8 Conclusion

UNIFY is a new scheme for maintaining consistency between aggregated and disaggregated levels. It does away with explicit switching between aggregated and disaggregated states, and instead, maintains all state information for all levels. Units can comply with interactions at different levels. Perceivers are responsible for demanding information they need.

We solve the temporal consistency problem by requiring atomic entity interactions. We resolve chain disaggregations because disaggregation is non-existent. We alleviate network flooding by causing fewer entities to be created and making message aggregation simpler. Also, we provide the mechanisms for aggregations of dissimilar entities and dynamic aggregations.

UNIFY requires more memory and CPU cycles, but reduces network traffic. It is not clear how computation of complex Lanchester equations will match speeds at which virtual entities perform interactions, and we recommend research in this area. Existing simulations may not be able to support UNIFY, but this may be an advantageous way to design new aggregate-level simulations. We intend implementing UNIFY to give a proof-of-concept.

9 Acknowledgments

We thank Professor Paul Reynolds in the Department of Computer Science at the University of Virginia for conducting a Distributed Simulations course in Fall 1994. We are grateful to him for his suggestions, guidance and encouragement. We thank the participants of our many discussions in class — Chenxi Wang, Sudhir Srinivasan, Bronis de Supinski, Adam Ferrari and Andrea Salas. We also thank Sally McKee for proofreading.

10 References

[Allen92] Allen, P. D., *Combining Deterministic and Stochastic Elements in Variable Resolution Models*, Proceedings of Conference on Variable-Resolution Modeling, Washington, DC, May 1992.

[Clark94] Clark, K. J. and Brewer, D., *Bridging the Gap Between Aggregate Level and Object Level Exercises*, Proceedings of the 4th Conference on Computer Generated Forces & Behavioral Representation, Orlando, Florida.

[Davis92] Davis, P. K., *An Introduction to Variable-Resolution Modeling and Cross-Resolution Model Connection*, Proceedings of Conference on Variable-Resolution Modeling, Washington, DC, May 1992.

[Davis93] Davis, P. K. and Hillestad, R. J., *Families of Models that Cross Levels of Resolution: Issues for Design, Calibration and Management*, Proceedings of the 1993 Winter Simulation Conference.

[DIS93] DIS Steering Committee, *The DIS Vision, A Map to the Future of Distributed Simulation*, Comment Draft, October 1993.

[DoD94] Under Secretary of Defense (Acquisition and Technology), *Modeling and Simulation (M&S) Master Plan*, Dept. of Defense, September 30, 1994.

[France93] Franceschini, R. W., *Intelligent Placement of Disaggregated Entities*, Institute for Simulation and Training, 12424 Research Parkway, Suite 300, Orlando FL 32826.

[Hill92] Hillestad, R. J. and Juncosa, M. J., *Cutting Some Trees to See the Forest: On Aggregation and Disaggregation in Combat Models*, Proceedings of Conference on Variable-Resolution Modeling, Washington, DC, May 1992.

[Horr92] Horrigan, T. J., *The "Configuration Problem" and Challenges for Aggregation*, Proceedings of Conference on Variable-Resolution Modeling, Washington, DC, May 1992.

[Karr83] Karr, A. F., *Lanchester Attrition Processes and Theater-Level Combat Models*, Mathematics of Conflict, Elsevier Science Publishers B.V. (North-Holland), 1983, ISBN: 0 444 86678 7.

[Karr94] Karr, C. R. and Root, E., *Integrating Aggregate and Vehicle Level Simulations*, Proceedings of the 4th Conference on Computer Generated Forces & Behavioral Representation, Orlando, Florida.

[Robkin92] Robkin, M., *A proposal to Modify the Distributed Interactive Simulation Aggregate PDU*, Hughes Training, Inc., February 28, 1992.

[Sher92] Sherman, R. and Butler, B., *Segmenting the Battlefield*, Loral WDL, June 9, 1992.

[Smith94] Smith, R., Mystech Associates, *Invited speaker to the Department of Computer Science, University of Virginia*, December 1, 1994.

[Stein94] Steinman, J. S. (Jet Propulsion Laboratory, California Institute of Technology) and Wieland, F. (Naval Research Laboratory), *Parallel Proximity Detection and the Distribution List Algorithm*.

[Weat93] Weatherly, R. M., Wilson, A. L. and Griffin, S. P., *ALSP - Theory, Experience and Future Directions*, Proceedings of the 1993 Winter Simulation Conference.

