GUIDE FOR MULTI-ARCHITECTURE
LIVE-VIRTUAL-CONSTRUCTIVE
ENVIRONMENT ENGINEERING AND
EXECUTION

JUNE 2010
Guide for Multi-Architecture
Live-Virtual-Constructive
Environment Engineering and Execution

June 2010

FOR:
Joint Training Integration and Evaluation Center
1200 Research Parkway, Suite 300
Orlando, FL 32826

BY:
Johns Hopkins University - Applied Physics Laboratory
11100 Johns Hopkins Road
Laurel, MD 20723
# TABLE OF CONTENTS

**EXECUTIVE SUMMARY** .................................................................................................................. ES-1

1 **INTRODUCTION** .......................................................................................................................... 1-1

1.1 BACKGROUND ................................................................................................................................. 1-1
1.2 SCOPE .............................................................................................................................................. 1-1
1.3 DOCUMENT OVERVIEW ............................................................................................................... 1-6
1.4 DEFINITIONS .................................................................................................................................. 1-7

2 **MULTI-ARCHITECTURE ISSUES AND SOLUTIONS** ................................................................... 2-1

2.1 STEP 1: DEFINE SIMULATION ENVIRONMENT OBJECTIVES .................................................... 2-1
2.1.1 Activity 1.1: Identify User/Sponsor Needs .................................................................................. 2-1
2.1.2 Activity 1.2: Develop Objectives ............................................................................................... 2-1
2.1.3 Activity 1.3: Conduct Initial Planning ....................................................................................... 2-1
2.2 STEP 2: PERFORM CONCEPTUAL ANALYSIS ........................................................................... 2-4
2.2.1 Activity 2.1: Develop Scenario ................................................................................................. 2-4
2.2.2 Activity 2.2: Develop Conceptual Model ................................................................................... 2-5
2.2.3 Activity 2.3: Develop Simulation Environment Requirements ............................................... 2-5
2.3 STEP 3: DESIGN SIMULATION ENVIRONMENT ......................................................................... 2-7
2.3.1 Activity 3.1: Select Member Applications ................................................................................. 2-8
2.3.2 Activity 3.2: Design Simulation Environment ......................................................................... 2-10
2.3.3 Activity 3.3: Design Member Applications .............................................................................. 2-26
2.3.4 Activity 3.4: Prepare Detailed Plan ......................................................................................... 2-28
2.4 STEP 4: DEVELOP SIMULATION ENVIRONMENT ..................................................................... 2-35
2.4.1 Activity 4.1: Develop Simulation Data Exchange Model ......................................................... 2-35
2.4.2 Activity 4.2: Establish Simulation Environment Agreements ............................................... 2-40
2.4.3 Activity 4.3: Implement Member Application Designs ............................................................. 2-42
2.4.4 Activity 4.4: Implement Simulation Environment Infrastructure ........................................... 2-43
2.5 STEP 5: INTEGRATE AND TEST SIMULATION ENVIRONMENT .......................................... 2-46
2.5.1 Activity 5.1: Plan Execution ...................................................................................................... 2-46
2.5.2 Activity 5.2: Integrate Simulation Environment ...................................................................... 2-47
2.5.3 Activity 5.3: Test Simulation Environment .............................................................................. 2-48
2.6 STEP 6: EXECUTE SIMULATION ................................................................................................. 2-51
2.6.1 Activity 6.1: Execute Simulation ............................................................................................... 2-51
2.6.2 Activity 6.2: Prepare Simulation Environment Outputs ......................................................... 2-52
2.7 STEP 7: ANALYZE DATA AND EVALUATE RESULTS ............................................................ 2-53
2.7.1 Activity 7.1: Analyze Data ........................................................................................................ 2-53
2.7.2 Activity 7.2: Evaluate and Feedback Results ........................................................................... 2-53

**APPENDIX A.** REFERENCES AND BIBLIOGRAPHY .............................................................. 1

**APPENDIX B.** MAPPING OF ISSUES TO EXISTING ARCHITECTURES .................................... 1
APPENDIX C. ABBREVIATIONS AND ACRONYMS ....................................................... 1

LIST OF FIGURES
Figure 1-1. Gateway Configuration ................................................................. 1-2
Figure 1-2. Middleware Configuration ............................................................. 1-3
Figure 1-3. Distributed Simulation Engineering and Execution Process (DSEEP), Top-Level Process Flow View ................................................................. 1-4
EXECUTIVE SUMMARY

Robust, well-defined systems engineering (SE) processes are a key element of any successful development project. In the distributed simulation community, there are several such processes in wide use today, each aligned with a specific simulation architecture such as Distributed Interactive Simulation (DIS), High Level Architecture (HLA), and Test and Training Enabling Architecture (TENA). However, there are an increasing number of distributed simulation applications within the Department of Defense (DoD) that require the selection of simulations whose external interfaces are aligned with more than one simulation architecture. This is what is known as a multi-architecture simulation environment.

Many technical issues arise when multi-architecture simulation environments are being developed and executed. These issues tend to increase program costs and can increase technical risk and impact schedules if not resolved adequately. The Live-Virtual-Constructive Architecture Roadmap (LVCAR) was initiated in 2007 to define the differences among the major simulation architectures from technical, business, and standards perspectives and to develop a time-phased set of actions to improve interoperability within multi-architecture simulation environments in the future.

One of the barriers to interoperability identified in the LVCAR Phase I Report was driven by a community-wide recognition that when user communities, aligned with the different simulation architectures, are brought together to develop a multi-architecture distributed simulation environment, the differences in the development processes native to each user community adversely affected the ability to collaborate effectively. To address this problem, a recommendation was made to establish a common cross-community SE process for the development and execution of multi-architecture simulation environments. However, rather than develop an entirely new process, it was recognized that an existing process standard should be leveraged and extended to address multi-architecture concerns. The process framework that was chosen is an emerging Institute of Electrical and Electronics Engineers (IEEE) standard called the Distributed Simulation Engineering and Execution Process (DSEEP). The DSEEP tailors widely recognized and accepted SE practices to the modeling and simulation domain and, more specifically, to the development and execution of distributed simulation environments. The strategy implemented in this case was to augment the major DSEEP steps and activities with the additional tasks that are needed to address the issues that are unique to (or at least exacerbated by) multi-architecture development. These tasks collectively define a “how to” guide for developing and executing multi-architecture simulation environments, based on recognized best practices.

This document defines a total of 40 multi-architecture related issues, based on an extensive literature search. Each of these issues is aligned with the activity in the DSEEP for which the issue first becomes relevant. Each issue comes with both a description and a
recommended action(s) to best address the issue. A set of inputs, outcomes, and recommended
tasks is also provided for each DSEEP activity to address the resolution of the multi-architecture
issues. This information is provided as an overlay to corresponding information already provided
in the DSEEP document for single-architecture development.

An appendix to this document identifies a tailoring of the guidance provided in the main
document to individual architecture communities. For each of three major simulation
architectures, a mapping is provided to indicate the relevance of each Issue–Recommended
Action pair to developers and users of that simulation architecture. Together with the guidance
provided in the main text, it is believed that this document will provide the guidance needed to
improve cross-community collaboration and thus reduce costs and technical risk in future multi-
architecture developments.
INTRODUCTION

1.1 BACKGROUND

Modeling and simulation (M&S) has long been recognized as a critical technology for managing the complexity associated with modern systems. In the defense industry, M&S is a key enabler of many core systems engineering functions. For instance, early in the systems acquisition process, relatively coarse, aggregate-level constructive models are generally used to identify capability gaps, define systems requirements, and examine/compare potential system solutions. As preferred concepts are identified, higher-fidelity models are used to evaluate alternative system designs and to support initial system development activities. As design and development continues, very high-fidelity models are used to support component-level design and development, as well as developmental test. Finally, combinations of virtual and constructive M&S assets are frequently used to support operational test and training requirements. Note that other industries (e.g., entertainment, medical, transportation) also make heavy use of M&S, although in somewhat different ways.

The advent of modern networking technology and the development of supporting protocols and architectures have led to widespread use of distributed simulation. The strategy behind distributed simulation is to use networks and support simulation services to link existing M&S assets into a single unified simulation environment. This approach provides several advantages as compared to development and maintenance of large monolithic stand-alone simulation systems. First, it allows each individual simulation application to be co-located with its resident subject matter expertise rather than having to develop and maintain a large stand-alone system in one location. In addition, it facilitates efficient use of past M&S investments, as new, very powerful simulation environments can be quickly configured from existing M&S assets. Finally, it provides flexible mechanisms to integrate hardware and/or live assets into a unified environment for test or training, and it is much more scalable than stand-alone systems.

There are also some disadvantages of distributed simulation. Many of the issues related to distributed simulation are related to interoperability concerns. Interoperability refers to the ability of disparate simulation systems and supporting utilities (e.g., viewers, loggers) to interact at runtime in a coherent fashion. There are many technical issues that affect interoperability, such as consistency of time advancement mechanisms, compatibility of supported services, data format compatibility, and even semantic mismatches for runtime data elements. The capabilities provided by today’s distributed simulation architectures are designed to address such issues and allow coordinated runtime interaction among participating simulations. Examples of such architectures include Distributed Interactive Simulation (DIS), the Test and Training Enabling Architecture (TENA), and the High Level Architecture (HLA).

In some situations, sponsor requirements may necessitate the selection of simulations whose external interfaces are aligned with more than one simulation architecture. This is what is
known as a *multi-architecture simulation environment*. There are many examples of such environments within the Department of Defense (DoD) (see references for examples). When more than one simulation architecture must be used in the same environment, interoperability problems are compounded by the architectural differences. For instance, middleware incompatibilities, dissimilar metamodels for data exchange, and differences in the nature of the services that are provided by the architectures must all be reconciled for such environments to operate properly. Developers have devised many different workarounds for these types of interoperability problems over the years. One possible solution is to choose a single architecture for the simulation environment and require all participants to modify the native interfaces of their simulations to conform to it. While this solution is relatively straightforward and easy to test, it is usually impractical (particularly in large applications) because of the high cost and schedule penalties incurred. Another approach is the use of gateways, which are independent software applications that translate between the protocols used by one simulation architecture to that of a different simulation architecture (see Figure 1-1). While effective, gateways represent another potential source of error (or failure) within the simulation environment, can introduce undesirable latencies into the system, and add to the complexity of simulation environment testing. In addition, many gateways are legacy point solutions that provide support only for a very limited number of services and only for very specific versions of the supported simulation architectures. Thus, it may be difficult to find a suitable gateway that fully supports the needs of a given application. For the relatively small number of general-purpose gateways that are configurable, the effort required to perform the configuration function can be significant and can result in excessive consumption of project resources.

![Gateway Configuration](image-url)

**Figure 1-1. Gateway Configuration**
The use of *middleware* is a similar approach but provides the translation services in software directly coupled to the simulation instead of an independent application\(^1\) (see Figure 1-2). While middleware approaches are also effective, they introduce many of the same technical issues that are associated with gateways (e.g., source of error, possible latency penalties). In general, all of these “solutions” have limitations and cost implications that increase technical, cost, and schedule risk for multi-architecture developments.

\[\text{Figure 1-2. Middleware Configuration}\]

Because of perceived increases in the number of multi-architecture simulation events anticipated in the future, along with the associated increase in costs, the DoD sponsored an initiative to examine the differences among the major simulation architectures from technical, business, and standards perspectives and to develop a time-phased set of actions to improve interoperability within multi-architecture simulation environments in the future. This initiative was called the Live-Virtual-Constructive Architecture Roadmap (LVCAR). The first phase of this effort began in the spring of 2007 and continued for approximately 16 months. The result of this activity was a final report and supporting documentation that collectively totaled over 1000 pages. The second phase of this initiative focused on the implementation of the recommended actions from this report.

A key conclusion of the LVCAR effort was that migrating to a single distributed simulation architecture was impractical, and thus multi-architecture simulation environments would remain the state of the practice for the foreseeable future. One of the key actions recommended in the LVCAR Phase I Report was the establishment of a common systems engineering process for the development and execution of multi-architecture simulation environments. The widely reported issue in this case was that when user communities of different architectures were brought together to develop a single multi-architecture distributed simulation environment, the differences in the development processes native to each user community were creating a persistent barrier to effective collaboration. That is, since these

\(^1\) Note that this use of the term “middleware” is different in some user communities, who may use this term to refer to the infrastructure elements that provide distributed simulation services (e.g., the HLA Runtime Infrastructure [RTI]).
communities had to work together toward common goals, differences in the practices and procedures these communities typically use to build new simulation environments were leading to misunderstandings, misinterpretations, and general confusion among team members. This was impacting risk from many different perspectives.

To develop the common systems engineering process, it was felt that leveraging and modifying/extending an existing systems engineering process standard was preferable to building an entirely new process description from scratch. Early in the project, the systems engineering process team considered several generalized and widely recognized systems and software standards (e.g., EIA-632, ISO/IEC 15288). However, the team decided that direct reuse of any process standard outside of the M&S domain would require a significant degree of tailoring, consuming resources that could be better applied in other ways. For that reason, the team selected an emerging Institute of Electrical and Electronics Engineers (IEEE) standard (IEEE 1730) as the foundation for the desired process. The name of this standard is the Distributed Simulation Engineering and Execution Process (DSEEP).

The DSEEP represents a tailoring of best practices in the systems and software engineering communities to the M&S domain. The DSEEP is simulation architecture-neutral, but it does contain annexes that map this architecture-neutral view to DIS, HLA, and TENA terminology. A top-level view of the DSEEP is provided in Figure 1-3.

A short description of each of these seven major steps follows:

**Step 1: Define Simulation Environment Objectives.** The user, the sponsor, and the development/integration team define and agree on a set of objectives and document what must be accomplished to achieve those objectives.

**Step 2: Perform Conceptual Analysis.** The development team performs scenario development and conceptual modeling and develops the simulation environment requirements based upon the characteristics of the problem space.
Step 3: Design Simulation Environment. Existing member applications that are suitable for reuse are identified, design activities for member application modifications and/or new member applications are performed, required functionalities are allocated to the member applications, and a plan is developed for development and implementation of the simulation environment.

Step 4: Develop Simulation Environment. The simulation data exchange model is developed, simulation environment agreements are established, and new member applications and/or modifications to existing member applications are implemented.

Step 5: Integrate and Test Simulation Environment. All necessary integration activities are performed, and testing is conducted to verify that interoperability requirements are being met.

Step 6: Execute Simulation. The simulation environment is executed and the output data from the execution is pre-processed.

Step 7: Analyze Data and Evaluate Results. The output data from the execution is analyzed and evaluated, and results are reported back to the user/sponsor.

In the DSEEP document, each of these seven steps is further decomposed into a set of interrelated lower-level activities. Each activity is characterized according to a set of required activity inputs, one or more output products, and a list of recommended finer-grain tasks. Although these activity descriptions are identified in a logical sequence, the DSEEP emphasizes that iteration and concurrency are to be expected, not only across activities within a step but across steps as well.

Although the DSEEP provides the guidance required to build and execute a distributed simulation environment, the implicit assumption within the DSEEP is that only a single simulation architecture is being used. The only acknowledgement that this assumption may be false is provided in the following paragraph from DSEEP Activity 3.2 (Design Simulation Environment):

In some large simulation environments, it is sometimes necessary to mix several simulation architectures. This poses special challenges to the simulation environment design, as sophisticated mechanisms are sometimes needed to reconcile disparities in the architecture interfaces. For instance, gateways or bridges to adjudicate between different on-the-wire protocols are generally a required element in the overall design, as well as mechanisms to address differences in simulation data exchange models. Such mechanisms are normally formalized as part of the member application agreements, which are discussed in Step 4.

