

DYNAMIC INFORMATION ARCHITECTURE SYSTEM (DIAS): MULTIPLE MODEL SIMULATION MANAGEMENT

**Kathy Lee Simunich, Software Engineer; Pamela Sydelko, Environmental Scientist;
Jayne Dolph, Environmental Scientist; and John Christiansen, Software Engineer;
Argonne National Laboratory, Argonne, Illinois
9700 S. Cass Ave., Argonne, IL 60439, (630) 252-3285, simunich@dis.anl.gov**

Abstract: The Dynamic Information Architecture System (DIAS) is a flexible, extensible, object-based framework for developing and maintaining complex multidisciplinary simulations of a wide variety of application contexts. The modeling domain of a specific DIAS-based simulation is determined by (1) software Entity (domain-specific) objects that represent the real-world entities that comprise the problem space (atmosphere, watershed, human), and (2) simulation models and other data processing applications that express the dynamic behaviors of the domain entities.

In DIAS, models communicate only with Entity objects, never with each other. Each Entity object has a number of Parameter and Aspect (of behavior) objects associated with it. The Parameter objects contain the state properties of the Entity object. The Aspect objects represent the behaviors of the Entity object and how it interacts with other objects. DIAS extends the “Object” paradigm by abstraction of the object’s dynamic behaviors, separating the “WHAT” from the “HOW.” DIAS object class definitions contain an abstract description of the various aspects of the object’s behavior (the WHAT), but no implementation details (the HOW). Separate DIAS models/applications carry the implementation of object behaviors (the HOW). Any model deemed appropriate, including existing legacy-type models written in other languages, can drive entity object behavior. The DIAS design promotes plug-and-play of alternative models, with minimal recoding of existing applications.

The DIAS Context Builder object builds a constructs or scenario for the simulation, based on developer specification and user inputs. Because DIAS is a discrete event simulation system, there is a Simulation Manager object with which all events are processed. Any class that registers to receive events must implement an event handler (method) to process the event during execution. Event handlers can schedule other events; create or remove Entities from the simulation; execute an Entity’s behavior; and, of course, change the state of an Entity.

In summary, the flexibility of the DIAS software infrastructure offers the ability to address a complex problem by allowing many disparate multidisciplinary simulation models and other applications to work together within a common framework. This inherent flexibility allows application developers to more easily incorporate new data, concepts, and technologies into the simulation framework, bringing the best available knowledge, science, and technology to bear on decision-making processes.

INTRODUCTION

A Brief Overview of the DIAS Framework: The Dynamic Information Architecture System (DIAS) is a flexible, extensible, object-based framework for developing integrated, multidisciplinary, dynamic simulations. DIAS is domain-neutral, meaning that it is not designed for simulations specific to any one discipline or subject area; rather it supports the development of simulations for virtually any type of application.

DIAS has been used successfully to build a wide range of simulations, including dynamic terrain- and weather-influenced military unit mobility assessment; integrated land management at military bases (Sydelko et al., 2000; Sydelko et al., 2001); a dynamic virtual oceanic environment; clinical, physiological, and logistical aspects of health-care delivery; avian social behavior and population dynamics (Rewerts et al., 2000); and studies of agricultural sustainability under environmental stress in ancient Mesopotamia (Christiansen, 2000).

An important design distinction of DIAS is that it is a framework – an environment and set of tools that developers utilize to build simulations. DIAS is not a model or a suite of models, but rather a flexible modeling environment within which developers build applications by either “wrapping” existing models and applications (including legacy models, database management [DBMSs], geographic information systems, etc.), coding new in-line models, or building any combination of external and in-line models to create a simulation. In this way, the DIAS framework allows new and/or existing legacy models and other applications to interoperate in the same object environment.

Models and applications interfacing with DIAS can be written in virtually any computer language. Legacy applications are run in their native language (e.g., FORTRAN, C, MODSIM, etc.), with only minimal (if any) restructuring of code required to interface with the DIAS framework. Internal DIAS models are coded in Java (as methods of subclasses of the DIAS core object classes) and can be run either within a DIAS simulation or as stand-alone models that can be incorporated into other integrated modeling suites.