6.2.1 Lanchester equations

Lanchester equations are differential equations that calculate attrition between aggregate entities [Karr83]. They are easily computed in their simplest form, but in order to model the capabilities of the aggregated entities better, a number of factors are added to the equations, which make them more time-consuming to compute. It is then possible to create scenarios where the aggregate units might have an inconsistent view of each other. Consider platoons A_1 and A_2, and an aircraft fighter T. A_1 and A_2 interact at the platoon level, but T interacts with A_1 at the tank level. A_1 may communicate its current strength to A_2, which computes attrition on A_1 using Lanchester equations. In the meantime, T may inflict significant damage on A_1. T may have much simpler interactions with A_1 (e.g., "blow up a tank"), and these may occur much faster than A_2's interactions with A_1. By the time A_2 finishes its equations, the results will be quite meaningless for A_1 because the computations were performed with data that is now stale.

This problem may be generalized by restating it as one occurring when simulations at different levels proceed at time steps that are orders of magnitude different. We feel it would be beneficial for future research to focus on finding cheaper alternatives to Lanchester equations, and work towards making multi-level simulations interact in more compatible time steps.

6.2.2 Legacy simulations

Over the years, substantial investment has been made in producing simulation programs that are, unfortunately, incompatible with each other. ALSP [Weat93] presents a framework for linking unlike simulations. But a large number of simulations were intended to be stand-alone and continue to be so. Increasingly, the view is that different simulations should be able to work together [DIS93]. There have been two initiatives towards this goal. One has been to make existing simulations — legacy simulations — work together. The second has been to devise standards for all future simulations, wherein interoperability is a requirement and not an afterthought [DoD94]. Existing aggregate-level simulations might be changed elegantly to support UNIFY, but we see our approach largely as supporting the second initiative.

7 Cost of UNIFY

We now address the cost of implementing UNIFY in terms of network, memory and CPU requirements. A detailed analysis will be presented in a future publication.

7.1 Network cost

UNIFY alleviates the network flooding problem by reducing the number of messages. This is due to the reduced number of active entities, i.e., entities that send and receive messages, the possibility of aggregating messages and by doing away with complex aggregation/ disaggregation protocols.

The worst case is if the underlying network is a broadcast network and an aggregated unit conducts interactions such that it sends messages about its aggregated attributes and *all* its constituent entities. In other schemes [Karr94], the unit would disaggregate, and all the DEs would send messages. In UNIFY, there would be messages for each entity and one more for the aggregated attributes. We reason that this is a pathological case. The worst case can be ameliorated by message aggregation, and the "overhead" of the aggregate attributes is small compared to the usual number of entities that are aggregated. In addition, interactions between sub-entities within the same unit would *not* generate network messages in our scheme, whereas they might in other schemes.

7.2 Memory requirements

UNIFY requires memory for the aggregate level attributes *and* each of the sub-entities. Other schemes require memory for only one level of aggregation at any given time. If n_i is the number of i-level entities per $i+1$-level entity, the memory requirements for a memory-efficient traditional scheme and UNIFY would be $O\left(\prod_{i=1}^{l} n_i\right)$ and $O\left(\sum_{j=1}^{l}\prod_{i=j}^{l} n_i\right)$ respectively, where l is the number of levels of aggregation. The constants for UNIFY are expected to be large since we need to store more data per entity in order to maintain consistency. We estimate that UNIFY's memory usage would be 2 to 5 times that of the most memory-efficient traditional scheme. Expending memory to achieve consistency and efficiency is acceptable.

7.3 Local CPU requirements

CPU requirements are also higher in UNIFY. In addition to all simulation activities, the CPU expends cycles maintaining consistency between levels. This involves ensuring that when a message arrives at one level, the attributes at that level are changed and compatible changes, if necessary, are made at other levels. With CPU speeds increasing, this will not be a significant bottleneck. Even with UNIFY implemented, the network will remain the bottleneck for a long time.

state information for these tanks in a single message to T. This causes fewer long messages to be sent rather than many short messages, thus increasing throughput.

Note that UNIFY permits aggregation of dissimilar entities. Either during the course of the battle or prior to the battle, certain dissimilar entities could be logically grouped. This merely entails the creation of the new AE and making the required DEs its fields. As required by the scenario, the DEs may now be dissociated from their previous AEs or may be a part of them.

UNIFY allows the perceiver to decide how it perceives entities because this mimics real-life situations, where the perceiver has the best knowledge of its own sensory capabilities. Also, the perceiver takes the responsibility for increased network traffic, computation and display processing required if it requests the perceived entity be presented at a finer level of detail. Lastly, this approach is conceptually scalable: the perceived unit requires no knowledge about the perceiver. It must only deal with requests about varying levels of aggregation — levels that the unit initially advertised to the rest of the simulation. Thus, new types of units could be added to the simulation at a later date, with minimal effect on the existing types.