Clearly, additional guidance is necessary to support the development of multi-architecture simulation environments. However, the major steps and activities defined in the DSEEP are generally applicable to either single- or multi-architecture development. Thus, the
DSEEP provides a viable framework for the development of the desired process, but it must be augmented with additional tasks as necessary to address the issues that are unique to (or at least exacerbated by) multi-architecture development. Such augmenting documentation is often referred to as an overlay. The tasks in this overlay collectively define a “how to” guide for developing and executing multi-architecture simulation environments, based on perceived best practices for issue resolution.

The remainder of this first section describes the organization of and the associated constraints upon the overlay specification. This is critical to understanding the technical description of the overlay as described in Section 0.

1.2 SCOPE

This document is intended for users and developers of multi-architecture simulation environments. It describes a comprehensive set of technical issues that are either unique to multi-architecture development or are more difficult to resolve in multi-architecture simulation environments. The solution(s) provided for each issue are focused on multi-architecture developments but may have applicability to single-architecture development as well.

This document is intended as a companion guide to the DSEEP. The simulation environment user/developer should assume that the guidance provided by the DSEEP is applicable to both single- and multi-architecture developments but that this document provides the additional guidance needed to address the special concerns of this class of the multi-architecture user/developer.

1.3 DOCUMENT OVERVIEW

This document is organized as an overlay to the DSEEP. Each subsection begins with a short description of the DSEEP activity. Next, the multi-architecture technical issue(s) that are relevant to that DSEEP activity are listed and described. After the statement of each issue, the recommended action(s) to address that issue are presented. Finally, the recommended action(s) for the issue are translated into an appropriate set of inputs, outcomes, and recommended tasks to augment corresponding DSEEP inputs/outcomes/tasks for that activity. This structure is repeated for all of the activities defined in the DSEEP document.

Note that some DSEEP activities do not have any technical issues associated with them. This indicates that the existing DSEEP activity description applies equally well to either single- or multi-architecture environments and that there are no additional multi-architecture-specific

---

2 Some issues impact multiple DSEEP activities. Rather than repeating the issue multiple times, it is elaborated at the first affected activity.
inputs, outcomes, or recommended tasks for that activity. This situation mainly occurs either early or late in the overall process.

1.4 DEFINITIONS

**Conceptual Model:** An abstraction of what is intended to be represented within a simulation environment, which serves as a frame of reference for communicating simulation-neutral views of important entities and their key actions and interactions. The conceptual model describes what the simulation environment will represent, the assumptions limiting those representations, and other capabilities needed to satisfy the user’s requirements. Conceptual models are bridges between the real world, requirements, and simulation design.

**Member Application:** An application that is serving some defined role within a simulation environment. This can include live, virtual, or constructive (LVC) simulation assets or can be supporting utility programs such as data loggers or visualization tools.

**Objective:** The desired goals and results of the activity to be conducted in the distributed simulation environment expressed in terms relevant to the organization(s) involved.

**Requirement:** A statement identifying an unambiguous and testable characteristic, constraint, process, or product of an intended simulation environment.

**Simulation Environment:** A named set of member applications along with a common simulation data exchange model and set of agreements that are used as a whole to achieve some specific objective.

**Live Simulation:** A simulation involving real people operating real systems.

**Virtual Simulation:** A simulation involving real people operating simulated systems. Virtual simulations inject human-in-the-loop (HITL) in a central role by exercising motor control skills (e.g., flying an airplane), decision skills (e.g., committing fire control resources to action), or communication skills (e.g., as members of a command, control, communications, computers, and intelligence [C4I] team).

**Constructive Simulation:** Models and simulations that involve simulated people operating simulated systems. Real people stimulate (make inputs) to such simulations but are not involved in determining the outcomes.
This page intentionally left blank.
2.1 STEP 1: DEFINE SIMULATION ENVIRONMENT OBJECTIVES

The purpose of Step 1 of the DSEEP is to define and document a set of needs that are to be addressed through the development and execution of a simulation environment and to transform these needs into a more detailed list of specific objectives for that environment.

2.1.1 Activity 1.1: Identify User/Sponsor Needs

The primary purpose of this activity is to develop a clear understanding of the problem to be addressed by the simulation environment. The needs statement may vary widely in terms of scope and degree of formalization. It should include, at a minimum, high-level descriptions of critical systems of interest, initial estimates of required fidelity and required behaviors for simulated entities, key events and environmental conditions that must be represented in the scenario, and output data requirements. In addition, the needs statement should indicate the resources that will be available to support the simulation environment (e.g., funding, personnel, tools, facilities) and any known constraints that may affect how the simulation environment is developed (e.g., required member applications, due dates, site requirements, and security requirements).

2.1.1.1 Issues

No multi-architecture issues have been identified for this activity.

2.1.2 Activity 1.2: Develop Objectives

The purpose of this activity is to refine the needs statement into a more detailed set of specific objectives for the simulation environment. The objectives statement is intended as a foundation for generating explicit simulation requirements, i.e., translating high-level user/sponsor expectations into more concrete, measurable goals. This activity requires close collaboration between the user/sponsor of the simulation environment and the development team to verify that the original needs statement is properly analyzed and interpreted and that the resulting objectives are consistent with the stated needs. Early assessments of feasibility and risk should also be performed as part of this activity.

2.1.2.1 Issues

No multi-architecture issues have been identified for this activity.

2.1.3 Activity 1.3: Conduct Initial Planning

The purpose of this activity is to establish a preliminary simulation environment development and execution plan. The intent is to translate the objectives statement, along with
the associated risk and feasibility assessments, into an initial plan with sufficient detail to effectively guide early design activities. The plan may effectively include multiple plans and should cover such considerations as verification and validation (V&V), configuration management, and security. The plan should also address supporting tools for early DSEEP activities, based on factors such as availability, cost, applicability to the given application, ability to exchange data with other tools, and the personal preferences of the development team.

2.1.3.1 Issues

2.1.3.1.1 Issue: Multi-architecture Initial Planning

DESCRIPTION

During initial planning, work breakdown structures are typically developed that define the required project tasks and the overall project schedule and that estimate funding expenditure rates. However, the identity of several participating member applications may be unknown this early in the process, and thus the requirement for a multi-architecture simulation environment design may be unknown. In the absence of better information, project managers frequently just assume single-architecture operation, which underestimates the time and resources necessary to establish the simulation environment. This increases project risk from several perspectives.

RECOMMENDED ACTION(S)

The scope of the distributed simulation environment effort should be established. The questions of what needs to be done and who needs to participate should be identified early in the development process. Although such considerations can be added during later development phases, omissions made during planning may increase the technical and schedule risk of the simulation development. In general, planners should use their best judgment as to what will be needed, based on the information available to them. If the initial plan assumes that the simulation environment development will be single-architecture, the sponsor should be made aware very early of the potential for significant rework of the plan and the potential need for additional resources if the assumption is later found to be false. If the initial plan assumes that the simulation environment development will be multi-architecture, the relatively high level of resources required should be communicated very early to the sponsor. In that way, certain objectives can be relaxed as appropriate if resource demands are considered overly excessive. Another system development approach may be to plan for two simulation environments, one implemented as a single-architecture simulation environment and a second implemented as a multi-architecture simulation environment. Multi-architecture systems are complex developments and have technical, financial, schedule, and programmatic issues that should preclude their use unless absolutely necessary to satisfy user/sponsor requirements. Sufficiently analyzing the benefits, feasibility, limitations, constraints, trade-offs, and risks of multi-
architecture engineering issues improves successful planning of a multi-architecture system. If the initial planning documents fail to reflect the additional developmental considerations required by a multi-architecture system, then the result will be major omissions in terms of what will eventually need to be integrated into a multi-architecture environment, both with respect to actual applications (e.g., gateways) and overarching requirements in the areas of performance, execution management, networking, and required complementary development activities (e.g. security and verification, validation, and accreditation [VV&A]).

2.1.3.1.2 Issue: Required LVC Expertise

DESCRIPTION

In the event that the user/sponsor requires the use of certain member applications, and those member applications have existing interfaces that cut across more than one architecture, lack of personnel experienced in the development of multi-architecture LVC environments on the initial development team may result in unachievable cost and/or schedule objectives, which will adversely affect the planning process.

RECOMMENDED ACTION(S)

Resolving the issue of having the required LVC expertise to successfully execute an effort where a multi-architecture environment is required typically takes one of two paths: adding the appropriate experienced personnel to the team permanently or adding them temporarily. Both approaches are valid, and the specific situation should dictate the action taken.

Temporarily adding multi-architecture LVC expertise is typically done by using a consultant or team of consultants. While the term “consultant” can have a negative connotation, here it refers to a person temporarily added to a team in order to provide the necessary guidance and oversight to allow successful execution of the required activity. This added expertise can come from inside or from outside the current company or program. Certainly, there are programmatic trade-offs associated with both approaches. The goal of outside “consultants” should be to render themselves obsolete while ensuring that the management goals for multi-architecture execution are met. For example, the TENA community provides a User Support team for simulation events using TENA. The goal of the TENA User Support team is to provide assistance as necessary to integrate TENA into the simulation environment; such assistance runs the gamut from software development/coding support to network configuration.

The addition of permanent team members experienced in multi-architecture LVC environments can have substantial long-term impact on the ability of a team to execute multi-architecture LVC events. When managed correctly, the new permanent team member(s) can have a significant positive impact on the long-term development and execution efforts of the team.
Both of the above approaches are valid even when multi-architecture expertise exists on a team but specific architecture expertise is missing. For example, experience exists in HLA to/from DIS multi-architecture environments, but the requirement is for HLA to/from TENA and no TENA expertise exists on the team. In this case the addition of expertise is constrained to the unfamiliar architecture.

2.1.3.2 Consolidation of “Conduct Initial Planning” Activities to Support Multi-architecture Events

**Multi-architecture-specific activity inputs**
- Personnel with experience in multi-architecture environment

**Multi-architecture-specific tasks**
- Plan for single- and multi-architecture environments alternatives.
- Select approach for adding personnel with multi-architecture experience—either through temporary or permanent staff augmentation.

**Multi-architecture-specific activity outcomes**
- Within “Simulation environment development and execution plan” (per the DSEEP)
  - Staffing plan to account for multi-architecture concerns
  - Contingency plans for single- or multi-architecture environments

2.2 STEP 2: PERFORM CONCEPTUAL ANALYSIS

The purpose of this step of the DSEEP is to develop an appropriate representation of the real-world domain that applies to the defined problem space and to develop the appropriate scenario. It is also in this step that the objectives for the simulation environment are transformed into a set of highly specific requirements that will be used during design, development, testing, execution, and evaluation.

2.2.1 Activity 2.1: Develop Scenario

The purpose of this activity is to develop a functional specification for the scenario. Depending on the needs of the simulation environment, the scenario may actually include multiple scenarios, each consisting of one or more temporally ordered sets of events and behaviors (i.e., vignettes). A scenario includes the types and numbers of major entities that must be represented within the simulation environment; a functional description of the capabilities, behavior, and relationships between these major entities over time; and a specification of relevant environmental conditions that impact or are impacted by entities in the simulation environment.
environment. Initial conditions (e.g., geographical positions for physical objects), termination conditions, and specific geographic regions should also be provided.

### 2.2.1.1 Issues

No multi-architecture issues have been identified for this activity.

### 2.2.2 Activity 2.2: Develop Conceptual Model

During this activity, the development team produces a conceptual representation of the intended problem space based on their interpretation of user needs and sponsor objectives. The product resulting from this activity is known as a conceptual model. The conceptual model provides an implementation-independent representation that serves as a vehicle for transforming objectives into functional and behavioral descriptions for system and software designers. The model also provides a crucial traceability link between the stated objectives and the eventual design implementation.

#### 2.2.2.1 Issues

No multi-architecture issues have been identified for this activity.

### 2.2.3 Activity 2.3: Develop Simulation Environment Requirements

As the conceptual model is developed, it will lead to the definition of a set of detailed requirements for the simulation environment. These requirements should be directly testable and should provide the implementation-level guidance needed to design and develop the simulation environment. The requirements should consider the specific execution management needs of all users, such as execution control and monitoring mechanisms, and data logging.

#### 2.2.3.1 Issues

#### 2.2.3.1.1 Issue: Requirements for Multi-architecture Development

**DESCRIPTION**

The initial LVC environment requirements can be derived from several sources, including the customer Use Cases, Joint Capability Areas (JCAs), Mission Threads, Universal Joint Task List (UJTL), and other operationally representative sources. During this requirement definition phase, the LVC environment design has typically not been completely determined and therefore potential multi-architecture design, development, integration, test, and execution requirements may be unknown. The selection of some specific simulations may, however, be directed by the sponsor and would require a multi-architecture environment as a result.
RECOMMENDED ACTION(S)

Three potential situations exist as a result of this issue. The first case is if this is the initial iteration through the development process and there is no simulation selection directed by the sponsor. In this situation, no multi-architecture requirements are noted; this could change, however, on subsequent iterations. The second case is if this is the first iteration and simulation selection is directed by the sponsor; this situation could result in a multi-architecture requirement. The third case is if this is a subsequent iteration though the process and a multi-architecture requirement has been determined.

The recommended action is the same for both the second and third cases. The data and interface requirements for the multi-architecture applications should be noted at this time. In order to create a testable set of requirements across architectures, the team should document the individual application and architecture requirements as necessary for the given simulation environment. The goal at this phase is to start the process of exposing the differences between architectures and to begin to understand the key differences that should be accounted for in order to successfully operate across the architectures and test the requirements.

2.2.3.1.2 Issue: Member Application Requirement Incompatibility

DESCRIPTION

By virtue of their fundamental design intent and implementation assumptions, different distributed simulation architectures are generally better suited for satisfying certain application requirements than they are for others. Member applications developed for different architectures often conform to and exhibit the design intent and assumptions of those architectures. However, incompatibilities in requirements may be introduced into the simulation environments as a result of inherent architectural differences between member applications from different architectures. These potential requirement incompatibilities should be considered during member application selection. The most important aspect of this issue is to note that there is a strong potential for requirement incompatibility as a result of using a multi-architecture environment.

RECOMMENDED ACTION(S)

The goal is to understand the differences and to start addressing the technical incompatibilities at this early stage of the process.

Understanding the technical incompatibilities introduced by the incompatibilities in requirements can manifest itself in many ways. For example, by virtue of DIS’s exploitation of specific network services and its protocol-embedded simulation data exchange model (SDEM), member applications developed for DIS are typically well suited for requirements related to virtual entity-level real-time training applications. However, a requirement for repeatability is potentially problematic for a DIS member application because of the architecture’s
unconstrained time, best effort (User Datagram Protocol [UDP] Packets over Internet Protocol [IP] [UDP/IP]) networking, and typical model sensitivity to slight differences in Protocol Data Unit (PDU) arrival time. For another example, TENA focuses on disparate live and virtual range member applications. Thus, member applications designed for TENA typically have difficulty supporting a non-real-time unit-level constructive simulation. Therefore, when member applications developed for different architectures are linked into a single multi-architecture simulation environment, some of the requirements for the multi-architecture simulation environment may be incompatible with the requirements that any particular member application can readily support.

The technical incompatibilities introduced by a multi-architecture environment are not always reconcilable. When this is the case, seeking a relaxation of the requirement (i.e., mandated use of given member applications) is advisable. For example, a trade-off may need to be made between a relaxation of the requirements and true repeatability of the simulation environment based on the known incompatibilities. While this is not always possible, exposing the technical risks at this point will at least allow risk mitigation to begin as early as possible.

2.2.3.2 Consolidation of “Develop Environment Requirements” Activities to Support Multi-architecture Events

**MULTI-ARCHITECTURE-SPECIFIC ACTIVITY INPUTS**

- None beyond those called for in the DSEEP

**MULTI-ARCHITECTURE-SPECIFIC TASKS**

- Define data and interface requirements for multi-architecture applications.
- Identify technical incompatibilities and risks specific to multi-architecture applications.