DIAS supports fully distributed operation via the use of Common Object Request Broker Architecture (CORBA), Java’s Remote Method Invocation (RMI), and most recently, the Simple Object Access Protocol (SOAP) to provide legacy models as web services. DIAS can use an object DBMS to provide persistence to DIAS objects, or a relational DBMS can be supported via commercial bridging software and the Java DataBase Connectivity (JDBC) interface.

DIAS Entity-centric Design: At the very core of DIAS-based simulations are the Entity objects that represent the real-world counterparts for the application problem space. Developers define the simulation “playing pieces,” or Entities, including attributes and behavior. This Entity-centric design trait is in contrast to other modeling paradigms centered on process interaction and linking.

Centering DIAS simulations on Entity objects and their associated interactions promotes greater flexibility and extensibility for DIAS-based applications. In DIAS, models communicate only through Entity objects, never directly with each other. DIAS infrastructure objects formally define and isolate models and other applications that implement Entity object behaviors.

This approach promotes modularity and makes it easier to add models or swap in alternative models to express behavior without re-coding other connected applications. Thus, DIAS scales well to increasingly complex problems. Figure 1 shows an illustration of a traditional model-to-model interaction approach and how models and applications running in a DIAS-based simulation interact indirectly with one another via Entity objects.

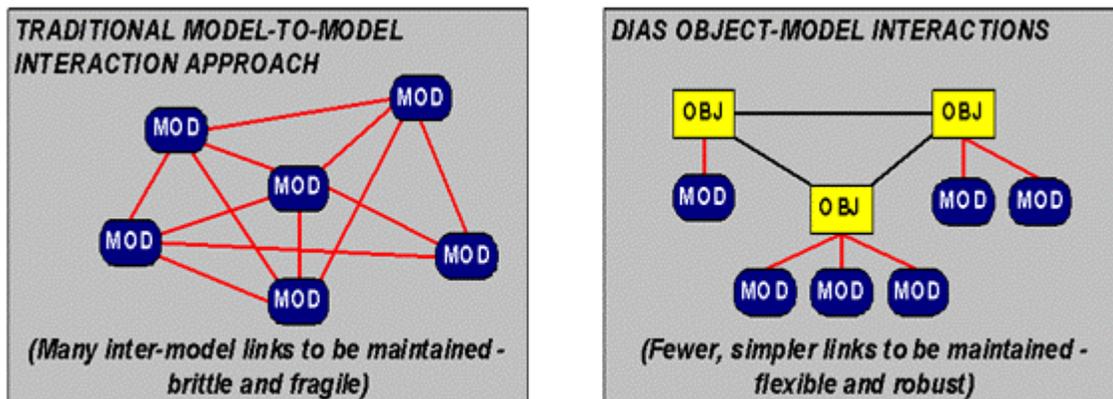


Figure 1: Traditional model-to-model interaction approach versus DIAS object-model interaction approach

Entity states are updated throughout the simulation via events triggered by simulation dynamics. However, Entities themselves have no knowledge of the mechanisms (models/processes) responsible for modifying their states. Entities simply respond, in an appropriate manner, as defined by the developer, to events triggered by simulation dynamics. This type of response allows several diverse mechanisms to affect Entity states, as is the case in real-world dynamics. Thus, while application development in DIAS is very specific, component development (e.g., Entity design) remains somewhat abstract by capitalizing on DIAS’s highly flexible and extensible modeling environment.

The Entity-centric design of DIAS also allows DIAS-based applications to more closely mimic the associations and relationships that exist in the real world, including dynamic

feedback. In general, most models and applications have been designed to operate independently, even though effective decisions often call for assessing several components of a system simultaneously – in terms of their relationship to each other as well as how they affect broader management decisions. This case is true, for instance, in the domain of environmental management. DIAS provides a framework for developing applications that address interprocess dynamics in a highly realistic way, in large part because the focus of the simulation is on the Entity objects and their interaction with one another. The DIAS design allows developers to articulate the dynamics of an ecosystem much more closely to the way we understand them, while at the same time not impose one world view on the development of an application. DIAS interprocess flows are realistic because model processes affect the state of Entity objects, and thus reflect the true dynamics of the real-world system.

