

An Approach for Designing and Enacting Distributed Simulation Environments¹

Luciano Baresi

Dipartimento di Elettronica e Informazione-Politecnico di Milano
Piazza L. da Vinci, 32 - 20133 Milano (Italy)
+39 022399 3638 - baresi@elet.polimi.it

Alberto Coen Porisini

Dipartimento di Ingegneria per l'Innovazione-Universita' di Lecce
via per Monteroni - 73100 Lecce (Italy)
+39 0832 320226 - coen@ingle01.unile.it

Abstract

Simulation provides a flexible and cost-effective way to assess system design, configuration and control strategies in many domains. Currently, the impact of simulation is less than what one could expect because of (1) the difficulties that arise in integrating different simulators and (2) the lack of an integrated design environment that could allow one to design simulations while designing a system. This paper addresses the second problem by presenting the work done in the ESPRIT ASIA project.

The paper presents a Simulation Architecture Description Language (SADL) whose aim is to describe all artifacts produced during the different design phases allowing one to smoothly move from the design phase to simulation within a single framework.

Keywords: Simulation, System Engineering, Customizable Environments

1. Introduction

Simulation plays an important role in system-engineering processes of many different domains such as space, telecom, traffic management, flexible manufacturing systems (FMS) etc. System engineers use simulation to design, configure and evaluate new applications, and to manage day-to-day activities during system operation. Thus, simulation provides a flexible and cost-effective way to assess system design, configuration and control strategies, and preserves the safety of the controlled environment.

Currently, the impact of simulation is less than what one could expect because of (1) the difficulties that arise in integrating different simulators and (2) the lack of an integrated design environment that could allow one to design simulations while designing a system. The former issue is due to the current situation in which simulators are used in a stand-alone way rather than in a cooperative way, which would provide the simulation of the overall system by making simulators interact. The latter issue requires to tightly couple design and simulation, thus providing a single and complete model of the overall system. Both problems are addressed by the ESPRIT ASIA project [1], whose aim is to solve the above mentioned problems by defining and implementing an open platform for supporting both design and simulation activities and allowing an effective integration of simulation tools.

In this paper we report on the work done in the ASIA project by focusing mainly on the definition of the design environment. In particular the paper presents a Simulation Architecture Description Language (SADL) whose aim is to describe all artifacts produced during the different design phases allowing one to smoothly move from the design phase to simulation within a single framework.

The definition of an environment for simulating complex systems requires to clearly identify all the activities that occur when designing/simulating a system along with the right SADL to properly represent the results of each activity. In fact the design process guides system engineers through the enactment of their systems. Both the design process and the SADL must be as "general" as possible in

¹ This work has been partially supported by CEC under the ESPRIT ASIA project (EP 28661)

order to be applicable to the different domains that rely on simulation. However, since designers are usually experts of their own domain, both the design process and the SADL need to be as tight as possible to the particular domain in which they are expected to be used.

In order to cope with these two conflicting requirements ASIA has defined a design process made up of a set of “macro” activities borrowed from current practice, which are general enough to be applied to almost any domain. The definition of the SADL is based on a *double language* approach [2] that provides a “general” abstract notation, which is domain-independent, along with specific concrete notations, one for each domain. The abstract notation, hereafter *core SADL*, is defined as a simulation-oriented “reuse” of UML (Unified Modeling Language [3]). Domain-specific notations are obtained by means of the customization facilities of UML, which allow one to define translation schemas from *core SADL* in order to present it in a way suitable to the different domain experts. In this way, one can define a specific notation for each domain as a transformation from the core notation. Once these transformations are defined, users can think and work using their own notations, relying on the core (hidden) notation to check and validate their models.

Both the core and the domain-specific notations use OCL (Object Constraint Language [4]) to increase the consistency and validation capabilities offered by SADL.

The rest of this paper is organized as follows. Section 2 presents the related works. Section 3 discusses the adopted simulation design process. Section 4 presents the metamodel of the SADL. Section 5 discusses how the different phases of the process are supported by SADL while Section 6 introduces domain-specific notations and exemplifies them on a concrete example from the space domain. Section 7 shortly describes a prototype supporting SADL. Finally, Section 8 draws some conclusions.