**MULTI-ARCHITECTURE-SPECIFIC ACTIVITY OUTCOMES**

- None beyond those called for in the DSEEP

2.3 STEP 3: DESIGN SIMULATION ENVIRONMENT

The purpose of this step of the DSEEP is to produce the design of the simulation environment. This involves identifying applications that will assume some defined role in the simulation environment (member applications) that are suitable for reuse, creating new member applications if required, allocating the required functionality to the member applications, and developing a detailed simulation environment development and execution plan.
2.3.1 Activity 3.1: Select Member Applications

The purpose of this activity is to determine the suitability of individual simulation systems to become member applications of the simulation environment. This is normally driven by the perceived ability of potential member applications to represent entities and events according to the conceptual model. Managerial constraints (e.g., availability, security, facilities) and technical constraints (e.g., VV&A status, portability) may both influence the selection of member applications.

2.3.1.1 Issues

2.3.1.1.1 Issue: Member Selection Criteria for Multi-architecture Applications

DESCRIPTION

The selection of member applications for multi-architecture environments requires additional criteria beyond those used for member application selection decisions in single-architecture environments. Some potential member applications of a multi-architecture environment may support only one of the architectures being employed while other potential member applications support all the architectures being employed. The selection decision becomes more complex for the system designers because the architecture support capabilities of a potential member application will need to be considered in addition to its simulation representational capabilities. A trade-off may become necessary between a highly capable member application that supports a single architecture and another less capable member application that supports multiple architectures. Such trade-offs are an important part of the selection process, and ignoring such considerations may result in schedule slippages and unanticipated technical problems.

RECOMMENDED ACTION(S)

The simulation architecture(s) that individual member applications support is perhaps the most obvious additional criterion to consider in selecting member applications for a multi-architecture simulation environment. All else being equal, maximizing the number of member applications using the same architecture reduces integration effort and overall technical risk [e.g., Blacklock and Zalcman, 1997]. The benefit of integrating a member application into a multi-architecture environment should be evaluated with respect to the effort required for the integration.
2.3.1.1.2 Issue: Non-conforming Interfaces

DESCRIPTION

It is possible that some member applications may have external interfaces that do not conform to any of the standard simulation architectures. Simulation applications that interface through alternative simulation architectures (e.g., OpenMSA, a parallel and distributed event processing simulation software framework [Lammers et al., 2008; Lammers et al., 2009]) or with other applications through web services may have high value to the goals of the simulation environment, but the solution as to how to integrate the application may require extensive engineering. Alternatively, a command and control (C2) system could be an example of such a member application. C2 systems typically exchange information through different mechanisms from those used by most simulation architectures. Linking C2 systems into a simulation environment requires that these different exchange mechanisms and underlying data models be reconciled, which can be very resource intensive and subject to runtime error.

RECOMMENDED ACTION(S)

A business case needs to justify the integration of an application with a non-conforming interface. The perceived value of that particular application needs to be evaluated against the time/effort required to perform necessary integration and test activities. If the integration of the application is justified, then the next decision is to select the architecture the potential member application will support. The technical characteristics of the member application’s interface should be compared with the different architectures in use within the simulation environment to determine which simulation architecture should be used as the basis for that member application’s interface.

2.3.1.2 Consolidation of “Select Member Applications” Activities to Support Multi-architecture Events

MULTI-ARCHITECTURE-SPECIFIC ACTIVITY INPUTS

- Potential member applications capable of supporting various architectures

MULTI-ARCHITECTURE-SPECIFIC TASKS

- Perform trade-off analysis so as to meet simulation environment requirements while maximizing the number of member applications using the same architecture.
- Select an architecture for selected member applications that currently have non-conforming interfaces.

MULTI-ARCHITECTURE-SPECIFIC ACTIVITY OUTCOMES

- List of architectures supported by the selected member applications
2.3.2 Activity 3.2: Design Simulation Environment

Once all member applications have been identified, the next major activity is to prepare the simulation environment design and allocate the responsibility to represent the entities and actions in the conceptual model to the member applications. This activity will allow an assessment of whether the set of selected member applications provides the full set of required functionality. A by-product of the allocation of functionality to the member applications will be additional design information that can embellish the conceptual model.

2.3.2.1 Issues

2.3.2.1.1 Issue: Object State Update Contents

**DESCRIPTION**

Some distributed simulation architectures (e.g., DIS) require updates of a simulated object’s state to include a complete set of the object’s state attributes. Other architectures (e.g., HLA) do not require object state updates to include attributes that have not changed. A multi-architecture simulation environment combining these two paradigms must resolve the difference.

**RECOMMENDED ACTION(S)**

The designer should ensure that the mechanisms used to link architectures with different state update requirements automatically produce updates that are compliant with the expectations of the receiving member applications. For example, DIS–HLA gateways typically perform these functions by maintaining a complete set of attributes for each simulated object [Cox et al., 1996; Wood et al., 1997; Wood and Petty, 1999]. When an HLA object attribute update for some object is received by the gateway, the gateway’s internal attributes for the object are updated and then a complete DIS Entity State PDU is produced from the gateway’s internal attributes for the object and sent. When a DIS Entity State PDU for some object is received by the gateway, the object attributes in the incoming PDU are compared to the gateway’s internal attributes for the object; those that are different are updated in the gateway’s internal set from the PDU and also sent via an HLA object attribute update service invocation. The gateway’s internal attributes for an object are initialized the first time the gateway receives an update for those attributes from either side.

2.3.2.1.2 Issue: Object Ownership Management

**DESCRIPTION**

Some distributed simulation architectures allow the transfer of responsibility for updating object attribute values from one member application to another during execution, effectively allowing the transfer of responsibility for simulating that object (or aspects of it). Some other
architectures do not provide comparable capabilities. A multi-architecture simulation environment combining these two paradigms must resolve the difference.

**RECOMMENDED ACTION(S)**

At least two resolutions for this issue are possible. First, when choosing member applications for inclusion in a multi-architecture simulation environment, select for inclusion all of the member applications that transfer object ownership between one another as a set. By doing so, those member applications can continue to perform object ownership transfer between themselves as before. Second, if it is not possible to include the entire set, then modify member applications that are not able to transfer object ownership to do so by adding ownership transfer capabilities to them, and/or modify member applications able to transfer object ownership to not do so; the latter may require enhancing member applications to perform functions internally that were previously handled externally through ownership transfer.

### 2.3.2.1.3 Issue: Time-managed Multi-architecture Applications

**DESCRIPTION**

An important multi-architecture consideration is time management. Some simulation architectures are intended to support only real-time operation, while others possess specialized services for non-real-time operation. The construction of a multi-architecture simulation environment may require the simulation environment development team to integrate time representations and time management schemes not previously used in a simulation environment.

Obviously, mixing architectures imposes some significant constraints on how time is managed in the environment. However, issues related to how messages are time-stamped, how time is calibrated across the various applications, and how to recover if the processing load causes some member applications to fall behind wall-clock time are all design issues with different solutions across the different architecture communities. Failure to reconcile these differences can lead to significant errors in how time is advanced across the multi-architecture environment.

**RECOMMENDED ACTION(S)**

The simulation environment development team may need to review and extensively revise existing software code and operating procedures or create new software and procedures to accommodate various time representations and time management schemes. Experiments should be performed to determine the best way to overcome the inherent difference between a device capable of manipulating time and a device that is not capable of doing so. Coordination schemes might require real-world events to be correlated to simulated events that occur.
Two use cases should be considered. The first is the integration of time-managed member applications into a non-time-managed simulation environment. The second is the integration of non-time-managed member applications into a time-managed simulation environment. The recommended action(s) for the first use case are (1) to adapt the interface of the time-managed member applications to respond with a time advance grant for each time advance request, and (2) to manage the execution speed of the member applications to keep pace with the other applications in the simulation environment. The recommended action for the second use case is to manage the execution speed of the member applications to keep pace with the other applications in the simulation environment.

2.3.2.1.4 Issue: Inconsistent Development and Execution Processes

DESCRIPTION

Well-defined and understood processes and procedures for building simulation environments exist within a given architecture community. What does not exist is consistency of those processes and procedures across architecture communities in a multi-architecture simulation environment. In addition, communication between these distinct architecture communities can be a significant problem because of differences in terminology and can lead to misunderstandings that can require significant rework at some point later in the process.

RECOMMENDED ACTION(S)

The key to allowing teams from different architecture communities to work together successfully in a multi-architecture environment is to provide a mechanism for understanding unique processes and procedures and to correlate the unique terminology.

Experts from each of the architecture communities involved should coordinate and correlate their individual processes and procedures so that expectations can be managed and schedules can be coordinated. For instance, one community may take extra time working on the details of the SDEM (e.g., HLA and TENA) while others may be more concerned about the details of how the data flows on the network (e.g., DIS). This area is ultimately less of a technical issue and more of a management issue.

A documented common set of terminology that correlates terms across the architectures used in the multi-architecture environment will allow the individual development teams to clearly understand the other’s intent when working together to design and implement the requisite architecture. The DSEEP provides annexes for DIS, HLA, and TENA that can be used as a starting point for relating terminology across architectures. This correlation of terminology is critical as it allows teams with their own “language” the ability to communicate effectively throughout the development process.
Another way to help bridge the knowledge gap across teams’ expertise is to have focused training sessions where the participating teams can learn about the other architectures and build a rapport. While ideally informal in format, each architecture team can have the opportunity to present the basics of what their architecture is and how it works. For example, the TENA community has created a TENA Overview Course and a more in-depth TENA Technical Introduction Course that can be presented and discussed. While these are likely sufficient to start a fundamental understanding of the TENA architecture, a more in-depth programming class, the TENA Hands-On Training, is also available when necessary.

### 2.3.2.1.5 Issue: Interest Management Capability Differences

**DESCRIPTION**

In a distributed simulation architecture, interest management refers to data filtering capabilities that may in some way limit network data transmissions so that member applications receive only the data they are interested in. Different distributed simulation architectures have different interest management capabilities, features, and power. In a multi-architecture simulation environment, the simulation environment designer should reconcile the different interest management capabilities of the linked architectures, and the architecture-specific interest management mechanisms used within single architectures, with each other. For example, DIS uses a broadcast scheme (all member applications receive all data). To reduce the load of incoming data for DIS member applications, a variety of DIS filters, varying in application specificity, have been developed. In some DIS simulation environments, PDUs are filtered based on the site/host identification of each PDU. In DIS member applications that include a gateway for interface with HLA or TENA member applications, the PDUs generated by the gateway may all have the gateway’s site/host identification, defeating the site/host filtering scheme. For another example, HLA includes significant interest management capabilities through its Declaration Management and Data Distribution Management services. When HLA and non-HLA member applications are connected using a gateway or middleware, it may be possible to replicate some or all of the filtering achievable within an HLA federation through some other mechanism within the gateway or middleware in order to provide the HLA member applications with an input data stream consistent with their expectations. For example, a DIS-HLA gateway was enhanced to mimic some HLA Data Distribution Management capabilities using experimental DIS PDUs [Williams and Smith, 2007]. Furthermore, attention should be given to the effects that such filtering would have in an architecture where member applications may assume that they are receiving all network messages (e.g., DIS).

**RECOMMENDED ACTION(S)**

Interest management is an area where different architectures/protocols vary widely [e.g., Specht, 1997]. Some of the interest management capabilities between architectures are
equivalent, or nearly so, whereas others may be quite different. For example, TENA has interest management capabilities analogous to HLA’s Declaration Management services (filtering based on object classes) [Cutts et al., 2006] but does not have capabilities analogous to HLA’s Data Distribution Management services (filtering based on overlapping attribute value ranges) [Morse and Steinman, 1997; MÄK Technologies, 2009]. Approaches to resolving interest management capability differences depend heavily on the specific set of architectures present in the multi-architecture simulation environment and the interest management features used and supported by the member applications within their respective architectures. Some approaches, more or less applicable to particular circumstances, can be identified.

Interest management capability differences are often resolved in the gateways and middleware used to link multi-architecture simulation environments. In a simulation environment linking HLA member applications to non-HLA member applications using a gateway or middleware that can use the HLA Declaration Management and Data Distribution Management services, configure the gateway/middleware to use those services to subscribe only to data the non-HLA member applications wish to receive [e.g., Griffin et al., 1997], so that only the desired data will be translated and sent on to the non-HLA member applications. (Note: This has the additional benefit of reducing the translation workload of the gateway/middleware.) This approach requires that the gateway/middleware subscriptions be configurable and that it subscribes to the complete set of data required by all of the non-HLA member applications that are connected through it. Such gateway/middleware subscriptions can be static (set during simulation initialization) or dynamic (set and changed by member applications during simulation execution); some gateway/middleware software can support both subscription types [e.g., Hougland and Paterson, 2000]. In a multi-architecture simulation environment linking HLA member applications to non-HLA member applications using a gateway or middleware that can use the HLA Declaration Management but not the Data Distribution Management services [e.g., Wood and Petty, 1999], or one that cannot use either class of services, configure the gateway/middleware to perform internal filtering on the input HLA data it receives before generating output non-HLA network messages. This requires that the gateway/middleware have an internal filtering capability and that the data requirements of the member applications connected via the gateway are somehow input to the gateway; the latter may happen during simulation environment design rather than dynamically during simulation environment execution.

In DIS, site/host filtering is sometimes used to provide a degree of interest management. For DIS member applications that are, or may be, used within a multi-architecture simulation environment, problems related to interest management capability differences can be preempted to an extent by using filtering schemes other than site/host filtering. Failing this, for DIS member applications that do use site/host or similar filtering [e.g., O’Connor et al., 2006] and are
connected to a multi-architecture simulation environment with a gateway, configure the gateway
to use the site/host values corresponding to the member applications that sent the original
messages when translating into DIS PDUs or disable site/host filtering in the receiving DIS
member applications [Gallo et al., 2006].

2.3.2.1.6 Issue: Gateway Usage and Selection Decisions

DESCRIPTION

In the context of multi-architecture simulation environments, gateways are
software/hardware systems that translate data messages and control commands from one
interoperability protocol (e.g., DIS) to and from another (e.g., HLA) during execution of the
simulation environment. Typically (but not always), gateways are stand-alone nodes on the
network that receive messages in one protocol from the network, translate them to another
protocol, and re-send them to the network. Historically, gateways have been widely used to
integrate multi-architecture simulation environments because they have attractive advantages,
including the possibility of integrating a member application developed for single-architecture
simulation environments into multi-architecture simulation environments without modification
and the ready availability of gateways for the most common interoperability protocols. However,
their use potentially incurs certain penalties within the simulation environment:

- Latency for the time required to translate the data from one protocol to another
- Latency for an additional network message transmission if the gateway(s) are separate
  network nodes
- Cost for additional computers if the gateway(s) are separate network nodes
- Computational burden for protocol translation on the member application host computer
  if an embedded gateway is used
- Potentially significant effort required to properly configure multiple gateways

Implementers considering the use of gateways in a multi-architecture simulation
environment face several decisions:

- Is a gateway, in general, an appropriate choice for the simulation environment in
  question?
- If a gateway is appropriate, which of the available gateways should be used, or should a
  new gateway be implemented?
- If an available gateway is used, does it meet the needs of the simulation environment as
  is, or will modifications to it (or the member applications) be required?
These decisions should be informed by knowledge of the gateways available. As to the first question, alternatives to gateway use include integrating common communications middleware into all member applications and modifying all member applications to use a single architecture. As to the second question, different gateways may have different performance, capacity, interoperability protocol coverage, ease of configuration and use, suitability for use in secure environments, and cost. These characteristics determine which gateway, if any, is best for the simulation environment.

