NEW TECHNOLOGIES FOR REALIZING MODEL-BASED BATTLE COMMAND

Dr. James Snyder
Dr. Patrick A. Muckelbauer
Carlos Henriquez

Lockheed Martin Advanced Technology Labs
1 Federal Street, A&E 3W
Camden, NJ 08102

Prof. Ouri Wolfson

Dept. of Electrical Engineering and Computer Science
University of Illinois
Chicago, IL 60607

ABSTRACT

Model-Based Battle Command has been proposed as an alternative way to realize information distribution in lower echelon environments where network bandwidth is constrained. We show why traditional DBMS technology is not well-suited and present results of an alternate approach. We provide an example scenario to compare and contrast the DBMS approach to our new approach as well as describe a software prototype that realizes the presented scenario.

INTRODUCTION

A principle purpose of Factor 3a is to investigate approaches to distributing information in lower-echelon environments where network bandwidth is especially scarce. Model-Based Battle Command (MBBC) has been proposed as a notional, guiding problem/solution description to this information distribution problem and is cited from [2] as:

> ... a different perspective towards building battle command systems, a perspective that focuses on truly exploiting available computational power through *model-based, computationally intensive* paradigms of battle command rather than the current *communications-intensive, message-based* approach. Using this approach, a formal data model of the battlefield serves as the hub of information flow. This is in contrast to the traditional message-based approach where the data model and its container, the database, are viewed as only supporting entities. Under the model-based paradigm, each node maintains its own independent database, reflecting the battlefield situation to the best of its ability. From this vantage point, the task of communications switches from exchanging message to updating each other's databases. Thus, the database provides the conduit, as well as the hub, by which information is transferred between different units and, often, between the applications within the same unit (or system).
>
> The preceding description is easily recognized as that of a distributed database system that is typically implemented above a distributed computing environment. One impetus for development of the model-based approach is to make a true distributed computing environment viable for the lowest-echelon (e.g., on platforms such a [*sic*] tanks, aircraft, or individual warriors) where communication links are often tenuous at best. At these echelons, automating battle command is further complicated by the real-time nature of many of the tasks (e.g., situational awareness, targeting, etc.)

The above perspective lead to the characterization of MBBC being supported by an *Active Info-Base* as illustrated in Figure 1 where Info-Bases exchange updates to information models rather than exchange minimally structured messages. In this context, data replication is the process of synchronizing two Info-Bases. Additionally, the knowledge of what and when to synchronize is moved from the application programs into the Info-Bases.

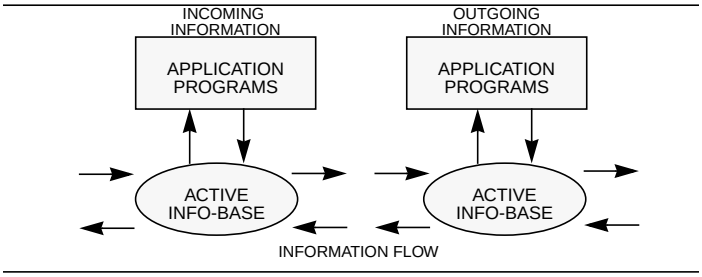

Figure 1: Active Information Base in MBBC (from [2])

WORKING SCENARIO

For illustrative purposes and to serve as a frame of reference for the remainder of the paper, we provide a simple, representative example of an information distribution problem. We have a platoon of tanks where: 1) each member shares position information with the others on a periodic basis according to a situationally aware policy, 2) one tank is designated the platoon leader, and 3) the leader is responsible

Prepared through collaborative participation in the Advanced Telecommunications & Information Distribution Research Program (ATIRP) Consortium sponsored by the U.S. Army Research Laboratory under the Federated Laboratory Program, Cooperative Agreement DAAL01-96-2-0002. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notion thereon.

to report the platoon's aggregate position (i.e. the bounding box of member positions) to the immediate upper echelon on a periodic basis again by a situationally aware policy. Figure 2 is a visual rendering of the information reported by the platoon leader where the shaded box represents the platoon's bounding box.