Furthermore, because Entity objects and their associated behaviors are developed within a framework that already houses a diverse array of both environmental and nonenvironmental objects, the connection to other models and processes is made easier. Developers do not have to build objects from scratch but can often use existing objects from the DIAS Entity object library and add attributes and behaviors as new applications dictate.

KEY DIAS ARCHITECTURAL COMPONENTS

The key DIAS architecture components (Figure 2) provide the means for transforming developer inputs into simulation outputs. The architecture components illustrated are Entity object, Process object, Model/ModelState object, the Entity-Aspect-Process-Model object linkage, Context Builder object, Context object, and Simulation Manager object.

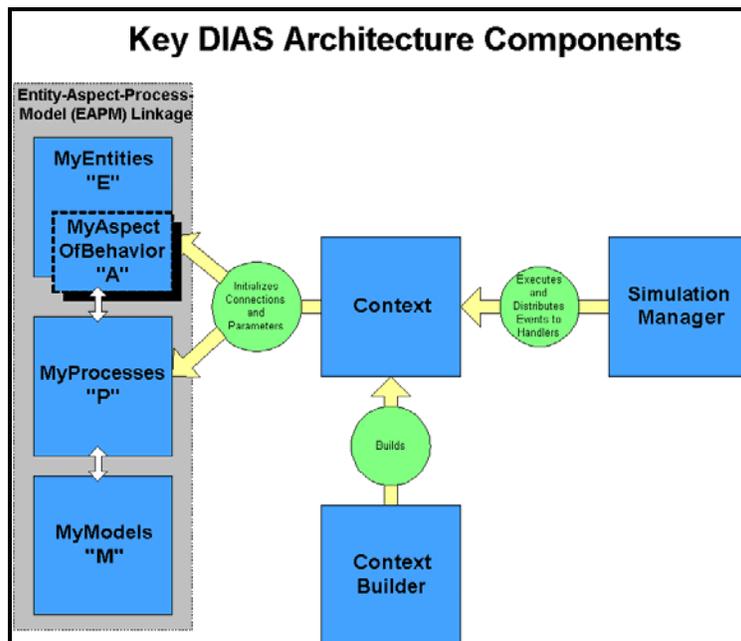


Figure 2: Key DIAS architectural components

Entity Object: The DIAS Entity objects reflect domain areas and applications that have been built for various DIAS simulations. DIAS extends the object paradigm of inheritance and encapsulation by abstraction of the Entity objects' dynamic behaviors. DIAS Entity class definitions contain an abstract description of the various aspects of the object's behavior via their Aspect (of Behavior) objects (the "WHAT"), but no implementation details (the "HOW"). For example, a stream object can have several Aspects of Behavior, such as "channel flow" and "solute transport." The actual implementation of these behaviors would come from one or more process models linked to the simulation via the Entity-Aspect-Process-Model (EAPM) linkage. Channel flow, for instance, might be implemented by an appropriate external (legacy) hydrologic model that calculates stream flow, or alternatively by an in-line DIAS stream flow model. Similarly, either an in-line or an external contaminant transport model algorithm could implement solute transport. In both cases, alternate models can be readily plugged in to the simulation to implement these behaviors of the stream object, with no modification to the Entity object directly because the Entity itself does not contain the implementation details, but rather abstracts these behaviors to appropriate models.

Process Object: DIAS Process objects represent and formally define specific models that can implement specific abstract Entity object behaviors (Entity Aspects). There is one DIAS Process object for each Aspect in a simulation. The Process object is responsible for all data translation, unit conversions, and data aggregation/disaggregation issues. The Process object also controls the packaging of Entity data needed as input to models, as well as the unpackaging of model output data and its distribution to the Entities.

Model/ModelState Objects: DIAS Model objects are responsible for performing the calculations for a Process. A Model object contains one method per Process object it serves. DIAS Model objects either call out to an external model via CORBA, RMI, or SOAP to perform the calculations, or directly execute the behavior through internal method source code. Thus, legacy code written in another language can be reused as the model implementation of an Entity's Aspect. Models can be written generically without referencing DIAS framework classes so as to be directly usable within other systems.

The ModelState object serves as a common data block, containing all data that can be used for a Model by an Entity of a particular type. ModelState objects are specific to individual instances of Entity, whereas Model objects are general to all instances of Entity. In other words, each Entity has its own set of data to be used for a model (the ModelState), but all Entities that call the same model point to that one instance of the Model object, and hence execute the model code, passing in the ModelState data as input/output parameters.

