A Multi-Formalism Modeling Composability Framework: Agent and Discrete-Event Models

Hessam Sarjoughian
Dongping Huang
Arizona Center for Integrative Modeling & Simulation
Department of Computer Science & Engineering
Fulton School of Engineering
Arizona State University, Tempe, Arizona, 85281-8809
Email: {sarjoughian|dongping}@asu.edu

Abstract

It is common practice to build complex systems from disparate sub-systems. Model composability is concerned with techniques for developing a whole model of a system from the models of its sub-systems. In this paper we present a new kind of Multi-Formalism Modeling Composability Framework which introduces the concept of Knowledge Interchange Broker for composing disparate modeling formalisms. The approach offers separation of concerns between model specifications and execution protocols across multiple modeling formalisms. The framework is exemplified via vehicle and agent models described in the Discrete-Event System Specification and Reactive Action Planning formalisms. A high-level software specification that illustrates an implementation of this framework is described. Ongoing and future research directions are also briefly presented.

1. Introduction

Systems built from sub-systems have interesting behavior and characteristics stemming from diverse capabilities of individual sub-systems as well as interactions among them. Some well-known examples of these types of systems are transportation [6] and supply-chain networks [15]. One of two recurring relationships are typically observed among sub-systems of such systems: either one sub-system is controlled or managed by one or more sub-systems or, alternatively, one sub-system controls several other sub-system processes. A simple example illustrating a *process* and its *control* is a vehicle traveling from one location to another under the guidance of an agent. A model of such a system can consist of a vehicle model and an agent model. The vehicle movement represents a process and the agent decisions represent the vehicle's control. We can consider the vehicle model as representing a *physical entity* and the agent model as representing a *logical entity*. Each entity has its own unique characteristics. A vehicle model specification describes how inputs are processed and outputs are produced to simulate its movement. An agent specification describes the rules dictating feasible roadway paths a vehicle can travel given some constraints.

The physical and logical sub-systems have a symbiotic relationship where the former sends its state to the latter in order to receive appropriate commands to satisfy the needs of the overall system—for example, the agent model can direct the vehicle to follow a path (e.g., a set of x and y coordinates). In such a scenario, the vehicle model can have state variables for speed, fuel consumption, and distance traveled with respect to a reference location. Given the dynamics of a traveling vehicle under control of an agent, different approaches may be used to model their combined behavior.

To describe complex systems, we can use a *monolithic modeling specification*. This choice, however, can result in a model that can be difficult to specify and develop because of the confounding disparities in formulating vehicle and agent dynamics. For example, vehicle dynamics and decision policies can be described using either discrete-event [27] or mathematical logic [10]. Relatively complex models for vehicle movement with simple routing schemes can be developed using a discrete-event modeling and simulation framework. Similarly, mathematical logic can be used to model complex plans and simple vehicle dynamics. Neither, however, lends itself for describing both procedural and declarative behavior of vehicles and agents in their respective modeling formalisms.

A common approach to composability is to integrate "models" as software modules or components. For exam-

Accepted for Publication

ple, vehicle and agent models can be combined by writing customized software to handle data and control interactions. A model of a vehicle navigated via an agent model is developed by adding "software hooks and wires" to accommodate sending and receiving information between these models [12]. Alternatively, middleware technologies may be used to combine different execution engines, especially when there is a requirement for distributed or parallel execution. These approaches offer higher-level programming concepts and interfaces (i.e., a set of generalized services) to integrate models as software components. Indeed, numerous studies in many domains illustrate that while data and control can be successfully exchanged between models described in distinct formalisms, their integration often degenerates to software engineering or computer programming.

The above approaches, however, suffer from a major shortcoming—the resultant models rely on arbitrary modeling syntax and semantics, which not only make composition of disparate models difficult, but also adversely affect the degree to which composed models can be formalized. Thus, instead of *integrating models*, we propose *composing modeling formalisms*. This concept has been used for closely related modeling formalisms (e.g., [19]), but not for inherently different formalisms such as discrete-event and those for agent modeling (e.g., RAP). To help with model composability, we have developed a novel concept called *Knowledge Interchange Broker* (KIB). It enables *composition of modeling formalisms*. This kind of composability offers a common, generic basis for describing models that conform to their respective formalisms, yet the resultant composed models have well-defined structure and behavior.

