Deriving Feasible Deployment Alternatives for Parallel and Distributed Simulation Systems

TURGAY ÇELİK, Hacettepe University
BEDİR TEKİNERDOGAN, Bilkent University
KAYHAN M. İMRE, Hacettepe University

Parallel and distributed simulations (PADS) realize the distributed execution of a simulation system over multiple physical resources. To realize the execution of PADS, different simulation infrastructures such as HLA, DIS and TENA have been defined. Recently, the Distributed Simulation Engineering and Execution Process (DSEEP) that supports the mapping of the simulations on the infrastructures has been defined. An important recommended task in DSEEP is the evaluation of the performance of the simulation systems at the design phase. In general, the performance of a simulation is largely influenced by the allocation of member applications to the resources. Usually, the deployment of the applications to the resources can be done in many different ways. DSEEP does not provide a concrete approach for evaluating the deployment alternatives. Moreover, current approaches that can be used for realizing various DSEEP activities do not yet provide adequate support for this purpose. We provide a concrete approach for deriving feasible deployment alternatives based on the simulation system and the available resources. In the approach, first the simulation components and the resources are designed. The design is used to define alternative execution configurations, and based on the design and the execution configuration; a feasible deployment alternative can be algorithmically derived. Tool support is developed for the simulation design, the execution configuration definition and the automatic generation of feasible deployment alternatives. The approach has been applied within a large-scale industrial case study for simulating Electronic Warfare systems.

Categories and Subject Descriptors: C.2.4 [**Computer-Communications Networks**]: Distributed Systems—*Distributed applications*; D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Computer-aided software engineering (CASE)*; D.2.8 [**Software Engineering**]: Metrics—*Performance measures*; D.2.10 [**Software Engineering**]: Design—*Methodologies*; D.2.11 [**Software Engineering**]: Software Architectures—*Domain-specific architectures*; G.1.6 [**Numerical Analysis**]: Optimization—*Constrained optimization*; I.6.5 [**Simulation and Modeling**]: Model Development—*Modeling methodologies*; I.6.7 [**Simulation and Modeling**]: Simulation Support Systems—*Environments*; I.6.8 [**Simulation and Modeling**]: Types of Simulation—*Distributed, parallel*

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: Parallel and distributed simulations, high-level architecture, DSEEP, software architecture, model transformations, metamodeling

ACM Reference Format:
Çelik, T., Tekinerdogan, B., and İmre, K. M. 2013. Deriving feasible deployment alternatives for parallel and distributed simulation systems. *ACM Trans. Model. Comput. Simul.* 23, 3, Article 18 (July 2013), 24 pages.
DOI: http://dx.doi.org/10.1145/2499913.2499917

18

1. INTRODUCTION

Parallel and distributed simulations (PADS) [Fujimoto 1999] realize the distributed execution of a simulation system over multiple physical resources. Developing PADS

Author's address: T. Çelik; email: turgaycelik@gmail.com.

DOI: http://dx.doi.org/10.1145/2499913.2499917

is not trivial and requires realizing the concerns such as declaration management, data exchange, time management, discovery mechanisms and data distribution management. To reduce the effort for developing PADS, common standard infrastructures have been introduced including Distributed Interactive Simulation (DIS) [IEEE 1998], Discrete Event System Specification (DEVS) [Zeigler 2003], High Level Architecture (HLA) [IEEE 2010a; Kuhl et al. 1999], and Test and Training Enabling Architecture (TENA) [Noseworthy 2008]. Among these, HLA is an important IEEE and NATO standard that defines a common architecture for supporting both parallel and distributed simulation systems [Kuhl et al. 1999; Perumalla and Fujimoto 2003].

A simulation system usually consists of multiple applications. For example, for realizing a traffic simulation as a PADS, the system can be decomposed in a number of applications that simulate vehicles, pedestrians, traffic lights, environment, drivers etc. The motivation for the decomposition of the system into multiple applications can be based on quality concerns such as performance, reusability and interoperability [Fujimoto 1999]. Usually, the decomposition and deployment of the applications to the physical resources can be carried out in many different ways. Further, the performance of the simulation system is largely influenced by the allocation of these applications to the available physical resources.

To support the development of HLA compliant simulation systems, the IEEE Recommended Practice for High Level Architecture Federation Development and Execution Process (FEDEP) [IEEE 2003] has been defined. Based on FEDEP, recently the IEEE standard DSEEP (Distributed Simulation Engineering and Execution Process) has been defined [IEEE 2010d]. DSEEP consists of a set of steps and activities that include recommended tasks for developing simulation systems.

DSEEP itself is an abstract process and deliberately does not mandate a particular realization of the process. An important recommended task in DSEEP is the evaluation of the performance of the simulation systems at the design phase. Various approaches can be identified in the literature that can be used to support the DSEEP process including conceptual model development approaches [Karagöz and Demirörs 2007; SISO 2006], tool support [Parr 2003; VT MAK 2010a] and simulation design approaches [Çetinkaya and Oguztüzün 2006; Topçu et al. 2008]. Yet, despite the benefit of these approaches, no adequate and explicit support for selecting and evaluating the deployment alternatives have been provided so far. As such, the evaluation of the design and the performance estimation is usually either deferred to the development phase or performed based on expert judgment in the design phase. Deferring these design tasks to the development phase, however, might lead to non-feasible implementations that may require unnecessary iteration of the design and the related project lifecycle artifacts such as detailed design, implementation, test artifacts, documentation, etc. As such, the search for a feasible solution can easily lead to delays in the project schedule, and due to the unnecessary rework of the lifecycle artifacts the cost can increase dramatically. Expert judgment can help to support this process, but finding experts that have both a broad and specialized knowledge on the corresponding domains is not easy. Further, expert judgments can be of help for small to medium systems but can be limited if the system gets too complex. Obviously, a more systematic and formal approach is required to guide the search for a feasible deployment alternative.

In this article, we provide a concrete approach for deriving a feasible deployment alternative based on the simulation system and the available physical resources. In the approach, first the simulation components and the physical resources are designed. The design is used to define alternative simulation execution configurations that refine the number and parameters of the corresponding design elements. Based on the simulation design and the execution configuration, a feasible deployment alternative can be algorithmically derived. The presented approach is supported by corresponding

tools that support the simulation design, the execution configuration definition and the automatic generation of feasible deployment alternatives. The approach has been validated within a large-scale industrial case study for simulating Electronic Warfare systems.

Concretely, the contributions of the article can be defined as follows.

— We provide a systematic approach that supports both the evaluation of the simulation system design with respect to physical resources, and the automatic generation of a feasible deployment alternative in the early phases of the system design. The overall approach integrates the well-known Capacitated Task Assignment Problem (CTAP) solving techniques with the Parallel and Distributed Simulation (PADS) system design process. As such, the approach can be used for impact analysis at the design level including, the impact analysis of adding new simulations modules to the system, analyzing suitability of the selected physical resources for the given simulation design, and impact analysis of the change in publish-subscribe relations.
— The approach is integrated in the DSEEP process and provides an implementation of the corresponding recommended DSEEP tasks. The DSEEP has recommended tasks for evaluating alternative design options and estimating the simulation performance in design phase but deliberately does not provide a detailed process and implementation for the indicated tasks. Our approach provides a detailed process to ease the realization of these important tasks.
— A toolset that supports the evaluation steps of the approach has been designed and developed. Of particular interest hereby is the automatic analysis and generation of a feasible deployment alternative. The toolset is based on a set of metamodels that we have defined for automatically deriving the simulation design.

The remainder of the article is organized as follows. In Section 2, we provide the background on PADS and DSEEP. Section 3 defines the case study that will be used in subsequent sections. Section 4 describes the problem statement. Section 5 presents the approach for evaluating alternative design options with the adopted models and algorithmic solutions for the approach. Section 6 briefly presents the tools that support the approach. Section 7 provides the discussion. Section 8 describes the related work and finally we conclude the article in Section 9. The list of acronyms is provided in Online Appendix A.

2. BACKGROUND AND CONTEXT

In this section, we describe the background for understanding and supporting the approach that we present in this article. In Section 2.1, we present the common reference architecture for PADS, followed by a discussion in Section 2.2 on DSEEP.

2.1. PADS Reference Architecture

It appears that the current PADS architectures share the similar concepts. Based on a domain analysis to simulation architectures such as DIS [IEEE 1998], HLA [IEEE 2010a; Kuhl et al. 1999], and TENA [Noseworthy 2008], we could derive the reference architecture for PADS, which is shown in Figure 1. A typical simulation system is deployed on a number of Simulation Nodes. Each Simulation Node includes one or more *Members* that are processes that together form the simulation execution. Each member includes a number of Simulation Module Instances and Local Infrastructure Component. Simulation Module Instances represent objects for simulating entities or events in the simulation. Local Infrastructure Component enables bi-directional interaction between members for data exchange and collaborative execution of the simulation.