**RECOMMENDED ACTION(S)**

Gateways have been and continue to be widely used to connect member applications into multi-architecture simulation environments (e.g., gateways were used beginning with the Platform Proto-Federation [Cox et al., 1996] and, more recently, in the Joint National Training Capability [Bizub et al., 2006] and the Joint Mission Environment Test Capability [LeSueur et al., 2009]). Some large multi-architecture simulation environments will use multiple gateways for different purposes [e.g., O’Connor et al., 2006; Testa et al., 2006]. Despite their ubiquity and utility, gateways are not a multi-architecture panacea. For any particular simulation environment, the decisions of whether to use a gateway and, if so, which one to use, should be informed by knowledge of the gateways available. Gateways should be used only for simulation environments where the advantages outweigh the disadvantages. In other situations, mechanisms/techniques other than gateways such as modifying all member applications to use a single architecture or the use of multi-protocol common middleware integrated into all member applications may be more appropriate. For example, common middleware was used for the Training and Doctrine Command (TRADOC) Battle Lab Collaborative Simulation Environment because of latency, data loss, and cost concerns [Rieger and Lewis, 2006].

When deciding which of the available gateways should be used, or whether a new gateway should be implemented, different gateways may have different performance [Gminder et al., 1996], capacity [Cox et al., 1997], interoperability protocol coverage [Wood et al., 1997; Wood and Petty, 1999], ease of configuration and use, suitability for use in secure environments, and cost. These characteristics determine which gateway, if any, is best for the simulation environment; the simulation environment designers should consider the requirements of the intended application and select the gateway that best meets those requirements. For many simulation environments, commercial gateways may be available and suitable [O’Connor et al., 2006; Rieger and Lewis, 2006]; they are among the product offerings of many simulation software vendors. Government-owned gateways may be found in M&S resource repositories. Some architectures, e.g., TENA, include code generators that can assist in producing application-specific gateways [Hudgins, 2009], although such generated gateways may require additional implementation effort [LeSueur et al., 2009].
Some multi-architecture simulation environments may require the integration of member applications that use more than two architectures, including distributed simulation interoperability protocols (e.g., DIS, HLA, TENA, Common Training Instrumentation Architecture [CTIA]) and other related protocols (e.g., SIMC4I Interchange Module for Plans, Logistics, and Exercises, Tactical Automated Data Information Link [SIMPLE, TADIL]). The situation is exacerbated if multiple versions of a single protocol (e.g., HLA 1.3 and IEEE 1516 versions) are used. Because gateways are typically developed to translate between a single pair of architectures (e.g., an HLA–DIS gateway), developers of a simulation environment with multiple protocols may face the need to acquire, configure, test, and support gateways for every pair of protocols in use or to acquire or develop more capable or configurable gateways. In these situations, they should attempt to acquire or develop gateways that can translate between multiple protocols and SDEMs.

2.3.2.1.7 Issue: Gateway Translation Paths

DESCRIPTION

Each gateway in a multi-architecture simulation environment constitutes a translation path between one pair of architectures. If a multi-architecture simulation environment is configured with two or more translation paths between a single pair of architectures, duplicate translation of data messages and/or control commands may occur, resulting in redundant and potentially inconsistent information being sent to member applications.

RECOMMENDED ACTION(S)

To resolve this issue, implementers should ensure that there is at most one translation path between a given pair of architectures in a multi-architecture simulation environment. Often this can be done by using at most one gateway for each pair of protocols. However, if the specialized translation needs of the member applications or the performance requirements of the overall simulation environment require the use of more than one gateway, then the data each gateway is translating should be partitioned to avoid redundant receipt and translation, either by separating the networks each gateway is connected to [O’Connor et al., 2006] or by configuring the gateways to translate mutually exclusive subsets of the incoming data.

However, even if there is only one gateway between each pair of protocols, multiple paths can arise indirectly. For example, suppose three single-architecture simulation environments A, B, and C are connected via three gateways (A–B, A–C, B–C) into a multi-architecture simulation environment; then data sent from A can reach C directly through the A–C gateway, and also indirectly, through the A–B and B–C gateways. A straightforward analysis of the simulation environment connectivity should reveal this problem. Physically separating the networks each gateway is connected to may resolve the issue [O’Connor et al., 2006].
2.3.2.1.8 Issue: DIS Heartbeat Translation

DESCRIPTION

The DIS protocol requires that Entity State PDUs be sent for all entities at a standard-specified minimum frequency, even if the entities are completely static; these PDUs are commonly known at the DIS “heartbeat.” This can create unnecessary message traffic and overhead in non-DIS simulation environments that send object attribute updates only when an attribute value has changed. On the other hand, failure to generate such “heartbeat” updates by non-DIS member applications can cause unintended entity deletion in DIS simulation environments as a result of “time out.”

RECOMMENDED ACTION(S)

Member applications that generate DIS Entity State PDUs for transmission to DIS member applications, either for their own entities or as a result of translating non-DIS updates (e.g., HLA-to-DIS translation in a gateway), should be implemented to do so for each simulated entity at a rate that satisfies the DIS heartbeat requirements, even for those entities that have not had any attributes updated. A straightforward approach is to implement a time-dependent update cycle within those member applications that produces the DIS heartbeat update at the required rate. This will require maintaining a complete set of attributes for each simulated object from which to produce the update. This approach was used in the Distributed Mission Operations Portal [Valle et al., 2006].

Member applications that receive DIS Entity State PDUs, either because they need the state of the simulated objects for their own models or to translate them into non-DIS updates (e.g., DIS-to-HLA translation in a gateway), should be implemented to appropriately handle DIS heartbeats for static entities (those where no attribute has changed since the last update). For example, a gateway performing DIS-to-HLA translation should determine, upon receiving a DIS update, which of the simulated object’s attributes have changed and only generate HLA object attribute updates for those changed attributes; it is possible that none changed and no HLA update at all is generated. This capability is present in middleware supporting DIS–HLA interoperability in the TRADOC Battlelab Collaborative Simulation Environment [Rieger and Lewis, 2006].

This issue and its recommended actions can be seen as a special case of the closely related Issue 2.3.2.1.1 (Object State Update Contents), and a solution to one issue will often be closely associated with a solution to the other issue. Both of the actions recommended for this issue are normally implemented in DIS–HLA gateways [e.g., Cox et al., 1996]. However, this issue could arise even if DIS is not involved, as some non-DIS applications have also found it useful to implement DIS-style heartbeats. For example, the Joint Experimental Federation, an
HLA simulation environment, did so to avoid spikes of network traffic caused by object attribute queries [Ceranowicz et al., 2002].

2.3.2.1.9  Issue: Multi-architecture and Inter-architecture Performance

**DESCRIPTION**

When multiple distributed simulation architectures and member applications developed for multiple architectures are linked into a single simulation environment, performance should be considered. In some (but not all) multi-architecture simulation environments, inter-architecture differences in runtime performance may exist. If present, inter-architecture performance differences may result from the fundamental design assumptions of the distributed simulation architectures or the implementations of the architectures’ supporting software (e.g., HLA RTI, TENA middleware). Performance issues may also arise from the technical solutions used to link the components of the multi-architecture simulation environment (e.g., gateways) and the implementations of the member applications. For some simulation environments and some applications, performance differences significant enough to affect the utility of the multi-architecture simulation environment are possible.

**RECOMMENDED ACTION(S)**

The first recommended action in this regard is to determine if a multi-architecture or inter-architecture performance issue actually exists. It is possible that there are no significant performance issues; in particular, the existence of performance issues should not be assumed from outdated preconceptions based on previous generation software implementations that have been superseded by improved software and higher speed networks. Careful and controlled measurement of performance within the simulation environment, perhaps using existing monitoring tools, can establish whether this issue is present or not; performance testing should include any gateways [White, 2001]. Such performance testing is likely to be necessary to determine if the simulation environment meets the performance requirements of the application [Williams and Smith, 2007], so this step does not necessarily imply great additional effort.

If multi-architecture and inter-architecture performance issues do exist, several non-mutually exclusive approaches are possible. First, designers may be able to preempt the issue altogether by selecting architectures and member applications with runtime performance appropriate to the application. As noted for Issue 2.3.2.1.6 (Gateway Usage and Selection Decisions), any performance penalties associated with gateways should be considered before choosing to integrate a multi-architecture performance environment with a gateway (or gateways) and when selecting a particular gateway. Whenever possible, member applications that will need to carry out high-performance, tightly bound interactions with each other should use the same architecture so as to avoid inter-architecture translation latency. Finally, existing
techniques such as smoothing, dead reckoning, and heartbeats may be adapted to provide object attribute updates at the rate and precision needed for performance-sensitive member applications in a multi-architecture simulation environment [e.g., Marsden et al., 2009].

2.3.2.1.10 Issue: Translating Non-ground-Truth Network Data

DESCRIPTION

A common assumption in distributed simulation architectures is that data sent on the network describing the state of the simulated world is “ground truth,” i.e., is correct with respect to that simulated environment. It is typically left to individual member applications to intentionally degrade or corrupt that ground-truth information in situations where a system or entity they are simulating would not have access to perfect and complete information. In some simulation environments, however, specialized member applications are used to perform that information degradation (e.g., modeling weather and terrain effects on communications in a communications effects server), and the degraded information is retransmitted on the network to other member applications. In multi-architecture simulation environments, such retransmitted degraded information must be translated from the originating architecture’s data model and protocol into the other architecture’s data model and protocol. This translation may occur in a gateway, in middleware, or elsewhere. The deliberately incorrect data may cause problems through its violation of the ground-truth assumption (e.g., seemingly inconsistent location data for a transmitting simulated entity) and the fact that information is transmitted twice (i.e., first in its original correct form and second in its degraded form).

RECOMMENDED ACTION(S)

This issue arises only if the deliberately incorrect non-ground-truth data is retransmitted in a form that is otherwise identical to the ground-truth data and thus indistinguishable from it, e.g., two correctly formatted DIS Entity State PDUs with different locations for the same simulated object. It is distinguishable from Issue 2.3.2.1.7 (Gateway Translation Paths) in that it arises not from inadvertent redundant translation in a gateway but from deliberate alteration and retransmission of data. One resolution to this issue is to use architecture features to distinguish ground-truth from non-ground-truth data. Such features could include different message types (e.g., a special HLA interaction class [Carr and Roberts, 1997; Lacetera and Torres, 1997]) or flags within a single message type. The translators (gateways, middleware) used to link the multi-architecture simulation environment must be able to correctly translate these non-ground-truth indicators into a form that conveys the same information after the translation.

If no suitable architecture/protocol features are available, it may suffice to modify the affected member applications to be aware of the sources of the different types of data (e.g., ground truth from the member application simulating an object, and non-ground truth from a
communications effects server) and to use only the incoming information from the desired source; this is an example of receiver-side filtering. Such modifications and intentions should be documented in advance in the simulation environment agreements (see Section 2.4.2 [Activity 4.2: Establish Simulation Environment Agreements]).

### 2.3.2.1.11 Issue: Object Identifier Uniqueness and Compatibility

**DESCRIPTION**

Many object simulation and management operations within an architecture require unique object identifiers. For example, in HLA the RTI generates a federation-wide unique object name string when the object is registered. Similar operations may occur in other architectures. In a multi-architecture simulation environment, measures should be taken to ensure that (1) the object identifiers are unique across architectures and (2) object identifiers generated within one architecture can be used in another architecture to reference the identified object.

**RECOMMENDED ACTION(S)**

The specifics of an approach to resolve this issue depend on the object identifier requirements of the different architectures linked into the multi-architecture simulation environment. However, in multi-architecture simulation environments that use a gateway or middleware, the gateway/middleware can be configured or modified to translate or map object identifiers used in one architecture to an object identifier acceptable in the other architecture. This translation or mapping must consider both format and uniqueness requirements.

Uniqueness can often be assured by using the services or conventions already available in each architecture for that purpose (e.g., the object registration services in HLA, which return object identifiers unique within the HLA federation execution [Simulation Interoperability Standards Committee of the IEEE Computer Society, 2000]) with the gateway/middleware using those services as if it were the originating member application for the simulated objects whose data it is translating. For classes of simulated objects with object attributes that may be references to other objects using their object identifiers, member applications may need to store extra information with those object references, such as object class (datatype) and originating member application of the referenced object, to resolve situations where an update to that object reference attribute is received before the referenced object itself is discovered [Nielsen and Salisbury, 1998].
2.3.2.1.12 Issue: Developing a Distributed Simulation Environment Composed of Classified and Unclassified Components

**DESCRIPTION**

When implementing multi-architecture environments, a Cross-Domain Solution (CDS) is often required to support the various users with different security clearances and to prevent users from obtaining access to information for which they lack authorization. A CDS is a combination of hardware and software that controls the passing of data between two or more domains with different levels of security classification, for instance, UNCLASSIFIED and SECRET or SECRET and TOP SECRET. The security requirements of CDS often take significant time to put into place. Software interfaces of CDS systems are typically not configurable. While a CDS is critical to supporting the classification and data distribution requirements, it may cause problems supporting development, integration, and test activities of the simulation environment.

**RECOMMENDED ACTION(S)**

The difficulty of implementing a CDS in a multi-architecture simulation environment introduces additional challenges over that of a single-architecture environment. The multi-architecture environment may force the use of gateways to get data in and out of the CDS and could potentially require data conversions that would be unnecessary in a single-architecture environment. The CDS may be either unidirectional or bidirectional and may involve the removal of portions of the data that flow through it to meet given security requirements. Data conversion could also be forced when an architecture version in use has been updated but the CDS has not been through the process of updating its internal architecture usage because of security constraints and timelines (e.g., the use of TENA v5.2.2 in a CDS and the use of TENA v6.0 by member applications).

The simulation environment designer should consider partitioning applications on either side of the CDS so as to minimize the conversion necessary through a gateway. For example, if TENA and DIS are participating architectures and the CDS only supports TENA, the recommended approach is to place all DIS application on the same side of the CDS if at all possible. Native TENA data can flow to and through the CDS on both sides, and translation to/from DIS would only be required at a single point. While it is recognized that this is not always possible, the key is to look at the how the CDS affects the overall architecture and to minimize gateway usage where possible.

2.3.2.1.13 Issue: Multi-Architecture Save and Restore

**DESCRIPTION**

For some applications of distributed simulation environments, such as those that execute for long periods of time to support a training exercise involving many facilities and personnel,
the ability to periodically save the state of an executing simulation environment and later restore to that saved state, perhaps after a catastrophic crash of the system or a planned break in the exercise, is very important. Accurate and reliable save and restore operations can be challenging even within a single-architecture simulation environment. Coordinating save and restore across architectures within a multi-architecture simulation environment may require special procedures and/or enhancements to the gateways/bridges/middleware connecting the simulation environments and the member applications within them.

**RECOMMENDED ACTION(S)**

The simulation environment designer should analyze and document the save and restore capabilities for each architecture in the simulation environment. The objective is to save a snapshot of the simulated state of the simulation environment as a reference point for future use. The primary practical means to achieve this objective in a multi-architecture simulation environment is to simultaneously (in simulation time) save the simulated state of each of the member applications. Automated processes to save and restore simulation states are preferred but manual procedures may have to be used. Simulation environment implementers should develop the procedures that will be used to initiate the collective saving of simulation state and, later, the restoring of the saved state, if necessary. Save and restore procedures are potentially the topic of a simulation environment agreement (see Section 2.4.2 [Activity 4.2: Establish Simulation Environment Agreements]).

### 2.3.2.1.14 Issue: Network Protocol Configuration

**DESCRIPTION**

Typically, distributed simulation protocols (such as DIS, HLA, and TENA) exploit configuration options and features in the underlying network (also known as over-the-wire) protocols that support them. In a multi-architecture simulation environment, different network protocol configuration options may be preferred by the different architectures (e.g., IP Multicast vs. IP Broadcast), introducing the potential for incompatibilities.