2. Related Work

The work presented in this paper borrows concepts and ideas from several domains: Customizable graphical editors, object-orientation, distributed systems, architecture description languages, and integrated simulation environments.

Customizable graphical editors [5,6] limit themselves to supply users with graphical editors whose shapes and symbols can properly be refined. [7, 8] allow

users to associate their languages with complex syntaxes through graph-based tailoring rules and an underlying repository that oversees the (syntactic) consistency of designed models. They usually offer more “tailoring features” than we need, but they deal with neither the semantics of the defined languages nor the features introduced by special-purpose design processes.

Architecture Description Languages (ADL) offer general purpose modeling features, but they do not cope with the needs introduced by simulation environments. Wright [9] Darwin [10], Rapide [11] and many others can be considered suitable languages -- to different degrees -- to model “general purpose” distributed systems, but not simulation environments. Including these notations in our approach would have meant proposing particular (meta) models defined using a given notation, instead of proposing a special-purpose notation.

As to software architectures, our work has more similarities with the work by Medvedovic et al. [12]. We both start from UML and we tailor it for architectural purposes. The main difference is once more the particular focus we chose for our approach and notation. They aim at studying and proposing UML as ADL for almost all software systems. In contrast we do not aim at such generality, but we would like to define powerful and suitable means for simulation experts.

Several works are also studying the application of object-oriented and distributed approaches to develop simulation software. In particular, [13] advocates the use of an object-oriented approach in FMS systems. The work done within the OMG has led to the definition of the CIM Framework architecture [14], which is a component-based architecture for the integration of simulators in the context of semiconductor environment. Other related works concerns the introduction of data standard or language definition to describe simulation models [15, 16]. Again, our approach differs from these proposals for its generality, that is domain independent, while the others are specific to an application domain. Moreover, we propose a unified approach, together with a supporting notation, for designing simulation environments, rather than aiming only at solving the problem of integrating different simulators. This latter problem is also addressed in the context of ASIA.

Finally, particular attention is deserved to the High Level Architecture proposed by DMSO [17]. This is the most widely known effort that defines requirements and guidelines for “integrating

simulators. The proposal considers only the last step of the approach we propose. They are not interested in an integrated methodology for designing distributed simulation environments. They define only practical rules to integrate (existing) simulation components and enact the whole system.

3. Simulation Design Process

Simulation can be viewed as a fundamental part of system engineering processes having a strong impact on the design activities that are carried out to build any non-trivial system. Thus, in what follows we will briefly discuss a process lifecycle which is referred to as the *simulation design process*, even though what it is taken into account it is actually the design process of a system supported by simulation activities. The simulation design process is presented in Figure 1 and comprises four main activities

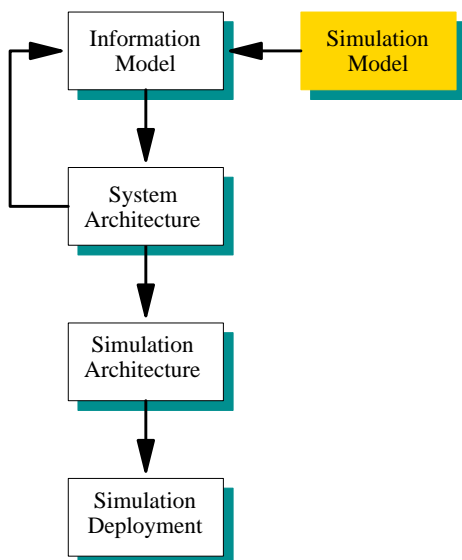


Figure 1: Simulation Process Lifecycle

1. Designing the *Information Model* means to define the elements that belong to an application domain. Usually such elements represent the different components every system can be made up of. The focus on simulation imposes that engineers provide also the *simulation models* that could be used to simulate one or more relevant characteristics of the elements. As a result the Information Model defines what can be simulated. Several simulation models can be associated with the same element. In fact, different simulators could be based on different mathematical models and/or address different aspects of the same element. For example, in the space domain, an element could be an antenna, which is

characterized by a set of properties such as its dimensions and the frequency range, and by one or more simulation models that can be executed to provide information about the functional behavior, the dependability etc. However, simulation models are external entities: They depend on the actual simulators used to enact the system and thus are not considered in this paper.