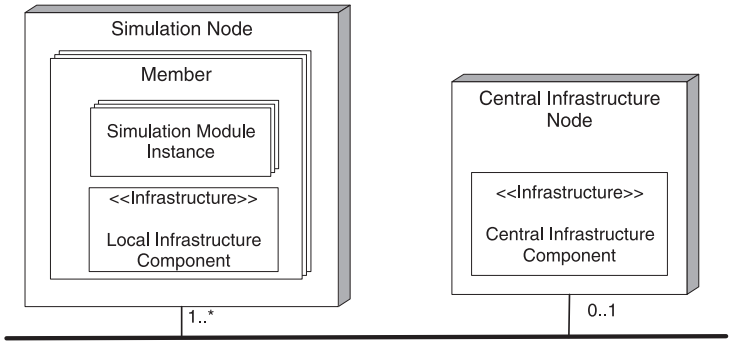

Fig. 1. Reference architecture for parallel and distributed simulations.

The simulation may also include an optional Central Infrastructure Node that contains a Central Infrastructure Component, which is responsible for managing the simulation lifecycle, timing and synchronization concerns, and discovery concerns. In case, this Central Infrastructure Component is missing, the services need to be supported by the Local Infrastructure Components. As such both the Local Infrastructure Component and Central Infrastructure Component provide similar services. In Figure 1, this is indicated through the stereotype <<Infrastructure>>.

HLA defines a specification of the architecture that largely conforms to the reference architecture in Figure 1. In HLA, *Members* called *Federates* connect to Runtime Infrastructure (RTI) Components (e.g., *Local RTI Component* (LRC)) that correspond to the Local Infrastructure Component in Figure 1. Federates in HLA define the simulation logic that correspond to the Simulation Module Instances of this reference architecture. Further, as a convention, Central Infrastructure Component is provided in major RTI implementations, in which this is named as Central RTI Component (CRC).

The CRC and LRC implementations together provide services for federation management, declaration management, object management, ownership management, time management, and data distribution management [IEEE 2010b]. DIS and TENA are largely similar to HLA, but these do not include a Central Infrastructure Component. Further, the provided services are different for the three architectures. For example, time management service is provided by HLA but is not explicitly included in DIS and TENA.

The common interaction model that is adopted in the three simulation architectures conforms to the Publish/Subscribe pattern [Eugster et al. 2003]. In the Publish/Subscribe pattern, the producer and consumer applications (members) are decoupled. This increases the reusability and interoperability, which are key concerns in simulation systems. The Publish/Subscribe interaction is realized by the <<Infrastructure>> components in the reference architecture in Figure 1. Members in the simulation execution can publish and subscribe data exchange model elements through the services provided by the <<Infrastructure>> components. The adopted data exchange models differ in the three architectures. In DIS, the data exchange model is fixed through protocol data units (PDU). The HLA standard defines the Object *Model Template* (OMT) that can be used to define different data exchange models [IEEE 2010c] which are called *Federate Object Model* (FOM) and Simulation Object *Model* (SOM). TENA specification provides a predefined object model called *Logical Range Object Model* that can be extended and customized [Noseworthy 2008].

2.2. DSEEP

As stated before, DSEEP is an abstract process and deliberately does not mandate a particular realization of the process. The DSEEP process consists of the following steps [IEEE 2010d].

(1) *Define Simulation Environment Objectives.* The objectives for the simulation environment are defined by stakeholders.
(2) *Perform Conceptual Analysis.* Scenarios and conceptual model are developed, and subsequently the simulation environment requirements and functionality is defined.
(3) *Design Simulation Environment.* Existing reusable members are identified, new members are defined and functionalities are allocated to members. Further a development plan is defined.
(4) *Develop Simulation Environment.* The data exchange model is developed, newly identified members are developed and if needed reusable members are customized.
(5) *Integrate and Test Simulation Environment.* The system components are integrated and tested.
(6) *Execute Simulation.* The simulation system is executed and the resulting output data is preprocessed for the next step.
(7) *Analyze Data and Evaluate Results.* The output data from the execution is analyzed and evaluated with respect to the objectives.

Different efforts have been provided to realize these steps. In general, the proposed solutions focus on a particular number of steps. For example, Base Object Model (BOM) [SISO 2006] and the model proposed in Karagöz and Demirörs [2007] focus on development of conceptual models that is needed for step 2 in DSEEP. The VR Forces [VT MAK 2010b] and VR Vantage XR [VT MAK 2010c] tools support step 4, MAK Data Logger [VT MAK 2010a] and Pitch Commander [Pitch Technologies 2009] support step 5. In this article, our focus is on step 3, which is explained in the following sections.

3. CASE STUDY - A LARGE SCALE ELECTRONIC WARFARE SIMULATION

In this section we provide a case study describing a large scale Electronic Warfare (EW) [Adamy 2001] simulation. EW simulations are very important in the defense industry to generate virtual exercises that are hard, expensive, and dangerous to perform with real exercises [Adamy 2006]. The case study is used throughout the article to clarify the problem statement and to illustrate our approach later on.

The logical view for the case study is given in Figure 2. The system includes platform simulators that interact with a tactical environment simulator. A platform simulator is a system that simulates the physical and behavioral properties of a platform. Each platform simulator has simulation models like motion model, 3D visualization, and specific equipments such as radars and communication devices. In our case study, there are four different types of platform simulators including ship, flight, submarine, and tank simulators. In the figure, no particular number for the platforms simulators is given, but '*' is used to indicate zero or more simulators. The specific number of simulators will be defined by the concrete scenario, which will be explained in the next subsection. Tactical environment simulator contains models for EW systems (e.g., radar, missiles, etc.) and environmental conditions (e.g., weather).

For the given case study, we can now apply the DSEEP to derive a simulation system. As stated before, the first step of DSEEP concerns the definition of simulation environment objectives. In this particular case, this includes for example, defining ship simulator, flight simulator, submarine simulator, tank simulator and tactical environment simulator. After the definition of the simulation environment objectives,

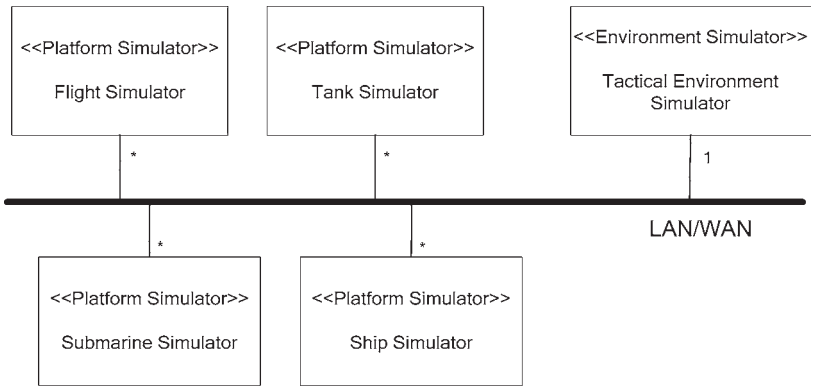

Fig. 2. Logical view of case study.

Table I. A Sample Scenario for the Case Study

Simulation Participant	Number	Simulation Participant Entities	# Total Entities
Ship Simulator	10	1 ship model, 8 EW systems, 5 Surface to Air EO missiles, 10 Surface to Surface RF missiles.	240
Submarine Simulator	8	1 submarine model, 10 EW systems, 10 Surface to Surface RF missiles.	168
Flight Simulator	12	1 flight model, 3 EW systems, 5 Air to Air EO missiles, 2 Air to Surface RF missiles.	132
Tank Simulator	15	1 tank model, 4 EW systems.	75
Tactical Environment Simulator	1	40 airplanes, 20 ships, 10 submarines, 100 tanks, 10 Command Control Systems, 500 EW systems such as Radars, ESM systems, EA systems, 300 Missiles, RF Propagation Model.	981

the next DSEEP step is performing the conceptual analysis of the simulation system. An important activity of the conceptual analysis is the development of scenarios. A scenario includes the types and numbers of major simulation entities according to the earlier defined simulation objectives. Table I shows a sample scenario for the case study.

The "Simulation Participant" column of the table indicates the simulators that together form the simulation of the system. The "Number" column defines the number of simulation participants of the simulator type in the given scenario. For example, in the scenario as defined in Table I there are 10 Ship Simulators. The column 'Simulation Participant Entities' shows the entity models that are required for realizing the particular simulators. Each simulator has its own specific type of entities. For example, in the given scenario the ship simulator has 1 ship model, 8 EW systems, 5 Surface to Air EO missiles, and 10 Surface to Surface RF missiles. The last column "**#** Total Entities" defines the total number of entities in the particular simulator. This number is calculated by multiplying the number of participants with the number of Simulation Participant Entities. For example, **#** total entities for 10 ship simulators is 10 x $(1 + 8 + 5 + 10) = 240$. As it can be observed for a given scenario, the total number of the required simulation entities might be quite large. For the given scenario, the total number of entities for all the simulators together is $240 + 168 + 132 + 75 + 981 = 1596$.