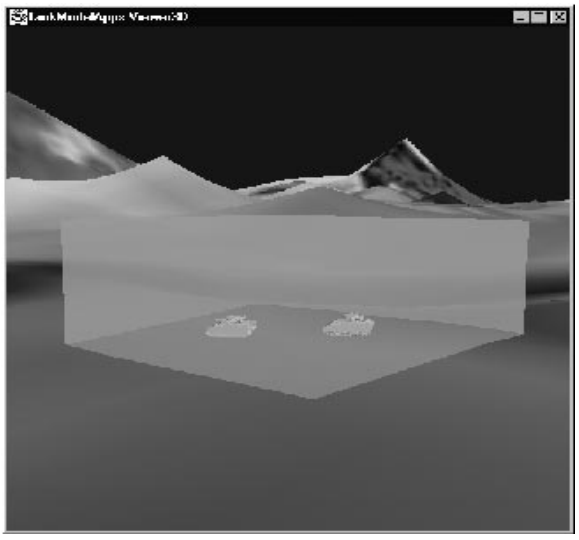

Figure 2: An Example Platoon Bounding Box

We want to realize an Active Info-Base that supports our stated scenario where we have a formal data model that represents the platoon, its bounding box, and the platoon members including positions. Additionally, we want our Info-Base's information model to capture the internal and external information flows, that is, the policy-based periodic synchronization in our scenario. Member position updates are *external* information flows (i.e. between servers), while the aggregation of member positions (i.e. the bounding box) is an *internal* information flow derived within the context of a server. Additionally, this derived computation could be passed to a server in a higher echelon (i.e. an external flow).

ON USING A DBMS TO REALIZE MBBC

Given the above MBBC scenario, it has been proposed to implement such a scenario using RDBMS technologies with triggers or Active Databases [1], [2]. The essential information flow abstraction that these technologies use is an Event-Condition-Action (ECA) model. An event, such as a row insertion into a table, causes the evaluation of the associated condition, and if the condition evaluates to true, the associated action is executed. Significant work has been done to research and develop active databases for use in many kinds of problems. To understand how active databases applies to our scenario we need to characterize how it might be essentially realized in such a technology.

The basic primitives provided by an RDBMS are 1) a named schematic structure (e.g. a database table), 2) a trigger or rule attached to a schematic structure, and 3) a stored procedure. A trigger or rule is described by a three-tuple where the elements are: 1) a database event (e.g. insert or update), 2) a matching condition (e.g. boolean expression or query), and 3) what action to take (e.g. a stored procedure). Given these constructs, we can build an information model that captures the semantics of our working scenario. However, we need to understand the complexity involved in using these constructs. Suppose we represent our working scenario using three DBMS tables shown in Figure 3.

```
create table platoon (
      pid integer PRIMARY KEY,
      lowX integer, lowY integer, lowZ integer,
      highX integer, highY integer, highZ integer
);

create table tank(
      tid integer PRIMARY KEY,
      X integer, Y integer, Z integer
);

create table platoonMembers (
      pid integer NOT NULL,
      tid integer NOT NULL,
      PRIMARY KEY (pid, tid),
      FOREIGN KEY (pid) references platoon(pid),
      FOREIGN KEY (tid) references tank(tid)
);
```

Figure 3: Proposed Database Schema

The schema in Figure 3 essentially represent the information portion of our scenario. To complete the implementation, we need to somehow model the scenario's information *flows*, which is where the complexity starts to be introduced. We must describe the computation of the platoon's bounding box based on the leader's cumulative tank positions. The bounding box depends not only on the individual tank positions but also on the *set* of tanks in the platoon, that is, any time a tank is added or deleted from the platoon or we update tank positions, we must also recompute the bounding box. The triggers and stored procedures needed to realize the derivation is show in Figure 4.

Programming information flows with stored procedures and triggers is complex and error-prone because the dependency analysis is done by hand and relies heavily on "good" software engineering techniques. Observe from Figure 4 that the triggers and stored procedures are defined in terms of the schematic structure of the database tables and not on the contents of the tables (i.e. rows in a table), that is, the logic of the information flows is decentralized. As a result, chang-

```
create procedure InitializeBBox(in pid int) begin end;
create procedure DeriveBBoxByTID(in tid int) begin end;
create procedure DeriveBBoxByPID(in pid int) begin end;

create trigger InsertPlatoonMem after insert on platoonMembers
    referencing new as new_member
    for each row begin
        Call DeriveBBoxByTID(new_member.tid)
    end;

create trigger UpdatePlatoonMem after update on platoonMembers
    referencing new as new_member old as old_member
    for each row begin
        if (new_member.pid <> old_member.pid) then
            Call DeriveBBoxByPID(old_member.pid)
        end if;
        Call DeriveBBoxByPID(new_member.pid)
    end;

create trigger DeletePlatoonMem after delete on platoonMembers
    referencing old as old_member
    for each row begin
        Call DeriveBBoxByPID(old_member.pid)
    end;

create trigger InsertTank after insert on tank
    referencing new as new_tank
    for each row begin
        Call DeriveBBoxByTID(new_tank.tid)
    end;

create trigger UpdateTank after update on tank
    referencing new as new_tank old as old_tank
    for each row begin
        Call DeriveBBoxByTID(new_tank.tid)
    end;

create trigger DeleteTank after delete on tank
    referencing old as old_tank
    for each row begin
        Call DeriveBBoxByTID(new_tank.tid)
    end;
```

Figure 4: Triggers and Procedures for Bounding Box

ing the database schema requires that the logic of derived elements (i.e. the triggers and stored procedures) be manually validated against the changes by finding all affected constructs and validating that they still capture the correct dependencies. We characterize this technique as a coarse-grained solution that is unnecessarily complicated for our working scenario.