2. Designing the *System Architecture* means to build a system by instantiating and composing the elements composing the information model. The outcome of the system architecture is a "complete" system, that is, a set of components instantiated from the information model, their mutual relationships, and all associated simulation models. As a result a system architecture defines what has to be simulated.
3. Designing the *Simulation Architecture* means defining how the system architecture has to be simulated, that is system engineers define which simulations will be performed, what simulation models are used, and how they are grouped and organized to carry out the simulation. In the previous phases the simulation models have been (implicitly) associated with every component. In this phase, we must focus on a single aspect and identify which simulation models will interact to provide the desired results. In other words the simulation architecture defines the data and control flows among simulation models.
4. Designing the *Simulation Deployment* means defining the deployment of the simulation architecture. The components of the simulation architecture, that is, the simulation models, are associated with simulators using the necessary software tools². This activity might require sub-phases to set all simulation parameters and map simulators onto available physical machines.

The design process presented so far can be considered as a reasonable approximation of an actual process of simulation-based development. The different phases are described using a waterfall approach for the sake of clarity. It goes without saying that any real design process is not a bare sequence of activities, but feedback from one phase to the previous ones is often necessary and even desirable [18].

² Simulators can be virtual machines that execute the components identified in the simulation architecture, or can be obtained by "compiling" the aforementioned components

For example, when designing a system architecture, engineers might need to introduce new components, which were not previously defined in the information model. Thus, the arrow of Figure 1 between system architecture and information model represents possible iterations between the definition of new elements and the design of the system, that is, their instantiation.

4. SADL Metamodel

SADL is the architectural notation that oversees all design activities. It is defined by tailoring UML to simulation problems and aims at being the notation that can be used to document the four phases of the design simulation process. The definition of SADL is based on a metamodel whose aim is to provide a formal description of all the entities that are relevant in the simulation design process. The metamodel is described by means of a UML class diagram as shown in Figure 2. In what follows we discuss the main classes of the metamodel with reference to the activities identified in the previous section. For the sake of simplicity, the metamodel contains only the main (meta)classes; interested readers can refer to [19] for the complete metamodel.

4.1 Information Model

An *Information Model* (modeled by class InfoModel)