2. Background

Research in model composability has been underway from many disciplines including systems engineering, software engineering, and artificial intelligence. The basic (informal and formal) notion of composability and its challenges are well recognized from science and engineering points of view. In particular, composability concepts, theories, and techniques can be said to consist of *abstraction, modularity, hierarchy (aggregation, disaggregation), and encapsulation*. These provide a set of primitives for component *(de-)composition* and *reuse* and therefore model composability. Numerous studies have described characteristics of composability, some detailing technical and non-technical challenges in achieving composability (see [2, 7, 17, 21, 25]).

Systems theory offers a sound framework for composition of models. It has traditionally focused on creation of theories for design of systems and, in recent years, their implementation aspects. The system-theoretical concepts, principles, and formal treatment of time provide a sound foundation for continuous and discrete modeling formalisms. For continuous and discrete-event models, time base is continuous (i.e., real) and for discrete-time models, time base is discrete (i.e., integer). Formal specifications for input, output, and state trajectories, and their transition functions support rigorous model development. Two well-known formalisms are *discrete-time specification* [26] and *discrete-event system specification* [27].

Systems theory offers two fundamental concepts. First, it enables characterization of a system at different levels of specification (e.g., input/output functions and state-space systems). Second, different types of formal specifications—continuous, discrete-time, and discrete-event dynamics—are supported within a framework such that they can be composed under well-defined conditions.

A system-theoretic specification is founded on the basic characterization of a system in terms of its *structure* and *behavior*. Behavior refers to the system's outwardly (visible) time-based manifestation as governed by its structure. Structure describes a system's architecture in terms of each component's structure and behavior and components' interactions. For example, the internal structure of a component can specify state set, state transitions, and pair-wise timed input/output data sets. Complex structures can be specified through hierarchical decomposition.

Another research direction which offers important concepts and methods for separation and composition of knowledge comes from the artificial intelligence (AI) community. An example of an AI approach for modeling complex systems is INTERRAP [18] which is based on Reactive Action Planner (RAP) [4]. It models a system's processes, control, and planning via a layered architecture in order to enable separation of concerns and systematic interaction and cooperation among agents.

2.1. Related Work

The software engineering paradigm largely leans toward specifying a system in terms of semi-formal modeling techniques. This paradigm often results in composing systems where the specifications of sub-systems have to be represented within a single modeling framework such as UML. Since a generalized framework must account for a variety of specifications (e.g., Activity and Statechart Diagrams) which do not have a common semantics, it becomes necessary to depend on the lowest common denominator to develop models and rely on defining concepts and abstractions using programming language constructs.

Other approaches focus on component-based modeling where a computational framework is used to execute finite state machines and, discrete-event, and continuous models (e.g., [16]). The theoretical basis of this approach, known as

Ptolemy II, relies on token-based dataflow to combine execution of models of mixed signal and hybrid systems. This approach formalizes input/output interactions among actors (model components) under the control of directors using interface automata [3]. This formalization is specialized for different computational domains including Communicating Sequential Processes and Process Networks. This approach can also be viewed as multi-formalism but with a software engineering focus. Another approach to composability is based on mapping some related formalisms (differential and algebraic equations, Petri nets, and finite state machines) into the DEVS formalism—this is a realization of a meta-modeling concept [14].

Synthesis of simulation and physical components can also be carried out using *computational composability* and in particular interoperability. Mature technologies exist for enabling interoperability with simulation engines and other applications (see [8]). Interoperability enables different software programs to exchange information using service-oriented framework—i.e., offering different suites of generic services which can be brought together to satisfy specific needs.

Interoperability by itself, however, cannot ensure that the integrated models perform in a semantically consistent manner. Instead, if models are composed in a manner that is syntactically and semantically well-defined, then interoperability can provide a precise basis (methods and technology) for realization using appropriate software environments.