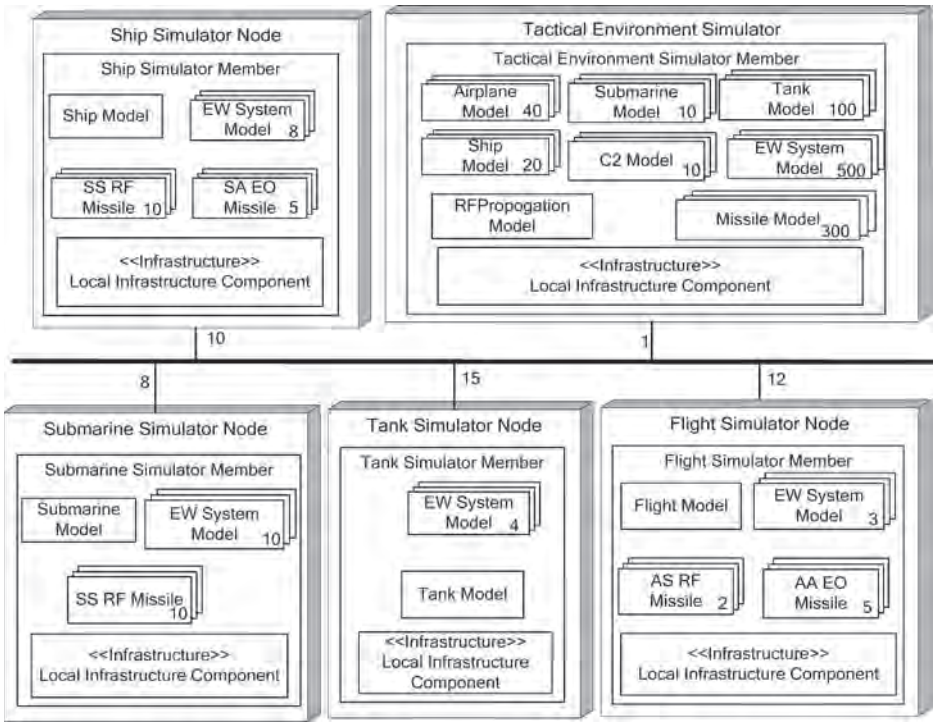

Fig. 3. Deployment alternative for scenario of Table I - One tactical environment simulator and distributed EW system models.

4. PROBLEM STATEMENT

After the required scenario is defined and the conceptual analysis is completed, we can start designing the simulation system in step 3 of DSEEP. Using the reference architecture as shown in Figure 1 and the given scenario in Table I, we can derive a deployment alternative. A deployment alternative defines the mapping of the simulation entity models in the scenario to the nodes and members. Here we assume that each node has exactly one member that acts as a container for all simulation module instances. The reason for adopting one member per node is to reduce the overhead of interprocess communications, which is important in case we have to deal with a large number of simulation module instances [Lees et al. 2007].

An example deployment alternative is shown in Figure 3. In this deployment alternative, each platform simulator and the tactical environment simulator is deployed on a separate node. Each platform simulator node models its own EW systems and the other simulation entities. This alternative actually follows the conceptual separation of concerns in which each simulator is logically defined as a node and defines its own responsibilities. Further the centralized tactical environment simulator decreases the communication overhead among the tactical environment entities. Although this alternative is easy to understand because of the logical separation of concerns, it can result in inefficiencies with respect to additional communication overheads. This is because the EW System models may need to interact very frequently with each other to model the coordination and interaction of electronic warfare systems.

A second example deployment alternative is shown in Figure 4. Here, each platform simulator is deployed on a separate node, but the tactical environment simulator is

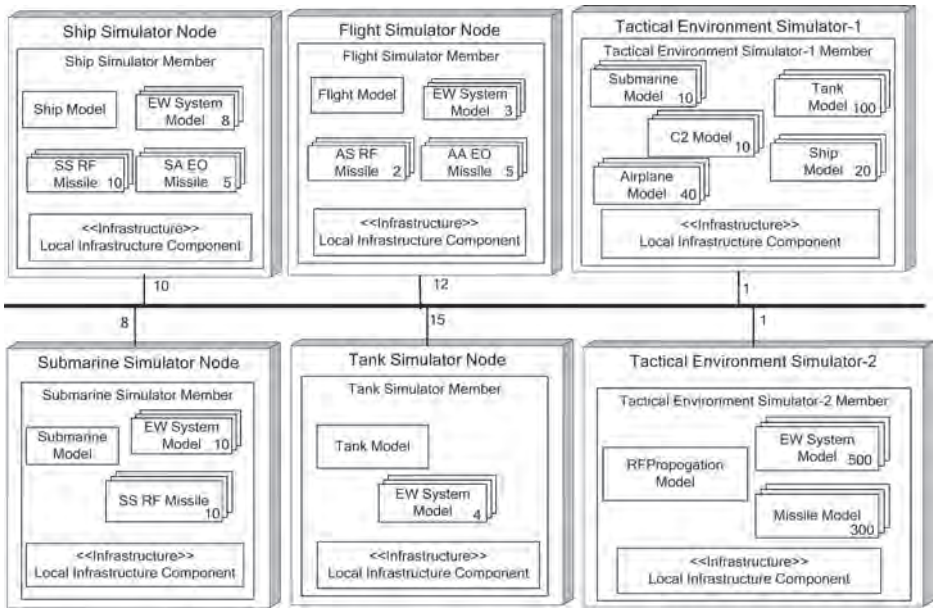

Fig. 4. Deployment alternative for scenario of Table I - Two tactical environment simulators and distributed EW system models.

divided and deployed on two nodes. The platform simulator nodes follow the logical separation of concerns and as such are easy to understand. The separation of tactical environment simulators over two nodes enhances the computation power, but on the other hand increases the communication overhead between the tactical environment simulator nodes.

We can also derive many other different deployment alternatives. These deployment alternatives may differ with respect to the number of deployment nodes, the mapping of simulators to the nodes, the distribution of the simulation entities over the nodes etc. Obviously, the number of deployment alternatives is very large and each deployment alternative will perform differently with respect to different quality considerations such as logical separation for understandability, optimizing communication overhead, enhancing utilization of physical resources, etc.

According to DSEEP, it is important to identify deployment alternatives and evaluate their performance. Concretely, DSEEP defines the following two recommended tasks within the step "Design Simulation Environment" [IEEE 2010d].

— Evaluate alternative design options, and identify the simulation environment design that best addresses stated requirements.
— Estimate simulation environment performance, and determine if actions are necessary to meet performance requirements

Since DSEEP is deliberately defined as an abstract process, it does not define how to realize these tasks. Moreover, currently there is no adequate tool support yet to support these tasks. As stated before, the evaluation of the design and the performance estimation is either deferred to the development phase or performed based on expert judgment in the design phase. However, deferring these design tasks to the development phase might lead to nonfeasible implementations that may require unnecessary iteration of the design and the related project lifecycle artifacts such as detailed design, implementation, test artifacts, documentation, etc. On its turn, this will lead to delays

and higher cost in the project. On the other hand, expert judgments are also limited if the system gets too complex. In the following section, we will provide a systematic approach to guide the search for a feasible deployment alternative.

5. APPROACH FOR DERIVING FEASIBLE DEPLOYMENT ALTERNATIVES

In this section, we provide a concrete approach for deriving and evaluating feasible deployment alternatives. The approach is represented as an activity diagram as shown in Figure 5.

We assume that the first two DSEEP steps shown in Figure 5 are performed using the approaches as defined in the literature (e.g., see SISO [2006]). Therefore the activities in the first two steps are shown in grey scale. As stated before, our focus is on step 3 of DSEEP. Further, it is assumed that the system is not developed yet, and the code is not available. The final deployment model is actually built up over several iterations of the DSEEP steps 3, 4, and 5. Hereby, the initial deployment model is prototyped and tested in steps 4–5, and the results are fed back into the concept definition until the best alternative is identified and defined to the proper level for implementation. The iterative nature of the DSEEP is shown through back arrows between the DSEEP steps.

In the following sections, we will explain the concrete activities that we have defined to realize our approach within the context of DSEEP step-3.

5.1. Design Simulation Data Exchange Model

The activity *Design Simulation Data Exchange Model* defines an initial design of the Simulation Data Exchange Model (SDEM) that is necessary to enable data exchange among simulation modules. Actually, in DSEEP the development of the simulation data exchange model (SDEM) is defined in step 4. However, since SDEM elements are required for designing the pub-sub relations among simulation modules, the SDEM design is started in step 3 in our process. Because the basic elements that are required for the design of SDEM are defined in step 2 and sub-steps of step 3, the SDEM design can be started earlier in step 3 and finalized in step 4 without problems since DSEEP is also an iterative process.

For the SDEM we could use existing standard models such as Real-time Platform Reference Federation Object Model (RPR-FOM) [SISO 1999], extend standard models or even develop from scratch as stated in DSEEP. In this context, the HLA OMT [IEEE 2010c] standard defines a standard metamodel for deriving SDEMs. To represent simulation entities HLA OMT specification defines the three key elements of Object Classes, Interactions and DataTypes. ObjectClasses are used to define the simulation entities. In our case, Object Classes are used to represent, for example, Surface Vessel, AirPlatform, Tank, Radar etc. Interactions are used to represent the messaging semantics among simulation participants. For example, messages like MunitionDetonation, SystemStatusChange, MissileLaunch are example of interactions in our EW simulation domain. Finally, DataTypes represent types of the attributes of ObjectClasses and parameters of Interactions. For example, the ObjectClass Platform could have an attribute *position* of type Position3D, and the Interaction SystemStatusChange can have a parameter *systemState* of enumeration type with the values *on, off.*

We have adopted HLA OMT standard and enhanced it to define our simulation data exchange metamodel. The resulting Simulation Data Exchange Metamodel corresponds to the HLA OMT artifacts with an addition of the average size attribute to array datatype. Later on, this is necessary to allow the estimation of the size of an exchanged object during feasible deployment analysis at the design phase. Figure 6 shows modified version of arrayDataType element in HLA OMT schema [IEEE 2010c].