Entity-Aspect-Process-Model Linkage: In external (e.g., legacy) model integration, a Process object is associated with a specific process (e.g., subroutine) in the external

model source code. For DIAS inline models, the Process object is connected to a specific Model object method that contains the calculation and thus provides the Entity behavior.

The Process object is the only object with knowledge of both the Entity “world view” and the Model “world view.” This capability promotes a component-based approach to model development, where models are not simulation or framework specific and therefore can be readily swapped in/out of any simulation suite without time-consuming reworking of existing application code.

Linkage of Entity abstract behaviors to appropriate modeling functionality (via the EAPM object connection) is done at run time to meet the specific needs of a given simulation context. This EAPM linkage (Figure 3) is a keystone of the DIAS design philosophy. The EAPM abstraction promotes greater flexibility and extensibility than would otherwise be possible in a “hard-wired” system, because Entity behavior can be driven by any model(s) deemed appropriate, rather than a fixed set. The EAPM design trait promotes plug-and-play of alternative models, with virtually no recoding of existing applications.

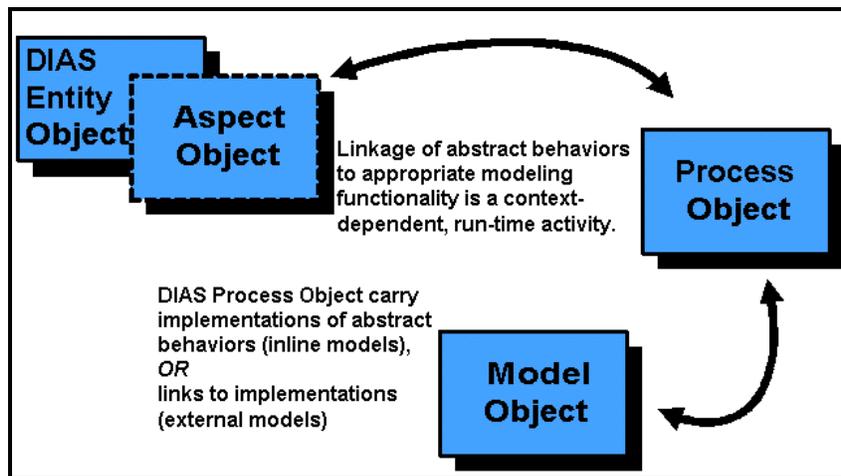


Figure 3: The DIAS Entity-Aspect-Process-Model (EAPM) linkage

Writing an application in DIAS requires the developer to code those methods of the Entity and Process objects that must be overridden by the domain subclasses. These methods are called from within the DIAS execution framework in a specified sequence during the simulation. Since developers can override specific methods, they only need to write code that specifically tailors the discrete event simulation to the domain.

DIAS Context/Context Builder Objects: Figure 4 illustrates the context building process in DIAS. The DIAS Context object houses all the contextual information about the simulation, primarily provided by the user through the graphical user interface (GUI). The Context Builder object gathers the information to create a Context and maintains information about all the models to be used. From that information, the Context Builder determines which types of Entities need to be created and which parameters of those entities must be instantiated for the simulation. The Context Builder is also used to set

any “global” or overall simulation parameters, such as simulation start and end times, and the initial set of Entity instances to be modeled.

Building the Context in DIAS: The first phase of a DIAS simulation consists of building the context from information, data, and object definitions provided by the developer (Figure 4). The simulation user can choose the models to use for a particular simulation from a list of models available for the domain entities. The user also selects the data sources that will be used to instantiate the specific Entities for the simulation run as well as other simulation parameters, such as regional extent, and start/end times for the run. From this information, the context builder makes all the necessary DIAS connections and instantiations for the context of the simulation. Thus, the context is built at run-time, which provides the “plug-and-play” ability of the DIAS system. At this point, execution of the context is taken over by the Simulation Manager.