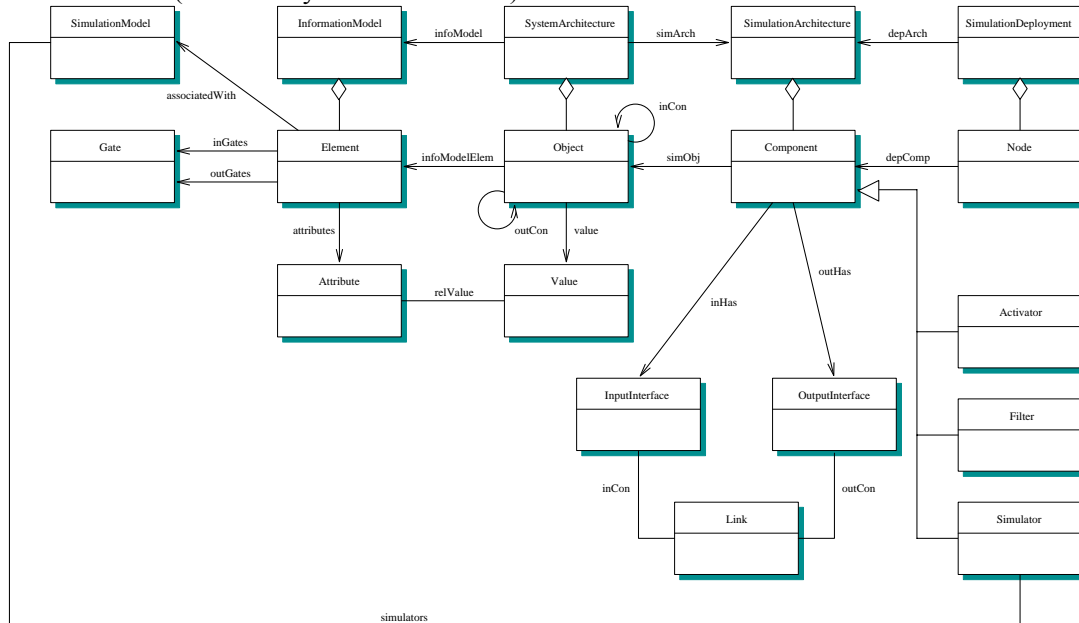


Figure 2: The simplified metamodel of CoreSADL

4.3 Simulation Architecture

A *Simulation Architecture* identifies the simulation

defines a library of reusable elements specific to a particular simulation domain. Thus, class InfoModel is associated with class Element. Each instance of class Element represents a single element belonging to InfoModel. The features of an Element are described by means of the associated instances of class Attribute. Moreover each Element is associated with 1 or more SimulationModels and defines one or more input (output) gates. Each gate represents an entry (exit) point through which information can flow. Gates are defined as follows:

```

element Gate {
    String name;
    String type;
    String constraints;
    String comments;}

```

4.2 System Architecture

A *System Architecture* is designed by instantiating and combining elements from a particular information model. Thus, each system architecture is modeled by class SystemArchitecture, which is composed of one or more instances of class Object each of which represents an instance of an element. Class Object is associated with class Value, whose instances represent the values associated with the corresponding instances of class Attribute.

components of the designed system architecture. A system architecture can be simulated using several different simulation architectures each of them

focusing on a different quality (functional behavior, dependability, etc). Moreover, simulation architectures must define how simulators cooperate to carry out the simulation. This is done by providing both a data flow description, that is, which information is exchanged among components, and the control flow description, that is, the way in which the different components interact to complete the simulation.

A Simulation Architecture (modeled by class SimulationArchitecture) is composed of instances of the abstract class Component each of which can be a:

- *SimulationComponent*, representing a simulation model of an object of the System Architecture. There must be at least one SimulationComponent for each Object in the System Architecture.
- *Filter*, representing a component that can perform some syntactic transformation on data. The role of a Filter is to transform data from one format to another so that two simulation models can actually exchange information even if they are based on different data representation.
- *Activator*, representing a component that can control the actual flow of execution within a Simulation Architecture.

Each Component comprises Input and Output gates, which are in turn linked by means of a SimulationLink. In particular, an input interface is the gate through which a component will receive data, an output interface is the gate through which a component sends data.

5. Core SADL

Core SADL is the language used to describe the different entities involved in the simulation design process. It is based on the SADL metamodel and provides the users with both a graphical and a textual notation based on UML. The textual notation is used to provide all the information that cannot be defined using the graphical representation.

5.1 Information Model

An Information Model is defined by a class diagram in which each class represents an element of the domain considered. It can comprise single classes and class collections. Single classes represent “simulable” entities. Aggregated classes can be used to both represent *simulation models*, and components of higher-level elements. Specialized classes represent “generic” entities together with their specifications. Aggregated classes and packages account for different

simulations to require either the higher level view of a simulation entity or the in-depth representation, that is, all its components. Thus, the information model allows different abstraction levels when defining a new element.

For example, Figure 3 shows the simple classes Ground Station and Satellite the latter being specialized in HEO Satellite and LEO Satellite together with its two simulation models mac and IP.

Each class (domain element) is characterized by a set of Attributes and gates whose aim is to define the properties of designed components. For example, connectivity among elements depends on the types associated with their input and output gates:

```

element Satellite {
    String name;
    Gate* inGates;
    Gate* outGates;
    String comments;}

```

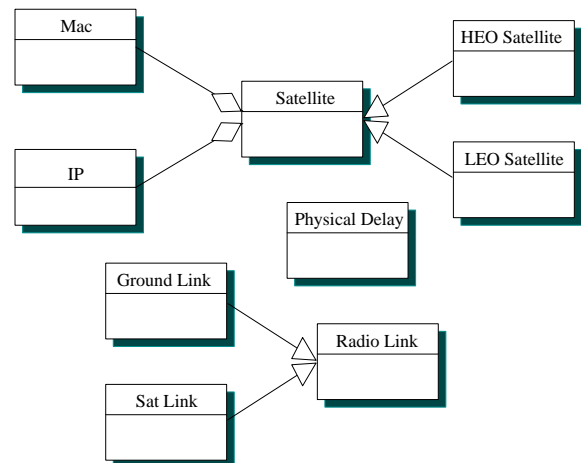


Figure 3: A sample Info Model for the space domain

Besides domain-specific information model, there is also a *standard* information model named Simulation InfoModel that comprises all elements that model general simulation components introduced in simulation architectures, that is, *Simulators*, *Filters*, *Activators*, *Gates* and *Links*. The role of the Simulation InfoModel is discussed in the Simulation Architecture Subsection.