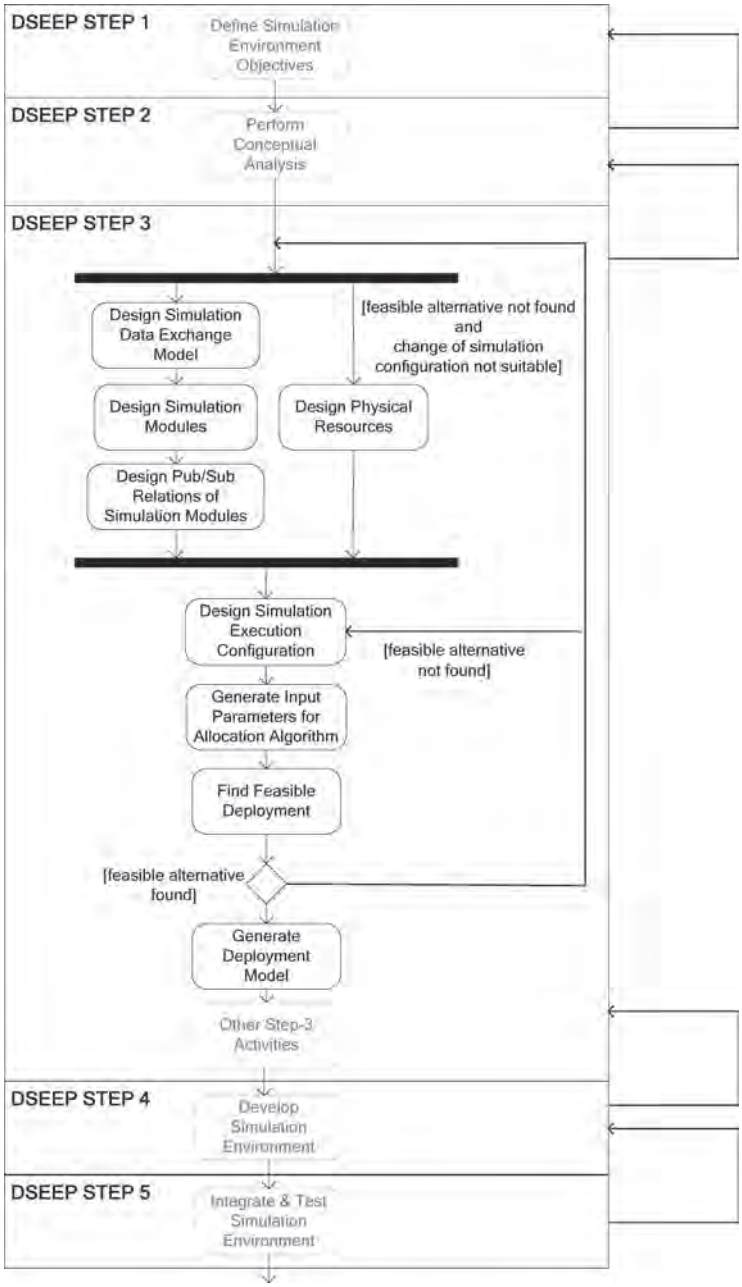

Fig. 5. Activity flow of alternative design evaluation and deriving feasible deployment.

5.2. Design Simulation Modules

The activity *Design Simulation Modules* includes the definition of simulation entities of the simulation participants. Simulation modules are artifacts of a simulation system that are responsible for modeling a part of the system. In the given example scenario as given in Table I simulation modules are, for example, Ship, Submarine,

```
...
<xs:complexType name="arrayDataType">
    <xs:sequence>
      <xs:element name="name" type="IdentifierType" />
      ...
      <xs:element name="averageSize" type="xs:integer" />
      <xs:element name="semantics" type="String" minOccurs="0" />
      <xs:any namespace="##other" minOccurs="0" />
    </xs:sequence>
    <xs:attributeGroup ref="commonAttributes" />
  </xs:complexType>
...
```

Fig. 6. Extension of HLA OMT schema for the approach.

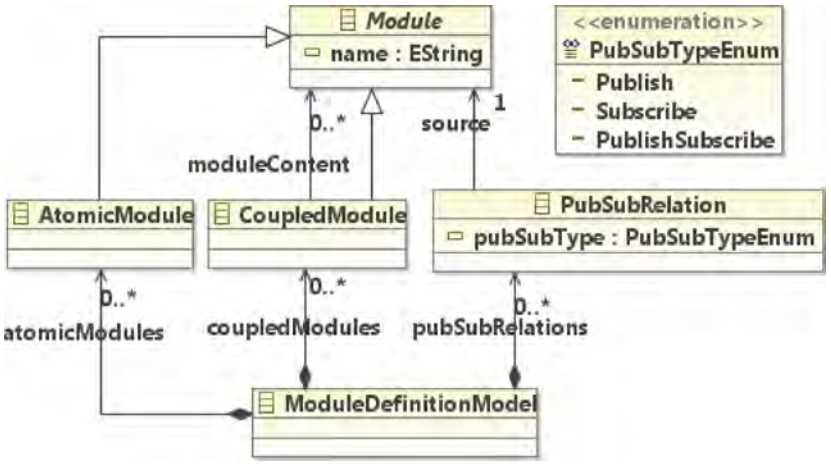

Fig. 7. Metamodel for module and publish subscribe definitions.

Tank, SA EO Missiles, SS RF Missiles, EW Models, etc. In addition to defining the simulation modules, the activity also aims to define the composition relations among the simulation entities. For example, a Radar Module may contain a receiver module and a transmitter module. We have defined a common metamodel (Figure 7) that can be used to define both the simulation modules and the composition relations. Similar to the Discrete Event Virtual Simulation (DEVS) [Zeigler 2003] model the metamodel defines atomic and coupled models that form the simulation systems.

The *ModuleDefinitionModel* is the root class of the metamodel that represents a module definition model that defines modules and their Publish/Subscribe relations. We will elaborate on the Publish/Subscribe relations in the next subsection. The Module is the abstract base class for simulation module definitions. The Module has a name attribute that identifies the module. An *AtomicModule* represents elementary simulation models while *CoupledModule* represents more complex simulation models that may contain other atomic or coupled modules. This containment relation is shown as *moduleContent* reference in the metamodel.

5.3. Design Publish/Subscribe Relations of Simulation Modules

The activity *Design Publish/Subscribe Relations of Simulation Modules* defines the publish/subscribe relations of simulation modules based on the simulation data

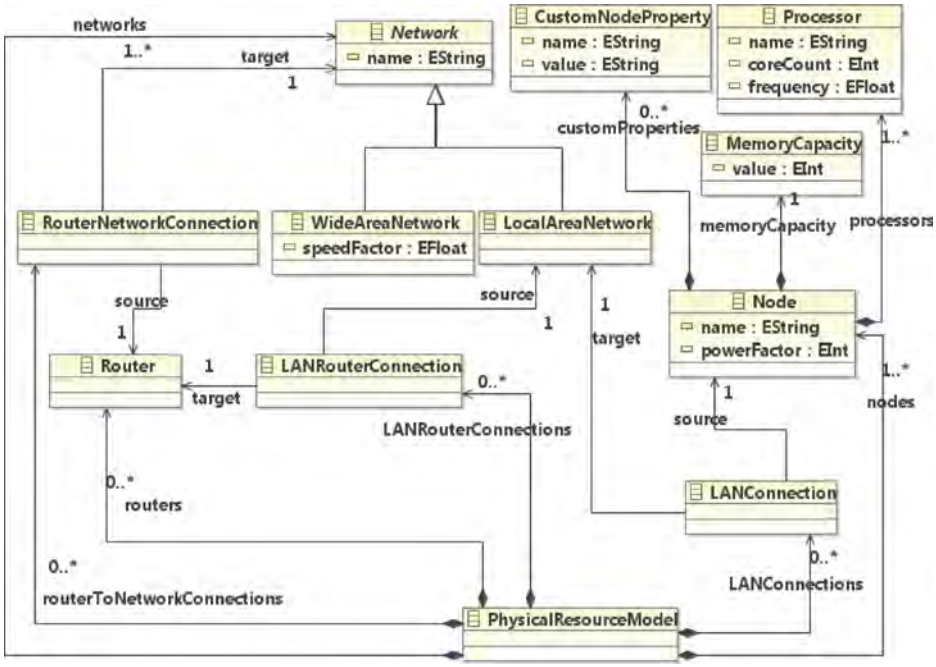

Fig. 8. Physical resource metamodel.

exchange model. For example, Radar Module can publish RadarBeam object while RFPropagation Module subscribes to RadarBeam object.

PubSubRelation class in the metamodel of Figure 7 defines a publish/subscribe relation between a simulation module *Module* and a SDEM element *ObjectModelElement*. *ObjectModelElement* class is defined in Simulation Data Exchange Metamodel and it is the base class for SDEM elements as described in the previous section. *PubSubRelation* class has further a pubSubType attribute which defines the type of the relation. The possible types are defined by the values of the enumeration class *PubSubTypeEnum* that defines the values *Publish*, *Subscribe*, and *PublishSubscribe*.