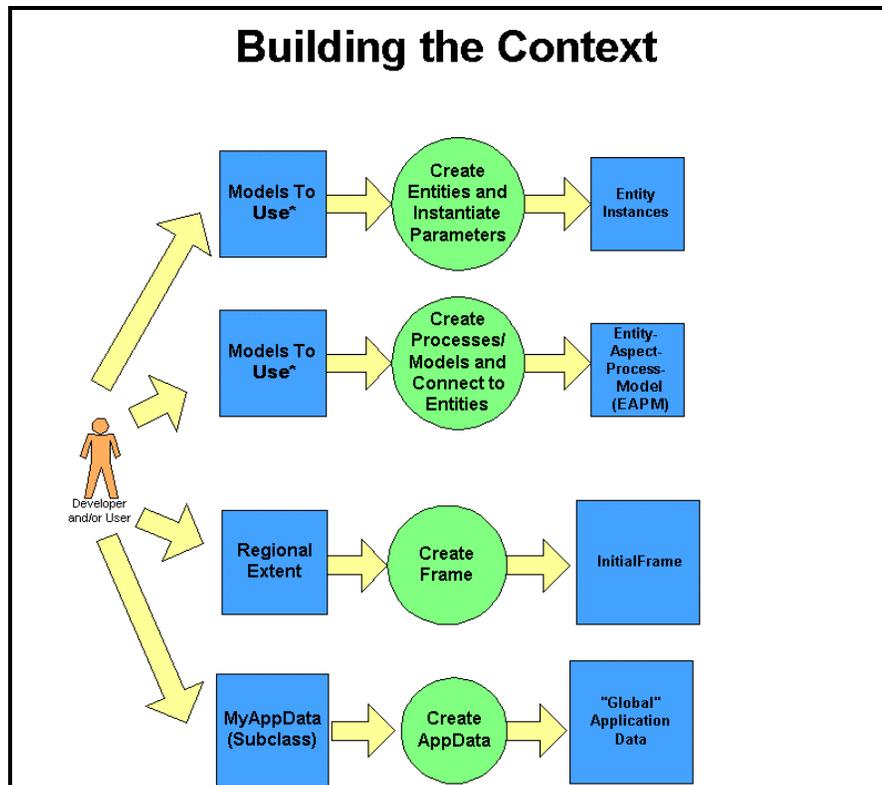


Figure 4: Building the Context in DIAS

DIAS SIMULATION EXECUTION MANAGEMENT

The execution of a DIAS simulation consists of three phases: (1) simulation initialization, (2) simulation execution, and (3) simulation completion and cleanup (Figure 5). During the simulation initialization step, all the instantiated Entities register with the Simulation Manager to receive notification of the specific events in which they are interested.

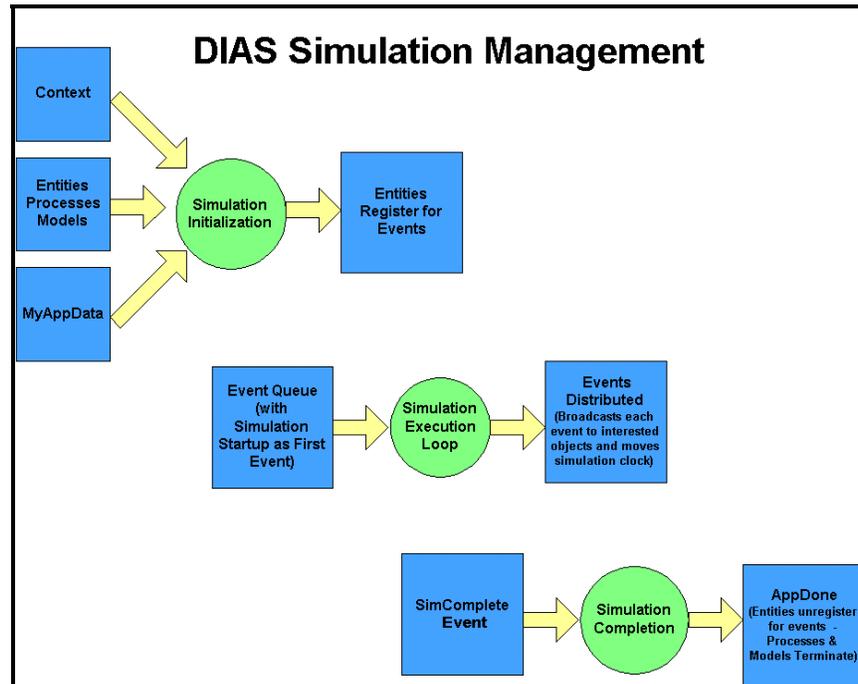


Figure 5: DIAS execution management

Because DIAS is a discrete event simulation system, DIAS provides a Simulation Manager object through which all events are processed. All Entity subclasses, Process subclasses (optionally), and the ApplicationData subclass register and unregister to receive broadcast notification of events from the Simulation Manager.