To address *external* information flows, a DBMS usually supports some form of selective replication, for example, explicitly defined publish—subscribe elements that allow for loose consistency between tables on different server hosts. The logic for achieving replication is based on the same constructs as in Figure 4 and suffer from the same complexity problems. In our working scenario, supporting external information flows effectively doubles the number of triggers we need.

While we acknowledge that current DBMS technology is suitable for some problems, we believe they are not suited for problems where information and information flows, as in our scenario, need to be treated in a uniform, fine-grained manner. We propose that an alternate set of suitable representational constructs is needed.

AN ALTERNATE APPROACH

To more directly and simply realize our working scenario, we need representational constructs that allow us to express both the information and the information flows in an integrated manner. For example, we need to observe that a specific platoon is composed of specific tanks. To more directly realize the bounding box (an internal flow) from our scenario, we should only have to consider the set of tanks that belong to a platoon unlike the approach using a DBMS. We also want to be able to more directly capture an information-based (rather than a procedural) model of our synchronization policy.

We believe that explicit, distinct constructs are needed to capture internal and external information flows. To this end, we have been leveraging internal LM-ATL research to realize technologies that are alternatives to the DBMS. While more modern database systems, such as the object-oriented DBMS, are being developed, they have not moved far enough away from their RDBMS roots to wholly subsume our scenario. Another way of stating this claim is that “object databases make nice database construction toolkits” [4], [9]. So while object databases can play a significant role in our solution, they are not enough.

Based on our scenario, we want to be able to state two important things. First, we want to be able to state (in a declarative manner) that the bounding box is derived by aggregating all platoon member positions. Secondly, we want to be able to monitor an individual tank such that synchronization occurs between platoon members based on a policy. Figure 5 abstractly shows the kinds of information flows we want.

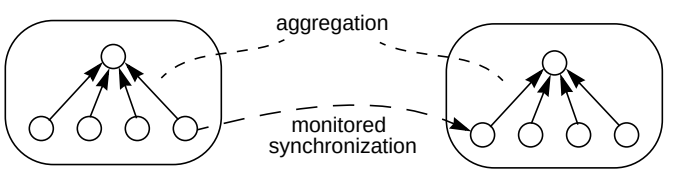

Figure 5: Idealized Fine-Grained Information Flows

We can characterize the above approach as an active object server with fine-grained information flows. A technology that allows us to directly express these notions is called a Conceptual Modeling Environment [6], [7]. This approach allows us to capture a *conceptual* model of both our information and the policies of information flow in manner that is independent of underlying implementation technologies (e.g. an object database or middleware such as CORBA or JavaRMI, etc.). Once a model is developed, information servers can be automatically constructed that are targeted towards specific implementation technologies. Additionally, given such a model, sharing information with (or integrating) existing or new applications into the shared information base can also be automated using formal integration specification languages that complement the shared models.

REALIZING THE WORKING SCENARIO

To bring some concreteness to the notions outlined in the previous sections, in Figure 6 we capture the working scenario in our conceptual modeling language. Note that this representation *wholly* subsumes Figure 3 and Figure 4 and more directly captures our problem *without* introducing unnecessarily complexity by *eliminating* the decentralized logic of the information flow.

Recall that we have a server on each tank with one being designated the platoon leader. Note that each server is using the same information model. Also recall that the objective of each server is to realize a *synchronized* picture of the platoon situation (e.g. the set of positions). There are two sets

```
domain Point tuple ( x, y, z : sprout.lang.UnitlessReal ) end

domain BoundingBox tuple ( lowCorner, highCorner : Point ) end

class Tank specializes sprout.lang.Object
    value position : Point;
end

relationshiptype hasMany set propagate( copy,  delete ) end

class Platoon specializes sprout.lang.Object
    relationship members hasMany( Tank );
    derived zone : BoundingBox is boundingBox( members.position );
    method boudingBox( points : sequence of Point ) : BoundingBox  is
        // the implementation is left as an exercise to the reader
    end
end
```

Figure 6: Conceptual Model of the Working Scenario

of information flows of interest in this example: the set of internal information flows of each model server for calculating the bounding box of the platoon and the set of external information flows for updating position information between servers. The internal information flows are declaratively specified using a *derived* value:

```
derived zone : BoundingBox is boundingBox( members.position );
```

This simple *derived* value declaration *wholly* subsumes the functionality in Figure 4 and captures the complete computation of the bounding box. Additionally, a rather complex set of event notification registrations is automatically deduced. The complexity required by using a traditional DBMS is eliminated because the information dependencies are automatically analyzed by our environment.