**RECOMMENDED ACTION(S)**

The goal should be to limit the number of network protocols to the greatest extent possible in order to maximize throughput and minimize latency over the network. For example, one architecture may be more efficient and configurable over the network (e.g., TENA vs. DIS). The design of the simulation environment should include a conversion of the data to the more efficient protocol, and within that protocol, exploitation of available network configuration options to enhance performance [e.g., Moulton et al., 1998]. This conversion typically includes the use of gateways to effectively convert the data between the distributed simulation protocols.
An exception to this recommendation would be when the simulation environment executes within a single local area network (LAN) and throughput and latency are not a concern as they are when using a wide area network (WAN) [O’Connor et al., 2006]. While gateways may be required to convert the data, they would not be in place specifically to address the maximization of throughput and minimization of latency over the network.

An additional consideration is the configuration of network devices such as routers and switches to support multicast over a WAN [Lasch and Paschal, 2000]. The use of UDP multicast traffic over the WAN provides many efficiencies over Transmission Control Protocol (TCP) traffic but is difficult to configure. This is especially true over secure networks that include encryption equipment such as Tactical Local Area Network Encryption (TACLANE) devices. Configuration by network experts who understand multicast and know how to correctly configure network devices to support it is critical and should be accounted for when designing the simulation environment.

2.3.2.2 Consolidation of “Design Simulation Environment” Activities to Support Multi-architecture Events

MULTI-ARCHITECTURE-SPECIFIC ACTIVITY INPUTS

- List of architectures necessary to support the selected member applications
- Development and execution processes for the selected architectures
- Information on common communications middleware
- Information on available gateways

MULTI-ARCHITECTURE-SPECIFIC TASKS

- Perform trade-off analysis on whether to use gateways or common communications middleware, or to modify member applications to migrate to a different architecture.
- Allocate member applications to architectures.
- Select over-the-wire protocols.
- Allocate member applications and architectures to enclaves in CDS.
- Analyze translation paths and decide on an approach to avoid redundant translations.
- Perform preliminary testing to identify multi-architecture and inter-architecture performance issues.
- Conduct architecture training sessions.
- Document agreed-upon common terminology across architecture communities.
- Select gateways.
- Identify an approach for mapping object identifiers across architectures.
- Depending on simulation environment requirements, some of the following may also be necessary:
  - Identify necessary gateway modifications.
  - Develop procedures for initiating save and restore of the simulation environment state across member applications.
  - Decide on an approach to perform object state updates.
  - Decide on an approach to perform object ownership transfer.
  - Decide on a time management scheme.
  - Decide on a DIS heartbeat approach.
  - Decide on an approach to interest management.
  - Decide on an approach to differentiate between ground-truth and non-ground-truth data.

**MULTI-ARCHITECTURE-SPECIFIC ACTIVITY OUTCOMES**

- Within “Design simulation environment” (per the DSEEP)
  - Architecture design identifying how:
    - Gateways partition the environment into segments for the different architectures
    - Member applications are assigned to architectures
    - Member applications and architectures are mapped to CDS segments
    - Selection of over-the-wire-protocols

- Within “Implied requirements for member applications modifications” (per the DSEEP)
  - Requirements to modify member applications to meet object state update agreements
  - Requirements to modify member applications to meet object ownership agreements
  - Requirements to modify member applications to meet the selected time management scheme, which can involve
2.3.3 Activity 3.3: Design Member Applications

The purpose of this activity is to transform the top-level design for the simulation environment into a set of detailed designs for the member applications. The scope of the design task will depend on the amount of previous design work that can be reused. New member applications will generally require a substantial amount of design effort whereas modifications to existing member applications will require less effort.

2.3.3.1 Issues

2.3.3.1.1 Issue: New Member Application Architecture

DESCRIPTION

There are applications for which the requirements of a simulation environment dictate that a new member application be developed. In a single-architecture simulation environment,
the new member application is implemented to operate within that architecture. However, in a multi-architecture simulation environment, an additional decision arises: For which of the architecture(s) in the simulation environment is the new member application implemented? This design decision has both short-term and long-term implications, such as implementation effort and reusability of the member application, respectively.

**RECOMMENDED ACTION(S)**

Although the choices are likely to be clear (should the new member application use architecture A, B, or C?), the factors that influence the decision are varied and inter-related. Designers should consider each of them, appropriately weighted for the specific situation, before choosing the architecture for the new member application. Those factors include the following:

- *Architectures in the simulation environment.* It is unlikely that it makes sense to select an architecture for the new member application that is not among those already planned for the simulation environment.
- *Requirements of the current simulation environment.* Does one candidate architecture or another for the new member application better support the simulation environment’s requirements?
- *Possible future uses of the new member application.* Which of the candidate architectures allows the member application to be reused most effectively in the future?
- *Expertise of the development team.* Which of the candidate architectures are the new member application developers best able to work with?
- *Selected architecture integration method and effort.* Which architecture integration method(s) (i.e., gateway, middleware, or native integration) for the selected architecture are being considered for the new member application, and how much effort will be required for each of those methods?
- *Non-selected architecture integration method and effort.* How will the new member application be integrated with the other non-selected architectures in the simulation environment (i.e., gateway, middleware, or native integration), and how much effort will be required for each of the non-selected architectures?
- *Multi-purpose middleware.* Is there existing multi-purpose communications middleware software supporting multiple architectures that could be integrated with the new member application?
- *Member application-specific architecture services.* Does the member application, because of its design objectives, make use of architecture services specific to or limited to one of the candidate architectures (e.g., time management)?
- *Testing tools.* Are there testing tools available and suitable for the new member application that operate within one of the candidate architectures?
• Security/classification level. Will the member application be required to operate at security/classification levels higher than unclassified, and if so, what is the availability of security cross-domain solutions for each of the candidate architectures?
• Standards and mandates. Are there standards or mandates relevant to the new member application that require the use of a specific architecture?
• Sponsor guidance. What guidance regarding the candidate architectures does the organization sponsoring the development of the new member application have?

2.3.3.2 Consolidation of “Design Member Applications” Activities to Support Multi-architecture Events

MULTI-ARCHITECTURE-SPECIFIC ACTIVITY INPUTS
• None beyond those called for in the DSEEP

MULTI-ARCHITECTURE-SPECIFIC TASKS
• Within “Design member applications” (per the DSEEP)
  o Decide on which architecture to use for new member applications.

MULTI-ARCHITECTURE-SPECIFIC ACTIVITY OUTCOMES
• Within “Member application designs” (per the DSEEP)
  o For new member applications, identification of which architecture to use

2.3.4 Activity 3.4: Prepare Detailed Plan

The purpose of this activity is to develop a coordinated plan to guide the development, test, and execution of the simulation environment. This requires close collaboration among all participants in the simulation environment to ensure a common understanding of the various goals and requirements and also to identify (and agree to) appropriate methodologies and procedures based on recognized systems engineering principles. The plan should include the specific tasks and milestones for each member application, along with proposed dates for completion of each task.

2.3.4.1 Issues

2.3.4.1.1 Issue: Cost and Schedule Estimating for Multi-architecture Development

DESCRIPTION

In the detailed planning phase, existing work breakdown structures should be refined to account for implementation of defined design solutions. Depending on the specified roles and
responsibilities for the various technical teams and the nature of the design solutions, initial schedule and cost estimates may be significantly impacted. This is particularly true in multi-architecture applications, as proper coordination across disparate developer teams requires frequent technical interchange meetings and extensive management controls. If sponsor resources are deemed to be insufficient to fully implement the revised plans, undesirable “shortcuts” could be implemented to address resource gaps, potentially affecting the quality of simulation results.

**RECOMMENDED ACTION(S)**

Experience shows that three factors influence the estimation of cost and schedule for multi-architecture simulation environment development. The first factor is whether team members can support the defined schedule of development, integration, and test events with the resources provided. Certain personnel will be critical to the success of these events, and thus some degree of deconfliction of personal schedules may be required, as well as downscoping of responsibilities if available resources for certain individuals are insufficient. The second factor is the availability of the design articles identified in the previous DSEEP activity (e.g., gateways, CDS, networking tools) and the affordability of these articles given resource constraints. The third factor is the availability of the appropriate facilities. Simulation facilities are normally high-use assets and consequently require a lengthy lead time for scheduling. All of these factors should be accounted for in revised schedules and funding estimates.

**2.3.4.1.2 Issue: Tools for Verifying Multi-architecture Data Exchanges**

**DESCRIPTION**

Tool availability is an important issue with respect to multi-architecture development. The complexity of constructing a multi-architecture simulation environment could require additional or different computer-aided software engineering (CASE), testing, or monitoring tools to support integration, test, and execution of such an environment. One critical test for the integrity of the simulation environment is verifying the ability of gateways or middleware to correctly perform their intended functions. Significant errors can occur during integration that can, in turn, lead to costly rework if the detailed planning documents do not adequately address the need for testing and monitoring tools during simulation environment integration and testing.

**RECOMMENDED ACTION(S)**

The obvious recommendation is to use tools that the team has available and has experience using. But what if those tools are inadequate for multi-architecture development? Or worse, what if there are no consistent tools used that would benefit the multi-architecture environment?
At a minimum, tools should be available to verify the application interfaces and verify that the data flowing across those interfaces conforms to established data exchange agreements. This is especially critical in a multi-architecture environment because of the potentially significant differences in syntactic and semantic meaning of interface definitions and data across the architectures. Ideally, tools should be available to perform the verification on each side of the interface. If the necessary interface verification tools are not available on both sides of the interface, there are several options that can be taken: search for existing tools, use tools that may exist for only one side of a multi-architectural interface, or build tools that don’t exist on either side. This issue reinforces the point of Issue 2.3.2.4 (Inconsistent Development and Execution Processes).

Searching for existing tools can be accomplished through an architecture-specific repository such as the TENA website/wiki (https://www.tena-sda.org/), a government off-the-shelf (GOTS) repository such as the DoD Modeling and Simulation Information System (MSIS) (a DoD modeling and simulation search engine that has the ability to search across individual service repositories, http://msrr.dod-msiac.org/), or even searches through standard Internet-based search engines.

If a tool exists on one side of the architectural interface, it may be used for interface verification. The caveat is that there is no easy way to verify the data on the “other” side of the interface. Generally, it is always preferable to have tools on each side of the architectural interface.

If no tool exists or none is found that meets the needs of the simulation environment, development of a new tool may be undertaken. This is always a trade-off of resources and time not typically built into the development schedule; however, the ability to quickly confirm and verify interfaces can reduce testing time and buy back some potentially lost time sacrificed to build a new tool. One advantage is that the tool will be available for future use and will likely reduce development time in the future.

In addition, the way the different architectures are built and operate should be correlated and documented as part of the simulation environment detailed planning process. For instance, there should be a way for the test controller to have a quick-look picture of the health and status of the individual applications and capabilities participating in a given test. While each architecture involved may provide its own unique way of displaying health and status updates, these need to be correlated into a usable picture so that the test conductor can make quick and effective go/no-go decisions based on what is currently executing.
2.3.4.1.3 Issue: VV&A for Multi-architecture Applications

DESCRIPTION

VV&A are important issues with respect to multi-architecture simulation environment development. VV&A activities should occur concurrently with implementation activities to the extent possible. Multi-architecture environment developers and users are likely to need additional information and expert assistance because of their lack of familiarity with the implementation details of other architectures. Multi-architecture environment infrastructure functions need to be checked to identify elements that could have adverse impacts upon the validity of the simulation environment. The performance of the multi-architecture environment needs to be verified to ensure that information exchange among the participating members happens as planned. Failure to plan for the VV&A activities required by a multi-architecture simulation environment can result in costly problems later in the multi-architecture engineering and execution process.

RECOMMENDED ACTION(S)

VV&A, in the general sense, has a large amount of literature consistent with its importance in the practice of M&S [e.g., Balci, 1998; Petty, 2010]. However, aspects of VV&A specific to multi-architecture simulation environments are much less well documented. Some important VV&A considerations for multi-architecture simulation environment developers include the following:

1. **Multi-architecture-specific components.** The simulation environment may include components specific to its multi-architecture design, such as gateways and middleware. These “additional” components should be verified, to confirm that they are operating as specified and intended (e.g., verification that a gateway is translating all of the message types it is expected to), and validated, to confirm that they are not degrading the simulation’s validity (e.g., validation of real-time response in a training simulation including a gateway that imposes latencies) [e.g., Harkrider and Petty, 1996].

2. **Distributed simulation support operations.** Executing a distributed simulation typically requires a set of specialized architecture services that provide functionality that is part of the simulation’s infrastructure rather than part of its modeling. Examples include object naming, distributed logging, enumeration control, and pause and resume. In a multi-architecture simulation environment these operations are more complex because they must be coordinated and reconciled across multiple architectures. Ensuring that these services operate correctly in a multi-architecture simulation environment will require additional specialized verification [Williams and Smith, 2007].
3. **Correlation.** Because of the different design heritage and development histories of its separate architectures, a multi-architecture simulation environment is arguably more likely than a single-architecture simulation environment to have multiple representations of the same object, e.g., multiple terrain databases in different formats or multiple dynamics models of the same entity. Correlation (agreement) between multiple representations of the same object is a VV&A consideration in any simulation environment, but it is exacerbated in the multi-architecture context. Additional V&V testing focused on correlation is likely to be required (e.g., terrain correlation in the Platform Proto-Federation [Petty et al., 1996] and the Urban Resolve 2015 experiment [Williams and Smith, 2007]).

4. **Validation of architecture elements.** Some architectures include standard models as part of the architecture itself (e.g., dead reckoning in the DIS specification or coordinate conversion in the TENA middleware). When member applications that use such architectures are linked into a multi-architecture simulation environment, it may be necessary to validate those architecture-embedded models in the context of their interactions with other parts of the overall simulation environment.

5. **Multiple accreditors.** Because the simulation environment is multi-architecture, there may be multiple accreditors (organizations responsible for approving the use of the simulation environment for the intended application). Additional V&V testing, and additional documentation of those tests, may be needed to meet the different testing and documentation requirements of the multiple accreditors. Simulation designers should allow for this effort during planning.

6. **Multiple architecture communities.** Similar to the previous consideration, the communities associated with the different architectures may have different expectations for V&V testing and for documentation of those tests, thus adding to the VV&A effort for a multi-architecture simulation environment. Simulation designers should allow for this effort during planning.

### 2.3.4.1.4 Issue: Multi-architecture Data Collection

**DESCRIPTION**

Data collection in a multi-architecture simulation environment execution is critical for effectively determining and using the results of a given test/exercise/experiment. The multi-architecture design of the event offers multiple sources, locations, and architectures/protocols from which to extract data to determine the appropriate selection and application of data collection, analysis, and management tool(s). This greatly complicates efforts to translate the raw data into desired measures.
**RECOMMENDED ACTION(S)**

Two strategies are used to collect data in simulation environments. One is to centralize data collection at a single point. The other is to localize data collection at individual member applications. Centralized data collection simplifies the collection by isolating the data collection to a single location. However, centralized data collection has been shown to be a system performance inhibitor as comprehensive data collection can be a time-consuming process when great quantities of data are produced during the simulation event. On the other hand, while localized data collection does not impose a performance burden on system performance, ensuring the collection of data is complete is more difficult and correlating the data is more complex. The use of multi-architectures makes data collection even more difficult in that architecture-defined data formats will be used. The amount of data and the complexity of correlating the data will be increased because of the use of multiple architectures. Simulation environment developers should implement and test the data collection and correlation scheme that seems to be most appropriate for the simulation environment and should be prepared to make modifications if problems are discovered.