6 Pros and cons

6.1 Advantages

In addition to solving or reducing the problems presented earlier, UNIFY offers the additional advantages outlined below. The effect of these advantages may not be readily apparent in battlefield simulations, but we believe that there exist other simulations for which these may find application.

6.1.1 Configuration Problem

The configuration problem is presented in [Horr92]. Aggregation causes some information, such as configuration, to be partly or completely lost. A purely mathematical approach towards modeling units does not take the limits imposed by configuration into account. By retaining all information in UNIFY, we ensure that the configuration problem can be tackled. The configuration of the unit's sub-units can be stored as part of the representation, or can be re-created from the attributes of the sub-units.

6.1.2 Dynamic aggregation

UNIFY supports the concept of dynamic aggregation. During the course of the simulation, certain dissimilar entities may be grouped together in a logical fashion. Current aggregation schemes do not handle this possibility well. In UNIFY, this can be done by creating a new unit and making the individual entities that are to be aggregated part of this new unit. The individual entities may or may not be dissociated from any other unit that they were part of depending on the semantics of the simulation. For example, if one were simulating a machine assembly, one could either simulate each component of the assembly or decide at run-time that certain sub-assemblies may be simulated as an aggregate. This could be done by making the sub-assembly a unit and having the components that make it up as data structures inside this unit.

UNIFY requires that the data structures and mapping functions for the dynamically-aggregated unit be in place before the simulation begins, but the instances of these entities could be created only when desired. This form of dynamic aggregation is not quite as powerful as the ability to create new *types* of units during run-time. While the latter can be implemented using UNIFY, it is a harder problem because the mapping functions between the attributes at different levels would also have to be created dynamically.

6.1.3 Information degradation

One may want to model factors that might cause information and/or requests between entities to be somehow affected. Since UNIFY allows a very flexible coupling between units, incorporating information degradation into a simulation is simplified. Each unit would receive a message from a designated sender (which may or may not be the originator of that message), and then respond to that message.

6.1.4 Localized operations

There is a hierarchy of networks — none (single CPU), interconnection networks (multiprocessors/ multicomputers), LANs, WANs — over which a unit may be simulated. UNIFY encourages simulating a unit locally. For example, on a single CPU, consistency across levels can simply be maintained by having the CPU expend additional cycles. In case of non-local units, the costs of maintaining consistency increases as one moves up the network hierarchy. In schemes where an AE disaggregates into many new DEs, consistency is expensive since the DEs send messages to each other across the network (at the highest level) to determine "who can see whom" [Stein94]. With CPU speeds increasing more rapidly than network transmission speeds, a CPU-intensive approach is preferable to a network bottleneck.

6.2 Disadvantages

We address some expected criticisms of UNIFY.

4 Related Work

An approach to simulations involving aggregates is Selective Viewing and is criticized in [Davis93]. Here, the simulation is carried out at a higher resolution. When information about a lower resolution is requested, it is generated from the data at the higher resolution. A problem with this approach is that often, the lower-resolution information cannot be generated from the higher-resolution information easily. Also, by simulating at a higher resolution even when not necessary, computing resources are wasted. A key difference between Selective Viewing and UNIFY is that information flow is one-way in the former (high-resolution to low-resolution), but bidirectional in the latter.

Davis' work with Variable Resolution Modeling (VRM) is particularly relevant here [Davis93]. VRM deals with making simulations work at different levels of resolution. There are process hierarchies and these processes may be modeled such that they can be simulated at varying levels of resolution. The sub-processes themselves, at any level, may possess sub-subprocesses, or may be parametrized as constants read from a database. Davis says:

> The hierarchies treated here involve processes, not objects or entities… While hierarchical representation of objects is rather widely valid and natural in combat modeling, straightforward hierarchical modeling of processes is only sometimes feasible. More generally, the relevant processes have a complex relationship to each other, with connections across branches of the hierarchical tree and, in some cases, with iterations or cycles of data flow.

Our approach is concerned with finding the relationships between aggregate and corresponding disaggregate attributes. Admittedly, the relationships may not be simple accumulative or distributive functions, but knowledge about the applications would help find these relationships. It is difficult to put forth general formulae to model these relationships, because different applications require different models. Indeed, at times these models are hard to find [Hill92].

While designing a simulation, not only are process hierarchies important, as suggested by Davis, but so are object hierarchies. We recognize that VRM and UNIFY deal with related issues, and hence are not entirely independent. If a process is simulated at a low resolution, it is likely that the objects in the simulation will also be at a low resolution at that time. Likewise, if a process is being simulated at a very high resolution, the objects also are likely to be at a high resolution.