In addition, platform-independent business logic model development can be used to derive a platform-specific software model and subsequently automatic generation of target application code (known as *Model Driven Architecture* (MDA)). Two of the MDA specifications are Meta-Object Facility and XML Metadata Interchange, which are important for ensuring different syntactic specifications work together.

Related to this work is the integration of system-theoretic modeling with a non-monotonic logical reasoning to support inductive modeling. This approach enables reasoning of time-based input/output model dynamics in a synchronous setting [20]. More closely related to the work presented here is the composability of (a) discrete-event (DEVS) and (b) linear programming (LP) [11] for the semiconductor supply-chain domain, in which we describe a knowledge interchange broker to support i) detailed process and decision models for semiconductor supply-chain networks and ii) configurable data aggregation/disaggregation mapping. Another closely related work addresses composition of the DEVS and Model Predictive Control model specifications [22].

3. Modeling Composability Framework

A useful concept underlying the success of modeling approaches is the separation of a model's specification from its execution protocol. For example, separation of a simulation model description (specification) from its execution (simulator) allows composed models, expressed within one modeling formalism, to be executed using alternative simulation protocols in both single or multi-processors [24]. This separation is important when applied to formalism composability. In particular, it provides a basis to address the consensus that approaches and technologies based on the concept of interoperability[1] by themselves cannot support component-based (modeling & simulation) composability (e.g., see [2]).

We consider a modeling formalism to consist of a model specification and an execution protocol with a unique syntactic and semantic characterization[2] . We use "the separation of model specification and execution protocol" as our conceptual basis for the multi-formalism modeling composability framework. As stated earlier, the main idea is to "compose modeling formalisms" instead of "composing models". The difference is that without composability at the level of modeling formalisms, composability can only be achieved via interoperation ranging from low-level programming constructs to high-level middleware services. Consider two modeling formalisms suitable for specifying and executing a vehicle guided to a destination as an example. One modeling formalism can be discrete-event specification and its associated simulation protocol. Another can be of an agent specification and its interpreter. Formalism composability, therefore, is defined as a pair consisting of two specifications (e.g., DEVS and RAP specifications) and the interaction of their executors (e.g., DEVS simulator and RAP interpreter) [23].

As shown in Figure 1 the DEVS formalism is suitable for representing the vehicle dynamics and the RAP formalism for the agent decisions. Here, we refer to process flow (movement of vehicles) and decision making (paths to follow) as distinct *layers* to emphasize that the agent model and vehicle simulation model represent two levels of abstraction of the overall system-i.e., the agent model describes higher-level knowledge (decisions) compared to the vehicle simulation model (operations). An important benefit of this approach is that the general-purpose multi-formalism model composition can support different do-

[1] Interoperability refers to how two or more simulation/execution applications can work together either using middleware (e.g., message exchange and time management protocols) or inter-process services. As mentioned above, several differences between composability and interoperability have been described.

[2] A formalism may have several implementations. While each can be consistent with the formalism specification, the implementations may differ due to lower level expressiveness used in their designs.

mains (see Figure 1). Furthermore, this approach naturally applies to distributed systems where physical process flows and logical decision making can be executed separately.

Therefore, with this framework, we have a basis to characterize *model heterogeneity*—i.e., composition of a larger model composed from smaller model components described in different modeling formalisms. For example, this approach supports composability of DEVS and RAP formalisms upon which a separate layer of domain-specific network systems can be specified such that the structure and behavior of the composed models are consistent within the DEVS/RAP multi-formalism framework using the *Composition Specification* and its corresponding *Executor* (see Figure 1). The KIB formalism, therefore, allows specification of all models to include proper syntactic structure and behavioral semantics with generalized support for data and control between the DEVS and RAP formalisms.

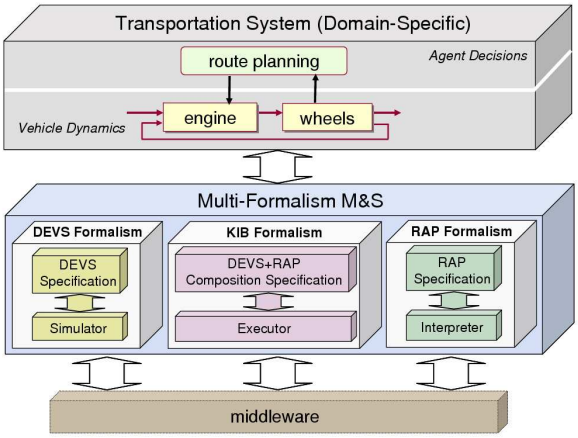


Figure 1. Multi-formalism modeling composability framework.