**2.3.4.1.5 Issue: Tool Incompatibility**

**DESCRIPTION**

Tools used in the development and execution process may be incompatible across architectures. This applies to both development tools and execution tools. An example of developmental tool incompatibility is that different object model editors cannot exchange data because of a lack of syntactic interoperability (incompatible Data Interchange Format [DIFs]) or a lack of semantic interoperability (incompatible object model structure or data elements). An example of execution tool incompatibility is that different data loggers have incompatible timestamps because of architecture time management characteristics.

**RECOMMENDED ACTION(S)**

Since it is unlikely that teams across architectures will be familiar with the tools commonly used with the other architecture(s), an evaluation should be performed of the tools available in each architecture and tool usage decisions should be made as early as possible in the development life cycle. If possible, agreeing to use tools primarily from a single architecture has the potential to reduce conflict and confusion as development proceeds.

Standardizing on one tool set as provided by a given architecture may introduce schedule risk because of unfamiliarity and additional training necessary to use that architecture’s tool within the context of another architecture. However, the positive aspect is that it could reduce technical risk by minimizing the tools necessary to perform a given function as well as minimizing the tool instances and installations required. Tools such as the RTI Console and
TENA Console (event management utilities that can be used for monitoring applications joined to an execution) are specific to a given architecture and would not be directly replicated in other architectures.

The goal is to select common tools where available and applicable. Questions that should be asked include: Are there tools from one architecture that could be used in the other architecture(s) (e.g., gateway data verification)? Is it necessary to resort to duplicative usage of multiple tools across architectures simply because that is what exists in the other architecture(s)? Is it possible to update or convert existing tools to support the other participating architecture(s)? Focusing on minimizing redundant instances of similar tools across the multi-architecture environment will have a positive impact on the ability to execute as efficiently as possible.

On-going efforts within the Live-Virtual-Constructive Architecture Roadmap Implementation are working to reduce this issue.

2.3.4.2 Consolidation of “Prepare Detailed Plan” Activities to Support Multi-architecture Events

Multi-architecture-specific Activity Inputs

- Personnel with experience in multi-architecture environment
- List of selected gateways
- Information on available facilities

Multi-architecture-specific Tasks

- Refine cost and schedule to account for multi-architecture concerns.
- Extend the existing DSEEP Task to include testing and verification tools for multi-architecture environments:
  - “Complete selection of necessary management tools, reusable products and simulation environment support tools, test and monitoring tools, and develop plan for acquiring, developing, installing and utilizing these tools and resources. Select tools that are applicable across architectures.”
- Extend the existing DSEEP Task to include multi-architecture V&V concerns:
  - “Revise verification and validation plan and test plan (based on simulation environment test criteria), including multi-architecture concerns—V&V of gateways, implementation of support functions, multiple representations, and representations included as part of the architectures, as well as issues with multiple accreditors.”
• Extend the existing DSEEP Task to include multi-architecture data collection concerns:
  o “Finalize the data management plan showing plans for data collection, data correlation, management, and analysis.”

MULTI-ARCHITECTURE-SPECIFIC ACTIVITY OUTCOMES
• Within “Simulation environment development and execution plan” (per the DSEEP)
  o Revised cost estimate to account for multi-architecture concerns

2.4 STEP 4: DEVELOP SIMULATION ENVIRONMENT

The purpose of this step of the DSEEP is to define the information that will be exchanged at runtime during the execution of the simulation environment, modify member applications if necessary, and prepare the simulation environment for integration and test.

2.4.1 Activity 4.1: Develop Simulation Data Exchange Model

In order for the simulation environment to operate properly, there must be some means for member applications to interact. At a minimum, this implies the need for runtime data exchange, although it could also involve remote method invocations or other such means of direct interaction among cooperating object representations. Clearly, there must be agreements among the member applications as to how these interactions will take place, defined in terms of software artifacts like class relationships and data structures. Collectively, the set of agreements that govern how this interaction takes place is referred to as the SDEM.

2.4.1.1 Issues

2.4.1.1.1 Issue: Meta-model Incompatibilities

DESCRIPTION

Differences in the underlying data exchange model structures used by the various architectures can cause incompatibilities in a multi-architecture environment. Specifically, the set of data fields that compose an HLA Federation Object Model (FOM) (as specified in the HLA Object Model Template [OMT]), the set of fields that compose a TENA Logical Range Object Model (LROM) (as specified in the TENA metamodel), and the set of fields that define DIS PDU structures are not the same.

Since the SDEMs must align among the architectures in the multi-architecture environment, the team establishing the SDEM must understand these metamodel differences,
understand the equivalencies and differences across the metamodels, and take actions so that each architecture’s metamodel specifications are met in a consistent manner.

**RECOMMENDED ACTION(S)**

The fundamental representation of data in the metamodel of a given SDEM must be correlated across the architectures used in a given simulation environment. It is critical to the success of the simulation environment that the semantic meanings of the data representation in each SDEM be consistent. There are two recommended ways of resolving these incompatibilities: using an architecture-neutral way of representing the metamodel and the use of gateways.

The ideal solution is to use an architecture-neutral way of representing the metamodel. While there is on-going work to develop architecture neutral modeling mechanisms, as in the Joint Composable Object Model (JCOM) effort, it is most likely that the participating architectures model their attributes and behaviors in unique ways. Monitoring efforts such as JCOM and looking for opportunities to implement them into existing architectures is recommended.

The use of gateways is the primary recommended action for resolving existing metamodel incompatibilities. Several factors need to be addressed once the decision is made to use a gateway to link disparate architectures. The main questions to ask are: How do you choose the gateway? How do you know that the metamodel incompatibilities have been addressed? Is there a tool available to support gateway development across architectures?

Choosing the right gateway for a multi-architecture simulation environment is not necessarily a difficult task. It is recommended to search any repositories of the participating architecture communities first [Lutz et al., 2010]. For example, the TENA community maintains a set of gateways between TENA and several different HLA variants as well as DIS. Information on these gateways as well as the gateway itself is available in the TENA repository. Other architecture-neutral repositories such as service-specific Model and Simulation Resource Repositories (MSRRs) are available and can be easily searched. In addition, a number of commercial gateway products are available and can be readily used. While the decision about which gateway to use may be dictated by schedule and budget, there are many available gateway solutions and it is unlikely that the environment developers will need to build one from the ground up.

Testing the gateway to ensure it is accurately translating the metamodel data is critical to the success of the simulation environment. Data on both sides of the gateway should be verified for syntactic and semantic accuracy. In addition to any tools provided by the gateway for data verification, architecture-specific data verification tools, if they exist, should be used to confirm or identify problems in the data translation.
Sometimes there is no existing gateway that meets the requirements of the simulation environment; in that case, one should be built. Tools are available that greatly facilitate the generation of custom gateways that can then be reused on future projects. One such tool is the Gateway Builder (GWB) tool. The advantage of a tool like GWB is that mapping between architectures becomes an easy task once architectural details have been modeled in GWB. GWB already supports many architectures, including TENA R5.2.2 and R6, HLA (MATREX and Pitch), and DIS.

2.4.1.1.2 Issue: SDEM Content Incompatibilities

DESCRIPTION

Once the differences in metamodels are understood and addressed, the alignment of SDEMs across the multi-architecture environment should occur. The semantics of data to be exchanged must be understood and should be equivalent across the architectures in use.

The issues of SDEM alignment across architectures is not tremendously different from the issues that arise within a single-architecture environment. This is not to say it is a simple or quick process. Although names of classes, attributes, and enumeration values give a good clue as to consistencies and inconsistencies across SDEMs, the process of comparison is often largely a manual process. If not performed adequately, semantic inconsistency across the multi-architecture environment can occur.

In some architectures, the metamodel and the content of the SDEM are defined as part of the architecture specification. DIS and TENA both take this approach, although they also both provide mechanisms to extend the standard content with additional SDEM elements based on the needs of the application (e.g., DIS expedient PDUs, TENA LROM). HLA takes a somewhat different approach, standardizing the SDEM metamodel while allowing users to define SDEM content on an application-by-application basis. While users typically enjoy the flexibility to tailor SDEM content to their immediate needs, it often comes at a price. Specifically, when working in a multi-architecture simulation environment, the wide spectrum of SDEM content across different architecture communities must be fully reconciled (within the context of the current application) if the various member applications are to interoperate correctly. This reconciliation process can be very expensive in terms of both time and resources, and can increase technical risk if not done correctly.

RECOMMENDED ACTION(S)

There are two recommended paths to explore when faced with SDEM content incompatibilities in a multi-architecture environment. First, member applications may be changed to support the native interface of a given architecture. Second, gateways may be used to bridge SDEM content incompatibilities. This could include the use of an architecture-agnostic
gateway/middleware solution. No matter which approach is taken, alignment must occur across the SDEMs: alignment of classes and class hierarchies, alignment of attribute assignments to classes, and alignment of domains (including enumerations) to attributes. As already noted, this aspect of the recommended action is not unlike that required when reconciling member applications’ SDEMs in some single-architecture environments, and solutions used in a single-architecture environment [e.g., Bowers and Cutts, 2007] may be helpful in a multi-architecture environment.

When considering a change to the native interface of member applications, there is a trade-off to consider on how much the SDEM will force changes to the interfaces of each member application. Sometimes this trade-off will constrain the selection of member applications to those that most closely align with the other member applications, within the same architecture and across architectures. Changing the native interface of member applications is usually the most expensive option of the three recommended approaches in both time and resources.

When using a gateway solution, time and resources should be spent to ensure the mappings in the gateway(s) are valid. In order for a gateway to accomplish its task, it is necessary to create detailed data mappings across architectures that specify the data level and data type, exactly what will be passed through the gateway, and how it will be represented on each side. This mapping becomes part of the documentation required to verify the correct operation of the gateway and also serves as a synchronization point between the architecture teams. This is typically a manual process that requires coordination across teams representing the incompatible architectures. When performing this manual process, the simulation environment developers should consider the following types of analysis and similarity metrics:

- **Morphological analysis.** An understanding of word forms (e.g., understanding that “aircraft,” “air_vehicle,” and “UAV” are related).
- **Grammatical analysis.** An understanding of the parts of speech (e.g., the use of “target” as a verb in an operations order versus “target” as a noun indicating an entity being engaged by a weapons system).
- **Semantic analysis.** An understanding of the semantics behind the use of an entity descriptor (e.g., an HLA class attribute) that clarifies the purpose or use of an entity in a distributed simulation environment.
- **Entity name similarity.** If two entities have the same (or nearly the same) name, an analysis should be performed to determine if they represent the same thing in the simulation space.
• **Entity descriptor name similarity.** If two entity descriptors have the same (or nearly the same) name, an analysis should be performed to determine if they represent the same characteristic of the entity.

• **Semantic/usage similarity.** If two entities are used the same way in a distributed simulation environment, an analysis should be performed to determine if they are functionally the same or similar.

An additional option for the use of a gateway would be to select a commercially available, architecturally neutral middleware/gateway product. While this may seem like the easiest solution, the trade-off to be considered here is between time/resources and cost/performance.

### 2.4.1.2 Consolidation of “Develop Simulation Data Exchange Model” Activities to Support Multi-architecture Events

**MULTI-ARCHITECTURE-SPECIFIC ACTIVITY INPUTS**

- Requirements for gateway modifications
  - Separation of ground-truth and non-ground-truth data

- Requirements for gateway configuration, including
  - Interest management configuration
  - DIS heartbeat configuration
  - Object identify mapping
  - Over-the-wire protocol selection and conversion

**MULTI-ARCHITECTURE-SPECIFIC TASKS**

- Identify additional gateway configuration and modification requirements
  - Resolve metamodel incompatibilities.
  - Align SDEM content across architectures.

**MULTI-ARCHITECTURE-SPECIFIC ACTIVITY OUTCOMES**

- Extension of existing DSEEP Outcome to include multi-architecture SDEMs:
  - “Simulation data exchange model for each architecture”

- SDEM mappings

- Updated requirements for gateway modifications
Updated requirements for gateway configuration

2.4.2 Activity 4.2: Establish Simulation Environment Agreements

There are other operating agreements that should be reached among developers and managers that are not documented in the SDEM. Such agreements are necessary to establish a fully consistent, interoperable simulation environment. While the actual process of establishing agreements among all participants in the development effort begins early in the DSEEP and is embodied in each of its activities, this may not result in a complete set of formally documented agreements. It is at this point in the overall process that developers need to explicitly consider what additional agreements are required and how they should be documented.

2.4.2.1 Issues

2.4.2.1.1 Issue: Agreements to Address Multi-architecture Development

DESCRIPTION

Besides the SDEM, there are a number of different types of agreements that are unique to multi-architecture development. These agreements are necessary as a result of the differences of participating simulation architectures. When working in multi-architecture environments, adjudicating these differences may be very difficult, especially for services that are included in the architectures but specified and implemented in different ways. Significant problems can result if such agreements are not formalized and developers operate semi-independently of one another.

RECOMMENDED ACTION(S)

Differences in the way unique architectures operate should be documented. A set of agreements should be written to alleviate confusion as to which architecture is responsible for each aspect of the simulated environment throughout the development of the simulation environment.

Each architecture community tends to use different tools (in fact, some architectures, like TENA, identify certain tools in their core specifications), and agreements should be reached that identify exactly which tools will be used, where in the overall simulation environment those tools exist, what each tool does, how tool outputs are exchanged across architecture boundaries, and how different output formats can be interpreted outside the simulation architecture community that the tool was designed to support. Execution monitors, loggers, and execution manager applications may have to be extended to receive/interpret data feeds from tools that support different simulation architectures, and agreements should be established as to the functionality required and how data will be interchanged. Other types of agreements unique to
multi-architecture environments include scheduling of assets (e.g., personnel, facilities) across multiple architecture communities and identification of lead integrators and testers when such activities involve assets from several communities.

Some simulation architectures include services such as reference frame conversion and data marshalling, while others consider such concerns as important but outside the scope of the architecture. Procedures for initialization and synchronization (see Issue 2.5.3.1.2 [Initialization Sequencing and Synchronization]) as well as for save/restore can be quite different across architectures, potentially requiring some enhancements to the gateways or bridges connecting the simulation environments and the member applications within them. Agreements that reflect the resolution of these issues may require that some member applications adopt unfamiliar methods, which can impact success at integration time.

Recommended best practices include archiving previous versions of federation agreements for reuse/modification in the future and reusing existing templates whenever possible; on-going Live-Virtual-Constructive Architecture Roadmap Implementation efforts are addressing this [Morse et al., 2010].

2.4.2.2 Consolidation of “Establish Simulation Environment Agreements” Activities to Support Multi-architecture Events

**Multi-architecture-specific Activity Inputs**
- Requirements for gateway modifications
- Requirements for gateway configuration

**Multi-architecture-specific Tasks**
- Allocate supporting functions (data logging, execution management, etc.) to previously identified tools.
- Decide on data marshalling scheme.
- Decide on reference frames to be used.
- Establish initialization and synchronization procedures.

**Multi-architecture-specific Activity Outcomes**
- Within “Simulation environment agreements” (per the DSEEP):
  - Allocation of supporting functions to selected support tools
  - Reference frame conversion
  - Data marshalling
o Scheduling of assets
o Assignment of personnel responsibilities
o Initialization and synchronization procedures

2.4.3 Activity 4.3: Implement Member Application Designs

The purpose of this activity is to implement whatever modifications are necessary to the member applications to ensure that they can represent assigned objects and associated behaviors as described in the conceptual model, produce and exchange data with other member applications as defined by the SDEM, and abide by the established simulation environment agreements.