5 Solutions to the problems outlined

UNIFY solves the temporal inconsistency problem. Individual entities (here, tanks) have their own attributes that may be computed independently or may be derived from global attributes (here, platoon-level attributes). For example, let us address the problem of the position of the tanks.

Some number of tanks in platoon A_1 engage a single tank T (Fig. 3). These tanks change position as necessary to enact that battle. After completing all interactions with T, A_1 behaves purely as a platoon towards other units (since there are no tank-level entities currently interacting with it). A_1's position is simulated at an aggregate level, and the individual tanks' positions are ignored. If another tank entity T′ begins interacting with A_1, A_1 interacts with T′ at the tank level. Let us assume that T′ engages the same tanks as T did. If T′ comes on to the scene soon after T leaves, the previously computed position attributes of A_1's tanks will be used for T′ (with some perturbation if the platoon has moved). But if T′ comes on to the scene much later, then the global position, coupled with doctrine and terrain would be used to position the individual tanks. This makes sense because in a real battle, after reacting to interactions with enemy tanks and completing those interactions, the tanks in a platoon will tend to regroup and preserve doctrinal formation. This is what UNIFY captures. Properly designed, UNIFY can be made to mimic any real-life situation.

Chain disaggregation is a phenomenon in which entities are forced to disaggregate along a chain because the entities with which they are having interactions disaggregate. This is not a problem in UNIFY because there is no concept of aggregation or disaggregation. Each unit decides the level of detail at which it wants to perceive any other unit, and the perceived unit is able to consistently present different views of itself to its perceivers. UNIFY is not a partial disaggregation scheme. In fact, it is not a disaggregation scheme at all since there is no explicit aggregation or disaggregation involved.

We alleviate the network flooding problem by reducing the number of messages in the system. UNIFY does not cause entities to unnecessarily disaggregate, thereby reducing the number of entities in the simulation. This means that there are fewer receivers and senders of messages. By putting the burden of specifying the required level of perception detail on the perceiver, we force the perceiver to take responsibility for the level of detail it wishes to see. Also, we can reduce the load on the network by message aggregation. When tank T interacts with platoon A_1's tanks, A_1 knows which tanks are of interest to T and can pack the