The developer defines all events that would make up the dynamics of a simulation run. The simulation execution phase, during which event dispatch takes place, is the core of the domain-specific application. Through the execution of the event handlers within the Entities, the DIAS application models the processes of the specific context (scenario). The Simulation Manager broadcasts a simulationStarted event, and a simulationCompleted event that the Entities can register for in order to initialize the simulation or finalize Entity state. All other events are scheduled as part of the specific application and are broadcast according to the simulation time clock.

Within DIAS, the Simulation Manager executes in its own Java thread. This is done so that the GUI portion of an application need not block during simulation execution, and the user is allowed to pause, resume, or stop the simulation at any time.

All events defined for a simulation can optionally carry data along with (1) a simulation time stamp as to when the event has occurred and (2) the identity of the sender of the event. Any class that registers to receive events must implement an event handler (method) in order to process the event during execution. The mechanism of registering and unregistering for events and event handling are synonymous with the event/listener model employed by the Java Swing Application Programming Interface (API). Whereas the Swing classes predefine Java interfaces with specific event methods to implement, DIAS allows developers to name their own event handlers that receive a simulation event from which they can extract the domain-specific data. In this manner, DIAS applications can be written with an infinite set of events without performing undue subclassing.

Event handlers can schedule other events; create or remove Entities from the Frame; perform an Aspect (schedule an executeProcess event) of an Entity; and, of course, change the state of an Entity. The DIAS philosophy is that the EAPM linkage implements the algorithms that model the behavior of an Entity, and the scheduling and handling of events implement the dynamics of the simulation.

DIAS Model Process Flow: An Illustrative Example: Figure 6 illustrates the connections between the core DIAS infrastructure objects discussed previously and shows the flow of execution for an example external model during a DIAS simulation. The execution for an internal model follows the same sequence; however, DIAS then does not use CORBA to call out to an external model method via the Model Controller. In the example represented by Figure 6, the legacy model is MM5, a large parallel Fortran model for mesoscale weather forecasting. This legacy model provides the evolving state of the Atmosphere Entity within a DIAS Simulation (the actual Atmosphere Entity is not shown in Figure 6.) The developer must write a Model Controller that follows the DIAS initialize, execute, finalize design pattern and provides the interface between the DIAS simulation and the legacy model. The Model Controller can be written in any language that has a CORBA interface or, alternatively, can be a SOAP Web Service. Specifically, Interface Definition Language (IDL) code is needed if CORBA is selected, and a Web Services Description Language (WSDL) file is needed if SOAP is used.

In Figure 6, the legacy model and the Model Controller are external to the DIAS framework. Internal to the DIAS framework, the MM5 EvolveAtmosphere Process is written to provide the Atmosphere Entity with the implementation of its “Evolve Atmosphere” Aspect and is connected at run-time. When the EvolveAtmosphere Process is called to execute within the simulation, it begins by building the process input data from the DIAS Entities and packages it with the Entity’s ModelState (MM5ModelState). This ModelState is passed into the MM5Model. In this example, CORBA communicates with the external model; therefore, the MM5Model extends our DIAS CORBAExternalModel class and takes care of the marshalling and unmarshalling of the data outside DIAS to the MM5 Model Controller. (If this were an internal DIAS model, it would execute code directly via the MM5Model subclass.) At this point, DIAS blocks within the MM5Model until the Model Controller executes the legacy model and returns the output within the ModelState back to the MM5Model. The MM5Model then passes it

back to the EvolveAtmosphere Process where it (1) unpacks the output data from the ModelState and (2) updates the Entity parameters and/or schedules more events. The EvolveAtmosphere Process then waits to be called again for either the same Entity instance or another of that type.