It should be noted that no modeling formalism can ensure validity of models universally; rather, models can be shown to be i) syntactically correct and ii) semantically valid for a well-defined set of requirements. The key (two-fold) capability is to support development of DEVS and RAP models separately and also to allow their composition. That is, the DEVS and RAP modeling formalisms are composed to allow for (i) exchange of appropriately mapped outputs to inputs data (messages) and (ii) appropriate data transformation under a well-defined interaction protocol (at various levels of homomorphism).

Therefore, the composition of two formalisms is defined in terms of the Composition Specification, and its corresponding Executor as depicted in Figure 1. In contrast, "pseudo composability" can be supported by using low-level generalized services (e.g., [13]) that are specialized for high-level modeling formalisms. Next, we consider the KIB's Composition Specification and Executor for the DEVS and RAP formalisms.

3.1. Composition of DEVS and RAP Formalisms

We begin with brief descriptions of the Discrete-Event System Specification (DEVS) [27] and Reactive Action Planning (RAP) [4, 5] formalisms. The DEVS atomic model specification—$\langle X, S, Y, \delta_{int}, \delta_{out}, \delta_{conf}, \lambda, ta \rangle$—allows representing state transitions using internal transition function (δ_{int}) in the absence of external events which can be received on input ports. Modeling of arbitrary arrival of input events is characterized using the external transition function (δ_{out}). Concurrent internal and event transitions can be specified with the confluent transition function (δ_{conf}). The output function (λ) specifies mapping of states (S) to outputs (Y). The time advance function (ta) determines the holding times for states (S). Input and output events are associated with ports (i.e., $X \doteq (port, value)$). The atomic models can be coupled hierarchically using output to input, input to input, and output to output port couplings.

The Reactive Action Planning system consists of an Interpreter, Task Agenda, Monitor Agenda, and a library of Reactive Action Packages (RAP) where hierarchical, sequential, or parallel tasks can be executed given the state of the world and actions to be taken to satisfy some logical conditions. Each RAP specification is described using a set of well-defined formulas in mathematical logic—i.e., $A_1 \ldots A_k \vdash W$ where A_i's for $i = 1, \ldots, n$ are facts and W represents all possible logical deductions given a set of axioms. Therefore, as shown in Figure 2, the RAP library consists of a collection of rules expressed as logical clauses where index is a unique identifier, succeed, fail, precondition, and constraints specify conditions in which the RAP can be used, and retries limits the number of retries (see below). The on-start, query, etc are task nets (i.e., method) to be executed when the RAP task begins and finishes.

(define-RAP index (succeed query)(fail query)(precondition query)(constraints query)(retries query)(on-start |query |failure |success |finish task-net forms)(method plan-for-carrying-out tasks))

Figure 2. Syntax for Reactive Action Packages.

Each RAP defines a group of possible ways a task may be carried out given different world situations. RAP events are asynchronous messages which may be generated internally or externally. The elementary constructs of RAP are

query and action (command) events. These events change the memory of RAP. Each of these is specified using (event-name. args)-e.g., (position A x y) specifies vehicle A's x and y coordinates. The expression used in the events can have numeric and logical operators.

The Task Agenda consists of a set of tasks which may be generated externally or internally. These tasks may come from a planner supporting a particular set of goals for a specific domain (e.g., guide a vehicle to a destination from a starting location). To support asynchronous handling of tasks, the RAP Monitor is defined similar to RAPs where start, active, trigger, and reset constructs are included. The primitives wait-for-time and wait-for-event time constructs are defined for the RAP Monitor. These can be used to manage situations within which the RAPs can be processed since RAPs are not specified in terms of time. The Monitor Agenda, although not strictly necessary, can be used to monitor situations when specific events in the world require placing tasks in the Task Agenda (e.g., upon detecting low fuel level, the agent guides the vehicle to a location for refueling). The Interpreter is a type of resolution theorem-prover that can correctly execute the logical formulae in the memory and the RAPs.