The external information flows are characterized within our model as server-to-server replication, which we defined as the ability to selectively replicate objects/composites between a federated set of servers. The behavior of the replication and the semantics of synchronization for given replicated composites is controlled by notification, update, and failure policies. The notification policy defines under what conditions the source notifies the destination. The update policy defines under what condition composites are synchronized with the destination, and the failure policy defines recovery semantics when notification, updating, or subscription (initial setup) fails.

The policies are realized as a set of objects that are declaratively specified in the modeling language. In Figure 7, *Monitors* on the source server specify the notification policy and *Handlers* specify the update and failure policy (see *TankPositionMonitor* and *TankPositionHandler* respectively). Given these policies, replication proceeds in a two step process. First, during notification a predefined set of attributes is copied from the source to the destination and made available to the handler on the destination. The handler then uses this data to decide if the composite should be updated or not. The two step process provides fine-grained information sharing by allowing partial, per attribute updates versus whole composite updates.

In the example model, a very simple replication policy is specified—replicate whenever the position changes. While this is not a very useful policy, it shows how we can succinctly realize them. The modeling language features, namely *values*, *relationships*, *derived values*, *methods*, *constraints*, *monitors* and *handlers*, provide a rich set of constructs for specifying arbitrary policies. For example, we can specify a policy that switches to a “prediction” mode when a server becomes unavailable. Network state can also be explicitly specified in the information model and can be

```
class TankPositionMonitor monitors Tank
      value oldposition : Point;
      constraint positionTest except( oldposition ) is
            target.position != target.oldposition
      satisfied is
            oldposition := target.position;
            Notify();   // Notify() is a builtin method for all monitor classes
      unsatisfied is
      end
end

class TankPositionHandler handles Tank( position )
      // Notification() is implicitly declared but not the implementation.
      method Notification( position : Point ) is
            Update(); // Update() is a builtin method for all handler classes;
                  // replicate state from the source object/composite
      end
end
```

Figure 7: Working Scenario External Flow Model

supplemented by runtime capabilities supplied by lower-level network information; Factor 3 has coined the term *stack cognizance* to describe this capability. We believe that incorporating network state into the information model will allow us to more comprehensively address the fundamental trade-off between network bandwidth and CPU utilization. Note that the failure policy is conspicuously missing from the example and has not been directly addressed. We have not overlooked this issue but rather have incorporated the modeling of replication policy failure directly into our research agenda.

The above constructs can also be used to support information flows between clients and servers. Figure 2 is a screen shot of just such an application that uses monitors to maintain data dependencies between the 3D model and a tank's information server. These information flows are akin to publish—subscribe mechanisms found in distributed computing frameworks but such systems also suffer from many similar problems found with Active Database triggers, that is, they are too course-grained and unnecessarily complex. Also note that the software prototype is written completely in Java and can be easily demonstrated on any platform that supports the Java3D API.

RELATED WORK

In September 1998, many of the top researchers in the database field produced the Asilomar Report on Database Research that laid the foundation for the next decade of database research [5]. The issues described in the Asilomar report echo many common themes of Model-Based Battle Command particularly with respect to distributed database updates (e.g. large-scale trigger systems) and database awareness of applications.

Although the past decade of database research has produced significant breakthroughs, current research is focused on producing what has been termed "delta-X" research, that is, research based on incremental refinements to what has come before. This kind of research is often focused on short-term goals whose aim is to improve some widely understood idea. The net result of the Asilomar workshop was to set a significantly different agenda for database research that moves away from the "delta-X" research goals towards new areas.

For some future capabilities to be realized, sophisticated data models and applications must be supported inside the database, thereby, requiring sophisticated database services. For example, the unification of application logic and data storage in current database technology is at best an afterthought (e.g. triggers and stored procedures). While the incorporation of programming systems into the database server (e.g. putting the Java VM in a database) is moving in the right direction, progress needs to be maintained. Even given this progress, we still rely far too heavily on "good" software engineering techniques and attention to programming languages, and as a result, database systems need to become more problem-aware. The Asilomar authors characterize this problem as "code is not a first class object and is not co-equal to data in current database systems."