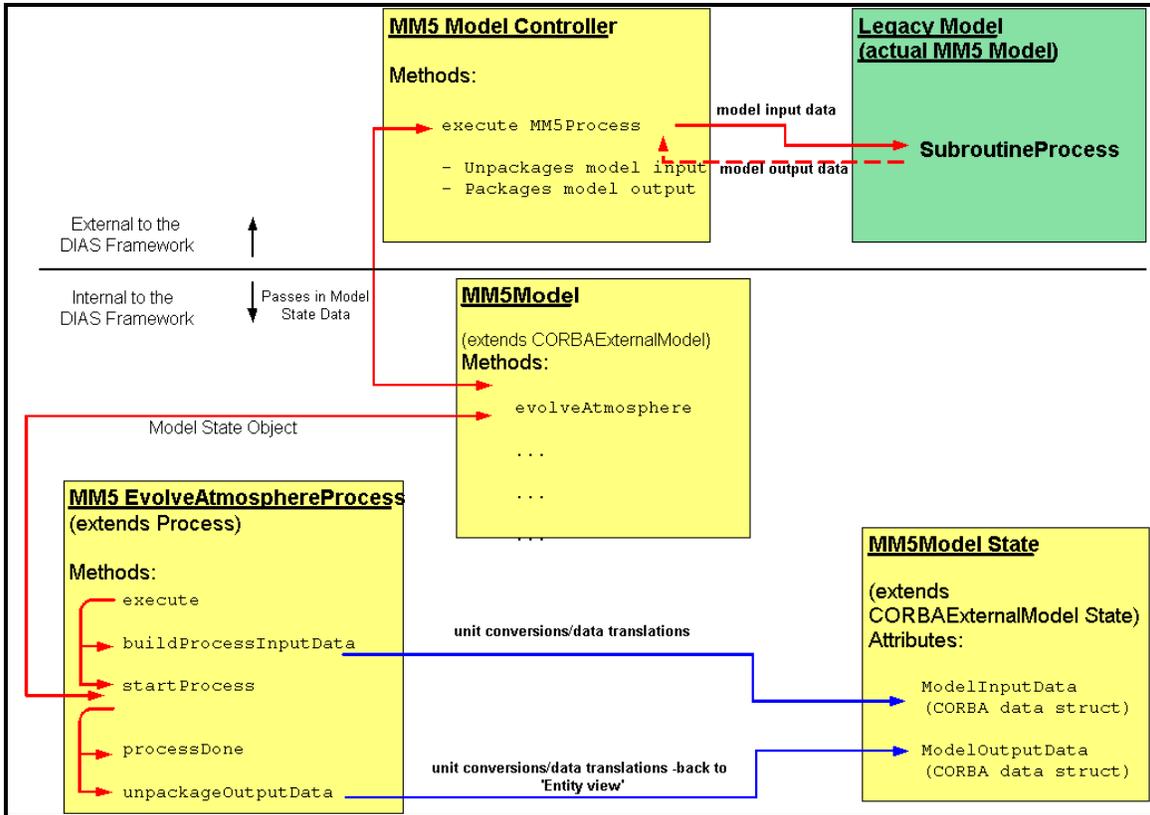


Figure 6: Process flow for execution of an example external model, MM5

SUMMARY AND FUTURE DIRECTIONS

DIAS is a process-based, discrete event simulation framework that can be used to develop and maintain a wide variety of multidisciplinary simulations. The framework is completely object-oriented and is domain neutral, making it useful for many application domains. A key benefit of the DIAS framework for development of multidisciplinary models is the ability to allow new and/or existing legacy models as well as other applications to operate in the *same* object environment. The modular design of DIAS promotes this flexibility and extensibility, which enhances cost-effectiveness and the evolution of applications.

DIAS continues to be used to develop a wide variety of applications. Currently, efforts are underway to utilize DIAS for a three-compartment multimedia prototype model sponsored by the U.S. Environmental Protection Agency (EPA). In addition, there are ongoing efforts sponsored by the U.S. Navy in building the Dynamic Integrated Ocean

Prediction System application, a DIAS-based virtual maritime environment within which existing models are employed to simulate the transition of wind-generated waves in the deep water to waves in the near-shore environment, then to surf characteristics and currents. Another new application built using DIAS is CASCADE, the Complex Adaptive System Countermeasure Analysis Dynamic Environment sponsored by the Joint Chiefs of Staff. CASCADE is an object-oriented software system for building and running agent-based multidisciplinary simulations that concurrently address socioeconomic, psychological, environmental, etc., factors to support countermeasures analysis. The first full implementation of CASCADE is CASCADE-CD, a counter-drug simulation software system prototype to aid drug interdiction analysts in deriving and justifying force structure and operational planning recommendations for combating the South American cocaine trade.