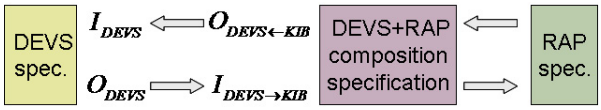
3.2. KIB Specification

The composition of two modeling formalisms is defined via a generalized *Knowledge Interchange Broker*[3] (KIB), which specifies *event mapping*, *synchronization*, *concurrency*, and *timing* properties. These properties can be described in terms of structure and behavior of the composition specification and executor as exemplified using the DEVS and RAP formalisms [23]. The proposed approach to multi-formalism composition, therefore, is based on characterizing how two formalisms can interact with one another. Input and output mappings are required to support composition of models based on message types and interactions which are syntactically and semantically well-defined for the modeling formalism as shown in Figure 3.

For example, the vehicle needs to receive and process input events such as adjust-speed (object, v) from the KIB. The agent command must be mapped into the input event of the vehicle model (speed, v) where speed is the input port name and v is the agent's demand velocity. In contrast, since the RAP events do not have ports, the KIB needs only to send the values of the DEVS output messages. These mappings or translation ($I_{KIB} \mapsto O_{KIB}$) are carried out in the KIB instead of being handled inside the DEVS and RAP models (refer to Figure 3).

[3]The term KIB is coined in reference to the Knowledge Interchange Format (KIF) [9]. KIF is useful for information interchange between disparate computer languages. Its level of abstraction (programming languages), however, does not lend it to describing model specifications and therefore model composability as presented here.

The messages (events) exchanged between DEVS and RAP can be arbitrarily complex—e.g., an input message sent from DEVS to RAP can be moveToPosition(vehicle, x, y). Obviously, the KIB is tasked with bi-directional message translation and mapping and therefore it accounts for the structural aspect of model composability—it must handle the disparity between the DEVS and RAP inputs and outputs. Message mappings can be specified such that two messages have *identity*, *isomorphic*, and *homomorphic* relationships. The messages expressed in two distinct modeling formalisms can have simple structures (e.g., a string) or complex structures (e.g., a list of objects). Thus, a suite of *data translation* schemes based on the two participating modeling formalisms and domain knowledge mappings is needed. The data translation can be arbitrarily complex since the generalized DEVS and RAP message structures can be specialized to meet domain specific needs.



$$I_{KIB} = I_{DEVS \rightarrow KIB} \vee I_{KIB \leftarrow RAP} \text{ and}$$
$$O_{KIB} = O_{DEVS \leftarrow KIB} \vee O_{KIB \rightarrow RAP} \text{ where}$$
$$I_{DEVS} \doteq O_{DEVS \leftarrow KIB}, O_{DEVS} \doteq I_{DEVS \rightarrow KIB},$$
$$I_{RAP} \doteq O_{KIB \rightarrow RAP}, O_{RAP} \doteq I_{KIB \leftarrow RAP}$$

Figure 3. DEVS/RAP KIB I/O mapping and transformation.

The message mapping $M \mapsto N$mentioned above is based on the well-known concepts of knowledge reduction and augmentation. *Knowledge reduction* is simpler, in comparison with knowledge augmentation, since the KIB needs to discard information in the process of translating one message type to another. For example, given any DEVS model message (port, value), the port information needs to be discarded. In contrast, for *knowledge augmentation*, consider a RAP message. Since the RAP messages do not have ports associated with them, the KIB must assign ports to these messages before they can be sent out as DEVS messages. To handle this, the DEVS formalism can be extended to have a pair of unique single input and output ports dedicated to receiving and sending messages to and from the KIB. The KIB therefore needs to support not only message *decoding* but *encoding* as well. Further more, to deal with the complexity of information mapping, the KIB can provide a basis for handling different types of data abstraction and concretization (i.e., aggregation or disaggregation).