2.4.3.1 Issues

2.4.3.1.1 Issue: Nonstandard Algorithms

DESCRIPTION

In some cases, the interoperability protocol associated with simulation environment architectures standardizes on specific algorithms, e.g., the dead reckoning algorithms in the DIS protocol. If a simulation environment uses nonstandard algorithms, commercial off-the-shelf (COTS) and GOTS gateways developed to support the standard protocols will not be able to properly translate messages in situations that depend on these algorithms (e.g., DIS-HLA gateways generating heartbeat PDUs for the DIS side).

RECOMMENDED ACTION(S)

Two approaches are available to resolve this issue, both straightforward in concept. The first approach is to simply avoid the use of nonstandard algorithms, i.e., whenever a multi-architecture simulation environment uses an algorithm that has been standardized as part of one of the architectures (such as DIS dead reckoning algorithms), it uses the standard form (or one of the standard forms) of that algorithm. COTS and GOTS gateways and middleware should already be able to work with the standard algorithms [Valle et al., 2006]. It will be necessary to examine member applications from all of the linked architectures to ensure their conformance to this approach.

There may, however, be specialized circumstances when a nonstandard algorithm is essential to a particular simulation environment. For example, a simulation environment with a large number of simulated entities that move in a specific way that is not well predicted by any of the standard dead reckoning algorithms but can be well predicted by a custom dead reckoning algorithm, or special radio propagation effects modeling based on nonstandard transmitter antenna location values [Ross and Clark, 2005]. In such circumstances, the alternative approach
is to modify the gateways and middleware to support the nonstandard algorithm. Gateways and middleware might incorporate the nonstandard algorithms and apply them during the translation process [Lin and Woodyard, 1996]. In this approach, where the gateway or middleware needs to be modified, the ability (or lack thereof) to make these modifications (as a result of considerations such as source code availability) becomes a consideration in selecting the tool.

2.4.3.2 Consolidation of “Implement Member Application Designs” Activities to Support Multi-architecture Events

**MULTI-ARCHITECTURE-SPECIFIC ACTIVITY INPUTS**
- Available algorithms for implementation
- Selected common communications middleware

**MULTI-ARCHITECTURE-SPECIFIC TASKS**
- Select algorithms for implementation.

**MULTI-ARCHITECTURE-SPECIFIC ACTIVITY OUTCOMES**
- None beyond those called for in the DSEEP

2.4.4 Activity 4.4: Implement Simulation Environment Infrastructure

The purpose of this activity is to implement, configure, and initialize the infrastructure necessary to support the simulation environment and verify that it can support the execution and intercommunication of all member applications. This involves the implementation of the network design (e.g., WANs, LANs), the initialization and configuration of the network elements (e.g., routers, bridges), and the installation and configuration of supporting software on all computer systems. This also involves whatever facility preparation is necessary to support integration and test activities.

2.4.4.1 Issues

2.4.4.1.1 Issue: Live Entity TSPI Update Rates

**DESCRIPTION**

In TENA, live entities are not dead reckoned, so they require frequent updates of their Time Space Position Information (TSPI). A typical update rate is approximately 10 updates per second. Telemetry systems used to track live entity positions and generate TSPI updates may produce those updates at a faster rate. By comparison, dead reckoning can be used in DIS and HLA to reduce entity TSPI update rates and thereby reduce network traffic. When TENA is
linked with DIS or HLA, live-to-virtual network traffic as a result of TENA live entity TSPI updates can become excessive, resulting in undesirable network and data processing loads.

**RECOMMENDED ACTION(S)**

A lack of reliable velocity and acceleration data from instrumentation sources and the resulting gaps in TSPI data should be smoothed out to reduce visual jitter, particularly in training environments. As with DIS and HLA data, TENA data should be dead reckoned to support the simulation environment requirements and present an appropriate track picture to the simulation environment operator. While a basic dead-reckoning algorithm has been successfully used, it is suggested that the least-squares fit method, a parabolic filter, or Kalman filter methods be applied to generate smoother TSPI-based motion trajectories for live air platforms being represented in a virtual environment [Marsden et al., 2009].

**2.4.4.1.2 Issue: Network Design, Configuration, and Management**

**DESCRIPTION**

Multi-architecture simulation environments introduce network-related complexities as a result of the variety of network port, protocol, and performance requirements used. Designing and configuring the network to support the various data formats and transport mechanisms used in the multi-architecture simulation environment requires significant planning, integration, and testing over and above single-architecture simulation environments.

**RECOMMENDED ACTION(S)**

Defining and documenting all of the ports and protocols used in a multi-architecture simulation environment is critical to the success of the environment. The Joint Mission Environment Test Capability (JMETC) has defined a standard format for representing the necessary ports and protocols down to the individual machine at each participating site in a multi-architecture environment that includes

- Member application name
- Application protocol over the WAN (TENA, Link16, Variable Message Format [VMF], etc.)
- Network protocol (IP/TCP/UDP)
- Network port number(s)/range
- Direction (in/out/both)
- Destination IP (or multicast group address)
- Member application description
• Additional information

In addition to addressing the ports and protocols required to support multi-architecture simulation environments, additional performance-related decisions should be made with respect to how the member applications are partitioned across the WAN and/or LAN. Over-the-wire simulation protocols should be selected in order to maximize performance over a WAN. For instance, JMETC has mandated that TENA be the only simulation protocol used over the WAN because of its network efficiency. In multi-architecture environments where TENA applications participate with other architectures such as DIS and HLA, all simulation data is converted to TENA at each site before being transmitted over the WAN. One exception to this technique is the use of tactical message data that should be left in its native format to ensure correct transmission reception in a native format.

2.4.4.2 Consolidation of “Implement Simulation Environment Infrastructure” Activities to Support Multi-architecture Events

MULTI-ARCHITECTURE-SPECIFIC ACTIVITY INPUTS

• Requirements for gateway modifications
• Requirements for gateway configuration
• Architecture neutral data exchange model
• SDEM for each architecture
• Data exchange model mappings

MULTI-ARCHITECTURE-SPECIFIC TASKS

• Transform data exchange model mappings into gateway configurations.
• Modify and configure gateways to address prior requirements; in addition,
  o Implement TSPI smoothing.
• Within “Implement infrastructure design” (per the DSEEP)
  o Configure ports and protocols to implement the network architecture.

MULTI-ARCHITECTURE-SPECIFIC ACTIVITY OUTCOMES

• Modified and configured gateways
2.5 STEP 5: INTEGRATE AND TEST SIMULATION ENVIRONMENT

The purpose of this step of the DSEEP is to plan the execution of the simulation environment, establish all required interconnectivity between member applications, and test the simulation environment prior to execution.

2.5.1 Activity 5.1: Plan Execution

The main purpose of this activity is to fully describe the execution environment and develop an execution plan. For instance, performance requirements for individual member applications and for the larger simulation environment along with salient characteristics of host computers, operating systems, and networks that will be used in the simulation environment should all be documented at this time.

2.5.1.1 Issues

2.5.1.1.1 Issue: Multi-architecture Planning Considerations

**DESCRIPTION**

Multi-architecture development implies special consideration for execution planning beyond that normally required for a single-architecture simulation environment. Failure to account for the additional complications of a multi-architecture simulation environment will result in an unrealistic execution plan.

**RECOMMENDED ACTION(S)**

The execution plan should address both the technical and soft (i.e., non-technical) factors associated with the operation of a multi-architecture environment. Examples of technical factors include the development of startup and shutdown procedures that are compatible with all of the architectures in use and a method for reconciling the different mechanisms used by the different architectures to pass large amounts of data over the simulation infrastructure. Examples of soft factors include a procedure for training personnel to work with unfamiliar software and operational procedures and the scheduling of personnel and facilities across users of multiple architectures [e.g., Williams and Smith, 2007]. The simulation environment agreements often provide a good basis for identifying the considerations that need to be addressed in an execution plan for a multi-architecture simulation environment.

2.5.1.1.2 Issue: Distributed Simulation Environment Integration Testing

**DESCRIPTION**

The integration and testing of a multi-architecture simulation environment is likely to be a highly complex undertaking as a result of the diversity in experience, knowledge, and skills of
the simulation environment development team. The amount of work required to integrate and test a multi-architecture simulation environment may surpass normal resource planning estimates and result in the overloading of application developers during simulation environment integration test events.

**RECOMMENDED ACTION(S)**

A key for successfully performing integration testing is to reduce the complexity of the required testing. An integration testing strategy successfully used in single-architecture simulation environments is to plan for a spiral series of integration tests that increase in complexity. Simulation environment developers should first ensure that member applications operate satisfactorily in a single-architecture environment before attempting to operate across architecture boundaries. Integration testing should begin as soon as possible to allow time to troubleshoot unforeseen problems that are likely to occur during the integration of the multi-architecture simulation environment.

### 2.5.1.2 Consolidation of “Plan Execution” Activities to Support Multi-architecture Events

**MULTI-ARCHITECTURE-SPECIFIC ACTIVITY INPUTS**

- None beyond those called for in the DSEEP

**MULTI-ARCHITECTURE-SPECIFIC TASKS**

- Identify technical and soft factors associated with multi-architecture environments.
- Refine/augment the execution plan to include a spiral series of integration tests.

**MULTI-ARCHITECTURE-SPECIFIC ACTIVITY OUTCOMES**

- None beyond those called for in the DSEEP

### 2.5.2 Activity 5.2: Integrate Simulation Environment

The purpose of this activity is to bring all of the member applications into a unified operating environment. This requires that all hardware and software assets are properly installed and interconnected in a configuration that can support the SDEM and simulation environment agreements.

#### 2.5.2.1 Issues

No multi-architecture issues have been identified for this activity. However, the modified and configured gateways produced in Activity 4.4 (Implement Simulation Environment Infrastructure) are multi-architecture-specific inputs to this activity.
MULTI-ARCHITECTURE-SPECIFIC ACTIVITY INPUTS

- Modified and configured gateways

MULTI-ARCHITECTURE-SPECIFIC TASKS

- None beyond those called for in the DSEEP

MULTI-ARCHITECTURE-SPECIFIC ACTIVITY OUTCOMES

- None beyond those called for in the DSEEP

2.5.3 Activity 5.3: Test Simulation Environment

The purpose of this activity is to verify that all of the member applications can interoperate to the degree required to achieve core objectives. Distributed applications are tested at three levels:

- Member application testing
- Integration testing
- Interoperability testing

2.5.3.1 Issues

2.5.3.1.1 Issue: Complexities of Testing in a Multi-Architecture Environment

DESCRIPTION

During simulation environment testing, earlier agreements on resolutions of multi-architecture issues and any subsequent solutions should be properly tested. The multi-architecture solutions may include applications representing LVC assets as well as applications that support environment instrumentation, control, data management, and interoperability. Testing of the full multi-architecture simulation environment should be performed at the system level to ensure that these solutions meet performance and functionality requirements. As with single-architecture environments, inadequate testing can fail to discover simulation environments that have not properly met the requirements.

RECOMMENDED ACTION(S)

The additional complexities of a multi-architecture environment require more detailed and specific testing to ensure that the exchange of data across architectures is valid. It is critically important that all data be tested for correctness when it is published or consumed across architectural boundaries. Tools to support this interface verification could be found in a gateway
itself or in a data analysis tool that reads gateway and/or application log data and provides reports based on the data at key interface points.

While each architecture may have its own unique tools to test member applications and their interfaces, the communities should come together as early as possible to lay out a test strategy based on available tools. This strategy should start to take shape as early as DSEEP Step 3 during the design of the simulation environment.

2.5.3.1.2 Issue: Initialization Sequencing and Synchronization

DESCRIPTION

Initialization in a distributed simulation is a non-trivial and sequential process. For example, in HLA, the federation execution must be created before federates can join and objects can be registered. Explicit sequencing and synchronization of the initialization actions of member applications in a simulation environment is frequently needed to ensure that each member application is ready for the next action in the initialization process. In a multi-architecture simulation environment, these issues may be exacerbated. Initialization sequencing needs may be greater because, for example, mechanisms used to link the multiple architectures, such as gateways or middleware, may require the architectures’ executions to be started in a specific order. Such a sequencing constraint may be difficult to enforce. Moreover, explicit initialization synchronization may be more difficult in a multi-architecture simulation environment because the requisite synchronization mechanisms and messages (e.g., HLA synchronization services) are more likely to be architecture specific and less likely to be directly translatable across the architectures’ protocols than more generic operations such as object attribute updates.

RECOMMENDED ACTION(S)

Some architectures offer protocol services that can be used, with varying degrees of effort, for initialization sequencing and synchronization. For example, synchronization points were used for this purpose in one HLA simulation environment, although they did not function as initially expected, necessitating a carefully planned common multi-phase initialization process that included planned pauses to allow initialization operations to complete [Nielsen, 1999].

If the protocol services of one architecture are used to coordinate initialization across architectures in a multi-architecture simulation environment, the mechanism used to link the architectures (such as a gateway or middleware) should be configured or modified to translate those services from one architecture to the other. If synchronization services cannot be translated by the mechanism used to link the architectures, techniques outside of the simulation environment execution, such as manual control, may be used. Even if they can be translated, protocol services can only implement synchronization constraints that are known. To that end,
specific attention should be given to initialization when planning the simulation environment execution and testing the simulation environment. Any synchronization constraints or initialization sequence decisions should have been documented in the simulation environment agreements. These initialization and synchronization services and procedures should be thoroughly tested at this point.

Depending on the needs of the specific simulation environment, software tools designed to monitor and control simulation environment execution (e.g., the TENA tools Starship and StarGen [TENA Software Development Activity (SDA), 2008]) may be useful in sequencing and synchronizing initialization. Finally, to avoid these issues, member applications should be designed and implemented to be as independent of initialization sequence as possible [Nielsen, 1999].

### 2.5.3.1.3 Issue: Control of Multi-architecture Simulation Environment Execution

**Description**

No single set of capabilities exists to control the execution of a multi-architecture simulation environment. Some distributed simulation architectures have more extensive capabilities for controlling and coordinating the execution of a simulation environment (e.g., HLA Federation Management services) than others. Ensuring that a multi-architecture simulation environment can be executed as an integrated collective will require the development of specific operational procedures.

**Recommended Action(s)**

The control of a multi-architecture simulation environment will be manual by necessity. Specific user guidance will be required in order for the operators to monitor the health of the member applications and infrastructure as well as control the execution of the event scenario. The specific user guidance should be tested during the integration and testing of the multi-architecture simulation environment so that the record runs of the simulation environment can occur without incident.

### 2.5.3.1.4 Issue: Data Collection Configuration and Testing

**Description**

A central characteristic of multi-architecture simulation environments is the production of data of different formats. Various mechanisms of network protocol utilization, architecture rule sets, and locations of data collector(s), and the ability to correctly integrate architecture-unique data into a common event database for real-time or post-event analysis, may make an initial data collection plan impractical once the multi-architecture simulation environment has been integrated.
**RECOMMENDED ACTION(S)**

This is the last opportunity the simulation environment developers have to test their data collection plans prior to the multi-architecture event execution. During testing, data collection and integration procedures should be exercised, with special attention paid to the integration of data across architectural boundaries.

**2.5.3.2 Consolidation of “Test Simulation Environment” Activities to Support Multi-architecture Events**

**Multi-architecture-specific Activity Inputs**
- Architecture-specific procedure for initialization and synchronization

**Multi-architecture-specific Tasks**
- Test validity of data exchanges across architectural boundaries.
- Test initialization and synchronization procedures.
- Test data collection procedures and tools.

**Multi-architecture-specific Activity Outcomes**
- Within “Tested simulation environment” (per the DSEEP)
  - Assessment of the validity of cross-architecture data exchanges
  - Modified and configured gateways
  - Updated user guidance

**2.6 STEP 6: EXECUTE SIMULATION**

The purpose of this step is to execute the integrated set of member applications (i.e., the “simulation”) and to pre-process the resulting output data.