The DIAS framework is capable of integrating the wide variety of models, simulations, and data necessary to address complex problems. Entity objects can be created that represent the multiple interrelated aspects of ecosystems, both natural and societal. The Entity objects can exhibit specific behaviors, either directly coded within DIAS or provided by linked external models and applications. DIAS can address both the spatial and temporal scale issues required for complex dynamic modeling.

While the general-purpose design of DIAS makes it a robust framework for multidisciplinary modeling, this benefit necessitates a more complex development environment than typically seen in more domain-specific simulation development software. Currently, DIAS users must have fairly advanced object-oriented software engineering skills to readily implement applications in the framework. The DIAS API provides instruction and examples that help guide application developers through creation of simulations; however, at this time, there is presently no GUI to assist this process, and developers must extend the framework objects directly by adding new source code. As a first step in assisting programmers, an API has been developed to help application developers utilize the framework.

Currently, DIAS users are typically Java programmers. The DIAS paradigm, however, is beginning to be adopted by other research groups who are building and enhancing the application development interfaces for specific areas. The Army Corps of Engineers Engineering Research and Development Center has acquired DIAS to build modeling and simulation applications related to military land management. The EPA is developing the Multimedia Integrated Modeling System (MIMS) framework, a software infrastructure or environment that will support constructing, composing, executing, and evaluating complex modeling studies. MIMS is being developed to support complex cross-media modeling. The MIMS framework uses DIAS for its model coupling paradigm and execution management software. The DIAS software library provides basic templates for domain objects and models and capabilities for constructing interacting sets of models and executing those models in the proper order. The MIMS project is supporting the addition of new capabilities to the DIAS framework. In addition, the MIMS framework layers on top of DIAS (1) generic user interfaces; (2) well-defined parameter types; (3) functionality that minimizes the programming effort required to incorporate new models

into a MIMS simulation; (4) general controllers to perform repetitive work, such as sensitivity and uncertainty studies; and (5) additional tools to support modelers.

Through the participation on an interagency work group, Software System Design and Implementation for Environmental Modeling, the DIAS team has been identifying needs and exploring possible collaborative efforts that can help make DIAS more compatible with other modeling systems being developed. In particular, the DIAS team has taken the lead on the Execution Management subgroup and is exploring ways to make this DIAS component available for use by other research and development teams.

REFERENCES

- Christiansen, J.H., 2000, A Flexible Object-Based Software Framework for Modeling Complex Systems with Interacting Natural and Societal Processes. Proc. of the 4th Int. Conf. on Integrating GIS and Environmental Modeling (GIS/EM4): Problems, Prospects and Research Needs, Banff, Alberta, Canada.
- Rewerts, C.C., Sydelko, P.J., Dolph, J.E., Shapiro, A.M., and Taxon, T.N., 2000, An Object-Oriented, Individual-Based, Spatially Explicit Environmental Model: A Discussion of the Approach to Implementing the System. Proc. of the 4th Int. Conf. on Integrating GIS and Environmental Modeling (GIS/EM4): Problems, Prospects and Research Needs, Banff, Alberta, Canada.
- Sydelko, P.J., Hlohowskyj, I., Majerus, K., Christiansen, J.H., and Dolph, J., 2001, An Object-Oriented Framework For Dynamic Ecosystem Modeling: Applications For Integrated Risk Assessment. *The Science of the Total Environment* 274, 271-281.
- Sydelko, P.J., Majerus, K.A., Dolph, J.E., and Taxon, T.N., 2000, An Advanced Object-Based Software Framework for Complex Ecosystem Modeling and Simulation. Proc. of the 4th Int. Conf. on Integrating GIS and Environmental Modeling (GIS/EM4): Problems, Prospects and Research Needs, Banff, Alberta, Canada.