In addition to the structural specification of the KIB, we also need to specify its *behavioral aspect*. The specification must account for ordering of messages exchanged. The control of messages among DEVS, KIB, and RAP can be described in terms of *synchronization*, *concurrency*, and *timing* concepts and their specifications. A DEVS model may send an output message and receive an input message from the RAP model. Consider a situation where RAP is guiding the vehicle based on its amount of available fuel. Since the vehicle may send this information while also receiving a query from RAP about its speed, the KIB specification must support synchronization of multiple messages to be sent and received.

Related to synchronization is *concurrency* of messages. The parallel DEVS formalism supports concurrent execution of model components and multiple input and output events. It supports sending and receiving multiple events through multiple ports. RAP, unlike DEVS, can send its command and query messages sequentially. The capability to handle concurrent events is important. Consider the situation where the vehicle sends the distance left to reach its destination and its current location at the same time. The KIB needs to be able to receive these concurrent messages from the vehicle model and send them to the agent model in some order. Since these DEVS can send messages simultaneously, the KIB must order the DEVS messages before sending them to the RAP. This is appropriate since concurrent DEVS messages are not causally dependent, and the messages sent to the RAP need to be ordered based on the domain-specific choices to be supported in the RAPs.

The KIB may be defined to execute while consuming (logical) *time*. Since the DEVS formalism explicitly accounts for passage of time, and the RAP formalism does not, the use of time in the KIB is not strictly required. This is because the RAP can respond to queries in zero logical time. Similarly, it can generate commands while consuming no logical time w.r.t. the DEVS simulation protocol. As mentioned earlier, the RAP tasks may account for passage of time via it's the RAP Monitor. In this case, the RAP tasks can be executed using wait-for-time and wait-for-event time constructs defined within the RAP Monitor. The presence of logical time in both DEVS and RAP can require the KIB to manage the ordering of its messages. This may be achieved with a KIB that uses a reference time assuming its mappings do not consume logical time. Alternatively, the processing of messages may be defined using time-based (causal) ordering.

The combination of synchronization and concurrency is used to define the DEVS+RAP Composition Specification control protocol. The ordering of messages in the KIB—messages sent to and received from the DEVS and RAP models—is based on DEVS sending a message to RAP followed by receiving a response (message) from the RAP. In this scheme the interaction between RAP and DEVS is circumscribed based on the DEVS abstract simulator protocol. Once the output messages of the DEVS are sent to the KIB, the DEVS simulator is stopped until a response is received from the KIB (and therefore RAP). The KIB's executor, therefore, is responsible for synchronous message interactions between DEVS and RAP while no logical time is consumed with respect to the DEVS simulator. That is, each cycle of interaction between DEVS and RAP (messages sent and received via the KIB) occurs during one DEVS simulation cycle—i.e., the simulation clock remains unchanged and therefore no clock is necessary in the specification of the KIB executor. Under this interaction regime, local control in the process and planning models can be specified independently of the KIB control protocol. This type of synchronization is implemented in the DEVS/RAP environment which is briefly described in the next section.

Aside from the just described synchronous executor control protocol, DEVS and RAP may also interact via an asynchronous executor. In general, we can define two types of asynchronous interactions depending on whether or not one or both of the formalisms are time based. For example, the RAP system may use a Monitor to manage creation of RAP tasks in response to unexpected events. Without the RAP Monitor, the synchronous KIB specification is simpler and sufficient if the RAP responses can be considered to happen instantaneously. In contrast, the KIB can be asynchronous. The KIB asynchronicity can be defined based on the causality of the DEVS and RAP messages generated according to the DEVS simulation protocol and the RAP interpreter. With asynchronous process flow, the vehicle can continue to travel along its current route while the agent is devising new (updated) routes. The asynchronous interaction with explicit accounting for time can be defined in terms of (logical or real-time) clocks. That is, with a time-based KIB executor, the DEVS simulator and the RAP interpreter can execute concurrently while handling message mapping and interaction.