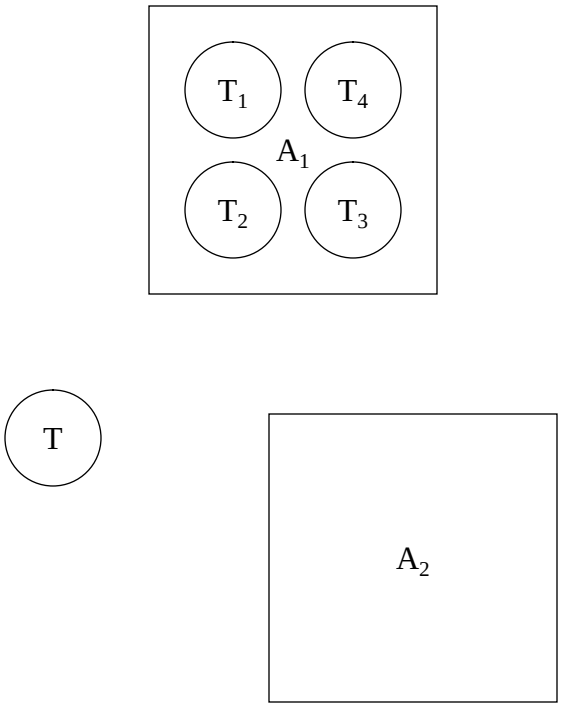

Figure 3a - Entities in a simulation

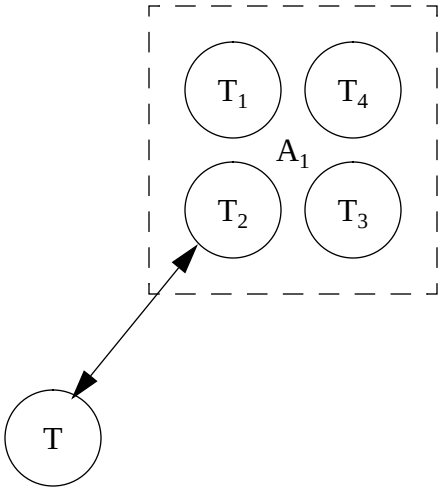

Figure 3b - T's view of the entities

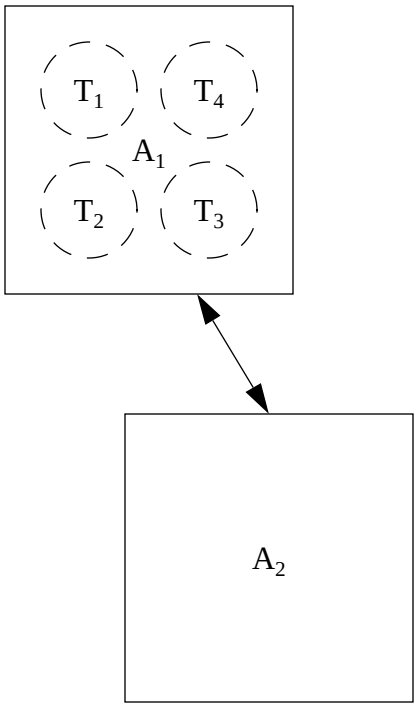

Figure 3c - A_2's view of the entities

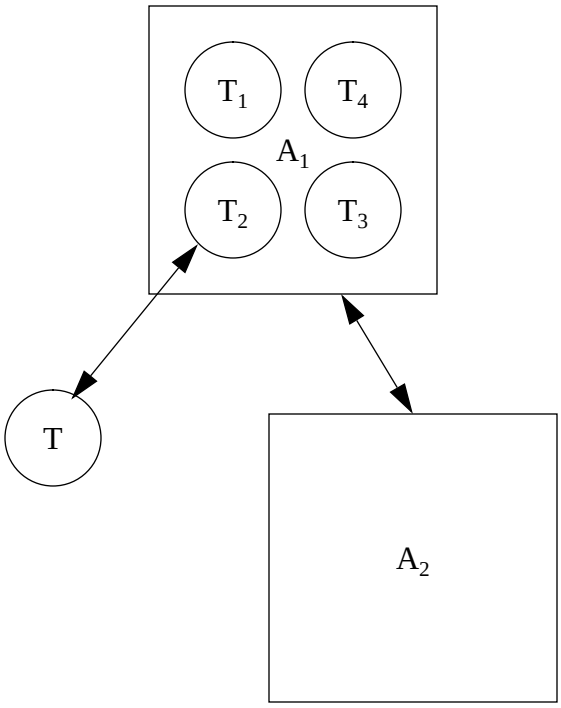

Figure 3d - A_1's view of the entities

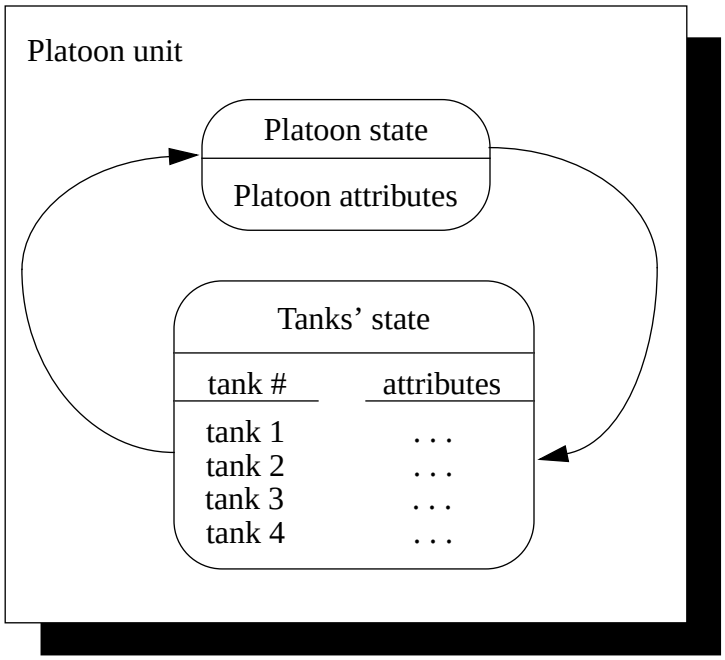

Figure 1 - Higher-Level Unit composed of Lower-level Entities

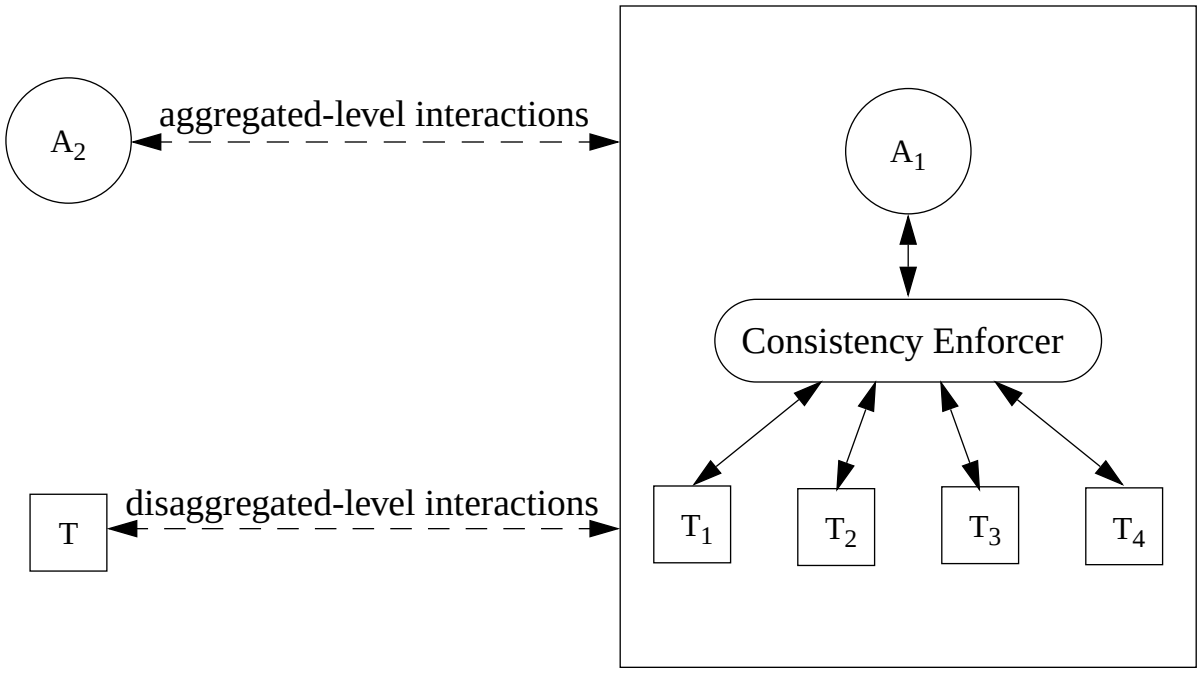

Figure 2 - Enforcing consistency between 2 levels

2.6 Perceiver problem

Different entities might choose to view a particular AE at different levels of aggregation. Most schemes handle this case by having all the participating entities disaggregate to the lowest common level. Instead, it should be possible for a perceiver to specify at what level of aggregation it wants to perceive entities. A perceived entity should be designed such that it presents a choice of views to the rest of the simulation. A perceiver can specify a view, and the perceived unit, which gets the request in unambiguous terms, can then send the requested information. Note that it is the responsibility of the *perceiver* to demand the kind of information and the level of detail from the perceived. Obviously, there would be a "base case" — the lowermost level of units — agreed upon by the entire simulation beforehand. This design is scalable, and more closely resembles real-life interactions.

3 UNIFY

We believe that the dichotomy between aggregated and disaggregated states is a false one. In the scheme we propose, each unit either maintains state information at all allowed levels of aggregation or must be able to furnish it on demand. Simulation of the unit entails handling incoming interactions about all levels. Each unit is responsible for enforcing logical consistency across aggregation levels. The effect of any incoming interaction has to be reflected in the attributes of all the levels of the unit.