5.4. Design Physical Resources

Parallel to these three activities, the activity *Design Physical Resources* defines the available nodes together with their processing power and memory capacity, as well as the network connections among the nodes. For example, one may decide to adopt 25 nodes on which the simulation participants need to be deployed. Further, it could be decided that each node has a memory capacity of 12280 MB and contains two processing units with four cores at the frequency of 2.3 MHz. Equally, the nodes could also have different memory capacity and processing power.

The Physical Resource Metamodel given in Figure 8 can be used to represent the artifacts for modeling the available physical resources. *PhysicalResourceModel* is the root class of the metamodel that defines a physical resource model. There can be one or more Nodes in a physical resource model, which represents computation resources. Each node has a *name* attribute that identifies the node. The *powerFactor* attribute defines the processing power of the node relative to other nodes. A node can have one or more processors, one or more custom node properties, and memory capacity. *Processor* defines properties of a processing unit using the attributes *name, frequency*, and

coreCount. The attribute *name* is the symbolic name of the processor like "Intel Core I7". The attribute *coreCount* defines the number of cores that the processor has. The attribute *frequency* defines the frequency of the processor in MHz. *MemoryCapacity* has a *value* attribute that represents the memory capacity of the node in terms of megabytes. *CustomNodeProperty* can be used to define additional properties for the node. The properties are defined as name-value pairs. For example, one may decide to include a specific property *diskCapacity* with value 340 Gb.

There can be one or more networks in a physical resource model. The *Network* class is the abstract base class for *LocalAreaNetwork* (LAN) and *WideAreaNetwork* (WAN) classes. The *name* attribute of the *Network* class is the symbolic name of the network. *WideAreaNetwork* class has *speedFactor* attribute that defines the speed of the network in comparison with a LAN. *LANConnection* represents the connection of a node to a LAN. *Router* represents routers for connecting networks with each other. The *name* attribute of the *Router* class is the symbolic name of the router. *LANRouterConnection* class represents connection of a LAN to a router while the *RouterNetworkConnection* class represents connection of a router to a network.

5.5. Design Simulation Execution Configuration

The activity *Design Simulation Execution Configuration* defines the runtime properties of the modules defined in the previous steps. This includes the definition of the number of simulation module instances, the definition of the update rate for module instances for each publication (in the publish/subscribe definition), and the definition of the execution cost of each module instance on each target node. For example, we could decide to have 300 Radar module instances each of them publishing RadarBeam objects with update rate of 5 times per second. The execution cost (with respect to processing power) for each Radar module instance is defined using scaled value and defined as 6 over 10 for one node, 4 over 10 for another node, etc.

Simulation Execution Configuration Metamodel as shown in Figure 9 is used to define the artifacts to model the simulation execution configuration. *SimulationExecutionConfiguration* is the root class of the metamodel that defines a simulation execution configuration which includes *Metadata, ModuleInstances* and *Publications*. *Metadata* defines *name*, *version*, *creator*, and *creation date* of a simulation execution configuration. *ModuleInstance* represents an instance of a simulation module that was defined in the activity *Design Simulation Modules*. The attribute *name* of *ModuleInstance* defines the symbolic name of the module instance. For example, the name of a module instance that instantiates "AirplaneModel" simulation module can be "F16 Block 50". Each module instance can have a different execution cost for different nodes. For this *ModuleInstance* includes the parameter *nodeExecutionCostTable* that defines the execution cost values for the nodes on which the module instance can execute. Note that the execution cost is dependent on the selected execution configuration. For example, the execution cost of a radar model changes according to existing signal reflection sources (e.g. air planes) in execution configuration. Usually, the execution cost of a module instance can be measured exactly when it is developed and executed on target nodes [Lauterbach et al. 2008]. During design time, the value for the execution cost can be estimated using, for example, design phase, complexity calculation methods such as proposed by Podgorelec and Hericko [2007], or prototyping. The execution cost is a scaled value and shows the execution cost of a Module Instance in comparison with other Module Instances of the same execution configuration. For example, a radar model instance can have estimated execution cost of 20 while a relatively simple Electro Optic Sensor has the value of 5. The attribute *requiredMemoryAmount* of *ModuleInstance* represents the estimated memory amount that the module instance will require during execution. Similar to the execution

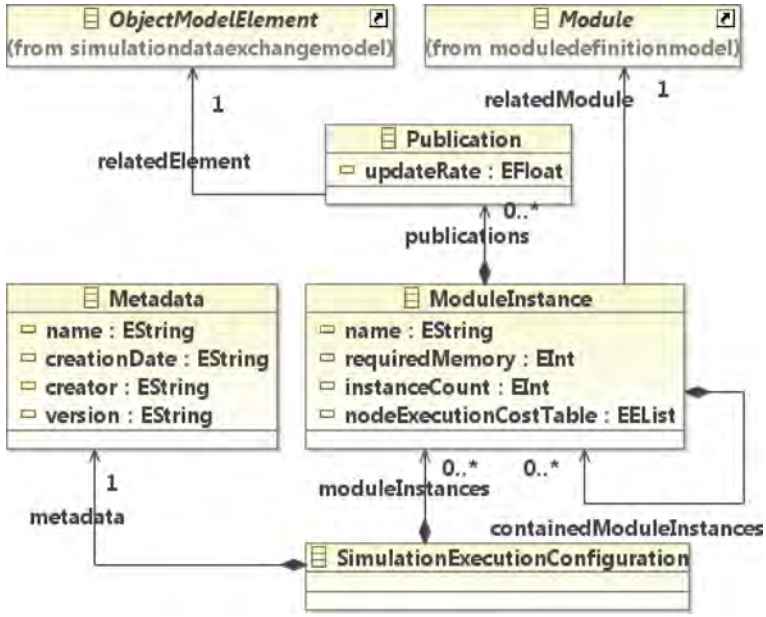

Fig. 9. Simulation execution configuration metamodel.

cost, this parameter can be estimated at design time. The attribute *instanceCount* of *ModuleInstance* defines the number of instances in the execution configuration. This attribute is added because there may be multiple instances of the same module in an execution configuration. For example, in a large-scale EW scenario, there can be hundreds of IR guided missiles and it is not feasible to add one module instance for each of them to the execution configuration separately.

The relation *containedModuleInstances* of *ModuleInstance* class shows the contained Module Instances. For example, this association can be used to show systems on a platform such as radars on a surface vessel. The relation *relatedModule* associates a *ModuleInstance* with a Module that is defined in the activity Design Simulation Modules. *ModuleInstance* can have zero or more Publications that represent the update rate and the related element from SDEM. Each publish is associated with an object class or interaction class defined in SDEM. The update rate shows how many times a module instance will update a SDEM element in a second.

5.6. Generate Input Parameters for Allocation Algorithm

After defining the static and runtime properties of the simulation participants, the simulation entities and the physical resources through these steps, we can start searching for a deployment alternative. That is, we need to allocate the simulation module instances to the nodes by considering execution costs, memory requirements, communication costs, processing power and memory capacity restrictions defined in the simulation design. In fact, the allocation problem that we want to solve here is equivalent to the Capacitated Task Allocation Problem (CTAP) [Pirim 2006], which is a special form of the Task Allocation Problem (TAP) [Stone 1977]. The TAP considers the allocation of a set of tasks to a set of processors according to execution cost, communication cost, I/O cost, and memory. The CTAP specializes the TAP by including constraints on memory capacity and processing power. In the literature, we can observe several studies that describe different versions of the TAP and the

$$(\mathbf{M}) \quad \text{Minimize} \quad \sum_{i=1}^{m}\sum_{p=1}^{n} a_{ip}x_{ip} + \sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{p=1}^{n} a_{ip}(1-a_{jp})c_{ij}$$

Subject to

$$\sum_{p=1}^{n} a_{ip} = 1, \qquad \forall i = 1,.....,m$$

$$\sum_{i=1}^{m} m_i a_{ip} \le M_p, \quad \forall p = 1,.....,n$$

$$\sum_{i=1}^{m} x_{ip} a_{ip} \le C_p, \quad \forall p = 1,.....,n$$

$$a_{ip} = \{0,1\}, \qquad \forall i = 1,.....,m, \ \forall p = 1,.....,n$$

Fig. 10. Formulation of the problem.

CTAP focusing on different parameters and constraints [Lo 1988; Mehrabi et al. 2009; Pirim 2006; Ucar et al. 2005].

Formally, in alignment with the general definitions of the TAP and the CTAP, we define our problem as follows. There exists m tasks, where task i requires m_i units of memory. There are n nonidentical processors, where processor p has a memory capacity of M_p and processing power of C_p. The cost of executing task i on processor p is x_{ip}. In addition, c_{ij} denotes the communication cost of tasks i and j. Communication frequencies shall be taken into account while calculating communication costs. A higher communication frequency between tasks i and j results in a higher communication cost, c_{ij}. We aim to assign each task to a processor without violating the memory and the processing power constraints of each processor. Therefore, the decision variable of the problem is:

$a_{ip} = 1$, if task i is assigned to processor p, 0 otherwise.

The problem can be formulated as a 0-1 program (**M**) similar to Ucar et al. [2005], Pirim [2006], and Mehrabi et al. [2009], in which the objective function is to minimize the sum of the completion time of all tasks and the communication overhead. Based on these definitions, we can formulate our objective as an optimization problem with binary decision variables.