4. DEVS/RAP Environment

The multi-formalism modeling framework presented in the previous section can be developed using different software design techniques and programming languages. Here we describe a high-level software design for the DEVS/RAP KIB environment. We highlight its basic capabilities—message mapping and synchronization of events—and how they are supported. Two main elements of the DEVS/RAP KIB environment are shown in Figure 4 [23].

The KIB was developed in Java to simplify its interaction with DEVSJAVA [1]. Similarly, the interaction be-

tween the KIB and the RAP was developed in C++ since the RAP interacts with its outside world via C++ wrapper around MZScheme [5]. The software design specification of the KIB was developed in accordance with Figure 3—the structure and behavior of the KIB message mappings and synchronization are consistent with those that are defined above and those of the DEVS and RAP formalisms. The executor of the DEVS+RAP Composition Specification is responsible for message mapping and controlling the interactions between DEVSJAVA and RAP. In this design, ordering and mapping of DEVSJAVA and RAP messages are assigned respectively to the JSimMessageManager and CRAPMessageManager. The ordering of messages to and from the KIB was designed into the RapBridge and SimulatorBridge packages and their interfaces, respectively. We used JNI as our primitive connectivity between the Java and C++ programming languages.

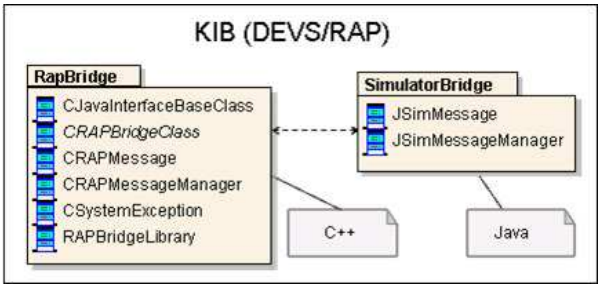


Figure 4. DEVS/RAP KIB implementation.

The control of the composed models is initiated by DEVSJAVA and is maintained by the KIB. This choice of initialization and control was in part due to the need for the Java Virtual Machine to create the RAP system and to load the RAPs or agent models. As mentioned above, the KIB synchronization between the DEVSJAVA and RAP system is referenced with respect to the DEVSJAVA simulator cycle.

5. Conclusions

We have proposed a new framework where combined process dynamics and decision making models—in particular, discrete-event processes and agent-based planning—can be developed within a model-theoretic composability approach. This framework is useful for characterizing the structure and behavior of a system's model whose overall dynamics are derived from its sub-systems' dynamics. This work offers a generalized basis and a methodology toward model validation and execution verification for heterogeneous systems. Our ongoing research is focusing on time-based asynchronous composability and a hierarchical knowledge interchange broker where three or more modeling formalisms are to be composed [11, 22].

6. Acknowledgements

This research was initially supported by Advanced Simulation Center, Lockheed Martin in Sunnyvale, California and subsequently by the Intel Research Council, Chandler Arizona. Steven Hall from the Advanced Simulation Center provided key insights on difficulties for integrating the RAP and the DEVS-C++ models. The first author acknowledges Gary Godding of the Arizona State University and Karl Kempf of the Intel Corporation for stimulating discussions about data transformation in complex domains. The DEVS/RAP environment has been developed jointly with Jeff Plummer of General Dynamics and with the help from Preston Cox of Lockheed Martin. Thanks to Elizabeth Pulcini for her help in preparing this manuscript.

References

[1] ACIMS. *Arizona Center for Integrative Modeling and Simulation*, 2001 [cited 2005]. Available from: http://www.acims.arizona.edu/SOFTWARE.

[2] P. Davis and R. Anderson. *Improving the Composability of Department of Defense Models and Simulations*. RAND, Santa Monica, CA, 2004.

[3] L. de Alfaro and T. Henzinger. Interface automata. In *Proceedings of the 8th European Software Engineering Conference*, Vienna, Austria, 2001.

[4] R. Firby. *Adaptive Execution in Complex Dynamic Worlds*. PhD thesis, Computer Science Department, Yale University, New Heaven, CT, 1989.

[5] R. Firby and W. Fitzgerald. *The* RAP *System Language Manual, Version 2.0*. Neodesic Corporation, Evanston, IL, 1999.