5.2 System Architecture

A System Architecture is designed by instantiating and combining elements from a particular information model. Thus, each system architecture is a UML collaboration diagram with objects that are grouped using ellipses to identify possible sub-components.

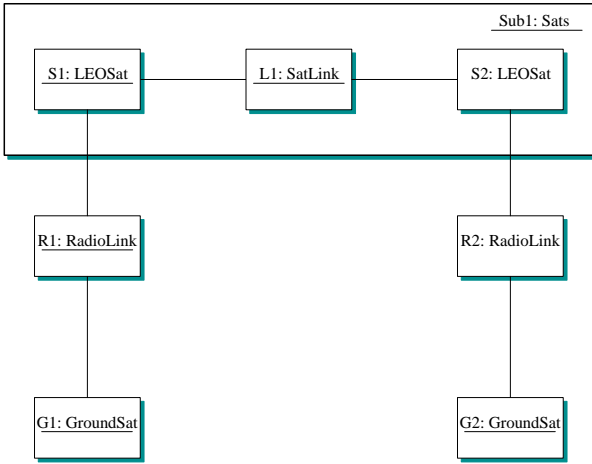


Figure 4: A system architecture for the components of Figure 3

For example, the system architecture of Figure 4 comprises eight elements: two objects of class LEO Satellite, two objects of class Radio Link and one object of class SatLink, specialization of the former. Finally two objects of class GroundStat. Links identify information exchange in terms of requested services (shown using the usual arrow with dotted tail). The architecture identifies also subsystem Sub1, which becomes a new reusable block at system architecture level.

5.3 Simulation Architecture

A Simulation Architecture is represented as collaboration diagram made up of instances of the elements defined in the *Simulation InfoModel*.

Simulators represent the simulation model associated with each component declared in the system architecture. Currently, the binding between simulator and simulated component is set through object references set according to the metamodel presented in Section 4.

The simulation architecture of Figure 5 is one of the possible solutions to “logically” enact the system architecture of Figure 4. All components of the system architecture are simulated by means of a single simulator, except component G2 that requires two distinct simulators (G2a and G2b). Input and output ports are properly paired. Fil1 transforms the output of L1 in a format suitable to both S1 and S2, while Fil2 does the same between S1- S2 and R2. The activator Act is used to activate either simulator G2a or simulator G2b as soon as the output from R2 is available.

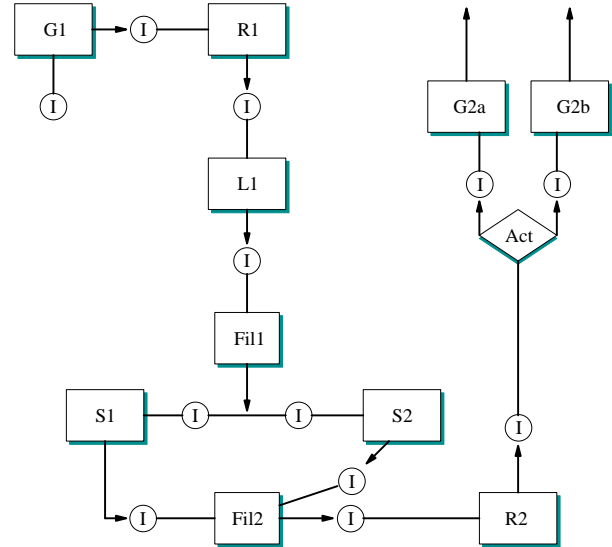


Figure 5: A Simulation Architecture from the system architecture of Figure 4

5.4 Simulation Deployment

A Simulation Deployment maps a simulation architecture on physical simulation nodes. Tailored UML deployment diagrams represent this information. A deployment diagram states how the components identified by the simulation architecture must be allocated on available processes (processors). An example is shown in Figure 6: Simulators are represented with cubes, components with the small tabbed rectangles, and inclusion represents component allocation. Cubes can be both processes on the same workstation or different processors on different workstations. Cubes are connected through undirected arcs, which represent the communication means