The activity Generate Input Parameters for Allocation Algorithm derives all of the parameters of the model (M) that is defined in Figure 10 from the simulation design defined in the previous activities, as explained in Table II.

5.7. Find Feasible Deployment

The activity *Find Feasible Deployment* takes the parameter values of the previous activity as input and aims to find a good feasible deployment alternative, if one is available. In this context, a good feasible deployment alternative refers to a feasible deployment alternative that is not too far from an optimal deployment. As such, it should be noted that the term "feasibility" does not correspond to the feasibility of the simulation system itself but the feasibility of the selected deployment alternatives.

We can identify different approaches in the literature that can be used to solve different forms of the TAP and the CTAP [Chen and Lin 2000; Hamam et al. 2000; Lo 1988; Mehrabi et al. 2009; Pirim 2006; Ucar et al. 2005]. For a detailed comparison of these approaches, the reader might refer to Ucar et al. [2005] and Pirim [2006]. Since

Table II. Deriving CTAP Parameters from the Design

CTAP Parameter	Derivation from the design
m tasks	Each *module instance* defined in Simulation Execution Configuration Development activity corresponds to a task.
n processors	Each *node* defined in Physical Resource Design activity corresponds to a processor.
M_p	The *memoryCapacity* attribute of each *node p* defined in Physical Resource Design activity corresponds to the memory capacity of processor p.
C_p	The *powerFactor* attribute of *node p* defined in Physical Resource Design activity corresponds to the processing power of processor p.
m_i	The *requiredMemory* attribute of *ModuleInstance i* defined in Simulation Execution Configuration Development activity corresponds to memory requirement of task i.
x_{ip}	The *nodeExecutionCostTable* attribute of *ModuleInstance i* defined in Simulation Execution Configuration Development activity defines execution cost for *node p* which corresponds to cost of executing task i on processor p.
c_{ij}	The communication cost c_{ij} for tasks i and j is calculated by using: — *Publications* defined in Simulation Execution Configuration Design activity, — *Subscriptions* defined in Publish/Subscribe Relations of Simulation Modules Design activity, — *Object model elements* defined in Simulation Data Exchange Model Design activity

our deployment alternative selection problem defined in Section 5.6 is a form of CTAP, the corresponding model (**M**) provided in Figure 10 can be solved through one of the proposed approaches. Please note that we do not mandate a particular algorithm or tool but recommend using a practical one for the corresponding case. We focused on Pirim [2006] and Mehrabi et al. [2009], since their problem definition resembles our problem. Pirim [2006], redefines the CTAP as a quadratic problem with capacity constraints, and solves the problem through a commercial optimization problem solver, CPLEX [IBM 2010]. Pirim [2006] also compares the results of CPLEX with several alternative methods for solving CTAP. Mehrabi et al. [2009] proposes a heuristic algorithm to address the CTAP, and considers execution costs, communication costs, and memory requirements similar to our problem.

If a feasible deployment is found, the output of this activity is a table that represents the mapping of tasks (module instances) to processors (nodes). If the algorithm was not successful in finding a feasible solution, the process returns to the activity *Develop Simulation Execution Configuration*. This can be repeated several times until a feasible deployment is found. If it appears that a feasible deployment cannot be found by changing just the simulation execution configuration, then the designer can decide to return to the beginning of step 3 to refine/update the design. To find a feasible design alternative, the cost parameter values that are extracted from the simulation design including memory/execution costs and the communication cost between modules can be adapted. Even though the optimization of the PADS design parameters is highly dependent on the nature of the specific application (in our case, the EW Simulation system), we provide the following generic recommendations to find a feasible deployment alternative.

(1) *Reduce the Update Rates in the Simulation Execution Configuration.* The update rates affect the communication cost between publisher and subscribers. Moreover, the higher update rates results in higher execution costs for publishers. Likewise, to optimize the PADS design, the update rates of some publications in the simulation execution configuration can be reduced. For example, in our case study, the

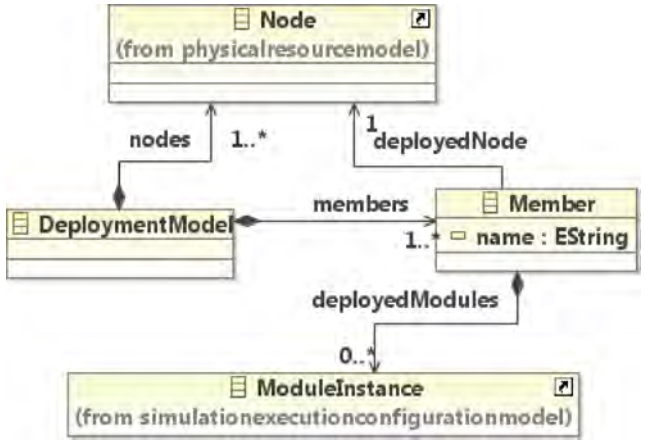

Fig. 11. Deployment metamodel.

update rates for surface vessels can be reduced because they are moving slow when compared to air platforms.

(2) *Check Subscribed Data Sets.* A subscription causes the delivery of the updates of the specific data class at runtime. In many cases, however, the subscriber only requires a specific set of data class attributes (e.g. position of the object). As such, the PADS design can be optimized to provide more precise subscription definitions with respect to the needs of data set requirements. For example, in our case study, the propagation model only requires the position and the orientation of the platforms, likewise it should only subscribe to base Platform class instead of specific platform classes such as Aircraft, Surface Vessel, Submarine, etc.

(3) *Check Reliability of the Data.* In the Publish/Subscribe model, a data object can be shared among participants with two different reliability levels: *reliable* and *best effort*. Sharing data in a reliable manner is higher than best effort sharing with respect to the communication cost. Likewise, the reliability level of the data objects could be reduced to best effort, if possible, to further optimize the simulation design. For example, in our case study, the position information of the platforms is frequently updated and can be defined as best effort if the subscribers are using dead reckoning methods [Fujimoto 1999] for calculating platform positions.

(4) *Upgrade Physical Resources.* If all software-level design optimizations described here are applied and it is still not possible to find a feasible deployment alternative, the only alternative is upgrading the physical resources.

5.8. Generate Deployment Model

The task-processor mapping table that is the output of the previous activity will be used in the activity *Generate Deployment Model* to map this to a deployment model. The deployment metamodel as shown in Figure 11 contains members and nodes. Each member is deployed on one of the nodes defined in Physical Resource Model. One or more module instances can be deployed on a member.

6. TOOLS AND APPLYING THE APPROACH TO THE CASE STUDY

To support the activities of the approach described in the previous section, we developed an IDE (Integrated Development Environment) tool that provides an integrated development environment for designing the simulation system and automatic generation of deployment models.

Table III. Time to Generate Values for Scenarios using an Implementation of CTAP Algorithm

Number of Simulation Entities	Number of Nodes	Time to generate (seconds)
5	4	20
17	4	50
81	4	182
141	5	325
1596	6	360

The automatic deployment model generation algorithm that is used by the tool is given in the Online Appendix B.

We have used the tool for the case study that is described in Table I. We have defined the Simulation Data Exchange Model, Simulation Modules, Publish/Subscribe Relations of the modules and Physical Resource Model (with 6 heterogeneous nodes with different processor and memory capacities). After that, we have defined a sample execution configuration for the system and executed the feasible deployment generation algorithm. An example result of the generation algorithm is shown in Figure 1 in Online Appendix C. As it can be observed from the figure, the resulting deployment model includes 6 nodes as it has been defined. Further, the simulation module instances defined in the execution configuration model have been deployed to the physical nodes to optimize the values for the metrics execution cost, communication cost and resource requirements. A close analysis of the generated alternative shows that the total resource requirements of the simulation module instances that are deployed on each node do not exceed the capacity of the corresponding node. Further, based on the adopted genetic algorithm, it appears that simulation module instances that interact frequently and which have high communication costs, are as much as possible co-located on the same node. For example, simulation module *Propagation Model* appeared to have frequent interaction with other simulation modules in the simulation design, and we can observe that it has a high update-rate. Likewise, the adopted algorithm has allocated *Propagation Model* in a node together with as much as possible simulation module instances it interacts with. The remaining simulation instances interacting with *Propagation Model* that would exceed the resource capacity of this node are deployed to other nodes in a similar manner.

In this particular case, we have adopted the genetic algorithm as defined by Mehrabi et al. [2009] for the CTAP algorithm. The generation algorithm is implemented in Java and executed on an Intel Core I-7 2.70-GHz 64-Bit computer with 4 GBytes of RAM. As an illustration, the performance results for a number of scenarios is shown in Table III. The last row represents the scenario that is used in this case example. Each scenario has been actually defined, executed and the time to generate has been measured. It can be observed that the generation times of deployment alternatives are acceptable for evaluation at design time. Even for the case study with 1596 simulation entities, we were able to generate the deployment alternative in 360s. Please also note that different algorithm implementations or tools might be used to obtain different results. However, since the approach does not mandate a particular implementation of the CTAP algorithm, we consider the analysis of the algorithm implementations beyond the scope of this work.

7. DISCUSSION

In this article, we have provided a concrete approach and tool support for deriving a feasible deployment alternative based on the simulation system and the available physical resources. The approach supports the realization of two recommended tasks defined in DSEEP step 3 (*Design Simulation Environment)*. The necessity and

practical value of the approach is based on the recommendations as defined by the IEEE standard DSEEP. Both the approach and the corresponding tools assist the designer to derive a feasible deployment model in early system design phase.