For example, a platoon unit composed of four tanks would contain information regarding the platoon as well as the individual tanks (Fig. 1). Similarly, a battalion unit would have information at the battalion level regarding each of its platoons. In turn, each of the platoons would contain information regarding the individual tanks. Note that in this example we have arbitrarily chosen a tree structure to model military organization.

3.1 Functional description

The perceiver initiates interactions when another unit is within its perception envelope [Sher92]. At this point, the perceiver requests information from the perceived unit at the level of aggregation it desires. This allows units to be perceived differently by different perceivers.

Let A_1 and A_2 be platoons of tanks and T be a solitary tank. The interactions between A_1 and A_2 occur at an aggregated level. For instance, battalion state information such as velocity and strength may be exchanged and acted upon. When T comes into contact with A_1 it requests entity-level information. A_1 proceeds to send information regarding the tanks of interest to T (Fig. 2). Typically, this information would be culled from data A_1 maintains on each tank.

A_2 receives information sent from A_1's "global" fields — fields that are either common to all entities or can be deduced from the individual attributes of the units (Fig. 3). For example, if T is absent, the velocity of the individual tanks in A_1 would not be important, and a global velocity vector in A_1 could be sent to A_2. However, if T is present, then A_1's velocity vector could be computed as a weighted average of the individual velocities of its constituent tanks.

