9], P/S and PG can be distinguished by means of anonymous versus named communication. TOGA lies in between because the TOGA protocols allow to access and propagate group (composition) information, but on the other hand it broadcasts all events regarding the shared data.

At last we want to point out some differences to GroupKit ([8], [23]), which is a Tcl/Tk-based toolkit offering core technology for implementing synchronous and distributed conferencing systems. In contrast, TOGA is a service for already existing applications. One of the GroupKit core technologies is the support for shared data. Concurrency control is only rudimentarily implemented, since it has to support a wide range of applications (see also [7]). Thus it might be necessary to implement a transactional protocol, like the one of TOGA, for GroupKit. Data sharing and synchronized views are enabled through the choice between three abstractions: multicast RPC, events, and environments. The environment concept is closest to the information space aspect of TOGA. It uses a structured model for name/value pairs. Users can register, to be notified when any changes have been made. However, there is no voting phase as in TOGA to ensure the validity of the modifications.

Because TOGA centers on shared, distributed data it might be interesting to compare it to federated database technology. However, TOGA does not provide a common data storage facility. Instead, it synchronizes (by means of coordination of the operations) the state of only those data objects that are part of the common information space.

4. Prototype Implementation

A prototype implementation has been built to prove the concepts presented in Section 2. It consists of a single TOGA server and several clients that include a simple test application as described in Section 4.1. All components are written in C++ and have been tested on a cluster of SUN Ultra 1 workstations, connected via a 10 MBit Ethernet. The underlying CORBA system is IONA's Orbix, version 2.3MT.

4.1 Test Application

A simple application has been developed to test the functionality and performance of the TOGA server as well as its related client layers. Collaborative work is simulated by

5 coins that can be moved on a fixed desktop. If a single user performs a move operation (using the mouse), our system ensures that this operation will be propagated to all other applications which belong to the same group. If this move operation is in conflict with an operation performed by other group members, it will be undone. Please note that our desktop with the 5 coins symbolizes the common information space and the movements resemble the design operations to simulate a collaborative work scenario on shared data objects. For example, in the real world scenario of Figure 1, the information space for the "engine construction" comprises objects like transmission (as depicted), gear, and motor.

4.2 Client Layers

The application layer as described in Section 2 has been integrated into the application code. The session and communication layer are realized by a class hierarchy that omits unnecessary method implementations. More precisely, the session layer contains only declarations of virtual methods that are implemented by the communication layer. Upcalls from the TOGA server are processed by the communication layer and either directly propagated to the application layer (events due to application operations) or further processed within the session layer (caching of group data).

4.3 TOGA Server

Implementing the prototype, we decided to build a single, central TOGA server only. Though this might reduce scalability, it omits additional protocols for synchronization and distributed event handling. Parallel processing of events can be enabled by choosing the appropriate thread mode of the TOGA server: Using mode 1, the entire server process is single threaded. Mode 2 shows one thread per group and mode 3 refers to one thread per event.

4.4 Experiences

In addition to the graphical application described in Section 4.1, we have developed a script-based client. It has been used to initiate a well defined number of events for measurements. Since a workstation has only a single mouse and keyboard, each graphical application has been run on a separate host (guaranteeing a realistic scenario). The TOGA server and script-based clients have been run on other hosts.

In a first test, we have determined differences between thread modes, thereby using three groups with three applications[3]. In theory, assuming linear scalability, the throughput of each mode should be three times higher than before. In practice, we measured a slightly lower factor (appr. 2.5). Since the server has been run on workstations with a single CPU, threads can only be used for parallel communication, but not for parallel execution of code within the TOGA server. Thus we conclude that the different thread modes achieved very positive and acceptable results.

The next test has been set up to measure scalability related to the number of applications per group (using thread mode 3). It turned out that the throughput

[3] One script-based application to initiate events and two passive, graphical applications that receive events.

decreased by 10-15% for a group that had five instead of a single member. Once again, we think that this is a promising result.