[6] K. Fisher, J. Müller, and M.Pischel. Cooperative transportion scheduling: An application domain for dai. *Applied Artificial Intelligence, Special Issue on Intelligent Agents*, 1(10):1–33, 1996.

[7] P. Fishwick. Next generation modeling: A grand challenge. In *International Conference on Grand Challenges for Modeling and Simulation*, San Antonio, Texas, USA, 2002. SCS.

[8] R. Fujimoto. *Parallel and Distributed Simulation Systems*. John Wiley and Sons Inc., 2000.

[9] M. Genesereth and R. Fikes. *Knowledge Interchange Format Reference Manual, Version 3*. Computer Science Department, Stanford University, 1992.

[10] M. Genesereth and N. Nilsson. *Logical Foundation of Artificial Intelligence*. Morgan Kufmann, San Mateo, CA, 1987.

[11] G. Godding, H. Sarjoughian, and K. Kempf. Multi-formalism modeling approach for semiconductor supply/demand networks. In *Proceedings of Winter Simulation Conference*, Washington DC, USA, 2004.

[12] M. Hocaoglu, H. Sarjoughian, and C. Firat. DEVS/RAP: Agent-based simulation. In *AI, Simulation and Planning in High-Autonomy Systems*, Lisbon, Portual, 2002. SCS.

[13] IEEE. *HLA Framework and Rules*. IEEE, 2000. IEEE 1516-2000.

[14] J. Jaramillo, H. Vangheluwe, and M. Moreno. Using meta-modelling and graph grammar to create modelling environments. *Electronic Notes in Theoretical Computer Science*, 3(72):1–15, 2002.

[15] K. Kempf. Control-oriented approaches to supply chain management in semiconductor manufacturing. In *Proceedings of IEEE American Control Conference*, Boston, MA, USA, 2004.

[16] E. Lee. What's ahead for embedded software. *IEEE Computer*, 9(33):18–26, 2000.

[17] P. Mosterman and H. Vangheluwe. Guest editorial: Special issues on computer automated multi-paradigm modeling. *ACM Transaction on Modeling and Computer Simulation*, 4(12):249–255, 2002.

[18] J. Müller. The design of intelligent agents: A layered approach. In *Lecture Notes in Artificial Intelligence*. Springer, 1996.

[19] H. Praehofer. *System Theoretic Foundations for Combined Discrete Continuous System Simulation*. PhD thesis, Institute of Systems Science, Department of Systems Theory and Information Engineering, Johannes Kelper University, 1991.

[20] H. Sarjoughian. *Inductive Modeling of Discrete-event Systems: A TMS-based Non-monotonic Reasoning Approach*. PhD thesis, Electrical and Computer Engineering, The University of Arizona, Turson, AZ, USA, 1995.

[21] H. Sarjoughian and F. Cellier, editors. *Discrete Event Modeling and Simulation Technologies: A Tapestry of Systems and AI-Based Theories and Methodologies*. Springer Verlag., 2001.

[22] H. Sarjoughian, D. Huang, and et al. Hybrid discrete event simulation with model predictive control for semiconductor supply-chain manufacturing. In *Proceedings of Winter Simulation Conference*, Orlando, FL, USA, 2005.

[23] H. Sarjoughian and J. Plummer. Design and implementation of a bridge between RAP and DEVS. Computer Science and Engineering, Arizona State University, Tempe, AZ, 2002.

[24] H. Sarjoughian and B. Zeigler. DEVS and HLA: Complementary paradigms for modeling and simulation? *Transaction of the Society for Modeling and Simulation International*, 4(17):187–197, 2000.

[25] E. Weisel, M. Petty, and R. Mielke. Validity of models and classes in semantic composability. In *Simulation Inteoperability Workshop*, Orlando, FL, USA, 2003.

[26] A. Wymore. *Model-based Systems Engineering: an Introduction to the Mathematical Theory of Discrete Systems and to the Tricotyledon Theory of System Design*. CRC, Boca Raton, 1993.

[27] B. Zeigler, H. Praehofer, and T. G. Kim. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press, 2nd edition, 2000.