3.2 Maintaining consistency between levels

Consistency must be maintained between levels of aggregation. Interactions from T and A_2 with A_1 will have to be serialized and operated on A_1 atomically (Fig. 2). When A_1 receives a message regarding an interaction from any other unit, it must process that message fully before beginning to process any other message that might arrive. The atomic constraint is necessary because interactions at any level may affect the other levels. If an interaction is from T, then its effect on A_1 should be reflected in subsequent interactions between A_1 and A_2. In this instance, information flows from the tank level to the platoon level in A_1. Likewise, if an interaction comes from A_2, then the state of each tank in A_1 may have to be updated. Atomicity of interactions, while strict, ensures that all levels are consistent with each other. It is worthwhile to note that we have reduced the problem of maintaining consistency between aggregated and disaggregated entities to the task of serializing and atomically handling each request arriving to the unit.

There exists a two-way relationship between the attributes at different levels. If there exists a function f mapping a set of disaggregated attributes to an aggregated attribute, then there must exist at least one inverse function f^{-1} which maps the aggregated attribute to the disaggregated attributes. Lack of such an inverse function may lead to temporal inconsistency manifesting itself again. The "rules-of-thumb" suggested in [Allen92] are useful guidelines in the design of f and f^{-1}.

There is a range of options for implementing a unit — single-CPU to distributed-network implementations. The latter introduces high network latencies while trying to maintain consistency. The former may put an unacceptable burden on the CPU. Modern parallel computers, with their fast interconnection networks, should provide an efficient compromise.

recognized by the community. The next three are non-traditional issues, which we envisage will be faced by designers as the simulation world encompasses other applications, such as environment modeling, socio-economic modeling and control systems modeling.

2.1 Temporal inconsistency

Temporal inconsistency [Davis93] occurs when an entity performs actions in an interval of time in a simulation, which it could not have done in a real-life situation. This may happen, for example, during a disaggregation-aggregation-disaggregation sequence. The information stored at an aggregated level is not sufficient to provide temporal consistency at the disaggregated level. In other words, in the first transition, i.e., disaggregated to aggregated, some information pertaining to the DEs may be lost, thus the second transition may result in a disaggregated state that is inconsistent with the first disaggregated state.

A set of DEs may reaggregate after the current set of interactions that caused it to remain disaggregated are completed. While reaggregating, certain information, for example, the actual positions of the DEs, might be lost. After reaggregation, it may so happen that within a short time, a disaggregation has to be effected. On disaggregating again, a standard algorithm or doctrine [France93] [Clark94] would be applied to position the entities. This might cause unrealistic discontinuities in entity states, such as “jumps” in position.

2.2 Chain disaggregation

In a two-level simulation, interactions between AEs should naturally occur at an aggregated level and those between DEs at a disaggregated level. However, many options arise when AEs interact with DEs. A naïve approach is to disaggregate an AE whenever it comes into sensor proximity of a DE. This, however, could cause chain disaggregation, wherein many AEs are forced to disaggregate in a short period of time [Smith94]. Consider a simple case where four AEs are interacting in the following linear fashion ($A \leftrightarrow B$ indicates that entities A and B interact with each other), i.e. $AE_1 \leftrightarrow AE_2 \leftrightarrow AE_3 \leftrightarrow AE_4$. AE_1 comes into contact with a DE, resulting in its disaggregation. This forces AE_2 to disaggregate, followed by AE_3 and AE_4. The naïve approach causes unnecessary disaggregation, and puts a burden on computing and network resources.

Another option is to disaggregate up to some distance in the chain, but this is a rather inelegant solution since the question of how to handle AE-DE interactions is still left open. Yet another option is to partially disaggregate the AE interacting with the DE, but this can cause consistency problems.

2.3 Network flooding

Network resources may be strained by the acts of aggregation and disaggregation, depending on the scheme used. For example, a proposal in [Robkin92] required on the order of 10 seconds to complete the aggregation process. This was because of the complexity of the algorithm in which each DE could request to be reaggregated, and each could also refuse to be reaggregated, thus stopping the entire process. This decentralized control means that the network can be flooded if every DE decides to reaggregate at the same time.

Regardless of the disaggregation scheme used, there exist cases where it is better *not* to disaggregate. We do not know of any scheme that handles the following case [Smith94] well. Consider an airborne reconnaissance over a section of the battlefield. The aircraft is interested only in the positions of the individual entities and does not affect them in any way. Most schemes to date would require a disaggregation sequence as the aircraft flies over an AE’s location, because the positions of individual entities are not kept at the aggregated level. Furthermore, once the aircraft is out of range, the DEs would reaggregate. This scenario is more catastrophic if the aircraft returns shortly after every reaggregation!