A valid question in this context is whether the adopted algorithm leads to a solution and whether this solution is optimal with respect to the metric values. To define the realization of the approach, we have applied the well-known CTAP solving algorithms that have been widely addressed in the literature. The approach does not mandate the usage of a particular algorithm but provides the required input parameters for these algorithms. The correctness of these algorithms has been discussed in the corresponding articles and based on this we can assume that a good feasible solution is derived. In addition, depending on the state of the system, different CTAP solving algorithms may be used to optimize the metric values. For the comparison of the algorithms we refer to, for example, Ucar et al. [2005] and Pirim [2006]. We have also defined general rules to improve the CTAP cost parameter values to be able to find a feasible deployment alternative with respect to the project requirements, if the original parameters do not result in a feasible solution.

Besides of the algorithmic performance, we also focus on the organization level performance gain. Existing practices usually base the generation of the deployment model on the expert judgment or defer the generation of the deployment model to the implementation phase. Unfortunately, expert judgment is limited due to the manual effort. We go one step further by integrating the existing CTAP solution techniques early in the system design, and automate the decision process to support the evaluation of the design alternatives by the experts. As stated before in Section 4 (problem statement), deferring the definition of the deployment to the development phase might lead to nonfeasible implementations which will require iterating the design and the related project lifecycle artifacts such as detailed design, implementation, test artifacts, documentation, etc. On its turn, this will lead to delays and higher cost in the project. This is also the reason why DSEEP recommends evaluating the design alternatives in the early phases of the development life cycle. At design time, the values for execution cost and memory requirements are estimated while the communication costs are calculated. Obviously, the better the estimation the more feasible the deployment model will be. The estimation of the values can be enhanced by analyzing existing similar models or by developing prototypes. Likewise, the identified deployment model may be refined and optimized if more accurate information is available in subsequent phases of the project lifecycle. The approach itself can actually be used at any time during the project life cycle and, if possible, even after the system has been developed. In the latter case, the measured runtime parameter values can be used, instead of estimated values, to derive the deployment model. The runtime parameter values can be, for example, collected using Management Object Model (MOM) services as defined in IEEE [2010b] for HLA based simulation systems. A valid question in this context is whether the cost of changing the selected deployment alternative is reasonable or not. In fact, changing the selected deployment alternative does not require changing the simulation modules, because the publish/subscribe paradigm abstracts the location of the publishers from subscribers and vice-versa. Selecting a different deployment alternative, as such, will only require changing the execution locations of the simulation modules (e.g., by changing a configuration file that defines the location of each simulation module).

The approach is developed for simulation architectures that adopt publish-subscribe model, such as HLA and TENA. However, the approach can also be used for other architectures that adopt publish-subscribe model, such as OMG DDS [OMG 2006]. The basic steps as defined in Figure 5 will remain the same and, if needed, with relatively small efforts the realization of the steps can be updated for the target architecture. For example, the data exchange model may need to be changed due to the particular data

exchange models for the given infrastructure. In our future research, we will analyze this in more detail.

The approach has been defined based on the actual problems that were experienced in a real industrial context. In addition, the case study in this article has been defined reflecting the similar complexity level as we have experienced in our earlier Electronic Warfare simulation projects. The case that we have defined is inspired from a real industrial project and includes 1596 simulation entities that were deployed on 6 nodes. In general, this is a realistic case. Unfortunately, due to confidentiality reasons, company names and project names have not been given.

In large-scale parallel and distributed simulation systems, many simulation objects may exist and communicate with each other, which can result in increased communication overhead across the network. The Data Distribution Management (DDM) is a set of optional services to optimize the communication among the different simulation members, typically by limiting and controlling the data exchanged in a simulation. As such, using DDM services in PADS execution can affect the communication patterns among the nodes. DDM requires knowing the values of the exchanged data at runtime, for dynamically deciding the transmission of the data to the relevant subscribers. In our approach, though, we have explicitly focused on deriving feasible decomposition alternatives at design time where values of the simulation data objects are not necessary yet. However, in case DDM is used, it is required to consider the actual data values to further optimize the deployment alternative. With our approach, the derived feasible alternatives will form an improvement at design time whereas DDM optimizes the communication among the simulation members at runtime.

The scalability of the approach could be analyzed with respect to the number and characteristics of simulation entities, the number of nodes, and the adopted implementation of the algorithm. In our particular case, the approach seemed to generate deployment models in a reasonable time as shown in Table III. In our future work we will also consider the analysis and comparison of various algorithm implementations to further optimize the approach.

In the problem formulation of CTAP in Section 5.6, different processors can perform a task with different costs. The rationale behind minimizing the sum of total processing costs is to assign each task to the processor that can perform it with minimum cost. The given CTAP formulation may cause imbalanced deployments if the total capacity of the processors is quite large with respect to the total execution cost of tasks. We do not consider this always as a bad assignment because this might be favorable for reducing the communication costs. In such a situation, the required action can be different according to the targeted objective. If the objective is to find the minimum necessary resource configuration, then the physical resources can be reduced. However, the physical resources can have been deliberately defined to be large in order to enhance the level of parallelism. As such, we have not provided the constraint for maximum load on each processor. Yet, we consider the load-balancing issue together with the trade-off analysis between resource usage and communication costs as an interesting future work.

8. RELATED WORK

Various approaches can be identified in the literature that can be used to support the DSEEP including conceptual model development approaches [Karagöz and Demirörs 2007; SISO 2006], tool support [Parr 2003; VT MAK 2010a] and simulation design approaches [Çetinkaya and Oguztüzün 2006; Topçu et al. 2008]. Yet, despite the benefit of these approaches, no adequate and explicit support for selecting and evaluating the deployment alternatives have been provided so far.

Different metamodels for modeling parallel and distributed simulations have been provided in the literature. BOM (Base Object Model) Template Specification [SISO 2006] defines the semantics and the syntax for specifying aspects of the conceptual model of simulations that can be used and reused in the design, development, and extension of interoperable simulations. A BOM is a platform independent model that contains *conceptual entities* and *conceptual events* representing items in the real world. Further, BOMs define the interactions among the simulation items in terms of *patterns of interplay* and *state machines*. A BOM includes an interface description (Object Model Definition) that is defined using the High Level Architecture Object Model Template (HLA OMT) [IEEE 2010c]. The HLA OMT constructs include object classes, interaction classes, and their attributes and parameters. In the literature different realizations of the HLA OMT have been proposed [Çetinkaya and Oguztüzün 2006; Parr 2003]. The Simulation Data Exchange Metamodel that we have defined in this article enhances the HLA OMT for supporting the derivation of feasible deployment alternatives.

Based on the conceptual models as defined, for example, using BOM, simulation systems can be designed using the UML [Fowler 2003] or UML profiles [Çelik 2005; Guiffard et al. 2006]. In Topçu et al. [2008], Live Sequence Charts [Brill et al. 2004] are adopted to model the behavior of federates in simulation systems.

To support the automation of the development of simulation systems, model-driven development has been promoted; see, for example, Tolk [2002]. In this context, for example, in the OMG's Model Driven Architecture (MDA) approach [Frankel et al. 2004; Schmidt 2006] a distinction is made between platform independent models (PIM) and platform specific models (PSM). A PIM defines a model of the system independent of the platform it abstracts away from. A PSM defines a refinement of the PIM using platform specific concern details. PSMs are automatically or semi-automatically transformed from PIM. In a similar sense, PSM can be used to generate code that is specific to the adopted programming language platform. Using the MDA approach, the developed metamodel of the simulation system can be used to define platform independent models (PIM) that abstract away from simulation architectures such as HLA, DIS, and TENA. The metamodels that we have defined in this article are all platform independent and we can derive platform specific models from these. For example, the Simulation Data Exchange Model (SDEM) is a PIM that can be used to define PSMs for particular simulation architectures. On the other hand, the HLA Federation Object Model (FOM) is a PSM that is derived from this SDEM.

Our approach supports DSEEP Activity 3 which is "*Design Simulation Environment*" as explained before. Different approaches have been carried that can be used to support the DSEEP. For example, Karagöz and Demirörs [2007] define a methodology to support DSEEP Activity 1, "*Define Simulation Environment Objectives*" and Activity 2, "*Perform Conceptual Analysis*". BOM [SISO 2006] provides a solution for DSEEP Activity 2. The HLA Object Model Metamodel [Çetinkaya and Oguztüzün 2006], and FAMM [Topçu et al. 2008] support the DSEEP Step-3, *Design Simulation Environment*. VR Forces [VT MAK 2010b], VR Vantage XR [VT MAK 2010c], and the code generation tool defined in [Adak et al. 2010] support the DSEEP Step-4, "*Develop Simulation Environment*". The design and code generation tools given in [Guiffard et al. 2006; Parr 2003] support both DSEEP Step-3 and Step-4. The Pitch Commander tool [Pitch Technologies 2009] and RTI Spy module of MAK RTI [VT MAK 2010d] support DSEEP Step-5, "*Plan, Integrate and Test Simulation Environment*".