Finally, please keep in mind that pure system measurements do not provide a comprehensive assessment of TOGA. In real-world scenarios it is not expected that applications initiate events every millisecond. The main factors comprise thinking time of the user, processing time for operations and undo operations, as well as the amount of conflicting actions that result into conflicting events. The acceptance discussion of TOGA will not be decided w.r.t. the performance at the event level (as e.g. number of events processed per second), but w.r.t. the effectiveness and appropriateness of the semantical support for collaboration.

5. Summary and Outlook

TOGA has been developed in order to meet the demand for coupling existing engineering applications. Thus the TOGA service provides a particular technique towards component-oriented system engineering. It offers customizable group management and collaboration facilities to stand-alone application systems in order to set up a cooperative information system. Our collaboration approach is characterized by the coordination of synchronous work on a common information space. The collaboration model can be adapted to application needs by specifying first, the shared data objects that comprise the common information space, second, the operations that manipulate these data, and third, the conflict as well as conflict-resolution scenarios (i.e. undo algorithms). This conceptual flexibility is further enhanced by its layered and modular architecture that enables implementation flexibility as multiple communication protocols and basic transaction properties can be easily supported. Furthermore, these flexibility issues are the primitives towards extensibility. As discussed in Section 3 it is a platform to implement sophisticated algorithms for concurrent work as needed for certain environments.

We are currently investigating into several directions. One refers to extensions to the collaboration model. We view TOGA as a basic layer that provides the necessary functionality to build higher-level collaboration features as e.g. hierarchical groups referring to hierarchically organized information spaces, or more sophisticated conflict management functionality. Another direction of further work concentrates on implementation issues like multi-protocol communication, multi-threading, and other performance enhancing measures. The third direction of current and future research deals with questions on how to efficiently integrate the TOGA service and its enhancements into different kinds of application systems. Here we want to gain more knowledge on how to customize TOGA and how to integrate it via its application layer to existing application systems. So far, we concentrated on a modular and extensible architecture and therefore only hand-crafted this customization, but in the long run a methodology and clear model seems to be useful or even necessary. Last but not least, we want to employ TOGA within typical application areas of workflow and CSCW systems as for example business process applications or multi-user editors. Therewith, we want to investigate whether a system like TOGA can be used as a basis to a new generation of systems that integrate workflow and CSCW issues.