2.4 Aggregation of dissimilar entities

There exist scenarios where it would make sense to aggregate entities that might not normally be aggregated. In climate-modeling, dust and water-vapor may be modeled independently but when dealing with large bodies of air, one may choose to aggregate these. The view that AEs are collections of like DEs has come predominantly out of battlefield simulations. As the field diversifies, this view may prove restrictive. Aggregation of dissimilar entities makes the configuration problem [Horr92] more acute.

2.5 Dynamic aggregation

Dynamic aggregation — where the entities that may be aggregated are decided on-the-fly — might be a requirement for certain kinds of applications. Dynamic aggregation implies that the entities which could be aggregated are not known beforehand. Instead, this decision may be postponed until the simulation runs. “What-if” scenarios are an application of this idea. In the context of battlefield simulations themselves, commanders may wish to make unorthodox force groupings. We have not seen any current schemes that support dynamic aggregation.

To disaggregate or not to disaggregate, that is *not* the question

Anand Natrajan
Anh Nguyen-Tuong

{anand,nguyen}@virginia.edu
Dept. of Computer Science,
University of Virginia

Abstract

The dichotomy between aggregated and disaggregated states is a false one. It is possible for an aggregated entity to be at many levels of aggregation by storing the relevant data of all levels. In this paper, we propose a scheme, `UNIFY`, wherein each unit either maintains state information at all allowed levels of aggregation or furnishes it on demand. We present problems with traditional approaches towards aggregation such as temporal inconsistency, chain disaggregation and network flooding. We also deal with issues that we envisage will beset the simulation world such as aggregation of dissimilar entities, dynamic aggregation and the perceiver problem. We describe a framework with which these problems could be solved or tackled better. We study the benefits and disadvantages of `UNIFY` and propose new directions for research. Finally, we analyze the demands made by our scheme on network, memory and CPU resources.

1 Introduction

Distributed simulations can be broadly classified as either aggregate-level simulations (constructive) or entity-level simulations (virtual). Simulated objects in constructive simulations are said to be *aggregated* since they contain information pertaining to a collection or group of entities. On the other hand, simulated objects in a virtual simulation are basic entities in the sense that they tend not to be broken down further.

There are two schools of thought regarding constructive simulations. The first is that aggregate-level simulations are valid and useful. The second is that the validity of constructive simulations cannot be proved. Our paper is based on the belief that constructive simulations are useful [Davis92], and when properly designed, valid. We believe that consistency models for aggregation and disaggregation, such as the one in [Davis93] exist, and when they are found, our scheme will be a strong alternative to the way aggregate simulations are currently done.

In recent years, there has been a push to link constructive and virtual simulations, especially battlefield simulations. Examples of successful linkages include the `BBS` to `SIMNET`, `EAGLE` to `SIMNET`, and `AWSIM` to `ModSaf` efforts. The common approach for handling interactions between these two worlds has been to designate areas of the battlefield as “virtual playboxes” in which all interactions are performed at the entity level [Karr94]. When an aggregated entity (AE) enters the playbox, it goes through a disaggregation process whereby the AE is separated into its constituent units. These units are said to be disaggregated entities (DEs). Upon leaving the playbox, these units may reaggregate.

The virtual playbox approach has several shortcomings: (1) the playbox(es) must be chosen *a priori*, (2) their boundaries are static in many cases, which means that virtual entities may encounter the boundary of the box sooner than is desirable, and (3) by definition, no aggregate-level simulations may occur inside. However, this approach is simple, since aggregation and disaggregation decisions are reduced to determining when the boundary of the playbox is crossed.

A more generic scheme, where aggregation decisions are made dynamically, is clearly preferable. The playbox approach hides some issues, which will gain importance once alternative approaches are examined: (1) temporal inconsistency, (2) chain disaggregation and (3) network flooding. We propose a scheme, `UNIFY`, which solves 1 and 2, and alleviates 3.

The remainder of the paper is organized as follows: Section 2 focuses on several issues in designing multi-level simulations. Section 3 describes our framework, `UNIFY`, for designing multi-level simulations. Section 4 discusses other related work. Sections 5 and 6 demonstrate how `UNIFY` tackles the issues discussed earlier, and point out the advantages and disadvantages of our scheme. Section 7 addresses the cost of `UNIFY`. We present our conclusions in Section 8.

2 Issues in multi-level simulations

We highlight some of the issues faced by multi-level simulations. The first three are traditional problems in the sense that they have been well-

To disaggregate or not to disaggregate, that is *not* the question

Anand Natrajan and Anh Nguyen-Tuong

Technical Report No. CS-95-18
March 23, 1995

Contact : anand@virginia.edu

Web : ftp://ftp.cs.virginia.edu/pub/techreports/CS-95-18.ps.Z

Published : ELECSIM 95