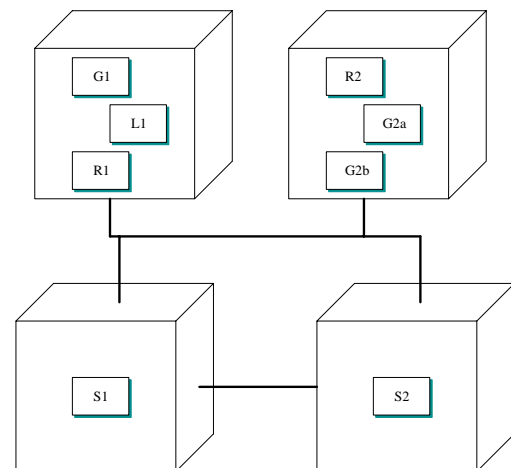


Figure 6: A Simulation Deployment from the simulation architecture of Figure 5

5.5 Consistency Checks

One of the main advantages of coupling a graphical notation with a textual notation is that the former is more suited for interacting with users, while the latter can more proficiently be employed to check the consistency of what has been defined. The simulation design process of Figure 1 identifies four phases. Each of these phases is based on the results obtained during the previous phases, and thus it is necessary to check whether each phase is consistent with respect to the previous ones. For instance, as stated before when designing a system architecture, it is necessary to check that the objects that are connected together can actually be connected according to their declared connectivity.

In what follows we identify the main consistency checks as invariant associated with classes of the meta-schema presented in Figure 2

- Check the consistency between a system architecture against the information model it belongs to, that is, all objects in the system architecture must be related to elements in the information model. This property can be stated in OCL as an invariant on class `SystemArchitecture`

```
self.objects->forall(syo |
  syo.infoModelElem->notEmpty and
  self.infoModel.elements->
    includes(syo.infoModelElem))
```

Given all objects of a system architecture `self`, for each selected object `syo`, its `infoModelElem` must be set and it must belong to the `elements` of the corresponding `infoModel`.

- Check the consistency of the connections in a system architecture, that is, each object can talk with the objects to which it is connected. This property can be stated in OCL as an invariant on class `SystemArchitecture` of Figure 2:

```
self.objects->forall(syo |
  syo.infoModelElem.inGates.compatible(
    syo.inCon.infoModelElem.outGates) and
  syo.infoModelElem.outGates.compatible(
    syo.outCon.infoModelElem.outGates))
```

Given all objects of a system architecture `self`, for each selected object `syo`, its `inGates` must be compatible with the `outGates` of the object `inCon` to `syo` and the same must hold true `outGates`.

- Check that the simulation architecture is a correct

refinement of the system architecture, that is, every object in a system architecture is modeled at least by one simulation model in the corresponding simulation architecture. This property can be stated in OCL as invariant on class `SystemArchitecture` of Figure 2:

```
self.objects->forall(syo |
  self.simArch.components->
    exists(sic | sic.simObj = syo))
```

Given all objects of a system architecture `self`, for each selected object `syo`, there must exist a simulation component `sic`, which must belong to the `simArch` with which the system architecture is associated, that simulates (`simObj`) `syo`.

6. Domain Specific Notations

Core SADL is not oriented to any particular application domain. It is a customization of a general-purpose object oriented notation, and thus has the advantage of covering all the possible domains, since what guided us in selecting the different diagrams was the concepts underlying the simulation process lifecycle.

However, one of the major drawbacks of this notation is the gap that exists with the current notations used in the different application domains. They are usually *ad hoc* notations rooted into the concepts specific to the domain they address. Thus, to bridge this gap, without losing generality, we must use an approach in which different representations of the same concepts coexist, that is, a dual language approach [2]. The core notation is used internally. It is not viewed by users (except when defining the information model). The domain-specific representation is the means employed by users to define their models.