9. CONCLUSION

To realize the execution of the Parallel and Distributed Simulation Systems (PADS) different simulation infrastructures have been defined in the literature such as HLA,

DIS and TENA. One of the important problems in PADS is the allocation of the different applications to the available physical resources. Usually, the deployment of the applications to the physical resources can be done in many different ways. In this article we have provided a concrete approach for deriving a feasible deployment alternative based on the simulation system and the available physical resources at the design phase. The approach has been integrated in the Distributed Simulation Engineering and Execution Process (DSEEP), which includes a recommended step for evaluation of the performance of the simulation systems at the design phase. To illustrate the approach, we have adopted a realistic case study concerning the development and integration of a simulation system that contains simulators in the context of Electronic Warfare (EW). We have defined several models that are needed in the approach including *Simulation Data Exchange Model*, *Simulation Modules, Publish-Subscribe Relations, Physical Resources,* and *Simulation Execution Configuration*. Based on these models the necessary parameter values for the CTAP algorithm have been defined to generate a feasible deployment alternative. The generation times of the deployment alternatives were acceptable for evaluation at design time.

Obviously, the approach would not be feasible without adequate tool support. Therefore, we have developed an IDE tool that provides an integrated development environment for designing the simulation, automatic generation of CTAP algorithm parameters, and the automatic generation of the deployment alternative with respect to the defined parameter values. We have used a relatively large case study that could be easily supported in the tool, and we believe that the tool can be used for even larger case studies without substantial problems.

The approach builds on various metamodels that we have defined, and which are used to support the automatic generation of feasible deployment alternatives. In our future work we will focus on further automation of the simulation development process using the developed metamodels. In particular, we will focus on automatic code generation for HLA using the metamodels. This will include the generation of member templates and the mapping of data exchange model elements defined in SDEM to the target platform. Further, we aim to integrate behavioral modeling to consider also dynamic aspects of simulation systems.

ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library.

ACKNOWLEDGMENTS

We would like to thank the reviewers for the critical but constructive feedback, which has improved the overall quality of this article. Further, we would like thank Oge Bozyigit, Hakan Serçe, and Eda Yücel for proofreading and discussions.

REFERENCES

Adak, M., Topçu, O., and Oguztüzün, H. 2010. Model-based code generation for HLA federates. *Softw. Pract. Exper. 40,* 2, 149–175.

Adamy, D. 2001. *EW 101: A First Course in Electronic Warfare* 1st Ed. Artech House.

Adamy, David L. 2006. *Introduction to Electronic Warfare Modeling and Simulation.* SciTech Publishing.

Brill, M., Damm, W., Klose, J., Westphal, B., and Wittke, H. 2004. Live sequence charts: An introduction to lines, arrows, and strange boxes in the context of formal verification. Lecture Notes in Computer Science, vol. 3147, Springer Verlag, 374–399.

Çelik, T. 2005. A software modeling tool for the high level architecture. Master's thesis, Hacettepe University Institute of Science, Ankara, Turkiye.

Çetinkaya, D. and Oguztüzün, H. 2006 A metamodel for the HLA object model. In *Proceedings of the 20th European Conference on Modeling and Simulation (ECMS)*. 207–213.

Chen, W. and Lin, C. 2000. A hybrid heuristic to solve a task allocation problem. *Comput. Oper. Res. 27,* 287–303.

Eugster, Patrick Th., Felber, Pascal A., Guerraoui, R., and Kermarrec, A. 2003. The many faces of publish/subscribe. *ACM Comput. Surv. 35*, 2, 114–131.

Fowler, M. 2003. *UML Distilled: A Brief Guide to the Standard Object Modeling Language* 3rd Ed. Addison-Wesley Professional.

Frankel, David S., Parodi, J., and Soley, R. 2004. *The MDA Journal: Model Driven Architecture Straight from the Masters.* Meghan Kiffer Pr.

Fujimoto, Richard M. 1999. *Parallel and Distributed Simulation Systems* 1st Ed. Wiley-Interscience.

Guiffard, E., Kadi, D., Mochet, J., and Mauget, R. 2006. CAPSULE: Application of the MDA methodology to the simulation domain. In *Proceedings of the European Simulation Interoperability Workshop (SIW).*

Hamam, Y. and Hindi, K. S. 2000. Assignment of program modules to processors: A simulated annealing approach. *Europ. J. Oper. Res. 122*, 2, 509–513.

IBM. 2010. Modeling with IBM ILOG CPLEX CP Optimizer – Practical scheduling examples, http://www-01.ibm.com/software/integration/optimization/cplex-optimization-studio/.

IEEE. 1998. IEEE STD 1278.1A-1998, Standard for Distributed Interactive Simulation – Application Protocols.

IEEE. 2003. IEEE STD 1516.3-2003, Recommended Practice for HLA Federation Development and Execution Process (FEDEP).

IEEE. 2010a. IEEE STD 1516-2010, Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules.

IEEE. 2010b. IEEE STD 1516.1-2010, Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification.

IEEE. 2010c. IEEE STD 1516.2-2010, Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)- Object Model Template (OMT) Specification.

IEEE. 2010d. IEEE STD 1730-2010, Recommended Practice for Distributed Simulation Engineering and Execution Process (DSEEP).

Karagöz, A. and Demirörs, O. 2007. Developing conceptual models of the mission space (CMMS) - A metamodel based approach. In *Proceedings of the Spring Simulation Interoperability Workshop (SIW).*

Kuhl, F., Weatherly, R., and Dahmann, J. 1999. *Creating Computer Simulation Systems: An Introduction to the High Level Architecture* 1st Ed. Prentice Hall.

Lauterbach, C., Lin, M. C., Dinesh. M., Borkman, S., Lafave, E., and Bauer, M. 2008. Accelerating line-of-sight computations in large OneSAF terrains with dynamic events. In *Proceedings of the Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC).*

Lees, M., Logan, B., and Theodoropoulos, G. 2007. Distributed simulation of agent-based systems with HLA. *ACM Trans. Model. Comput. Simul. 17,* 3, Article 11.

Lo, V. M. 1988. Heuristic algorithms for task assignment in distributed systems. *IEEE Trans. Comput. 37,* 11, 1384–1397.

Mehrabi, A., Mehrabi, S., and Mehrabi, Ali D. 2009. An adaptive genetic algorithm for multiprocessor task assignment problem with limited memory. In *Proceedings of the World Congress on Engineering and Computer Science*. Vol II.

Noseworthy, J. Russell. 2008. The test and training enabling architecture (TENA) supporting the decentralized development of distributed applications and LVC simulations. In *Proceedings of the 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*. 259–268.

OMG. 2006. Data distribution service for real-time systems Ver 1.2.

Parr, S. 2003. A visual tool to simplify the building of distributed simulations using HLA. *Inf. Secur. J. 12,* 2, 151–163.

Perumalla, Kalyan S. and Fujimoto, Richard M 2003. Scalable RTI-based parallel simulation of networks. In *Proceedings of the 17th Workshop on Parallel and Distributed Simulation (PADS'03)*. IEEE Computer Society, 97–104.

Pitch Technologies. 2009. Pitch Commander. http://www.pitch.se/images/files/productsheets/PitchCommanderWeb_0904.pdf.

Pirim, T. 2006. A hybrid metaheuristic algorithm for solving capacitated task allocation problems as modified XQX problems. The University of Mississippi. ProQuest Dissertations and Theses, http://search.proquest.com/docview/305312128?accountid=11248.

Podgorelec, V. and Hericko, M. 2007. Estimating software complexity from UML models. *SIGSOFT Softw. Eng. Notes 32*, 2, 1–5.

Schmidt, Douglas C. 2006. Guest editor's introduction: Model-driven engineering. *IEEE Comput. 39,* 2, 25–31.

SISO. 1999. Realtime platform reference (RPR) FOM. SISO-STD-001.1.

SISO. 2006. Base Object Model (BOM) template specification. SISO-STD-003-2006.

Stone, H. S. 1977. Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Trans. Softw. Eng. 3*, 1, 85–93.

Tolk, A. 2002. Avoiding another green elephant – A proposal for the next generation HLA based on the model driven architecture. In *Proceedings of the Fall Simulation Interoperability Workshop (SIW).*

Topçu, O., Adak, M., and Oguztüzün, H. 2008. A metamodel for federation architectures. *ACM Trans. Model. Comput. Simul. 18*, 3, Article 10.

Ucar, B., Aykanat, C., Kaya, K., and Ikinci, M. 2005. Task assignment in heterogeneous computing systems. *J. Parall. Distrib. Comput. 66*, 1, 32–46.

VT MAK. 2010a. MAK Data Logger. http://www.mak.com/component/docman/doc_download/13-maek-data-logger--simulation-recording-a-playback.html.

VT MAK. 2010b. VR-Forces. http://www.mak.com/resources/white-papers/doc_download/15-vr-forces--the-complete-computer-generated-forces-solution-for-simulation.html.

VT MAK. 2010c. VR-Vantage XR. http://www.mak.com/component/docman/doc_download/24-vr-vantage-xr--common-operating-picture.html.

VT MAK. 2010d. MAK RTI. http://www.mak.com/component/docman/doc_download/14-maek-high-performance-rti.html.

Zeigler, B. P. 2003. DEVS today: Recent advances in discrete event-based information technology. In *Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems (MASCOTS'03)*. 148–161.

Received October 2011; revised January 2012, May 2012, June 2012, October 2012; accepted January 2013