Of particular interest for our work is the characterization in the report that existing DBMS-based trigger solutions are not scalable and are an interesting, open area of research. Additionally, the Asilomar authors state that "it is incumbent on the database community to help evolve these models to support types and procedures well-integrated with the database system." We believe that our approach is an important step down this path.

FUTURE WORK

In resource-poor environments, the separation between the network layer and the information server (e.g., a database management system), which is assumed in all existing work, is no longer appropriate. For example, when replicating data, the assumption is that the information server tells the network the data to replicated, and is notified when the transmission is completed successfully. This may be inappropriate in resource-poor environment since the message itself may need to change if it has to be retransmitted several times, or if it takes a relatively long time until it is transmitted successfully. Similarly, often the information server

needs to be aware whether the network has a broadcast capability or not; its decision on whether to do lazy or eager replication may depend on this fact. As another example, when bandwidth is plentiful the information server may compute and transmit a more refined polygon that delimits the platoon, rather than a simple bounding box; and when bandwidth is scarce, a simple bounding box will have to be used.

On the other hand, the separation between the network and the information server layers are motivated by "good software engineering practices". The challenge is to find the modeling constructs that make system development as natural and modular as possible, while at the same time eliminating the barrier between the network and information server layer.

The notification, update, and failure constructs constitute a step in this direction. In future work we will develop a full set of constructs for information sharing in resource-poor environments.

Specifically, we plan to leverage Ouri Wolfson's work on databases for tracking mobile units[3]. In this work, he developed an information cost model for analyzing tradeoffs between the value of the data and the cost of communication. His cost model can be used to help minimize overall information cost where the information cost is the sum of the cost of communication plus the cost of not sending the data. In [3], he presented three different dead-reckoning update policies and analyzed them using his cost modeling approach:

> ... a dead-reckoning update policy for a moving object *m* dictates that at any point in time there is a database-update threshold *th*, of which both the DBMS and *m* are aware. When the deviation of *m* reaches *th*, *m* sends to the database an update consisting of the current location, the predicted speed, and the new deviation threshold **K**. The objective of the dead reckoning policies that we introduce in this paper is to set **K** (which the DBMS installs in the *L.uncertainty* subattribute), such that the total information cost is minimized.

We envision two broad areas of research based on the above work. First, we will use the information cost model approach to investigate replication policies as described in our working scenario. As a consequence, we may adapt several existing update policies from [3]. Second, we will incorporate explicit predictive policies that may include modeling uncertainties associated with information.

REFERENCES

[1] Chamberlain, S. (1995). "Model-Based Battle Command: A Paradigm Whose Time Has Come" *Proceedings of the 1995 International Symposium on Command & Control Research and Technology.* National Defense University; 19-22 Jun 95.

[2] Chamberlain, S. (1996). "Resilient Data Replication Mechanism for Battle Command Systems" *Proceedings of the 1996 International Symposium on Command & Control Research and Technology.* NPGS; 25 Jun 95.

[3] Wolfson, O., Jiang, L., Sistla, A. P., Chamberlain, S., Rishe, N., and M. Deng (1999) "Databases for Tracking Mobile Units in Real Time" in *Joint Proceedings of the 3rd Annual Federated Laboratory Symposium*, College Park, MD. Feb. 2-4, 1999. pp. 15-33.

[4] Maier, D. (1998). Personal conversation with J. Snyder, Boston, MA.

[5] Bernstein et al. (1998). *The Asilomar Report on Database Research* (URL: http://db.cs.berkeley.edu/papers/Asilomar_Final.htm)

[6] Snyder and Thaker (1999) "Model-Based Battle Command and Change Propagation in Advanced Database Systems" TTCP Workshop on Data Replication in Low-Bandwidth Wireless Military Environments. April 20-22, Ft. Leavenworth, KS.

[7] Snyder, J. (1998) *Conceptual Modeling and Application Integration in CAD: The Essential Elements*. Ph.D. Dissertation. Department of Architecture and the Engineering Design Research Center. Carnegie Mellon University. Pittsburgh, PA 15213.

[8] Kowalski, V. J. (1995) "The POSC Solution to Managing E&P Data" in W. Kim (ed.) *Modern Database Systems: The Object Model, Interoperability, and Beyond*. ACM Press. pp. 281-301.

[9] Kim, W. (1995) "Object-Oriented Database Systems: Promises, Reality, and Future" in W. Kim (ed.) *Modern Database Systems: The Object Model, Interoperability, and Beyond*. ACM Press. pp. 255-280.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government.