The transition from one representation to the other is extremely simple in SADL. We exploit the customization facilities of UML to define the translation. In UML, each particular stereotype, that is, specific class of elements, can be associated with a user-defined shape. This simple mechanism allows us to almost freely move from the core notation to all possible “customizations”. In a given sense, these capabilities let us confuse core and domain-specific representations, without requiring complex translation mechanism.

In this way, the definition of a new domain-specific instantiation of SADL becomes only a problem of selecting the right shapes, that is, shapes that represent

concepts with which domain experts are familiar, and associating them with the equivalent concepts of the core notation.

7. Prototype

In the ASIA project a prototype of the environment described in the previous sections is under development. Such an environment consists of toolset supporting both the design activities and the actual integration of simulators by means of CORBA. In this paper, we present only the main features of the design environment.

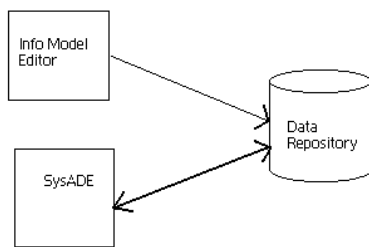


Figure 7 The ASIA Architecture

The design environment is made up of two different tools that can exchange information by sharing a common data-repository as shown in Figure 7.

The first tool is the information model editor and allows one to develop information model for different domain. The output of this tool is stored in the data repository and can be used as input to the second tool, named SysADE, which aims at designing both System Architectures and Simulation Architectures.

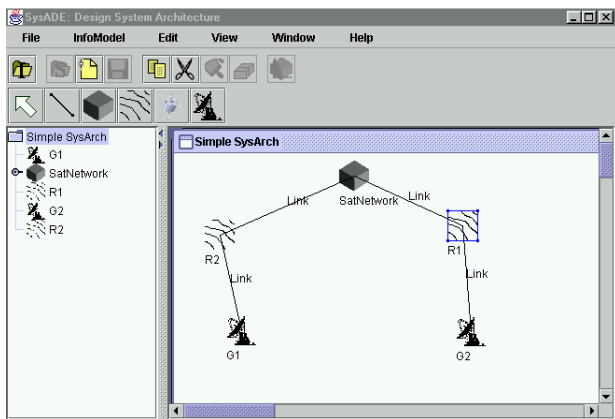


Figure 8: A System Architecture

Figure 8 shows an example of a simple System Architecture (named Simple SysArch) for the space

domain. The palette above the canvas reports the elementary elements defined in the Space Info Model (Satellite, GroundStation, RadioChannel). Simple SysArch contains also a subsystem named Satellite Network, which is composed of three different satellites and two Radio Channels (Figure 9).

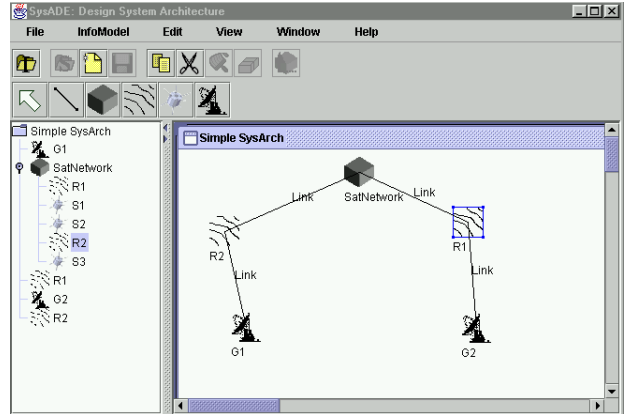


Figure 9: A Subsystem

Finally, the Data Repository supports the metamodel of Section 4 and thus it is used by the InfoModel Editor to store Information models and by SysADE both to retrieve an Information model and to store System Architectures.

The ASIA environment has been developed using JAVA and runs in a Windows environment.

8. Conclusions

The paper presents a design and enactment approach for simulation systems based on well-known notations (UML) and technology (CORBA). Within a single architecture description language, we propose a design approach for domain-specific simulation environments. The approach aims at being both “general” and specific at the same time. To meet this goal, we defined a core SADL that supplies a core set of concepts to provide system engineers with enough generality to apply the proposal in the particular domains.