**2.6.1 Activity 6.1: Execute Simulation**

The purpose of this activity is to exercise all member applications of the simulation environment in a coordinated fashion over time to generate required outputs and thus achieve stated objectives. The simulation environment should have been tested successfully before this activity can begin.
2.6.1.1  Issues

2.6.1.1.1  Issue: Verifying Multi-architecture Execution

DESCRIPTION
Verifying the satisfaction of simulation environment requirements is difficult once the event execution has begun. No single set of tools is available to monitor event execution across a multi-architecture simulation environment.

RECOMMENDED ACTION(S)
Some architectures possess an inherent capability for fault detection/resolution/tolerance. However, different execution managers may provide an uneven picture of the state of the execution. A consistent and complete perception of execution state is critical for producing desired results. It is also necessary for results validation (i.e., for supporting required VV&A activities). The recommended action is to task different groups of people to monitor the event execution and report their observations in a timely manner to event execution management to enable corrective actions to be taken if required.

2.6.1.2  Consolidation of “Execute Simulation” Activities to Support Multi-architecture Events

MULTI-ARCHITECTURE-SPECIFIC ACTIVITY INPUTS
- None beyond those called for in the DSEEPP

MULTI-ARCHITECTURE-SPECIFIC TASKS
- Monitor execution.
- Report observations.

MULTI-ARCHITECTURE-SPECIFIC ACTIVITY OUTCOMES
- None beyond those called for in the DSEEPP

2.6.2  Activity 6.2: Prepare Simulation Environment Outputs
The purpose of this activity is to pre-process the output collected during the execution, in accordance with the specified requirements, prior to formal analysis of the data. This may involve the use of data reduction techniques to reduce the quantity of data to be analyzed and to transform the data to a particular format.

2.6.2.1  Issues
No multi-architecture issues have been identified for this activity.
2.7 **STEP 7: ANALYZE DATA AND EVALUATE RESULTS**

The purpose of this step of the DSEEP is to analyze and evaluate the data acquired during the execution of the simulation environment and to report the results back to the user/sponsor. This evaluation is necessary to ensure that the simulation environment fully satisfies the requirements of the user/sponsor.

2.7.1 **Activity 7.1: Analyze Data**

The purpose of this activity is to analyze the execution data. This data may be supplied using a range of different media (e.g., digital, video, audio), and appropriate tools and methods will be required for analyzing the data. These may be COTS or GOTS tools or specialized tools developed for a specific simulation environment. The analysis methods used will be specific to a particular simulation environment and can vary between simple observations (e.g., determining how many targets have been hit) and the use of complex algorithms (e.g., regression analysis or data mining).

2.7.1.1 **Issues**

No multi-architecture issues have been identified for this activity.

2.7.2 **Activity 7.2: Evaluate and Feedback Results**

There are two main evaluation tasks in this activity. In the first task, the derived results from the previous activity are evaluated to determine if all objectives have been met. This requires a retracing of execution results to the measurable set of requirements originally generated during conceptual analysis and refined in subsequent DSEEP steps. The second evaluation task in this activity is to assess all products generated in terms of their reuse potential within the domain or broader user community. Those products identified as having such reuse potential should be stored in an appropriate archive.

2.7.2.1 **Issues**

2.7.2.1.1 **Issue: Multi-architecture Simulation Environment Assessment**

**DESCRIPTION**

In evaluating the derived results of the simulation environment execution in a multi-architecture environment, the challenge is to determine if all objectives have been met. The use of a multi-architecture design introduces another factor to consider in the event that certain objectives were not met: Was the multi-architecture design itself the reason for a discrepancy between the objectives and outcomes?
RECOMMENDED ACTION(S)

Assessment of the performance of a multi-architecture simulation environment is heavily dependent upon the planning documentation generated during the multi-architecture simulation environment development process. Problems always occur during event execution because a simulation environment is a complex composition of computer hardware components, software libraries, and networking resources. Execution problems are likely to fall into one of three categories: (1) internal to a member application, (2) internal to a single-architecture implementation, and (3) across architecture implementation boundaries. A multi-architecture simulation environment staff requires insight and knowledge beyond that required to operate a single-architecture simulation environment to accurately troubleshoot problems. The normal tendency is to attribute any problem that may arise to the newest, and probably least understood, component of a simulation environment. The multi-architecture simulation environment staff can assess the impact of each problem once the true cause of a problem has been isolated. The multi-architecture simulation environment staff can provide feedback and make recommendations for follow-up activities and remedies to the user/sponsor to complete the simulation environment performance assessment. In addition, a decision needs to be made as to what should be archived from the recently executed simulation environment event and when it should be archived.

2.7.2.2 Consolidation of “Evaluate and Feedback Results” Activities to Support Multi-architecture Events

MULTI-ARCHITECTURE-SPECIFIC ACTIVITY INPUTS

- None beyond those called for in the DSEEP

MULTI-ARCHITECTURE-SPECIFIC TASKS

- Assess performance of multi-architecture simulation environment and categorize problems:
  - Internal to member application.
  - Internal to a single-architecture implementation.
  - Across architecture implementation boundaries.

MULTI-ARCHITECTURE-SPECIFIC ACTIVITY OUTCOMES

- None beyond those called for in the DSEEP
APPENDIX A. REFERENCES AND BIBLIOGRAPHY


## APPENDIX B. MAPPING OF ISSUES TO EXISTING ARCHITECTURES

This appendix is intended to provide tailoring of the guidance provided in the main document to specific architecture communities. More specifically, for each of three major simulation architectures, a mapping is provided to indicate the relevance of each Issue–Recommended Action pair to developers and users of that simulation architecture. The presence of a checkmark should emphasize the need for developers/users in each architecture community to consider the issue identified and the associated user guidance for how to address the issue when working in a multi-architecture simulation environment development activity.

<table>
<thead>
<tr>
<th>Step/Activity/Issue</th>
<th>DIS</th>
<th>HLA</th>
<th>TENA</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1. Step 1: Define LVC Environment Objectives</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1.1. Activity 1.1 Define Identify User/Sponsor Needs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1.2 Activity 1.2 Develop Objectives</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1.3 Activity 1.3 Conduct Initial Planning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1.3.1.1 Issue: Multi-architecture Initial Planning</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>2.1.3.1.2 Issue: Required LVC Expertise</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>2.2 Step 2: Perform Conceptual Analysis</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.2.1 Activity 2.1 Develop Scenario</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.2.2 Activity 2.2 Develop Conceptual Model</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.2.3 Activity 2.3 Develop Simulation Environment Requirements</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.2.3.1.1 Issue: Requirements for Multi-architecture Development</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>2.2.3.1.2 Issue: Member Application Requirement Incompatibility</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>2.3 Step 3: Design Simulation Environment</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.3.1 Activity 3.1 Select Member Applications</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.3.1.1.1 Issue: Member Selection Criteria for Multi-architecture Applications</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>2.3.1.1.2 Issue: Nonconforming Interfaces</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Step/Activity/Issue</td>
<td>DIS</td>
<td>HLA</td>
<td>TENA</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----</td>
<td>-----</td>
<td>------</td>
</tr>
<tr>
<td>2.3.2 Activity 3.2 Design Simulation Environment</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.3.2.1.1 Issue: Object State Update Contents</td>
<td>√</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.3.2.1.2 Issue: Object Ownership Management</td>
<td></td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>2.3.2.1.3 Issue: Time-managed Multi-architecture Applications</td>
<td></td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>2.3.2.1.4 Issue: Inconsistent Development and Execution Processes</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>2.3.2.1.5 Issue: Interest Management Capability Differences</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>2.3.2.1.6 Issue: Gateway Usage and Selection Decisions</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>2.3.2.1.7 Issue: Gateway Translation Paths</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>2.3.2.1.8 Issue: DIS Heartbeat Translation</td>
<td>√</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.3.2.1.9 Issue: Multi-architecture and Inter-architecture Performance</td>
<td></td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>2.3.2.1.10 Issue: Translating Non-ground-Truth Network Data</td>
<td>√</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>2.3.2.1.11 Issue: Object Identifier Uniqueness and Compatibility</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>2.3.2.1.12 Issue: Developing a Distributed Simulation Environment Composed of Classified and Unclassified Components</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>2.3.2.1.13 Issue: Multi-architecture Save and Restore</td>
<td>√</td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>2.3.2.1.14 Issue: Network Protocol Configuration</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>2.3.3 Activity 3.3 Design Member Applications</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.3.3.1.1 Issue: New Member Application Architecture</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>2.3.4 Activity 3.4: Prepare Detailed Plan</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.3.4.1.1 Issue: Cost and Schedule Estimating for Multi-architecture Development</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>2.3.4.1.2 Issue: Tools for Verifying Multi-architecture Data Applications</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>2.3.4.1.3 Issue: VV&amp;A for Multi-architecture Applications</td>
<td>√</td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>2.3.4.1.4 Issue: Multi-architecture Data Collection</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>2.3.4.1.5 Issue: Tool Incompatibility</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Step/Activity/Issue</td>
<td>DIS</td>
<td>HLA</td>
<td>TENA</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----</td>
<td>-----</td>
<td>------</td>
</tr>
<tr>
<td>2.4 Step 4: Develop Simulation Environment</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.4.1 Activity 4.1 Develop Simulation Data Exchange Model</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.4.1.1 Issue: Metamodel Incompatibilities</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>2.4.1.2 Issue: SDEM Content Incompatibilities</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>2.4.2 Activity 4.2 Establish Simulation Environment Agreements</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.4.2.1 Issue: Agreements to Address Multi-architecture Development</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>2.4.3 Activity 4.3: Implement Member Application Designs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.4.3.1 Issue: Nonstandard Algorithms</td>
<td>√</td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>2.4.4 Activity 4.4 Implement Simulation Environment Infrastructure</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.4.4.1 Issue: Live Entity TSPI Update Rates</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>2.4.4.2 Issue: Network Design, Configuration, and Management</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>2.5 Step 5: Integrate and Test Simulation Environment</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.5.1 Activity 5.1 Plan Execution</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.5.1.1 Issue: Multi-architecture Planning Considerations</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>2.5.1.2 Issue: Distributed Simulation Environment Integration Testing</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>2.5.2 Activity 5.2 Integrate Simulation Environment</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.5.3 Activity 5.3 Test Simulation Environment</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.5.3.1 Issue: Complexities of Testing in a Multi-architecture Environment</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>2.5.3.2 Issue: Initialization Sequencing and Synchronization</td>
<td>√</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>2.5.3.3 Issue: Control of Multi-architecture Simulation Environment Execution</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>2.5.3.4 Issue: Data Collection Configuration and Testing</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>2.6 Step 6: Execute Simulation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.6.1 Activity 6.1 Execute Simulation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Step/Activity/Issue</td>
<td>DIS</td>
<td>HLA</td>
<td>TENA</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-----</td>
<td>-----</td>
<td>------</td>
</tr>
<tr>
<td>2.6.1.1.1 Issue: Verifying Multi-architecture Execution</td>
<td>√</td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>2.6.2 Activity 6.2 Prepare Simulation Environment Outputs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.7 Step 7: Analyze Data and Evaluate Results</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.7.1 Activity 7.1 Analyze Data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.7.2 Activity 7.2 Evaluate and Feedback Results</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.7.2.1.1 Issue: Multi-architecture Simulation Environment Assessment</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
</tbody>
</table>
## APPENDIX C. ABBREVIATIONS AND ACRONYMS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>C2</td>
<td>Command and Control</td>
</tr>
<tr>
<td>C4I</td>
<td>Command, Control, Communications, Computers, and Intelligence</td>
</tr>
<tr>
<td>CASE</td>
<td>Computer-Aided Software Engineering</td>
</tr>
<tr>
<td>CDS</td>
<td>Cross-Domain Solution</td>
</tr>
<tr>
<td>COTS</td>
<td>Commercial Off-the-Shelf</td>
</tr>
<tr>
<td>CTIA</td>
<td>Common Training Instrumentation Architecture</td>
</tr>
<tr>
<td>DIF</td>
<td>Data Interchange Format</td>
</tr>
<tr>
<td>DIS</td>
<td>Distributed Interactive Simulation</td>
</tr>
<tr>
<td>DoD</td>
<td>Department of Defense</td>
</tr>
<tr>
<td>DSEEP</td>
<td>Distributed Simulation Engineering and Execution Process</td>
</tr>
<tr>
<td>EIA</td>
<td>Electronic Industries Alliance</td>
</tr>
<tr>
<td>FOM</td>
<td>Federation Object Model</td>
</tr>
<tr>
<td>GOTS</td>
<td>Government Off-the-Shelf</td>
</tr>
<tr>
<td>GWB</td>
<td>Gateway Builder</td>
</tr>
<tr>
<td>HITL</td>
<td>Human-in-the-Loop</td>
</tr>
<tr>
<td>HLA</td>
<td>High Level Architecture</td>
</tr>
<tr>
<td>IEC</td>
<td>International Electrotechnical Commission</td>
</tr>
<tr>
<td>IEEE</td>
<td>Institute of Electrical and Electronics Engineers</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization for Standardization</td>
</tr>
<tr>
<td>JCA</td>
<td>Joint Capability Area</td>
</tr>
<tr>
<td>JCOM</td>
<td>Joint Composable Object Model</td>
</tr>
<tr>
<td>JMETC</td>
<td>Joint Mission Environment Test Capability</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>LROM</td>
<td>Logical Range Object Model</td>
</tr>
<tr>
<td>LVC</td>
<td>Live-Virtual-Constructive</td>
</tr>
<tr>
<td>LVCAR</td>
<td>LVC Architecture Roadmap</td>
</tr>
<tr>
<td>M&amp;S</td>
<td>Modeling and Simulation</td>
</tr>
<tr>
<td>MSIS</td>
<td>Modeling and Simulation Information System</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MSRR</td>
<td>Model and Simulation Resource Repositories</td>
</tr>
<tr>
<td>OM</td>
<td>Object Model</td>
</tr>
<tr>
<td>OMT</td>
<td>Object Model Template</td>
</tr>
<tr>
<td>PDU</td>
<td>Protocol Data Unit</td>
</tr>
<tr>
<td>RPR FOM</td>
<td>Real-time Platform-level Reference Federation Object Model</td>
</tr>
<tr>
<td>RTI</td>
<td>Runtime Infrastructure</td>
</tr>
<tr>
<td>SDA</td>
<td>Software Development Activity</td>
</tr>
<tr>
<td>SDEM</td>
<td>Simulation Data Exchange Model</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>SIMC4I Interchange Module for Plans, Logistics, and Exercises</td>
</tr>
<tr>
<td>TACLANE</td>
<td>Tactical Local Area Network Encryption</td>
</tr>
<tr>
<td>TADIL</td>
<td>Tactical Automated Data Information Link</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>TENA</td>
<td>Test and Training Enabling Architecture</td>
</tr>
<tr>
<td>TRADOC</td>
<td>Training and Doctrine Command</td>
</tr>
<tr>
<td>TSPI</td>
<td>Time Space Position Information</td>
</tr>
<tr>
<td>UAV</td>
<td>Unmanned Aerial Vehicle</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>UDP/IP</td>
<td>UDP Packets over IP</td>
</tr>
<tr>
<td>UJTL</td>
<td>Universal Joint Task List</td>
</tr>
<tr>
<td>V&amp;V</td>
<td>Verification and Validation</td>
</tr>
<tr>
<td>VMF</td>
<td>Variable Message Format</td>
</tr>
<tr>
<td>VV&amp;A</td>
<td>Verification, Validation, and Accreditation</td>
</tr>
<tr>
<td>WAN</td>
<td>Wide Area Network</td>
</tr>
</tbody>
</table>