References

[1] M. Beaudouin-Lafon, A. Karsenty: *Transparency and Awareness in a Real-Time Groupware System*, Proc. of the ACM Symposium on User Interface Software and Technology (UIST'92), Monterey, CA., 1992.
[2] D. E. Carter and B. S. Baker: *Concurrent Engineering: The Product Development Environment for the 1990's*, Addison-Wesley Publishing Company, New York, NY, USA, 1991.
[3] D. Gerwin and G. Susman: *Special Issue on Concurrent Engineering*, IEEE Transactions on Engineering Management, Vol. 43, No. 2, May 1996, pp. 118-123.
[4] C. A. Ellis, S. J. Gibbs: *Concurrency Control in Groupware Systems,* Proc. of the ACM SIGMOD 1989, pp. 399-407, Seattle, Washington, USA, 1989.
[5] C. A. Ellis, S. J. Gibbs, G. L. Rein: *Groupware - Some Issues and Experiences*, CACM, 1991.
[6] J. Gray, A. Reuter: *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann Publ., 1993.
[7] S. Greenberg, D. Marwood: *Real Time Groupware as a Distributed System: Concurrency Control and its Effect on the Interface,* Proc. of the ACM CSCW'94, Oct. 22-26, N. Carolina, USA, ACM Press, 1994.
[8] S. Greeenberg, M. Roseman: *Groupware Toolkits for Synchronous Work,* in *Trends in CSCW*, edited by M. Beaudouin-Lafon, Jon Wiley & Sons Ltd., 1996.
[9] R. W. Hall et al.: *Corona: A Communication Service for Scalable, Reliable Group Collaboration Systems,* Proc. of the ACM Conf. on CSCW '96, Boston, USA, 1996.
[10] J. Klingemann, T. Tesch, J. Wäsch: *Enabling Cooperation among Disconnected Mobile Users*, Second IFCIS Intl. Conference on Cooperative Information Systems (CoopIS '97), Kiawah Island, USA, June 1997.
[11] M. Koch: *Design Issues and Model for a distributed Multi-User Editor*, CSCW, Intl. Journal, 5(1), 1996.
[12] R. Mayr: *Entwurf und Implementierung einer Event-Behandlung in CORBA-Umgebungen*, Diplomarbeit, Technische Universität München, 1996 (in German).
[13] Object Management Group*: The Common Object Request Broker Architecture: Architecture and Specification*, Revision 2.2, http://www.omg.org/corba/corbiiop.htm, Upd. February 1998.
[14] Object Management Group: *The Common Object Request Broker Architecture: Common Object Services Specification*, http://www.omg.org/corba/sectrans.htm, Upd. December 1997.
[15] Object Management Group: *Workflow Management Facility, Request for Proposal (RFP)*, http://www.omg.org/library/schedule/Workflow_RFP.htm, May 1997.
[16] J. Munson, P. Dewan: *Concurrency Control Framework for Collaborative Systems*, Proc. CSCW'96, ACM, Cambridge, MA, USA, 1996.
[17] S. Jablonski, C. Bussler: *Workflow Management, Modeling Concepts, Architecture and Implementation*, Intern. Thomson Computer Press, London, 1996.
[18] R. Orfali, D. Harkey, J. Edwards: *Essential Client/Server Survival Guide*, Van Nostrand Reinhold, 1994.
[19] R. Orfali, D. Harkey, J. Edwards: *Instant CORBA,* Wiley Computer Publ., 1997.
[20] A. Prakash, H. S. Shim: *DistView: Support for Building Efficient Collaborative Apllications using Replicated Objects,* ACM Conference on CSCW, Oct. 1994, pp. 153-164, 1994.
[21] M. Ressel, D. Nitsche-Ruhland, R. Gunzenhäuser: *An Integrating, Transformation-Oriented Approach to Concurrency Control and Undo in Group Editors*, Proc. CSCW'96, ACM, Cambridge, MA, USA, 1996.
[22] N. Ritter: *Group Authoring in CONCORD—A DB-based Approach,* Proc. ACM Symposium on Applied Computing (SAC), San Jose, California, February 1997.
[23] M. Roseman, S. Greenberg: *Building Real Time Groupware with GroupKit, A Groupware Toolkit,* ACM Transactions on Computer Human Interaction, 1996.
[24] J. Sellentin, B. Mitschang: *Data-Intensive Intra- & Internet Applications — Experiences Using Java and CORBA in the World Wide Web*, in: Proceedings of the 14th IEEE International Conference on Data Engineering (ICDE), Orlando, Florida, February 1998.
[25] J. Siegel: *CORBA: Fundamentals and Programming*, Jon Wiley & Sons, 1996.
[26] H. P. Steiert, J. Zimmermann: *JPMQ—An Advanced Persistent Message Queuing Service*, in: Proc. of the 16th British National Conference on Databases (BNCOD16), Cardiff, July 1998.
[27] R. Sturm, J. A. Müller, P. C. Lockemann: *Collision of Constrained Work Spaces: A Uniform Concept for Design Interactions*, 2nd Intl. Conf. on Coop. Information Systems (CoopIS '97), Kiawah Island, USA, 1997.
[28] C. Sun, X. Jia, Y. Zhang, Y. Yang, D. Chen: *Achieving Convergence, Causality Preservation, and Intention Preservation in Real-Time Cooperative Editing Systems*, ACM Transact. on HCI, Vol. 5(1), 1998.
[29] A. S. Tanenbaum: *Computer Networks*, 3rd Ed., Prentice-Hall, Amsterdam, 1988.