Domain-specific notations exploit the customization features offered by UML to present core concepts in a way suitable to the different application domains. The design process, together with OCL, constrains the use of the different models and augment them with a sufficiently precise semantics, which permits the consistency checking of designed models. We decided to adopt OCL because of its strict relation with UML; similar, and maybe more powerful, consistency checks would be possible by annotating designed models with other formal textual languages.

To fully understand and exploit the capability of the approach, nowadays we need to improve the supporting CASE tools. The first experiments in customizing Rational Rose [19] for our particular needs were not encouraging. The lack of specific expertise and the difficulty of obtaining significant results persuaded us to develop special-purpose editors using Java and Swing. Furthermore, we need to customize and apply the approach to a wider range of simulation domains to get more feedback and improve users' confidence on the approach. Currently, the CEC ESPRIT ASIA project targets the space telecom and road traffic management domains.

References

- [1] ASIA Consortium, Use of Integrated Graphical & Textual Formal Specification Languages in Industry, Technical Annex. August 1998.
- [2] L. Baresi, Formal Customization of Graphical Notations, PhD thesis, Dipartimento di Elettronica e Informazione -- Politecnico di Milano, 1997.
- [3] G. Booch, J. Rumbaugh, and I. Jacobson, The Unified Modeling Language User Guide, The Addison-Wesley Object Technology Series, 1998.
- [4] A.G. Kleppe J.B. Warmer, The Object Constraint Language: Precise Modeling With UML, The Addison-Wesley Object Technology Series, 1999.
- [5] D. Lewicki and G. Fisher, VisiTile - A Visual Language Development Toolkit, In Proceedings of the 1996 IEEE Symposium on Visual Languages, pages 114--121. IEEE-CS Press, 1996.
- [6] D.Q.Zhang and K. Zhang, VisPro: A Visual Language Generation Toolset, In Proceedings of the 1998 IEEE Symposium on Visual Languages, pages 195--202. IEEE-CS Press, 1998.
- [7] K. Smolander, P. Marttiin, K. Lyytinen, and V.P. Tahvanainen, MetaEdit - A Flexible Graphical Environment for Methodology Modelling. Advanced Information Systems Engineering, LNCS 498, pages 168--193, 1991.
- [8] DoME, Domain Modeling Environment www.htc.honeywell.com/dome
- [9] R. Allen, A Formal Approach to Software Architecture, PhD thesis, School of Computer Science, Carnegie Mellon University, 1997.
- [10] J. Magee, N. Dulay, S. Eisenbach, and J. Kramer, Specifying Distributed Software Architectures, In Proceedings the 5th European Software Engineering Conference, vol. 989 LNCS, pp 137—153, Springer-Verlag, September 1995.
- [11] D. Luckham, Rapide: A language and toolset for causal event modeling of distributed system architectures, In Proceedings of the 2nd International Conference on Worldwide Computing and Its Applications, volume 1368 LNCS, pp 88--103. Springer-Verlag, 1998
- [12] J. Robbins, N. Medvidovic, D. Redmiles, and D. Rosenblum, Integrating architecture description languages with a standard design method, In Proceedings of the 20th International Conference on Software Engineering, pages 209—218, IEEE-CS Press, Apr. 1998.
- [13] P. Kellert, N. Tchernev, and C. Force, Object Oriented Methodology for FMS modelling and Simulation, International Journal on Computer Integrated Manufacturing, 2(6), 1997.
- [14] The CIM Framework Architecture Guide 1.0, www.sematech.org/public/division/fi/cim/cimhome.htm.
- [15] ISO (International Organisation for Standardization), Industrial Automation Systems and Integration - Product Data Representation and Exchange, 1994.
- [16] IEEE, IEEE Standard for Distributed Interactive Simulation - Application Protocols, IEEE standard, 1278.1, 1995, May 1995.
- [17] DMSO (Defense Modeling and Simulation Office), High Level Architecture - Overview and Rules, February 1997.
- [18] B. Boehm, Software Engineering Economics, Prentice-Hall, 1981.
- [19] Rational Software Corporation, Rational Rose 98: User's Manuals, 1998.